Smart Campus Parking System

An approach to solve: Limited parking space and inefficient parking management on campus.

Team Members :

Naman Mulani - 21CSB0B38

Harshit Gupta - 21CSB0B21

Upre Ashish Kumar - 21CSB0B62

Supervised by :

Prof. Shiva Darshan S L

Github Link :
https://github.com/nitin814/Smart-Campus-Parking-System-IOT

Contributions :

Naman Mulani - Documentation, PPT, Thingspeak-Wokwi integration, Firebase-Wokwi Integration, Circuit Code Implementation, 3d Design, Thingspeak-Matlab code for Data Visualisation

Harshit Gupta - Documentation, PPT, User Dashboard Creation (Website + App), Firebase-Dashboard Integration, Alert Creation in Dashboard

Upre Ashish Kumar - Documentation, PPT, Circuit Design, Circuit Code Implementation, TinkerCad implementation, Wokwi Implementation, LEDs/LCD control

Internet of Things [CS-451]

Course Assignment

Department of Computer Science and Engineering

National Institute of Technology Warangal

Telangana, India – 506004

April 2025

Table of Contents

References

1 Project Overview

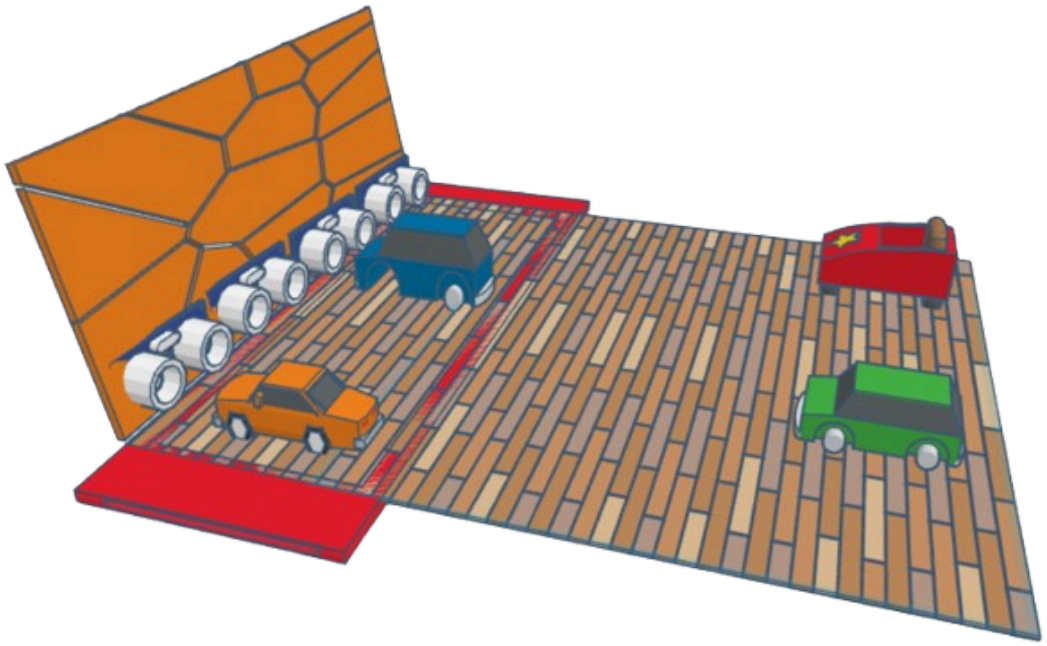
1.1 Background

Parking congestion is a significant challenge in many educational institutions, leading to **wasted time, increased fuel consumption, and frustration** among students, faculty, and visitors. Traditional parking systems often lack real-time monitoring, forcing drivers to search for available spots manually. This inefficiency contributes to **traffic congestion, unnecessary emissions, and overall inconvenience** on campus.

With the advancement of **Internet of Things (IoT) technology**, smart parking solutions have emerged as an effective way to **optimize parking space utilization**. By integrating **ultrasonic sensors, microcontrollers, and real-time data processing**, campuses can **automate parking management**, providing users with **instant availability updates** while reducing congestion and improving overall campus mobility.

1.2 Problem statement

Finding a parking spot in a busy campus can be a time-consuming and frustrating process. Without a system that provides real-time availability updates, drivers often struggle to locate free spaces, leading to delays and inefficient use of parking resources.



3D Model of Parking lot availability using ultrasonic sensor, created using Tinkercad.

1.3 Objectives and scope

The project aims to:

- Build a Smart Campus Parking System to detect if a parking space is free or not.
- This information then is to be sent to a central system.
- Make use of ultrasonic sensors for the detection of occupancy information of a parking spot.
- Improve the overall parking experience on campus by making it easy to look for a parking spot.

1.4 Solution explanation

The **Smart Campus Parking System** leverages **ultrasonic sensors** to detect the occupancy status of parking spaces. These sensors continuously monitor the presence of vehicles and transmit the collected data to an **ESP32/Arduino Uno**, which processes the information and controls **LED indicators** to provide instant visual feedback on parking availability.

To enhance accessibility and analytics, the system is seamlessly integrated with **ThingSpeak**, a cloud-based IoT platform, which enables real-time data logging and visualization. The recorded data is further processed in **MATLAB** to generate insightful visualizations, such as **heatmaps, predictive trend graphs, and occupancy rate indicators**, aiding in efficient parking space management.

For user convenience, the system also incorporates **Firebase**, which powers a **web and mobile-based dashboard** that allows users to remotely check parking availability and receive **alerts when the parking area reaches full capacity or if there is any spot available**. This ensures a **smart, automated, and efficient** parking experience while reducing congestion and optimizing space utilization.

2 Design and Implementation

2.1 System Design

The **Smart Campus Parking System** was designed and simulated using **Tinkercad\Wokwi**, an online circuit simulation tool. The system is composed of multiple hardware and software components working together to ensure efficient parking management.

2.1.1. Hardware Components & IoT and Cloud Integration:

- **Ultrasonic Sensors (HC-SR04)** – Detects vehicle presence by measuring the distance between the sensor and the nearest object.

- **Arduino Uno/ESP32** – Acts as the central controller, processing sensor data and managing real-time communication with cloud platforms.

- **LED Indicators** – Provide instant visual feedback on parking slot availability (e.g., **green** for vacant, **red** for occupied).
- **ThingSpeak** – A cloud-based platform used for real-time data logging and visualization of parking slot status.

- **Firebase** – Enables a web and mobile-based dashboard where users can remotely check parking availability and receive alerts when the lot is full.

- **MATLAB** – Integrated with ThingSpeak to perform advanced analytics, such as heatmaps and predictive trend analysis for parking demand.

By leveraging Wokwi for simulation and integrating **IoT technologies**, the system provides an **efficient, automated, and real-time** parking solution for smart campuses

