University of Moratuwa

Department of Electronic and Telecommunication Engineering

FINAL REPORT

Smart Water Tank

Internet of Things

Supervisors – Prof. (.Mrs) Dileeka Dias

Sairisan R. 200552V

Malanban K. 200373X

Mathusha S. 200611D

Team Echo

Due Date – 2023.23.27

Table of Contents

1. Abstract

The Smart Water Tank IoT project introduces an innovative system designed to revolutionize the monitoring and management of water storage. With the growing concern over water scarcity and resource management, this project aims to address these challenges by implementing an Internet of Things (IoT) framework.

The system integrates sensor technology and IoT principles to enable real-time monitoring of water levels, temperature, and usage within storage tanks. Utilizing ultrasonic sensors, the system accurately measures water levels, DS18B20 temperature sensor, the system accurately measures water temperature and transmits those data to a centralized platform via wireless communication protocols such as Wi-Fi & MQTT.

Through a user-friendly interface, users can access comprehensive data analytics, including consumption patterns, leak detection, and predictive maintenance insights. The platform provides actionable information to optimize water usage, prevent wastage, and streamline operational efficiency.

This project represents a critical step towards sustainable water management, offering a scalable, cost-effective solution to monitor and conserve water resources effectively.

2. Introduction of the Project

2.1 Description of the Project

The implementation of an IoT-based Smart Water Tank system using an ESP32 microcontroller demonstrates a sophisticated approach to monitoring and managing water storage. This project utilizes various sensors, including ultrasonic, temperature, and flow rate sensors, combined with MQTT (Message Queuing Telemetry Transport) communication protocol, to create an efficient and intelligent system.

The system operates on a Wi-Fi network, enabling seamless connectivity and data transmission to a designated MQTT broker, in this case, "test.mosquitto.org." The MQTT broker facilitates communication between the ESP32 device and other subscribing clients.

- **Sensors Integration:** The system incorporates multiple sensors for comprehensive monitoring. Ultrasonic sensors accurately measure water levels, while temperature sensors provide ambient temperature data. Additionally, flow rate sensors enable the system to measure water flow in real-time.

- **Data Transmission:** Collected sensor data is serialized into JSON format and published periodically to an MQTT topic ("echo_pro_pub"). This data includes temperature readings, water level calculations, flow rate measurements, and the status of the motor controlling the water flow.

- **MQTT Communication:** The system is designed to subscribe to a specific MQTT topic ("echo_pro_sub/1") to receive incoming messages. These messages instruct the system to turn the water motor on or off based on predetermined thresholds and conditions.

- **Control Mechanism:** The ESP32 microcontroller regulates the water motor's operation based on received MQTT messages and predefined conditions. It utilizes these instructions to manage the motor's status, ensuring optimal water levels within the tank.

The project aims to enhance water storage management by providing real-time data monitoring, automated control based on predefined thresholds, and efficient utilization of resources. This IoT-based system offers a promising solution for effective water conservation and management in various contexts.

2.2 Motivation

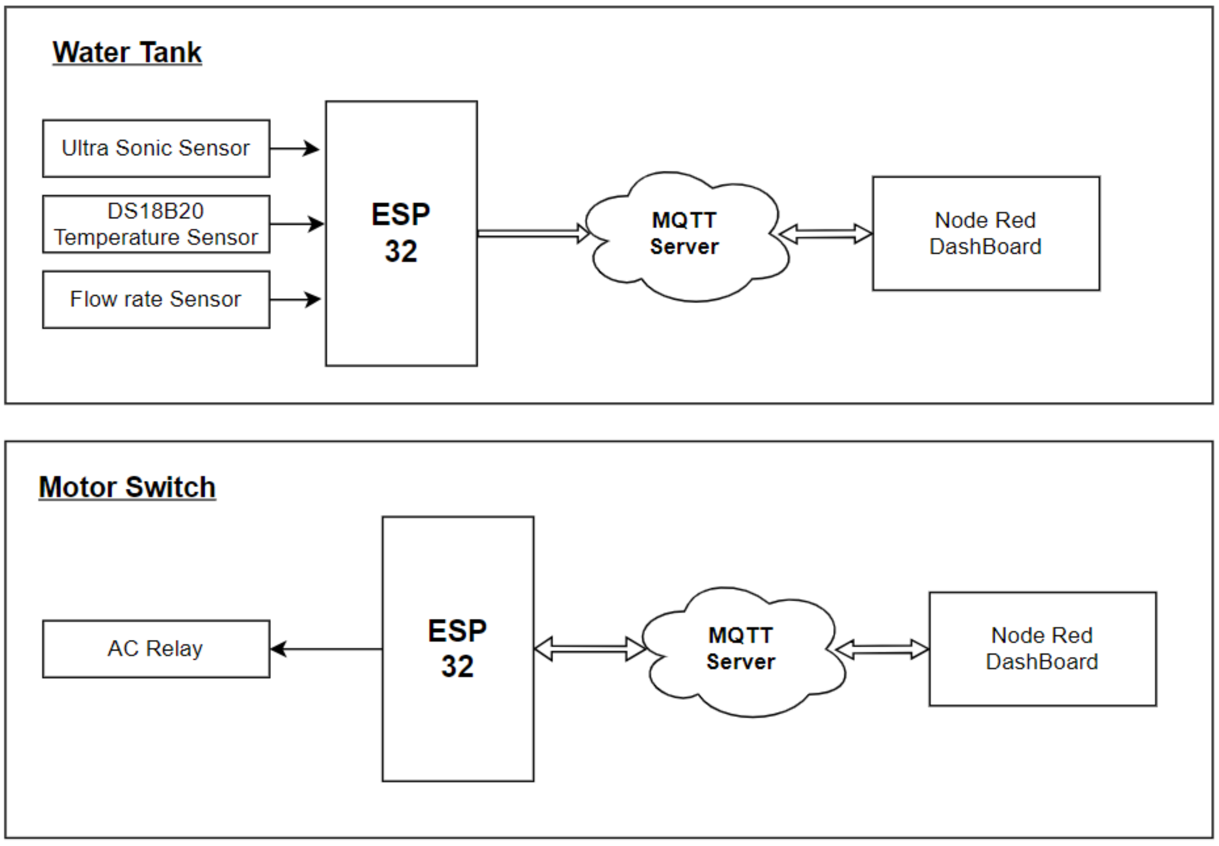
1. **Water Resource Conservation:** The project addresses the critical need for efficient water resource management. By implementing IoT technology to monitor water levels and optimize usage, it directly contributes to conserving this invaluable resource.
2. **Environmental Impact:** Efficient water management has a profound impact on the environment. The project's focus on reducing wastage and ensuring optimal utilization aligns with sustainability goals, positively impacting ecosystems and local communities.
3. **Technological Innovation:** Leveraging IoT and sensor technologies to create a smart water tank system showcases innovation in the realm of traditional infrastructure. It introduces a modern solution to age-old challenges in water storage and distribution.
4. **Real-Time Monitoring and Control:** Providing real-time data and control capabilities offers a significant advantage. It empowers users to make informed decisions promptly, preventing potential water shortages or overflows.
5. **Cost Efficiency:** The project's efficiency in managing water resources can translate into cost savings for individuals, industries, or communities. By minimizing water wastage, it reduces operational expenses and promotes cost-effective practices.
6. **Scalability and Adaptability:** The scalability of the system allows it to be tailored to various contexts, from individual households to large-scale water storage facilities. This adaptability ensures its applicability across diverse settings.
7. **Mitigating Water Scarcity:** In regions facing water scarcity or unreliable water supply, the IoT-based system can act as a proactive measure. It helps mitigate the impact of shortages by ensuring optimal use of available water resources.
8. **Educational and Awareness Building:** Implementing such projects also serves an educational purpose. It raises awareness about water conservation and the role of technology in addressing global challenges.
9. **Community Engagement:** Projects like these often involve communities and stakeholders, fostering a sense of involvement and responsibility toward sustainable resource management.

