Project Documentation: Smart Parking using IoT

Jeya Krishnan
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Dhivyan E
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Ajith Kumar P
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Mohammed Musthaba Majith A
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Project Objectives:

The project aims to create a smart parking system that provides real-time parking availability information to users. The key objectives include:
Real-Time Parking Availability: Implement IoT sensors to detect parking space occupancy and relay real-time data to a central server.
User-Friendly Mobile App: Develop a user-friendly mobile app that displays parking availability information to users.
Raspberry Pi Integration: Use Raspberry Pi as a server to process data from IoT sensors and serve it to the mobile app.
Efficient Utilization: Improve parking space utilization by guiding users to available parking spots, reducing congestion, and saving time.

IoT Sensor Setup:

Parking Sensors: Deploy ultrasonic or infrared sensors in parking spaces to detect vehicle presence. These sensors send occupancy data to a microcontroller.
Microcontroller: Use microcontrollers like Arduino or ESP8266 to gather data from parking sensors and send it to the Raspberry Pi server over a network.
Network Connectivity: Connect the microcontroller to the server using protocols like MQTT or HTTP to transmit real-time data securely.

Mobile App Development:

User Interface Design: Create an intuitive and user-friendly interface displaying parking availability. Include features like maps, user location, and real-time updates.
Frontend Development: Use Flutter or another mobile app framework to develop the app, allowing cross-platform compatibility for both Android and iOS devices.
Backend Integration: Integrate the app with backend services to fetch real-time data from the Raspberry Pi server. Implement logic to process and display parking availability status.

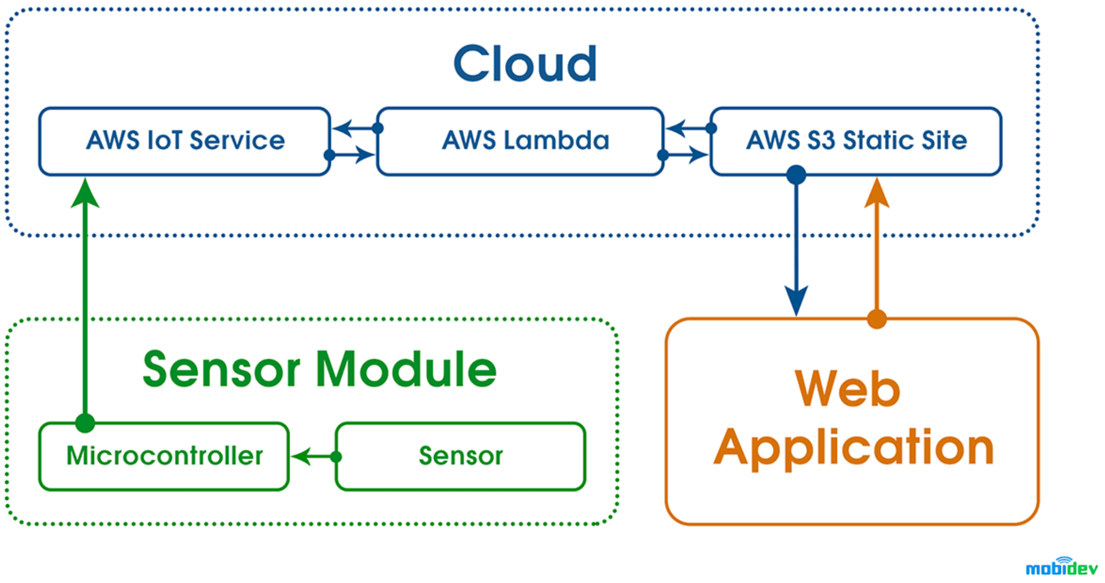
Raspberry Pi Integration:

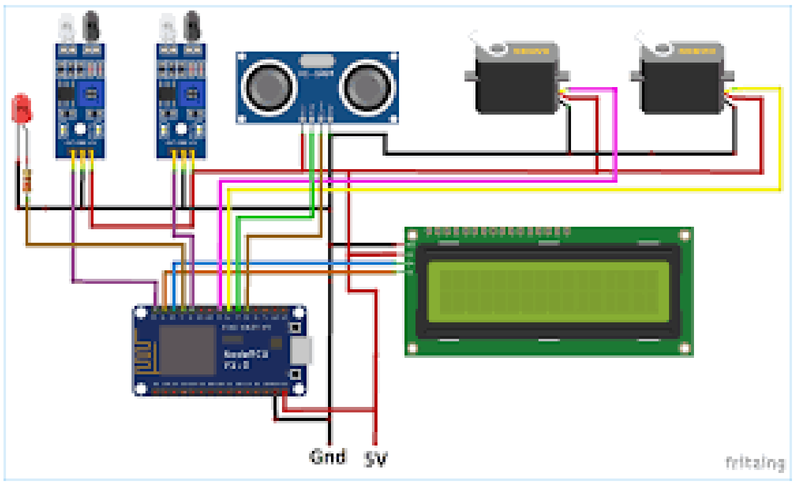
Flask Server: Set up a Flask server on the Raspberry Pi to receive data from IoT sensors. Implement REST API endpoints to handle sensor data transmission and retrieval.
Data Processing: Process the incoming sensor data, determine parking space availability, and update the database in real-time.
Database: Store parking availability data in a database (e.g., SQLite or MongoDB) for efficient retrieval and management.
API Endpoints: Create API endpoints such as `/api/parking` to serve parking availability data to the mobile app.

Code Implementation:

IoT Sensor Code: Write firmware for microcontrollers to read data from parking sensors and transmit it to the Raspberry Pi server using MQTT or HTTP protocols.
Flask Server Code: Develop Flask routes to handle sensor data, update the database, and serve data to the mobile app. Implement error handling and security measures.
Mobile App Code: Implement frontend logic to fetch data from the Flask server API endpoints. Update UI components dynamically based on the received data. Handle user interactions and display appropriate messages.

By following these steps and integrating the IoT sensors, Raspberry Pi, and mobile app effectively, the project can achieve its objectives of providing real-time parking availability information to users, enhancing parking space utilization, and improving overall user experience.
DIAGRAMS,SCHEMATICS





A real-time parking availability system in the context of Smart Parking IoT (Internet of Things) can significantly benefit drivers and help alleviate parking issues in several ways:

Time and Fuel Savings:Drivers can save time and fuel by quickly locating available parking spaces instead of driving around in search of a spot. Real-time information about parking availability reduces the time spent circling blocks, leading to fuel savings and reduced emissions.

Reduced Traffic Congestion: When drivers can efficiently find parking spaces, it reduces traffic congestion on the roads. Less time spent searching for parking means fewer vehicles on the road, contributing to smoother traffic flow.

Enhanced User Experience: Smart Parking IoT solutions often come with mobile apps or websites that provide real-time information to drivers. This enhances the overall user experience, making it convenient for drivers to plan their parking in advance.

