A blue text on a white background

Description automatically generatedA black and grey logo with a graduation cap and book

Description automatically generated**GOVERNMENT OF TAMILNADU**

DIRECTORATE OF TECHNICAL EDUCATION, CHENNAI

NAAN MUDHALVAN SCHEME (TNSDC) SPONSORED

STUDENTS DEVELOPMENT PROGRAMME

ON

IoT AND ITS APPLICATIONS

HOST INSTITUTION

xxxxx

COIMBATORE – 04

TRAINING PARTNER

ENTHU TECHNOLOGY SOLUTIONS INDIA PVT LTD

DATE:

D

|  |  |
| --- | --- |
| Name | Roll No |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Table Of Contents**

|  |  |  |
| --- | --- | --- |
| **S no** | **Title** | **Page no** |
| 1 | Introduction |  |
| 2 | Abstract |  |
| 3 | Hardware and Software Requirements |  |
| 4 | Block Diagram And  working |  |
| 5 | Code |  |
| 6 | Output |  |
| 7 | Cloud Output  &Applications |  |
| 8 | Conclusion |  |
| 9 | References |  |

**SMART PARKING SYSTEM**

**INTRODUCTION:**

As urbanization accelerates and vehicle ownership increases, efficient management of parking spaces has become a critical challenge for cities worldwide. The traditional methods of parking management are often inadequate, leading to congestion, wasted time, and increased stress for drivers. In response to these challenges, smart parking systems have emerged as a viable solution. Among these systems, one that leverages the ESP32 microcontroller and infrared (IR) sensors stands out for its effectiveness and affordability.

The ESP32 is a versatile, low-cost microcontroller that integrates both Wi-Fi and Bluetooth capabilities. This makes it an ideal choice for Internet of Things (IoT) applications, where connectivity and data transmission are crucial. In a smart parking system, the ESP32 serves as the central processing unit, gathering and processing data from various sensors deployed in parking spaces.

Infrared sensors play a pivotal role in this system by detecting the presence or absence of vehicles in individual parking spots. These sensors emit infrared light and measure the reflection from objects in the parking space. When a vehicle occupies a spot, the sensor's reflection changes, indicating occupancy. This data is then captured by the ESP32 microcontroller.

The ESP32 processes the sensor data and transmits it to a central server or cloud platform. This real-time data can be accessed by users through a mobile app or web interface, providing up-to-date information on parking space availability. Users can easily locate available parking spots, reducing the time spent searching for parking and minimizing traffic congestion.

Additionally, the smart parking system can integrate features such as reservation and payment options, further enhancing user convenience. By automating the parking management process and providing real-time updates, this system not only improves the efficiency of parking space utilization but also enhances the overall user experience.

**ABSTRACT:**

* The escalating demand for efficient urban parking solutions has driven the development of innovative smart parking systems. This report presents a smart parking system that employs ESP32 microcontrollers and infrared (IR) sensors to enhance parking management in urban environments. The system addresses common issues such as congestion, inefficient space utilization, and driver frustration by leveraging real-time data processing and connectivity.

* The ESP32 microcontroller, known for its low cost and robust connectivity features, serves as the core processing unit of the system. It integrates seamlessly with IR sensors installed in parking spaces to detect vehicle presence. These sensors emit infrared light and measure the reflected signal to determine whether a parking spot is occupied or available. The ESP32 processes this information and communicates it to a central server or cloud-based platform.

* This real-time data is made accessible to users through a mobile application or web interface, providing them with up-to-date information on parking space availability. The system not only facilitates quicker and more efficient parking for drivers but also contributes to reducing traffic congestion and lowering stress associated with finding parking.

* Additionally, the smart parking system can include advanced features such as space reservation and automated payment options, further enhancing user convenience and streamlining the parking experience. By improving the management of parking spaces and offering a user-friendly interface, the system optimizes space utilization and supports smarter urban infrastructure.

* In summary, the integration of ESP32 microcontrollers with IR sensors offers a cost-effective and efficient solution to contemporary parking challenges. This smart parking system exemplifies how technology can transform urban parking management, providing tangible benefits for both drivers and city planners.

**PROPOSED SYSTEM:**

**System Architecture:**

* **IR Sensors:**

**Function:** Installed in each parking space to detect the presence or absence of vehicles. They operate by emitting infrared light and measuring the reflection off objects in the parking spot.

