PROJECT: SMART PARKING
PHASE-5:

Project objectives:

Real-Time Monitoring System for Parking Space Management Services is the evolution of traditional parking system that it does not only provide live information to users in order to make it easy for them to look for vacant parking lot, it also gives authority to operators to monitor and perform simulations to illustrate the real parking system. The smart parking systems are powered by the internet of things (IoT) technology. What this technology does is collect the data from the parking area about free and occupied slots and sends it to users. The users can easily check things through their smartphones or tablets and choose the right parking place.

Parking guidance and information (**PGI**) systems, or **car park guidance systems**, present drivers with dynamic information on parking within controlled areas. The systems combine traffic monitoring, communication, processing and variable message sign technologies to provide the service.

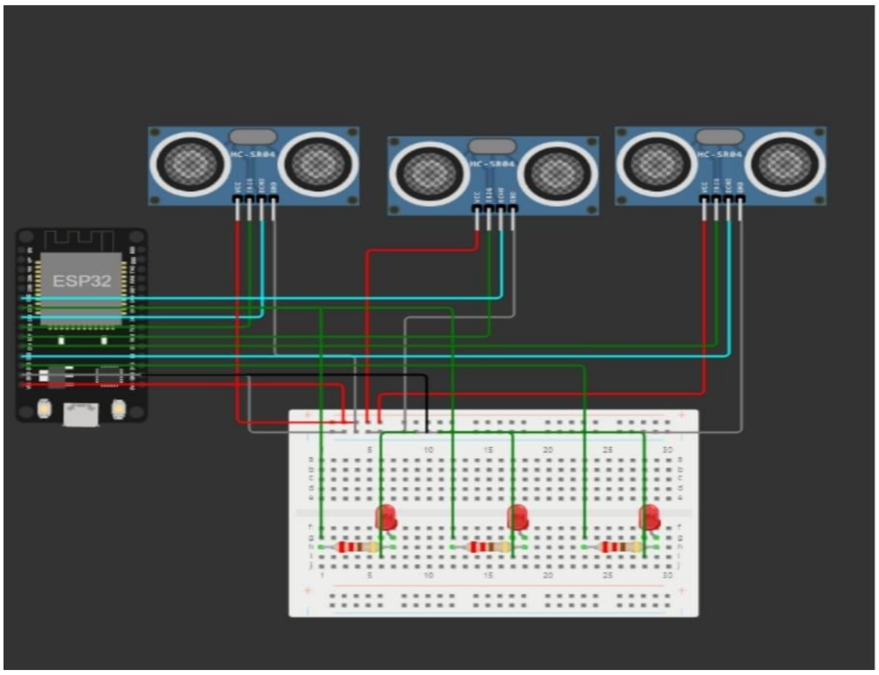
Parking guidance offers a smart solution for optimising available space in car parks. Through sensors and monitoring systems, drivers are guided precisely and efficiently to available spaces. This **reduces the time spent searching for a space by 45% to 55%**, avoiding congestion and improving traffic flow.
As a result, the operator can optimise parking capacity and the user saves time, making this solution advantageous for both drivers and car park owners.

IOT sensor setup:

Step 1: Circuit Setup
Connect the HC-SR04 sensor to the ESP32 as follows:
- HC-SR04 VCC to ESP32 5V or 3.3V
- HC-SR04 GND to ESP32 GND
- HC-SR04 Trig (Trigger) to an ESP32 GPIO pin (e.g., GPIO2)
- HC-SR04 Echo to an ESP32 GPIO pin (e.g., GPIO15)

You can also add LEDs to indicate parking space availability, connecting them to additional GPIO pins on the ESP32.
Step 2: Arduino IDE Setup
Make sure you have the Arduino IDE set up for ESP32 development. You'll need to install the ESP32 board in the Arduino IDE.