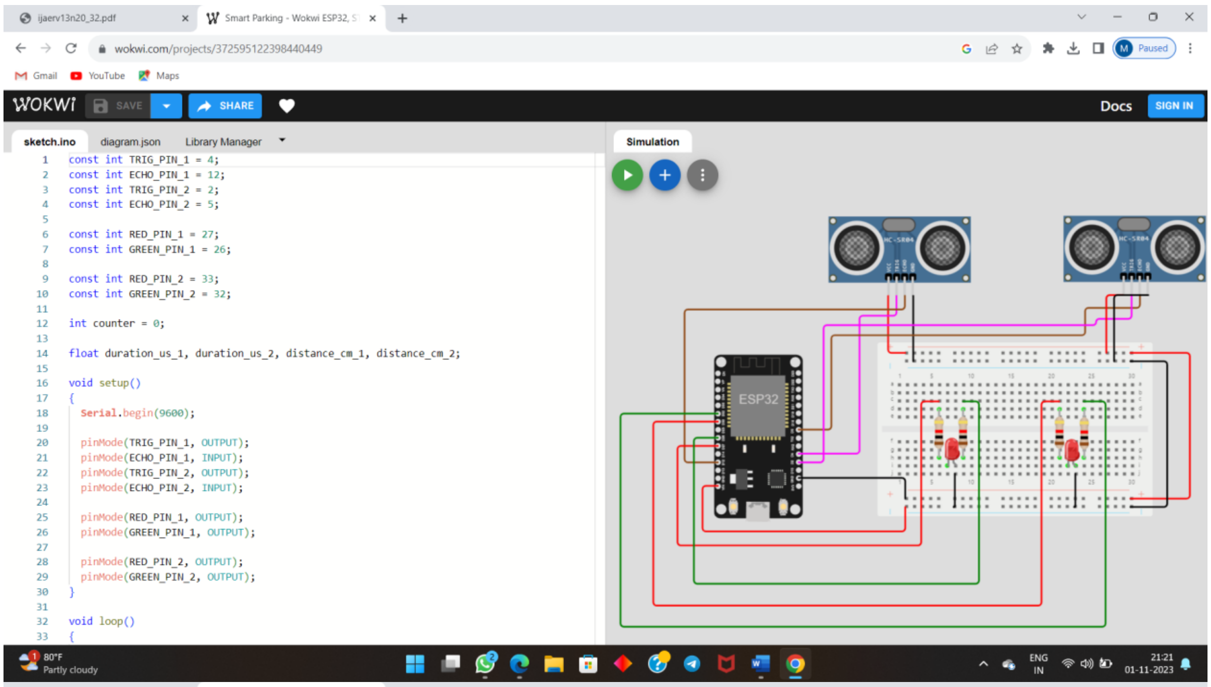
Optimized Space Utilization: Real-time data on parking availability allows parking operators to optimize the utilization of parking spaces. They can analyze patterns and demand, ensuring that spaces are used efficiently. This optimization can lead to the creation of additional parking spaces in high-demand areas.

Improved Revenue Generation:Parking operators can implement dynamic pricing strategies based on real-time demand. Higher prices during peak hours or special events can help regulate demand and maximize revenue. This revenue can be reinvested in urban infrastructure or public services.

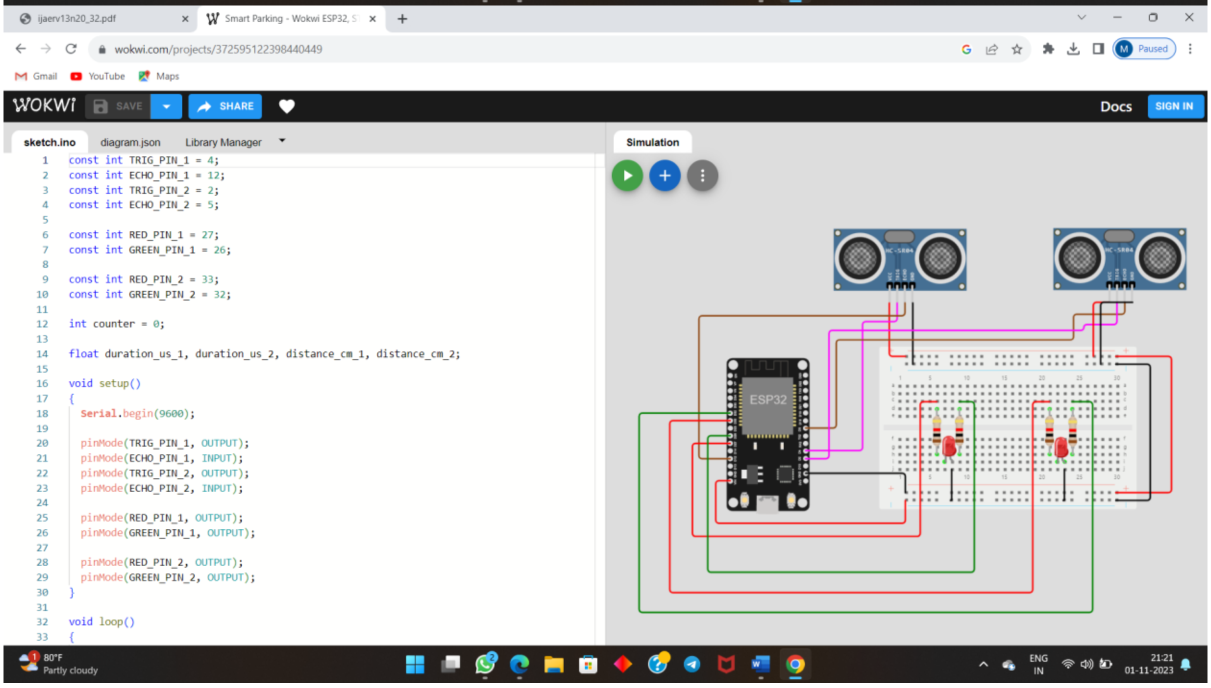
Integration with Navigation Systems: Real-time parking availability information can be integrated into GPS and navigation systems, allowing drivers to be guided directly to available parking spaces. This integration streamlines the parking process, making it seamless for users.

