SMART PARKING

DEVELOPING SMART PARKING SYSTEM USING ARDUINO:

Developing a smart parking system using Arduino is another viable option for creating a cost-effective and customizable solution. Here's a basic guide on how to develop a smart parking system using Arduino:

Components and Materials:

- **Arduino Board**: Choose a suitable Arduino board (e.g., Arduino Uno, Arduino Mega) depending on the number of parking spaces and sensors you plan to use.
- **Proximity Sensors**: Use ultrasonic sensors or IR sensors to detect vehicle presence in parking spaces.
- **LED Displays**: LED displays or status lights to show the availability of parking spots.
- **Wi-Fi/Bluetooth Module**: To connect your Arduino to the internet and enable data transmission and remote monitoring.
- **Mobile App/Web Interface**: Develop a mobile app or web interface for users to check parking availability and make reservations.
- **Database**: Set up a database to store information about parking spaces, reservations, and user data.
- **Server**: Develop a server application that communicates with the Arduino and the database. You can use platforms like Raspberry Pi or a cloud-based server.

Development Steps:

- **Set up Arduino:**
- Load the Arduino IDE and install the necessary libraries.
- Configure the Arduino board and Wi-Fi/Bluetooth module for network connectivity.
- **Sensor Setup:**
- Connect proximity sensors to the Arduino board.
- Write Arduino code to read data from the sensors and detect vehicle presence.
- **Database and Server:**
- Set up a database to store information about parking spaces, reservations, and user data.
- Develop a server application that communicates with the Arduino and the database. You can use platforms like Raspberry Pi or a cloud-based server.
- **User Interface (App or Web):**
- Create a mobile app or web interface that allows users to check parking availability and make reservations.

SMART

PARKING

- Integrate payment gateways for online payments if needed.
- **Data Processing and Display:**
- Write code on the Arduino to send data about parking space availability to the server.
- Develop code to update LED displays or status lights to indicate the availability of parking spots.
- **Security and Access Control:**
- Implement security measures to protect the system from unauthorized access.
- Create user authentication and authorization processes.
- **Testing and Debugging:**
- Thoroughly test the system to ensure accurate sensor readings, data transmission, and user interface functionality.
- Debug any issues that arise during testing.
- **Deployment:**
- Install the Arduino and sensors in the parking area.
- Ensure a stable power supply.
- **User Training and Support:**
- Educate users on how to use the smart parking system.
- Provide support channels for user inquiries and issues.
- **Monitoring and Maintenance:**
- Continuously monitor the system's performance and resolve any issues that may arise.
- Plan for regular maintenance and updates to the system.

Consider scalability, security, and the specific needs of your parking area when developing a smart parking system using Arduino. Regularly update and maintain the system to ensure its continued functionality and reliability.

ARDINO CODE:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
#include <Servo.h>
Servo myservo1;


int IR1 = 2;
int IR2 = 4;
int SmokeDetectorPin = 6;
int BuzzerPin = 7;
```

SMART PARKING

```
int Slot = 4;


bool flag1 = false;
bool flag2 = false;


unsigned long lastLcdUpdate = 0;
unsigned long lcdUpdateInterval = 1000;


void setup() {
  lcd.begin(16, 2);
  lcd.backlight();
  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);
  pinMode(SmokeDetectorPin, INPUT);
  pinMode(BuzzerPin, OUTPUT);


  myservo1.attach(3);
  myservo1.write(100);


  lcd.setCursor(0, 0);
  lcd.print("     ARDUINO    ");
  lcd.setCursor(0, 1);
  lcd.print(" PARKING SYSTEM ");
  delay(2000);
  lcd.clear();

  Serial.begin(9600);
}


void loop() {
  if (digitalRead(IR1) == LOW && !flag1) {
    if (Slot > 0) {
```

```
      flag1 = true;
      if (!flag2) {
        myservo1.write(0);
        Slot--;
      }
    } else {
      displayMessage("    SORRY :(    ", "  Parking Full  ");
    }
  }


  if (digitalRead(IR2) == LOW && !flag2) {
    flag2 = true;
    if (!flag1) {
      myservo1.write(0);
      Slot++;
    }
  }


  if (flag1 && flag2) {
    delay(1000);
    myservo1.write(100);
    Serial.println("Servo returned to initial position.");
    flag1 = false;
    flag2 = false;
  }



  if (millis() - lastLcdUpdate >= lcdUpdateInterval) {
    updateLcdDisplay();
    lastLcdUpdate = millis();
  }