1. Go to "File" > "Preferences" in the Arduino IDE.
2. In the "Additional Boards Manager URLs" field, add this URL: `https://dl.espressif.com/dl/package_esp32_index.json`
3. Click "OK" and close the Preferences window.
4. Go to "Tools" > "Board" > "Boards Manager."
5. Search for "esp32" and install the "esp32" board by Espressif Systems.
6. Select your ESP32 board under "Tools" > "Board."



Mobile App Development :

1. *Launch the App:*

   - Release the app on Google Play Store and Apple App Store.

   - Create a marketing strategy to promote the app.

2. *Marketing and Promotion:*

   - Use digital marketing, social media, and partnerships with parking facilities to attract users.

3. *Feedback and Iteration:*

   - Continuously collect user feedback and make improvements.

   - Add new features and enhancements based on user suggestions.

4. *Maintenance:*

   - Regularly update the app to fix bugs, improve performance, and maintain compatibility with the latest operating systems.

5. *Revenue Model:*

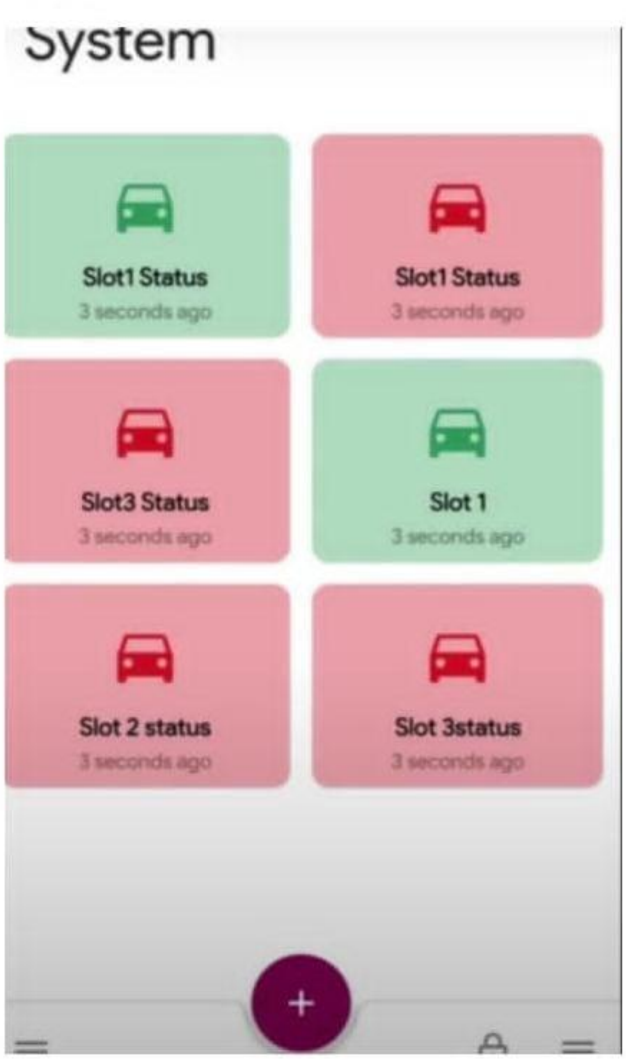
  - Determine how you will monetize the app, whether through booking fees, subscription models, or advertising.

6. *Legal Compliance:*

  - Ensure that your app complies with local parking regulations and permits.

7. *Scaling:*

  - Consider expanding to new cities and partnering with more parking facilities as your user base grows.



ESP 32

Integration:

Integrating an ESP32 (a popular microcontroller and Wi-Fi module) into a smart parking system can be a cost-effective and efficient way to monitor parking spot occupancy. Here's a high-level guide on how to integrate an ESP32 into a smart parking solution:

1. *Hardware Selection:*

  - Choose the ESP32 development board or module that best suits your project requirements, considering factors like power consumption, range, and the number of GPIO pins.

2. *Sensor Selection:*

  - Select appropriate sensors to detect parking spot occupancy. Common choices include ultrasonic sensors, infrared sensors, or magnetic sensors.

3. *Software Development:*

  - Develop firmware for the ESP32 using the Arduino IDE or the ESP-IDF (Espressif IoT Development Framework). The firmware should include the following functionalities:

    - Initialize the ESP32's Wi-Fi capabilities to connect to your network.