10. **Potential for Expansion:** The foundational technology and principles behind this project pave the way for further advancements. It opens doors for future innovations in smart infrastructure and IoT-based solutions for resource management.

Highlighting these motivational aspects can underscore the project's significance in addressing pressing global issues, demonstrating the power of technology in tackling environmental challenges, and contributing to a sustainable future.

3. Project Architecture

3.1 Overview of the Project Architecture



1. **Hardware Components:**
    - **ESP32 Microcontroller:** Acts as the central processing unit interfacing with various sensors and controlling the water motor.
    - **Sensors (Ultrasonic, Temperature, Flow Rate):** Collect data on water levels, temperature, and flow rates within the tank.
    - **Relay:** Controls the water motor based on received instructions and predefined conditions.
2. **Software Components:**
    - **Arduino IDE:** Used for programming the ESP32 microcontroller to read sensor data, communicate via MQTT, and control the water motor.
    - **Node-RED:** A visual programming tool used to create the user dashboard and handle MQTT communications.
3. **Communication Protocols:**
    - **Wi-Fi:** Enables the ESP32 to connect to the local network for internet connectivity.
    - **MQTT (Message Queuing Telemetry Transport):** Facilitates communication between the ESP32 and the Node-RED dashboard, allowing for message exchange and control commands.
4. **Data Flow:**
    - **Sensor Data Collection:** Ultrasonic, temperature, and flow rate sensors collect real-time data regarding water levels, temperature, and flow rates.
    - **ESP32 Processing:** The ESP32 processes this data, periodically publishing it in JSON format to an MQTT topic.
    - **MQTT Broker:** Acts as an intermediary, receiving and forwarding messages between the ESP32 and Node-RED.
    - **Node-RED Dashboard:** Receives MQTT messages from the ESP32, processes the data, and displays it in a user-friendly dashboard interface.

3.2 Components

- **ESP32 Micro-Controller**

The ESP32 is a versatile microcontroller and system-on-chip (SoC) manufactured by Espressif Systems. It's known for its powerful features, low cost, and widespread use in IoT, robotics, and embedded systems development.

Key Features:
Dual-Core Processor: The ESP32 features a dual-core Tensilica Xtensa LX6 processor, which allows for multitasking and efficient handling of multiple processes.

Connectivity Options: It supports various wireless connectivity options, including Wi-Fi (802.11 b/g/n) and Bluetooth (Bluetooth Classic and Bluetooth Low Energy or BLE). This dual-mode capability is one of its distinguishing features.

Rich Peripheral Interface: The ESP32 offers a wide range of peripheral interfaces, including SPI, I2C, UART, CAN, ADC, DAC, PWM, and more, making it suitable for interfacing with a variety of sensors, displays, and other devices.

Low Power Consumption: It has several power-saving modes and optimization features that make it suitable for battery-powered applications, extending the device's battery life.

- **Ultrasonic Sensor**

The HC-SR04 is a popular ultrasonic distance sensor widely used in robotics and automation projects. It works by emitting ultrasonic waves and then measuring the time it takes for those waves to bounce off an object and return to the sensor. This time measurement is then used to calculate the distance between the sensor and the object.

Key Features:

1. **Working Principle:** The sensor consists of an ultrasonic transmitter (which emits pulses) and a receiver (which detects the reflected signals). It operates based on the echo time between transmission and reception of ultrasonic pulses.
2. **Distance Measurement:** It measures distance by calculating the time interval between sending the ultrasonic signal and receiving the echo.
3. **Specifications:** The HC-SR04 typically has a range of 2cm to 400cm (or about 1 inch to 13 feet), with an accuracy of around 3mm.
4. **Operation:** To use the sensor, a trigger signal is sent to initiate the transmission of ultrasonic waves. When the waves hit an object, they bounce back to the sensor, and the sensor measures the time it took for the waves to return.
5. **Interface:** It often requires two pins for interfacing with microcontrollers:
    - Trigger Pin: To send the signal to start the measurement.
    - Echo Pin: To receive the signal and measure the time for the waves to return.
6. **Power Requirements:** It usually operates at 5V DC and consumes low power during operation.

- **<u>DS18B20 Temperature Sensor</u>**

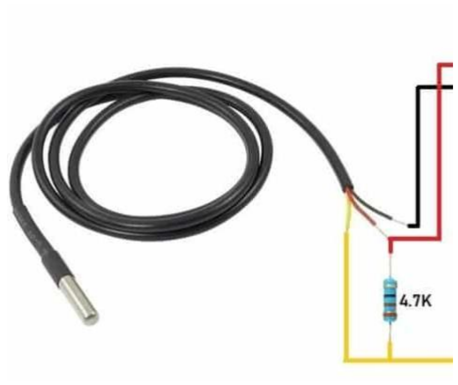
The DS18B20 is a digital temperature sensor known for its accuracy and simplicity in measuring temperature. It uses the 1-Wire communication protocol, enabling multiple sensors to communicate using only one data line.

Key Features:

1. **Digital Sensor:** The DS18B20 is a digital sensor, meaning it converts temperature readings into digital values, making it easy to interface with microcontrollers and other digital devices.

2. **Temperature Range:** It can measure temperatures ranging from -55°C to +125°C with a typical accuracy of ±0.5°C.
3. **Unique Addressing:** Each DS18B20 sensor has a unique 64-bit ROM code, allowing multiple sensors to be connected to the same data line without conflict. This feature enables easy identification of individual sensors on the same bus.
4. **Simple Interface:** It communicates using the 1-Wire protocol, which requires only a single data pin for communication with the microcontroller.
5. **Power Efficiency:** The sensor can operate in a low-power mode, consuming minimal power, which makes it suitable for battery-powered applications.
6. **Waterproof Variants:** Some versions of the DS18B20 come in waterproof housings, allowing them to be used in moisture-prone or submerged environments.

