# SMART WATER MANAGEMENT

## FINAL PROJECT SUBMISSION

## IOT PHASE 5

# Titles

- **Project definition**
- **Problem of Water scarcity**
- **Intro to Smart water management**
- **Challenges**
- **Design Thinking**
- **Technologies**
- **Benefits**
- **Innovations**
- **Water Level Detecting (Wokwi Simulator)**
- **Circuit Diagram**
- **Connecting Mobile App With Smart Water Management IOT**
- **Conclusion**

## Project Definition :

Objective:

- The project aims to promote water conservation by implementing IoT sensors for real-time water consumption monitoring in public places.

Key Goals:

- Monitor water usage in real-time.
  Raise public awareness about water conservation.
  Enable sustainable resource management.
  Scope: Public places such as parks and gardens

## Intro To Smart water Management :

- Smart water management combines advanced technologies with traditional water management strategies to optimize resource usage and minimize water wastage.
- By leveraging IoT, water systems can be monitored and managed in real-time, enabling proactive decision-making for sustainable water management.
- Smart water management relies on sensors and Internet of Things (IoT) devices strategically placed throughout water infrastructure. These sensors collect real-time data on water quality, quantity, pressure, and usage patterns.

## Challenges:

- Despite the benefits, implementing smart water management systems requires overcoming certain challenges.
- Common obstacles include high upfront costs, complex data integration, cybersecurity concerns and resistance to adopting new technologies.

- Overcoming these challenges is crucial for successful implementation and long-term sustainability.

## Design Thinking :

Project Objectives:

- Real-time Monitoring: Develop a system to continuously monitor waterconsumption.
- Public Awareness: Create a platform to share consumption data with the public.
- Water Conservation: Implement alerts and notifications for excessive water use.
- Resource Management: Analyze data to identify trends and optimize water usage.

## IoT Sensor Design:

Sensor Selection:

- Choose appropriate IoT sensors for water consumption measurement.

Deployment Strategy:

- Determine sensor placement in public areas.

Data Accuracy:

- Ensure sensors provide accurate consumption data.

Power Management:

- Plan for sensor power supply and energy-efficient operation.

## Real-Time Data Platform:

Development**:**

- Create a data-sharing platform for public access.

User Interface**:**

- Design a user-friendly mobile app to display real-time data.

Data Presentation**:**

- Visualize consumption data in a comprehensible manner.

User Engagement:

- Include features for user feedback and reporting.

## Project Execution Plan :

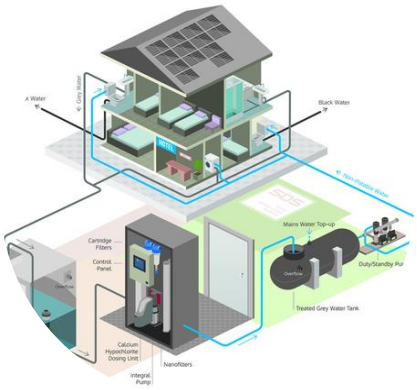
- Conduct a thorough site survey to identify suitable locations for sensor deployment.
- Procure and install IoT sensors according to the deployment plan.
- Develop the mobile app and data-sharing platform with real-time data display and user interaction features.
- Implement data transmission and processing mechanisms. Conduct user testing and gather feedback for refinement.
- Launch the IoT water consumption monitoring system in public places.
- Continuously monitor and maintain the system, addressing any issues promptly

## TECHNOLOGIES :

- Water technologies based on IoT encompass a wide range of solutions and innovations aimed at managing, conserving, purifying, and distributing water resources efficiently.

- Some key technologies of water technologies based on IoT include

## Greywater :




- Greywater recycling based on IoT (Internet of Things) involves using sensors, automation, and data analytics to efficiently collect, treat, and reuse greywater (wastewater from sinks, showers, and laundry) in a sustainable manner.
- Implementing greywater recycling with IoT requires careful design, integration, and adherence to local regulations and water quality standards. It can contribute to more sustainable and efficient water use in residential, commercial, and industrial settings.