    - Configure the sensor(s) to detect vehicle presence in parking spots.

    - Implement logic to read sensor data and transmit it to a central server or cloud platform.

    - Establish secure communication protocols (e.g., HTTPS) to send data to the server.

4. *Data Transmission:*

  - Send sensor data to a central server or cloud platform for processing. You can use MQTT, HTTP, or other communication protocols to transmit data.

5. *Server/Cloud Backend:*

  - Set up a server or cloud backend to receive, process, and store parking spot occupancy data.

  - Develop APIs for communication with the ESP32 devices.

  - Implement data analytics and visualization to display parking spot availability in real time.

6. *User Interface:*

  - Create a user-friendly interface (a mobile app or a web app) for users to check parking spot availability. This interface should fetch and display data from your server/cloud platform.

7. *Security and Authentication:*

  - Implement security measures to protect data transmission and storage.

  - Consider user authentication for administrative access and to prevent unauthorized changes to parking spot data.

8. *Power Management:*

  - Optimize power management to prolong the life of ESP32 devices, especially if they run on battery power. You can use sleep modes and other energy-saving techniques.

9. *Testing and Calibration:*

  - Rigorously test the system to ensure the accurate detection of parking spot occupancy. Calibrate sensors as needed to improve accuracy.

10. *Scalability:*

  - Plan for scalability by adding more ESP32 devices to cover a larger parking area.

11. *Maintenance:*

  - Regularly monitor the health of ESP32 devices and replace or maintain them as needed.

  - Keep the software up to date and add new features as required.

12. *Documentation:*

  - Document the system architecture, firmware, and any custom APIs for future reference and maintenance.

Integrating ESP32 devices into a smart parking solution provides a cost-effective way to monitor parking spot occupancy, and it can be customized to suit your specific requirements. It also allows for real-time tracking and data analysis, which can enhance the user experience and help optimize parking resources.

CODE IMPLEMENTATION :