Connection Circuit of DS18B20 Temperature Sensor



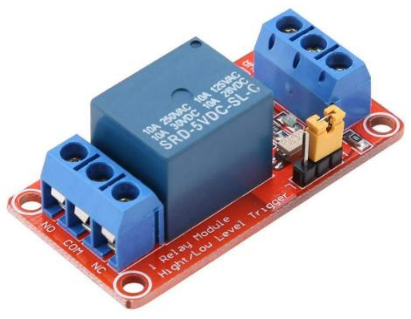
- **Flowrate Sensor**

Flow rate sensors, also known as flow meters, measure the rate of flow of a liquid or gas through a specific point in a system. These sensors are crucial in various industries for monitoring and controlling the flow of fluids. There are several types of flow rate sensors, each with its own principles of operation and applications:

Some common Types of Flow Rate Sensors:

1. **Differential Pressure Flow Meters:** These work by measuring the pressure drop across a constriction in a pipe. Examples include Orifice Plate, Venturi, and Pitot Tube flow meters.
2. **Ultrasonic Flow Meters:** Utilize ultrasonic waves to measure the velocity of the fluid. There are transit-time and Doppler ultrasonic flow meters.
3. **Electromagnetic Flow Meters:** Use Faraday's law of electromagnetic induction to measure the flow rate of conductive fluids.
4. **Vortex Flow Meters:** Measure flow by detecting the frequency of vortices formed as the fluid passes by a bluff body.
5. **Turbine Flow Meters:** These have a turbine that rotates in response to fluid flow. The rotational speed is proportional to the flow rate.
6. **Mass Flow Meters:** Measure the mass flow rate directly using techniques such as thermal, Coriolis, or calorimetric methods.

- <u>**AC Relay**</u>



Using an optocoupler (also known as an opto-isolator) with an AC relay is a common practice in electronics for safely interfacing low-voltage digital circuits with higher-voltage AC loads. The optocoupler provides electrical isolation between the low-voltage control circuit and the high-voltage AC side.

Here’s a basic setup using an optocoupler to control an AC relay:
Components Needed:

1. **Optocoupler (like a 4N25, 4N35, etc.):** This consists of an LED and a photodetector sealed in a light-proof case.
2. **Transistor (NPN or Darlington pair):** Often used to amplify the current from the optocoupler.
3. **Resistor(s):** Used to limit current to the LED side of the optocoupler and sometimes for base current in the transistor.

4. **AC Relay:** Choose a relay suitable for your voltage and current requirements.
5. **Diode:** A flyback diode (e.g., a freewheeling diode) is used to protect the circuit from voltage spikes when the relay is switched off.

3.3 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight, open, and simple messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. Originally developed by IBM in the late 1990s, it has become a widely used standard for machine-to-machine (M2M) communication and the Internet of Things (IoT).

At its core, MQTT operates on a publish-subscribe messaging pattern. It involves three key components: publishers, subscribers, and a broker.

1. **Publishers**: These are devices or applications that send messages to the MQTT broker. Messages are published to specific "topics," which act as channels for communication. Topics are strings that categorize messages and allow subscribers to filter the messages they want to receive.

2. **Subscribers**: These are devices or applications that receive messages by subscribing to specific topics. Subscribers express interest in certain topics and receive messages published to those topics from the broker.

3. **Broker**: The MQTT broker is a server that acts as an intermediary between publishers and subscribers. It receives all published messages, filters them based on the topic, and distributes them to the subscribers that are interested in those topics. The broker manages the routing of messages, ensuring they reach the right subscribers.

4. **Retained Messages:**
   In MQTT, a retained message is a feature that allows a broker to store the last message published on a specific topic and deliver it to new subscribers when they first subscribe to that topic.

1. **Storage of Last Message:** The broker retains only the last message published with the retained flag set for a specific topic. If a new message with the retained flag set is published to the same topic, the previously retained message is replaced by the new one.

2. **Initial Message for New Subscribers:** When a new client subscribes to a topic with retained messages, it immediately receives the last retained message, if available, before any other messages are published on that topic.

3. **Clearing Retained Messages:** To clear a retained message for a particular topic, a client can publish an empty message with the retained flag set to true for that topic. This action effectively removes the retained message from the broker for that topic.

In this project { MOTOR : “ON” }, { MOTOR : “OFF” } messages are published from Node-Red to topic “echo_pro_pub/1” as retained messages so that, even if the ESP32 is Standby mode also it can read messages once it came to Wakeup state.

```
void callback(char* topic, byte* payload, unsigned int length) {
  // Handle incoming messages
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  StaticJsonDocument<256> docu;
  DeserializationError error = deserializeJson(docu, payload, length);

  if (error) {
    Serial.println("Failed to parse JSON");
    return;
  }

  String Motor_Action = docu["MOTOR"].as<String>();

  // Publish an empty message with retained flag set to false
  if (client.publish(SUBSCRIBETopic, "", false)) {
    Serial.println("Retained message erased!");
  } else {
    Serial.println("Failed to erase retained message.");
  }
  ………………………………………………………………….. .
}
```

Erasing retained messages by publishing null messages is shown above in the highlighted code. Once read by the ESP32, msg will be deleted by it.

3.4 Node Red

Node-RED is an open-source flow-based programming tool designed for connecting hardware devices, APIs, and online services. It provides a visual development environment to wire together devices, APIs, and online services by creating flows using nodes.

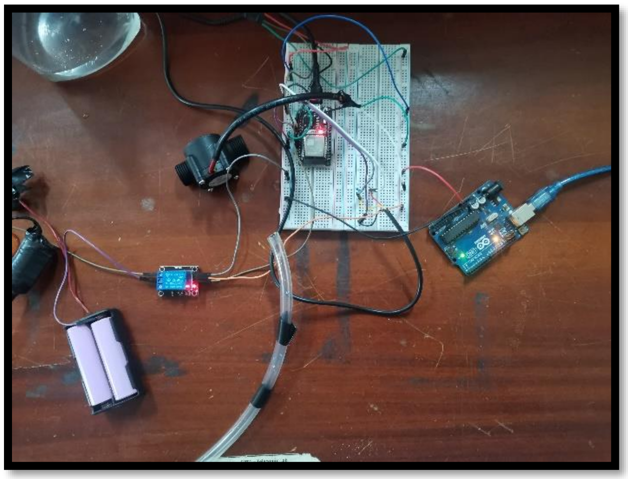
Flow-Based Programming:
The concept of Node-RED is based on flow-based programming, where each node performs a specific function or operation. Nodes can range from simple input/output operations to complex data processing, API calls, database interactions, and more.