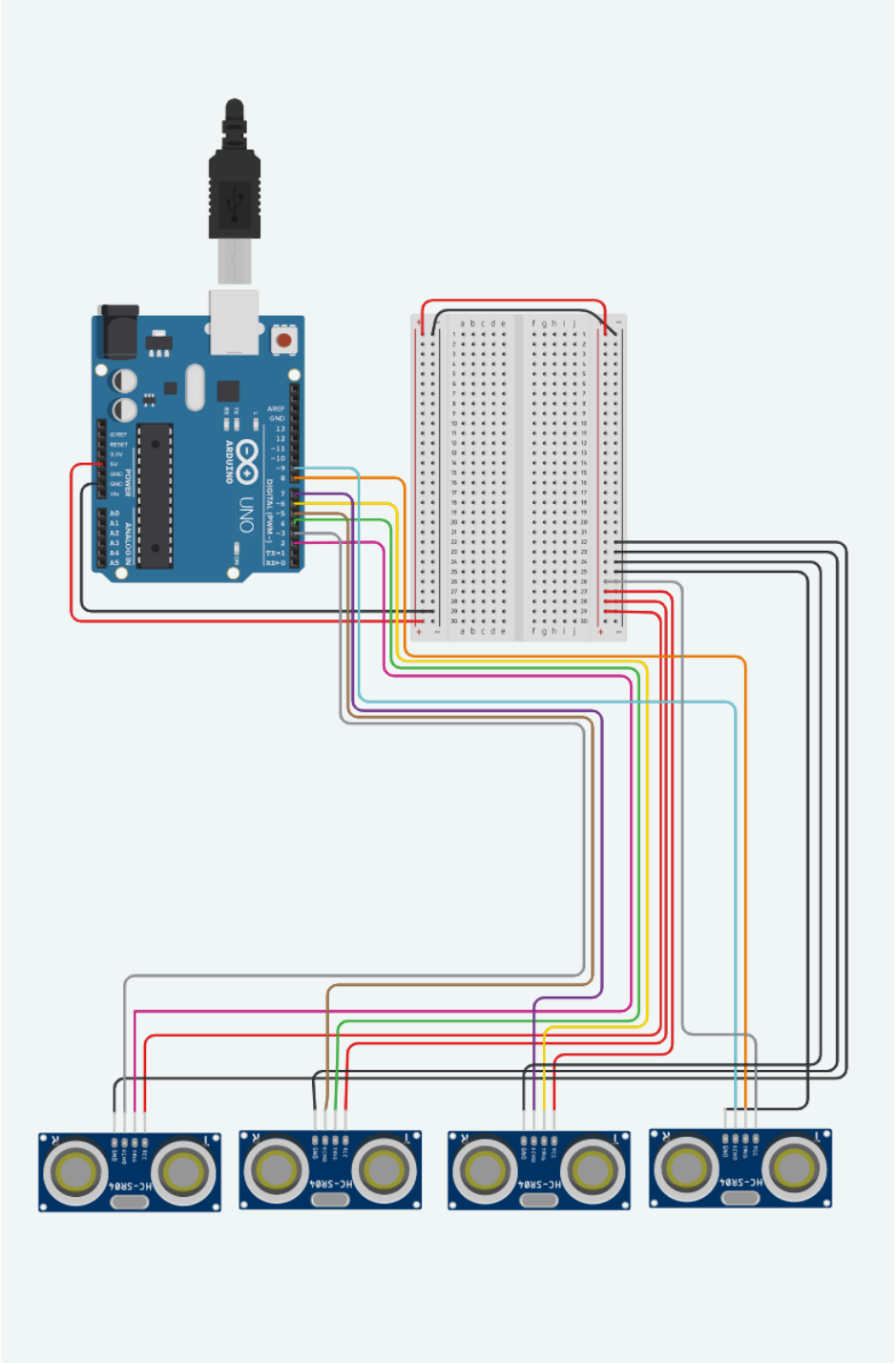
2.1.2 Components Used in Tinkercad Simulation & Simulation Details

1. **Arduino Uno R3** as the Microcontroller for sensor data processing.

2. **HC-SR04 Ultrasonic Sensors (x4)** which are for vehicle detention in parking sptos.

3. **Breadboard & Jumper Wires** for connecting components.

One of the choices for microcontrollers for this project is Arduino Uno R3. It is a microcontroller board based on the **ATmega328P**. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller.
The ATmega328P can easily be replaced, as it is not soldered to the board. The ATmega328P also features 1kb of EEPROM, a memory which is not erased when powered off. Arduino UNO features a barrel plug connector that works great with a standard 9V battery.



TinkerCAD layout using Arduino Uno R3

Some details of it are as follows :

Pins :

Built-in LED Pin 13

Digital I/O Pins	14
Analog input pins	6
PWM pins	6

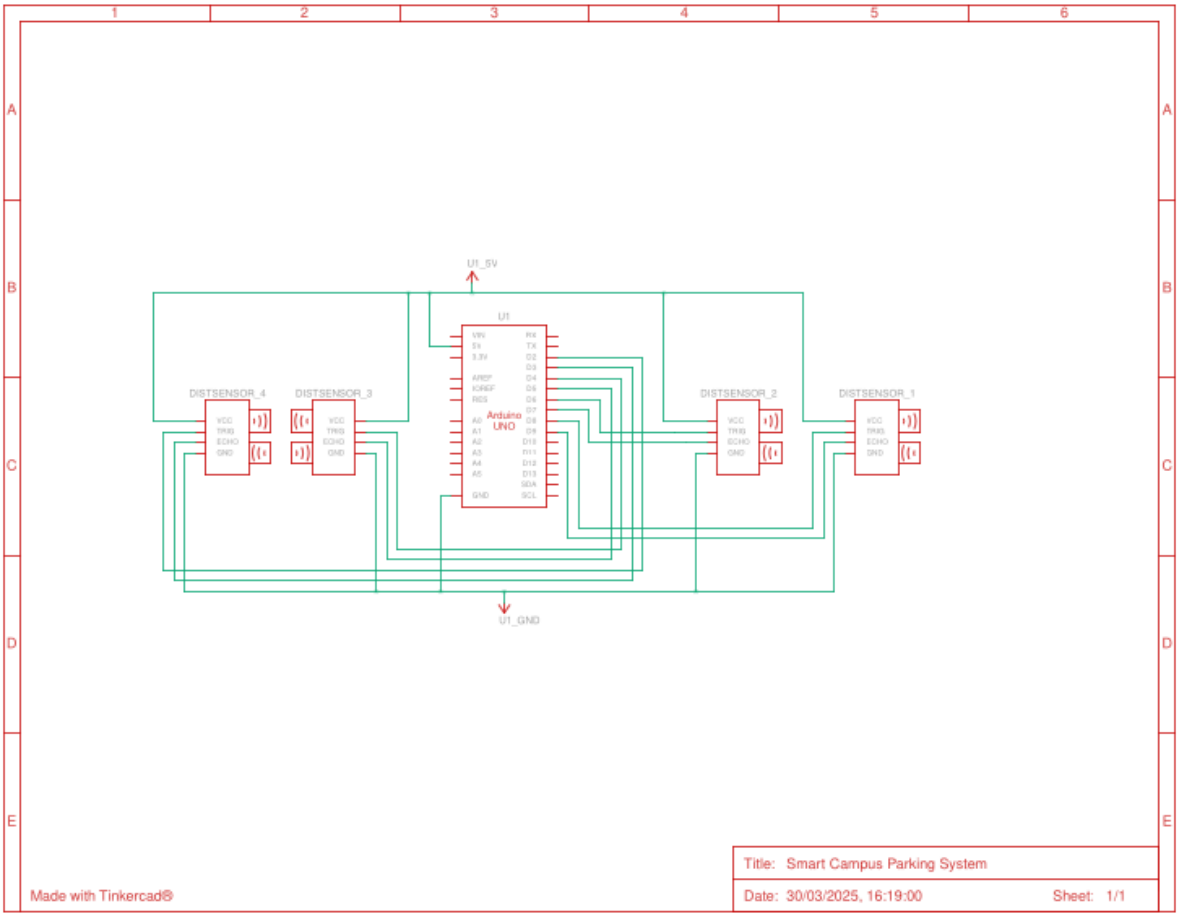
Clock speed :	
Main Processor	ATmega328P 16 MHz
USB-Serial Processor	ATmega16U2 16 MHz

Memory :	
ATmega328P	2KB SRAM, 32KB FLASH, 1KB EEPROM

We have used 4 **HC-SR04 Ultrasonic Sensors** as the sensor of choice for the implementation. The idea is that these sensors will be deployed in each of the parking spots for the purpose of getting information if the parking space is occupied or not. These sensors are chosen for this project. The characteristics of that sensor are:

- 400cm of non contact measurement.
- 5v operation voltage.
- Working Current: 15 mA
- Distance Range: 2cm to 400cm
- Stock code - HC-SR04
- 100% Arduino Compatible

The circuit diagram of the project is as follows:



Circuit diagram

Now this circuit is enough for standalone parking systems. Like those that don't need an internet connection. This system will work perfectly fine as long as no internet connectivity is required and just offline parking system work has to be done and all connections are hardwired. This is because Arduino does not have any wifi module built into it. **But for our project, we decided to take it up a notch with internet connectivity so that it can communicate with Thingspeak, hence we made use of ESP 32 instead of Arduino. The system was designed in Wokwi, where each parking slot was assigned an ultrasonic sensor. The microcontroller reads the data from these sensors and determines which of the slots are occupied or vacant. The data is processed and transmitted to cloud services such as Thingspeak integrated with Matlab for data visualization and analysis, along with Firebase to show a dashboard.**

ESP 32

ESP32 is a family of low-cost, energy-efficient microcontrollers that integrate both wifi and bluetooth functionality. We have made use of ESP32-DevKitC V4 for our project. ESP32-DevKitC V4 is a small-sized ESP32-based development board produced by Espressif. I/O pins are on both sides of this board. The pro characteristics of this are:

Component	Description
ESP32-WROOM-32	A module with ESP32 at its core.