**Data Collected:** Occupancy status of each parking space.

* **ESP32 Microcontroller:**

**Function:** Serves as the central unit for data collection and processing. Each ESP32 is connected to multiple IR sensors to gather data from several parking spots.

**Connectivity:** Equipped with Wi-Fi and Bluetooth capabilities for communication with the cloud platform and user devices.

**Data Handling:** Processes sensor data, formats it, and transmits it to the ThingzMate cloud platform.

* **ThingzMate Cloud Platform:**

**Function:** Provides cloud-based data management, storage, and analysis. It acts as the central repository for parking space data and facilitates communication between the ESP32 microcontrollers and user interfaces.

**Features:** Real-time data aggregation, storage, analytics, and API services for data retrieval and user interaction.

* **User Interface:**

**Function:** Includes a mobile app or web interface that allows users to view parking availability, get directions to open spaces, and access additional services.

**Interaction:** Retrieves real-time data from the ThingzMate cloud platform and displays it to users in an intuitive format.

**COMPONENTS REQUIRED:**

**SOFTWARE**

* Arduino Ide.
* Thingzmate.

**HARDWARE:**

1. ESP32 Microcontroller

2. Infrared (IR) Sensors

3. Power Supply

4. Relay Module

5. Jumper wires

6. Bread Board

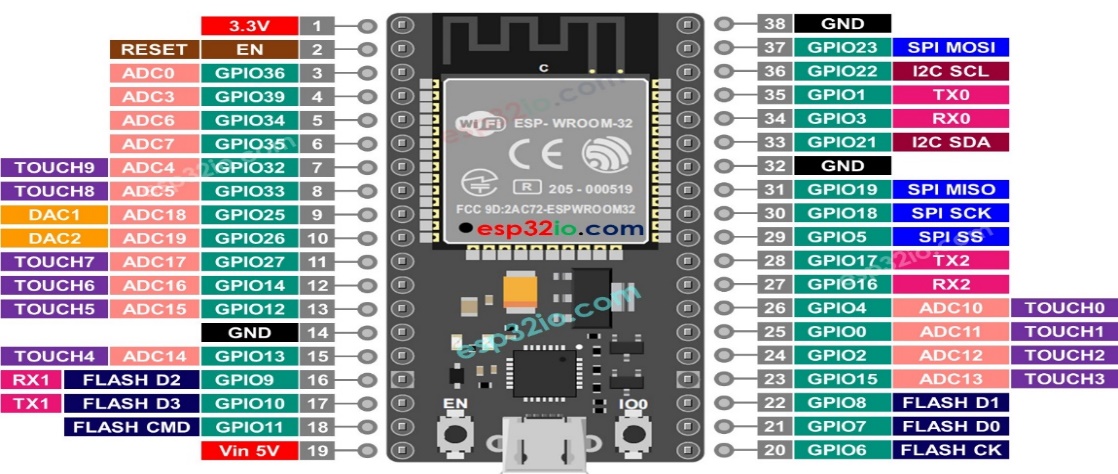
7. Servo motor

**ESP32 MICROCONTROLLER:**

The ESP32 is a highly versatile microcontroller developed by Espressif Systems, designed for a wide range of applications, particularly in the Internet of Things (IoT) space. It is renowned for its combination of high performance, integrated wireless connectivity, and a rich set of features, all at a low cost. The ESP32 is commonly used in projects that require both Wi-Fi and Bluetooth capabilities, making it suitable for smart home devices, sensor networks, and wearable technology.

**Key Features:**

* **Dual-Core Processor**: Features a dual-core Tensilica LX6 microprocessor running up to 240 MHz.
* **Connectivity**: Includes Wi-Fi (802.11 b/g/n) and Bluetooth (Classic and BLE).
* **Memory**: Typically comes with 520 KB of SRAM and supports external flash memory.
* **I/O Pins**: Offers numerous GPIO (General Purpose Input/Output) pins with various functionalities.



**2.IR SENSOR:**



**1. Vehicle Detection:**

* **Function:** The primary role of IR sensors in the smart parking system is to detect the presence or absence of vehicles in individual parking spaces.
* **Mechanism:** IR sensors emit infrared light and measure the reflection of this light. When a vehicle is parked in a space, it reflects the infrared light differently compared to an empty space. This change in reflection is detected by the sensor, indicating that the space is occupied.

**2. Data Collection:**

* **Function:** IR sensors continuously monitor parking spaces and provide real-time data on their status.
* **Output:** The sensor outputs signals that indicate whether a parking space is occupied or available. This information is crucial for the ESP32 microcontroller to process and transmit to the cloud platform.