Reduction in Illegal Parking:With real-time monitoring, parking operators can detect illegal parking more efficiently. Automated alerts can be sent to authorities, ensuring that parking rules are enforced effectively.

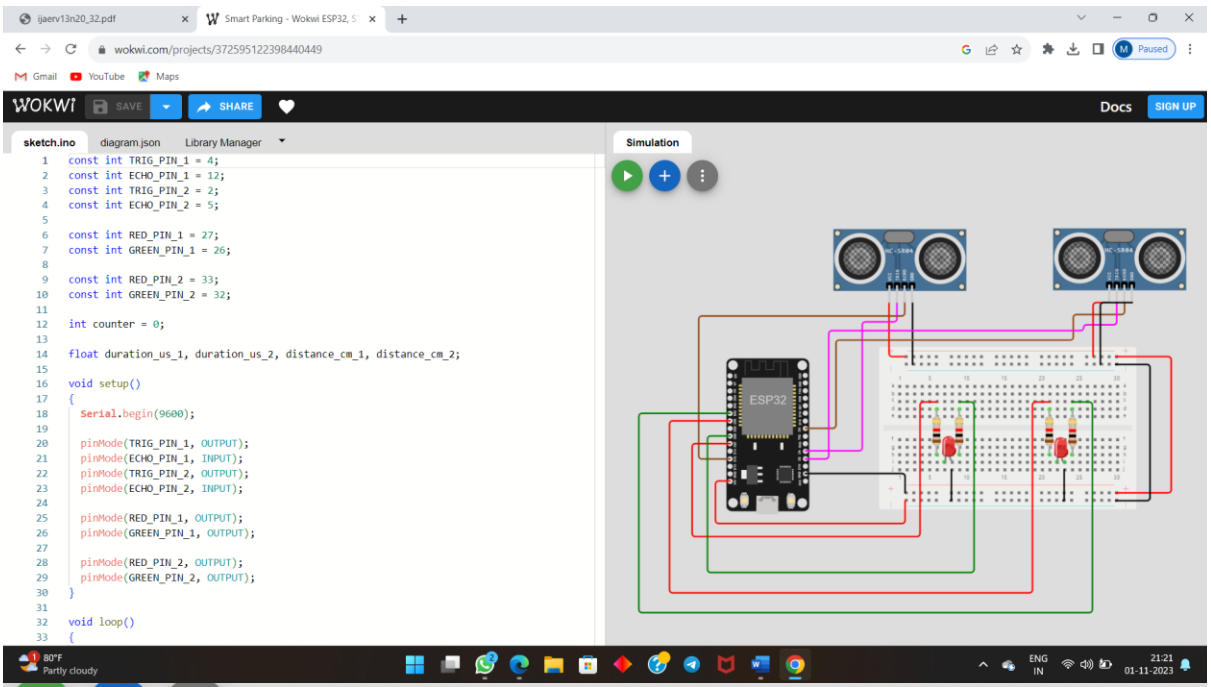
WOWKI STIMULATION:



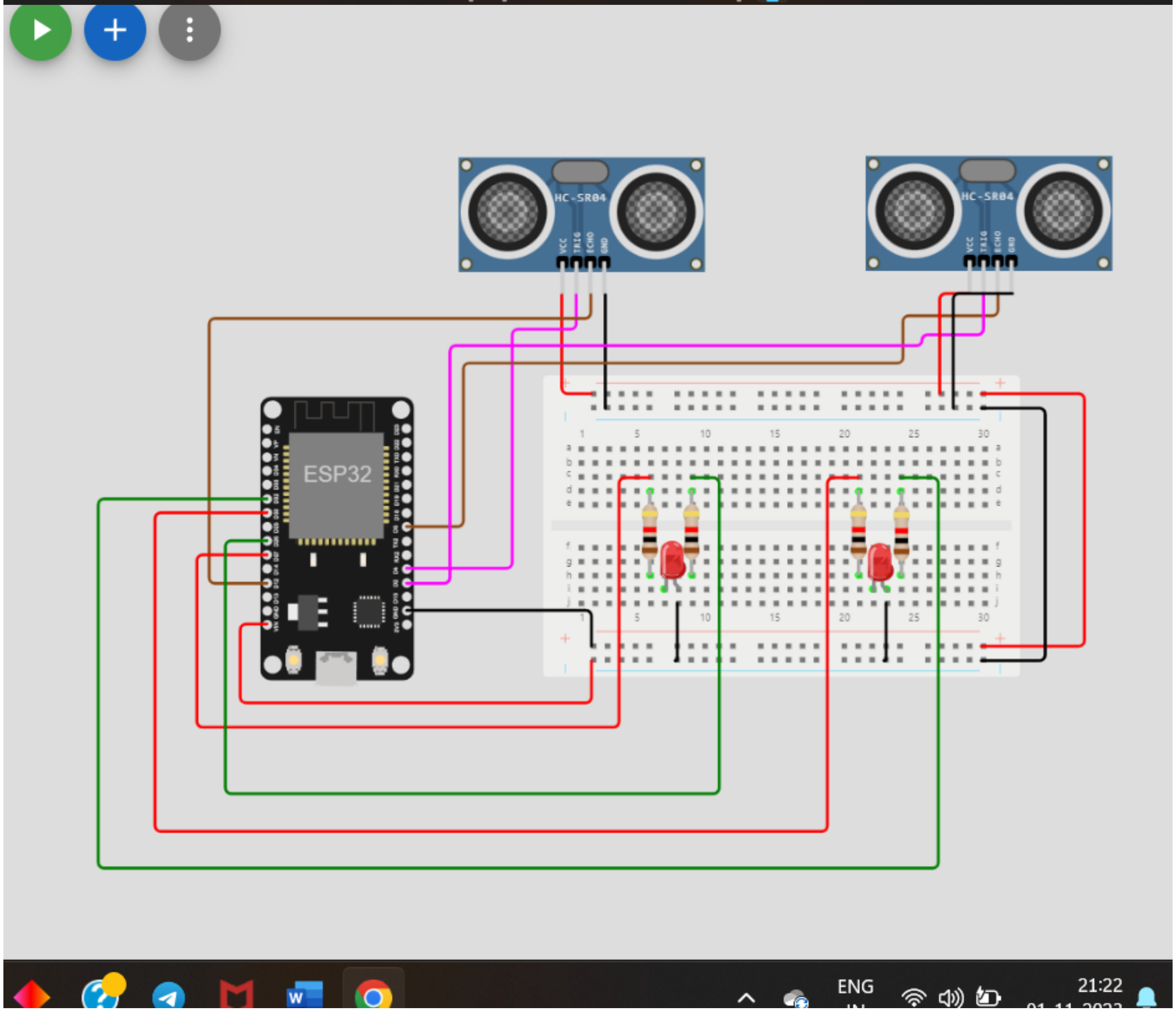
```
const int TRIG_PIN_1 = 4;
const int ECHO_PIN_1 = 12;
const int TRIG_PIN_2 = 2;
const int ECHO_PIN_2 = 5;

const int RED_PIN_1 = 27;
const int GREEN_PIN_1 = 26;

const int RED_PIN_2 = 33;
const int GREEN_PIN_2 = 32;

int counter = 0;

float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;

void setup()
{
  Serial.begin(9600);

  pinMode(TRIG_PIN_1, OUTPUT);
  pinMode(ECHO_PIN_1, INPUT);
  pinMode(TRIG_PIN_2, OUTPUT);
  pinMode(ECHO_PIN_2, INPUT);

  pinMode(RED_PIN_1, OUTPUT);
  pinMode(GREEN_PIN_1, OUTPUT);

  pinMode(RED_PIN_2, OUTPUT);
  pinMode(GREEN_PIN_2, OUTPUT);
}

void loop()
{
```