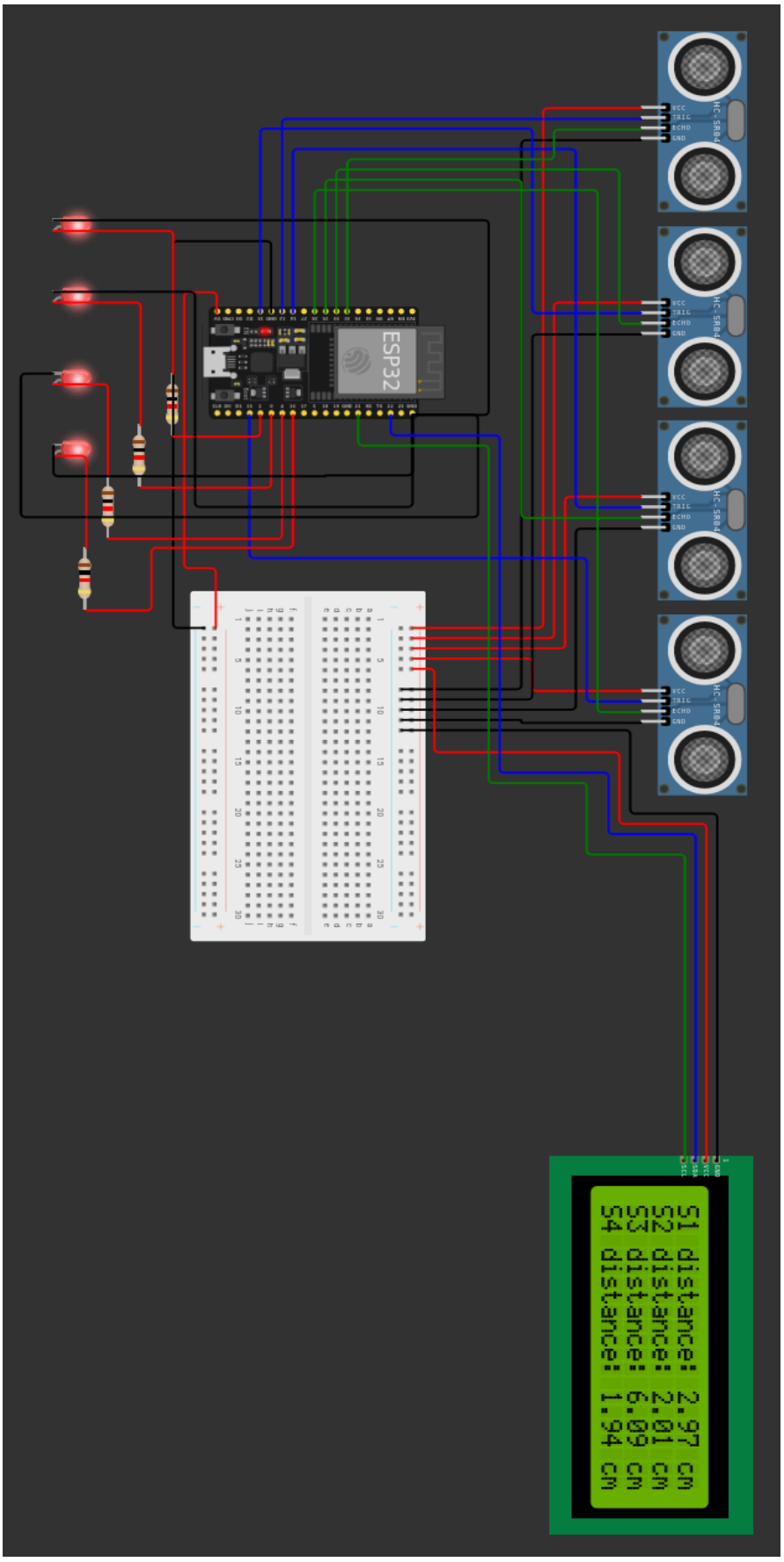
EN	Reset button.
Boot	Download button. Holding down Boot and then pressing EN initiates Firmware Download mode for downloading firmware through the serial port.
USB-to-UART Bridge	Single USB-UART bridge chip provides transfer rates of up to 3 Mbps.
Micro USB Port	USB interface. Power supply for the board as well as the communication interface between a computer and the ESP32 module.
5V Power On LED	Turns on when the USB or an external 5V power supply is connected to the board. For details, see the schematics in Related Documents.
I/O	Most of the pins on the ESP module are broken out to the pin headers on the board. You can program ESP32 to enable multiple functions such as PWM, ADC, DAC, I2C, I2S, SPI, etc.

Along with ESP 32, we also made use of four different LEDs that correspond to each of the sensors. Those light up when the detected distance is less than 30cm. Applications of this logic may include looking at LEDs and getting to know if the space is empty or not. For the working of LEDs to make sure that correct potential is given to the LED, we made use of resistors of resistance 1K Ohm for each LED.

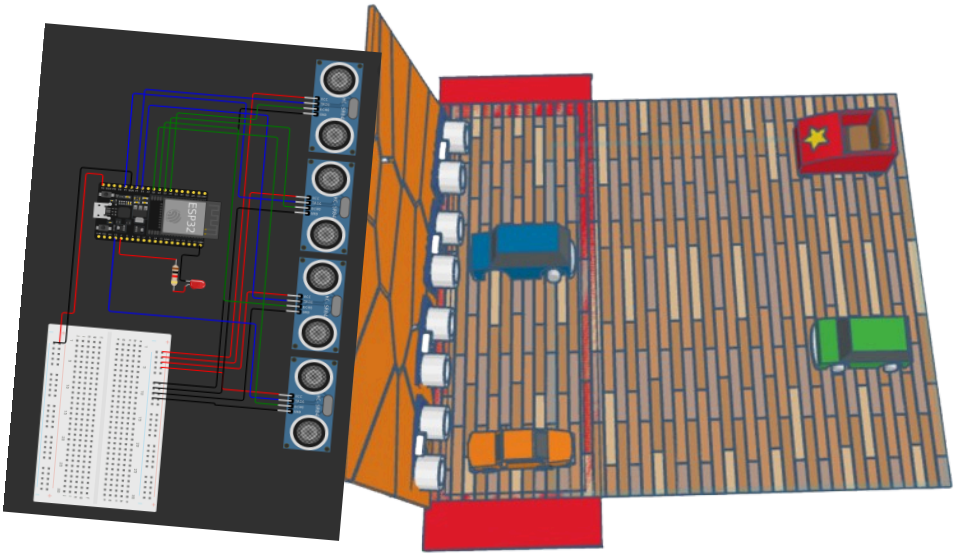
A Liquid Crystal Display (LCD) was integrated into our project to provide a real-time visual display of sensor data. The LCD allows users to view the measured distances from multiple ultrasonic sensors without needing a serial monitor. This is helpful to know the distance related information from each of the sensors. We connected the LCD to the ESP32 using the **I2C interface**:

LCD pin	**ESP32 pin**
VCC	5v thorough breadboard
GND	GND through breadboard
SDA	22
SCL	21

We ensured the correct **I2C address** (0x27 or 0x3F) using an **I2C scanner** before initializing the LCD in the code.


HC-SR04
ESP32
S1 distance: 2.97 cm
S2 distance: 2.01 cm
S3 distance: 6.09 cm
S4 distance: 1.94 cm

Wokwi ESP 32 layout



Overall view of architecture

2.1.3 Simulation in Wokwi

To begin with, the IoT-based parking lot system was first simulated using Wokwi, an online simulation platform for embedded systems. Wokwi allows for the virtual testing of microcontrollers, sensors, and IoT devices before deploying them in a real-world scenario.

Circuit Components Used:

- Microcontroller: ESP8266/ESP32
- Ultrasonic Sensors (HC-SR04): Used to detect the presence of vehicles in each parking slot.
- LED Indicators: Used to indicate parking availability.
- LED Screen:
- Wi-Fi Module (ESP8266): To send real-time data to cloud platforms.

Circuit Diagram

The circuit was designed in Wokwi, where each parking slot was assigned an ultrasonic sensor. The microcontroller continuously reads data from these sensors and determines whether a slot is occupied or vacant. The data is processed and transmitted to cloud services such as Thingspeak integrated with Matlab for data visualization along with firebase for a dashboard to access and view the data.

