CHAPTER 1

INTRODUCTION

Groundwater is one of the most essential natural resources, providing water for drinking, irrigation, and industrial use. However, in recent years, its excessive and unregulated consumption has led to a significant decline in groundwater levels, especially in urban areas like Bangalore and Chennai, and in various rural villages. Observing the growing issue of groundwater wastage in my surroundings inspired me to develop a system that can help reduce overconsumption and promote water conservation.

This project, titled **"Underground Water Level Consumption Monitoring System – A Societal Perspective,"** focuses on monitoring household-level groundwater usage through an automated and intelligent system. By integrating sensors, microcontrollers, and communication technologies, the system tracks water flow, alerts users when limits are reached, and even restricts usage until a recharge (via payment) is made. The system utilizes MQTT protocol as a mediator between the hardware (ESP32) and a mobile dashboard, providing real-time data, notifications, and QR-based payment integration.

This system offers scalability and adaptability, making it suitable not just for individual households but also for apartment complexes, schools, and small-scale industries. Its modular design allows for easy customization based on user requirements and local water consumption norms. By leveraging IoT technologies and cloud-based monitoring, the system creates a digital footprint of water usage that can be analyzed for trends, inefficiencies, and potential policy-making. In the long run, such data-driven solutions can support local governing bodies and urban planners in implementing smarter water management practices and promoting a culture of sustainability among citizens.

This solution aims to bring awareness, control, and accountability to individual water usage, ultimately contributing to groundwater conservation efforts on a societal level.

CHAPTER 2

PROBLEM SATEMENT

2.1.1 Background of the Problem:

Groundwater depletion has become a critical concern, particularly in fast-growing urban areas like Bangalore and Chennai. These cities, along with many others, rely heavily on underground water sources for domestic, industrial, and agricultural needs. However, the excessive and unmonitored extraction of groundwater has led to severe water shortages, environmental degradation, and an increased risk of drought conditions.

One of the primary causes of groundwater depletion is the lack of awareness among users regarding their daily water consumption. In both urban and rural areas, water wastage is rampant, driven by inefficient usage and the absence of real-time tracking mechanisms. Without a structured monitoring system, households and industries continue to exploit water resources without realizing the long-term consequences.

2.1.2 Observed Challenges:

Several challenges contribute to the issue of excessive water usage and wastage:

- **Lack of Affordable Monitoring Solutions:** There is no cost-effective system available at the household level to track daily water consumption in real time. Existing solutions are often expensive and not accessible to the general population.
- **Unintentional Overuse:** Users frequently overconsume water without being aware, as there are no alerts or reminders to indicate when consumption surpasses recommended levels.
- **No Automated Water Regulation:** Current water supply systems do not offer automatic cutoffs or restrictions based on consumption limits, leading to unnecessary wastage.
- **Payment and Accountability Issues:** In many cases, water usage charges are either fixed or estimated, which prevents users from understanding and managing their consumption accurately. This can lead to disputes over billing and inefficient water conservation efforts.

2.1.3 Need for the System

To effectively tackle groundwater depletion and encourage responsible water usage, a **smart monitoring system** is essential. The proposed system will provide a **real-time tracking mechanism** for underground water consumption, enabling users to manage their usage efficiently.

2.1.4 The **core components of this solution** include:

- **ESP32-Based Monitoring:** The system will integrate ESP32 microcontrollers to measure water usage and transmit data seamlessly.
- **MQTT Communication Protocol:** The solution will use MQTT to ensure fast and reliable data transmission to a mobile dashboard, allowing users to view live consumption updates.
- **Automated Alerts & Cutoff Mechanism:** Users will receive alerts when predefined water consumption limits are reached, preventing excessive usage. Additionally, an automatic shutdown feature will stop the water flow to avoid wastage.
- **QR-Based Payment System:** Once a user exceeds their allotted quota, water access can be resumed through a **QR-based payment system**, ensuring accountability and promoting conscious consumption.

By implementing this system, water wastage can be minimized, and users will develop better habits regarding resource conservation. Ultimately, this solution will support sustainable water management, address the ongoing groundwater crisis and contributing to long-term environmental stability.

2.2 Objectives:

The primary goal of this project is to design and implement a **smart water monitoring system** that helps households track their underground water consumption in **real time**. Given the alarming rate of groundwater depletion, particularly in **water-stressed regions like Telangana and other parts of India**, this system aims to **encourage responsible water usage** and **minimize wastage** through an efficient, automated approach.

Key Features and Functionality

- **Real-Time Monitoring with ESP32 and Flow Sensors**

The system integrates **ESP32** microcontrollers with **flow sensors** to continuously measure water usage at the household level. By collecting **accurate consumption data**, users can gain better insights into their daily water habits and identify areas where they can **reduce unnecessary usage**.

- **Automated Overconsumption Prevention & Relay Mechanism**

A critical feature of the system is its ability to **detect overconsumption**. When a user exceeds a predefined water limit, the system sends an **alert notification** and **automatically shuts off water flow** through a **relay-based control mechanism**. This ensures that excess usage is curtailed without requiring manual intervention, promoting **efficient resource utilization**.

- **Mobile Dashboard with MQTT Protocol for Live Updates**

Users can access a **mobile dashboard** powered by the **MQTT communication protocol**, which enables **seamless data transmission**. The dashboard displays **real-time consumption**

statistics, alerts, and historical usage trends, helping households monitor their water usage patterns and make informed decisions to **conserve groundwater**.

- **QR-Code-Based Payment System for Service Resumption**

To maintain accessibility while reinforcing accountability, the system incorporates a **QR-based payment feature**. If a user surpasses their allocated water quota, they can **resume service** after making a payment via the QR system. This model encourages **conscious usage** while ensuring fairness in water distribution.

Impact and Sustainability

By implementing this smart monitoring solution, households can significantly **reduce water wastage**, **optimize consumption**, and contribute to **long-term groundwater conservation**. The system is particularly beneficial in regions facing **severe water shortages**, where **unregulated usage** exacerbates depletion concerns. By increasing **awareness, accessibility, and control**, this solution serves as a **scalable approach** toward a **sustainable future**, making water conservation **both practical and manageable** at the individual level.

2.3 Related Work:

Several studies and projects have addressed groundwater monitoring and water conservation using IoT and smart technologies. Many systems focus on measuring water levels in tanks or reservoirs, but fewer emphasize real-time household consumption monitoring. Some existing solutions use basic flow sensors combined with microcontrollers, yet they lack advanced features like remote monitoring or automated flow control.

Recent advancements have seen the integration of MQTT protocols for efficient data transmission and dashboard visualization, improving user interaction and data accessibility. Payment-based control systems, such as QR-code activation, are also emerging to encourage responsible water use by linking consumption to payment.

However, most existing models lack a comprehensive approach combining real-time usage tracking, automated control, alert notifications, and integrated payment—all of which are crucial to effectively managing groundwater at the community level. This project builds upon these concepts to deliver an affordable and scalable solution aimed at reducing groundwater wastage in regions like Telangana and India.

2.4 System overview:

The Underground Water Level Consumption Monitoring System is designed to provide real-time monitoring and control of household groundwater usage. At the core is the ESP32 microcontroller, which interfaces with a water flow sensor to measure the volume of water consumed. The ESP32 processes this data and sends it to an MQTT broker, acting as a mediator between the hardware and the mobile app dashboard.

Users can view their current water consumption and receive alerts through the dashboard. When the water usage reaches a predefined limit, the system automatically activates a relay to stop the water flow. To resume usage, the system displays a QR code on an LCD screen, which

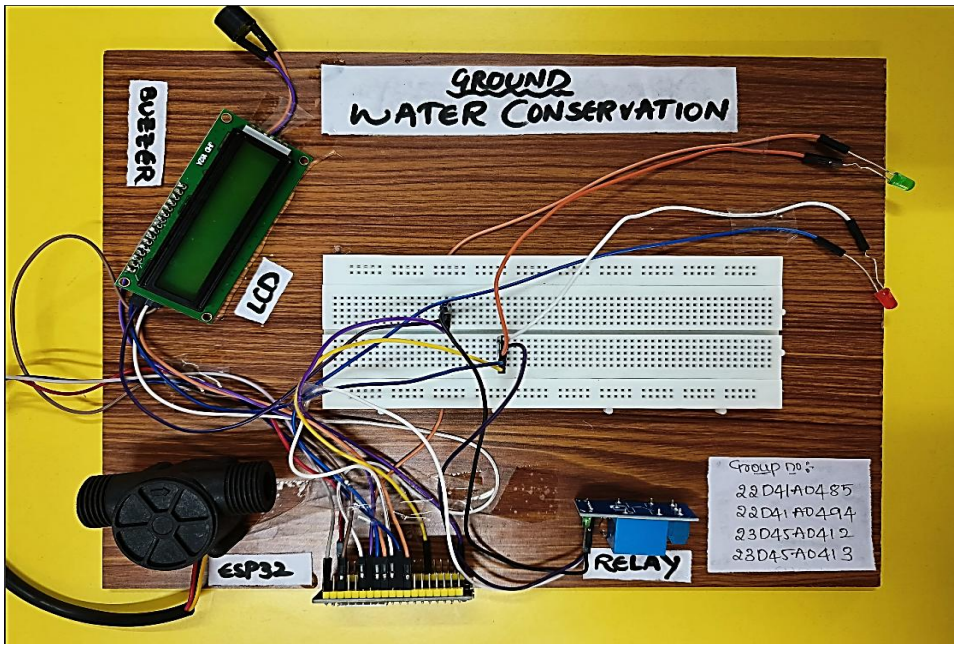
the user scans to make a payment. Once payment is confirmed, the relay reactivates, allowing water flow to continue.

2.5 Hardware system design:

The hardware design of the Underground Water Level Consumption Monitoring System consists of several key components working together to monitor and control groundwater usage:

- **ESP32 Microcontroller**: Acts as the central processing unit, collecting data from sensors, controlling the relay, and communicating with the MQTT server.
- **Water Flow Sensor**: Measures the volume of water passing through the pipe in real time. This sensor provides accurate flow rate data to the ESP32 for processing.
- **Relay Module**: Controls the water flow by switching the valve or pump ON/OFF based on signals from the ESP32. It stops the water supply when the consumption limit is reached.
- **LCD Display**: Shows real-time water usage information and displays the QR code for payment when the flow is stopped.
- **Buffer Circuit**: Ensures stable voltage and protects sensitive components from electrical noise or fluctuations.

All components are connected carefully to ensure seamless communication and operation. The ESP32 is programmed to read sensor inputs, send data to the MQTT broker, and respond to commands from the mobile app and payment system. The design prioritizes low power consumption, cost-effectiveness, and reliability for practical household use.



Fig(1) Hardware design

2.6 FLOW CHART:

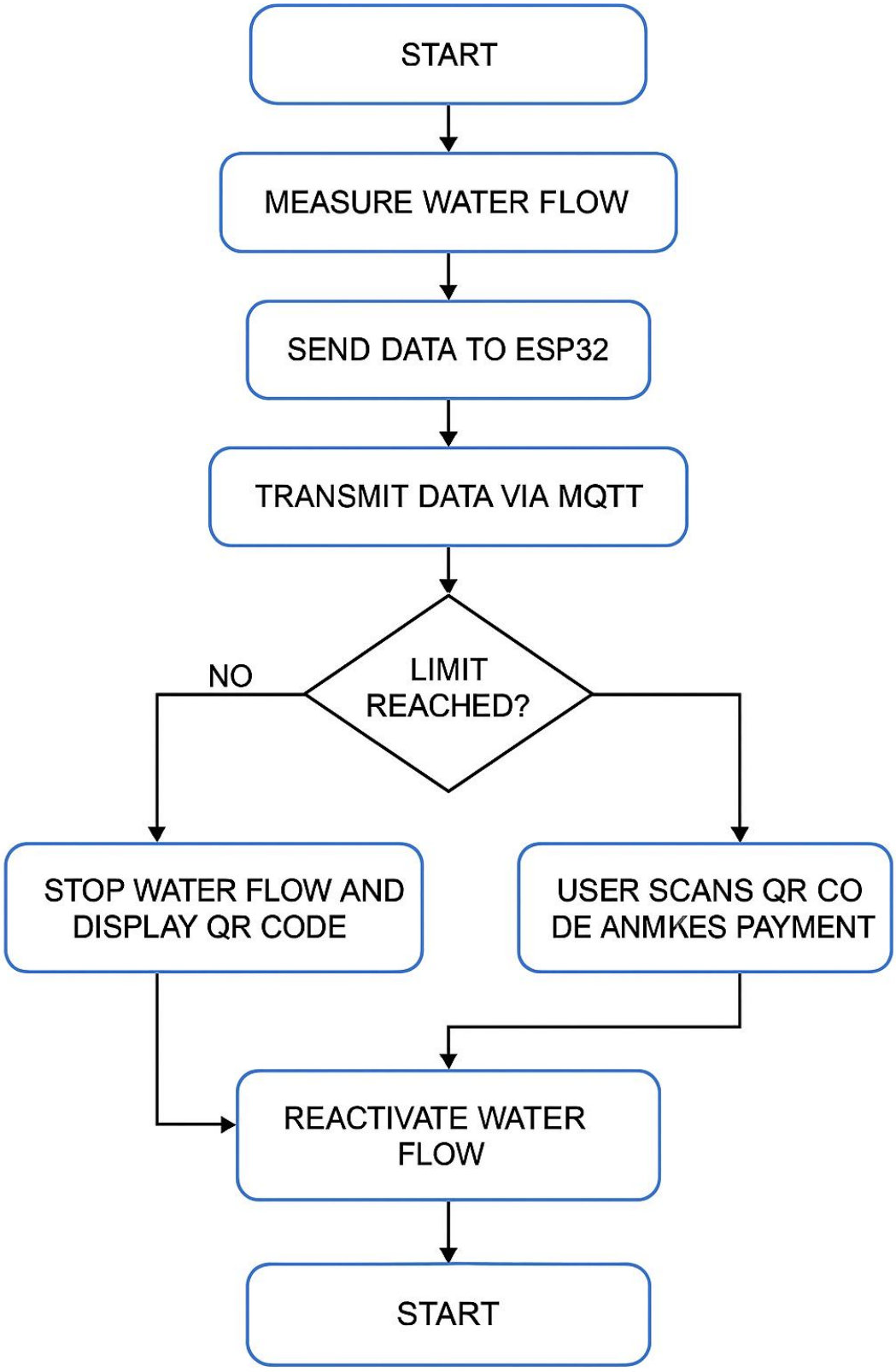


Fig-2.6 Flow diagram

Working:

1. System Initialization

When the system is powered on, the ESP32 microcontroller takes control and initializes all connected components, including the flow sensor, relay, LCD, and Wi-Fi module. The ESP32 then establishes a connection with the MQTT broker, which acts as the communication link between the hardware and the mobile dashboard. Once all components are successfully configured, the system is ready to start monitoring water consumption.