Extensive Library of Nodes:
Node-RED has a vast ecosystem with a library of pre-built nodes contributed by the community. These nodes cover a wide range of functionalities, enabling users to interact with hardware (like Raspberry Pi GPIO pins), IoT devices, cloud services (such as AWS, Azure, or Google Cloud), databases, social media platforms, and more.

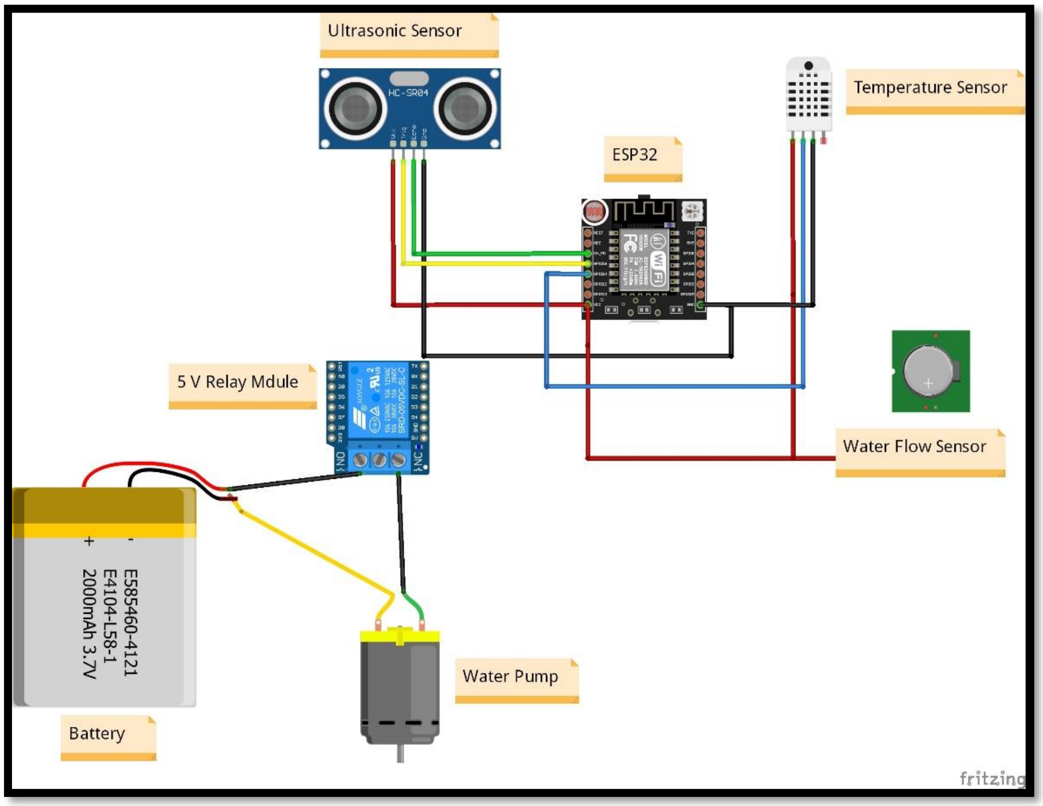
MQTT and IoT Integration:
Node-RED works particularly well in IoT scenarios due to its compatibility with MQTT, making it easy to connect and manage IoT devices and data.

4. Hardware Designs

4.1 Hardware Design



4.2 Graphic Circuit Design



5. Functionalities

1. Water Level Monitoring:

- Ultrasonic Sensor: Measure the water level inside the tank using an ultrasonic sensor like HC-SR04.
- Display: Show the water level on an LCD screen or send it to a display unit for visualization.

2. Real-time Monitoring and Alerts:

- Data Logging: Store water level data over time, enabling trend analysis and historical monitoring.
- Threshold Monitoring: Set thresholds for high and low water levels. Trigger alerts or actions when these thresholds are reached.

3. Connectivity and Communication:

- Wireless Connectivity: Use modules like Wi-Fi (ESP8266, ESP32) or Bluetooth to enable remote monitoring via a web interface or a mobile application.

- IoT Integration: Send water level data to cloud platforms (like IoT services) for remote access and monitoring.

4. Automation and Control:

- Pump Control: Automatically control a water pump based on the water level, turning it on or off to maintain a desired level.
- Notifications: Send alerts or notifications via SMS, email, or app notifications when the water level is critically high or low.

5. Energy Efficiency and Power Management:

- Low-Power Modes: Implement low-power modes for sensors and components to conserve energy when not actively monitoring.
- Battery Management: If the project involves battery power, include mechanisms to monitor battery levels and recharge or replace as needed.

6. User Interface and Interactivity:

- User Inputs: Allow users to set preferences, thresholds, or control parameters via buttons or a graphical interface.
- Visual Feedback: Use LEDs, buzzers, or displays to provide real-time feedback or status indicators.

The functionalities and features of the smart water tank project largely depend on the specific components, sensors, and capabilities integrated into the Arduino code and hardware setup. Each function serves to make the system more efficient, user-friendly, and reliable in monitoring and managing water levels in the tank.

6. Conclusions

The IoT smart water tank project effectively combines sensor technology, automation, and data-driven insights to create an efficient and reliable system. By continuously monitoring water levels, controlling motor operation, and measuring power usage, the project ensures optimal functionality, energy efficiency, and user convenience.

Moving forward, leveraging advanced analytics, remote accessibility, and safety enhancements can further elevate the system's capabilities, making it a robust solution for water management, conservation, and remote monitoring in various settings, from residential to industrial applications.

7. Future Improvements

Safety and Fail-Safes:

- Fail-Safe Mechanisms: Include fail-safe features to prevent overflow or empty tank conditions, such as emergency shut offs or alarms.

Double-Check Motor Working using Pressure Sensor:

- Pressure Sensor Placement: Install a pressure sensor at the bottom of the tank to verify the water level independently of the ultrasonic sensor. This redundancy ensures more accurate readings and acts as a backup system.
- Motor Activation Confirmation: Activate the motor based on readings from the ultrasonic sensor. Once the motor starts, cross-verify the change in pressure sensed at the bottom of the tank. If the motor operates but there's no change in pressure, it might indicate a motor malfunction or clogging in the pipeline.

Measure Power Usage with Current Sensor:

- Current Sensor Integration: Connect a current sensor in-line with the motor's power supply. This enables the measurement of current passing through the motor.
- Power Calculation: Use the measured current and the supply voltage to calculate the power consumption of the motor (Power = Voltage × Current). This data can help track energy usage and assess the efficiency of the system.

Motor Health Monitoring:

- Implement algorithms to analyze current sensor data over time to detect irregularities indicating motor wear, inefficiencies, or potential failures.