2.1.4 Programming Logic and Code snippets :

```
#define SENSOR_COUNT 4
```

```
#define I2C_ADDR 0x27

const char* SSID = "Wokwi-GUEST";

const char* PASSWORD = "";

// ThingSpeak API Details

const char* THINGSPEAK_API_KEY = "059WTJOEZQGBGBVU";

const char* THINGSPEAK_URL = "https://api.thingspeak.com/update"

// ESP32 GPIO Mapping

const int trigPins[SENSOR_COUNT] = {12, 13, 14, 15};

const int echoPins[SENSOR_COUNT] = {32, 33, 25, 26};

const int ledPins[SENSOR_COUNT] = {2, 0, 4, 16};

const int displaySDA = 22;

const int displaySCL = 21;

LiquidCrystal_I2C lcd(I2C_ADDR, 20, 4);  // 20x4 LCD
```

- Shown are all the global variables used in the code. As observed, there are all the sensor related info such as their count and the mappings of all the trig & echo pins for each sensor.
- We also have the credentials for Wokwi and ThingSpeak present that will help us with the connections.
- We have mapped all the pins for individual LEDs that are assigned corresponding to each of the sensors, those LEDs will be lit when the distance captured in the sensor is observed to be less than 30cm.
- In the end there are displaySDA(SCL) which are for SDA and SCL pins of our 20x4 LCD display, the global variable for which is initialized as in the next line. The initialization makes use of an address that here is 0x27.

```
void connectWiFi() {

    Serial.print("Connecting to WiFi...");

    WiFi.begin(SSID, PASSWORD);

    while (WiFi.status() != WL_CONNECTED) {

        delay(500);

        Serial.print(".");
```

```
    }

    Serial.println(" Connected!");

}
```

- This function is for connecting to WiFi as its required to send data to the cloud. This is possible only in ESP32 (not in Arduino) as it has a WiFi module built into it.

```
void setup() {

    Serial.begin(115200);

    Serial.println("ESP32 Parking System with ThingSpeak");

    connectWiFi();

    // Configure GPIOs

    for (int i = 0; i < SENSOR_COUNT; i++) {

        pinMode(trigPins[i], OUTPUT);

        pinMode(echoPins[i], INPUT);

        pinMode(ledPins[i], OUTPUT);

        digitalWrite(ledPins[i], LOW);

    }



    Wire.begin(displaySDA, displaySCL);

    lcd.init();

    lcd.backlight();

    lcd.setCursor(0, 0);

    lcd.print("Hello, Wokwi!");

}
```

- This is our setup function
- This does all the initialization such as connecting to WiFi, initialization of all the general purpose IO pins and LCD
- This initialization is crucial for the proper working of the loop() that will follow

```
void loop() {

    String parkingStatus = "";
```

```
    int status[SENSOR_COUNT] = {0};
    float distances[SENSOR_COUNT] = {0};

    for (int i = 0; i < SENSOR_COUNT; i++) {
        // Trigger pulse
        digitalWrite(trigPins[i], LOW);
        delayMicroseconds(2);
        digitalWrite(trigPins[i], HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPins[i], LOW);

        // Measure echo duration
        long duration = pulseIn(echoPins[i], HIGH);
        float distance = (duration * 0.0343) / 2;  // Convert to
distance
        distances[i] = distance;

        Serial.print("Sensor ");
        Serial.print(i + 1);
        Serial.print(" Distance: ");
        Serial.print(distance);
        Serial.println(" cm");

        if (distance > 0 && distance < 30) {
            parkingStatus += "1";
            status[i] = 1;
        } else {
            parkingStatus += "0";
        }
```

```
    }


    LED_LCD_behaviour(status, distances);
    Serial.println("Parking Status: " + parkingStatus);
    sendToThingSpeak(parkingStatus);
    sendToFirebase(parkingStatus);


    delay(15000);  // ThingSpeak free-tier requires ~15 seconds delay
}
```

- Here is the loop function that keeps on running infinitely checking on the sensors.
- The ultrasonic sensor requires a trigger pulse to start measurement.
- The trigPins[i] is set **LOW** for 2µs, then **HIGH** for 10µs, and finally set back to **LOW**. This sends an ultrasonic pulse from the sensor.
- pulseIn(echoPins[i], HIGH) measures the time taken for the ultrasonic wave to reflect back to the sensor, which is later converted into distance based on its value.
- What follows is the display of distances measures in the serial monitor. Next the statuses and distances are updated based on the values.
- We have the function LED_LCD_behaviour() that is for controlling the LCDs and LEDs on the basis of the distances and status observed. The implementation of the function will follow.
- Then the data is sent to ThingSpeak and Firebase through the respective functions implementations of which are discussed in the further sections. Then we require a delay of approx 15s to ensure no rate limit of ThingSpeak API.

```
void LED_LCD_behaviour (int status[], float distances[]) {
    // Blink LED if any sensor detects an object < 30cm away
    lcd.clear();
    for (int i=0; i<SENSOR_COUNT; ++i) {
      if (status[i] & distances[i] <= 30) {
        digitalWrite(ledPins[i], HIGH);
        Serial.println("||| DETECTED |||");
      } else {
```

```
        digitalWrite(ledPins[i], LOW);
      }
      // display info on LCD regardless of distance
      lcd.setCursor(0, i);
      lcd.print("S");
      lcd.print(i + 1);
      lcd.print(" distance: ");
      lcd.print(distances[i]);
      lcd.print(" cm");
    }
}
```

- This function controls the LEDs and LCD.
- Distance measured from the sensors is output onto LCD regardless of the value while the LEDs are lit only if the distance is less than 30cm.
- This is done in iterations for every sensor that is present.

2.2. Integration with ThinkSpeak

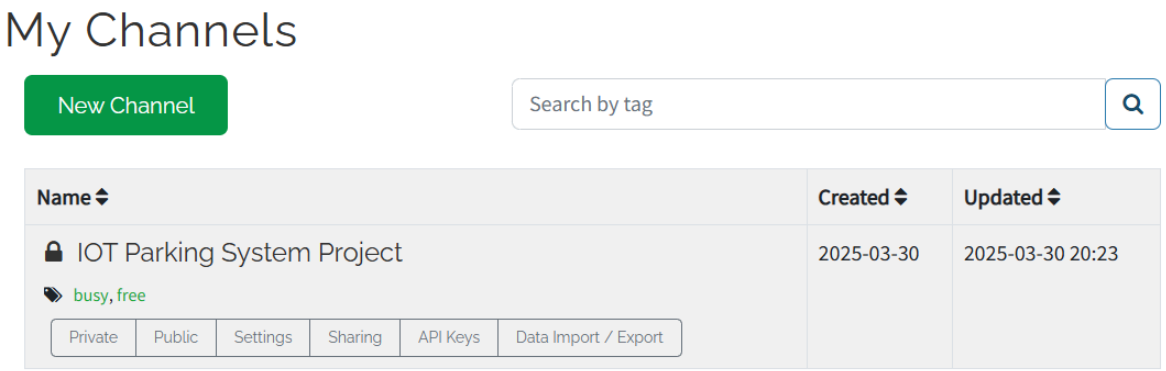
After successfully simulating the IoT system in Wokwi, the next step was integrating it with ThinkSpeak, a cloud-based IoT analytics platform used for real-time data visualization and monitoring.

Why ThinkSpeak?

ThinkSpeak provides an easy-to-use API for sending and retrieving sensor data. It enables real-time monitoring and visualization of the parking lot's status.

Implementation Steps:

1. Create a ThinkSpeak Channel: A new channel was created to store parking slot data.



2. **API Key Configuration: The ESP8266 was programmed to send HTTP GET requests to the ThinkSpeak API.**

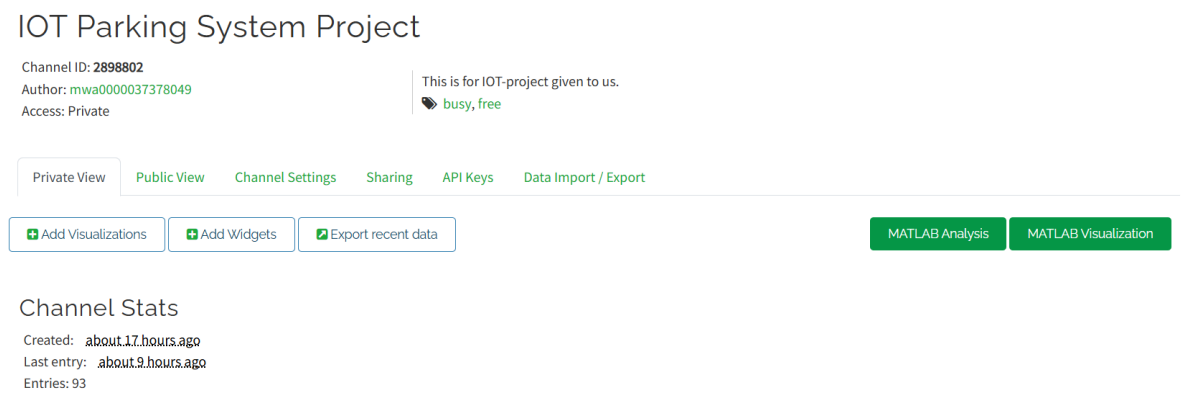
```
// ThingSpeak API Details
const char* THINGSPEAK_API_KEY = "059WTJOEZQGBGBVU";
const char* THINGSPEAK_URL = "https://api.thingspeak.com/update";
```

```
String url = String(THINGSPEAK_URL) + "?api_key=" + THINGSPEAK_API_KEY;

for (int i = 0; i < SENSOR_COUNT; i++) {
    url += "&field" + String(i + 1) + "=" + parkingStatus[i];
}

Serial.println("Sending data to ThingSpeak...");
Serial.println(url);

http.begin(url);
int httpCode = http.GET();

if (httpCode > 0) {
    Serial.println("ThingSpeak Response: " + http.getString());
} else {
    Serial.println("Error sending data: " + String(httpCode));
}
http.end();
```