## Smart water meters :

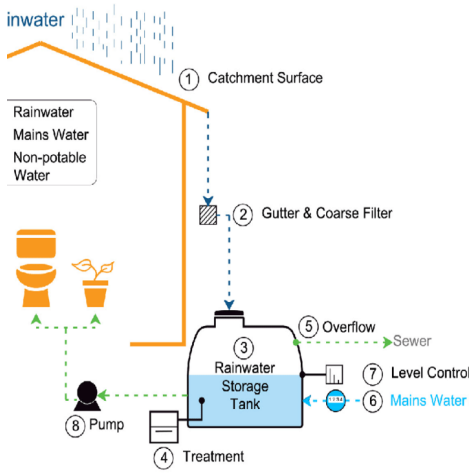
- Smart water meters based on IoT (Internet of Things) technology are designed to provide more accurate and efficient monitoring and management of water consumption.
- smart water meters based on IoT technology offer numerous benefits, including improved accuracy, efficiency, conservation, and cost savings for both utility providers and consumers.
- They play a crucial role in modernizing water management and ensuring the sustainable use of this essential resource.

## Smart leakage detection :


- Smart Leakage Detection based on IoT refers to a system that uses Internet of Things (IoT) technology to monitor and detect leaks or abnormal water flow in various applications, such as plumbing systems, pipelines, or industrial processes.
- This system typically involves the deployment of sensors and devices that can collect real-time data on water usage, pressure, and flow.
- When anomalies or leaks are detected, the system can immediately send alerts or notifications to relevant parties, allowing for quick identification and response to potential water-related issues.

- This technology aims to prevent water wastage, reduce damage, and improve overall water management efficiency

## Benefits of greywater recycling with IoT:




- Water Conservation: Greywater recycling reduces freshwater consumption for non-potable applications, conserving a valuable resource.
- Cost Savings: By reusing greywater, users can reduce their water bills and operational cost.
- Sustainability: It promotes sustainable water management practices and reduces the environmental impact of wastewater disposal.
- Automation: IoT enables automated control and remote monitoring, ensuring efficient greywater treatment and distribution.

## Benefits of Smart water meter :

- Consumer Engagement: Smart water meters can empower consumers with information about their water usage, encouraging more responsible consumption and environmental awareness.
- Environmental Benefits: By reducing water waste, smart meters contribute to environmental conservation by conserving precious resources.
- Accurate Billing: With precise consumption data, billing becomes more accurate, reducing disputes and ensuring that consumers pay only for the water they use.
- Real-time Monitoring: IoT-enabled smart water meters provide real-time data on water consumption, allowing consumers and utility companies to monitor usage patterns and detect leaks promptly.

**Benefits of Smart water lekage:**


- Early Detection: IoT sensors can detect leaks in real-time or even before they become significant issues, allowing for prompt action to prevent water damage or loss.
- Reduced Water Waste: By identifying leaks promptly, water conservation is promoted, saving both water resources and money on utility bills.
- Data Insights: These systems collect and analyze data over time, providing insights into water usage patterns, helping users make informed decisions about conservation and optimization

**Hardware components :**

**Water Sensors**

- These sensors can detect the presence of water and moisture.
- They come in various forms, such as water leak detectors, moisture sensors, or flood sensors.


**Microcontrollers:**

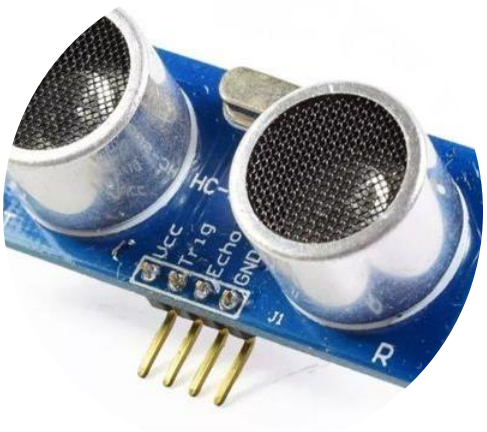
- IoT devices often use microcontrollers (e.g., Arduino, Raspberry Pi) to process sensor data, make decisions, and control other components.

**Power Source:**

- Depending on the application, devices can be powered by batteries, solar panels, or wired power sources. Battery-powered devices may need low-power components to conserve energy.

