Smart parking using Internet of Things
PROJECT DOCUMENTATION

TEAM ID	350
PROJECT NAME	Smart parking

1.INTRODUCTION

Our project idea is to implement an effective smart parking system using Internet of Things (IoT) to overcome the time consumption and frustration by providing parking slot availability to the drivers. By using microcontroller board, sensor, mobile application, cloud-based methodology. In this document, we will outline the problem statement, the steps involved in solving it, and the design thinking approach that will guide our project.

2.PROBLEM STATEMENT

The current state of urban parking is inefficient, causing congestion, pollution, and driver frustration due to the lack of available parking spaces and ineffective management. We have a dataset containing various features of (e.g., size, location, number of parking slots, ownership, parking fare) of parking area by considering time consumption and traffic due to in search of available parking space.

- Parking management influences drivers search time and cost for parking spaces.
- It may also cause traffic congestion.
- Finding a parking space in most metropolitan areas, especially during the rush hours, is difficult for drivers.

3.OBJECTIVE

- Parking space reservation can help drivers to reduce the search time dramatically.
- With the real-time parking slot representation, the drivers can find reserve their desired vacant parking spaces quickly.
- It reduces time in search of vacant parking spaces is reduced so it reduces traffic congestion caused due that.

4.PROBLEM IDENTIFIED

1.Scalability:

Adapting the smart parking system to handle increasing numbers of parking spaces and users can be challenging. Scaling the infrastructure to accommodate a growing user base while maintaining system performance and reliability is a significant character.

2.Power Consumption and Energy Efficiency:

IoT devices like sensors and communication modules require power to operate. Balancing the need for accurate and real-time data collection with power consumption is a key challenge. Energy-efficient designs and sustainable power sources must be considered.

3.Parking congestion:

Traditional parking systems struggle to manage high-density parking areas efficiently, leading to congestion during peak hours.

5.DESIGN THINKING APPROACH

Empathize:

Before diving into solving the problem, In this case, we need to understand the needs and pain points of both drivers and parking lot availability.

Actions:

- Conduct surveys, observations to gain insights into their experiences and challenges.
- Seek feedback from the drivers about the parking lot operators.

Define:

Based on our understanding of the problem, we will define the problem by synthesizing the information gathered during the empathy stage.

Objectives:

- Identify specific pain points, such as difficulty in finding parking slot.
- By using smart sensors in parking area.
- providing real time data accessing through application.

Ideate:

Brainstorm creative solutions to address the identified problems.

Actions:
-Encourage a diverse group of stakeholders to generate a wide range of ideas. Consider technologies like sensors, mobile apps, and data analytics.
-Implement a reliable communication network (e.g., WIFI, Lora wan) to connect sensors to a central server or cloud platform. Ensure consistent and real-time data transmission between sensors and the centralized system.
-Integrate navigation features into the mobile app that guide users to available parking spots using the shortest and most efficient routes.

Prototype
By creating a prototype of the "Smart Parking model" using IOT and to provide real time data transfer to the drivers on availability of parking lot.
Actions:
- Create a tangible representation such as a mock-up of a mobile app or a small-scale parking lot with sensor installations.
- Prototyping allows for early testing and refinement of ideas.
- By testing the prototype with a subset of the dataset to ensure it meets performance objectives.

Test
Evaluate the model's performance using appropriate metrics and gather feedback from users.

Actions:
- By gathering feedback from users and stakeholders by testing our prototype.
-Understand how well our solution addresses the defined problem and make adjustments based on user insights.
- Invite a small group of users to interact with the prototype. Ask them to use the app to reserve a parking spot and monitor the real-time availability data.

Implement

Once the prototype meets the defined objectives and receives positive feedback, proceed with full implementation.

Actions:
- Providing solution based on user feedback. Be open to making changes and improvements to ensure it meets user needs effectively.
-This may involve deploying sensors, developing a mobile app, or integrating existing infrastructure.
- Deploy the model as part of a production-ready web application.

6. DESIGN AND INNOVATIVE STRATEGIES

1. Microcontroller Selection:

For our project we chosen ESP32 as a microcontroller for their capabilities, power efficiency, and community support.

ESP8266 and ESP32 are low-cost, power-efficient microcontrollers with built-in Wi-Fi capabilities. They are well-suited for IoT applications, including smart parking, due to their connectivity features.

2. Sensor Selection:

Ultrasonic sensors for detecting vehicle presence. Also to detect the distance of the vehicle from the parking slot.

3. Connectivity:

WIFI enabling seamless communication between the devices and the central system. WIFI allows us to communicate via Wi-Fi with sensors or actuators mounted on ESP32 to create easily and quickly of our IoT System.

4. Cloud Platform:

Data transferred to cloud platform, we used Blynk for data storage, processing, and real-time analytics. Blynk is used to create a mobile app to receive data from the sensor, which is a popular Internet of Things (IoT) platform that allows us to build mobile applications to control and monitor our IoT devices.

5. Protocol:

The IoT device can communicate with Blynk either using the Blynk library, or via the Blynk HTTPs API. In order to minimize cellular communication between Blynk and the device, the Blynk HTTPs API will be used.
We used Blynk Library as a protocol in our project.

6. Public Platform:

Integrating a user-friendly mobile app for drivers to find available parking spaces and enhancing the overall experience. Here Blynk act as a mobile app.

7.LITERATURE SURVEY

ESP32: Role of ESP32 in smart parking

The operation of ESP32 in such a system starts with data collection. Ultrasonic sensors, discreetly installed in each parking space, continuously measure the distance to detect vehicle presence. These sensors transmit data to the ESP32, acting as the central controller for real-time processing and management of parking information. It processes sensor data to determine parking space occupancy, marking each space as occupied or available. The ESP32 ensures the status of each parking space is updated in real-time, providing accurate information to users. The ESP32 is configured to communicate with an IoT platform, establishing a secure connection to the cloud through protocols like MQTT or HTTP. This connectivity facilitates the transmission of parking data to the cloud, where it can be stored and accessed for remote monitoring. The ESP32 is responsible for data transmission. It sends information about parking space occupancy to the cloud platform, which includes data such as parking space status, sensor readings, and timestamps. A dedicated mobile app or web-based interface offers users real-time information on parking space availability, streamlining the parking experience. Users can easily find available spaces and make informed decisions about their parking needs. The system includes a real-time occupancy data which is facilitated by the ESP32. The ESP32 continues to manage the real-time data flow, ensuring that parking space status remains up-to-date and accessible to users. This operation enhances parking management, user experience, and urban mobility, offering a technology-driven solution without the need for camera modules.