2. Real-Time Water Monitoring

The water flow sensor continuously detects and measures the volume of water being consumed. This data is sent to the ESP32 microcontroller, which processes the information and forwards it to the MQTT server. The MQTT broker relays the processed data to the user's mobile dashboard, where live updates on water usage trends are displayed. This enables the user to monitor consumption in real time, allowing better awareness and control over daily usage.

3. Limit Check and Alert Mechanism

As water consumption progresses, the system constantly compares the current usage against a predefined threshold limit. If the water consumption remains within the set limit, the system continues monitoring normally. However, once the limit is exceeded, the system triggers an alert notification, notifying the user that they have reached their allowed quota. Simultaneously, the relay mechanism is activated, cutting off the water supply to prevent further excess usage.

4. QR Code Display and Payment Processing

Once the water flow is stopped, the LCD screen displays a QR code, prompting the user to scan and make a payment to resume service. The QR code is linked to a payment gateway or mobile application, allowing seamless transactions for water access renewal. This payment-based approach encourages responsible consumption, ensuring users pay attention to their water usage and prevent unnecessary wastage.

5. Resumption of Water Flow

Upon successful payment, the MQTT broker sends a confirmation signal back to the ESP32. The ESP32 processes this verification and reactivates the relay, restoring the water flow for further usage. The system then resets and continues monitoring, ensuring a continuous cycle of tracking, alerting, restricting, and resuming based on water consumption patterns. This automated approach fosters sustainability, accountability, and efficient water management at the household level.

2.7 Advantages:

1. Real-Time Monitoring

Users can instantly track their daily water consumption via the mobile dashboard, providing live updates on usage patterns. This feature helps individuals stay informed about their water consumption, allowing them to adjust habits and make responsible decisions to prevent wastage.

2. Automatic Control System

The relay mechanism automatically stops water flow once a predefined limit is exceeded, ensuring that overconsumption is prevented without requiring manual intervention. This automation enhances efficiency while reinforcing responsible resource management.

3. Seamless Payment Integration

The QR-code-based payment system allows users to quickly recharge and resume their water service after reaching the allocated limit. By implementing this prepaid utility model, households gain more control over their consumption, while promoting accountability and mindful usage.

4. Remote Accessibility

Thanks to MQTT and ESP32 integration, users can monitor and manage their water consumption remotely using a smartphone or IoT-enabled device. This ensures convenience and flexibility, making it possible to track and regulate water usage from anywhere, anytime.

5. Supports Groundwater Conservation

The system actively encourages sustainable water usage, reducing excessive consumption and promoting conservation efforts. Particularly beneficial for water-scarce regions like Telangana, this solution helps mitigate groundwater depletion, ensuring long-term availability for future generations.

6. Cost-Effective & Scalable

Designed using low-cost components, the system remains affordable and practical for widespread implementation. It can be easily adapted to suit individual households, apartment complexes, or even small communities, making it a scalable solution for sustainable water management.

2.8 Disadvantages:

1. Internet Dependency

The system heavily relies on internet connectivity for MQTT communication and mobile dashboard updates. If there is a Wi-Fi outage or poor network coverage, users may lose access to real-time monitoring, alerts, and payment functionality. This could pose challenges in areas with unreliable internet infrastructure, limiting the effectiveness of the system.

2. Initial Setup Cost & Technical Complexity

While the system utilizes cost-effective components, the initial installation may require technical expertise for proper configuration. Users in rural areas or non-tech-savvy households might face difficulties in setting up the ESP32, flow sensors, relays, and MQTT integration, leading to additional setup costs for professional assistance.

3. Limitations of Predefined Thresholds

The system operates based on preset water consumption limits, which may not always accurately reflect a household's actual needs. If the threshold is set too low, it could result in frequent interruptions, frustrating users. Conversely, if the limit is too high, it may not effectively encourage conservation, making it essential to fine-tune settings based on realistic consumption patterns.

4. Component Maintenance & Reliability

Regular maintenance may be required for flow sensors, relays, and LCD screens, especially in hard water areas where scaling can occur. Over time, sensor calibration issues, relay malfunctions, or dust buildup could impact system accuracy and responsiveness, requiring periodic inspections and replacements to maintain optimal performance.

5. Payment System Reliability & User Experience

The QR-based payment system introduces a dependency on mobile applications and banking platforms for transaction processing. If there are technical failures, network issues, or errors in the payment gateway, users may experience delays in water service resumption, causing inconvenience. Ensuring a robust fallback mechanism, such as offline recharge options, could mitigate such risks.

2.9 Applications:

1. Household Water Management

The system can be installed in individual homes to provide real-time monitoring of groundwater usage, helping residents stay aware of their consumption and **prevent** excessive wastage. By offering automated alerts and control mechanisms, it encourages responsible usage, ensuring that households make informed decisions to conserve resources.

2. Apartments and Gated Communities

Multi-family housing units, apartment complexes, and gated communities can benefit from this system by managing shared underground water sources more efficiently. The centralized tracking and automated cutoff mechanisms help regulate individual unit consumption, ensuring fair distribution among residents and minimizing overall water wastage.

3. Rural and Semi-Urban Areas

Groundwater is often the primary water source in villages and towns, where unregulated consumption and unnoticed wastage pose significant challenges. Implementing this system in rural settings ensures better monitoring, helping local communities track usage patterns and adopt conservation practices without requiring complex technical expertise.

4. Government and Municipal Water Projects

Local water supply systems can integrate this smart monitoring solution to enhance billing accuracy, resource allocation, and conservation efforts. By tracking real-time usage across communities, municipal authorities can make data-driven decisions to optimize water management policies, reducing the strain on groundwater reserves.

5. Smart City Initiatives

Aligning with urban smart infrastructure programs, this system provides IoT-enabled water management solutions that enable real-time tracking, automation, and analytics in urban centers. Integration with smart meters and cloud-based dashboards can help cities implement sustainable water distribution frameworks, contributing to efficient groundwater conservation.

6. Educational and Awareness Campaigns

The system can be leveraged in schools, universities, and community-driven programs as an educational tool to demonstrate the importance of groundwater conservation. Interactive workshops can showcase technology-driven solutions, helping students and communities understand sustainable water management practices while fostering environmental responsibility.

CHAPTER 3

METHODOLOGY

3.1 Proposed system:

The proposed system is designed to be a smart, IoT-based groundwater consumption monitoring solution, leveraging the ESP32 microcontroller and the MQTT protocol for efficient communication and automation. This system focuses on real-time tracking of water usage, automated flow control, and user interaction via a mobile dashboard, ensuring responsible and sustainable water management at the household level.

The system continuously monitors water usage using a flow sensor, and once the preset consumption limit is reached, it cuts off the water supply using a relay.

To ensure easy reactivation, the system displays a **QR code** on an LCD screen, enabling users to **make payments digitally**. Once the payment is confirmed, the system resumes the water flow. MQTT serves as the communication bridge between the ESP32 and the mobile app, providing efficient data exchange and remote accessibility.

Unlike traditional water systems, this model promotes **self-regulation**, **accountability**, and **conservation**. It is scalable, cost-effective, and suitable for individual homes, apartments, or rural setups where groundwater misuse is common.

3.2 Block diagram:

The block diagram illustrates the overall structure and flow of the Underground Water Consumption Monitoring System. At the core of the system is the **ESP32 microcontroller**, which acts as the main control unit. It receives input from the **water flow sensor**, which detects and sends real-time data on water usage.

The ESP32 processes this data and sends it to an **MQTT broker**, which communicates with a **mobile application**. This allows the user to view live updates and receive alerts regarding water consumption.

Simultaneously, the ESP32 also controls an **LCD display**, which shows usage information and the **QR code** when the usage limit is exceeded. The **relay module** is connected to control the water supply—it cuts off the flow when the preset threshold is reached and reactivates it once payment is completed.

This design ensures effective water monitoring, user alerts, automated control, and digital payment—all working together to encourage groundwater conservation.

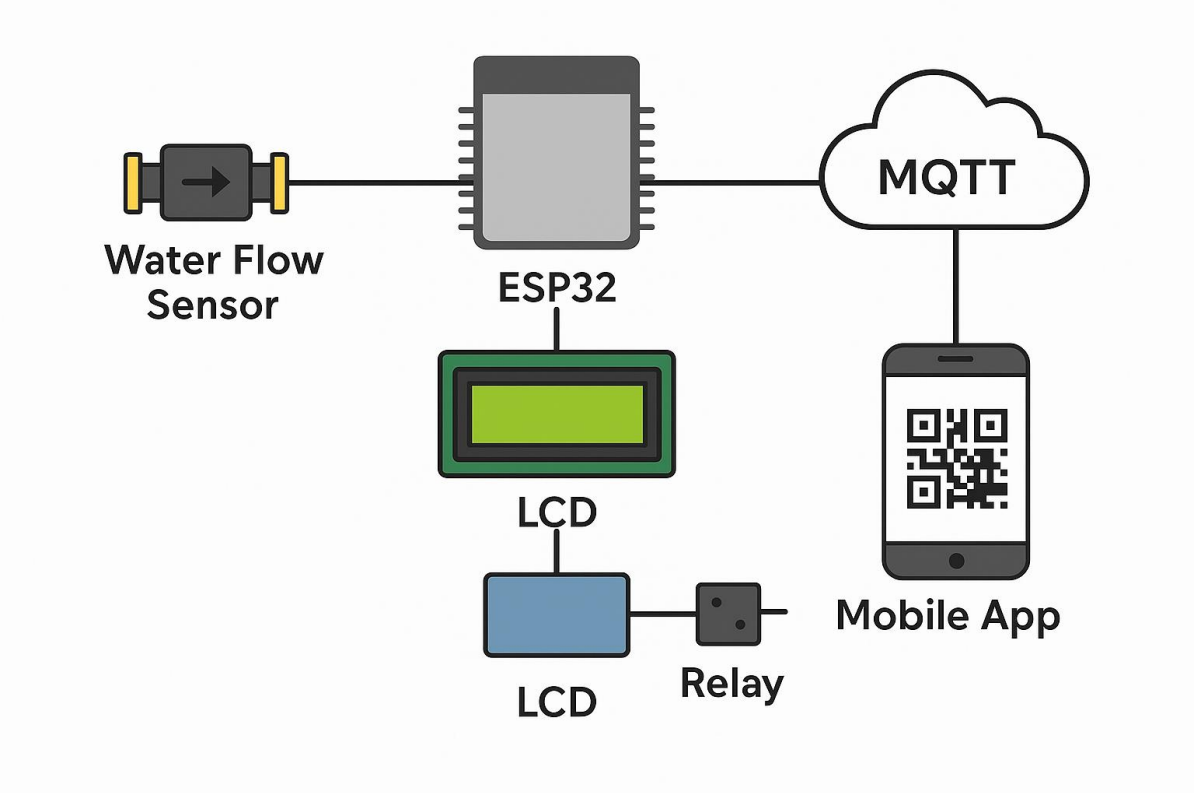


Fig-3.2 circuit diagram

3.3 Components:

1. ESP32 Microcontroller
2. Water Flow Sensor
3. Relay Module
4. LCD Display
5. Buffer Circuit
6. Wi-Fi Network
7. MQTT Broker
8. Power Supply

3.3.1 ESP-32 Microcontroller:

The ESP32 is a highly integrated, low-cost microcontroller with built-in Wi-Fi and Bluetooth, making it ideal for IoT-based applications like smart water monitoring systems. It is based on a dual-core Tensilica LX6 processor, running at up to 240 MHz, which allows it to handle multiple tasks simultaneously. In this project, the ESP32 acts as the brain of the system, responsible for collecting data from the water flow sensor and executing logic to manage water consumption.

It reads digital pulse inputs from the flow sensor and calculates the exact volume of water consumed. The ESP32 then processes this data and determines whether the usage limit has been reached. If so, it controls the relay module to cut off the water supply automatically. It is also responsible for sending real-time data to the MQTT broker through its Wi-Fi capability, ensuring the mobile dashboard reflects accurate, live updates.

Fig 3.3.1

The ESP32 is programmed using the Arduino IDE, which supports various libraries for MQTT communication, sensor reading, and LCD interfacing. It features multiple GPIO pins for flexible connection to peripherals such as LCDs, relays, and sensors. Its compact size, energy efficiency, and powerful performance make it well-suited for continuous operation in embedded and real-time systems. Additionally, its ability to handle both digital and analog signals allows it to manage a variety of inputs and outputs with high precision. Overall, the ESP32 is the central, intelligent controller that ensures seamless automation, communication, and system logic execution in this project.