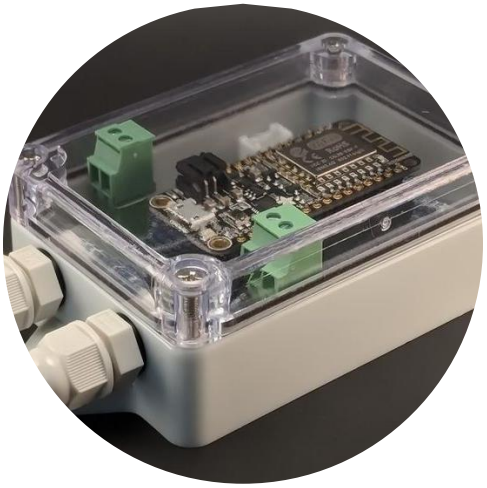
**Alarms and Notifications:**

- Devices may include speakers, LEDs, or other indicators to provide immediate alerts on-site. Notifications can also be sent via email, SMS, or push notifications to inform users of detected leaks.




**Additional Sensors:**

- Depending on the specific use case, additional sensors like temperature sensors, humidity sensors, or even cameras may be integrated to provide more context or detect related issues.



**Enclosures:**

- To protect the hardware from environmental factors, waterproof and weather-resistant enclosures may be used, especially for outdoor installations.

**Cloud Platform:**

- Users access and monitor the data through a web or mobile application. These interfaces allow users to receive alerts and visualize data.


**Central Hub or Gateway:**

In many cases, data from multiple sensors is sent to a central hub or gateway device that aggregates and forwards the data to a cloud platform or user interface.

# SOFTWARE WEBSITE :

- Wokwi

## WOKWI:

- The WOWKI simulator is a versatile platform designed to simulate IoT (Internet of Things) environments for testing and development purposes.
- It provides a controlled and realistic virtual environment to mimic real-world scenarios and interactions that involve IoT devices and sensors.
- Here are some key points related to the WOWKI simulator and its use in

## IOT APPLICATIONS:

1. IOT SIMULATION:

- WOWKI simulator is dedicated to simulating IoT devices, networks, and interactions.
- It allows developers and researchers to create, test, and analyze IoT applications without the need for physical devices or real-world environments.

- You can use it to show real-time measurements, alerts, or any relevant information from the monitoring system, making it easier for users or operators to access and understand the data.
- It provides a compact and clear display, which is particularly useful for quick on-site analysis and decision-making in water monitoring applications.

## 2.DEVICE EMULATION:

- WOWKI provides the capability to simulate real-world scenarios, such as smart cities, industrial automation, agriculture, and environmental monitoring.
- This enables the testing of IoT applications in contextually relevant environments..

## 3.REALISTIC SCENARIOS:

- WOWKI provides the capability to simulate real-world scenarios, such as smart cities, industrial automation, agriculture, and environmental monitoring.
- This enables the testing of IoT applications in contextually relevant environments.

## 4.DATA GENERATION:

- It can generate realistic data streams, which is crucial for testing data analytics, machine learning algorithms, and the responsiveness of IoT systems under different conditions.

## 5.SECURITY TESTING:

- Security is a critical aspect of IoT, and the WOWKI simulator allows for the testing of IoT security measures in a controlled environment. This helps identify vulnerabilities and develop strategies to protect IoT networks.
- Overall, the WOWKI simulator plays a significant role in accelerating the development and testing of IoT applications, ensuring they are robust, efficient, and ready for real-world deployment. It offers a valuable playground for IoT innovation and research.

PROGRAMMING LANGUAGE:

PYTHON:

- Python is a versatile and high-level programming language known for its simplicity and readability.
- It is widely used for web development, data analysis, artificial intelligence, and more.
- Python's clean syntax and extensive libraries make it a popular choice for both beginners and experienced developers.

PROGRAM :