8.WORKING PRINCIPLE

- The distance values from the Ultrasonic Sensor based on that the Status of slot is sensed and based on that the occupancy status of parking slot is detected
- Ultrasonic sensor is connected to ESP32 with C/C++ CODE and connecting the sensor with respective GPIO pin.
- The data is transferred to cloud platform with the help of WIFI.
- Now the sensor will show the available parking slot in stimulation and also in cloud platform.

Transferring and receiving data from ultrasonic sensor to cloud platform:

The ultrasonic sensor, discreetly placed in each parking space, continuously measures distances to detect vehicle presence. The ESP32 at the heart of this system, employs code to collect and process the sensor data.

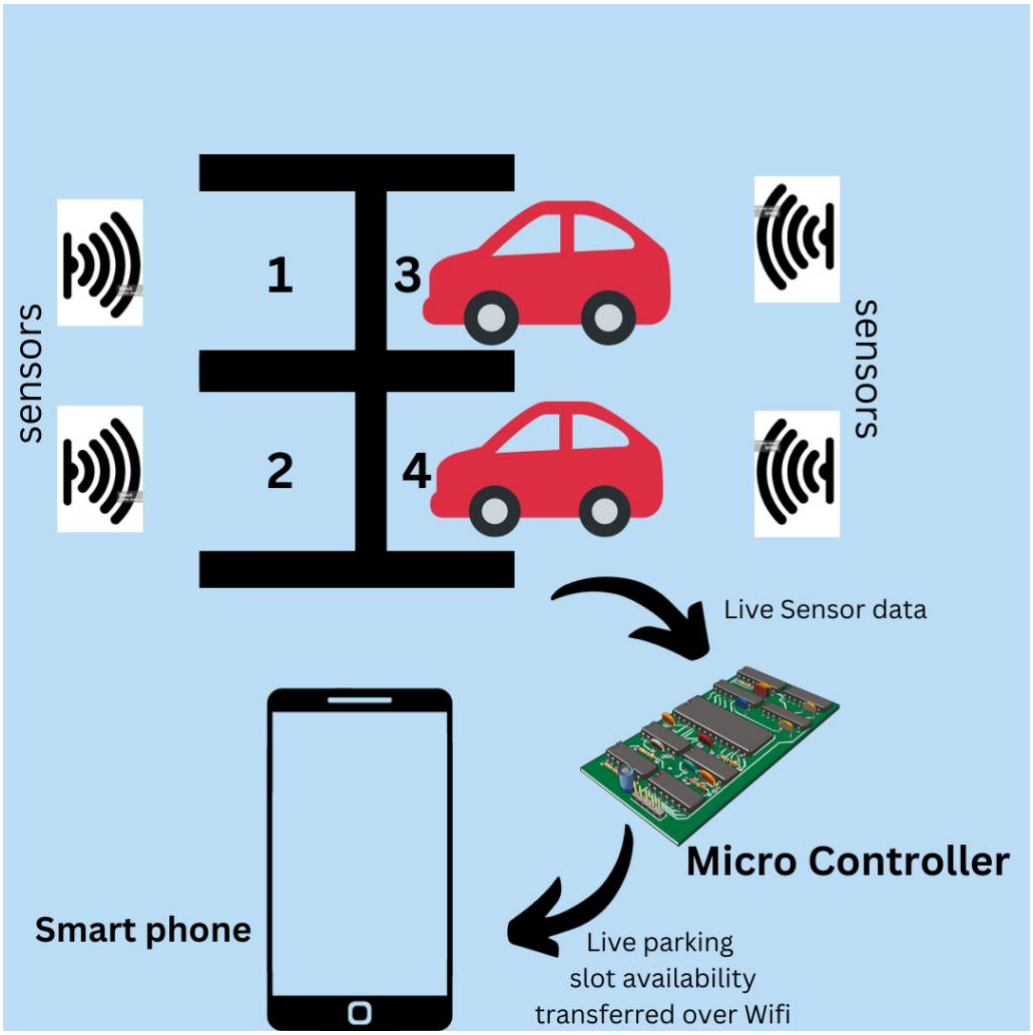
This data, including parking space, distance measurements is carefully packaged and transmitted to the cloud platform(Blynk)through an IoT protocol. The cloud platform, equipped with IoT services, efficiently receives, and ingests the incoming data, storing it securely.

Here, cloud-based (Blynk) data processing and analytics services come into play, extracting valuable insights, such as real-time parking space occupancy and trends.

Users are granted easy access to this information through mobile applications, facilitating informed parking decisions.

Moreover, the cloud platform can trigger alerts and notifications to users or management staff based on specific conditions, thereby optimizing parking operations and enhancing the overall parking experience. This data flow mechanism is the backbone of a smart parking system, contributing to improved urban parking management and smarter city living.

BLOCK DIAGRAM:

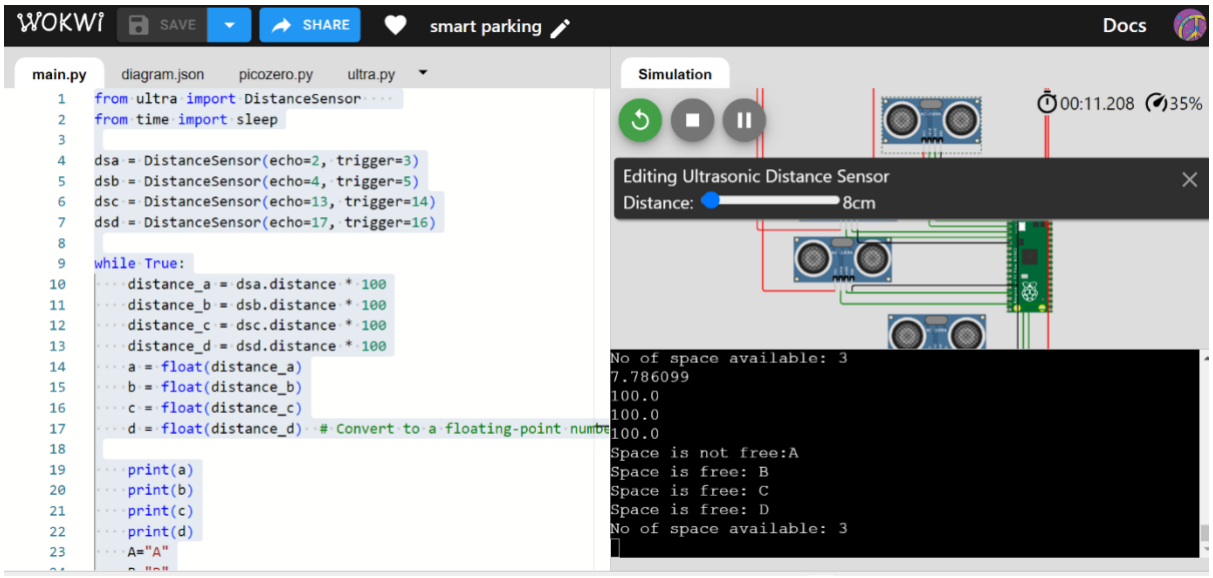


9.CODE:

Using Micro python & Raspberry pi Pico

```
 #You can simulate this code using Wokwi
from ultra import DistanceSensor
from time import sleep

dsa = DistanceSensor(echo=2, trigger=3)
dsb = DistanceSensor(echo=4, trigger=5)
dsc = DistanceSensor(echo=13, trigger=14)
dsd = DistanceSensor(echo=17, trigger=16)

while True:
    distance_a = dsa.distance * 100
    distance_b = dsb.distance * 100
    distance_c = dsc.distance * 100
    distance_d = dsd.distance * 100
    a = float(distance_a)
    b = float(distance_b)
    c = float(distance_c)
    d = float(distance_d)  # Convert to a floating-point number

    print(a)
    print(b)
    print(c)
    print(d)
    A="A"
    B="B"
    C="C"
    D="D"
    no=0

    def parking(distance, n,slot):
        if distance < 30:
        # Code to execute if the distance is less than 30
            print("Space is not free:"+slot)
            if(n==0):
                n=0
            else:
                n=n-1
        else:
        # Code to execute if the distance is not less than 30
            print("Space is free: "+slot)
```

```
            if(n==4):
                n=4
            else:
                n=n+1
        return n


    no=parking(a,no,A)
    no=parking(b,no,B)
    no=parking(c,no,C)
    no=parking(d,no,D)
    no=no
    print("No of space available:",no)



    sleep(0.1)
```



Output of parking slot availability using micro python on Raspberry pi Pico.

Link –
https://wokwi.com/projects/380204068912496641

NOTE:
Due to unavailability of libraries for micro python on Raspberry Pi Pico we cannot able to transfer the data from sensor to cloud platform .
Such as urequest, WIFI , think speak libraries

So, we used ESP32 & C/C++ programming language to work our Final project on real time data transfer.

Using C/C++ Code & BLYNK

```
#define BLYNK_TEMPLATE_ID "TMPL3Fa6eQf9e"
#define BLYNK_TEMPLATE_NAME "Smart parking"
#define BLYNK_AUTH_TOKEN "C5yd-C-cAOwEMMiqohaSeoJRModhyTzl"

//Including Library files
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

//Defining wifi credentials
char ssid[] = "Wokwi-GUEST";
char pass[] = "";


BlynkTimer timer;

// This function is called every time the Virtual Pin 0 state changes
BLYNK_WRITE(V0)
{
  // Set incoming value from pin V0 to a variable
  int value = param.asInt();

  // Update state
  Blynk.virtualWrite(V1, value);
}

// This function is called every time the device is connected to the
Blynk.Cloud
BLYNK_CONNECTED()
{
  // Change Web Link Button message to "Congratulations!"
  Blynk.setProperty(V3, "offImageUrl", "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations.png");
  Blynk.setProperty(V3, "onImageUrl",  "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations_pressed.png"
);
  Blynk.setProperty(V3, "url", "https://docs.blynk.io/en/getting-started/what-
do-i-need-to-blynk/how-quickstart-device-was-made");
}

// This function sends Arduino's uptime every second to Virtual Pin 2.
void myTimerEvent()
{
  // You can send any value at any time.
  // Please don't send more that 10 values per second.
  Blynk.virtualWrite(V2, millis() / 1000);
```

```
}

//Defining Pin numbers
#define ECHO_A 2
#define TRIG_A 18
#define ECHO_B 4
#define TRIG_B 5
#define ECHO_C 13
#define TRIG_C 14
#define ECHO_D 17
#define TRIG_D 16

int a,b,c,d;
int Slot;

//char full[] = "Full";
//char fr[] = "Free";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
//Defining Pin modes
  pinMode(TRIG_A, OUTPUT);
  pinMode(ECHO_A, INPUT);
  pinMode(TRIG_B, OUTPUT);
  pinMode(ECHO_B, INPUT);
  pinMode(TRIG_C, OUTPUT);
  pinMode(ECHO_C, INPUT);
  pinMode(TRIG_D, OUTPUT);
  pinMode(ECHO_D, INPUT);

  Serial.begin(115200);
  Serial.println("Hello, ESP32!");
}

void loop() {
  // put your main code here, to run repeatedly:

  a=check(TRIG_A,ECHO_A);
  if(a==0){
     Blynk.virtualWrite(V1, 0 );
  }
  else{
     Blynk.virtualWrite(V1, 1);
  }
```

```
  b=check(TRIG_B,ECHO_B);
  if(b==0){
     Blynk.virtualWrite(V2, 0 );
  }
  else{
     Blynk.virtualWrite(V2, 1);
  }

  c=check(TRIG_C,ECHO_C);
  if(c==0){
     Blynk.virtualWrite(V3, 0 );
  }
  else{
     Blynk.virtualWrite(V3, 1);
  }

  d=check(TRIG_D,ECHO_D);
  if(d==0){
     Blynk.virtualWrite(V4, 0 );
  }
  else{
     Blynk.virtualWrite(V4, 1);
  }

  Serial.printf("%d %d %d %d \n",a,b,c,d);
  Slot=a+b+c+d;
  Serial.printf("slots free=%d \n",Slot);
  Blynk.virtualWrite(V0, Slot);
  delay(2000);
  // this speeds up the simulation
}

//function to chech distance
float check(int trigerpin, int echopin){

    float distance, duration;
    digitalWrite(trigerpin, LOW);
    delay(2);
    digitalWrite(trigerpin, HIGH);
    delay(10);
    digitalWrite(trigerpin, LOW);
    duration=pulseIn(echopin,HIGH);
    distance=duration/2*0.034;
    Serial.print("distance (in cm)=");
    Serial.println(distance);
    delay(10);
    if(distance>30){
      return 1;
```

```
    }
    else{
      return 0;
    }
 }
```