**3. Real-Time Monitoring:**

* **Function:** IR sensors enable real-time monitoring of parking spaces, which is essential for providing up-to-date information to users.
* **Impact:** By detecting changes in parking space occupancy in real-time, the system can promptly update the status of parking spaces in the user interface, helping drivers find available spots more efficiently.

**4. Integration with ESP32:**

* **Function:** IR sensors are connected to the ESP32 microcontroller, which processes the data received from the sensors.
* **Role:** The ESP32 uses the data from the IR sensors to determine the occupancy status of each parking space and then communicates this information to the ThingzMate cloud platform or user interface.

**5. Enhancing System Accuracy:**

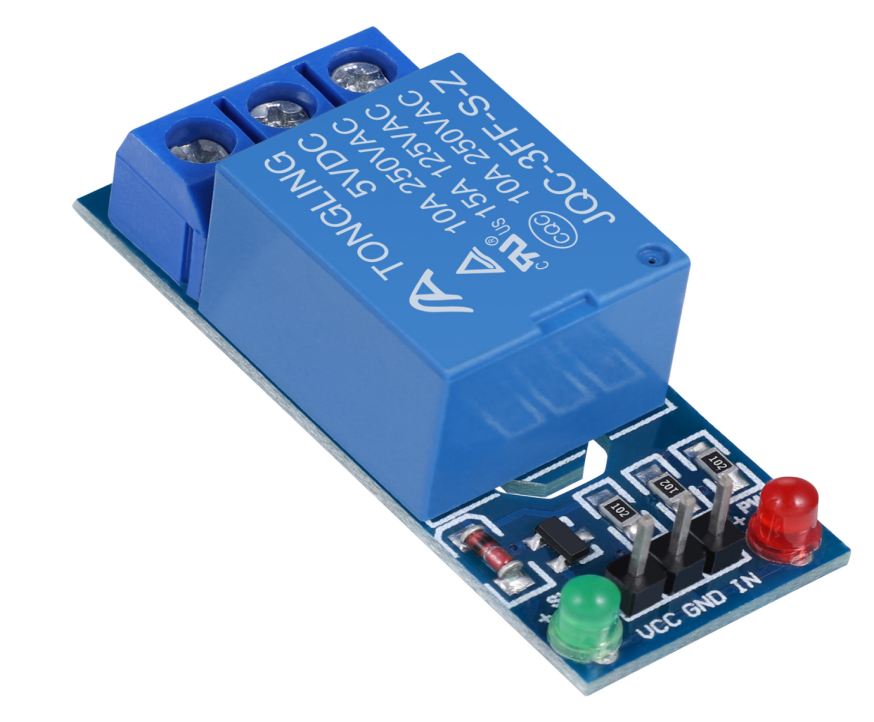
* **Function:** IR sensors contribute to the overall accuracy and reliability of the smart parking system.
* **Impact:** Accurate detection of vehicle presence helps in reducing false positives and negatives, ensuring that the parking availability information provided to users is precise and reliable.

**6. Supporting Automation:**

* **Function:** The automated detection capabilities of IR sensors support the seamless operation of the smart parking system.
* **Benefit:** By automating the process of detecting and reporting parking space status, the system reduces the need for manual intervention and improves operational efficiency.

**RELAY MODULE:**

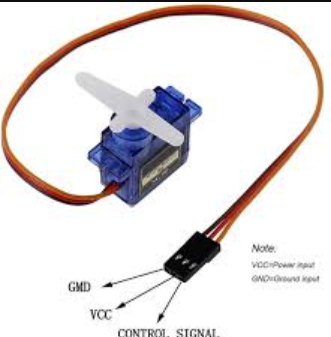
* A relay consists of an electromagnet, a movable armature, a set of contacts, and a spring.
* When a low-power signal (typically from a GPIO pin of a microcontroller) is applied to the relay's coil, it generates a magnetic field that pulls the armature, closing or opening the contacts.
* This action either completes or interrupts the circuit connected to the high-power device (in this case, street lights).