Energy Optimization:

- Use the power consumption data to optimize motor usage, perhaps by employing variable speed drives or optimizing pump cycles based on power usage patterns.

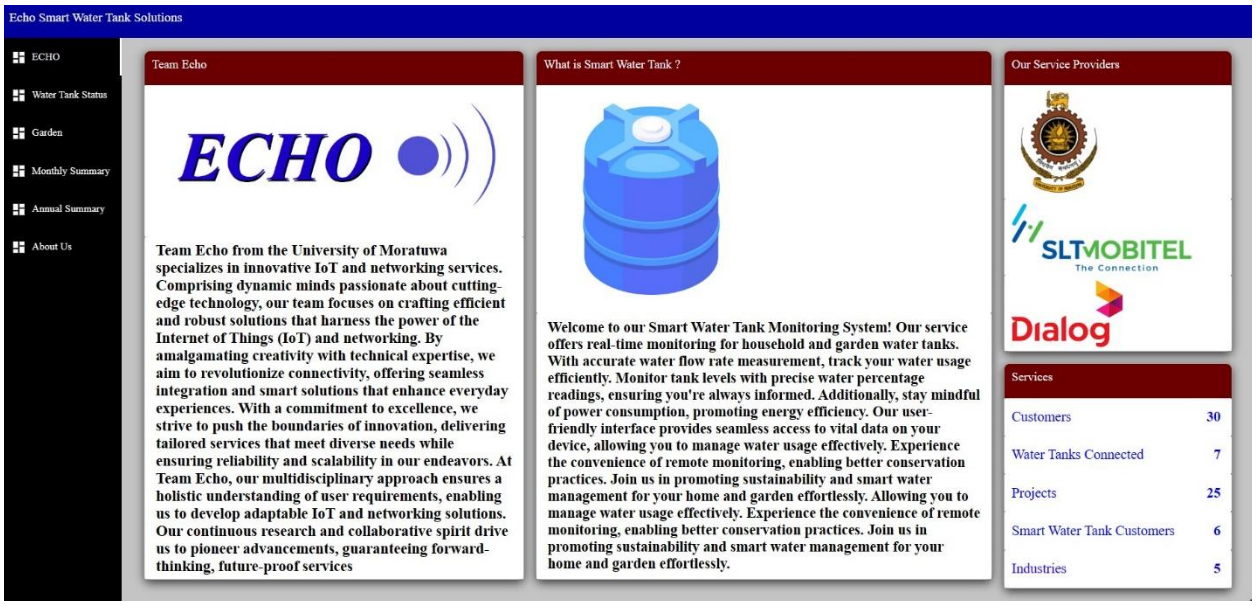
Data Logging and Analytics:

- Store power consumption data along with water level and motor activation information for long-term analysis. This can reveal usage patterns, trends, and potential improvements for efficiency.

Remote Monitoring:

- Integrate these features with a remote monitoring system (via IoT platforms or web interfaces) to allow users to track power usage and motor functionality remotely.

8. Node-RED Dashboard



Echo Smart Water Tank Solutions

Team Echo

Team Echo from the University of Moratuwa specializes in innovative IoT and networking services. Comprising dynamic minds passionate about cutting-edge technology, our team focuses on crafting efficient and robust solutions that harness the power of the Internet of Things (IoT) and networking. By amalgamating creativity with technical expertise, we aim to revolutionize connectivity, offering seamless integration and smart solutions that enhance everyday experiences. With a commitment to excellence, we strive to push the boundaries of innovation, delivering tailored services that meet diverse needs while ensuring reliability and scalability in our endeavors. At Team Echo, our multidisciplinary approach ensures a holistic understanding of user requirements, enabling us to develop adaptable IoT and networking solutions. Our continuous research and collaborative spirit drive us to pioneer advancements, guaranteeing forward-thinking, future-proof services

What is Smart Water Tank ?

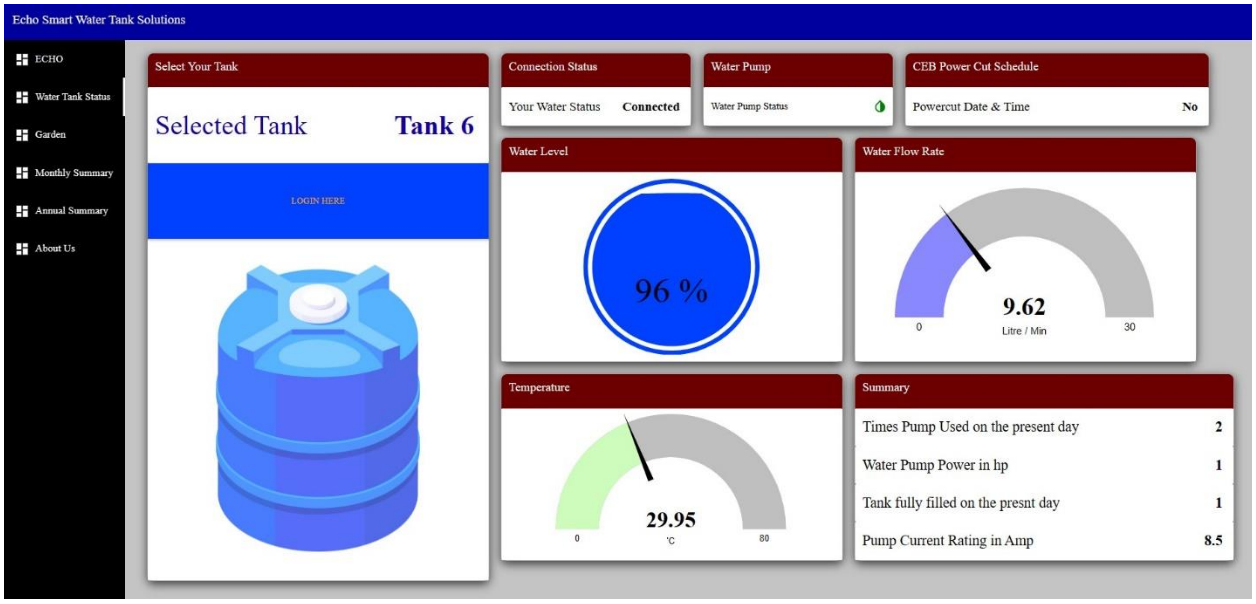
Welcome to our Smart Water Tank Monitoring System! Our service offers real-time monitoring for household and garden water tanks. With accurate water flow rate measurement, track your water usage efficiently. Monitor tank levels with precise water percentage readings, ensuring you're always informed. Additionally, stay mindful of power consumption, promoting energy efficiency. Our user-friendly interface provides seamless access to vital data on your device, allowing you to manage water usage effectively. Experience the convenience of remote monitoring, enabling better conservation practices. Join us in promoting sustainability and smart water management for your home and garden effortlessly. Allowing you to manage water usage effectively. Experience the convenience of remote monitoring, enabling better conservation practices. Join us in promoting sustainability and smart water management for your home and garden effortlessly.