3.3.2 Waterflow Sensor:

Fig 3.3.2

The water flow sensor is a key component for tracking the amount of water being consumed. It has a built-in turbine mechanism that rotates as water flows through it, producing electrical pulses. These pulses are sent to the ESP32, where they are counted and converted into liter's or milli litres using calibration logic. The sensor is typically placed in the pipeline and works on the principle of the Hall Effect. It is highly sensitive and provides accurate readings over a wide flow range. It plays a crucial role in defining when to cut off the water flow based on usage limits. The pulse rate increases or decreases depending on the flow speed, which allows the ESP32 to determine both rate and volume of usage. Without this sensor, the system cannot function for metering or alert generation. It is lightweight, durable, and easy to integrate with microcontrollers.

3.3.3 Relay Module:

The relay module is an electromechanical switch that is used to control high-power devices like water pumps or valves with low-power microcontroller signals. In this project, the relay is connected to the ESP32 and controls the flow of water based on the usage data. When the set limit of water is reached, the ESP32 sends a LOW or HIGH signal to the relay, turning off the water supply. Once the user completes the payment (via the QR code system), the ESP32 reactivates the relay and restores the water flow. This automation helps in reducing human intervention and encourages responsible water usage.

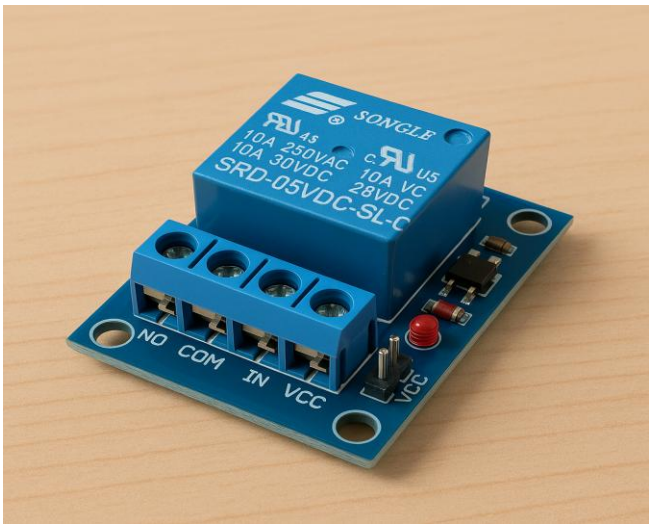


Fig 3.3.3

Relays used are typically 5V or 12V and capable of handling high current loads. The system ensures safe switching without manual operation and protects components from overcurrent.

3.3.4 LCD Display:

The LCD (Liquid Crystal Display) is used to display real-time data, alert messages, and the payment QR code. It helps bridge communication between the system and the user, especially when the mobile app is not accessible. In this system, a 16x2 or 20x4 LCD is used to show the current water usage and notify the user when the set limit is crossed. At that point, the LCD displays a QR code that can be scanned to pay and reactivate the system. The LCD is driven by the ESP32, usually using I2C protocol for easier wiring and control.

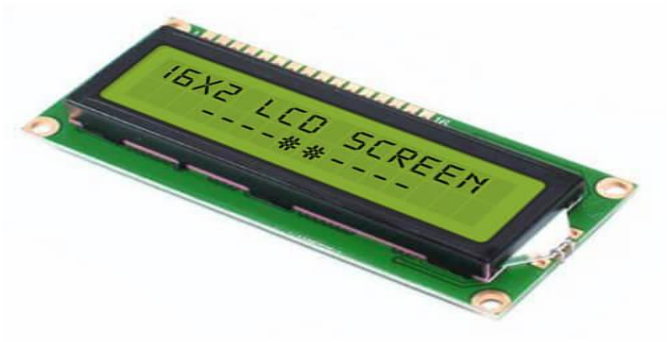


Fig 3.3.4

It updates dynamically as the water flows and ensures users are always informed of the system's status. It is low-power, low-cost, and an efficient visual communication tool for embedded systems.

3.3.5 Buffer circuit:

The buffer circuit is used to ensure voltage stability and protect the ESP32 from potential electrical noise or spikes coming from sensors. It acts as a signal stabilizer and voltage level translator. The buffer ensures that weak signals from the flow sensor are properly shaped and strong enough to be read by the ESP32 accurately. It prevents voltage mismatch and protects against reverse current that may damage sensitive microcontroller pins. In many cases, operational amplifiers (op-amps) or transistor-based buffers are used. By isolating input and output, the buffer maintains system integrity and improves overall reliability. It is especially important in real-time systems where sensor precision is critical. Using a buffer enhances the lifespan and accuracy of the sensor-microcontroller interface.

3.3.6 Power Supply Adapter:

The power supply adapter provides consistent and regulated voltage and current to power all modules in the system. Typically, a 5V or 12V DC adapter is used depending on the rating of the components like the relay and sensor. The ESP32 operates on 3.3V internally, so voltage regulation is done either through onboard regulators or external modules. A reliable power supply is essential for continuous operation without fluctuations or resets. It also prevents overheating or component damage due to overvoltage. The adapter connects to a wall socket and supplies power through a barrel jack or USB. It is important to check amperage ratings to match the total current requirements of the system. Clean and regulated power ensures that the system runs 24/7 without interruption.

3.3.7 Connecting Wires:

Connecting wires (male-male, male-female, or female-female jumper wires) are used to interconnect all components physically on a breadboard or PCB. They allow for flexible and

reusable prototyping. Good quality wires reduce signal loss, ensure proper voltage levels, and prevent short circuits. In this project, wires connect the ESP32 to

Fig 3.3.7

the flow sensor, relay, LCD, and power modules. Colour coding is often used to distinguish between power, ground, and data lines. Proper connection layout is important to avoid confusion and errors during testing or troubleshooting. While simple, jumper wires play a fundamental role in ensuring reliable circuit behaviour.

3.3.8 Buzzer: A **buzzer** is an electronic component used to generate sound alerts in various applications. It operates by converting electrical signals into audible tones, making it ideal for **notifications, alarms, and warnings** in embedded systems. Buzzers can be **piezoelectric or electromagnetic**, with piezo buzzers being more energy efficient. In a **groundwater monitoring system**, a buzzer can alert users when water consumption exceeds predefined limits, ensuring timely action to prevent wastage. Its **compact size, low power consumption, and reliability** make it a valuable addition to smart automation projects.

Fig 3.3.8

3.3.9 Breadboard:

A **breadboard** is a reusable platform used for **prototyping electronic circuits** without the need for soldering. It consists of a **grid of interconnected holes** where components like

resistors, sensors, and microcontrollers can be easily inserted and connected using jumper wires. Breadboards are widely used in **electronics development and testing**, allowing quick modifications and troubleshooting.

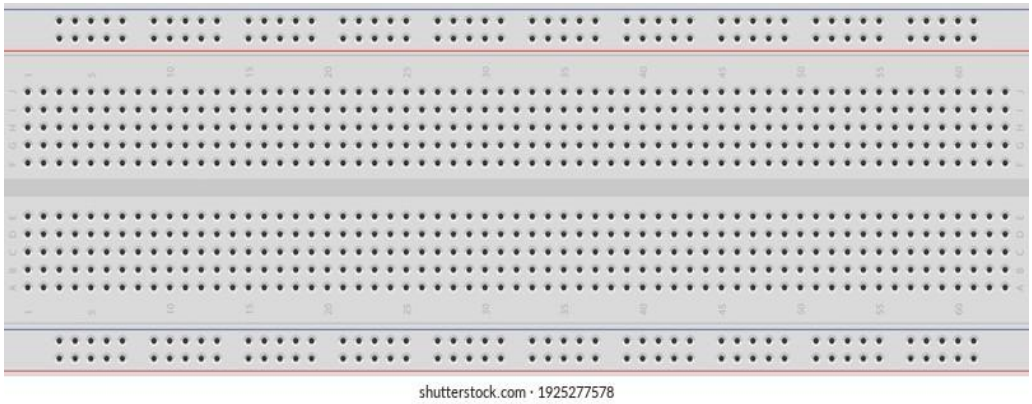


Fig 3.3.9

3.3.10 Wi-Fi Router:

The Wi-Fi router provides internet access, allowing the ESP32 to connect with the MQTT broker over a wireless network. This connection enables real-time data communication, pushing updates from the ESP32 to the cloud and retrieving control signals from the app. A stable Wi-Fi connection is essential for smooth operation of the monitoring system, especially for receiving alerts and viewing live consumption data.

3.3.11 MQTT Broker (Cloud-Based):

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol ideal for small devices like ESP32. In this system, the MQTT broker is hosted in the cloud (e.g., Mosquito) and acts as the communication hub. The ESP32 publishes messages such as water usage data, alerts, or QR code triggers, while the mobile app subscribes to these messages. It also receives responses like payment confirmations or control commands. MQTT is chosen due to its low bandwidth consumption, simplicity, and real-time performance. The broker allows secure, reliable, and asynchronous data exchange, enabling real-time dashboard updates and system responsiveness.

3.3.12 Smartphone with MQTT Dashboard App:

The smartphone functions as the user interface for real-time monitoring and interaction. With an MQTT dashboard app installed, the user can see live updates on water consumption, receive alerts, and scan the displayed QR code to make payments. Once payment is made, the app can send a control message via MQTT to restart the system. This setup eliminates the need for physical monitoring, offering full control at the user's fingertips.

CHAPTER 4

ESP-32 MICROCONTROLLER

4.1 Introduction to ESP32:

The **ESP32** is a powerful, low-cost microcontroller with integrated **Wi-Fi and Bluetooth** capabilities, designed by **Espress if Systems**. It is built for modern Internet of Things (IoT) applications that require wireless communication and fast processing. With dual-core processors, high-speed clock (up to 240 MHz), and low power consumption modes, the ESP32 stands out as a versatile platform for embedded projects. Its extensive I/O support, real-time performance, and onboard wireless stack make it ideal for smart home, automation, and sensor-monitoring systems like groundwater usage tracking.

4.2 Common Components of ESP32 Development Board:

1. **ESP32 SoC (System on Chip)** – The main chip, covered with a metal shield, performs all processing and communication tasks.
2. **Voltage Regulator** – Converts 5V input to 3.3V required for ESP32 operation.
3. **Micro USB Port** – Used for power supply and uploading code via a USB cable.
4. **Flash Memory** – Stores user programs and libraries.
5. **Boot and Reset Buttons** – Used to enter bootloader mode and restart the chip.
6. **Onboard Antenna** – Supports Wi-Fi/Bluetooth communication.
7. **Power LED** – Indicates the board is powered on.
8. **Pin Headers (GPIO)** – Connect external devices like sensors, displays, or motors.

4.2.1 Digital Pins:

The ESP32 has **30+ digital GPIO pins** which can be programmed as input or output. These pins are used for digital signal interfacing, like reading button states or controlling LEDs and relays. Most GPIOs support PWM, I2C, SPI, and UART functionalities.

- Example Digital Pins: GPIO0, GPIO2, GPIO4, GPIO5, etc.
- Voltage: Operates at **3.3V logic level**.

OVERVIEW OF ESP-32 MICROCONTROLLER:

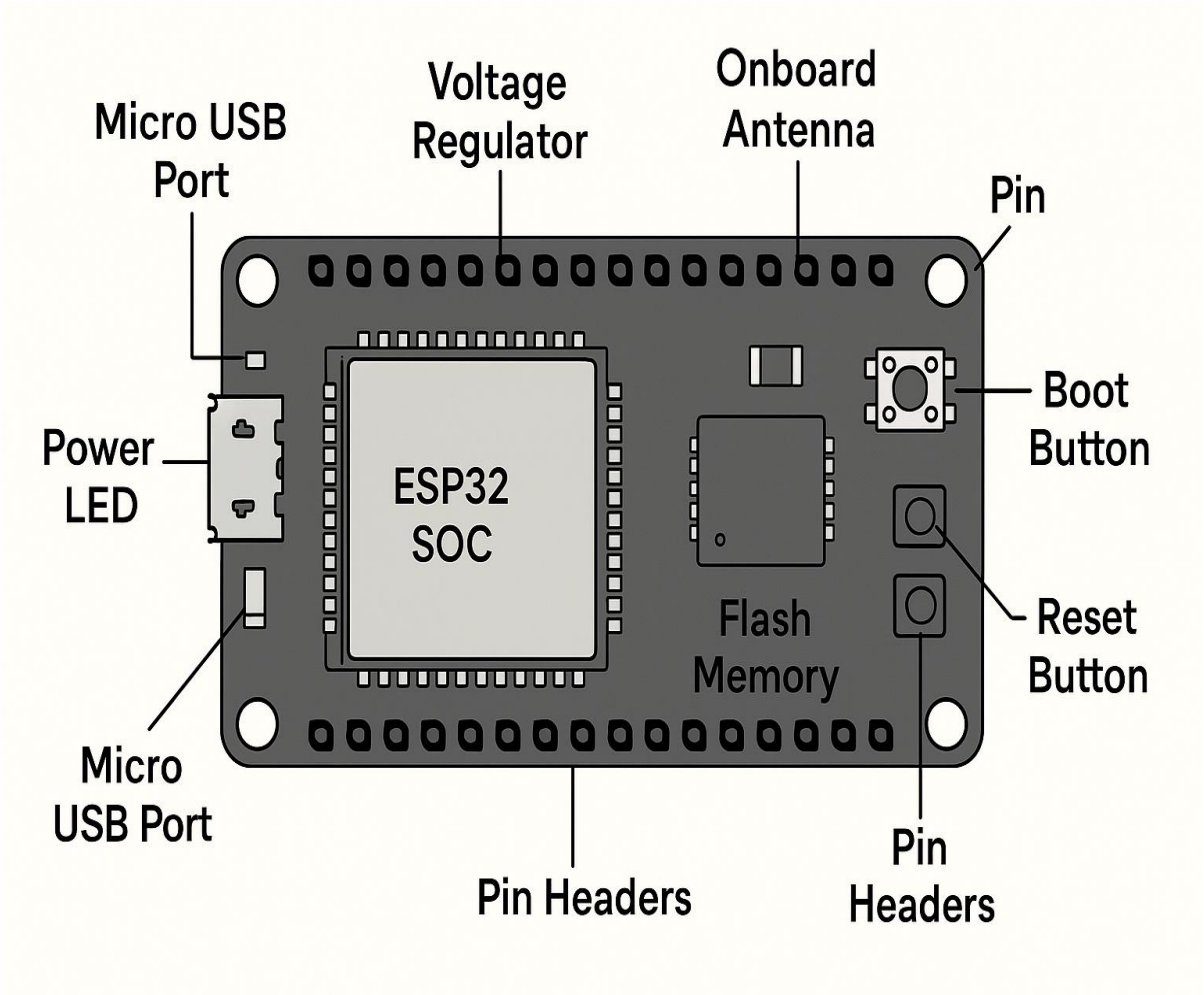


Fig-4.2 Common pins of ESP-32

 ESP32 SoC (System on Chip):
This is the central chip of the board, responsible for all processing tasks. It includes built-in Wi-Fi and Bluetooth modules and is covered with a metal shield for protection.