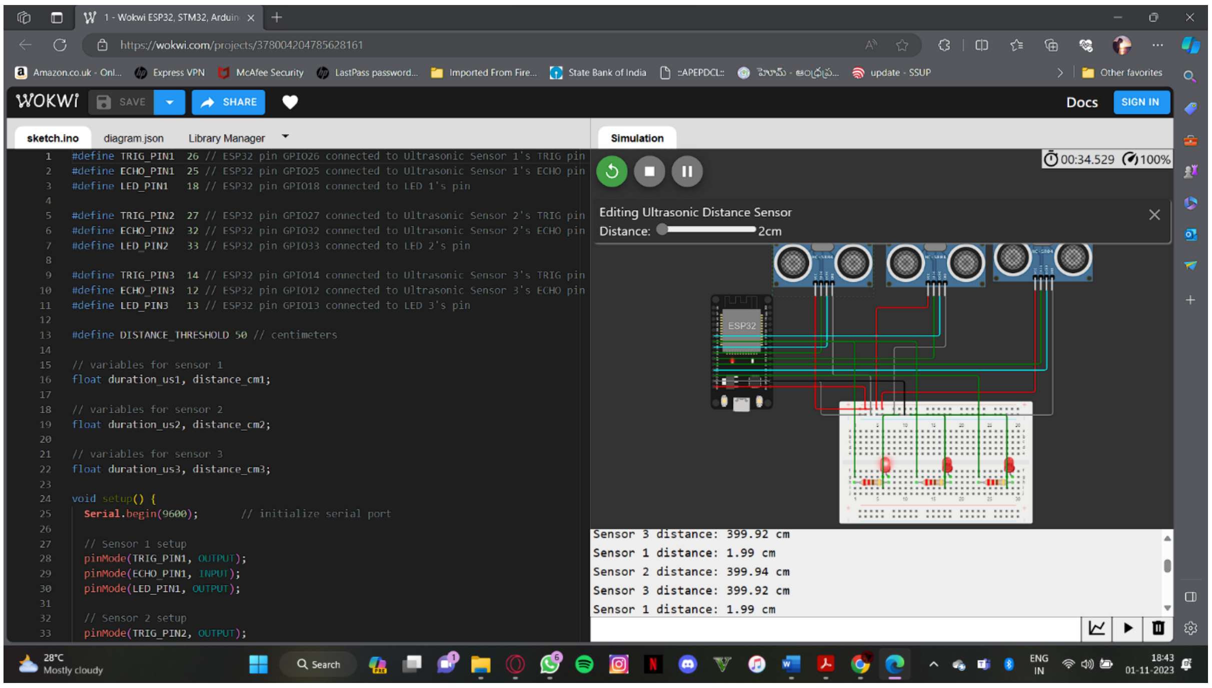
CODE:

```
#define TRIG_PIN1 26 // ESP32 pin GPIO26 connected to Ultrasonic Sensor 1's TRIG pin

#define ECHO_PIN1 25 // ESP32 pin GPIO25 connected to Ultrasonic Sensor 1's ECHO pin

#define LED_PIN1 18 // ESP32 pin GPIO18 connected to LED 1's pin

#define TRIG_PIN2 27 // ESP32 pin GPIO27 connected to Ultrasonic Sensor 2's TRIG pin

#define ECHO_PIN2 32 // ESP32 pin GPIO32 connected to Ultrasonic Sensor 2's ECHO pin

#define LED_PIN2 33 // ESP32 pin GPIO33 connected to LED 2's pin

#define TRIG_PIN3 14 // ESP32 pin GPIO14 connected to Ultrasonic Sensor 3's TRIG pin
```

```
#define ECHO_PIN3 12 // ESP32 pin GPIO12 connected to Ultrasonic Sensor 3's
ECHO pin

#define LED_PIN3 13 // ESP32 pin GPIO13 connected to LED 3's pin

#define DISTANCE_THRESHOLD 50 // centimeters

// variables for sensor 1

float duration_us1, distance_cm1;

// variables for sensor 2

float duration_us2, distance_cm2;

// variables for sensor 3

float duration_us3, distance_cm3;

void setup() {

Serial.begin(9600); // initialize serial port

// Sensor 1 setup

pinMode(TRIG_PIN1, OUTPUT);

pinMode(ECHO_PIN1, INPUT);

pinMode(LED_PIN1, OUTPUT);

// Sensor 2 setup

pinMode(TRIG_PIN2, OUTPUT);

pinMode(ECHO_PIN2, INPUT);

pinMode(LED_PIN2, OUTPUT);

// Sensor 3 setup

pinMode(TRIG_PIN3, OUTPUT);

pinMode(ECHO_PIN3, INPUT);

pinMode(LED_PIN3, OUTPUT);

}

void loop() {

// Sensor 1 measurements

digitalWrite(TRIG_PIN1, HIGH);

delayMicroseconds(10);

digitalWrite(TRIG_PIN1, LOW);

duration_us1 = pulseIn(ECHO_PIN1, HIGH);

distance_cm1 = 0.017 * duration_us1;

// Sensor 2 measurements
```

```
digitalWrite(TRIG_PIN2, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN2, LOW);
duration_us2 = pulseIn(ECHO_PIN2, HIGH);
distance_cm2 = 0.017 * duration_us2;
// Sensor 3 measurements
digitalWrite(TRIG_PIN3, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN3, LOW);
duration_us3 = pulseIn(ECHO_PIN3, HIGH);
distance_cm3 = 0.017 * duration_us3;
// Control LED 1 based on sensor 1
if (distance_cm1 < DISTANCE_THRESHOLD) {
digitalWrite(LED_PIN1, HIGH);
} else {
digitalWrite(LED_PIN1, LOW);
}
// Control LED 2 based on sensor 2
if (distance_cm2 < DISTANCE_THRESHOLD) {
digitalWrite(LED_PIN2, HIGH);
} else {
digitalWrite(LED_PIN2, LOW);
}
// Control LED 3 based on sensor 3
if (distance_cm3 < DISTANCE_THRESHOLD) {
digitalWrite(LED_PIN3, HIGH);
} else {
digitalWrite(LED_PIN3, LOW);
}
// Print values to Serial Monitor
Serial.print("Sensor 1 distance: ");
Serial.print(distance_cm1);
Serial.println(" cm");
Serial.print("Sensor 2 distance: ");
Serial.print(distance_cm2);
Serial.println(" cm");
Serial.print("Sensor 3 distance: ");
Serial.print(distance_cm3);
Serial.println(" cm");
// Delay for a short time before repeating the measurement
delay(100); // Adjust this delay as needed}
```