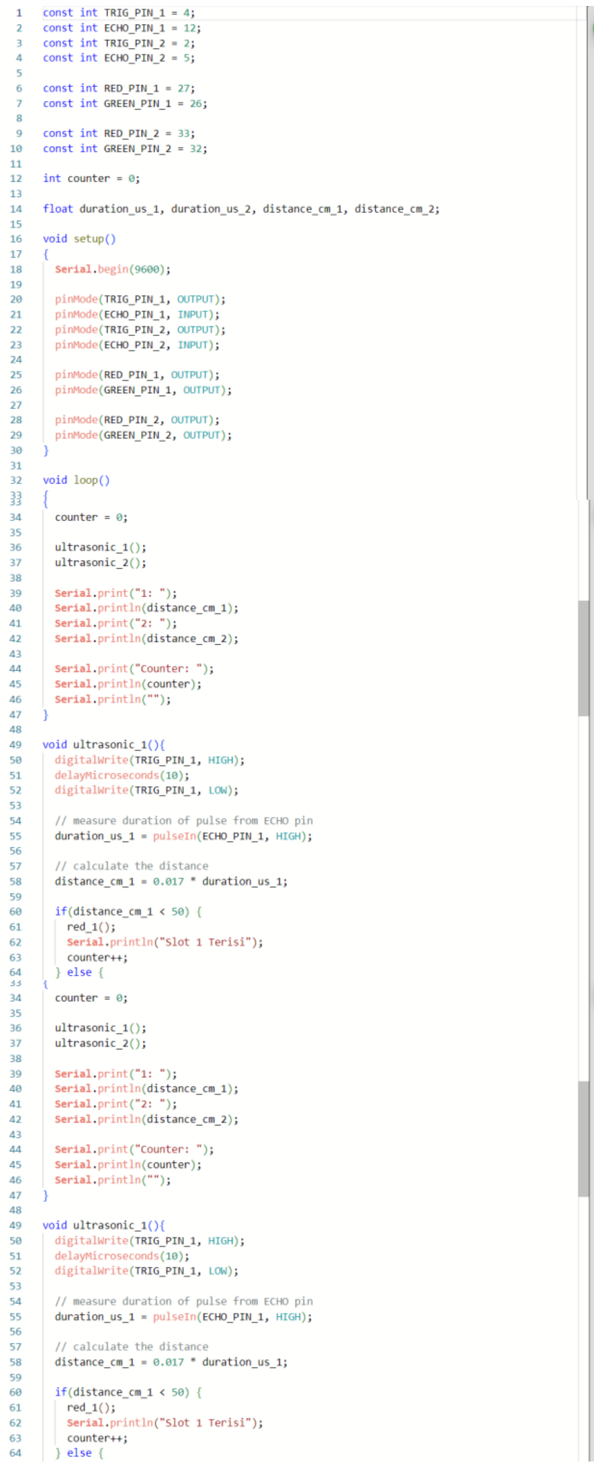


```
const int TRIG_PIN_1 = 4;
const int ECHO_PIN_1 = 12;
const int TRIG_PIN_2 = 2;
const int ECHO_PIN_2 = 5;

const int RED_PIN_1 = 27;
const int GREEN_PIN_1 = 26;

const int RED_PIN_2 = 33;
const int GREEN_PIN_2 = 32;

int counter = 0;

float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;

void setup()
{
  Serial.begin(9600);

  pinMode(TRIG_PIN_1, OUTPUT);
  pinMode(ECHO_PIN_1, INPUT);
  pinMode(TRIG_PIN_2, OUTPUT);
  pinMode(ECHO_PIN_2, INPUT);

  pinMode(RED_PIN_1, OUTPUT);
  pinMode(GREEN_PIN_1, OUTPUT);

  pinMode(RED_PIN_2, OUTPUT);
  pinMode(GREEN_PIN_2, OUTPUT);
}

void loop()
{
```



