PHASE-4: Development Part 2

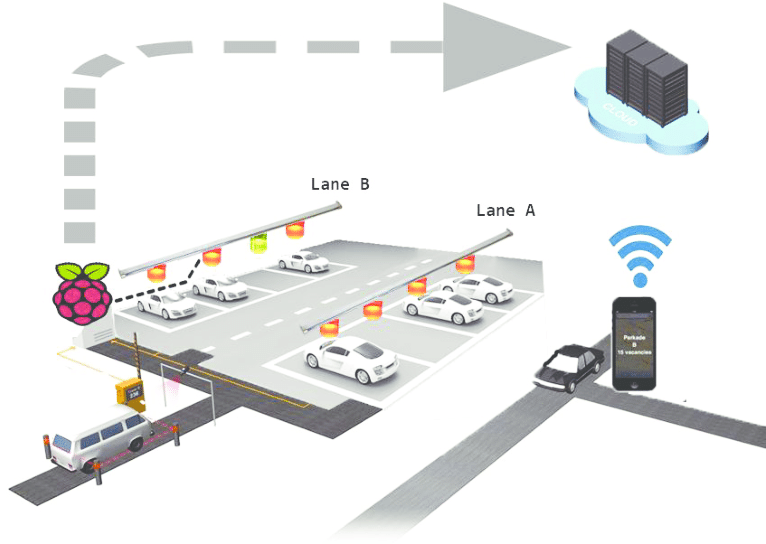
Project Title: Smart Parking System with IoT

Presented By : Sivaranjini M

Roll No : 312821243036

# PROJECT DESCRIPTION:

Develop a Smart Parking System that leverages IoT technology, including IoT sensors and Raspberry Pi devices, to efficiently manage parking spaces, reduce congestion, and enhance the overall parking experience for users. This system will provide real-time parking space availability information and optimize parking space allocation.



To collect data from an IoT sensor and send it to a database using Python, these steps are followed:

Creating a complete Smart Parking System with feature engineering, model training, and evaluation is a substantial project that requires significant time and resources. I'll provide a simplified outline of these steps to guide you in building the project:

**1. Hardware and Sensor Setup:**

* Set up Raspberry Pi devices with IoT sensors as described in the previous response.

**2. Data Collection:**

* Collect data on parking space availability and occupancy using IoT sensors. This data will serve as your raw dataset.

**3. Data Preprocessing and Feature Engineering:**

* Process the raw sensor data to extract meaningful features. Some potential features might include:
* Time of day
* Day of the week
* Weather conditions
* Historical occupancy data
* Label the data (occupied or available parking spaces).

**4. Model Selection and Training:**

* Choose an appropriate machine learning model. For predicting parking space availability, you could use regression or classification models.
* Split the dataset into training and testing sets.
* Train the model using historical data.

**5. Real-time Data Collection and Prediction:**

* Continuously collect real-time data from IoT sensors.
* Use the trained model to make real-time predictions on parking space availability.

**6. User Interface Development:**

* Create a web application to display real-time parking space availability to users.
* Implement features for booking and reserving parking spaces.

**7. Optimization and Notifications:**

* Implement an optimization algorithm to allocate parking spaces efficiently.
* Develop a notification system to alert users when their reserved parking space is about to expire or if there are any changes in space availability.

**8. Evaluation:**

* Continuously monitor and evaluate the system's performance.
* Collect user feedback for improvement.
* Track metrics like user satisfaction, system uptime, and the accuracy of space availability predictions.

**9. Security and Privacy:**

* Ensure the system's security and protect user data.
* Comply with data privacy regulations.

**10. Scalability and Maintenance:**

* Design the system to be easily scalable as the parking area grows.
* Regularly update and maintain the software and hardware components.

# PROGRAM:

**Requirements:**

1. Raspberry Pi (with Raspbian OS)

2. Ultrasonic Sensor (HC-SR04 or similar)

3. Python libraries: RPi.GPIO (for GPIO control), time (for time delays)

**Python Code:**

import RPi.GPIO as GPIO

import timeS

# GPIO pins for the Ultrasonic sensor

TRIG = 23

ECHO = 24

# Initialize GPIO settings

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG, GPIO.OUT)

GPIO.setup(ECHO, GPIO.IN)

def distance\_measurement():

GPIO.output(TRIG, False)

time.sleep(2) # Allow some time for sensor to settle

# Trigger the ultrasonic sensor

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO) == 0:

pulse\_start = time.time()

while GPIO.input(ECHO) == 1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150 # Speed of sound in cm/s

return distance

try:

while True:

dist = distance\_measurement()

if dist < 10: # Adjust this threshold based on your parking space size

print("Parking space occupied.")

else:

print("Parking space available.")

time.sleep(1) # Delay before next measurement

except KeyboardInterrupt:

GPIO.cleanup()

```

This program uses an ultrasonic sensor to measure the distance between the sensor and any obstacle in front of it. If the distance is less than a specified threshold (in this example, 10 cm), it indicates that the parking space is occupied; otherwise, it's considered available. You can adjust the threshold value to match the size of your parking space.

**NOTE:**

* In a real Smart Parking System, you would integrate multiple sensors, connect to a database to store occupancy data, and provide a user interface to display real-time information and make reservations.