}


void updateLcdDisplay() {
```

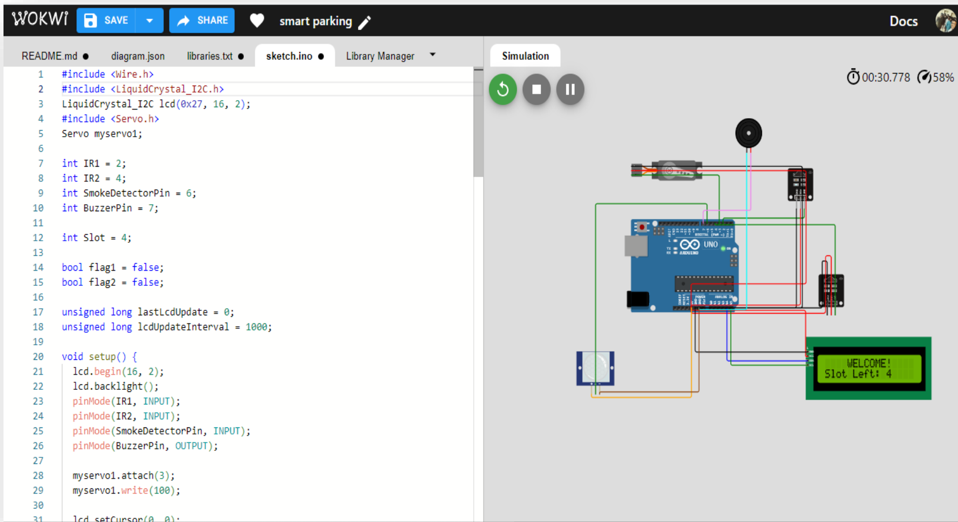
SMART
PARKING

```
  if (digitalRead(SmokeDetectorPin) == HIGH) {
    displayMessage("   WARNING!   ", " Smoke Detected ");
    digitalWrite(BuzzerPin, HIGH);
  } else {
    displayMessage("    WELCOME!    ", "Slot Left: " + String(Slot));
    digitalWrite(BuzzerPin, LOW);
  }
}


void displayMessage(const char *line1, const String &line2) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(line1);
  lcd.setCursor(0, 1);
  lcd.print(line2);
}
```

SMART PARKING

CIRCUIT:



To collect data from sensors connected to a Raspberry Pi and send that data to a cloud server or a mobile app server, you'll need to write Python scripts that use suitable libraries and APIs for data transmission. Below is a basic example of Python scripts for collecting data from sensors (assuming ultrasonic distance sensors) and sending it to a cloud server.

First, make sure you have the necessary libraries installed. You might need to install **`requests`** for making HTTP requests and **`RPi.GPIO`** for working with GPIO pins. You can do this using pip:

Python script that collects data from an ultrasonic sensor and sends it to a cloud server:

```
import RPi.GPIO as GPIO
import time
import requests

TRIG_PIN = 23
ECHO_PIN = 24

GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

def get_distance():
    # Send a short pulse to trigger the ultrasonic sensor
```

SMART
PARKING

```
    GPIO.output(TRIG_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, False)

     while GPIO.input(ECHO_PIN) == 0:
        pulse_start = time.time()
    while GPIO.input(ECHO_PIN) == 1:
        pulse_end = time.time()


    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150  # Speed of sound in cm/s
    distance = round(distance, 2)

    return distance

try:
    while True:
        distance = get_distance()
        print(f"Distance: {distance} cm")


        server_url = " "//URL
        payload = {"distance": distance}
        headers = {"Content-Type": "application/json"}
        response = requests.post(server_url, json=payload, headers=headers)

        if response.status_code == 200:
            print("Data sent successfully")
        else:
            print("Failed to send data")

        time.sleep(5)

except KeyboardInterrupt:
    GPIO.cleanup()
```

SMART PARKING

This script continuously reads distance data from the ultrasonic sensor, sends it to a cloud server (or mobile app server) via an HTTP POST request, and waits for 5 seconds before taking the next measurement.

Make sure to replace with the actual URL of your cloud server's API endpoint. Additionally, handle any authentication and security requirements according to your server's specifications.

This is a basic example. Depending on your project's complexity, you may need to implement more advanced error handling, security measures, and scalability for data collection and transmission.