

**A  
PROJECT REPORT  
ON  
“SALINEIQ-AN IOT BASED SALINE MONITORING  
SYSTEM”**

*Submitted in  
Partial Fulfillment of requirement for the Award Of*

**DIPLOMA  
In  
COMPUTER ENGINEERING**

**Submitted by**

<b>T.RITHVIKESH GOUD</b>	<b>22259-CS-017</b>
<b>G.VIKAS GOUD</b>	<b>22259-CS- 026</b>
<b>M.RAKESH YADAV</b>	<b>22259-CS-033</b>
<b>M.NIHARIKA</b>	<b>22259-CS-057</b>
<b>K.AMRUTHA</b>	<b>22259-CS-038</b>
<b>B.SRAVANI SANDHYA</b>	<b>22259-CS-043</b>

**Under the esteemed guidance of  
Mr. Vamsikrishna (M. Tech)**



**SAMSKRUTI COLLEGE OF ENGINEERING & TECHNOLOGY  
II SHIFT POLYTECHNIC  
KONDAPUR (V), GHATKESAR (M), Medchal (D), Telangana, Accredited by  
AICTE, Affiliated to JNTUH/SBTET, Hyderabad.  
UGC AUTONOMOUS INSTITUTION (2022-2025)**

---



**SAMSKRUTI COLLEGE OF ENGINEERING & TECHNOLOGY  
II SHIFT POLYTECHNIC**

**KONDAPUR (V), GHATKESAR (M), Medchal (D), Telangana,  
Accredited by AICTE, Affiliated to JNTUH/SBTET, Hyderabad  
UGC AUTONOMOUS INSTITUTION**

**Department of Computer Engineering**



**CERTIFICATE**

This is to certify that the project report entitled "**SALINEIQ -AN IOT BASED SALINE MONITORING SYSTEM**" is being submitted by T.RITHVIKESH GOUD(22259-CS-017), G.VIKAS GOUD(22259-CS-026), M.RAKESH YADAV(22259-CS-033), M.NIHARIKA(22259-CS-057), K.AMRUTHA(22259-CS-038), B.SARVANI SANDHYA(22259-CS-043).

in partial fulfillment of the requirements for the award of DIPLOMA from SBTET Hyderabad in COMPUTER ENGINEERING. This record is a bonafide work carried out by them under my guidance and supervision. The results embodied in this report have not been submitted to any other university for the award of any degree.

**INTERNAL GUIDE**

Mrs. B. KOUSARI  
(MTech)

**HEAD OF SECTION**

Mr. J. VAMSHIKRISHNA  
(MTech)

**PROJECT COORDINATOR**

**Place:**

**EXTERNAL EXAMINER**

**Date:**



## ACKNOWLEDGEMENT

Firstly, we would thank GOD for having helped us through this work. We would like to express our sincere gratitude to our honorable principal **Mr. K. SHIVA KESAVA REDDY** for granting us permission to do the project in our college and for his encouragement.

We profoundly thank **Mr. S. VAMSHI KRISHNA**, Head of section of computer engineering, Samskruti College of Engineering and Technology II Shift Polytechnic for his help and encouragement.

We would like to express our deep sense of gratitude and whole-hearted thanks to our project guide. Mr. J. VAMSHI KRISHNA (MTech), Samskruti College of Engineering and Technology for giving me the privilege of working under the esteemed guidance in carrying out his project work.

Finally, we extend our gratefulness to one and all directly or indirectly involved in the successful completion of this project work.

With Sincere regards,

<b>T.RITHVIKESH GOUD</b>	<b>22259-CS-017</b>
<b>G.VIKAS GOUD</b>	<b>22259-CS-026</b>
<b>M.RAKESH YADAV</b>	<b>22259-CS-033</b>
<b>M.NIHARIKA</b>	<b>22259-CS-057</b>
<b>K.AMRUTHA</b>	<b>22259-CS-038</b>
<b>B.SRAVANI SANDHYA</b>	<b>22259-CS-043</b>



# INDEX PAGE:

1) Thesis.....	1-3
2) Abstract.....	6
3) Introduction.....	7-8
4) Objective.....	9-10
5) Components Used.....	11-16
6) ESP-32 Program.....	17-18
7) ESP-32 Code.....	19-27
8) Web Interface(HTML&CSS).....	28-30
9) Working Principle.....	31-33
10) Features Of SalineIQ.....	34-36



11)Output of SalineIQ.....	37-42
12)Challenges and Solutions.....	43-45
13)Future Enhancements.....	46-48
14)Conclusion.....	49
15)References.....	50-51



## ABSTRACT:

Salines play a critical role in the medical field. where precision and accuracy are important for using them, these are used for treatments, diagnosis. These are mostly used in Normal hospitals, clinics. these Salines are very concentrated,

To ensure the patient safety and to give them the right amount of saline As required hospitals many saline sensor devices, which can control the right amount of the saline solutions required for the patient and it also sends quick live messages to the doctor/nurse.

While addressing these things. we have an idea to develop a saline sensor project Which is cost efficient and it focuses on the flow of the saline, as required for the patient it also sends the live updates of the saline level.

This project uses hardware components like ARDUINO and sensors for real time data collections. for the software side this project mainly uses the web technologies like HTML ,CSS , and for the backend JAVA SCRIPT .it uses PHP(MY SQL) for data storage this project focuses on the control of the saline and providing the live updates to the doctors or nurses to make their work more easy and efficient.



## INTRODUCTION:

SalineIQ – Smart Saline Monitoring System is an innovative healthcare IoT solution developed to enhance the efficiency and safety of intravenous (IV) saline administration in clinical environments. In many hospitals and healthcare centers, the manual monitoring of saline bottles remains a time-consuming and error-prone task, often leading to delays in replacing empty bottles. This delay can cause air embolism or patient discomfort, posing serious health risks. To address this critical challenge, SalineIQ introduces an automated, real-time monitoring and alert system built around the robust ESP32 microcontroller.

The system leverages IoT (Internet of Things) principles to display real-time readings on a web-based dashboard, accessible via any browser on the same network. In addition to visual feedback, SalineIQ incorporates an alert mechanism using a buzzer that activates when certain sensor thresholds are crossed—such as low saline levels or bottle absence—thereby helping medical staff intervene before any harm can occur.

This innovative solution aims to reduce human error, prevent dry IV runs, and improve efficiency in medical care delivery by ensuring saline levels are always adequately monitored.

The SalineIQ system is equipped with a set of sensors including ultrasonic sensors to detect the fluid level in saline bottles, an IR sensor to confirm the presence of the bottle, a DHT11 sensor to monitor room temperature and humidity, and a load cell with HX711 amplifier to measure the weight of the saline bottle. These sensors work together to provide comprehensive data about both the saline solution and the environment in which it is administered.

SalineIQ is designed to be cost-effective, easy to deploy, and scalable for use

---



in hospitals, clinics, and even home healthcare setups. By reducing reliance on manual monitoring and enabling quick response through timely alerts, SalineIQ not only improves patient care but also reduces the workload on healthcare staff.

This documentation provides an in-depth overview of the system's objectives, hardware components, working principle, software implementation, and deployment procedure, offering a comprehensive guide for replicating or extending the project.



## OBJECTIVE:

The primary objective of SalineIQ is to develop a smart, accessible, and affordable solution for real-time saline monitoring in medical environments. Our initiative is driven by the goal of enhancing patient care and safety through automation, while also reducing the burden on healthcare workers who currently rely on manual observation of IV saline levels.

