SMART PARKING USING IOT

TEAM MEMBER

Phase -2 Document submission	**J.ARUNKUMAR**
Project:**SMART PARKING**	**510421106003**

INTRODUCTION:

Smart parking in the Internet of Things (IoT) is a cutting-edge technology-driven solution that aims to revolutionize the way we find, access, and manage parking spaces in urban areas. Traditional parking systems are often plagued by issues such as congestion, inefficiency, and frustration for both drivers and parking operators. Smart parking leverages IoT sensors, data analytics, and connectivity to address these challenges and create a seamless parking experience.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_PARKING_SPACES 20
// Define a structure for a parking space
typedef struct {
    int spaceNumber;
    bool isOccupied;
    int vehicleNumber;
ParkingSpace;
// Initialize the parking spaces
ParkingSpace parkingSpaces[MAX_PARKING_SPACES];
// Function to initialize the parking spaces
void initializeParkingSpaces() {
    for (int i = 0; i < MAX_PARKING_SPACES; i++) {
        parkingSpaces[i].spaceNumber = i + 1;
        parkingSpaces[i].isOccupied = false;
        parkingSpaces[i].vehicleNumber = -1;
    }
}
```

```
// Function to find an empty parking space
int findEmptyParkingSpace() {
    for (int i = 0; i < MAX_PARKING_SPACES; i++) {
        if (!parkingSpaces[i].isOccupied) {
            return i;
        }
    }
    return -1; // No empty parking spaces available
}
// Function to park a vehicle
bool parkVehicle(int vehicleNumber) {
    int emptySpace = findEmptyParkingSpace();
    if (emptySpace != -1) {
        parkingSpaces[emptySpace].isOccupied = true;
        parkingSpaces[emptySpace].vehicleNumber = vehicleNumber;
        printf("Vehicle %d parked in space %d\n", vehicleNumber,
parkingSpaces[emptySpace].spaceNumber);
        return true;
    } else {
        printf("No empty parking spaces available\n");
        return false;
    }
}
// Function to unpark a vehicle
bool unparkVehicle(int vehicleNumber) {
    for (int i = 0; i < MAX_PARKING_SPACES; i++) {
        if (parkingSpaces[i].isOccupied &&
parkingSpaces[i].vehicleNumber == vehicleNumber) {
            parkingSpaces[i].isOccupied = false;
            parkingSpaces[i].vehicleNumber = -1;
            printf("Vehicle %d unparked from space %d\n", vehicleNumber,
parkingSpaces[i].spaceNumber);
            return true;
        }
    }
    printf("Vehicle %d not found in the parking lot\n", vehicleNumber);
    return false;
}
int main() {
    initializeParkingSpaces();
 // Simulate parking and unparking vehicles
    parkVehicle(1);
    parkVehicle(2);
    parkVehicle(3);
    unparkVehicle(2);
    parkVehicle(4);

    return 0;
}
```

ALGORITHM:

Step 1: Define Requirements

Determine the specific requirements for your smart parking system, including the number of parking spaces, the type of sensors or cameras you'll use to detect vehicles, and the user interface for managing the system.

Step 2: Hardware Setup

Set up the hardware components for your smart parking system, including:

Sensors or cameras to detect vehicles entering and exiting parking spaces.

Microcontrollers or Raspberry Pi to process sensor data.

LED displays or signage to indicate available parking spaces.

Network connectivity to transmit data to the software.

Step 3: Software Development

Develop the software for your smart parking system. This can be done using C or any other programming language that suits your needs. Here's a simplified algorithm in pseudocode:

Step 4: User Interface Development

Create a user interface for the smart parking system. This could be a web-based interface accessible from a computer or mobile device. The interface should allow users to:

Step 5: Database Integration

Integrate a database to store information about parking spaces, reservations, and user data. You may use a relational database (e.g., MySQL) or a NoSQL database (e.g., MongoDB) depending on your requirements.

Step 6: Connectivity and Communication

Implement communication protocols to connect hardware components, such as sensors and LED displays, to the software. Use protocols like MQTT or HTTP for data transmission.

Step 7: Testing

Thoroughly test the system to ensure that it accurately detects vehicle occupancy, updates the status correctly, and communicates with the user interface and database.

Step 8: Deployment

Deploy the smart parking system in the target parking area. Make sure all hardware components are installed and properly configured. Ensure network connectivity and power supply for all devices.

Step 9: Maintenance and Monitoring

Regularly maintain and monitor the smart parking system to ensure it continues to function correctly. Address any issues or errors promptly.

Step 10: User Training

Train users, parking attendants, and administrators on how to use the smart parking system and its user interface effectively.

Step 11: Scalability and Expansion

Consider future scalability and expansion options. You may want to add features like mobile app support, online reservations, or integration with other smart city systems.

Remember that this is a simplified overview, and the complexity of your smart parking system may vary depending on your specific requirements and budget. Be prepared to adapt and modify the system as needed to meet changing demands and technology advancements.

CONCLUSION:

smart parking systems offer significant benefits in terms of efficiency, convenience, and environmental impact. While challenges exist, ongoing technological advancements and a growing awareness of the importance of efficient parking solutions are likely to drive continued adoption and innovation in this field. Smart parking systems have the potential to transform urban mobility and improve the overall quality of life in cities around the world.