 Voltage Regulator:
Converts the 5V input from USB or other sources to 3.3V, which is the required voltage for the ESP32 to operate safely.

 Micro USB Port:
Used to power the board and upload code from a computer. It also enables serial communication with the ESP32.

 Flash Memory:
Stores user-written code, libraries, and system firmware. It allows the ESP32 to retain programs even after power is removed.

 Boot and Reset Buttons:
The boot button is used to enter programming mode. The reset button restarts the microcontroller without disconnecting power.

 Onboard Antenna:
Enables wireless communication by supporting Wi-Fi and Bluetooth. It's essential for IoT applications.

 Power LED:
A small indicator that lights up when the board is powered on, confirming that it is receiving power.

 Pin Headers (GPIO):
Rows of pins that connect external components like sensors, relays, and LCDs. They support both digital and analog input/output.

4.3 Analog Pins

ESP32 includes **18 channels** of **12-bit ADC** (Analog to Digital Converter), allowing it to read analog voltages like those from sensors.

- Example Analog Pins: GPIO32 to GPIO39.
- Used for: Reading sensor values like temperature, voltage, or flow rate from analog-output sensors.

4.4 Power Pins

- **3V3 Pin** – Provides regulated 3.3V output.
- **GND Pins** – Ground connection.
- **VIN Pin** – Input voltage (5V) from USB or external adapter.

These pins power the board and external modules connected to it.

4.5 Software Used

- **Arduino IDE** – The most common and beginner-friendly platform to program ESP32 boards.
- **ESP32 Board Manager** – Installed in Arduino IDE to enable ESP32 support.
- **Libraries** – Includes WiFi, PubSubClient (for MQTT), LiquidCrystal_I2C (for LCD), etc.
- **Other Tools** – PlatformIO, MicroPython, and ESP-IDF (official framework) can also be used for advanced applications.

CHAPTER 5

MQQT

5.1 MQTT Broker:

An MQTT broker is the central part of the publish-subscribe communication system. It receives all messages published by clients and routes them to appropriate subscribers. The broker maintains topic lists and manages connections from multiple clients simultaneously. It ensures efficient data delivery, even in cases where network bandwidth is limited. Brokers can support quality of service (QoS), retained messages, and secure communication via SSL/TLS. Mosquito is one of the most commonly used open-source MQTT brokers. In your project, the broker acts as a middleman between the ESP32 microcontroller and the mobile application dashboard. Whenever the ESP32 sends data about water usage, the broker forwards that information to the mobile app. Similarly, commands from the app (like restarting flow after payment) are sent back to ESP32 through the broker. Without a broker, the direct communication between devices would be complicated and less efficient.

5.2 MQTT Client:

An MQTT client can be any device or software application that connects to the broker to either publish or subscribe to messages. In this project, the ESP32 acts as a client by publishing data such as water flow status and consumption levels. Likewise, your mobile app also acts as an MQTT client, subscribing to updates and sending control commands. Clients must establish a TCP connection to the broker and remain connected to send or receive messages. They also support Last Will and Testament (LWT) messages for unexpected disconnections. The advantage of using MQTT clients is that they allow real-time, bi-directional communication without needing constant polling or complex setup. Each client has a unique ID, and secure communication can be maintained through authentication settings on the broker.

5.3 MQTT Topics:

Topics in MQTT define channels or paths for message delivery. They act like folders in a file system, organized in a hierarchical structure separated by slashes (e.g., "water/flow/status"). When a client publishes a message on a topic, any client subscribed to that topic receives the message. Topics are flexible and can include wildcards (+ or #) for broad subscriptions. For instance, subscribing to "water/flow/#" would receive messages from all subtopics under that hierarchy. In your project, topics are used to send flow rate, alert messages, or control signals. Properly structured topics ensure clarity, modularity, and easy scaling of the system.

5.4 OVERVIEW OF MQTT:

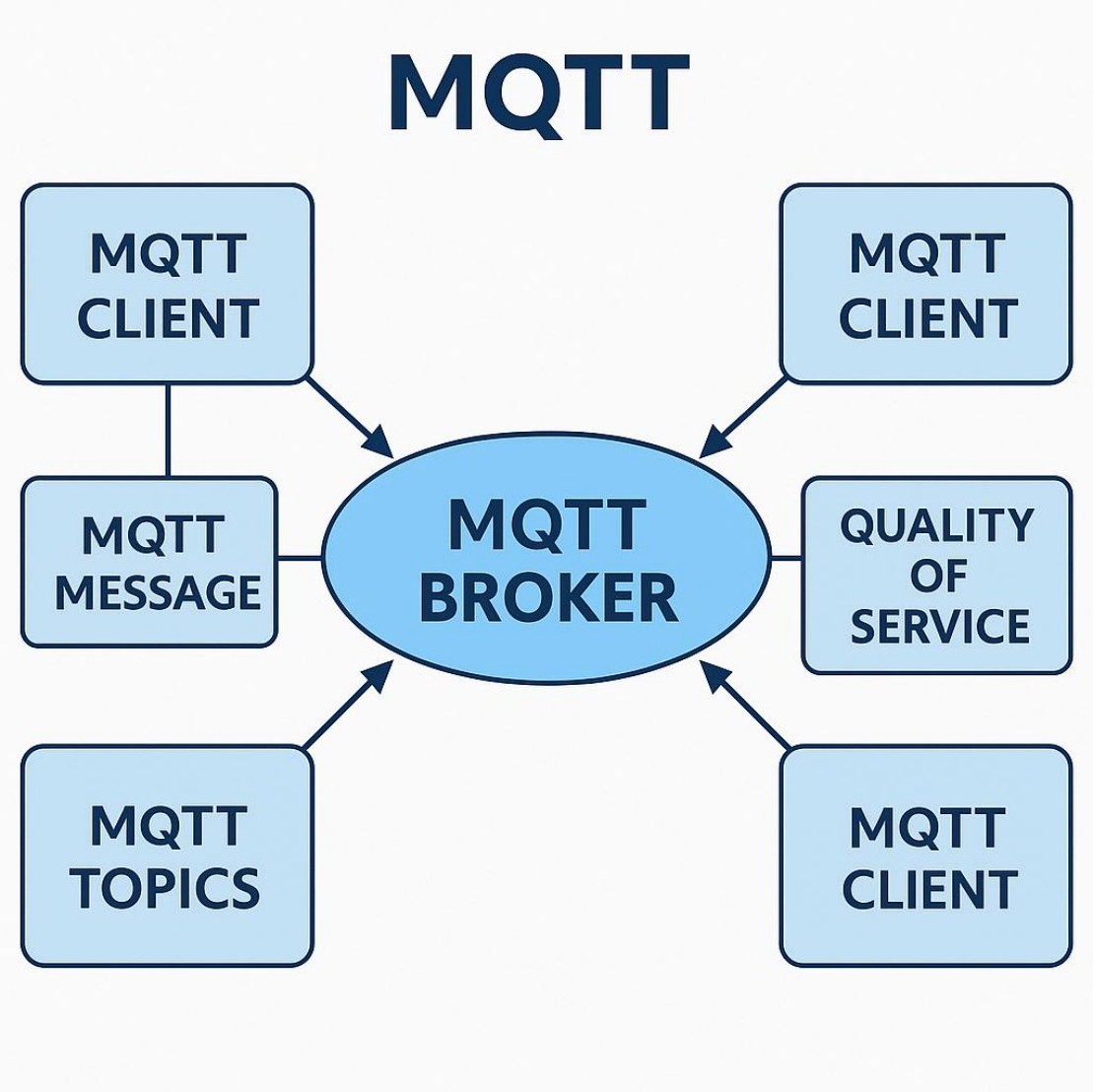


Fig:5.4 Function of MQTT

5.5 MQTT Message:

An MQTT message contains the actual payload (data) that is transmitted from the publisher to the subscriber. The message can be in any format — text, JSON, or binary — and includes metadata such as topic, QoS level, and retain flag. In your project, messages might carry values like "Water flow: 3.2 L/min" or "Limit reached." These messages are published by ESP32 and displayed on the mobile app. Messages can be retained so that new subscribers get the last known value immediately. This makes MQTT highly efficient for IoT systems that need regular, lightweight, and reliable updates.

5.6 Quality of Service (QoS):

MQTT supports three levels of QoS to control message delivery reliability. **QoS 0** ensures the message is delivered at most once, ideal for non-critical data. **QoS 1** delivers the message at least once, which may cause duplicates but ensures reliability. **QoS 2** delivers messages exactly once, the most reliable but with the highest overhead. In your project, QoS 1 may be suitable for most updates to ensure that critical messages (like alerting water usage limits) are not lost. Using the right QoS level helps balance performance and reliability depending on the message's importance.

5.7 Applications of MQTT:

1. **IoT and Smart Home Automation** – Used for connecting sensors, devices, and applications with minimal bandwidth.
2. **Real-time Data Monitoring** – Suitable for live dashboards that show sensor data like water flow, temperature, etc.
3. **Remote Industrial Monitoring** – Monitors machinery, pipelines, or energy systems from afar.
4. **Healthcare Monitoring Systems** – Transmits patient data securely and efficiently in wearable or home-care setups.
5. **Agricultural Automation** – Controls irrigation systems or monitors soil/water levels in smart farming.
6. **Logistics and Tracking Systems** – Communicates location or cargo status in transport and fleet management.
7. **Energy Monitoring** – Tracks usage of electricity or water in buildings and industries.
8. **Smart Cities** – Integrated in traffic control, water management, pollution monitoring, etc.
9. **Educational Projects** – Used by students for IoT-based academic prototypes and learning.
10. **Mobile Messaging Apps (like Facebook Messenger's backend)** – Supports real-time messaging infrastructure.

5.8 Advantages of MQTT:

- **Lightweight Protocol** – Ideal for low-bandwidth, high-latency environments.
- **Low Power Usage** – Efficient for battery-powered devices like ESP32.
- **Simple Architecture** – Publish/Subscribe model is easy to set up and scale.
- **Reliable Communication** – Supports QoS levels to ensure message delivery.
- **Bi-directional Communication** – Both sending and receiving possible with the same client.
- **Real-time Updates** – Enables immediate alerts and actions.
- **Supports Offline Messages** – Retained messages and LWT help manage disconnections.
- **Highly Scalable** – Can connect thousands of devices across a network.
- **Cross-Platform Compatibility** – Works on mobile, PC, microcontrollers, cloud, etc.
- **Open-Source Broker (Mosquitto)** – Easy to set up and integrate into projects.

5.9 Disadvantages of MQTT:

- **Requires Internet or Network Access** – Can't operate without Wi-Fi or mobile data.
- **Not Suitable for Large Data Transfers** – Best for small messages, not media or files.
- **Security Risks If Not Configured** – Lacks built-in encryption unless SSL/TLS is added.
- **Complex Authentication** – Advanced security requires manual configuration of username/password or certificates.
- **Limited Built-in Debugging** – Troubleshooting needs external tools like MQTT.fx or Mosquitto logs.

CHAPTER 6

NODE-RED

6.1 Introduction

Node-RED is a flow-based programming platform designed to integrate hardware devices, APIs, and online services. It provides a visual programming environment through a browser-based editor where users can wire together devices and services using pre-built nodes. In this project, Node-RED is utilized to monitor and control underground water usage data in real time. It interacts with the ESP32 via the MQTT protocol to display data and control the water flow mechanism effectively.

6.1.1 Why Node-RED is Used

Node-RED is selected for this project due to its flexibility, ease of use, and compatibility with MQTT-based IoT systems. It simplifies the process of creating a responsive user interface without writing HTML or JavaScript code manually. It allows real-time visualization of sensor data and remote control of hardware components, making it ideal for applications like prepaid water monitoring and community water usage management.

The implementation begins with setting up Node-RED on a system with Node.js installed. The MQTT-in nodes in Node-RED subscribe to topics such as water/flow, water/total, and water/status which are published by the ESP32. These values are then passed to dashboard nodes like text display, gauge, and chart. Function nodes are used to compare water usage against a limit and publish control messages like "DISABLE" to the water/control topic. This message is then received by ESP32 to turn off the relay. This loop creates a real-time interactive control system.

6.2 Components Used in Node-RED Setup

- **ESP32 Microcontroller** – Publishes flow and status data using MQTT.
- **Water Flow Sensor** – Measures real-time water consumption.
- **Relay Module** – Controls the water supply based on MQTT commands.
- **Mosquitto MQTT Broker** – Manages topic-based message communication.
- **Node-RED (with Dashboard Plugin)** – Displays data and allows user interaction.
- **Wi-Fi Network** – Ensures communication between ESP32 and Node-RED system.
- **Arduino IDE** – Used to upload the ESP32 code.