**POWER SUPPLY:**

* Power supply is a reference to a source of electrical power.
* A device or system that supplies electrical or other types of energy to an output load or group of loads is called a power supply unit or PSU.
* The term is most commonly applied to electrical energy supplies, less often to mechanical ones, and rarely to others.
* This power supply section is required to convert AC signal to DC signal and also to reduce the amplitude of the signal.
* The available voltage signal from the main is 230V/50Hz which is an AC voltage, but the required is DC voltage with the amplitude of +5V and +12V for varies applications

**SERVO MOTOR:**



A **servo motor** is a specialized type of electric motor designed for precise control of angular position, speed, and acceleration. It is commonly used in applications where accurate and consistent movement is required. Here’s a detailed overview of servo motors:

#### **1. Components of a Servo Motor:**

* **Motor:** Provides the rotational motion.
* **Feedback Device (Encoder or Potentiometer):** Measures the position of the motor shaft and provides feedback to the control system to ensure accurate positioning.
* **Control Circuit:** Receives commands from a controller and adjusts the motor’s position based on feedback.
* **Gearbox:** Reduces the speed of the motor and increases torque, which is essential for precise control.

#### **2. Working Principle:**

* **Input Signal:** The servo motor receives a control signal (typically a pulse-width modulation, or PWM signal) that specifies the desired position.
* **Feedback Loop:** The feedback device continuously monitors the actual position of the motor shaft and compares it with the desired position.
* **Adjustment:** The control circuit adjusts the motor’s position by changing the power supplied to the motor until the actual position matches the desired position.
* **Positioning:** Once the desired position is reached, the motor maintains that position with high accuracy.

#### **3. Types of Servo Motors:**

* **Standard Servo Motors:**
  + **Application:** Commonly used in hobby applications like remote-controlled vehicles.
  + **Features:** Typically has a range of about 180 degrees of movement.
* **Continuous Rotation Servo Motors:**
  + **Application:** Used where continuous rotation is required, such as in robots and automated systems.
  + **Features:** Can rotate continuously in both directions but lacks precise positioning control.
* **Digital Servo Motors:**
  + **Application:** Offers more precise control and faster response times compared to analog servos.
  + **Features:** Utilizes a digital signal to control the motor, improving accuracy and performance.

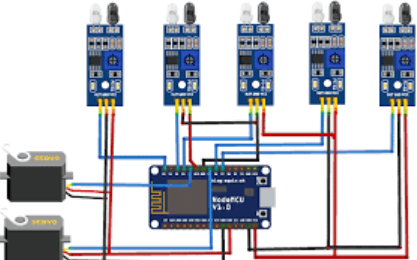
**Jumper Wires:**

* **Function:** Jumper wires are flexible electrical wires used to make connections between different components in a circuit.
* **Features:**
  + **Types:** Available in different lengths and colors, typically with connectors at both ends (male-to-male, female-to-female, or male-to-female).
  + **Usage:** Commonly used in prototyping and breadboarding to establish connections without soldering.
  + **Application:** Ideal for linking components like microcontrollers, sensors, and other electronic parts in development and testing phases.

**Breadboard:**

* **Function:** A breadboard is a reusable, solderless platform used for prototyping electronic circuits.
* **Features:**
  + **Structure:** Consists of a grid of holes arranged in rows and columns, with internal conductive strips that allow for easy insertion of components and wires.
  + **Usage:** Components are inserted into the breadboard’s holes and connected by jumper wires to create and test circuit designs without permanent connections.
  + **Application:** Useful for experimenting with circuit designs, debugging, and rapid prototyping before finalizing a circuit design.

**CIRCUIT DIAGRAM:**



**CODE:**

#include <WiFi.h>

#include <HTTPClient.h>

#include <ESP32Servo.h> // Include the Servo library

// Define GPIO pins for the IR sensors

#define IR\_SENSOR\_PIN\_1 14

#define IR\_SENSOR\_PIN\_2 27

#define IR\_SENSOR\_PIN\_3 26

#define IR\_SENSOR\_PIN\_4 25

// Define GPIO pin for the Servo motor

#define SERVO\_PIN 12

// Optional LED pins for visual feedback

#define LED\_PIN\_1 32

#define LED\_PIN\_2 33

#define LED\_PIN\_3 34

#define LED\_PIN\_4 35

// WiFi credentials

const char\* ssid = ""; // Replace with your WiFi SSID

const char\* password = ""; // Replace with your WiFi password

// Thingzmate settings

const String serverName = "<https://console.thingzmate.com/api/v1/device-types/523454/devices/smart/uplink>"; // Replace with your Thingzmate endpoint URL

const String authorizationToken = "Bearer 4d20a39989ca171ae028e8b19d8dd743"; // Replace with your Thingzmate API token