SalineIQ is designed with the following key goals in mind:

- 1) Cost-Effectiveness: By using affordable and widely available components like the ESP32 microcontroller, ultrasonic sensors, and other basic electronic modules, SalineIQ ensures that hospitals, clinics, and even home caregivers can implement the system without high financial investment. This makes it a viable option for both well-equipped medical centers and resource-limited healthcare settings.
- 2) User-Friendly Operation: One of the core principles behind SalineIQ is simplicity. The system is built to be intuitive and easy to use, even for individuals with minimal technical expertise. From setting up the hardware connections to accessing the web-based dashboard, every aspect of the system is designed to offer a seamless user experience.
- 3) Clear and Understandable Interface: The web interface presents real-time sensor data in a clean, readable format, ensuring that nurses, caregivers, or medical staff can quickly interpret the information without the need for technical training. Important alerts are visually and audibly communicated through the interface and buzzer system, eliminating ambiguity and allowing for swift decision-making.
- 4) Scalability and Accessibility: Our initiative supports scalability, enabling the system to be used in single-bed settings or across multiple saline stations



in larger facilities. With open-source code and standard components, SalineIQ can be customized or expanded based on specific institutional needs.

Through this initiative, we aim to democratize access to smart healthcare tools, ensuring that life-saving monitoring technologies are no longer limited to high-end hospitals but are made available to all, regardless of location or budget. SalineIQ stands as a step forward in combining healthcare with technology for a safer, smarter future.



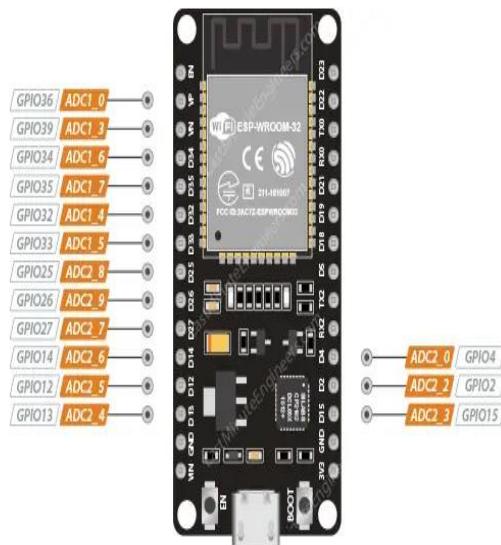
## COMPONENTS:

The development of SalineIQ – Smart Saline Monitoring System is based on the integration of both hardware components for real-world data sensing and web technologies for user-friendly visualization. Every component has been carefully selected to keep the system affordable, scalable, and easy to use, without compromising performance.

### A. Hardware Components:

#### -ESP32 Development Board

The central microcontroller used in this project. It is a powerful, dual-core Wi-Fi and Bluetooth-enabled board capable of handling multiple tasks simultaneously. It reads data from all sensors, processes the information, hosts a web server, and triggers alerts when needed. The ESP32 was chosen over other microcontrollers due to its high performance, built-in connectivity, and low cost.



#### -Ultrasonic Sensors (HC-SR04) x2

These sensors are used to measure the distance between the sensor and the saline fluid surface. Two sensors offer better accuracy and redundancy to

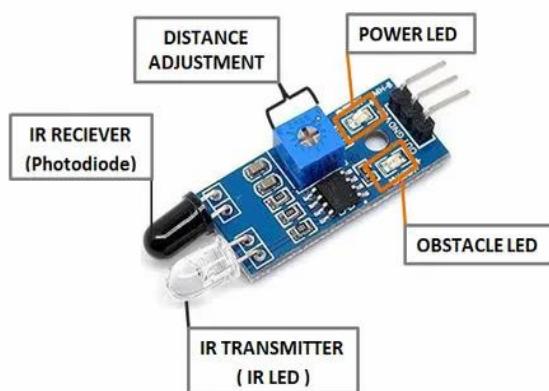


detect fluid level variations. By calculating the distance, the system estimates the volume of saline remaining in the bottle.



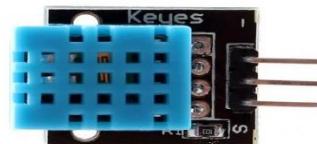
### -IR Sensor

The infrared sensor is used to detect the presence or absence of the saline bottle. If no object (bottle) is detected in the expected position, the system identifies this as a potential issue and raises an alert. It acts as a fail-safe in case the bottle is removed or misaligned.



### -DHT11 Temperature and Humidity Sensor

This sensor provides environmental data such as room temperature and humidity. Monitoring these values is important for ensuring the patient is in a controlled environment and that sensor accuracy isn't affected by extreme conditions.



### -Load Cell with HX711 Amplifier Module

The load cell is used to measure the weight of the saline bottle, offering a secondary method to confirm the saline level. The HX711 amplifier converts the weak analog signals from the load cell into digital data that the ESP32 can interpret. This redundancy helps improve the reliability of the system.



### -Buzzer

The buzzer serves as an immediate audio alert when the system detects critical conditions like a near-empty saline bottle or a missing bottle. It draws attention to the issue even when the caregiver is not actively monitoring the web dashboard.



### -Breadboard and Jumper Wires

These are used to connect all components together in a modular, solderless setup. The breadboard allows for quick prototyping, testing, and modifications during development.

### -USB Cable for ESP32

Provides power to the ESP32 and also allows for flashing the firmware from the development environment.

## B. Software & Web Technologies:

### -ESP32 Programming Tools (Non-Arduino Based)

The ESP32 in this project is programmed using ESP32-specific development tools, such as PlatformIO or other native ESP-IDF environments. These tools allow direct flashing of the firmware and provide greater control and flexibility than standard Arduino IDE-based workflows. The program is written in C++, utilizing libraries for Wi-Fi, sensors (DHT11, HX711), and asynchronous web server handling.



### -HTML (HyperText Markup Language)

Used to structure the web dashboard hosted by the ESP32. It displays real-time data such as fluid level, bottle presence, temperature, humidity, and weight. The HTML structure ensures that the interface is clean and intuitive for caregivers and medical staff to interpret quickly.

### -CSS (Cascading Style Sheets)

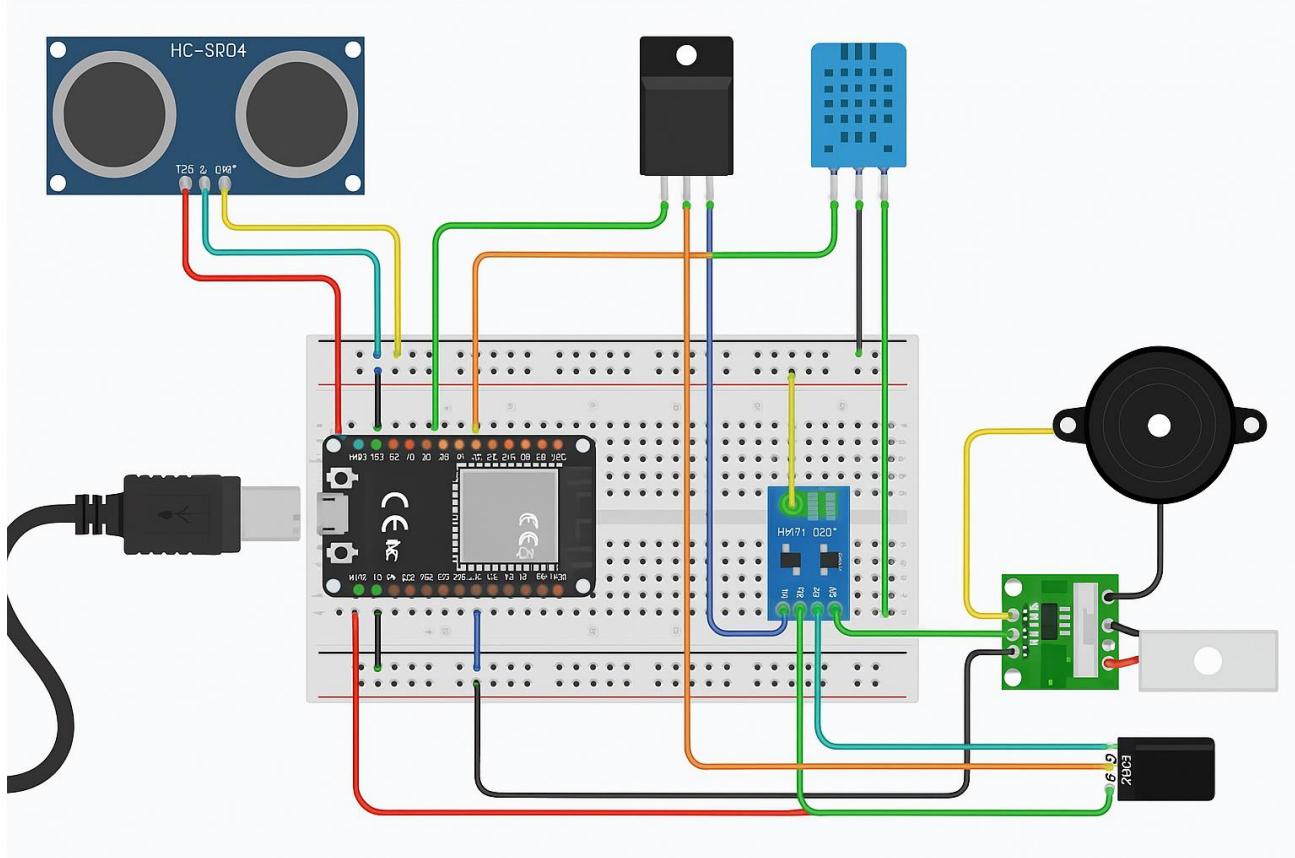
Provides styling to the HTML interface. It ensures that the web dashboard is not only functional but also visually appealing and readable. Colors, fonts, and layout are customized to suit low-light environments like hospital wards, making the interface user-friendly and accessible.