6.3 NODE-RED Dashboard:

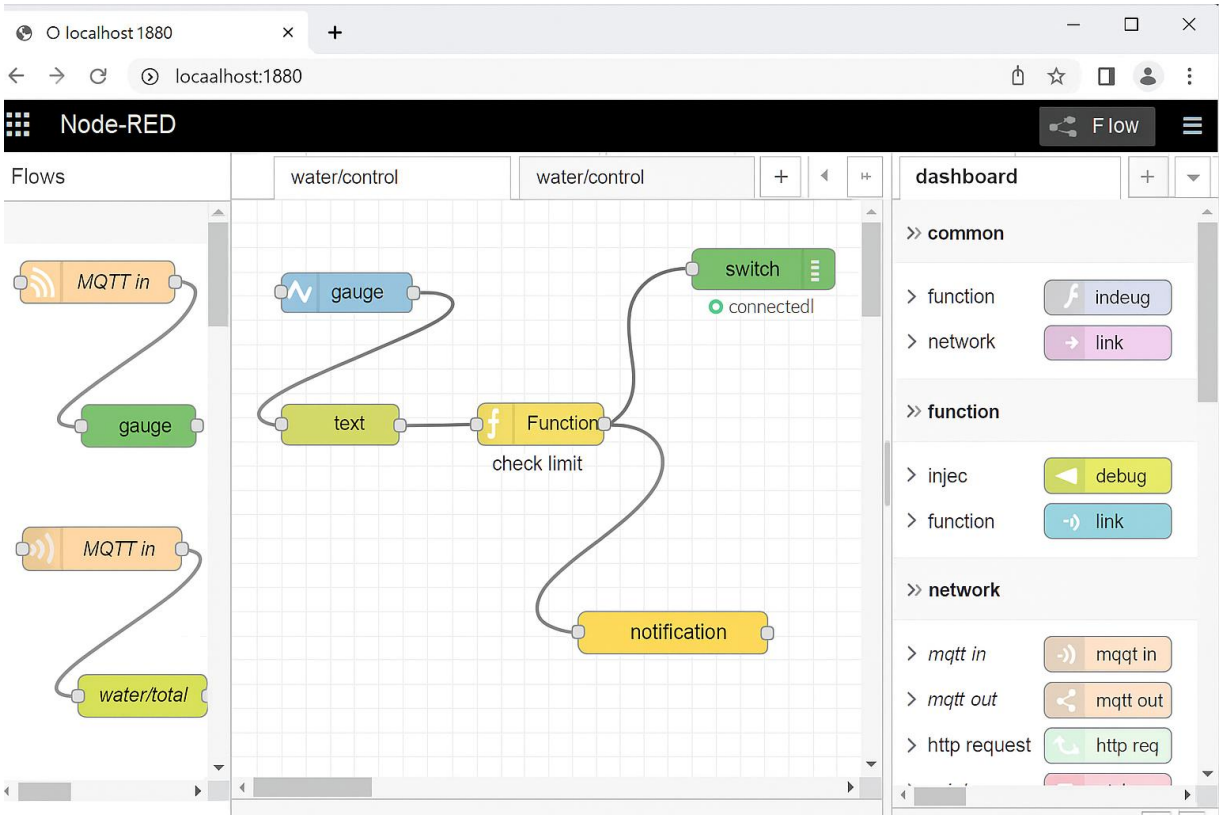


Fig-6.2 Node-Red Dashbooard

6.3 Applications of Node-RED :

- Remote water usage monitoring by users or authorities.
- Implementation of a prepaid water system with auto cutoff.
- Monitoring and control of groundwater usage in rural or urban communities.
- Integration with payment gateways via QR code and notifications.
- Extension to agriculture or industrial domains for water management.

6.4 Key Features of Node-RED in This Project

- **Real-time Flow Rate Monitoring** – via MQTT-in and gauge nodes.
- **Usage Display** – using text nodes subscribed to total usage topic.
- **Relay Control** – controlled via MQTT-out nodes based on usage conditions.
- **Auto Cut-off Logic** – implemented using function nodes.
- **Dashboard Interface** – developed using the node-red-dashboard plugin.
- **Live Notifications** – alerts when consumption crosses the threshold.
- **Secure MQTT Communication** – between ESP32 and Node-RED.
- **Expandable Logic** – can add cloud logging, payment API, or SMS alerts.

6.5 Advantages

- Simplified visual programming with drag-and-drop nodes.
- Live data display and control via browser dashboard.
- Supports MQTT, HTTP, serial, and cloud services.
- Reduces time and effort needed for frontend development.
- Easily customizable and extensible.
- Platform-independent — works on Windows, Linux, Raspberry Pi, etc.
- Provides real-time alerts and decision-making support.

6.6 Disadvantages

- Requires an always-on system to run Node-RED continuously.
- Not ideal for highly complex data visualization tasks.
- Basic dashboard UI — not as rich as custom-developed web apps.
- Limited offline support for mobile dashboards.
- Slight learning curve for advanced flows using function nodes.

6.7 ESP32 API (Arduino IDE Environment):

The ESP32 API provided in the Arduino IDE is a set of libraries and functions that help in controlling the ESP32 hardware. It includes:

1. **WiFi.h**: Used to connect ESP32 to a Wi-Fi network.
2. **Arduino.h**: Core library that supports general Arduino functions like pinMode(), digitalWrite(), and delay().
3. **HardwareSerial.h**: For communication through UART (Serial ports).
4. **EEPROM.h**: To store and retrieve non-volatile data (useful for settings or limits).
5. **Wire.h**: For I2C communication with components like LCDs.
6. **SPI.h**: To communicate with SPI devices if needed.
7. **PWM** and **ADC** APIs: Used for analog input from flow sensors or output control signals.

ESP32's APIs allow GPIO control, analog/digital reads and writes, serial communication, and timing. They make it possible to control flow sensors, relays, and LCDs easily through code.

6.8 MQTT API (PubSubClient Library):

The MQTT communication on ESP32 is usually done using the **PubSubClient** library. It provides functions to connect to an MQTT broker, publish data, and subscribe to topics. Key APIs include:

1. client.setServer(broker, port): Sets MQTT broker IP and port.
2. client.connect("clientID"): Connects ESP32 to MQTT broker using a unique client ID.
3. client.publish("topic", "message"): Publishes a message to a topic.

4. client.subscribe("topic"): Subscribes ESP32 to a specific topic to receive data.
5. client.setCallback(callbackFunction): Links a function to handle received messages.
6. client.loop(): Keeps connection alive and checks for incoming messages.
7. client.connected(): Returns whether ESP32 is connected to the broker.

These functions enable real-time communication with a mobile dashboard or MQTT platform like Mosquito.

6.9 Program Structure for ESP32 with MQTT:

Here is a typical structure of your program in the Arduino IDE:

1. **Include Libraries**

cpp

Copy code

#include <WiFi.h>

#include <PubSubClient.h>

2. **Define WiFi & MQTT Credentials**

cpp

Copy code

const char* ssid = "yourSSID";

const char* password = "yourPASS";

const char* mqtt_server = "broker.hivemq.com"; // or your Mosquitto IP

3. **Initialize WiFi and MQTT Clients**

cpp

Copy code

WiFiClient espClient;

PubSubClient client(espClient);

4. **Setup Function**

cpp

Copy code

void setup() {

Serial.begin(115200);

WiFi.begin(ssid, password);

```
client.setServer(mqtt_server, 1883);
client.setCallback(callback);
}
```

5. **Loop Function**

```
cpp
Copy code
void loop() {
if (!client.connected()) {
reconnect();
}
client.loop();

// Read water sensor value
int flow = analogRead(flowSensorPin);
String msg = String(flow);
client.publish("water/flow", msg.c_str());
}
```

6. **Callback Function**

```
cpp
Copy code
void callback(char* topic, byte* message, unsigned int length) {
// Handle messages received
}
```

7. **Reconnect Function**

```
cpp
Copy code
void reconnect() {
while (!client.connected()) {
if (client.connect("ESP32Client")) {
client.subscribe("water/control");
```

}

}

}

6.9 Sketch Description:

sql

Copy code

[Water Flow Sensor]

ESP32 MCU

[Analog Input] - Flow Sensor Data

[GPIO Pins] - Controls Relay

[Wi-Fi Module] - Sends data to MQTT

[MQTT Broker (e.g., Mosquitto)]

[Mobile App] [Dashboard/Web App]

(Receives alerts) (Monitors real-time flow)

Main Interconnections in the Sketch

- **Flow Sensor**: Connected to ESP32 analog input (reads water usage).
- **Relay**: Connected to ESP32 GPIO (to stop water if limit exceeded).
- **Wi-Fi**: ESP32 connects to router and communicates with MQTT broker.
- **MQTT Broker**: Acts as middleman, transferring messages between ESP32 and the app.
- **Mobile App/Dashboard**: Displays water consumption and shows alerts/payment QR.

6.10 Software Tools:

1. Arduino IDE

- **Purpose**: Used to write, compile, and upload code to the ESP32 microcontroller.
- **Why Used**: It supports ESP32 board extensions and offers a user-friendly interface for embedded programming.
- **Features**:
- Serial monitor for debugging sensor outputs.
- Easily integrates with libraries like WiFi.h and PubSubClient.h.

- Offers community support and extensive documentation.
- **Compatibility**: Windows, macOS, Linux.

2. MQTT Broker (Mosquitto)

- **Purpose**: Acts as the central hub for communication between ESP32 and the mobile dashboard using the MQTT protocol.
- **Why Used**: Lightweight and ideal for real-time IoT applications.
- **Features**:
- Supports publish-subscribe model.
- Runs locally or on the cloud.
- Compatible with MQTT client tools and dashboards.

3. MQTT.fx / MQTT Explorer (Optional Tools)

- **Purpose**: GUI-based MQTT client used for testing, debugging, and subscribing to topics.
- **Why Used**: Helps you visualize messages sent by ESP32 and verify the topics are correct.
- **Features**:
- Easy topic management.
- Real-time message tracking.

4. Mobile Dashboard Application (e.g., Blynk, IoT MQTT Panel, etc.)

- **Purpose**: Displays water usage, alert messages, and QR code for payment.
- **Why Used**: Interfaces with MQTT broker to receive real-time data from ESP32.
- **Features**:
- Custom UI with graphs and buttons.
- Push notifications when a limit is reached.
- QR code integration (via link or image widget).

5. Fritzing (for Circuit Design – optional)

- **Purpose**: To draw neat circuit diagrams for presentation/documentation.
- **Why Used**: Visually represents how ESP32, relay, sensors, and LCD are connected.

6.11 WORKFLOW:

1. Power ON and Initialization

- When powered on, the ESP32 initializes all peripherals:
    - Flow sensor input pin
    - Relay output pin
    - LCD display (if any)
    - Serial communication
- Wi-Fi credentials are loaded and WiFi.begin(ssid, password) is called.
- It attempts to connect to the local Wi-Fi network.