Our Service Providers

Services

Customers	30
Water Tanks Connected	7
Projects	25
Smart Water Tank Customers	6
Industries	5

Login Page



Echo Smart Water Tank Solutions

Selected Tank Tank 6

Connection Status: Your Water Status Connected

Water Pump: Water Pump Status

CEB Power Cut Schedule: Powercut Date & Time No

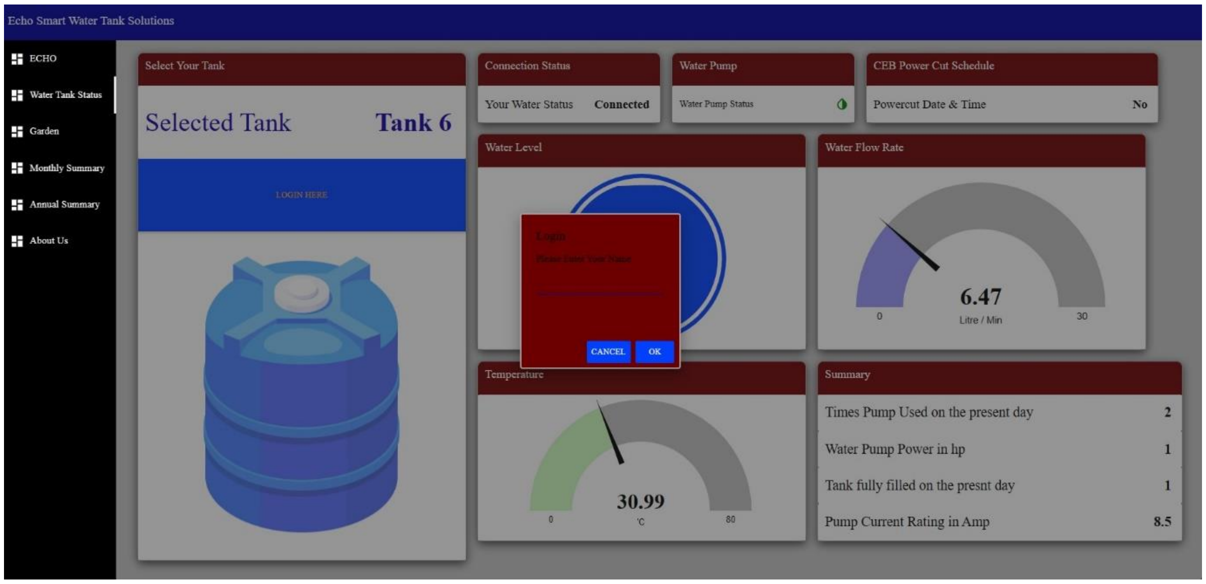
Water Level: 96 %

Water Flow Rate: 9.62 Litre / Min (0 – 30)

Temperature: 29.95 °C (0 – 80)

Summary

Times Pump Used on the present day	2
Water Pump Power in hp	1
Tank fully filled on the presnt day	1
Pump Current Rating in Amp	8.5