## PIN CONFIGURATION AND CIRCUIT DIAGRAM:

The circuit for SalineIQ involves connecting multiple sensors and modules to the ESP32 development board. Each sensor is assigned to a specific GPIO pin on the ESP32 to ensure accurate data collection and efficient communication. The wiring is done using a breadboard and jumper wires to allow flexibility during prototyping.

The first ultrasonic sensor is connected with its TRIG pin to GPIO 26 and ECHO pin to GPIO 25, while the second ultrasonic sensor uses GPIO 14 for TRIG and GPIO 27 for ECHO. The IR sensor is connected to GPIO 33, and the DHT11 temperature and humidity sensor is connected to GPIO 32. The buzzer used for audio alerts is attached to GPIO 15. For weight sensing, the HX711 load cell amplifier connects with its DOUT pin to GPIO 4 and SCK pin to GPIO 5. All components are powered and interfaced via a breadboard using jumper wires, ensuring a clean and organized circuit layout.





## ESP-32 PROGRAM -

The ESP32 program is the core of the SalineIQ system, handling everything from sensor integration and Wi-Fi connectivity to real-time data broadcasting on a web interface.

This code is written in C++ using the Arduino framework and leverages powerful libraries such as:

ESPAsyncWebServer and AsyncTCP for serving real-time web content efficiently.

DHT for reading temperature and humidity.

HX711 for converting analog weight data from the load cell.

ArduinoJson for creating structured JSON sensor payloads.

### Wi-Fi Connection:

The ESP32 connects to a local Wi-Fi network and starts a server accessible on the browser using its IP address.

### Sensor Initialization & Reading:

Ultrasonic Sensors (2): Measure saline level using TRIG/ECHO pins.

IR Sensor: Detects the presence or absence of the bottle.

DHT11 Sensor: Measures ambient temperature and humidity.

Load Cell with HX711: Measures bottle weight accurately.



**Real-Time Web Interface:**

The ESP32 serves a beautifully designed HTML page styled with CSS. Sensor data is updated in real time using Server-Sent Events (SSE), making it highly responsive and lightweight.

**Alert Mechanism:**

If any reading goes below preset thresholds (e.g., low distance or weight), a buzzer connected to the ESP32 sounds an alarm.

**Resilience Features:**

Wi-Fi auto-reconnection logic is implemented.

Timeout and noise filtering in sensors to ensure accuracy.



## Esp32 code:

```
#define CONFIG_ASYNC_TCP_RUNNING_CORE 1
#define ASYNC_TCP_PRIORITY 3
#define SERVER_TASK_PRIORITY 2
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <HX711.h>
#include <DHT.h>
#include <ArduinoJson.h>

// Wi-Fi credentials
const char* ssid = "ihithitler";
const char* password = "ihithitlerr";

// Pin Definitions
#define TRIG_PIN_1 26
#define ECHO_PIN_1 25
#define TRIG_PIN_2 14
#define ECHO_PIN_2 27
#define IR_PIN 33
#define DHT_PIN 32
#define BUZZER_PIN 15
#define LOADCELL_DOUT_PIN 4
#define LOADCELL_SCK_PIN 5

// Constants
#define DHT_TYPE DHT11 // Change to DHT22 if using that sensor
#define DISTANCE_THRESHOLD 50.0 // cm
#define WEIGHT_THRESHOLD 50.0 // grams
```



```
// Add these definitions at the top after other #defines
#define CONFIG_ASYNC_TCP_RUNNING_CORE 1
#define ASYNC_TCP_PRIORITY 3
#define SERVER_TASK_PRIORITY 2

// Global Variables
HX711 scale;
DHT dht(DHT_PIN, DHT_TYPE);
AsyncWebServer server(80);
AsyncEventSource events("/events");

// HTML Page
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Sensor Dashboard</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body {
      font-family: 'Poppins', sans-serif;
      text-align: center;
      margin: 0;
      padding: 0;
      background-color: #121212;
      color: #f1f1f1;
    }
    h1 {
      background: linear-gradient(135deg, #007bff, #00c6ff);
      color: white;
      padding: 20px;
      margin: 0;
      font-size: 24px;
    }
  </style>
</head>
<body>
  <h1>ESP32 Sensor Dashboard</h1>
  <p>Sensor Data will be displayed here</p>
</body>
</html>
)rawliteral";
```



```
text-transform: uppercase;
letter-spacing: 2px;
}
.container {
  max-width: 500px;
  margin: 30px auto;
  background: #1e1e1e;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.3);
}
.sensor {
  font-weight: bold;
  font-size: 24px;
  color: #00c6ff;
  display: inline-block;
  transition: transform 0.3s ease-in-out;
}
.sensor:hover {
  transform: scale(1.1);
  color: #ff4081;
}
p {
  font-size: 18px;
  margin: 15px 0;
  color: #bbb;
}
footer {
  margin-top: 20px;
  font-size: 14px;
  color: #888;
}
</style>
</head>
```



```
<body>
<h1>ESP32 Sensor Dashboard</h1>
<div class="container">
<p>Distance 1: <span id="distance1" class="sensor">--</span> cm</p>
<p>Distance 2: <span id="distance2" class="sensor">--</span> cm</p>
<p>IR Sensor: <span id="irDetected" class="sensor">--</span></p>
<p>Humidity: <span id="humidity" class="sensor">--</span> %</p>
<p>Temperature: <span id="temperature" class="sensor">--</span>
°C</p>
<p>Weight: <span id="weight" class="sensor">--</span> g</p>
</div>
<footer>&copy; 2025 Your Name</footer>
<script>
if (!!window.EventSource) {
  var source = new EventSource('/events');
  source.addEventListener('message', function(e) {
    var data = JSON.parse(e.data);
    document.getElementById('distance1').innerText = data.distance1;
    document.getElementById('distance2').innerText = data.distance2;
    document.getElementById('irDetected').innerText = data.irDetected ?
    "Object Detected" : "No Object";
    document.getElementById('humidity').innerText = data.humidity;
    document.getElementById('temperature').innerText =
    data.temperature;
    document.getElementById('weight').innerText = data.weight;
  }, false);
}
</script>
</body>
</html>
)rawliteral";
```