OUTPUT:



```
#define TRIG_PIN1  26 // ESP32 pin GPIO26 connected to Ultrasonic Sensor 1's TRIG pin
#define ECHO_PIN1  25 // ESP32 pin GPIO25 connected to Ultrasonic Sensor 1's ECHO pin
#define LED_PIN1   18 // ESP32 pin GPIO18 connected to LED 1's pin

#define TRIG_PIN2  27 // ESP32 pin GPIO27 connected to Ultrasonic Sensor 2's TRIG pin
#define ECHO_PIN2  32 // ESP32 pin GPIO32 connected to Ultrasonic Sensor 2's ECHO pin
#define LED_PIN2   33 // ESP32 pin GPIO33 connected to LED 2's pin

#define TRIG_PIN3  14 // ESP32 pin GPIO14 connected to Ultrasonic Sensor 3's TRIG pin
#define ECHO_PIN3  12 // ESP32 pin GPIO12 connected to Ultrasonic Sensor 3's ECHO pin
#define LED_PIN3   13 // ESP32 pin GPIO13 connected to LED 3's pin

#define DISTANCE_THRESHOLD 50 // centimeters

// variables for sensor 1
float duration_us1, distance_cm1;

// variables for sensor 2
float duration_us2, distance_cm2;

// variables for sensor 3
float duration_us3, distance_cm3;

void setup() {
  Serial.begin(9600);       // initialize serial port

  // Sensor 1 setup
  pinMode(TRIG_PIN1, OUTPUT);
  pinMode(ECHO_PIN1, INPUT);
  pinMode(LED_PIN1, OUTPUT);

  // Sensor 2 setup
  pinMode(TRIG_PIN2, OUTPUT);
```

```
Sensor 3 distance: 399.92 cm
Sensor 1 distance: 1.99 cm
Sensor 2 distance: 399.94 cm
Sensor 3 distance: 399.92 cm
Sensor 1 distance: 1.99 cm
```

BENEFITS OF REAL TIME PARKING AVAILABILTY:

Real-time parking using IoT (Internet of Things) technology offers several benefits for both parking management and users. Here are some of the key advantages:

1. *Efficient Space Utilization:* Real-time parking systems can provide up-to-the-minute information on parking spot availability. This helps drivers quickly locate an open parking space, reducing the time and fuel wasted in searching for parking. It optimizes space utilization, making parking facilities more efficient.

2. *Reduced Congestion:* By guiding drivers directly to available parking spots, real-time IoT systems can decrease traffic congestion and minimize the environmental impact associated with circling in search of parking. This benefits both urban mobility and air quality.

3. *Improved User Experience:* Real-time parking apps and systems make it easier for users to find and secure parking. They reduce frustration and stress, leading to a more pleasant experience for drivers and passengers.

4. *Cost Savings:* Reduced time spent searching for parking translates to cost savings in terms of fuel and vehicle wear and tear. Drivers can also avoid expensive parking fines and tow-away fees.

5. *Environmentally Friendly:* By minimizing traffic congestion and the associated carbon emissions, real-time parking contributes to a more sustainable environment. It aligns with green initiatives and reduces the carbon footprint of urban transportation.

6. *Enhanced Safety:* Real-time parking systems can incorporate safety features like well-lit pathways, security monitoring, and emergency assistance. This helps create a safer environment for both drivers and pedestrians