6.12 Arduino IDE:

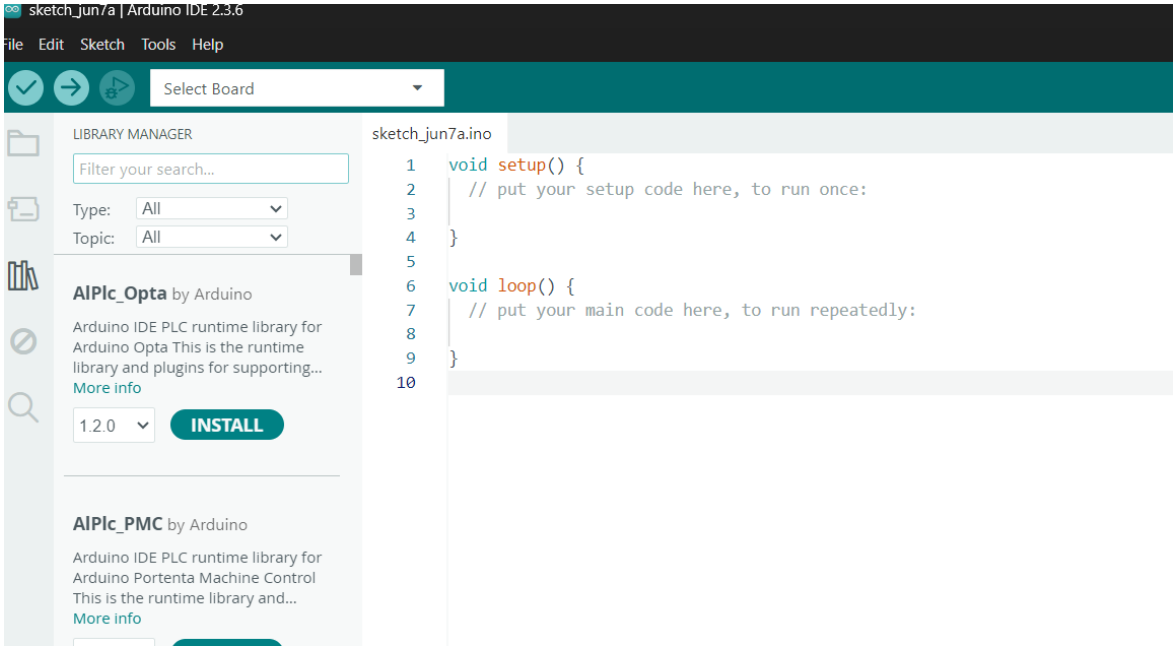


Fig 6.12.1 The classic Arduino

6.12.2 Cloud Editor:

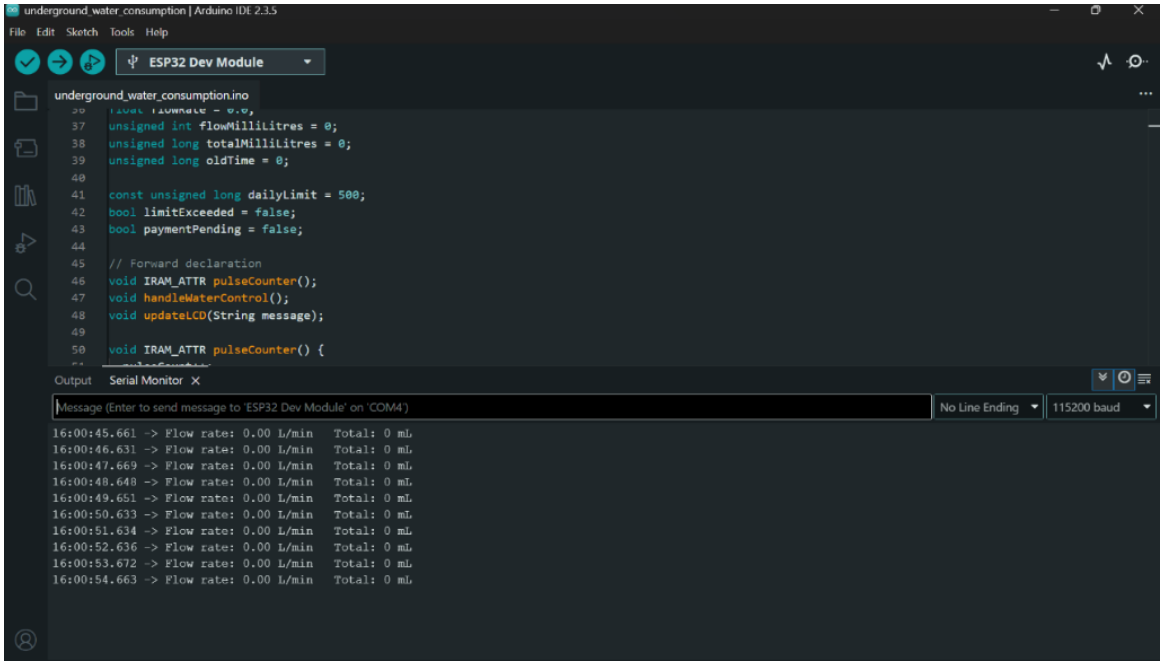


Fig 6.12.2 cloud editor

6.12.3 Serial Monitor:

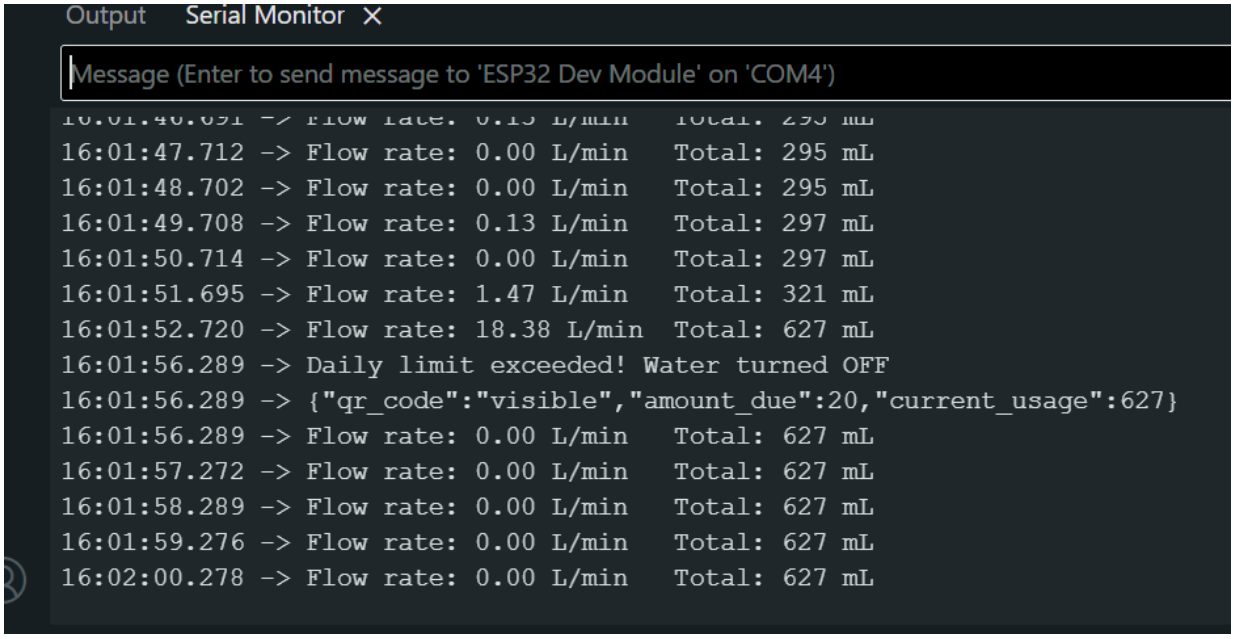


Fig 6.12.3 serial monitor

6.13 MQTT Explorer:

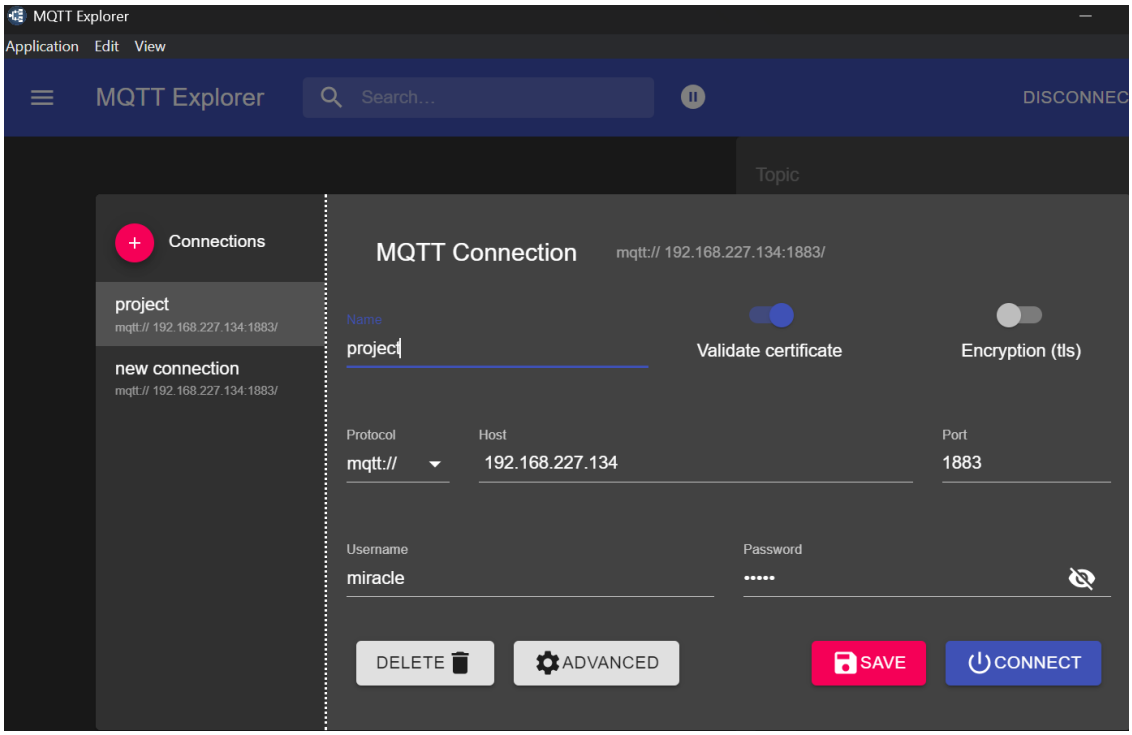


Fig 6.13 MQTT Dashboard

6.14 NODE-RED Dashboard:

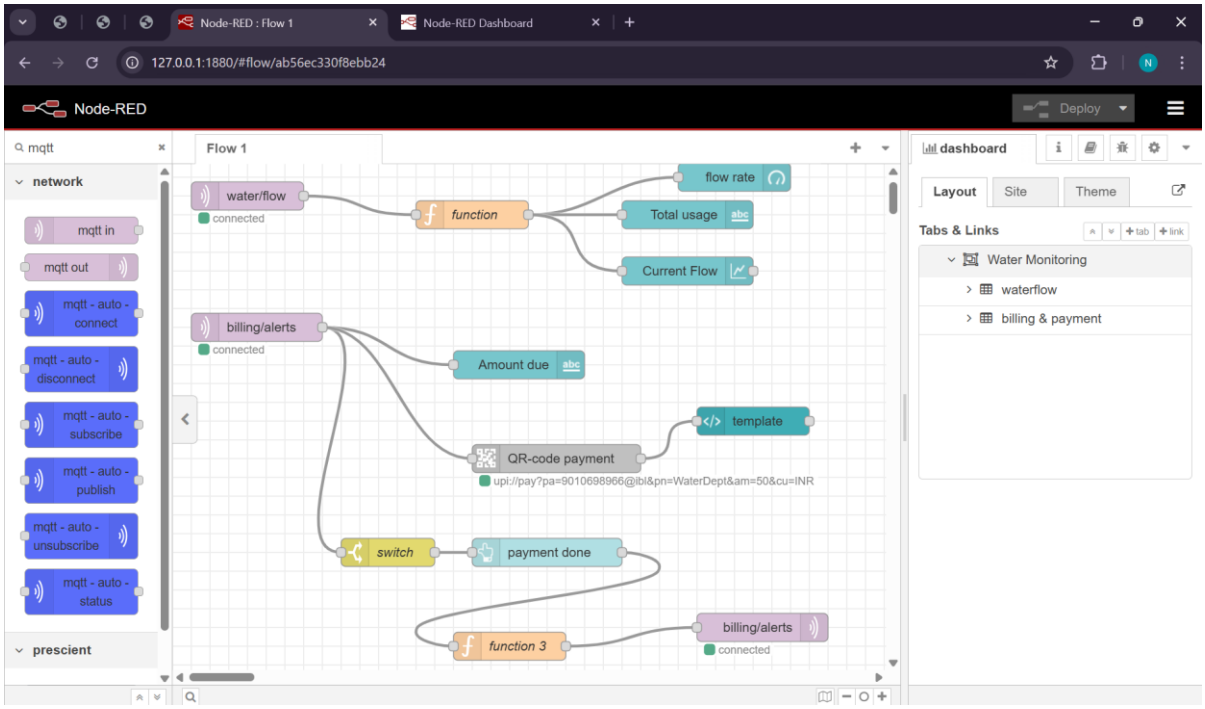


Fig 6.14 node-red dashboard

CHAPTER 7

RESULT AND DISCUSSION

The proposed groundwater consumption monitoring system was successfully implemented and tested using the ESP32 microcontroller, a flow sensor, relay, and MQTT protocol integrated with a mobile dashboard. The system actively monitored the flow of water in real-time, displayed the consumption data on an app, and enforced usage limits with automated control mechanisms. During testing, the **ESP32** efficiently captured flow rate data from the sensor and computed total consumption accurately. This data was transmitted to the **MQTT broker**, which reliably pushed updates to the **mobile application dashboard**, offering live tracking of water usage. The interface provided real-time visualization in both numerical and graphical formats, enhancing user awareness.

When the water usage reached the predefined threshold, the system automatically triggered a **relay** to cut off the water supply. Simultaneously, the user received an **alert notification** on the mobile app with a **QR code for payment**. Once payment was simulated or verified, the app sent a signal back to the ESP32 via MQTT, which then **reactivated the relay**, resuming water flow. This cycle proved effective in **enforcing water usage limits and promoting conservation**.

The system was tested under different flow rates and durations, and it consistently provided **accurate readings and responsive control actions**. The **LCD display** showed useful on-device information, such as current flow, Wi-Fi status, and operational messages, improving ease of use and diagnostics.

One of the most notable outcomes was the **seamless integration of MQTT communication**, which provided low-latency, reliable data exchange between the hardware and the mobile platform. The user experience was enhanced by immediate feedback and remote accessibility.

The system proved to be **low-cost, scalable, and adaptable**, suitable for rural and urban settings alike. It highlights how IoT technology can contribute to **sustainable water management practices** by making underground water consumption **visible, accountable, and controllable**.

Overall, the project demonstrated both **functional success and societal relevance**, aligning with the objective of addressing water wastage and encouraging mindful consumption through technology-driven solutions.