---

```
float measureDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
```



```
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

unsigned long timeout = 30000; // 30ms timeout
long duration = pulseIn(echoPin, HIGH, timeout);

if (duration == 0) {
    Serial.println("Distance measurement timeout");
    return 999.9;
}

float distance = duration * 0.034 / 2;
return (distance > 400.0) ? 999.9 : distance; // Filter invalid readings
}

void initWiFi() {
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.print("Connecting to WiFi");
    WiFi.begin(ssid, password);

    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 20) {
        delay(500);
        Serial.print(".");
        attempts++;
    }

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("\nConnected!");
    }
}
```



```
Serial.print("IP: ");
Serial.println(WiFi.localIP());
} else {
Serial.println("\nConnection failed - Restarting");
ESP.restart();
}
}

void setup() {
Serial.begin(115200);
delay(1000);

// Initialize pins first
pinMode(TRIG_PIN_1, OUTPUT);
pinMode(ECHO_PIN_1, INPUT);
pinMode(TRIG_PIN_2, OUTPUT);
pinMode(ECHO_PIN_2, INPUT);
pinMode(IR_PIN, INPUT);
pinMode(BUZZER_PIN, OUTPUT);
digitalWrite(BUZZER_PIN, LOW);

// Initialize sensors
dht.begin();

// Initialize scale
scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN);
if (scale.wait_ready_timeout(1000)) {
Serial.println("HX711 initialized");
scale.set_scale();
scale.tare();
} else {
Serial.println("HX711 not found!");
delay(1000);
ESP.restart();
```



```
}

// Initialize WiFi first, before web server
initWiFi();
delay(1000); // Give WiFi time to stabilize

// Initialize web server with protective delay
DefaultHeaders::Instance().addHeader("Access-Control-Allow-Origin",
" *");

// Server routes with error checking
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    if (request->client()->space() > 0) {
        request->send(200, "text/html", index_html);
    }
});

events.onConnect([](AsyncEventSourceClient *client) {
    if (client->lastId()) {
        Serial.printf("Client reconnected! Last ID: %u\n", client->lastId());
    }
});

server.addHandler(&events);

// Start server with protective delay
delay(100);
server.begin();
delay(100);

Serial.println("Server started");
}

void loop() {
```



```
static unsigned long lastUpdate = 0;
static unsigned long lastWiFiCheck = 0;
const unsigned long UPDATE_INTERVAL = 1000;
const unsigned long WIFI_CHECK_INTERVAL = 5000;

// Regular sensor updates
if (millis() - lastUpdate >= UPDATE_INTERVAL) {
    // Create JSON document with exact size calculation
    const size_t capacity = JSON_OBJECT_SIZE(6) + 100;
    StaticJsonDocument<capacity> doc;

    // Read sensors only when needed
    doc["distance1"] = measureDistance(TRIG_PIN_1, ECHO_PIN_1);
    doc["distance2"] = measureDistance(TRIG_PIN_2, ECHO_PIN_2);
    doc["irDetected"] = digitalRead(IR_PIN);

    float h = dht.readHumidity();
    float t = dht.readTemperature();
    doc["humidity"] = isnan(h) ? 0 : h;
    doc["temperature"] = isnan(t) ? 0 : t;

    float weight = 0;
    if (scale.wait_ready_timeout(100)) {
        weight = scale.get_units(3);
        if (isnan(weight) || weight < -999999 || weight > 999999) {
            weight = 0;
        }
    }
    doc["weight"] = weight;

    // Use char array instead of String for JSON
    char jsonBuffer[256];
    serializeJson(doc, jsonBuffer);
```



```
// Send only if WiFi is connected
if (WiFi.status() == WL_CONNECTED) {
    events.send(jsonBuffer, "message", millis());
    Serial.println(jsonBuffer);
}

// Threshold checks
bool alert = false;
float d1 = doc["distance1"].as<float>();
float d2 = doc["distance2"].as<float>();

if (d1 > 0 && d1 < DISTANCE_THRESHOLD) alert = true;
if (d2 > 0 && d2 < DISTANCE_THRESHOLD) alert = true;
if (weight > 0 && weight < WEIGHT_THRESHOLD) alert = true;

digitalWrite(BUZZER_PIN, alert ? HIGH : LOW);

lastUpdate = millis();
}

// WiFi check with more delay between reconnection attempts
if (millis() - lastWiFiCheck >= WIFI_CHECK_INTERVAL) {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi disconnected - Reconnecting");
        WiFi.disconnect(true);
        delay(1000);
        initWiFi();
    }
    lastWiFiCheck = millis();
}

// Increased delay to reduce CPU load
delay(20);
}
```



## WEB INTERFACE:

### 1) WEB INTERFACE:

The web interface of SalineIQ acts as a live dashboard that displays real-time sensor readings in an intuitive and user-friendly layout. It is hosted directly on the ESP32 microcontroller and accessed via a browser using the ESP32's IP address.

This dashboard is developed using HTML for structure and CSS for styling, making it lightweight, responsive, and easy to use on both desktop and mobile devices without any external dependencies.

#### Features of the Web Dashboard:

Live Sensor Data Visualization:

The dashboard displays real-time data for:

Distance from two ultrasonic sensors

IR bottle detection status

Temperature and humidity

Weight measured by the load cell

### 2) Dynamic Updates with SSE (Server-Sent Events):

Unlike traditional web refreshes, the dashboard uses SSE to push sensor updates automatically from the ESP32 to the browser — ensuring seamless and lag-free monitoring.

### 3) Modern, Dark-Themed UI:

A clean and minimal dark mode design enhances readability and provides a sleek look suitable for medical or professional environments.



#### 4)Interactive Styling:

Sensor readings respond to hover actions, making the interface more interactive and engaging.

Interface Snippet (HTML & CSS):

Html:

```
<h1>ESP32 Sensor Dashboard</h1>
<p>Distance 1: <span id="distance1" class="sensor">--</span> cm</p>
<p>Distance 2: <span id="distance2" class="sensor">--</span> cm</p>
<p>IR Sensor: <span id="irDetected" class="sensor">--</span></p>
<p>Humidity: <span id="humidity" class="sensor">--</span> %</p>
<p>Temperature: <span id="temperature" class="sensor">--</span>
°C</p>
<p>Weight: <span id="weight" class="sensor">--</span> g</p>
```

CSS:

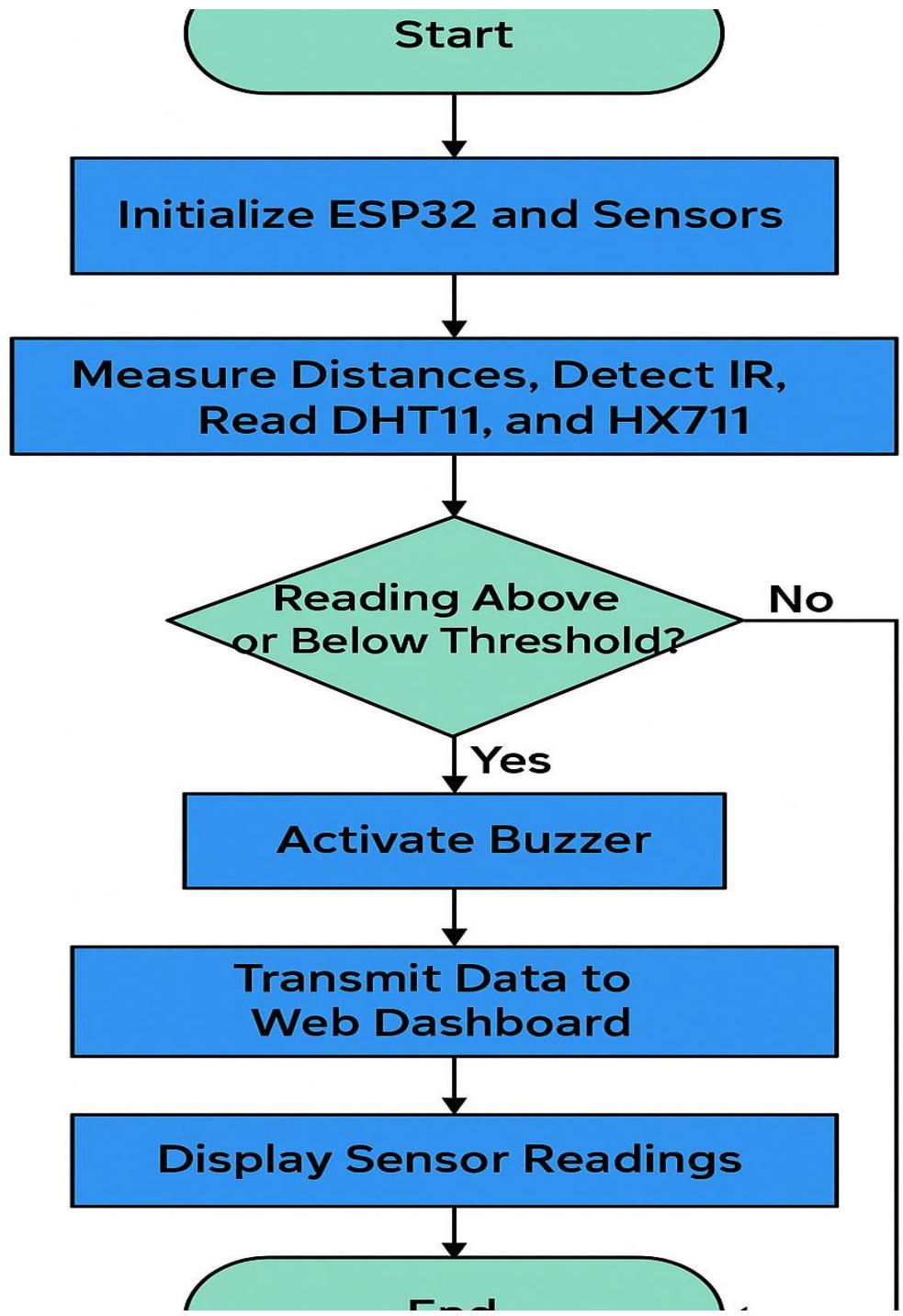
```
body {
  background-color: #121212;
  color: #f1f1f1;
  font-family: 'Poppins', sans-serif;
}
.sensor {
  font-weight: bold;
  font-size: 24px;
  color: #00c6ff;
```



```
transition: transform 0.3s ease-in-out;  
}  
.sensor:hover {  
    transform: scale(1.1);  
    color: #ff4081;  
}
```