Water level Detection Program

```
/* Fill-in your Template ID (only if using Blynk.Cloud) */
```

```
#define BLYNK_TEMPLATE_ID "TMPLlcLQu4bQ"

#define BLYNK_TEMPLATE_NAME "water monitor"

#define BLYNK_AUTH_TOKEN "OgvenxCWu9sG7-9deFGLFCLE4rWCGW7N"

// Your WiFi credentials.

// Set password to "" for open networks.

char ssid[] = "Wokwi-GUEST";   //WiFi Name

char pass[] = "";   //WiFi Password

//Set Water Level Distance in CM

int emptyTankDistance = 150 ;  //Distance when tank is empty

int fullTankDistance =  40 ;  //Distance when tank is full (must be greater than 25cm)

//Set trigger value in percentage

int triggerPer =   10 ;  //alarm/pump will start when water level drop below triggerPer

#include <Adafruit_SSD1306.h>

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <AceButton.h>

using namespace ace_button;

// Define connections to sensor

#define TRIGPIN    27  //D6

#define ECHOPIN    26  //D7

#define wifiLed    2  //D0
```

```
#define BuzzerPin  13  //D3

#define RelayPin   14 //D5

#define ButtonPin1 12   //RX   //Mode

#define ButtonPin2 33  //SD3  //Relay

#define ButtonPin3 32  //D4   //STOP Buzzer

#define fullpin    25

//Change the virtual pins according the rooms

#define VPIN_BUTTON_1    V1

#define VPIN_BUTTON_2    V2

#define VPIN_BUTTON_3    V3

#define VPIN_BUTTON_4    V4

#define VPIN_BUTTON_5    V5

#define SCREEN_WIDTH 128 // OLED display width, in pixels

#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)

#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);

float duration;

float distance;

int   waterLevelPer;

bool  toggleBuzzer = HIGH; //Define to remember the toggle state

bool toggleRelay = false; //Define the toggle state for relay
```

```
bool modeFlag = true;
bool conection = true;
String currMode;
char auth[] = BLYNK_AUTH_TOKEN;
ButtonConfig config1;
AceButton button1(&config1);
ButtonConfig config2;
AceButton button2(&config2);
ButtonConfig config3;
AceButton button3(&config3);
void handleEvent1(AceButton*, uint8_t, uint8_t);
void handleEvent2(AceButton*, uint8_t, uint8_t);
void handleEvent3(AceButton*, uint8_t, uint8_t);
BlynkTimer timer;
void checkBlynkStatus() { // called every 3 seconds by SimpleTimer

  bool isconnected = Blynk.connected();
  if (isconnected == false) {
    //Serial.println("Blynk Not Connected");
    digitalWrite(wifiLed, LOW);
    conection = true;

  }
```

```
  if (isconnected == true) {
    digitalWrite(wifiLed, HIGH);
    //Serial.println("Blynk Connected");
    conection = false;
  }
}
// When App button is pushed - switch the state
BLYNK_WRITE(VPIN_BUTTON_3) {
  modeFlag = param.asInt();
  if(!modeFlag && toggleRelay){
      digitalWrite(RelayPin, LOW);  //turn off the pump
      toggleRelay = false;
    }
    controlBuzzer(500);
    currMode = modeFlag ? "AUTO" : "MANUAL";
}
BLYNK_WRITE(VPIN_BUTTON_4) {
  if(!modeFlag){
    toggleRelay = param.asInt();
    digitalWrite(RelayPin, toggleRelay);
    controlBuzzer(500);
  }
  else{
```

```
    Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  }
}
BLYNK_WRITE(VPIN_BUTTON_5) {
  toggleBuzzer = param.asInt();
  digitalWrite(BuzzerPin, toggleBuzzer);
}
BLYNK_CONNECTED() {
  Blynk.syncVirtual(VPIN_BUTTON_1);
  Blynk.syncVirtual(VPIN_BUTTON_2);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
}
void displayData(){
  display.clearDisplay();
  display.setTextSize(3);
  display.setCursor(30,0);
  display.print(waterLevelPer);
  display.print("");
  display.print("%");
  display.setTextSize(1);
  display.setCursor(0,25);
```

```
  display.print(conection ? "OFFLINE" : "ONLINE");
  display.setCursor(60,25);
  display.print(currMode);
  display.setCursor(110,25);
  display.print(toggleRelay ? "! ON" : "OFF");
  display.display();
}
void measureDistance(){
  // Set the trigger pin LOW for 2uS
  digitalWrite(TRIGPIN, LOW);
  delayMicroseconds(2);
  // Set the trigger pin HIGH for 20us to send pulse
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(20);
  // Return the trigger pin to LOW
  digitalWrite(TRIGPIN, LOW);
  // Measure the width of the incoming pulse
  duration = pulseIn(ECHOPIN, HIGH);
  // Determine distance from duration
  // Use 343 metres per second as speed of sound
  // Divide by 1000 as we want millimeters
  distance = ((duration / 2) * 0.343)/10;
