**Project Report: Water Level Monitoring and Control System**

**1. Project Overview**

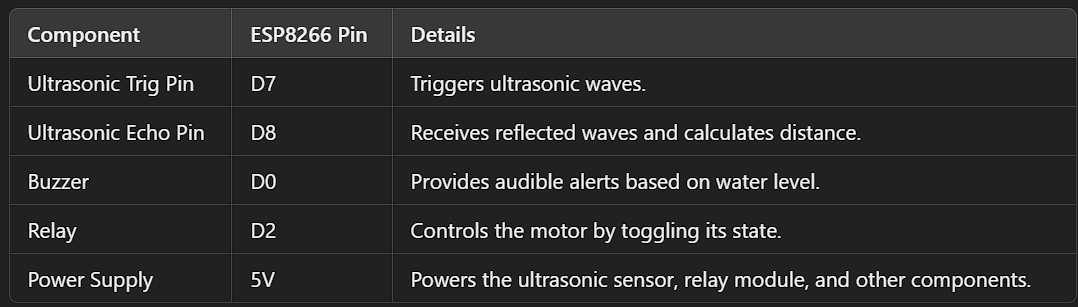
The Water Level Monitoring and Control System is a microcontroller-based project designed to monitor the water level in a tank and control a motor (via a relay) and buzzer based on specific thresholds. The project also includes a web-based interface to display real-time data such as the water level, motor status, and buzzer status.

#### ****2. Features****

* **Water Level Monitoring:** Measures the distance of water from an ultrasonic sensor to determine the water level.
* **Motor Control:** Automatically switches the motor ON/OFF depending on water level thresholds.
* **Buzzer Alerts:** Provides audio alerts for extreme water levels (tank empty or full).
* **Web Interface:** Displays real-time water level data, status, and control actions via a web page accessible through Wi-Fi.
* **Threshold-Based Actions:** Implements multiple levels to categorize water level statuses (Very Low, Low, Medium, High, Full).
* **API Integration:** A REST API endpoint (/level) provides JSON data for external integration.

#### ****3. Components and Tools****

* **Hardware:**
  + ESP8266 Microcontroller (e.g., NodeMCU)
  + HC-SR04 Ultrasonic Sensor
  + Buzzer
  + Relay Module (for motor control)
  + Jumper Wires and Breadboard
  + 5V Power Supply
* **Software:**
  + Arduino IDE
  + Embedded C Programming
  + HTML/CSS for the Web Interface
  + JavaScript (for fetching and updating real-time data)

**Connections**

**Note:** The ESP8266 operates at 3.3V logic, so ensure proper level shifting or use compatible modules where required.

### ****Process Flow****

1. **Setup**:
   * The ESP8266 connects to a Wi-Fi network using the credentials provided.
   * All pins (trigger, echo, relay, and buzzer) are initialized.
2. **Main Loop**:
   * The water level is measured using the ultrasonic sensor.
   * The distance is categorized into levels (Very Low to Full) based on thresholds.
   * The relay and buzzer are toggled based on these levels.
   * The web server provides JSON responses for the front-end interface.
3. **Web Interface**:
   * Displays real-time water level, motor, and buzzer status using JavaScript fetch() requests.
   * Refreshes automatically every second.

### ****Threshold Levels****

* Maximum Tank Level: **20 cm**
* Level Thresholds:
  + **Very Low**: Distance ≥ 75% of MaxLevel (15 cm)
  + **Low**: Distance ≥ 65% of MaxLevel (13 cm)
  + **Medium**: Distance ≥ 55% of MaxLevel (11 cm)
  + **High**: Distance ≥ 45% of MaxLevel (9 cm)
  + **Full**: Distance < 35% of MaxLevel (7 cm)

**Program**

#include <ESP8266WiFi.h>

// WiFi credentials

const char\* ssid = " Your\_WiFi\_SSID"; // Wi-Fi network name (SSID)

const char\* password = " Your\_WiFi\_Password"; // Wi-Fi network password

// Define component pins

#define trig D7

#define echo D8

#define buzzer D0   // Buzzer pin

#define relay D2    // Motor control relay

// Define tank max level (in CM)

int MaxLevel = 20;

int Level1 = (MaxLevel \* 75) / 100;

int Level2 = (MaxLevel \* 65) / 100;

int Level3 = (MaxLevel \* 55) / 100;

int Level4 = (MaxLevel \* 45) / 100;

int Level5 = (MaxLevel \* 35) / 100;

// Web server on port 80

WiFiServer server(80);

// Store the HTML web page

String webPage = R"rawliteral(

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Water Level Monitoring</title>