// Create a Servo object

Servo myServo;

void setup() {

Serial.begin(115200); // Initialize serial communication for debugging

// Connect to WiFi

WiFi.begin(ssid, password);

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

// Wait until connected to WiFi

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("Connected to WiFi");

Serial.print("IP Address: ");

Serial.println(WiFi.localIP());

// Set IR sensor pins as input

pinMode(IR\_SENSOR\_PIN\_1, INPUT);

pinMode(IR\_SENSOR\_PIN\_2, INPUT);

pinMode(IR\_SENSOR\_PIN\_3, INPUT);

pinMode(IR\_SENSOR\_PIN\_4, INPUT);

// Set LED pins as output (optional)

pinMode(LED\_PIN\_1, OUTPUT);

pinMode(LED\_PIN\_2, OUTPUT);

pinMode(LED\_PIN\_3, OUTPUT);

pinMode(LED\_PIN\_4, OUTPUT);

// Initialize the Servo motor

myServo.attach(SERVO\_PIN);

myServo.write(0); // Set initial position of the servo to 0 degrees

}

void loop() {

// Read the IR sensor values

int sensorValue1 = digitalRead(IR\_SENSOR\_PIN\_1);

int sensorValue2 = digitalRead(IR\_SENSOR\_PIN\_2);

int sensorValue3 = digitalRead(IR\_SENSOR\_PIN\_3);

int sensorValue4 = digitalRead(IR\_SENSOR\_PIN\_4);

// Send data to Thingzmate Cloud

sendSensorData(sensorValue1, sensorValue2, sensorValue3, sensorValue4);

// Control the servo based on sensor data (example logic)

if (sensorValue1 == LOW) {

myServo.write(90); // Move servo to 90 degrees if car detected in sensor 1

} else {

myServo.write(0); // Move servo back to 0 degrees if no car detected

}

delay(1000); // Wait for 1 second before the next reading

}

void sendSensorData(int s1, int s2, int s3, int s4) {

// Print sensor status to Serial Monitor

Serial.print("Sensor 1: ");

Serial.println(s1 == LOW ? "Car detected" : "No car detected");

Serial.print(", Sensor 2: ");

Serial.println(s2 == LOW ? "Car detected" : "No car detected");

Serial.print(", Sensor 3: ");

Serial.println(s3 == LOW ? "Car detected" : "No car detected");

Serial.print(", Sensor 4: ");

Serial.println(s4 == LOW ? "Car detected" : "No car detected");

if (WiFi.status() == WL\_CONNECTED) {

HTTPClient http;

// Prepare the URL and data to send

String postData = "{\"sensor1\": " + String(s1) + ", \"sensor2\": " + String(s2) +", \"sensor3\": " + String(s3) + ", \"sensor4\": " + String(s4) + "}";

http.begin(serverName); // Specify the URL

http.addHeader("Content-Type", "application/json"); // Specify content-type

http.addHeader("Authorization", authorizationToken); // Add authorization token

// Send HTTP POST request

int httpResponseCode = http.POST(postData);

// Check response

if (httpResponseCode > 0) {

String response = http.getString();

Serial.println("Response Code: " + String(httpResponseCode));

Serial.println("Response: " + response);

} else {

Serial.print("Error Code: ");

Serial.println(httpResponseCode);

}

http.end(); // Free resources

} else {

Serial.println("Error: Not connected to WiFi");

}

// Control LEDs (optional)

digitalWrite(LED\_PIN\_1, s1 == LOW ? HIGH : LOW);

digitalWrite(LED\_PIN\_2, s2 == LOW ? HIGH : LOW);

digitalWrite(LED\_PIN\_3, s3 == LOW ? HIGH : LOW);

digitalWrite(LED\_PIN\_4, s4 == LOW ? HIGH : LOW);

