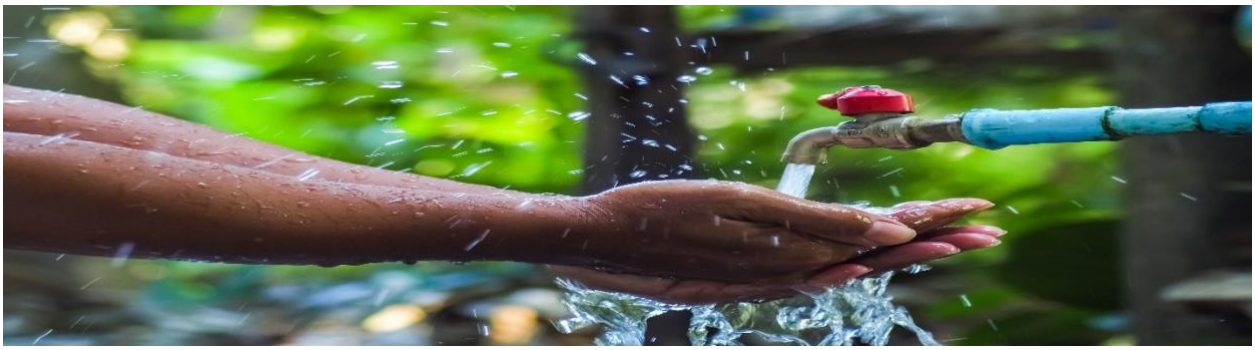
Smart water management using IoT



SUBMITTED BY:

P.ABINAYA

C.SAI PRAVEENA

R.MOHANA

K.VIJAYALAKSHMI....

OBJECTIVES :

1. Efficient Resource Allocation: Implement IoT to monitor water usage and distribution in real-time, enabling efficient allocation of water resources based on demand.
2. Detection and Prevention: Utilize IoT sensors to detect leaks in water infrastructure and trigger immediate alerts for maintenance, reducing water.
3. Quality Monitoring: Implement sensors to continuously monitor water quality parameters, ensuring safe and clean drinking water for consumers.
4. Forecasting: Use IoT data to predict future water demand patterns, enabling proactive planning for supply and infrastructure adjustments.
5. Engagement: Develop IoT-based applications to provide consumers with real-time water usage data, encouraging responsible water consumption.
6. Impact Reduction: Optimize water treatment processes using IoT to reduce energy consumption and minimize the environmental h.
7. Control and Automation: Enable remote control of water systems for adjusting water flow, pressure, and treatment processes as needed.
8. Monitoring and Early Warning: Implement IoT for flood detection and early warning systems to mitigate the impact of extreme weather events
9. Data analytics and Insights: Leverage IoT-generated data for analytics to identify trends, anomalies, and opportunities for further water conservation.
10. with Smart Cities: Integrate smart water management systems with broader smart city initiatives to enhance overall urban sustainability.

Smart water management using Esp32 code programming:

Include <WiFi.h>

```
#include <WiFiClient.h>

#include <WebServer.h>



Const char* ssid = “Your_SSID”;

Const char* password = “Your_Password”;



Const int moistureSensorPin = 34; // Analog pin for soil moisture sensor

Const int pumpRelayPin = 26; // Digital pin for controlling water pump



Webserver server(80);



Void setup() {

 pinMode(pumpRelayPin, OUTPUT);

 digitalWrite(pumpRelayPin, LOW);
```

```
 Serial.begin(115200);
 WiFi.begin(ssid, password);

 While (WiFi.status() != WL_CONNECTED) {
  Delay(1000);
  Serial.println("Connecting to WiFi...");
 }

 Server. On("/", HTTP_GET, handleRoot);
 Server.begin();
}

Void loop() {
 Server.handleClient();
}

Void handleRoot() {
 Int moistureLevel = analogRead(moistureSensorPin);
```

```
String webpage = "<html><body>";

Webpage += "<h1>Smart Water Management</h1>";

Webpage += "<p>Moisture Level: " + String(moistureLevel) + "</p>";



If (moistureLevel < 500) { // Adjust the threshold as per your needs

 Webpage += "<p>Watering the plants...</p>";

 digitalWrite(pumpRelayPin, HIGH); // Turn on the water pump

 delay(5000); // Run the pump for 5 seconds (adjust as needed)

 digitalWrite(pumpRelayPin, LOW); // Turn off the water pump

} else {

 Webpage += "<p>No need for watering.</p>";

}



Webpage += "</body></html>";



Server.send(200, "text/html", webpage);

}
```