    <style>

        body { font-family: Arial, sans-serif; background-color: #f4f4f4; margin: 0; padding: 0; }

        header { background-color: #0078D7; color: white; padding: 1rem 0; text-align: center; }

        .container { max-width: 800px; margin: 2rem auto; background: white; padding: 2rem; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); }

        h1, h2 { color: #333; }

        footer { background-color: #0078D7; color: white; text-align: center; padding: 1rem 0; margin-top: 2rem; }

    </style>

</head>

<body>

    <header>

        <h1>Water Level Monitoring System</h1>

    </header>

    <div class="container">

        <h2>Water Level: <span id="level">Loading...</span> cm</h2>

        <h2>Status: <span id="status">Loading...</span></h2>

        <h2>Motor Status: <span id="motorStatus">Loading...</span></h2>

        <h2>Buzzer Status: <span id="buzzerStatus">Loading...</span></h2>

    </div>

    <footer>

        <p>&copy; 2024 Water Level Monitoring System</p>

    </footer>

    <script>

        setInterval(() => {

            fetch('/level')

                .then(response => response.json())

                .then(data => {

                    document.getElementById('level').textContent = data.level;

                    document.getElementById('status').textContent = data.status;

                    document.getElementById('motorStatus').textContent = data.motorStatus;

                    document.getElementById('buzzerStatus').textContent = data.buzzerStatus;

                });

        }, 1000);  // Update every second

    </script>

</body>

</html>

)rawliteral";

void setup() {

  Serial.begin(115200); // Debugging

  // Connect to Wi-Fi

  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");

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

    delay(100);  // Reduced delay to avoid long waiting time

    Serial.print(".");

  }

  Serial.println("\nWiFi connected!");

  Serial.print("IP Address: ");

  Serial.println(WiFi.localIP());

  // Start the server

  server.begin();

  // Set up pins

  pinMode(trig, OUTPUT);

  pinMode(echo, INPUT);

  pinMode(buzzer, OUTPUT);

  pinMode(relay, OUTPUT);

  digitalWrite(relay, HIGH); // Initially turn off relay (motor off)

  digitalWrite(buzzer, LOW); // Initially turn off buzzer

}

void loop() {

  WiFiClient client = server.available(); // Check for incoming client

  if (client) {

    Serial.println("New Client Connected");

    String request = client.readStringUntil('\r');

    client.flush();

    if (request.indexOf("/level") >= 0) {

      // Handle water level API request

      int distance = measureDistance();

      String status = getStatus(distance);

      String motorStatus = (digitalRead(relay) == LOW) ? "ON" : "OFF";

      String buzzerStatus = (digitalRead(buzzer) == HIGH) ? "ON" : "OFF";

      String jsonResponse = "{\"level\":" + String(distance) + ",\"status\":\"" + status + "\",\"motorStatus\":\"" + motorStatus + "\",\"buzzerStatus\":\"" + buzzerStatus + "\"}";

      client.println("HTTP/1.1 200 OK");

      client.println("Content-Type: application/json");

      client.println("Connection: close");

      client.println();

      client.println(jsonResponse);

    } else {

      // Serve the HTML page

      client.println("HTTP/1.1 200 OK");

      client.println("Content-Type: text/html");

      client.println("Connection: close");

      client.println();

      client.println(webPage);

    }

    delay(1);

    Serial.println("Client Disconnected");

  }

  // Measure distance and control buzzer/relay

  int distance = measureDistance();

  controlBuzzerAndRelay(distance);

  delay(100); // Reduced delay for better responsiveness

}

// Measure distance using ultrasonic sensor

int measureDistance() {

  digitalWrite(trig, LOW);

  delayMicroseconds(2);  // Reduced delay for more accurate readings

  digitalWrite(trig, HIGH);

  delayMicroseconds(10);

  digitalWrite(trig, LOW);

  long duration = pulseIn(echo, HIGH);

  int distance = duration / 29 / 2;

  return distance;

}

// Determine water level status

String getStatus(int distance) {

  if (distance >= Level1) return "Very Low";

  else if (distance >= Level2) return "Low";

  else if (distance >= Level3) return "Medium";

  else if (distance >= Level4) return "High";

  else return "Full";

}

// Control Buzzer and Relay based on water level

void controlBuzzerAndRelay(int distance) {

  if (distance >= Level1) {

    // Tank is empty, buzzer on/off twice

    digitalWrite(buzzer, HIGH);  // Turn buzzer ON

    delay(300);                   // Wait for 300ms

    digitalWrite(buzzer, LOW);    // Turn buzzer OFF

    delay(300);                   // Wait for 300ms

    digitalWrite(relay, LOW);     // Turn relay OFF (motor off)