}

**WORKING :**

* **1. IR Sensor Working**

**Components**:

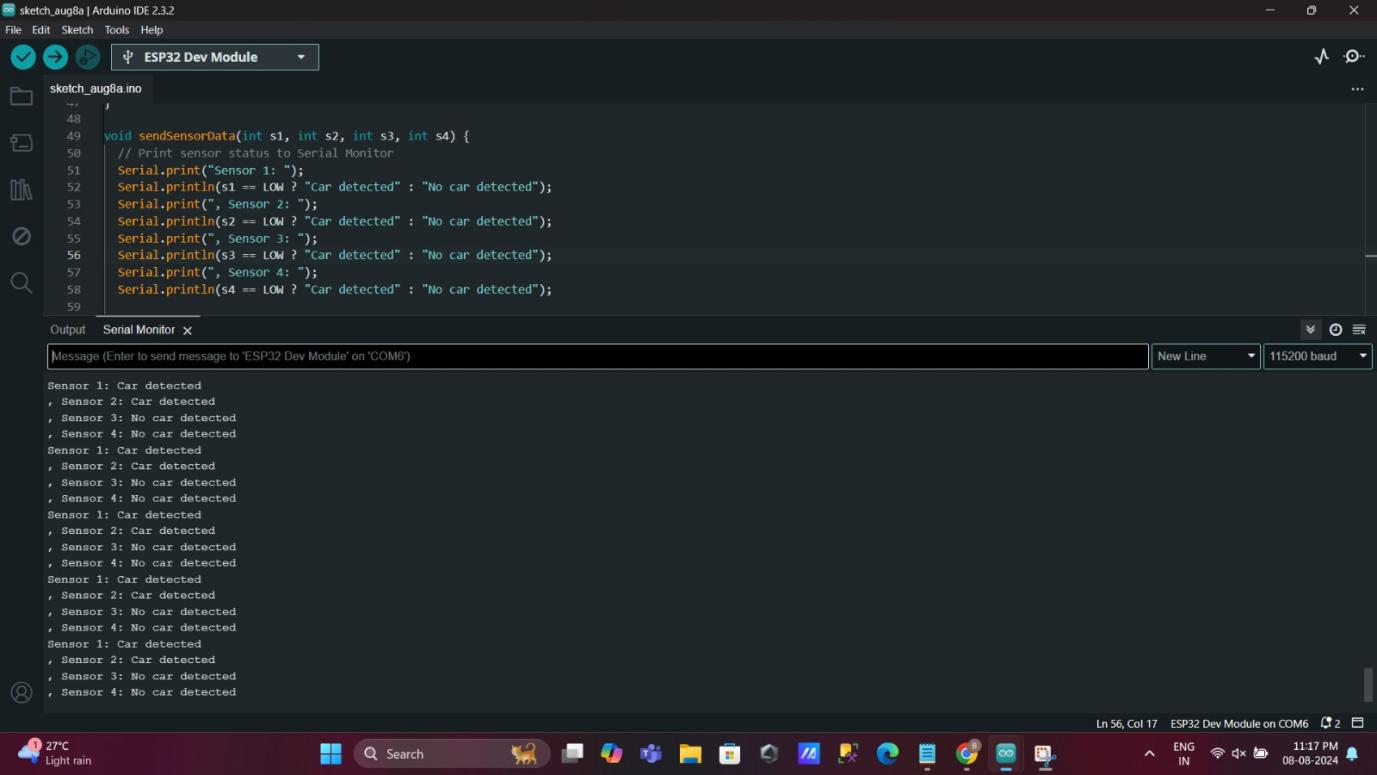
* + **IR Emitter**: Sends out infrared light.
  + **IR Receiver**: Detects the reflected infrared light.
* **How It Works**:
  + When a vehicle is present, it reflects the infrared light emitted by the IR LED back to the receiver.
  + **Digital Output**: High or low depending on vehicle detection.
  + **Analog Output**: Voltage proportional to the amount of reflected light.

### **2. Setting Up ThingzMate Cloud**

#### **a. Create a ThingzMate Account**

* **Sign Up**: Go to the ThingzMate Cloud website and sign up for an account if you don’t already have one.
* **Create a New Device**:
  + **Add Device**: Go to the "Devices" section and add a new device. This will involve providing a name for your device and getting an API key or device token.
* **Create a New Channel**:
  + **Add Channel**: Create a channel where your sensor data will be sent. Define the data fields (e.g., field1 for occupancy status).
* **Get API Key**:
  + Obtain the API Key for the device you created. This will be used to authenticate your data submissions.

**OUTPUT:**



**STEPS TO FOLLOW:**

**·** Check that Serial.begin(9600); is present in your setup() function.

· Match the baud rate in the Serial Monitor to 9600.