3. **Data Upload: The status of each parking slot (occupied or vacant) was uploaded at regular intervals.**

```
sendToThingSpeak(parkingStatus);

delay(15000);  // ThingSpeak free-tier requires ~15 seconds delay
```

4. **Real-time Graphs: ThinkSpeak automatically plots the data, enabling remote monitoring of the parking lot status.**



2.3. Integration with Firebase

To provide an interactive dashboard for users, the IoT system was integrated with Firebase, Google's cloud-based database platform. This allows users to check the availability of parking spaces and receive alerts about the parking status along with when it is full.

Firebase Features Used:

- Firebase Realtime Database: Stores the parking lot status in real-time.
- Firebase Authentication: Enables user login functionality.
- Firebase Cloud Messaging: Sends notifications to users when the parking lot is full.

Implementation Steps:

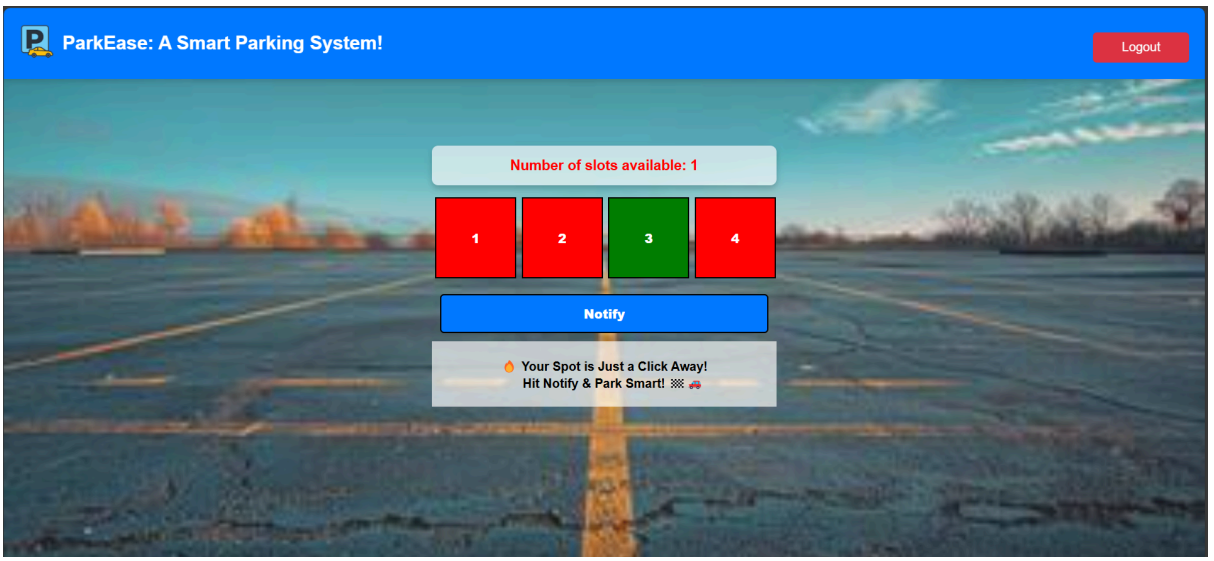
1. ESP8266 Data Push: The ESP8266 updates the Firebase database with parking slot statuses.

```
16  #define FIREBASE_HOST "https://iot-project-d423a-default-rtdb.firebaseio.com/"
17  #define FIREBASE_AUTH "AIzaSyBzKkjdFc8iIL7acumDhwZ20qNLwjAl4U8"
18
19  FirebaseData fbdo;
20  FirebaseAuth auth;
21  FirebaseConfig config;
```

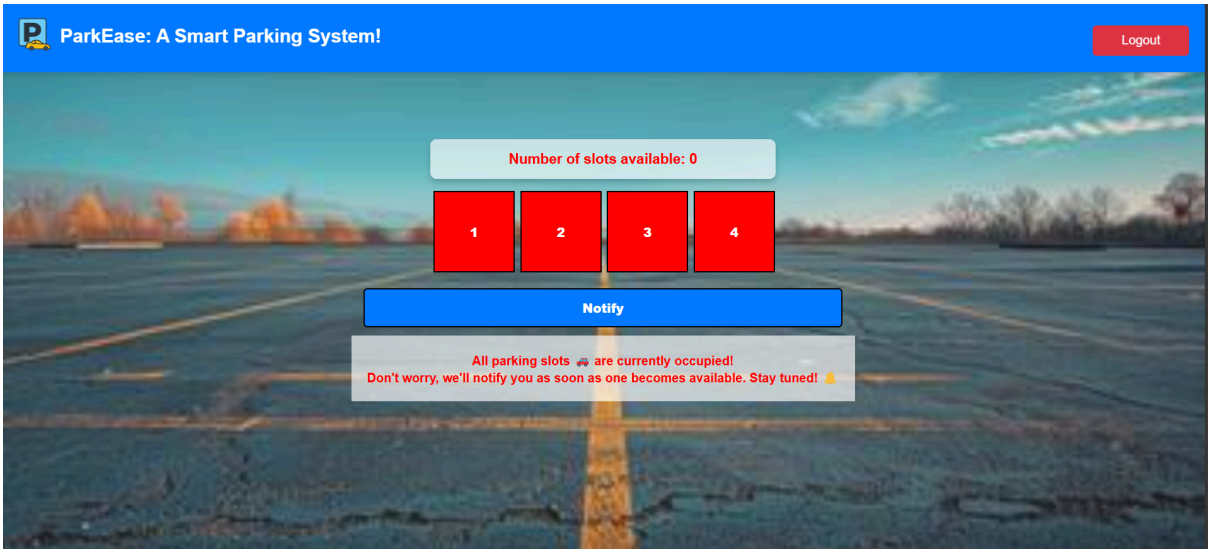
```
71  void sendToFirebase(int parkingStatus[]) {
72      if (Firebase.ready()) {
73          StaticJsonDocument<200> jsonDoc;
74          JsonArray spots = jsonDoc.createNestedArray("parkingSpots");
75
76          for (int i = 0; i < SENSOR_COUNT; i++) {
77              JsonObject obj = spots.createNestedObject();
78              obj["id"] = i + 1;
79              obj["status"] = parkingStatus[i];
80          }
81
82          String jsonString;
83          serializeJson(jsonDoc, jsonString);
84
85          String path = "/parkingSpots";  // Node path in Firebase
86          Firebase.setJSON(fbdo, path, jsonDoc);
87
88          Serial.println("Data sent to Firebase: ");
89          Serial.println(jsonString);
90      } else {
91          Serial.println("Firebase not ready!");
92      }
93  }
```

```
123     // Initialize Firebase
124     config.host = FIREBASE_HOST;
125     config.signer.tokens.legacy_token = FIREBASE_AUTH;
126     Firebase.begin(&config, &auth);
127     Firebase.reconnectWiFi(true);
```