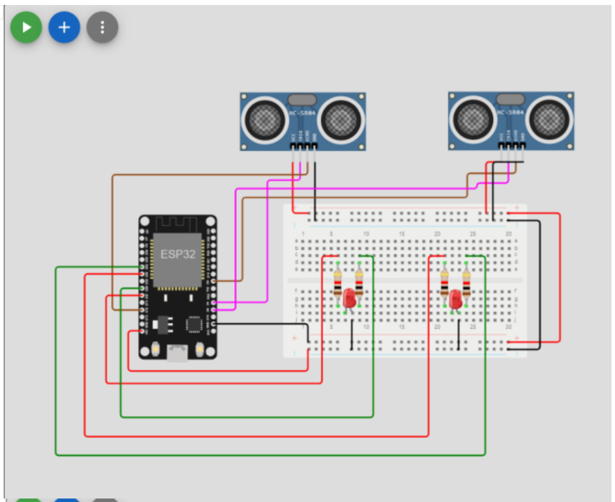
```
const int TRIG_PIN_1 = 4;
const int ECHO_PIN_1 = 12;
const int TRIG_PIN_2 = 2;
const int ECHO_PIN_2 = 5;

const int RED_PIN_1 = 27;
const int GREEN_PIN_1 = 26;

const int RED_PIN_2 = 33;
const int GREEN_PIN_2 = 32;

int counter = 0;

float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;

void setup()
{
  Serial.begin(9600);

  pinMode(TRIG_PIN_1, OUTPUT);
  pinMode(ECHO_PIN_1, INPUT);
  pinMode(TRIG_PIN_2, OUTPUT);
  pinMode(ECHO_PIN_2, INPUT);

  pinMode(RED_PIN_1, OUTPUT);
  pinMode(GREEN_PIN_1, OUTPUT);

  pinMode(RED_PIN_2, OUTPUT);
  pinMode(GREEN_PIN_2, OUTPUT);
}

void loop()
{
```





```
const int TRIG_PIN_1 = 4;
const int ECHO_PIN_1 = 12;
const int TRIG_PIN_2 = 2;
const int ECHO_PIN_2 = 5;

const int RED_PIN_1 = 27;
const int GREEN_PIN_1 = 26;

const int RED_PIN_2 = 33;
const int GREEN_PIN_2 = 32;

int counter = 0;

float duration_us_1, duration_us_2, distance_cm_1, distance_cm_2;

void setup()
{
  Serial.begin(9600);

  pinMode(TRIG_PIN_1, OUTPUT);
  pinMode(ECHO_PIN_1, INPUT);
  pinMode(TRIG_PIN_2, OUTPUT);
  pinMode(ECHO_PIN_2, INPUT);

  pinMode(RED_PIN_1, OUTPUT);
  pinMode(GREEN_PIN_1, OUTPUT);

  pinMode(RED_PIN_2, OUTPUT);
  pinMode(GREEN_PIN_2, OUTPUT);
}

void loop()
{
  counter = 0;

  ultrasonic_1();
  ultrasonic_2();

  Serial.print("1: ");
  Serial.println(distance_cm_1);
  Serial.print("2: ");
  Serial.println(distance_cm_2);

  Serial.print("Counter: ");
  Serial.println(counter);
  Serial.println("");
}

void ultrasonic_1(){
  digitalWrite(TRIG_PIN_1, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN_1, LOW);

  // measure duration of pulse from ECHO pin
  duration_us_1 = pulseIn(ECHO_PIN_1, HIGH);

  // calculate the distance
  distance_cm_1 = 0.017 * duration_us_1;

  if(distance_cm_1 < 50) {
    red_1();
    Serial.println("Slot 1 Terisi");
    counter++;
  } else {
```

```
{
  counter = 0;

  ultrasonic_1();
  ultrasonic_2();

  Serial.print("1: ");
  Serial.println(distance_cm_1);
  Serial.print("2: ");
  Serial.println(distance_cm_2);

  Serial.print("Counter: ");
  Serial.println(counter);
  Serial.println("");
}

void ultrasonic_1(){
  digitalWrite(TRIG_PIN_1, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN_1, LOW);

  // measure duration of pulse from ECHO pin
  duration_us_1 = pulseIn(ECHO_PIN_1, HIGH);

  // calculate the distance
  distance_cm_1 = 0.017 * duration_us_1;

  if(distance_cm_1 < 50) {
    red_1();
    Serial.println("Slot 1 Terisi");
    counter++;
  } else {
```


ESP32
HC-SR04
HC-SR04