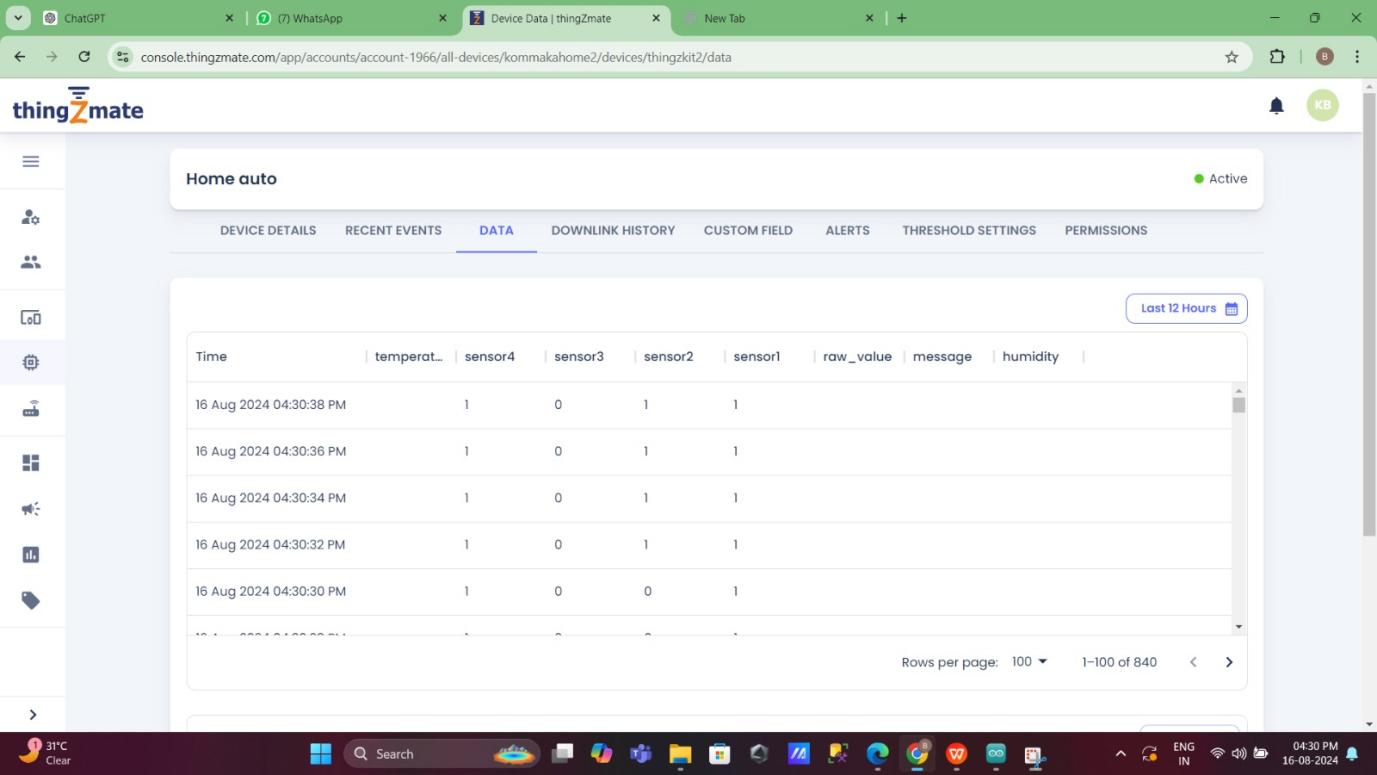
· Verify the COM port in the Serial Monitor is correctly selected.

· Ensure no other application is using the same COM port.

· Restart your IDE and try again.

**IoT Integration:**

* **Cloud Integration:** Describe how data is sent to a cloud service for monitoring (e.g., using MQTT or HTTP).
* **CLOUD OUTPUT:**



* **Dashboard:** Discuss creating a user interface for monitoring and controlling the street lights remotely.

**Testing and Results:**

* **Test Cases:** Describe various scenarios tested, such as different lighting conditions and motion detection.
* **Results:** Provide screenshots or data showing the system's response to the test cases.

**APPLICATIONS:**

### **1. Parking Management Systems**

* **Real-Time Space Availability**: Provides real-time updates on available parking spaces, helping drivers find open spots more quickly.
* **Dynamic Pricing**: Can be integrated with dynamic pricing models where the cost of parking changes based on demand and occupancy.

### **2. Smart Cities**

* **Urban Planning**: Data on parking space usage can inform city planners and help in designing better urban spaces.
* **Traffic Management**: Reduces congestion by guiding drivers to available parking spots, leading to smoother traffic flow.