## WORKING PRINCIPLE:



The SalineIQ system operates as an intelligent monitoring solution that continuously observes the level, volume, temperature, and presence of saline in an IV setup, alerting medical staff or caretakers if any irregularities are detected. The system integrates multiple sensors and real-time data communication to ensure efficient and automated saline tracking.

### How It Works:

#### -Ultrasonic Sensors for Saline Level Detection:

Two ultrasonic sensors are placed at different points near the IV fluid container.

They measure the distance between the sensor and the surface of the fluid.

Based on the measured distance, the system calculates whether the saline level is sufficient or running low.

#### -IR Sensor for Bottle Presence:

An infrared sensor detects whether the saline bottle is present in its designated place.

If the bottle is missing or not correctly placed, the system alerts the user.

#### -Load Cell for Weight Measurement:

A load cell sensor is used to determine the actual weight of the saline bottle.

This data acts as a secondary validation to confirm the volume of saline left, providing redundancy to the ultrasonic readings.



### -Temperature & Humidity Monitoring:

A DHT11 sensor records the ambient temperature and humidity to ensure the saline is stored and administered under ideal environmental conditions.

These values are displayed on the dashboard and stored for reference.

### -Real-Time Data Transmission:

All collected sensor data is processed by the ESP32 microcontroller.

It hosts a lightweight web server, which continuously streams data to a real-time web interface using Server-Sent Events (SSE).

### -Web Interface for Monitoring:

The dashboard shows live updates of all parameters in a visually intuitive format.

Medical personnel or caretakers can access this dashboard on any device within the same Wi-Fi network.

### -Automatic Alert System (Buzzer):

If:

Saline level drops below a threshold,

The weight falls too low,

The bottle is removed or missing,

Then the system activates the buzzer as an alert mechanism.



This ensures timely intervention and reduces the risk of air entering the IV line.

## **FEATURES:**

SalineIQ is designed to enhance patient care and reduce manual effort through intelligent automation. The following features make SalineIQ a powerful and practical solution for modern healthcare environments:

### **1. Real-Time Saline Level Monitoring**

Utilizes dual ultrasonic sensors to measure the fluid level in the IV bottle with high accuracy.

Continuously checks for any drop in fluid level and updates instantly.

### **2. Weight-Based Validation**

A load cell sensor adds another layer of precision by measuring the weight of the saline bottle.

Confirms the volume remaining, enhancing reliability and reducing false alerts.

### **3. Temperature and Humidity Sensing**

A built-in DHT11 sensor tracks the environment around the IV setup.

Ensures the saline is stored under appropriate conditions, especially useful in sensitive scenarios like neonatal or critical care.

### **4. IR-Based Bottle Detection**

An IR sensor monitors whether the saline bottle is correctly placed.

Sends alerts if the bottle is missing or misplaced.



## 5. Instant Alert Mechanism

An integrated buzzer provides audio alerts if any parameter crosses the safety threshold (e.g., low level, low weight, no bottle).

Ensures rapid attention by the caregiver or nurse.

## 6. Interactive Web Dashboard

A clean, responsive web interface built using HTML and CSS.

Displays all sensor data in real time using Server-Sent Events (SSE).

Can be accessed wirelessly from smartphones, tablets, or computers connected to the same Wi-Fi.

## 7. Wireless Connectivity

The system uses ESP32's built-in Wi-Fi capability to operate without physical connections to display or control units.

Makes the setup compact, mobile, and easy to install in any healthcare environment.

## 8. Cost-Effective and Scalable

Built with affordable components without compromising on accuracy or functionality.

Scalable design allows it to be deployed across multiple beds in a hospital or adapted for home use.

## 9. User-Friendly Setup

Simple wiring and minimal coding make it beginner-friendly for students, hobbyists, and professionals.

Clear dashboard design ensures ease of use for non-technical caregivers as well.