  }

  else if (distance <= Level5) {

    // Tank is full, buzzer ON for 5 seconds and relay OFF (motor off)

    digitalWrite(buzzer, HIGH);  // Turn buzzer ON

    delay(5000);                 // Wait for 5000ms (5 seconds)

    digitalWrite(buzzer, LOW);   // Turn buzzer OFF

    digitalWrite(relay, HIGH);   // Turn relay OFF (motor on)

  }

  else {

    // Tank is neither full nor empty, relay stays OFF

    digitalWrite(relay, LOW);    // Turn relay OFF (motor off)

  }

}

### ****Advantages****

1. Remote monitoring of water levels via a web interface.
2. Automation reduces manual intervention.
3. Prevents water wastage by turning off the motor when the tank is full.
4. Audible alerts provide an additional layer of notification.

### ****Challenges****

1. **Wi-Fi Connectivity**:
   * The system relies on a stable Wi-Fi connection for monitoring.
2. **Sensor Accuracy**:
   * Ultrasonic sensors may produce fluctuating readings due to water ripples.
3. **Power Supply**:
   * Ensuring consistent power for ESP8266 and connected modules is critical.

### ****Applications****

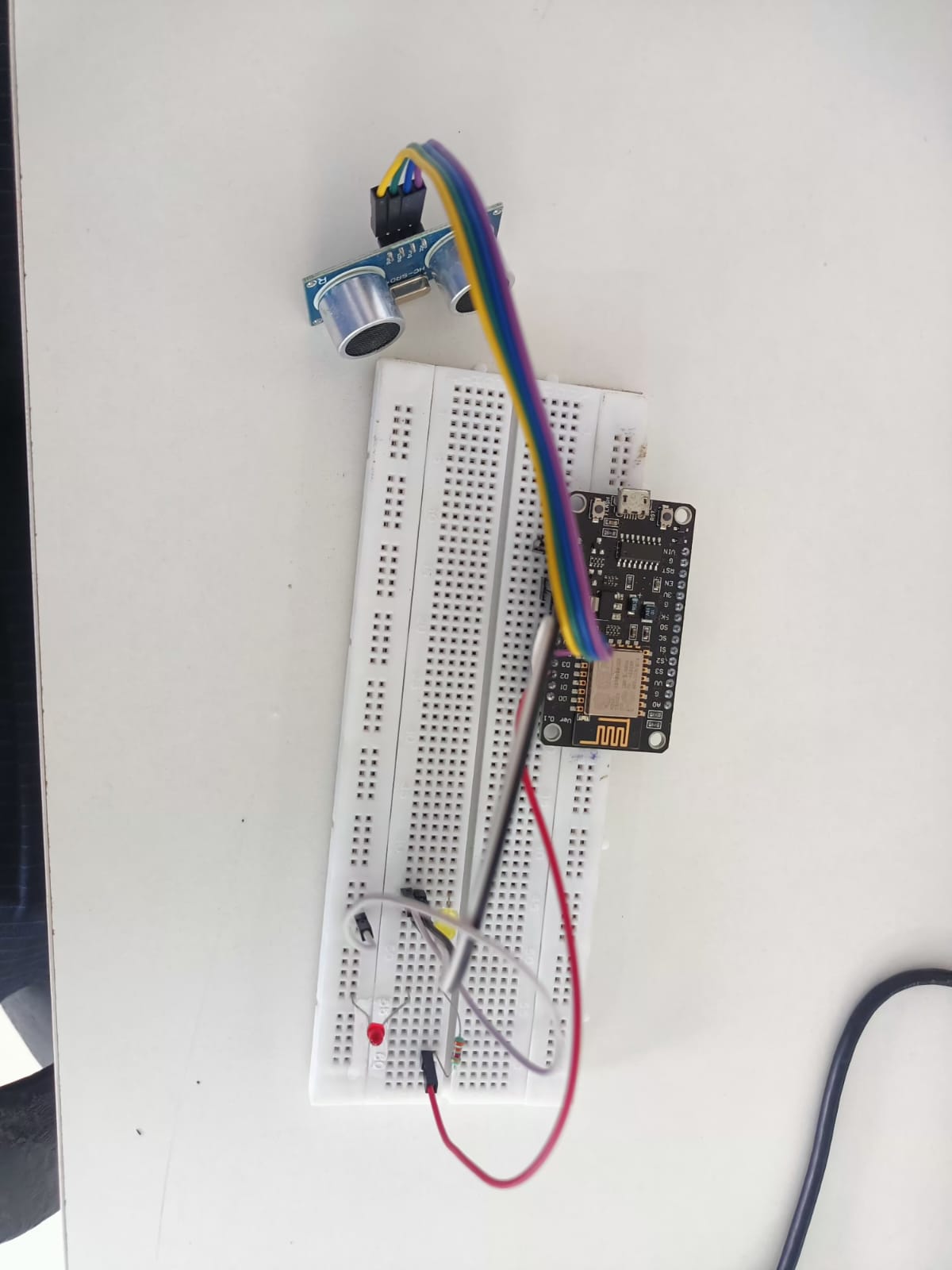
1. Domestic water tank monitoring.
2. Industrial water management systems.
3. Agriculture and irrigation automation.