### **3. Commercial Facilities**

* **Shopping Malls**: Enhances the customer experience by directing shoppers to available parking spaces, improving overall satisfaction.
* **Airports**: Streamlines the parking process for travelers, reducing the time spent searching for parking and improving airport operations.

### **4. Residential Areas**

* **Community Parking**: Helps in managing parking in residential complexes by showing available spaces, especially in densely populated areas.
* **Guest Parking**: Facilitates efficient management of guest parking spaces.

### **5. Event Venues**

* **Concerts and Sports Events**: Provides real-time parking information to attendees, helping them find parking more easily during large events.
* **Conference Centers**: Enhances the visitor experience by providing parking availability information.

### **6. Educational Institutions**

* **Campus Parking**: Manages parking spaces in large educational campuses, helping students and staff find parking more efficiently.
* **Event Management**: Assists in parking management during campus events like open houses or graduation ceremonies.

### **7. Healthcare Facilities**

* **Hospital Parking**: Improves the parking experience for patients and visitors at hospitals, reducing stress and wait times.
* **Emergency Services**: Ensures that parking spaces are available for emergency vehicles and staff.

### **8. Integration with Other Systems**

* **Navigation Apps**: Integrates with navigation systems to provide users with the best routes to available parking spots.
* **Smart Building Systems**: Works with smart building management systems to optimize space usage and resource allocation.

### **9. Environmental Benefits**

* **Reduced Emissions**: Decreases the time spent searching for parking, which can lead to lower vehicle emissions and reduced fuel consumption.
* **Efficient Use of Resources**: Helps in making better use of existing parking infrastructure, potentially reducing the need for new parking constructions.

### **10. Analytics and Reporting**

* **Usage Analytics**: Collects and analyzes data on parking space usage patterns, helping businesses and municipalities make data-driven decisions.
* **Operational Insights**: Provides insights into peak parking times and usage trends, enabling better operational planning and resource allocation.

### **11. Customer Service and Satisfaction**

* **Enhanced User Experience**: Improves overall user experience by reducing the time and effort required to find parking.
* **Customer Feedback**: Collects feedback on parking availability and system performance to continuously improve the service.

### **12. Security and Monitoring**

* **Surveillance Integration**: Can be integrated with surveillance systems to monitor the security of parking areas.
* **Automated Alerts**: Provides automated alerts for unauthorized parking or abnormal conditions.

**CONCLUSION:**

The smart parking system leveraging IR sensors, ESP32 microcontroller, and ThingzMate Cloud represents a significant advancement in the management and optimization of parking spaces. Here’s a summary of the key points:

### **1. System Integration and Functionality**

* **IR Sensors**: Provide accurate real-time detection of vehicle presence in parking spaces. They are straightforward to integrate and cost-effective for both digital and analog sensing.
* **ESP32 Microcontroller**: Acts as the central processing unit, reading sensor data and managing communication with the cloud. Its Wi-Fi capabilities facilitate seamless data transmission.
* **ThingzMate Cloud**: Offers a robust platform for data storage, visualization, and analysis. It enables users to monitor parking space availability remotely and in real-time.

### **2. Benefits**

* **Enhanced Efficiency**: Reduces the time spent searching for parking, improving overall efficiency and convenience for users.
* **Data-Driven Decisions**: Provides valuable insights into parking space usage and patterns, aiding in better planning and management.
* **Operational Improvements**: Helps businesses and municipalities manage parking resources more effectively, potentially reducing the need for new infrastructure.

### **3. Applications**

* **Urban Management**: Supports smart city initiatives by improving traffic flow and urban planning.
* **Commercial and Residential**: Enhances customer and resident satisfaction through efficient parking management.
* **Event Management**: Streamlines parking for large-scale events, improving the overall experience for attendees.

### **4. Environmental and Social Impact**

* **Reduced Emissions**: Decreases vehicle emissions by minimizing the time spent searching for parking.
* **Improved Safety and Security**: Potential integration with security systems for enhanced monitoring and protection of parking areas.

### **5. Future Potential**

* **Integration with Advanced Systems**: Can be integrated with other smart systems, such as navigation apps and building management systems, for a more comprehensive solution.
* **Scalability**: The system can be scaled to accommodate larger parking areas or adapted for different types of sensors and communication protocols.

**References:**

* Cite all references used in the project, including websites, papers, and books.
* Referred the IOT websites and esp32 websites for the circuit connections.