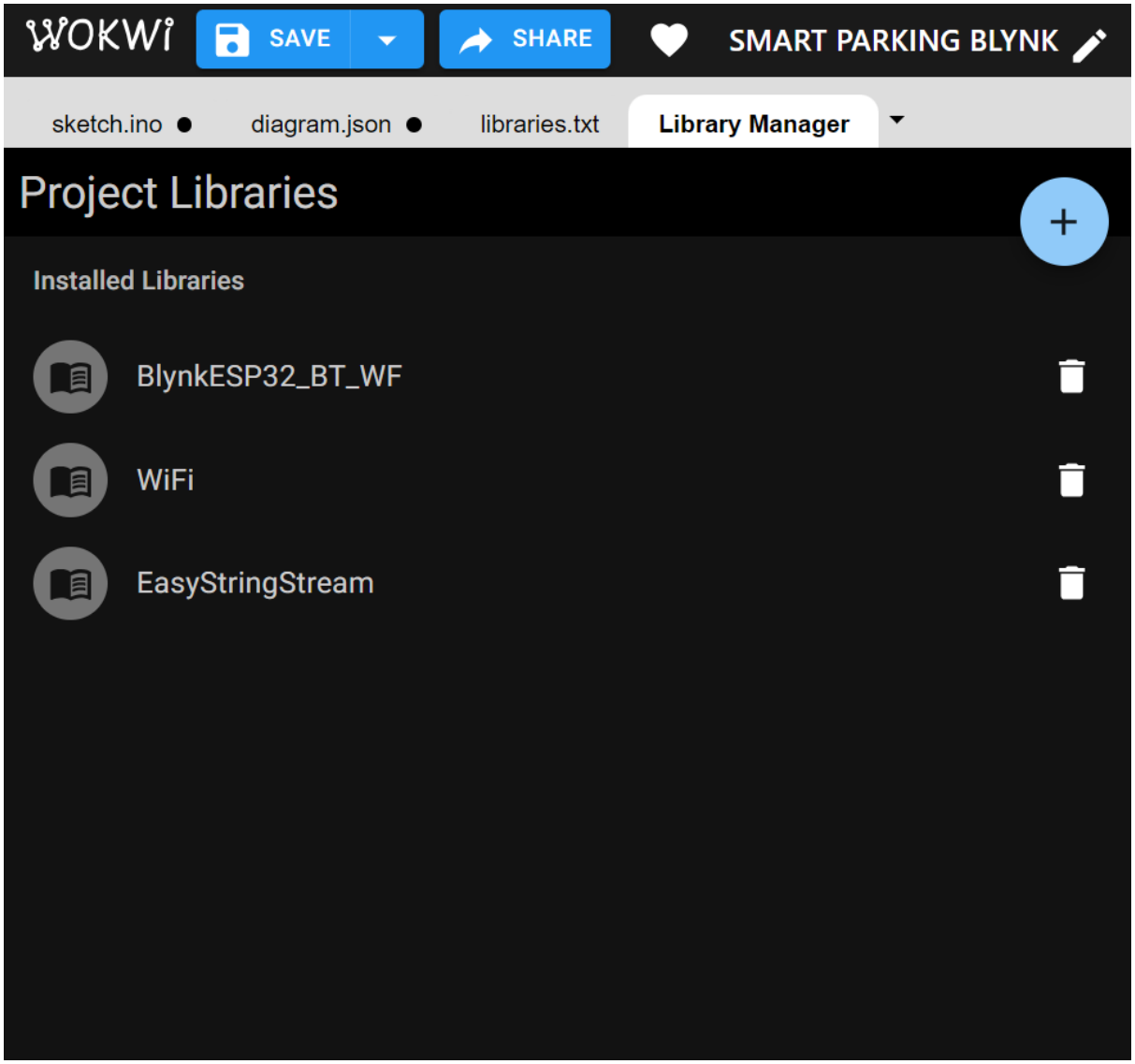
You can run our code in wokwi by this link
Link-