7.1 THE PROJECT CODE:

```
#include <WiFi.h>
#include <PubSubClient.h>

// Wi-Fi credentials
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// MQTT Broker settings
const char* mqtt_server = "broker.hivemq.com"; // Or your local broker IP
const char* flowTopic = "water/flow";
const char* alertTopic = "water/alert";
const char* controlTopic = "water/control";

WiFiClient espClient;
PubSubClient client(espClient);

// Pins
const int flowSensorPin = 34;   // Analog or interrupt-based pin
const int relayPin = 27;        // Relay control pin

// Flow sensor variables
volatile int pulseCount = 0;
float flowRate = 0.0;
float totalLitres = 0.0;
unsigned long oldTime = 0;
const float calibrationFactor = 4.5; // Based on your sensor spec
// Threshold limit
const float waterLimit = 5.0; // 5 Litres
```

```
// Function to handle incoming MQTT messages
void callback(char* topic, byte* payload, unsigned int length) {
  String messageTemp;
  for (int i = 0; i < length; i++) {
    messageTemp += (char)payload[i];
  }
  if (String(topic) == controlTopic) {
    if (messageTemp == "restart") {
      digitalWrite(relayPin, LOW); // Turn ON water supply
      totalLitres = 0;             // Reset usage
    }
  }
}

// Interrupt service routine (ISR) for flow sensor pulses
void IRAM_ATTR pulseCounter() {
  pulseCount++;
}

// Connect to Wi-Fi
void setup_wifi() {
  delay(10);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }
}

// Connect to MQTT
void reconnect() {
```

```
  while (!client.connected()) {
    if (client.connect("ESP32Client")) {
      client.subscribe(controlTopic);
    } else {
      delay(2000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(flowSensorPin, INPUT_PULLUP);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW); // Water ON initially
  attachInterrupt(digitalPinToInterrupt(flowSensorPin), pulseCounter, RISING);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  unsigned long currentTime = millis();
  if (currentTime - oldTime > 1000) { // Every 1 second
    detachInterrupt(digitalPinToInterrupt(flowSensorPin));

    flowRate = ((1000.0 / (currentTime - oldTime)) * pulseCount) / calibrationFactor;
    float litresPassed = (flowRate / 60.0); // Convert to litres per second
```

```
  totalLitres += litresPassed;
  Serial.print("Flow rate: ");
  Serial.print(flowRate);
  Serial.print(" L/min\tTotal: ");
  Serial.print(totalLitres);
  Serial.println(" L");
  // Publish to MQTT
  char msg[50];
  sprintf(msg, "Usage: %.2f Litres", totalLitres);
  client.publish(flowTopic, msg);

  // Alert and cut off water if limit exceeded
  if (totalLitres >= waterLimit) {
    digitalWrite(relayPin, HIGH); // Cut OFF water
    client.publish(alertTopic, "Water limit exceeded. Payment required.");
  }

  pulseCount = 0;
  oldTime = currentTime;
  attachInterrupt(digitalPinToInterrupt(flowSensorPin), pulseCounter, RISING);
}}
```

7.2 OUTPUTS:

Intial stage:

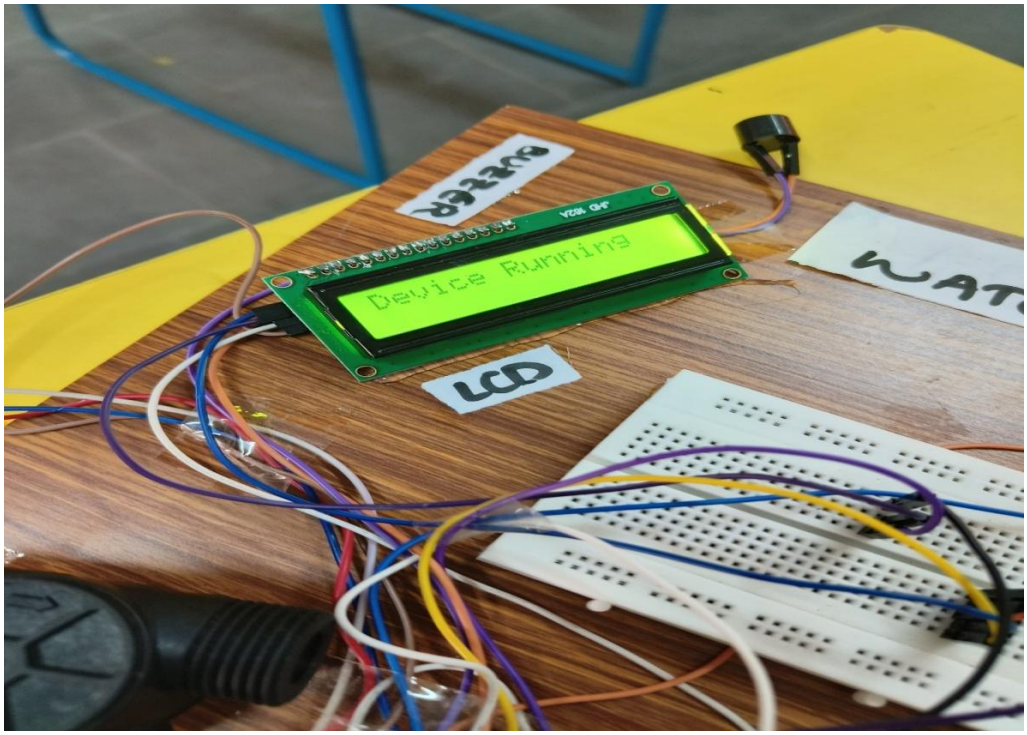


Fig 7.2.1

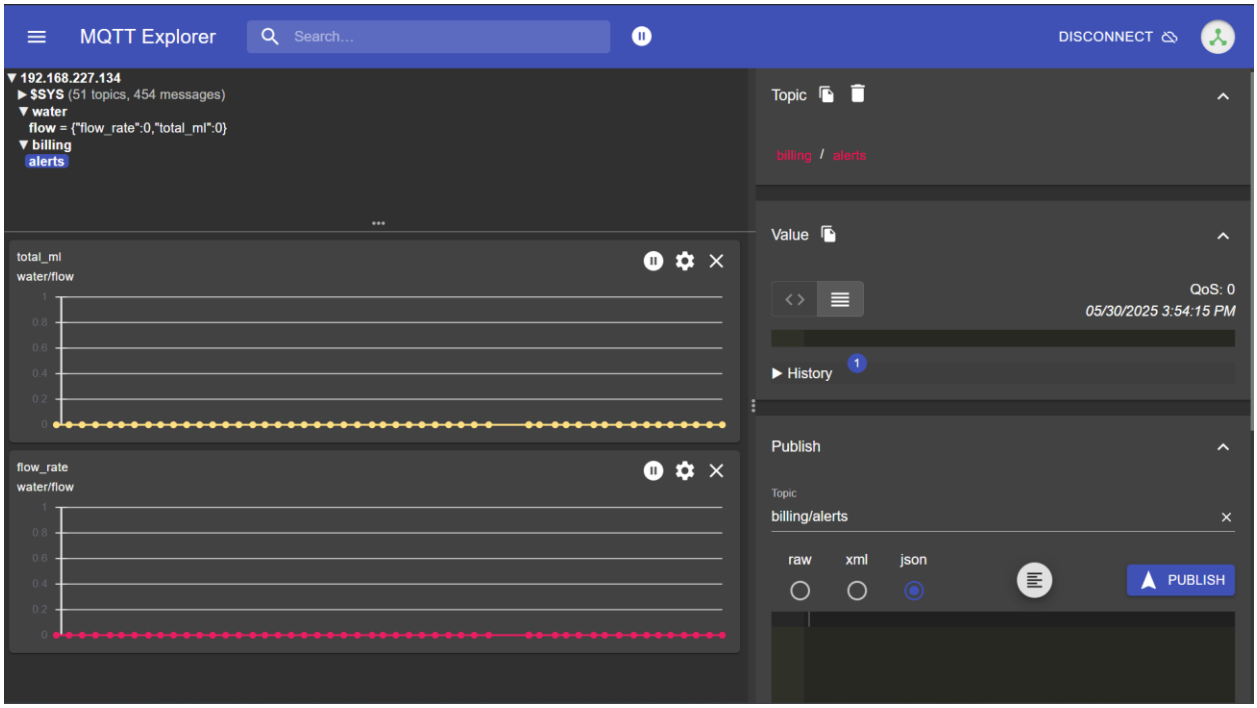


Fig 7.2.2

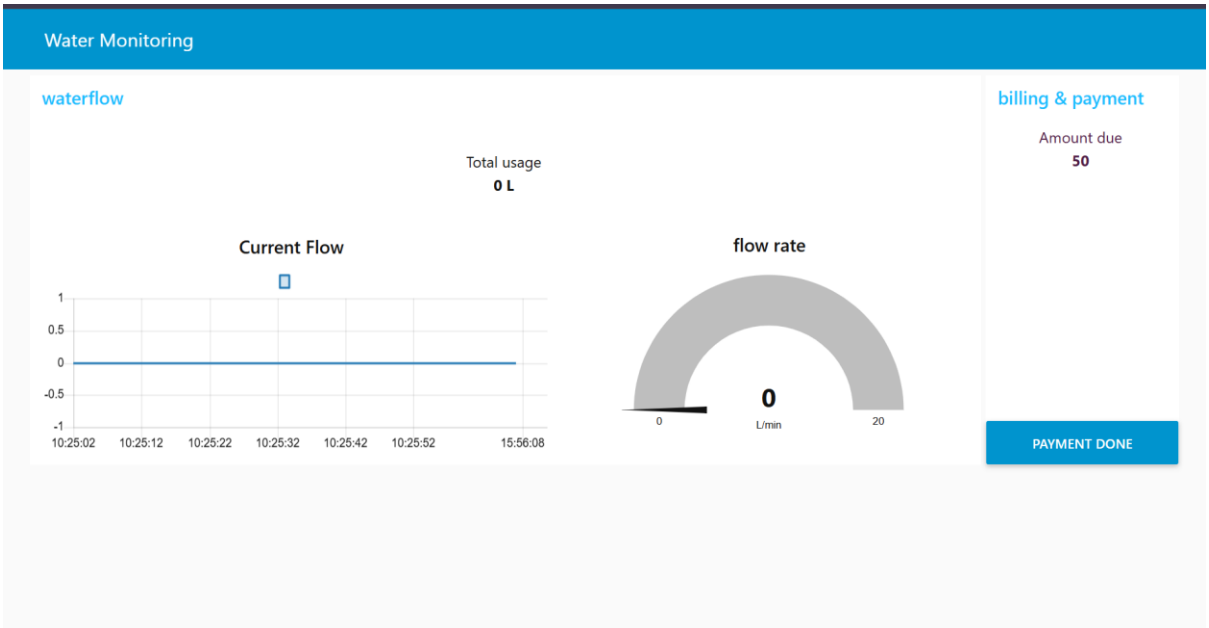


Fig 7.2.3

Intermediate stage:

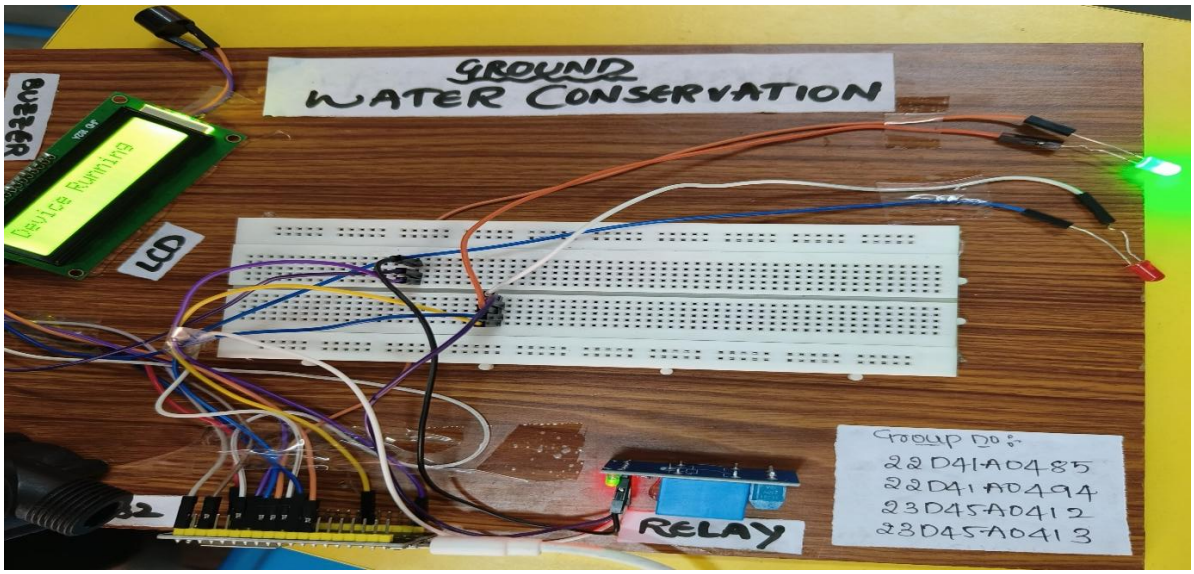


Fig 7.2.4

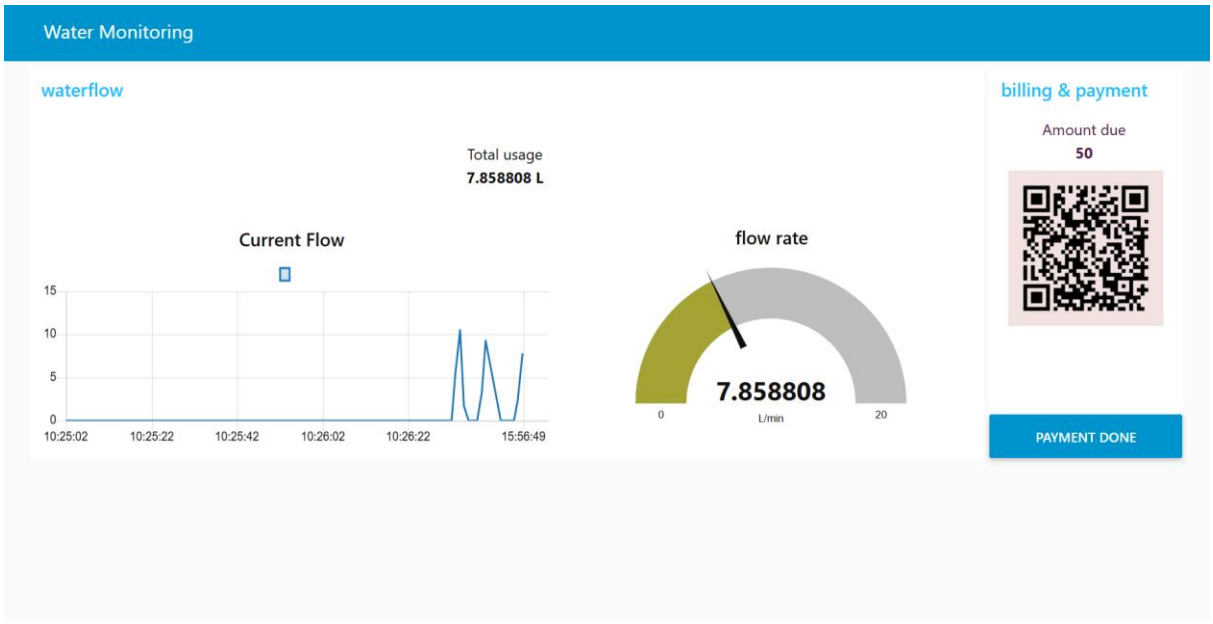


Fig 7.2.5

Final stage:

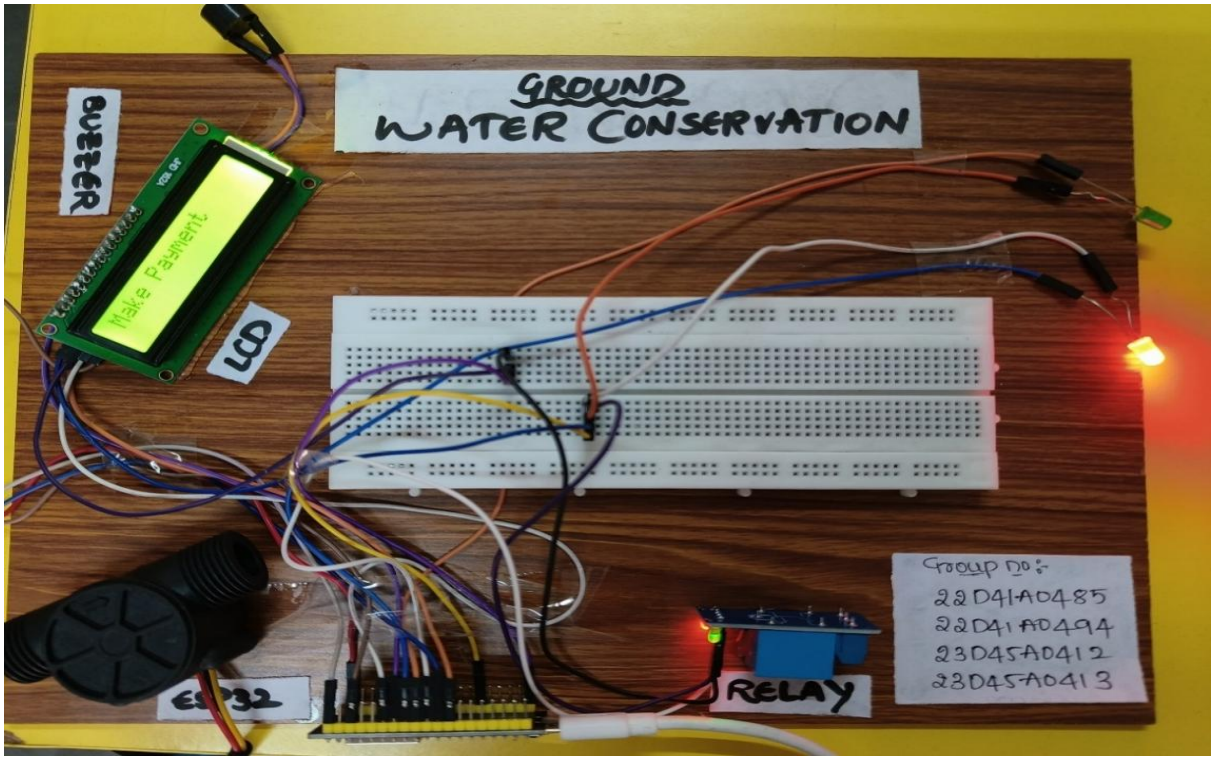


Fig 7.2.6

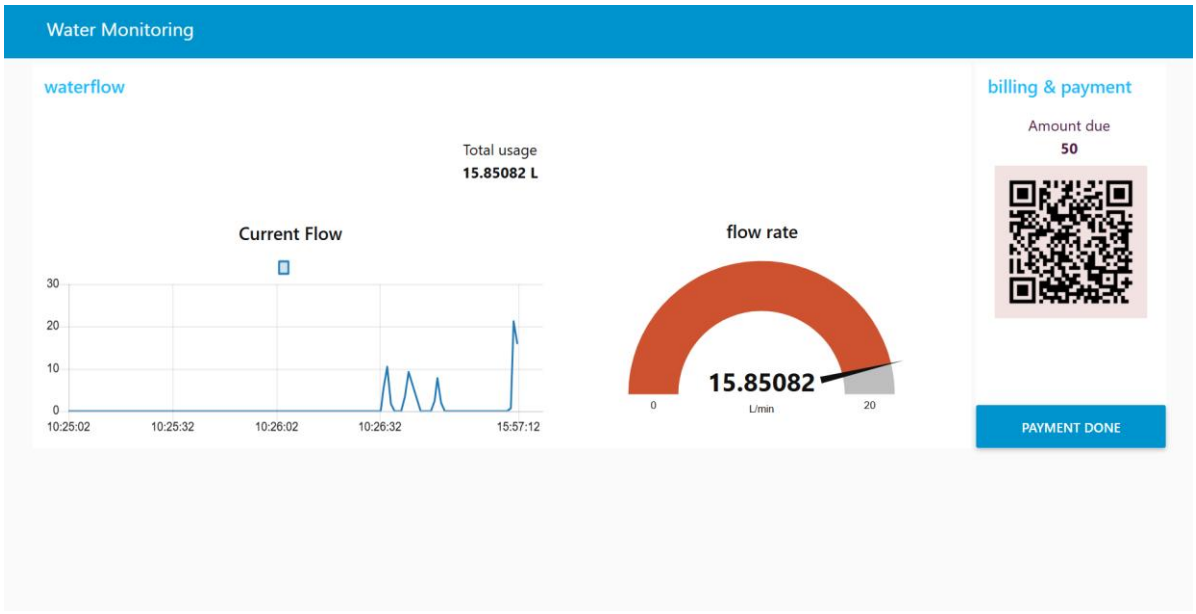


Fig 7.2.7

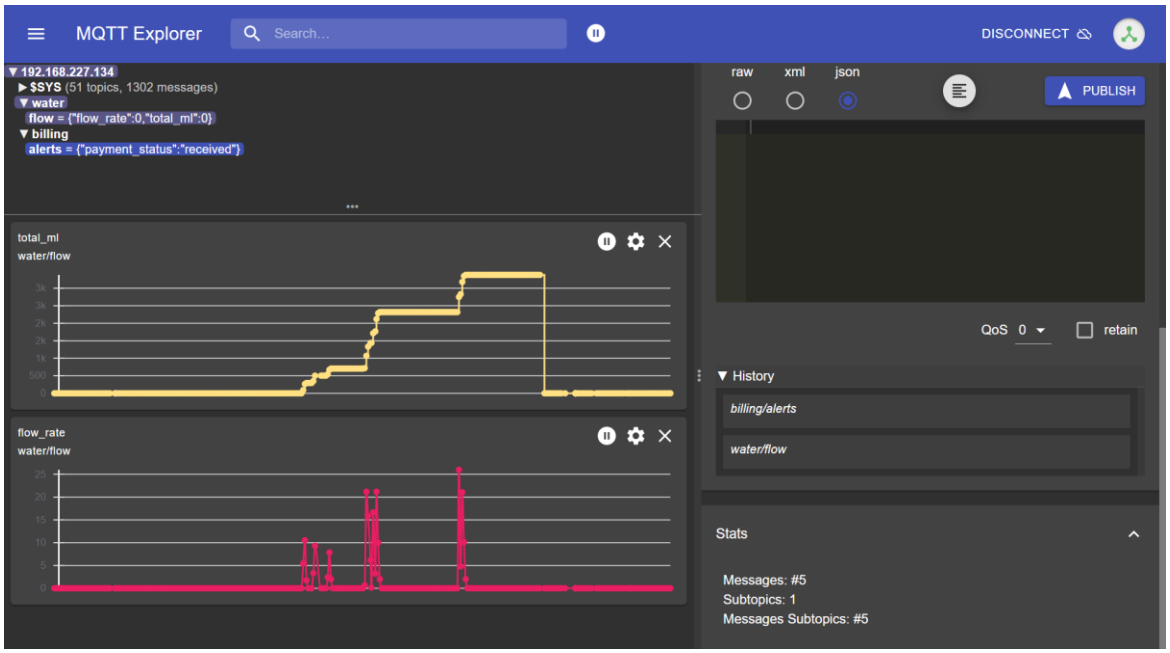


Fig 7.2.8

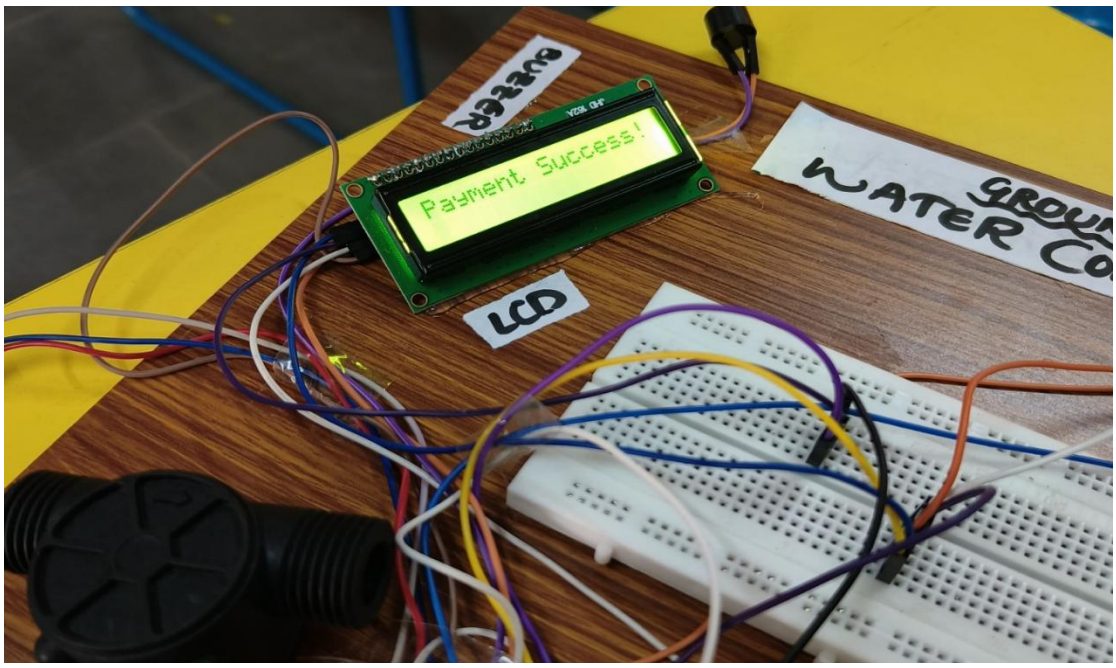


Fig 7.2..9

CONCLUSION

This project aimed to develop a real-time groundwater monitoring and consumption control system using an ESP32 microcontroller, flow sensor, relay, and MQTT communication. The core functionality involved tracking the volume of water consumed, displaying it on a mobile dashboard, and enforcing a usage limit. When the predefined limit was exceeded, the system automatically cut off the water flow and displayed a QR code for payment to resume supply. The MQTT protocol effectively enabled seamless communication between the ESP32 and the mobile application. The system demonstrated real-time responsiveness, automation, and the potential to manage water usage in domestic and community-based applications. Its implementation highlights the importance of technological interventions in water conservation.

- **Key benifits/ Achievements**

The following key outcomes were successfully achieved during the implementation of the system:

- Real-time monitoring of groundwater usage via a flow sensor and ESP32.
- Automatic cut-off of water flow using a relay upon exceeding the limit.
- Instant message alerts and usage updates delivered through MQTT to a mobile app.
- QR-based payment integration to re-enable flow after limit exhaustion.
- Mobile dashboard support for user-friendly interaction and monitoring.
- LCD/OLED display (optional) for on-device real-time visibility.
- Compact and modular design using minimal hardware.
- Low power consumption and effective use of ESP32 features.
- Code uploaded via Arduino IDE with stable performance.
- Demonstrated potential for extension to rural and urban water systems.

Societal Impact:

Water scarcity, especially in Indian regions like Telangana, Bangalore, and Chennai, has reached alarming levels. This system provides a practical solution to monitor and control underground water consumption, helping prevent misuse and over-extraction. It empowers users by visualizing their consumption and enforcing accountability through automated restrictions and feedback. The system's ability to stop flow when the limit is reached acts as a real-time deterrent to wastage. In areas with shared water resources, such systems promote equitable use. By reducing unnecessary usage and enabling fair distribution, the project contributes to sustainable water conservation goals. Its simplicity and affordability make it a viable solution for wide deployment in communities, schools, apartments, and farmlands.

References

The study by **Patel et al. (2020)** in the *International Journal of Scientific Research in Engineering and Management* highlighted the role of IoT in water level detection using flow sensors and microcontrollers like ESP32. **Kumar and Sharma (2019)** presented an MQTT-based communication framework in *IEEE Access*, emphasizing its efficiency in real-time water monitoring applications. **R. Singh et al. (2021)** provided insight into rural groundwater usage patterns and the importance of decentralized monitoring systems. Furthermore, tutorials and projects from platforms like *Circuit Digest*, *Electronics Hub*, and *Instructables* contributed to the practical implementation of ESP32 interfacing and MQTT dashboard creation. These works collectively served as the theoretical and practical backbone for the system's design, flow, and societal relevance.

 SP32 Datasheet – Espressif Systems. https://www.espressif.com/en/products/socs/esp32

 MQTT Protocol Specification – OASIS Open

. https://mqtt.org/

 YF-S201 Water Flow Sensor – Technical Specifications. https://www.electronicwings.com

 Arduino IDE Documentation – Arduino.cc.

https://docs.arduino.cc/

 Mosquitto MQTT Broker Setup – Eclipse Mosquitto.

https://mosquitto.org/

 Smart Water Monitoring Projects – ResearchGate, various authors.
https://www.researchgate.net

 Real-Time Water Usage Tracking Using IoT – IEEE Conference Papers.

 Groundwater Crisis in India – Central Ground Water Board (CGWB), Government of India.

http://cgwb.gov.in

 Adafruit ESP32 Feather V2 Tutorial – Adafruit Learning System.
https://learn.adafruit.com

 Implementation of IoT-Based Water Management – Journal of Environmental Science and Engineering, Volume XX, 2023.