CODING PROGRAM USING ESP32:

```
Import machine
Import time

# Pin assignments for the ultrasonic sensor
TRIGGER_PIN = 23  # GPIO23 for trigger
ECHO_PIN = 22     # GPIO22 for echo

# Pin assignment for the LED
LEAK_LED_PIN = 19  # GPIO19 for the LED

# Set the pin modes
Trigger = machine.Pin(TRIGGER_PIN, machine.Pin.OUT)
Echo = machine.Pin(ECHO_PIN, machine.Pin.IN)
Leak_led = machine.Pin(LEAK_LED_PIN, machine.Pin.OUT)

# Function to measure distance using the ultrasonic sensor
Def measure_distance():
   # Generate a short trigger pulse
   Trigger.value(0)
   Time.sleep_us(5)
   Trigger.value(1)
   Time.sleep_us(10)
   Trigger.value(0)
```

```
    # Measure the echo pulse duration to calculate distance
    Pulse_start = pulse_end = 0
    While echo.value() == 0:
        Pulse_start = time.ticks_us()
    While echo.value() == 1:
        Pulse_end = time.ticks_us()

    Pulse_duration = pulse_end – pulse_start

    # Calculate distance in centimeters (assuming the speed of sound is 343 m/s)
    Distance = (pulse_duration * 0.0343) / 2  # Divide by 2 for one-way travel

    Return distance

# Function to check for a water leak
Def check_for_leak():
    # Measure the distance from the ultrasonic sensor
    Distance = measure_distance()

    # Set the threshold distance for detecting a leak (adjust as needed)
    Threshold_distance = 10  # Adjust this value based on your tank setup

    If distance < threshold_distance:
        # If the distance is less than the threshold, a leak is detected
        Return True
    Else:
```

```
    Return False


# Main loop
While True:
  If check_for_leak():
    # Blink the LED to indicate a leak
    Leak_led.value(1)  # LED ON
    Time.sleep(0.5)
    Leak_led.value(0)  # LED OFF
    Time.sleep(0.5)
  Else:
    Leak_led.value(0)  # LED OFF


  Time.sleep(1)  # Delay between measurements
```

WEB APPLICATION:

1. **Hardware Setup:**
   First, you'll need to deploy IoT sensors and devices to collect data on water usage, quality, and other relevant parameters. These might include flow meters, water quality sensors, and control valves.
2. **Data Collection:**
   Set up these IoT devices to collect data and transmit it to a central server or cloud platform. This data can include water flow rates, temperature, water quality, and more.

3. **Data Storage:**
   Design a database to store the collected data securely. You might consider using databases like MySQL, PostgreSQL, or cloud-based options like AWS RDS or Azure SQL.
4. **Data Processing:**
   Implement data processing and analytics to derive valuable insights from the collected data. For instance, you can detect leaks, predict maintenance needs, and optimize water usage.
5. **User Interface:**
   Develop a web-based user interface where users can monitor and control the water management system. You can use web development technologies such as HTML, CSS, and JavaScript for this purpose.
6. **Real-time Monitoring:**
   Implement real-time data visualization, so users can see current water usage, system status, and alerts. Consider using libraries like D3.js or charting frameworks like Chart.js.
7. **Alerts and Notifications:**
   Set up a notification system to alert users or administrators in case of anomalies, such as leaks or water quality issues. You can use email, SMS, or push notifications for this.
8. **User Authentication and Authorization: Implement user accounts with secure authentication and role-based access control to ensure that only authorized personnel can control the system.**
9. **Remote Control:**
   Enable users to remotely control water devices and systems through the web application. This might include shutting off water supply, adjusting flow rates, or scheduling operations.
10. **Data Analytics and Reports:**
    Provide data analytics tools and reporting features, allowing users to view historical data, trends, and insights that can help them make informed decisions.
11. **Security:**
    Ensure the application is secure to protect against data breaches and unauthorized access. Use encryption, secure APIs, and follow best practices in web application security.
12. **scability:**
    Design the application to scale with the growth of your IoT network. Cloud services like AWS and Azure can help with this.
13. **Maintainance and Updates:**
    Plan for ongoing maintenance and updates to keep the system running smoothly and to incorporate new features or address security vulnerabilities.
14. **Compliance:**

Be aware of and adhere to any regulatory and compliance requirements related to water management and IoT data.

15. **Testing:**
    Thoroughly test the system to ensure its reliability and performance. This should include unit testing, integration testing, and user acceptance testing.
16. **Documentation and Training:**
    Provide documentation for users and maintenance personnel. Training might also be necessary to ensure effective use of the system.
17. **Feedback and Improvement:**
    Continuously gather feedback from users and use it to improve the system over time.

PYTHON CODING:

```
Import random

Class WaterTank:
    Def __init__(self, capacity):
        Self.capacity = capacity
        Self.level = 0

    Def fill(self, amount):
        Self.level = min(self.capacity, self.level + amount)

    Def use(self, amount):
        If self.level >= amount:
            Self.level -= amount
            Return True
        Return False

Class SmartWaterManager:
    Def __init__(self, tank_capacity):
        Self.water_tank = WaterTank(tank_capacity)

    Def monitor_water_level(self):
        Return self.water_tank.level
```

```
    Def simulate_water_usage(self):
        Usage = random.randint(0, 10)
        Return usage

    Def control_water_management(self):
        Current_usage = self.simulate_water_usage()
        If self.water_tank.use(current_usage):
            Print(f"Water used: {current_usage} units")
        Else:
            Print("Water shortage: Not enough water to meet demand.")
            Self.water_tank.fill(10)  # Refill the tank by 10 units

If __name__ == "__main__":
    Tank_capacity = 50  # Set the tank capacity
    Manager = SmartWaterManager(tank_capacity)

    For _ in range(10):  # Simulate 10 time steps
        Manager.control_water_management()
        Current_level = manager.monitor_water_level()
        Print(f"Current water level: {current_level} units\n")
```

CONCLUSION:

In conclusion, smart water management using IoT, with the implementation of the ESP32 microcontroller, offers a promising solution to address the challenges of water conservation and efficient utilization. By integrating IoT technology, sensor networks, and data analytics, this system enables real-time monitoring, data collection, and control of water resources. It can help detect leaks, optimize water distribution, and promote sustainable water usage. Additionally, the ESP32's low-power capabilities make it a suitable choice for battery-operated sensors in remote or off-grid areas, enhancing the system's versatility. Overall, the adoption of IoT and the ESP32 in water management contributes to a more sustainable and efficient use of this critical resource.