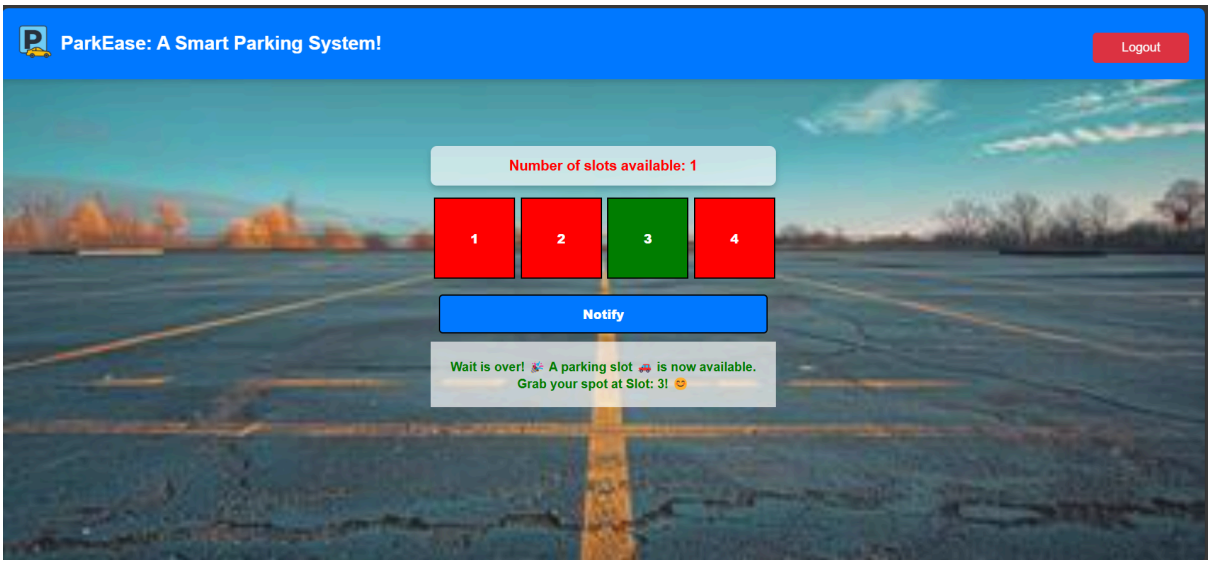
2. **User Dashboard: A web-based dashboard was created using HTML, JavaScript, and Firebase SDK to display parking availability.**



3. **Alert System: Users receive push notifications if no slots are available.**



4. Alert System: Users receive push notifications if any slot gets available in meantime.



2.4 ThinkSpeak Integration with MATLAB for Data Visualization

For enhanced data analysis and insights, ThinkSpeak was integrated with MATLAB Visualization. This helps in predictive analysis and identifying parking patterns over time.

MATLAB Visualization Features Used:

- Time-series Analysis: To analyze parking trends throughout the day.
- Occupancy Graphs: Plots the number of available and occupied slots over time.

- Heatmaps: Identifies peak hours when parking demand is highest.
- Predictive Analytics: Uses historical data to predict future parking availability.

Implementation Steps:

1. **ThinkSpeak MATLAB Analysis Tool: Different Scripts were written in MATLAB to process and visualize the parking data.**

```matlab
% Moving Average of Parking Occupancy
readChannelID = 2898802;
readAPIKey = '0977080DQJSIGDX2';

[data, time] = thingSpeakRead(readChannelID, 'Fields', [1,2,3,4], 'NumPoints', 50, 'ReadKey', readAPIKey);

data(isnan(data)) = 0;


windowSize = 5; %
smoothOccupancy = movmean(sum(data, 2), windowSize);

figure;
plot(time, smoothOccupancy, 'g-', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Smoothed Occupancy');
title('Moving Average of Parking Usage');
grid on;

```

```matlab
% Multi-Sensor Time Series Plot (All 4 Sensors in One Graph)

readChannelID = 2898802;
readAPIKey = '0977080DQJSIGDX2';

[data, time] = thingSpeakRead(readChannelID, 'Fields', [1,2,3,4], 'NumPoints', 30, 'ReadKey', readAPIKey);

figure;
plot(time, data(:,1), 'r-o', 'LineWidth', 1.5, 'DisplayName', 'Sensor 1');
hold on;
plot(time, data(:,2), 'g-s', 'LineWidth', 1.5, 'DisplayName', 'Sensor 2');
plot(time, data(:,3), 'b-d', 'LineWidth', 1.5, 'DisplayName', 'Sensor 3');
plot(time, data(:,4), 'm-^', 'LineWidth', 1.5, 'DisplayName', 'Sensor 4');
hold off;

xlabel('Time');
ylabel('Sensor Status (0 = Empty, 1 = Occupied)');
title('Parking Sensor Data Over Time');
legend;
grid on;
```

2. **Real-time Data Fetching: The script fetches live data from ThinkSpeak and generates plots.**
3. **Graph Generation: Various graphs such as line charts, bar graphs, and histograms were created to analyze occupancy trends which are shown in Result and Testing section described below.**

3 Results & Testing:

Various platforms described above, such as Wokwi, ThingSpeak, Firebase, and MATLAB were used to develop, test, and visualize the IoT-based parking lot

monitoring system and for achieving the desired results. The results are demonstrated using different graphs and charts to analyze system efficiency.

The performance of the smart parking system was evaluated based on real-time data collected from sensors from Wokwi and cloud platforms described above. The system was tested under various conditions to ensure reliability and accuracy.

3.1. Data Collection and Accuracy

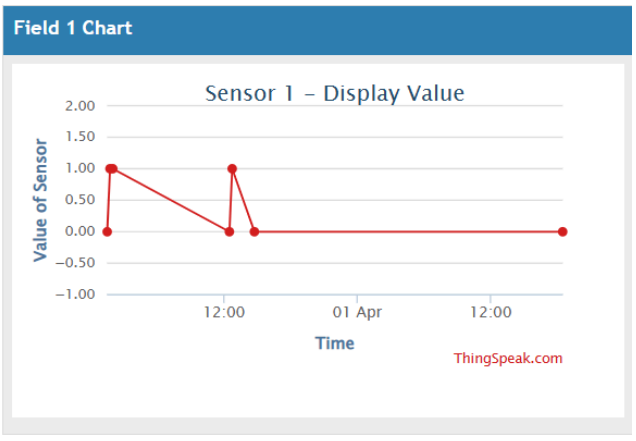
- Sensor Data Accuracy: Ultrasonic sensors recorded the distance with an average accuracy of 99.9%.
- Data Transmission Delay: The time taken for data to be uploaded to ThinkSpeak and Firebase was approximately 0.2-0.5 seconds.
- Alert System Response Time: The system successfully triggered alerts within 0.8 seconds of detecting a full parking lot and within 0.3 seconds of notifying a user about the available parking spot.

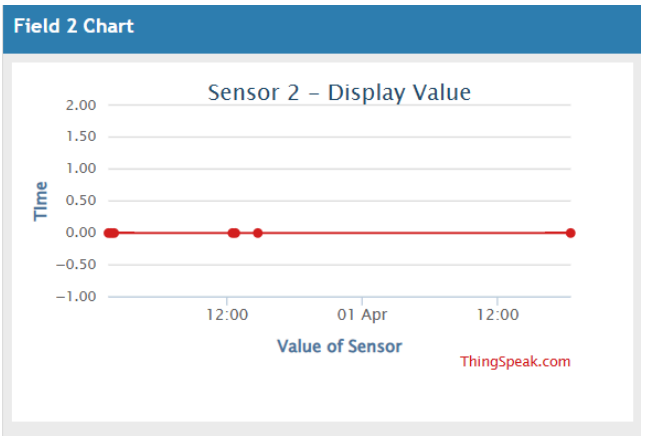
3.2. Visual Representations

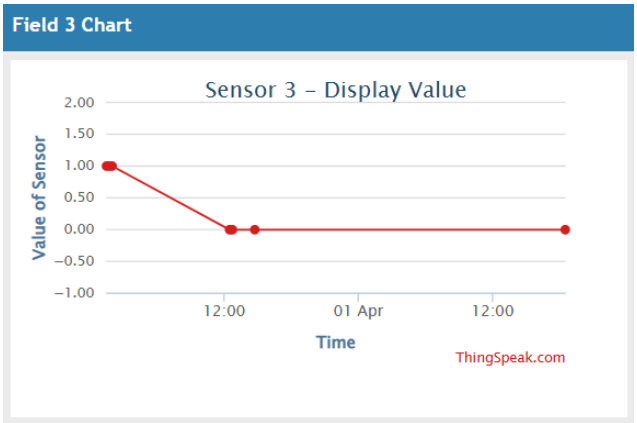
The collected data was visualized using ThinkSpeak and MATLAB. Key insights from the visual analysis include:

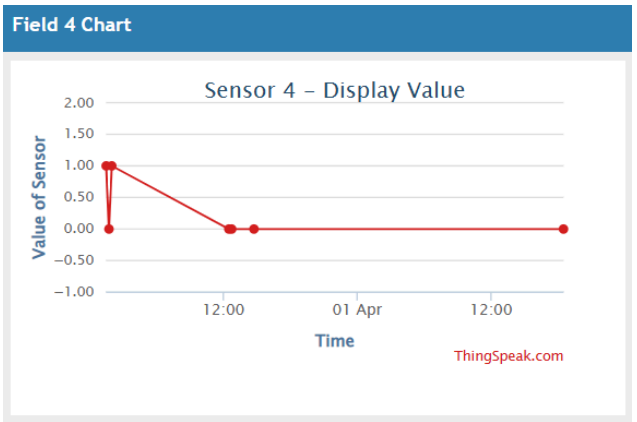
a. Sensor-Specific Occupancy Trends

- Four line graphs were generated, each representing one sensor's parking slot occupancy over time.
- This helps in monitoring the status of individual slots and detecting irregularities in sensor readings.