### ****Future Enhancements****

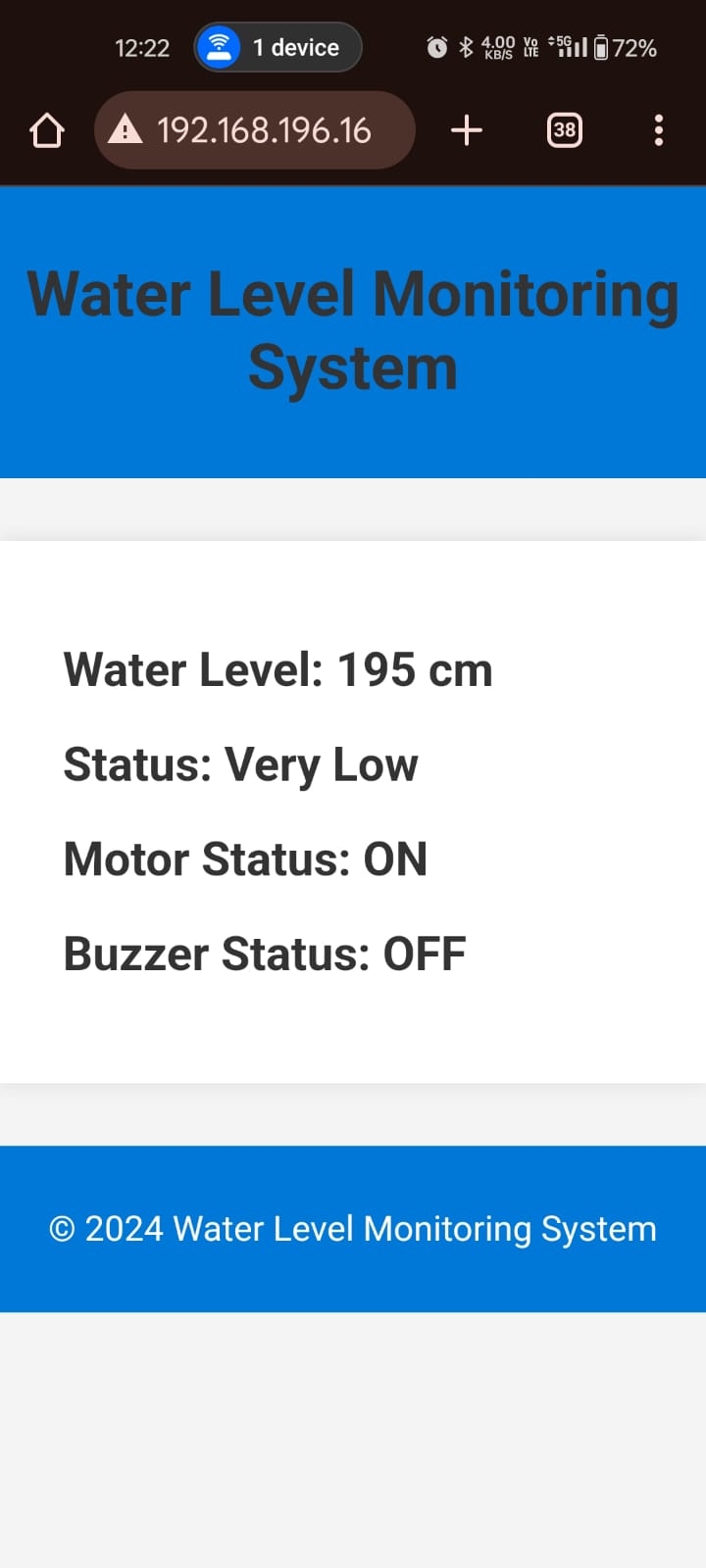
1. Add a mobile app for remote monitoring.
2. Use a water-proof ultrasonic sensor for outdoor applications.
3. Integrate a battery backup to handle power outages.
4. Add notifications via SMS or email when critical thresholds are reached.

### ****Conclusion****

This project successfully demonstrates an automated water level monitoring system with motor control, buzzer alerts, and a real-time web interface. It is a cost-effective and reliable solution for water management in households and industries.

**image of actual project**

**Image of web page**

****