Select Your Tank

Selected Tank **Tank 6**

Connection Status

Your Water Status **Connected**

Water Pump

Water Pump Status

CEB Power Cut Schedule

Powercut Date & Time **No**

Water Level

Water Flow Rate

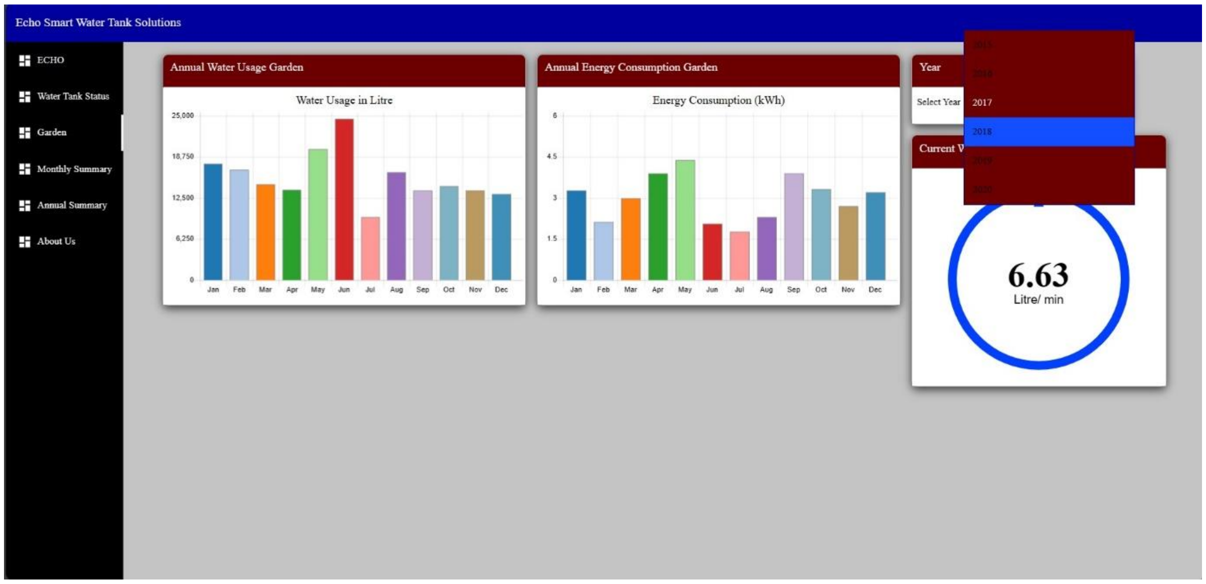
6.47 Litre / Min

Temperature

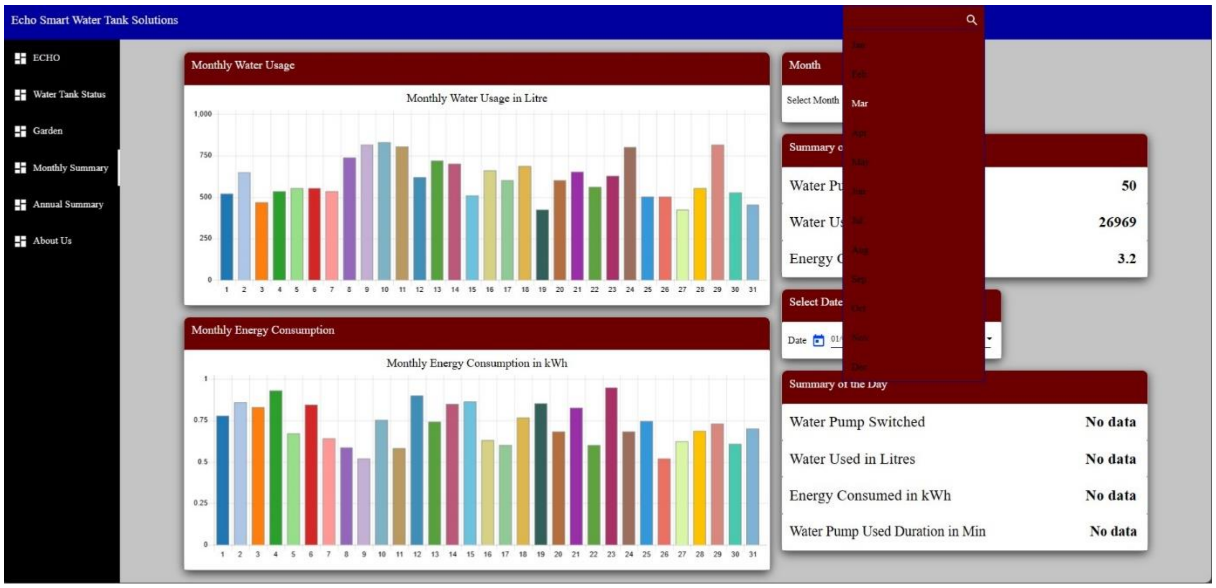
30.99 'C

Summary

Times Pump Used on the present day	2
Water Pump Power in hp	1
Tank fully filled on the presnt day	1
Pump Current Rating in Amp	8.5



6.63 Litre/ min

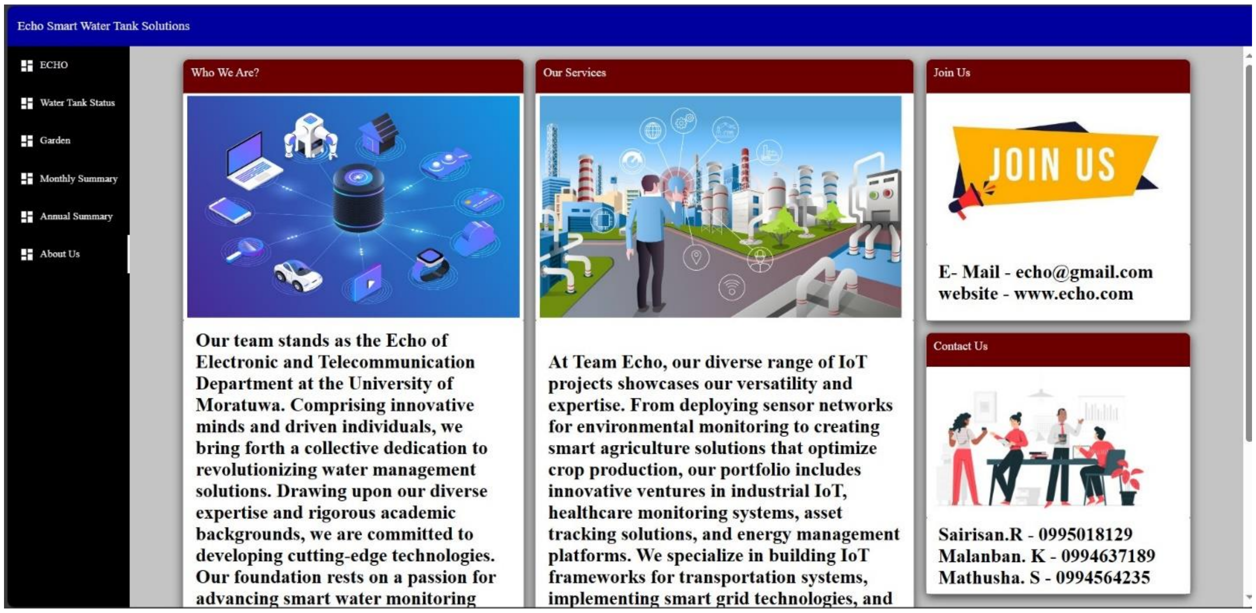


Summary o

Water Pu	50
Water Us	26969
Energy C	3.2

Summary of the Day

Water Pump Switched	No data
Water Used in Litres	No data
Energy Consumed in kWh	No data
Water Pump Used Duration in Min	No data



Who We Are?

Our team stands as the Echo of Electronic and Telecommunication Department at the University of Moratuwa. Comprising innovative minds and driven individuals, we bring forth a collective dedication to revolutionizing water management solutions. Drawing upon our diverse expertise and rigorous academic backgrounds, we are committed to developing cutting-edge technologies. Our foundation rests on a passion for advancing smart water monitoring

Our Services

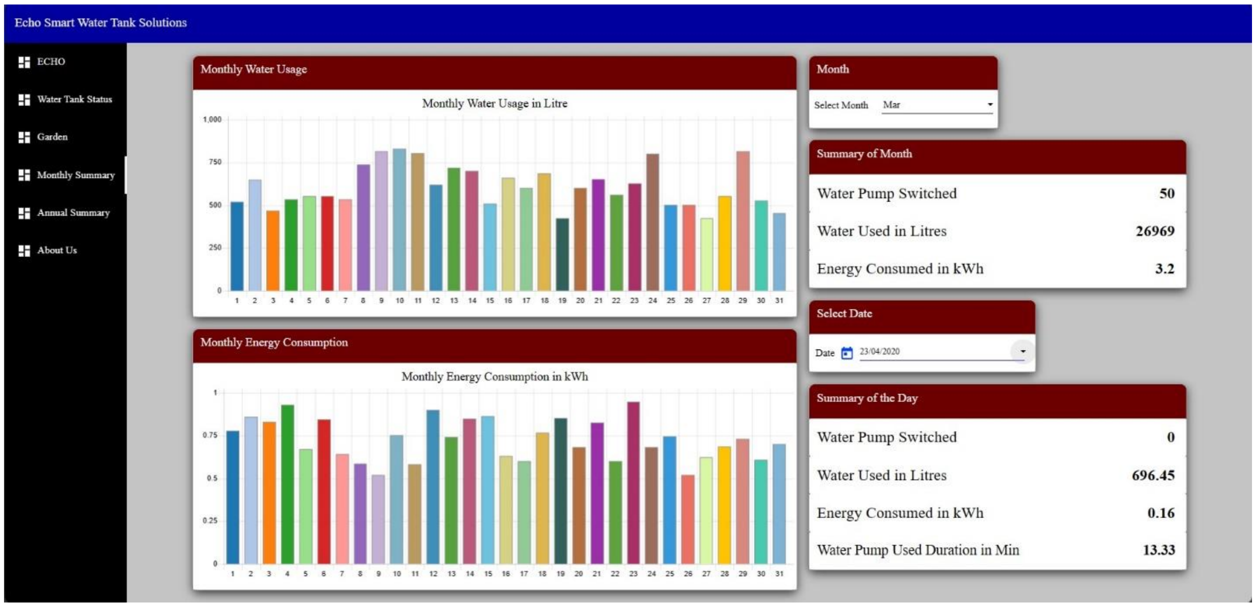
At Team Echo, our diverse range of IoT projects showcases our versatility and expertise. From deploying sensor networks for environmental monitoring to creating smart agriculture solutions that optimize crop production, our portfolio includes innovative ventures in industrial IoT, healthcare monitoring systems, asset tracking solutions, and energy management platforms. We specialize in building IoT frameworks for transportation systems, implementing smart grid technologies, and