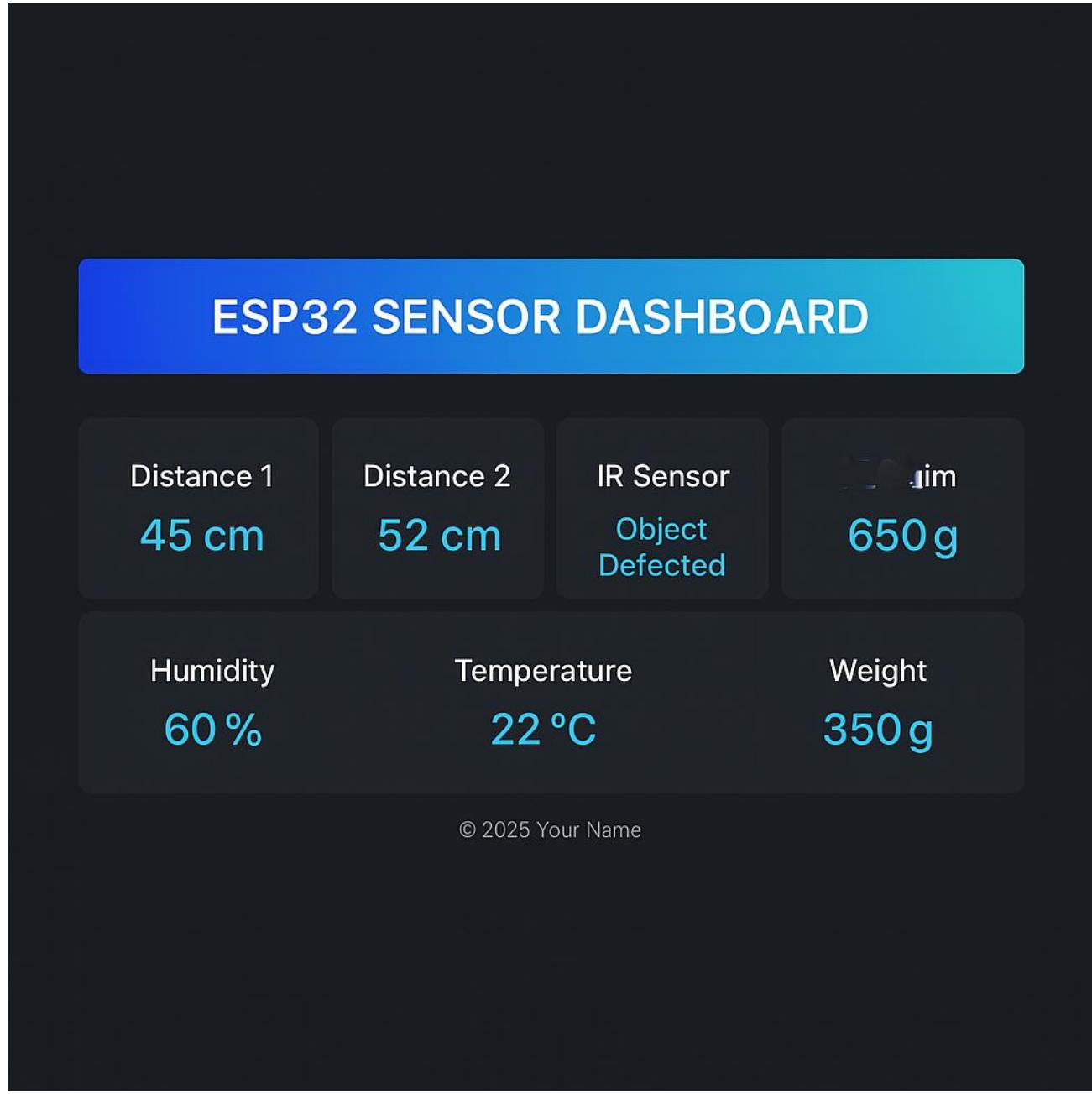
## 10. Fail-Safe Design

Includes redundancy in readings (ultrasonic + weight) to reduce risk of error.

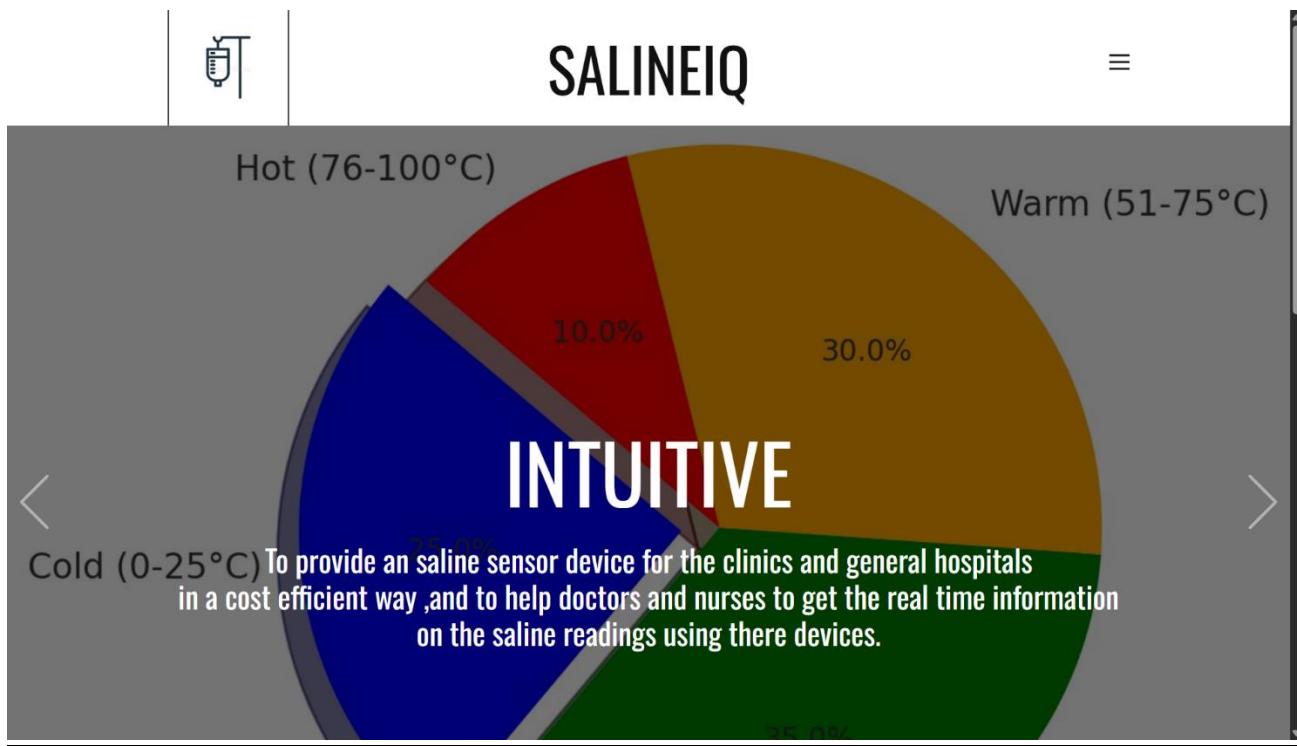
Wi-Fi reconnect and watchdog logic in software ensures system stays online and functional.



## OUTPUTS OF SALINE IQ PROJECT AND WEBPAGE:



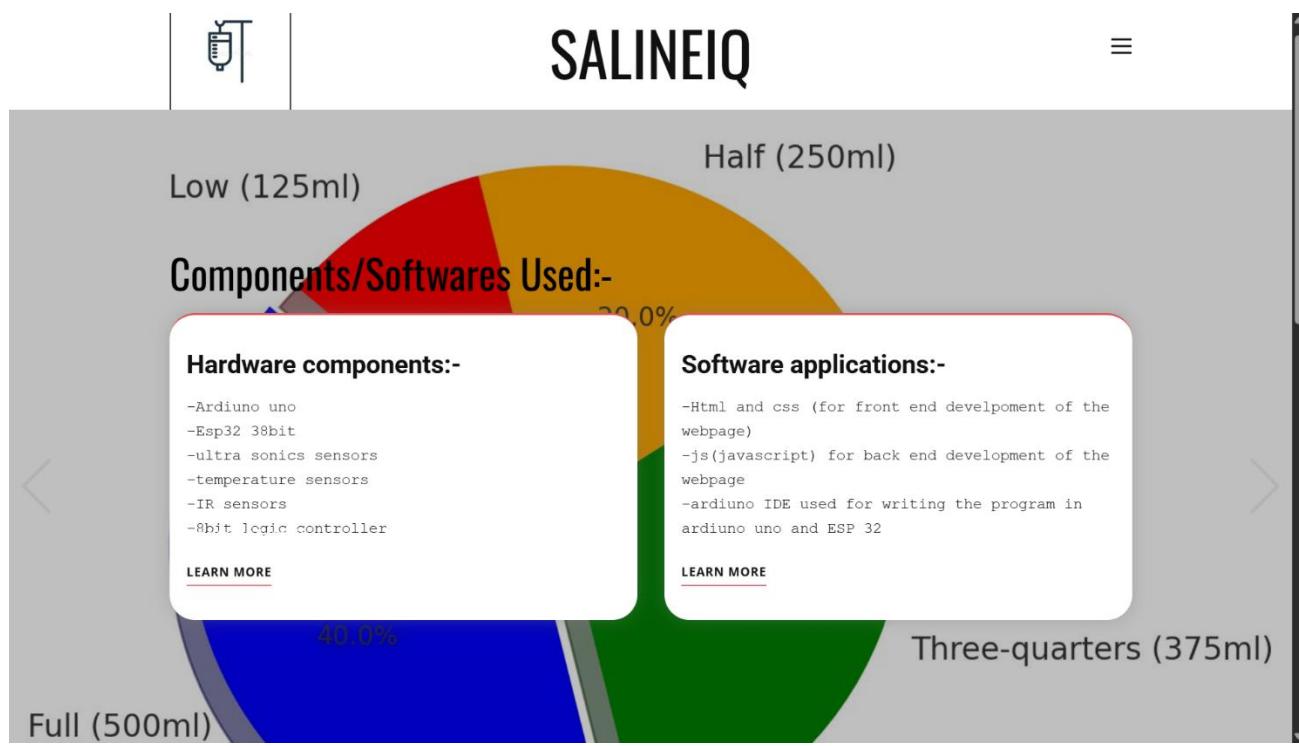
This image showcases the **ESP32 Sensor Dashboard**, a sleek and user-friendly web interface that displays real-time data from multiple sensors connected to the ESP32 microcontroller. The six key sensor readings—**Distance 1, Distance 2, IR Detection, Humidity, Temperature, and Weight**—are neatly aligned in two horizontal rows, enhancing readability and visual appeal. The dashboard reflects live values like 45.6 cm, 39.2 cm, “Object Detected”, 65%, 27.4°C, and 120.5g, simulating real-time monitoring. The dark-themed layout, paired with gradient highlights and responsive hover effects, offers a modern and professional appearance. This interface helps medical staff or technicians keep track of fluid levels, weight, and environmental conditions effortlessly.