https://wokwi.com/projects/380204745442260993

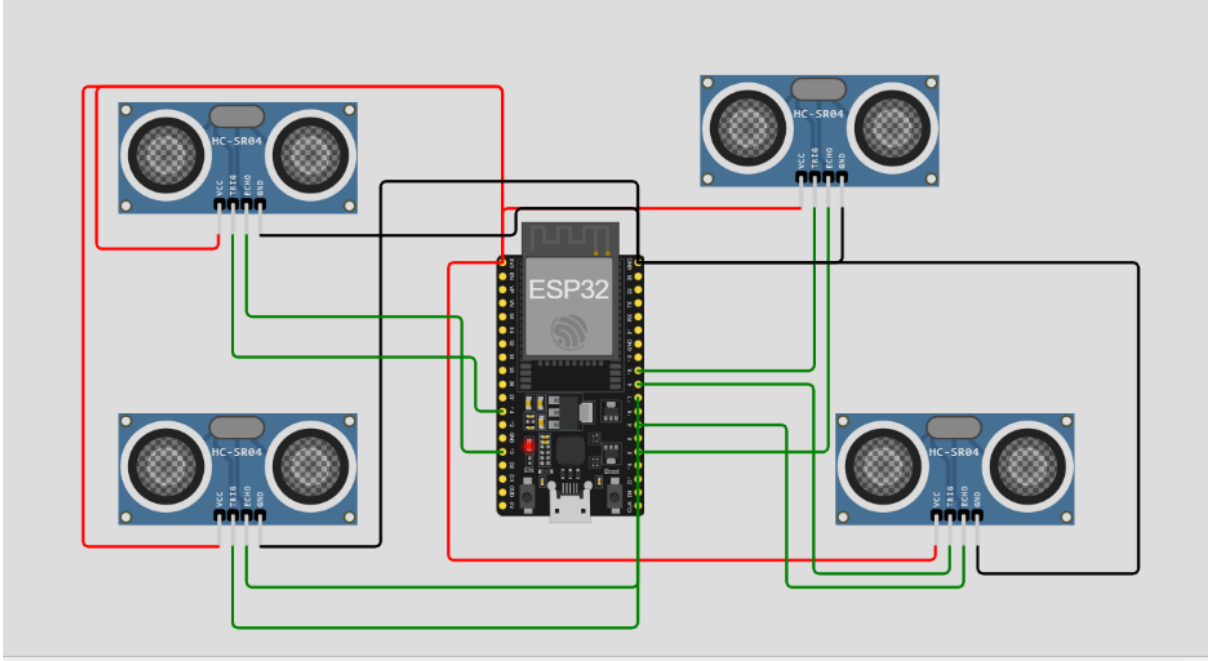
10.Project Development Steps:

Step-1: Choosing Suitable components
We Chose ESP32 & Ultrasonic (HC-SR04) Sensors.

Step-2: Searching and adding suitable library files



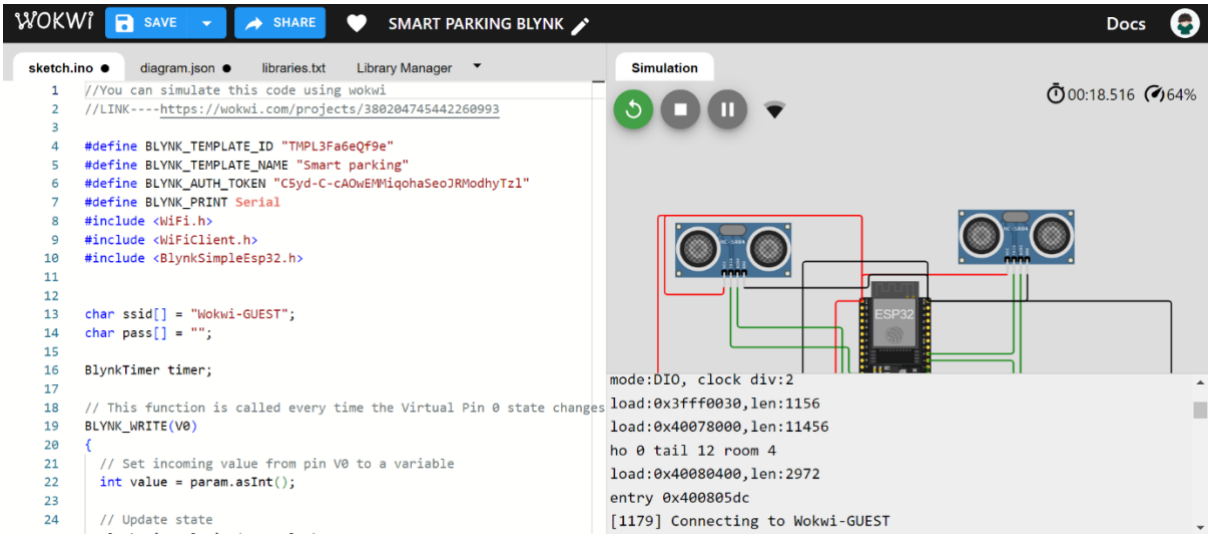
Step-3: Connecting the Components.