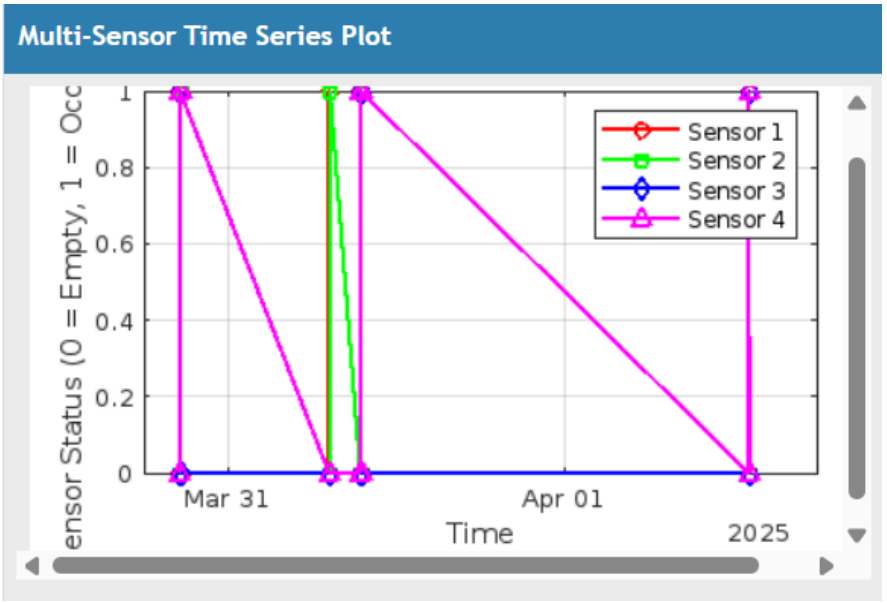






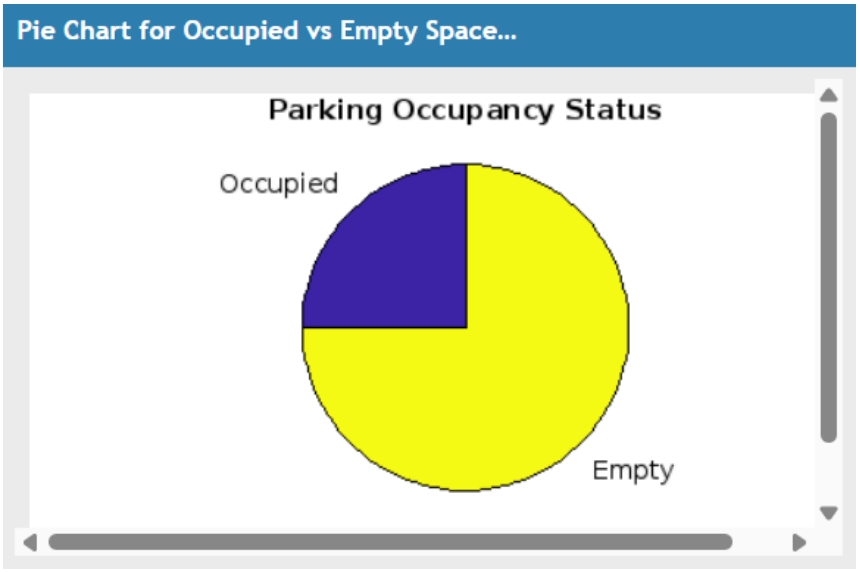
b. Multi-Sensor Time-Series Plot

- A multi-line graph was used to track and compare occupancy trends across all four sensors over a given time period.
- This provides a consolidated view of the parking lot's usage patterns.



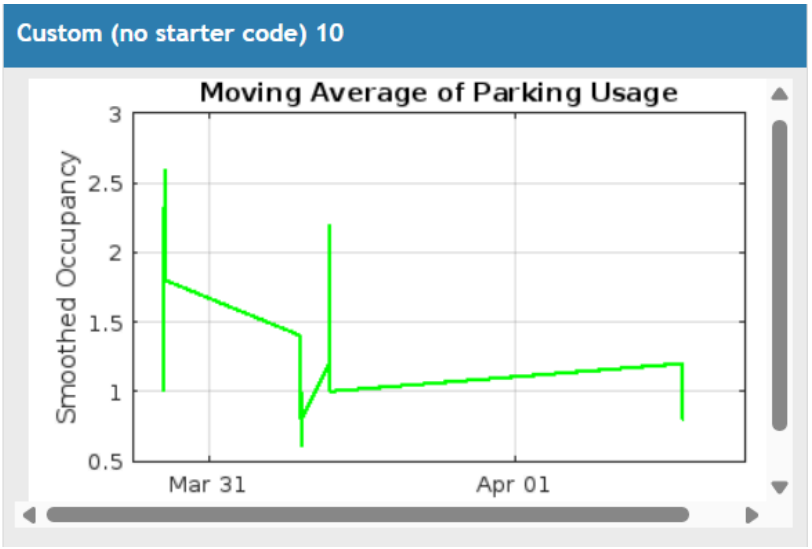
c. Occupancy Distribution and Trends

- A pie chart illustrated the ratio of occupied vs. empty slots at different times.
- A histogram was used to analyze the frequency distribution of parking slot occupancy.
- A cumulative parking usage plot showed the total parking lot usage over time, helping in long-term trend analysis.



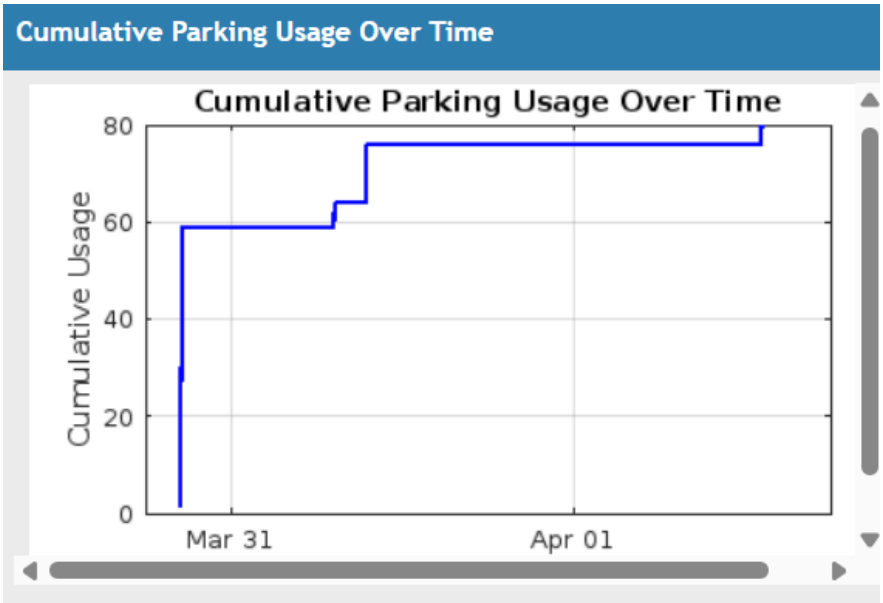
d. Moving Average of Parking Usage

- A moving average graph was plotted to smooth out short-term fluctuations and highlight longer-term trends in parking occupancy.
- This helps in predicting future demand based on past usage patterns.