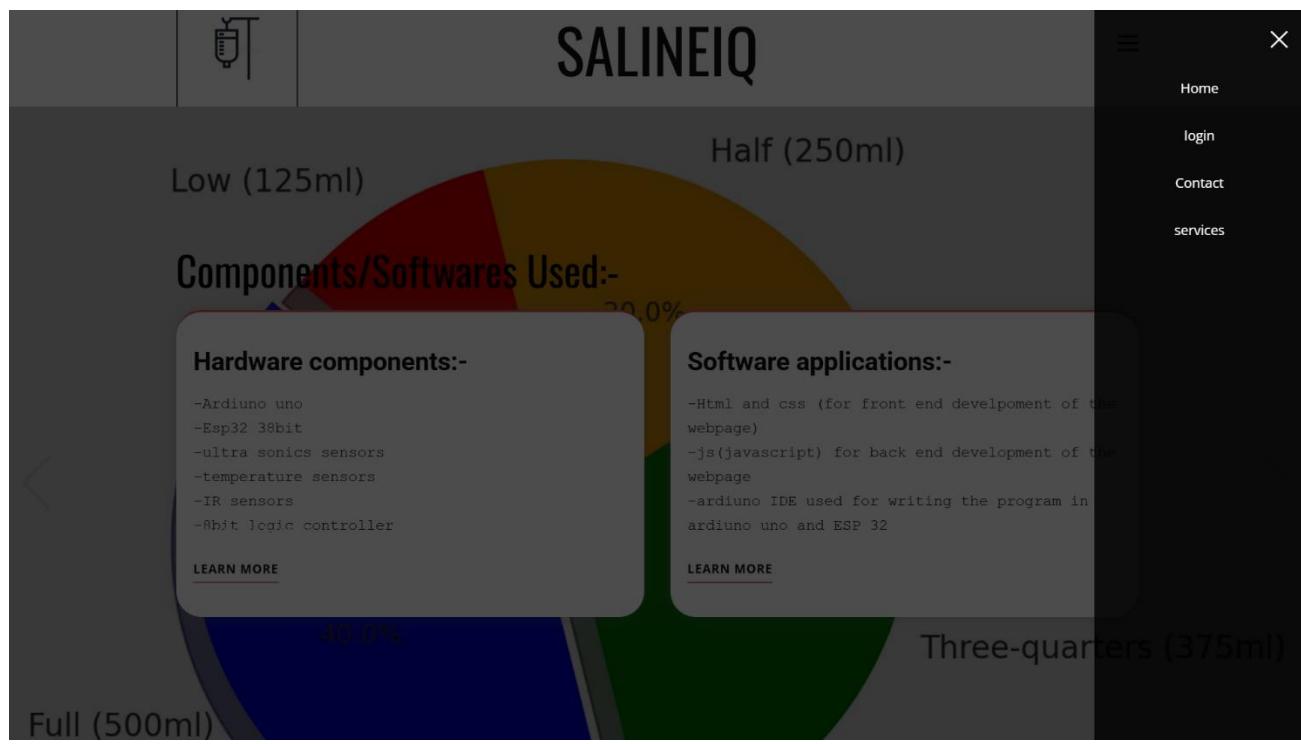
**SALINEIQ** is a smart and intuitive system designed to monitor saline levels and temperatures in medical environments. It offers a cost-efficient solution for clinics and hospitals, improving real-time tracking of IV fluids.



Using Arduino Uno, ESP32, ultrasonic and IR sensors, it accurately detects saline quantity and conditions. Temperature sensors ensure patient safety by monitoring fluid warmth, from cold to hot ranges. The software, built with HTML, CSS, and JavaScript, allows seamless interaction and live updates. The Arduino IDE manages device programming, enabling efficient data transfer and control. A built-in contact form supports user feedback, technical queries, and system-related assistance. Overall, SALINEIQ enhances healthcare efficiency by keeping doctors and nurses informed through smart devices.



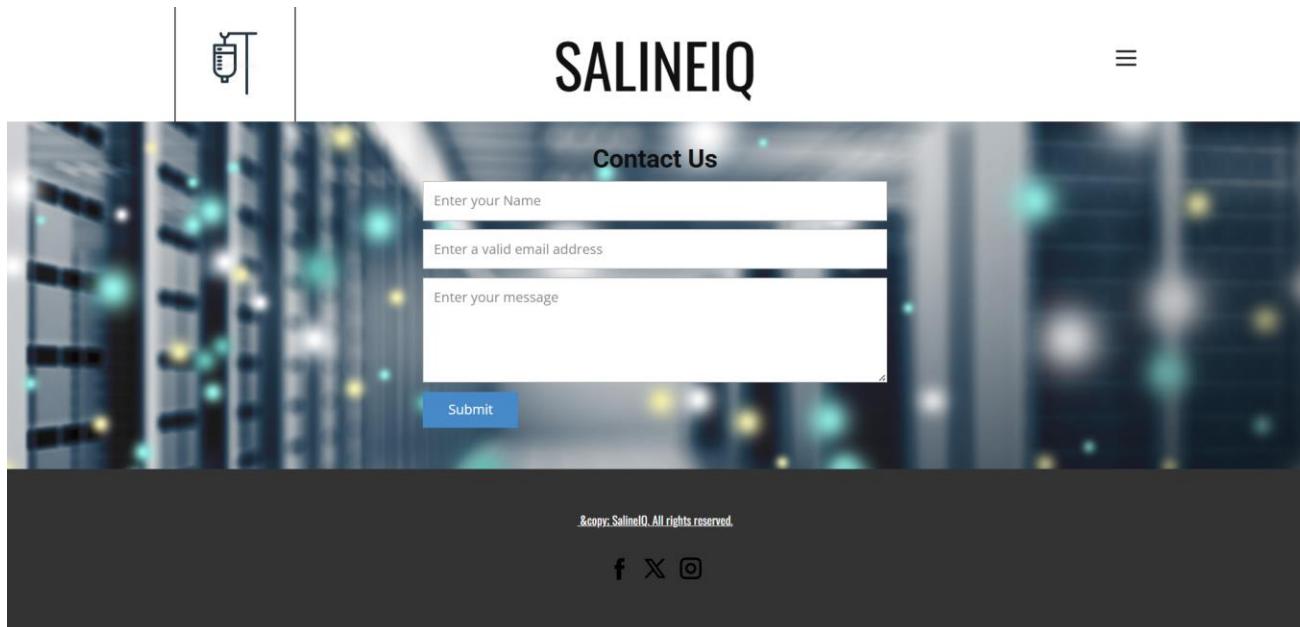
SalineIQ is designed to offer a cost-effective and efficient saline sensor device for clinics and hospitals. The goal is to assist doctors and nurses in monitoring real-time saline levels and temperatures directly from their devices, ensuring timely intervention and better patient care. The hardware setup includes components like Arduino Uno, ESP32, ultrasonic sensors, temperature sensors, IR sensors, and an 8-bit LCD controller. On the software side, the system uses HTML and CSS for frontend development, JavaScript for backend logic, and the Arduino IDE to program the microcontrollers.