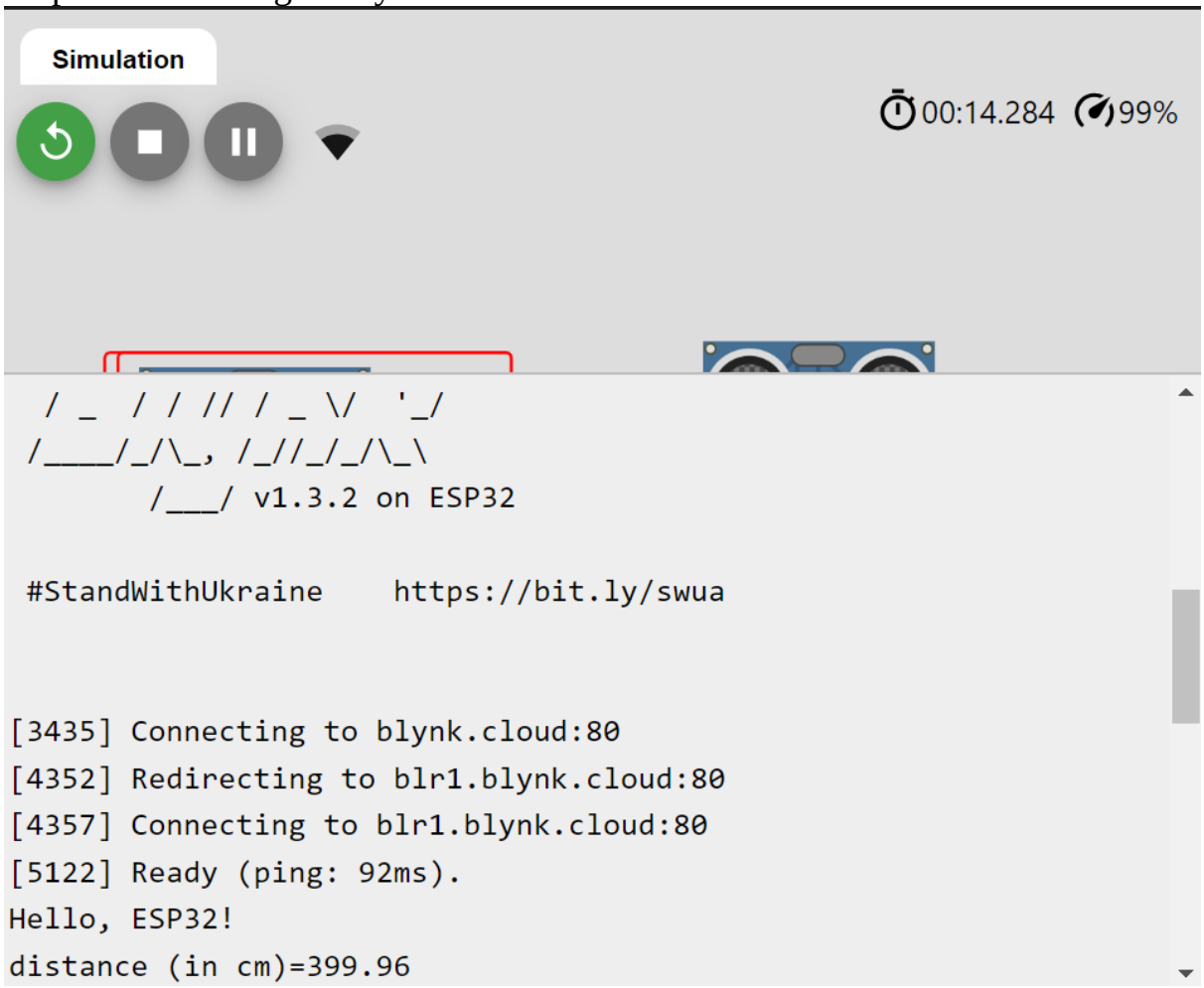
Step-4: Write Code based on the project

We have Shown the Code above.

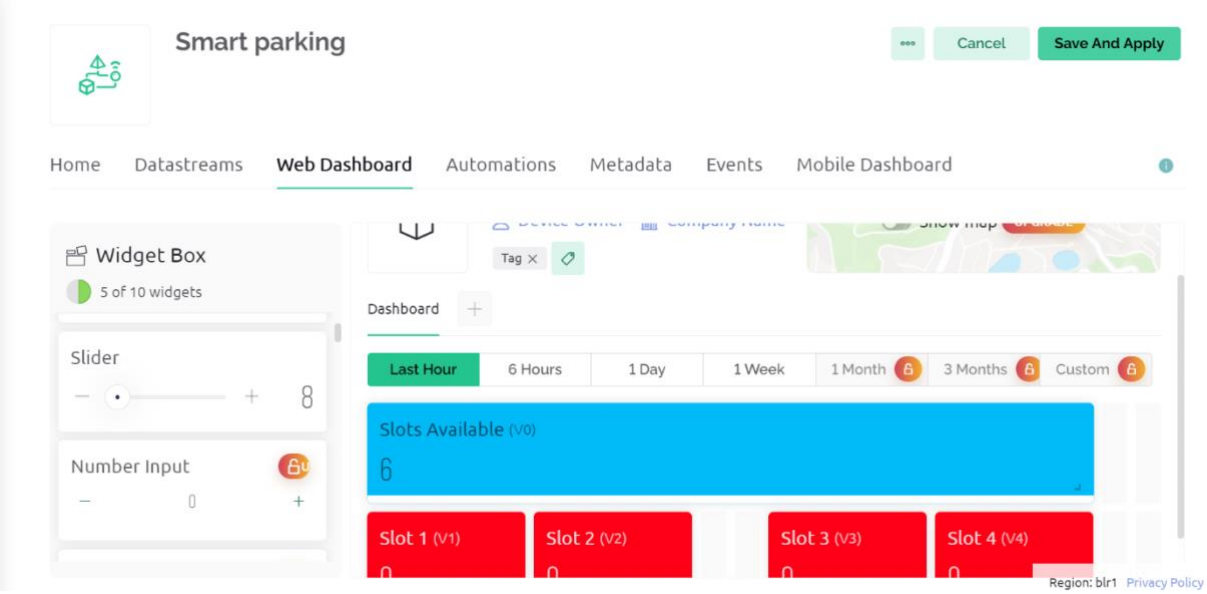
Step-5: Connecting to Wi-Fi.



Step-6: Connecting to Blynk.