Join Us

E- Mail - echo@gmail.com
website - www.echo.com

Contact Us

Sairisan.R - 0995018129
Malanban. K - 0994637189
Mathusha. S - 0994564235


Echo Smart Water Tank Solutions
ECHO
Water Tank Status
Garden
Monthly Summary
Annual Summary
About Us
Monthly Water Usage
Monthly Water Usage in Litre
Monthly Energy Consumption
Monthly Energy Consumption in kWh

Month

Select Month: Mar

Summary of Month

Water Pump Switched	50
Water Used in Litres	26969
Energy Consumed in kWh	3.2

Select Date

Date: 23/04/2020

Summary of the Day

Water Pump Switched	0
Water Used in Litres	696.45
Energy Consumed in kWh	0.16
Water Pump Used Duration in Min	13.33

11. Appendix

15.1 Source Code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// WiFi credentials
const char* ssid = "AnBU";
const char* password = "anbu12malan";

// MQTT broker details
const char* mqtt_server = "test.mosquitto.org";
const int mqtt_port = 1883; // Default MQTT port
// const char* mqtt_username = "YOUR_MQTT_USERNAME";
// const char* mqtt_password = "YOUR_MQTT_PASSWORD";
const char  *mqttClientID = "NodeMCU_group031"; // CHANGE THIS acording to your
group number
const char  *PUBLISHTopic = "echo_pro_pub"; // Topic for PUBLISH
const char  *SUBSCRIBETopic = "echo_pro_sub/1"; // Topic for SUBSCRIBE

//PIN Define
const int Triger_Pin = 18;
const int Ultra_Sonic_Pin = 19;
const int Temperature_Pin = 4;
const int Flow_Rate_Pin = 21;
const int Relay_Pin = 22;

//Delay Time
const int DELAY_OFF = 5000; //When Motor OFF
const int DELAY_ON = 500; //When Motor ON
int MOTOR_STATUS = 0;
//Other Variables
long duration;
volatile long pulse;
unsigned long lastTime;
int tankheight=17;
//Sensor Reading Function
float temperatureC;
int distance;
float flowrate;
//MQTT msg variables
float msg_temp;
```

```
int msg_dist;
float msg_flowrate;

// Data wire of Temperature Sensor is connected on ESP32
#define ONE_WIRE_BUS Temperature_Pin

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

// Pass the oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
  delay(10);
  // Connect to Wi-Fi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  // Handle incoming messages
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  StaticJsonDocument<256> docu;
  DeserializationError error = deserializeJson(docu, payload, length);

  if (error) {
    Serial.println("Failed to parse JSON");
    return;
  }

  String Motor_Action = docu["MOTOR"].as<String>();

  // Publish an empty message with retained flag set to false
  if (client.publish(SUBSCRIBETopic, "", false)) {
    Serial.println("Retained message erased!");
  } else {
    Serial.println("Failed to erase retained message.");
  }
  // client.publish(SUBSCRIBETopic, "");

  if (Motor_Action == "ON"){
    digitalWrite(Relay_Pin, HIGH);
    Serial.println("Motor is ON");
    MOTOR_STATUS = 1;
  } else if (Motor_Action == "OFF"){
    digitalWrite(Relay_Pin, LOW);
    Serial.println("Motor is OFF");
    MOTOR_STATUS = 0;
  }
}

void PINInit(){
  pinMode(Triger_Pin, OUTPUT);
  pinMode(Ultra_Sonic_Pin, INPUT);
  // pinMode(Temperature_Pin, INPUT);
  pinMode(Relay_Pin, OUTPUT);
  pinMode(Flow_Rate_Pin, INPUT);
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect(mqttClientID)) {
      Serial.println("connected");
      // Once connected, subscribe to desired topics
      client.subscribe(SUBSCRIBETopic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

```
void Sensor_Reading(float* temp, int* dist, float* flow){
  //Ultra Sonic Sensor ***********************************************************
  digitalWrite(Triger_Pin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(Triger_Pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(Triger_Pin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(Ultra_Sonic_Pin, HIGH);
  distance = duration * 0.034 / 2;

  if (distance<=0){
    Serial.println("Out of Range");
  } else{
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  }

  //Temperature Sensor************************************************************
  sensors.requestTemperatures(); // Send the command to get temperature readings
  // Use the index to get specific temperature (if multiple sensors are
connected)
  temperatureC = sensors.getTempCByIndex(0);

  if (temperatureC != DEVICE_DISCONNECTED_C) {
    Serial.print("Temperature: ");
    Serial.print(temperatureC);
    Serial.println("°C");
  } else {
    Serial.println("Error: Unable to read temperature");
  }

  //Flowrate Sensor***************************************************************
  flowrate = 2.663 * pulse;
  if (millis() - lastTime > 1000) {
    pulse = 0;
    lastTime = millis();
  }
  Serial.print(flowrate);
  Serial.println(" mL/s");

  // Assign values to pointers
  *temp = temperatureC;
  *dist = distance;
  *flow = flowrate;
}

void setup() {
  Serial.begin(115200);
  PINInit();
  sensors.begin();
  attachInterrupt(digitalPinToInterrupt(Flow_Rate_Pin), increase, RISING);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  // Publish messages periodically
  Sensor_Reading(&msg_temp, &msg_dist, &msg_flowrate);
  // Create a JSON object
  StaticJsonDocument<200> doc;

  // Add data to the JSON object
  doc["Temperature"] = msg_temp;
  doc["Distance"] = tankheight-msg_dist;
  doc["Flowrate"] = msg_flowrate;
  doc["MotorStatus"] = MOTOR_STATUS;

  // Convert the JSON object to a string
  char buffer[256];
  serializeJson(doc, buffer);

  // Publish the JSON-formatted message
  client.publish(PUBLISHTopic, buffer);

  if (MOTOR_STATUS == 0){
    delay(DELAY_OFF);
  }else if (MOTOR_STATUS == 1){
    delay(DELAY_ON);
  }
}

void increase() {
  pulse++;
}
```

15.2 Product Explanations Video - YouTube

Click the link below for the explanation video on YouTube.

https://youtu.be/yq0LR_PZ6Eo