SALINEIQ is an innovative smart system designed to monitor and manage the flow and level of saline in medical environments, using a well-integrated mix of both hardware and software components. On the hardware front, the system is equipped with an Arduino Uno and an ESP32 38-bit



microcontroller, which serve as the processing units. It incorporates ultrasonic sensors to measure the level of saline in the IV bag, temperature sensors to monitor surrounding conditions, and IR sensors for precise detection and automation. An 8-bit logic controller is used to coordinate the functioning of these components efficiently. Together, these hardware elements form a robust setup capable of delivering real-time, accurate data to ensure the timely refilling or replacement of saline bags, thereby reducing the risk of interruptions during patient care.



The software architecture of SALINEIQ complements its hardware by providing an interactive and functional user interface. The front-end is developed using HTML and CSS, offering a clean, responsive layout for users to interact with the system. JavaScript is employed to handle the backend logic, ensuring smooth communication between the web interface and the hardware modules. Programming for the Arduino Uno and ESP32 is done through the Arduino IDE, which allows precise control over data



collection and processing. In addition to the core monitoring features, SALINEIQ includes a dedicated contact section, allowing users to reach out for support, suggestions, or inquiries. This form collects the user's name, email address, and message, and is designed for efficient feedback management and troubleshooting assistance. The inclusion of social media icons below the form further enhances user engagement and support accessibility, making SALINEIQ not just a technical tool but a user-friendly service platform as well.

These are some outputs of our webpage and salineIQ website.



## CHALLENGES AND SOLUTIONS:

### 1. Sensor Inaccuracy and Fluctuation Challenge:

The ultrasonic and weight sensors occasionally gave fluctuating or inaccurate readings due to noise, ambient interference, or improper mounting.

Solution:

We implemented:

Software-based filtering and thresholding to ignore unrealistic spikes.

Stabilized sensor mounting using foam padding and clamps to reduce vibration.

Averaged multiple readings to improve consistency.

### 2. Unstable Wi-Fi Connection Challenge:

The ESP32 would sometimes disconnect from Wi-Fi, especially during long hours of operation.

Solution:

Implemented periodic Wi-Fi health checks in the loop function.

If disconnected, the ESP32 automatically reconnects to the Wi-Fi or restarts.



Delays were optimized to avoid overloading the ESP32 core.

### 3. False Buzzer Triggers Challenge:

The buzzer would sometimes go off due to minor or temporary sensor glitches.

Solution:

Added minimum threshold time before triggering the buzzer.

Logic was enhanced to check multiple sensors before initiating an alert.

Ensured the buzzer only activates when conditions are consistently below set limits.

### 4. Complexity of Sensor Integration Challenge:

Integrating multiple sensors like HX711, DHT11, IR, and ultrasonic sensors with one microcontroller caused coding and timing issues.

Solution:

Used non-blocking code and optimized delays to ensure smooth readings.

Prioritized sensor readings and modularized the code for better manageability.

Utilized real-time JSON updates to the web interface without blocking the main loop.

### 5. Web Interface Responsiveness Challenge:

The dashboard sometimes lagged or failed to update live values consistently.



Solution:

Utilized Server-Sent Events (SSE) for real-time updates with low overhead.

Optimized front-end code with lightweight HTML/CSS.

Ensured all sensor data is parsed as JSON, making it easy to display and manipulate.

#### 6. Power and Deployment Constraints Challenge:

Continuous power supply and safe deployment of open breadboard setup in a real-world environment was a concern.

Solution:

Recommended using a micro-USB power bank or adapter for stable supply.

Suggested housing the electronics in a 3D printed or plastic enclosure for safety and durability.



## FUTURE ENHANCEMENTS:

To ensure SalineIQ evolves into an even more effective and scalable solution, several future improvements have been identified. These enhancements aim to improve usability, reliability, scalability, and integration for broader real-world applications.

### 1. Mobile App Integration-

**Goal:** Develop a dedicated mobile application (Android/iOS) to allow real-time saline monitoring and alerts directly on a smartphone.

**Benefits:**

Improved accessibility for caregivers and hospital staff.

Push notifications for critical conditions (e.g., low saline level, disconnection).

### 2. Cloud Data Storage & Analytics-

**Goal:** Connect the system to a cloud platform (like Firebase, AWS, or ThingSpeak) for remote data storage and analytics.

**Benefits:**

Historical data tracking and visualization.

Advanced analytics for predicting patient needs and usage trends.

### 3. AI-Based Predictive Alerts-

**Goal:** Implement machine learning algorithms to predict saline depletion or



abnormal patterns.

Benefits:

Reduce response time.

Prevent overflows or dry saline conditions with smarter insights.

 4. Battery Backup and Portability-

Goal: Integrate a rechargeable battery module or solar-powered solution for uninterrupted operation.

Benefits:

Ensures functionality during power outages.

Enables use in rural or off-grid medical camps.

 5. Compact PCB Design-

Goal: Replace breadboard-based setup with a custom Printed Circuit Board (PCB) for permanent installation.

Benefits:

Reduces size and increases reliability.

Makes the device commercially viable and production-ready.

 6. Role-Based Access and Security-

Goal: Add login-based access control for hospital admin, doctors, and staff.

Benefits:



Protects patient data.

Enables custom views or permissions based on roles.

#### 🌐 7. Multi-Patient Support-

Goal: Extend the system to monitor multiple saline bottles in parallel, each with unique dashboards.

Benefits:

Ideal for ICU or ward-level deployment.

Central monitoring system for hospital staff.

#### 📢 8. Voice Alert System-

Goal: Integrate voice module to give verbal alerts when attention is needed.

Benefits:

Useful in noisy environments or for visually impaired caregivers.



## **CONCLUSION:**

The SalineIQ system represents a practical and innovative solution to a critical healthcare challenge—monitoring saline levels in real time to ensure timely intervention and prevent potential medical risks. Through the integration of ESP32, multiple sensors, and a user-friendly web interface, this project not only automates the saline monitoring process but also provides a cost-effective and efficient alternative to traditional manual checking.

By leveraging Internet of Things (IoT) technologies, SalineIQ successfully bridges the gap between modern technology and healthcare needs. It ensures better patient care, reduces the burden on hospital staff, and minimizes human errors.

From concept to deployment, the project demonstrates how smart systems can be designed using simple components, programmed logically, and deployed with minimal infrastructure. Its modular design, ease of use, and scalability make it suitable for both small clinics and large hospitals.

Looking ahead, with enhancements like mobile apps, cloud integration, and AI-based analytics, SalineIQ has the potential to become a comprehensive hospital automation tool. This project stands as a testament to how innovation and empathy can go hand in hand to solve real-world problems.



## REFERENCES:

### 1. **ESPAsyncWebServer Library**

<https://github.com/me-no-dev/ESPAsyncWebServer>

– Used for setting up the non-blocking web server on ESP32.

### 2. **AsyncTCP Library**

<https://github.com/me-no-dev/AsyncTCP>

– Required dependency for ESPAsyncWebServer to handle TCP connections asynchronously.

### 3. **ArduinoJson Library**

<https://arduinojson.org/>

– Used for creating and parsing JSON data for real-time updates.

### 4. **DHT-11Sensor Library**

<https://github.com/adafruit/DHT-sensor-library>

– For interfacing the DHT11 sensor to read temperature and humidity.

### 5. **HX711 Load Cell Library**

<https://github.com/bogde/HX711>

– Library for interfacing the load cell weight sensor via HX711.

### 6. **ESP32 Documentation**

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>

– Official technical documentation for the ESP32 microcontroller.



**7. HTML & CSS Styling Guides**

<https://developer.mozilla.org/en-US/docs/Web/HTML>

<https://developer.mozilla.org/en-US/docs/Web/CSS>

– For creating the responsive and styled web interface.

**8. ChatGPT**

– For generating explanations, helping structure documentation, writing code snippets, and making the whole process a bit easier (and cooler).