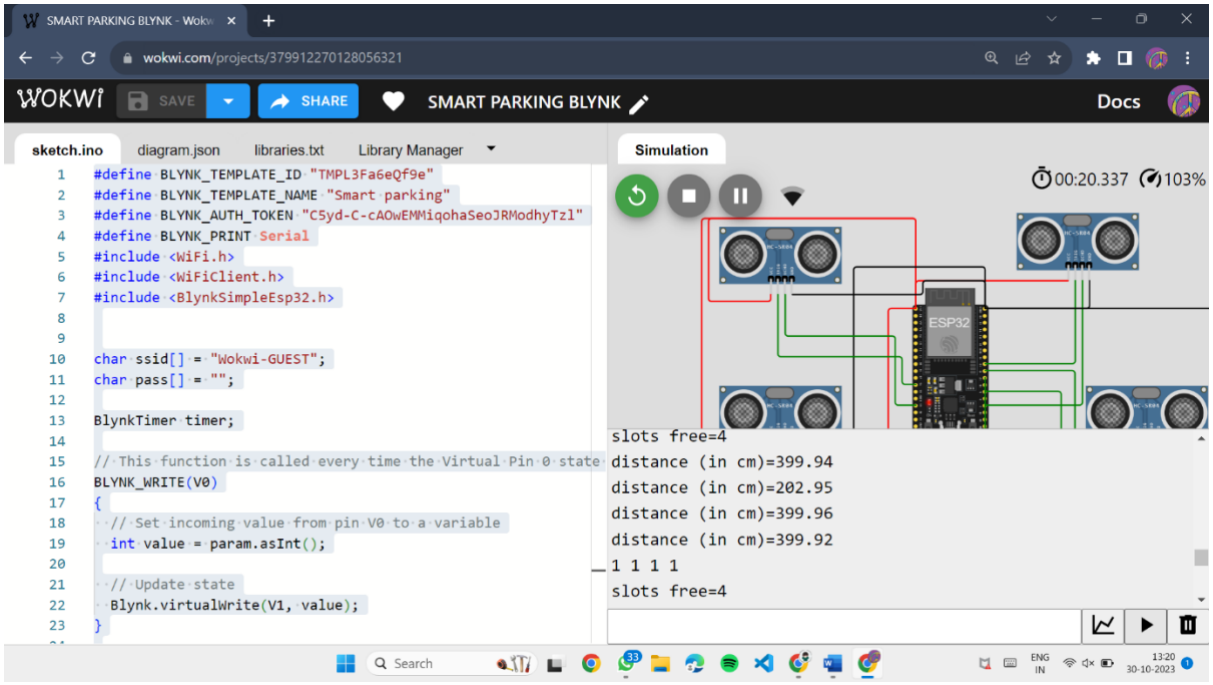


Step-7: Setting up Blynk Dashboard

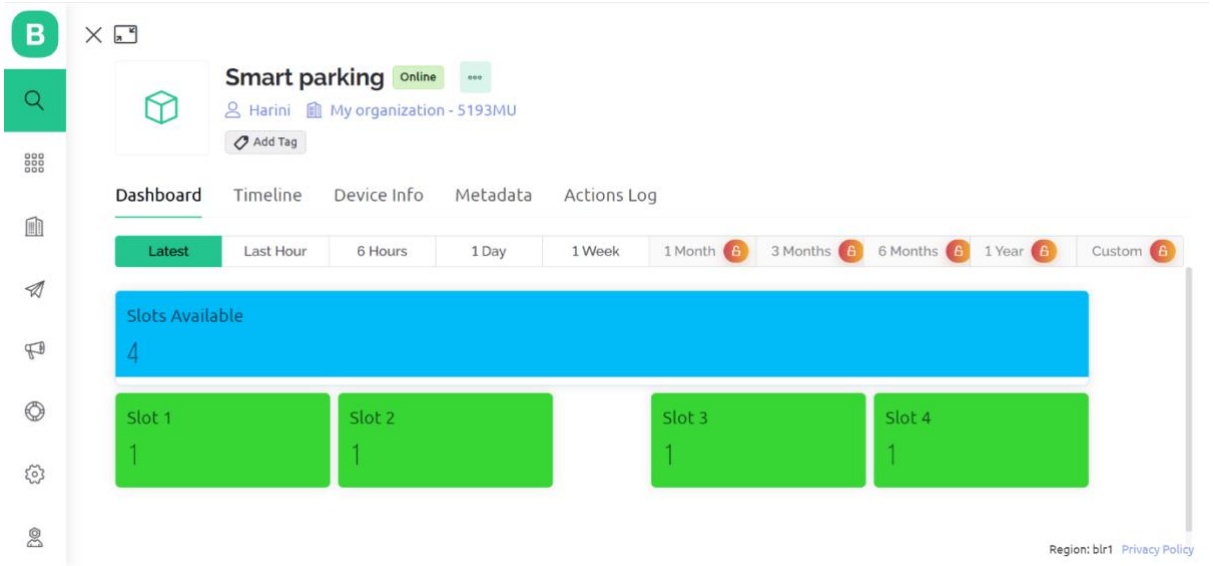


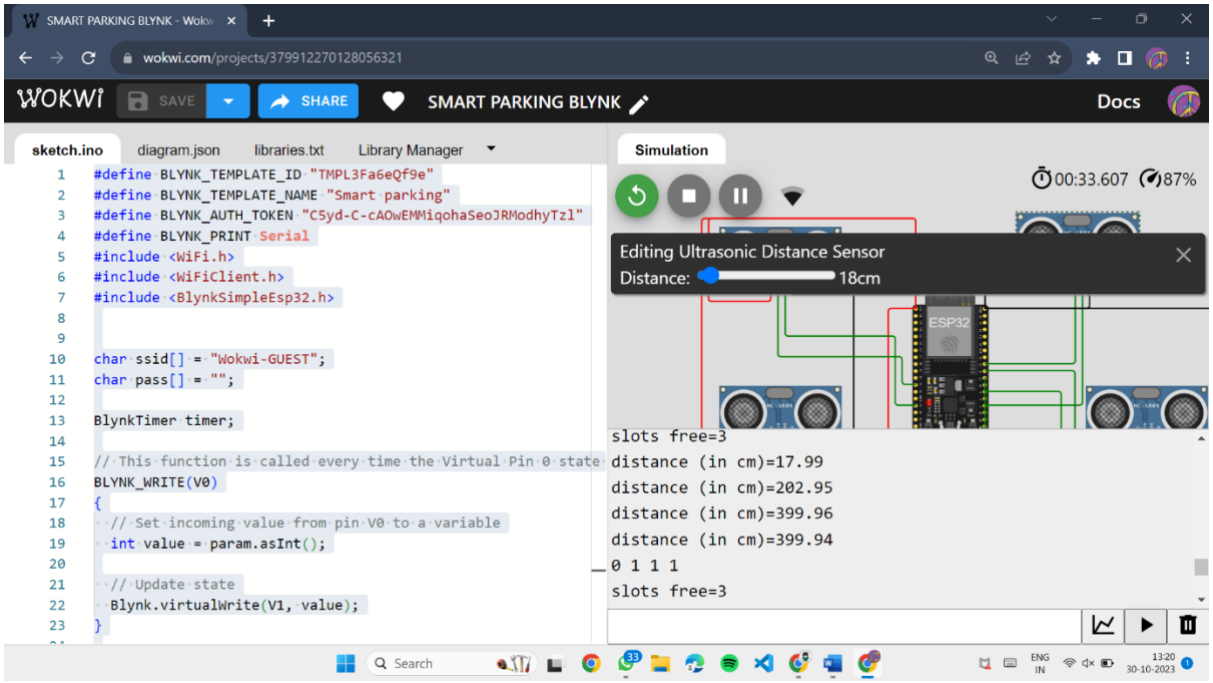
Step-8: Connection established.

-Live Status is now viewed in Blynk app.