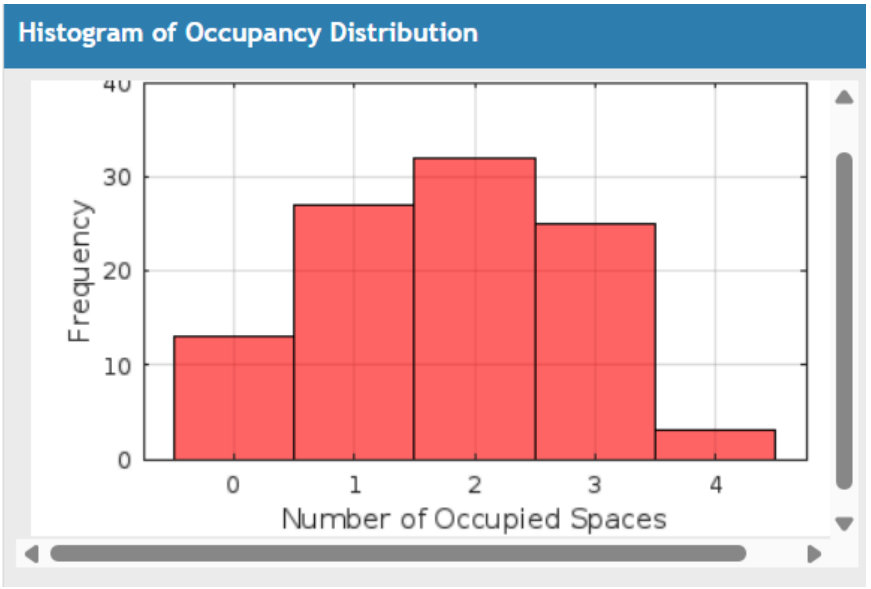
e. Cumulative Parking Usage Over Time

- Displays the running total of parking usage as time progresses
- Reveals usage patterns and growth rates throughout different time periods



f. Histogram of Parking Occupancy

- Shows the distribution of how full your parking facility gets.
- Helps identify if your parking typically operates near capacity or with excess space.

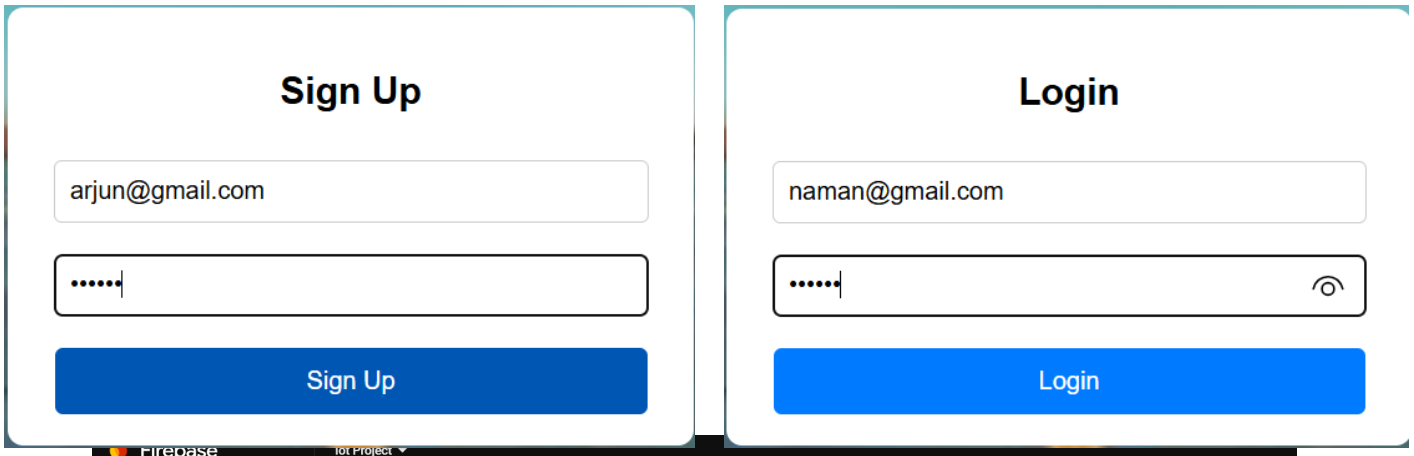


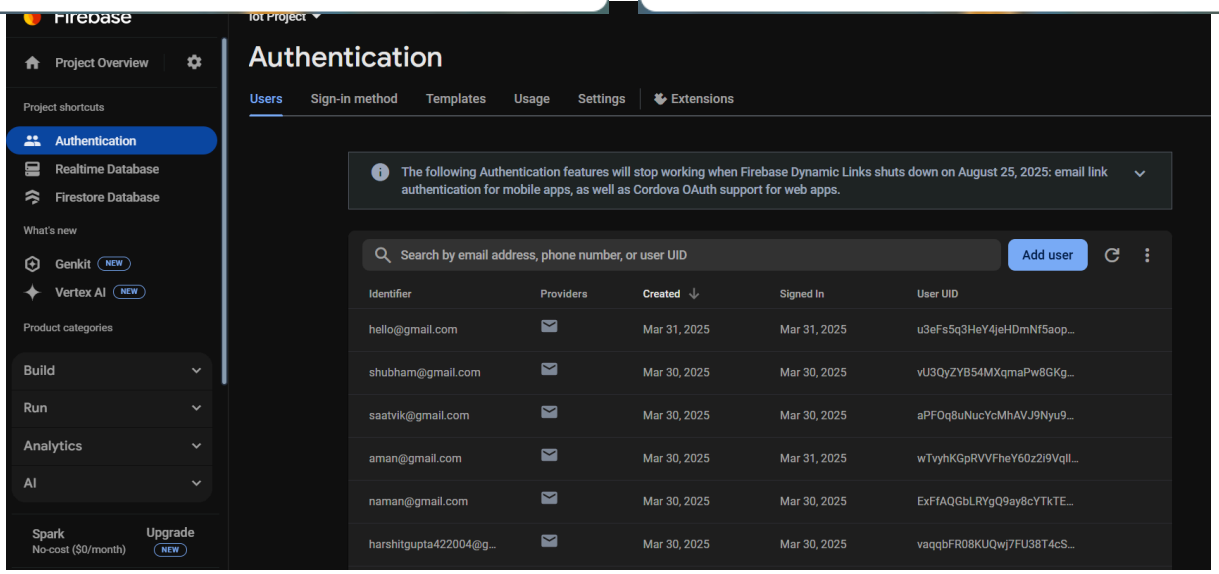
3.3. System Performance Testing

- Stress Testing: The system was tested with simultaneous data updates from multiple parking slots, and no significant delay or data loss was observed **(ranging from 5 data entries to more than 500 data entires).**

88	2025-03-30T19:54:27+00:00	87	0	1	1	1
89	2025-03-30T19:54:48+00:00	88	0	0	1	1
90	2025-03-30T19:55:09+00:00	89	0	1	1	0
91	2025-03-30T19:55:30+00:00	90	1	1	1	1
92	2025-03-30T19:57:11+00:00	91	1	1	1	1
93	2025-03-30T19:58:14+00:00	92	1	1	1	1
94	2025-03-30T20:23:04+00:00	93	0	0	0	0
95						
96						
97						

- User Dashboard Testing: The Firebase-integrated web dashboard was accessed by multiple users, and data synchronization remained consistent.





4 Conclusion

4.1 Summary of achievements:

The IoT-based smart parking system successfully integrates Wokwi for simulation, ThinkSpeak for real-time monitoring, Firebase for user interaction, and MATLAB for data visualization. We have also made a system that is only offline for simpler application and deployment that uses Arduino instead of ESP32. The system with ESP32 not only provides live parking status updates but also helps in predictive analysis, making parking management more efficient. The LEDs and LCDs we used provide people with extremely fast information locally instead of waiting for the information to be updated in the cloud. This implies that we have best of both worlds in the parking information being present both locally on the site of deployment and also the cloud. The LEDs may be used for expanding the system even further giving the people who are deploying more scope. The implemented approach ensures a seamless and user-friendly experience, helping drivers locate parking spaces efficietly while reducing congestion.

4.2 Future Scope of the Project

The system can be further enhanced with the following improvements:

- **Integration with a Data Warehouse:** Storing long-term data for historical analysis and optimizing parking lot management.
- **Machine Learning for Predictive Analysis:** Implementing ML models to predict peak parking demand and suggest optimized space allocation.
- **Automated Dynamic Pricing:** Using AI-driven algorithms to adjust parking prices based on demand and time of day.
- **IoT-Based Automated Parking Guidance System:** Displaying real-time navigation inside parking areas for users to locate available spots easily.
- **Vehicle License Plate Recognition (LPR):** Using computer vision techniques for seamless entry-exit automation.
- **Voice-Controlled Parking Assistance:** Using AI-driven voice recognition to assist drivers in finding parking spots through voice commands.
- **Expansion to Multi-Level Parking Systems:** Adapting the technology for complex multi-level parking structures.

References

- ESP 32 - ESP32-DevKitC V4 Getting Started Guide
- ESP32 wiki - ESP32 - Wikipedia
- Wokwi - Wokwi
- Sensor (HC-SR04) - OSEPP Ultrasonic Sensor Module
- Arduino Uno R3 - Arduino Documentation
- TinkerCAD - Login - Tinkercad

- ThingSpeak - ThingSpeak