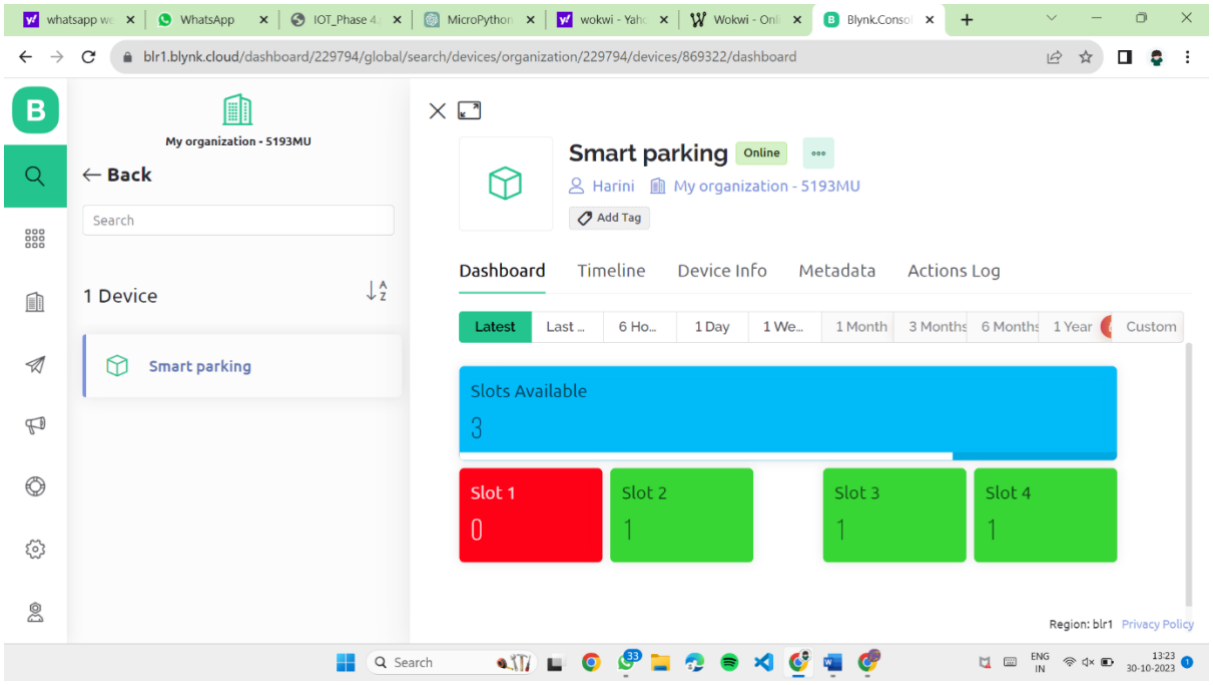


Blynk output :



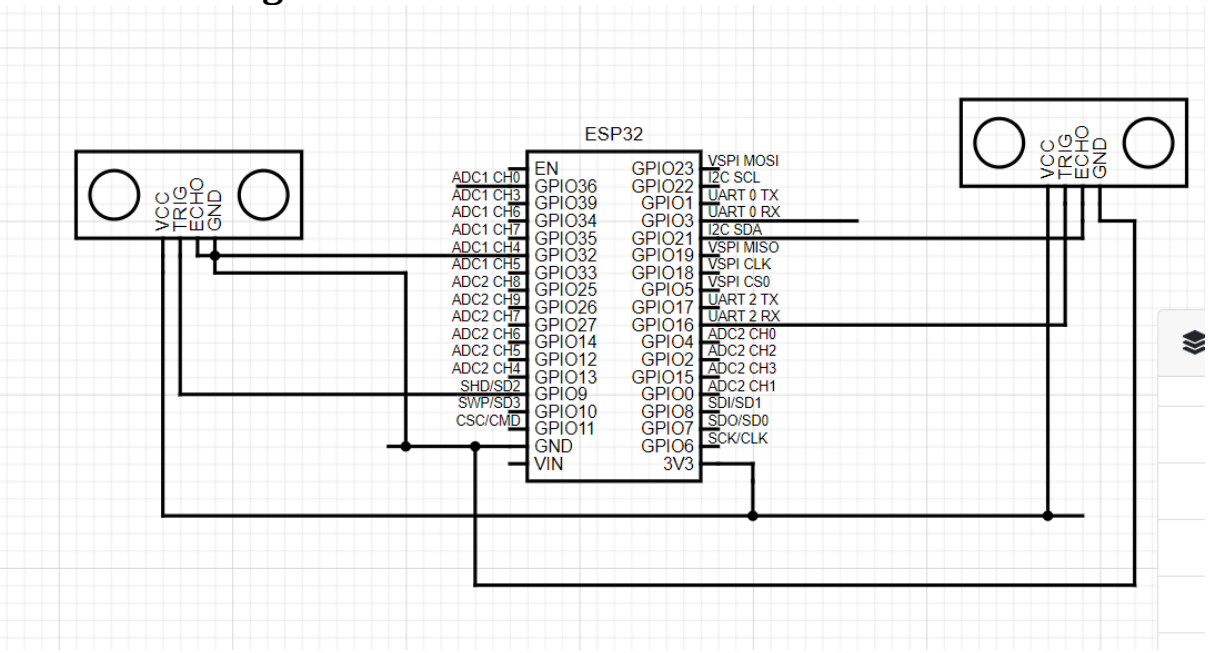


Here, 3 slots where free, also it mentions the distance in the parking slot in centimetre. By connecting it into Blynk using WIFI shows real time parking space availability.



In Blynk number of slots available is shown. The free parking slots were indicated in green label and occupied slots were indicated in red label.

Schematic Diagram:



11.BENIFITS

Improved User Experience: Drivers can quickly find available parking slots, reducing frustration and time wasted circling for parking.

Optimized Resource Utilization: Parking operators can better manage their parking facility by allocating resources efficiently based on real-time demand.

Reduced Congestion: Smart parking guidance systems can help reduce traffic congestion within parking facilities and in the surrounding areas

12.CONCLUSION

This represents a Transformative solution to urban parking inefficiencies. By integrating ESP32, sensors, and parking space availability and enhances user experiences. Real-time parking availability information, harmony to reduce congestion and pollution, making urban living more sustainable. The project's comprehensive documentation ensures long-term effectiveness, and it paves the way for smarter, more efficient urban mobility.