  if (distance > (fullTankDistance - 10)  && distance < emptyTankDistance ){
```

```
    waterLevelPer = map((int)distance ,emptyTankDistance, fullTankDistance, 0,
100);
    Blynk.virtualWrite(VPIN_BUTTON_1, waterLevelPer);
    Blynk.virtualWrite(VPIN_BUTTON_2, (String(distance) + " cm"));
    // Print result to serial monitor
//    Serial.print("Distance: ");
//    Serial.print(distance);
//    Serial.println(" cm");
    if (waterLevelPer < triggerPer){
      if(modeFlag){
        if(!toggleRelay){
          controlBuzzer(500);
          digitalWrite(RelayPin, HIGH); //turn on relay
          toggleRelay = true;
          Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
        }
      }
      else{
        if (toggleBuzzer == HIGH){
          digitalWrite(BuzzerPin, HIGH);
          Serial.println(" BuzzerPin high");
        }
      }
```

```
}
if (distance < fullTankDistance){
  digitalWrite(fullpin, HIGH);
  if(modeFlag){
    if(toggleRelay){
      digitalWrite(RelayPin, LOW); //turn off relay
      toggleRelay = false;
      Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
      controlBuzzer(500);
    }
  }
  else{
    if (toggleBuzzer == HIGH){
    digitalWrite(BuzzerPin, HIGH);
    }
  }
}
if (distance > (fullTankDistance + 5) && waterLevelPer > (triggerPer + 5)){
  toggleBuzzer = HIGH;
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
  digitalWrite(BuzzerPin, LOW);
}
if (distance = fullTankDistance){
```

```
    Serial.println(" udh bang ");
    }
  }
  displayData();
  delay(100);
}
void controlBuzzer(int duration){
  digitalWrite(BuzzerPin, HIGH);
  Serial.println(" BuzzerPin HIT");
  delay(duration);
  digitalWrite(BuzzerPin, LOW);
}
void setup() {
  // Set up serial monitor
  Serial.begin(9600);
  // Set pinmodes for sensor connections
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
  pinMode(wifiLed, OUTPUT);
  pinMode(RelayPin, OUTPUT);
  pinMode(BuzzerPin, OUTPUT);
  pinMode(fullpin, OUTPUT);
  pinMode(ButtonPin1, INPUT_PULLUP);
```

```
  pinMode(ButtonPin2, INPUT_PULLUP);
  pinMode(ButtonPin3, INPUT_PULLUP);
  digitalWrite(wifiLed, HIGH);
  digitalWrite(RelayPin, LOW);
  digitalWrite(BuzzerPin, LOW);
  config1.setEventHandler(button1Handler);
  config2.setEventHandler(button2Handler);
  config3.setEventHandler(button3Handler);
  button1.init(ButtonPin1);
  button2.init(ButtonPin2);
  button3.init(ButtonPin3);
  currMode = modeFlag ? "AUTO" : "MANUAL";
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  delay(1000);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.clearDisplay();
  WiFi.begin(ssid, pass);
  timer.setInterval(2000L, checkBlynkStatus); // check if Blynk server is connected
every 2 seconds
```

```
  timer.setInterval(1000L,  measureDistance); // measure water level every 1
seconds
  Blynk.config(auth);
  delay(1000);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
  delay(500);
}
void loop() {
  Blynk.run();
  timer.run(); // Initiates SimpleTimer
  button1.check(); //mode change
  button3.check(); //buzzer reset
  if(!modeFlag){  //if in manual mode
    button2.check();
  }
}
void button1Handler(AceButton* button, uint8_t eventType, uint8_t buttonState)
{
  Serial.println("EVENT1");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
```

```
      if(modeFlag && toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
        controlBuzzer(500);
      }
      modeFlag = !modeFlag;
      currMode = modeFlag ? "AUTO" : "MANUAL";
      Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
      controlBuzzer(200);
      break;
  }
}
void button2Handler(AceButton* button, uint8_t eventType, uint8_t buttonState)
{
  Serial.println("EVENT2");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
      }
      else{
```

```
      digitalWrite(RelayPin, HIGH);  //turn on the pump
      toggleRelay = true;
    }
    Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
    controlBuzzer(500);
    delay(1000);
    break;
  }
}
void button3Handler(AceButton* button, uint8_t eventType, uint8_t buttonState)
{
  Serial.println("EVENT3");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      digitalWrite(BuzzerPin, LOW);
      toggleBuzzer = LOW;
      Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
      break;
  }
}
```

## COMPONENTS :

- ESP32
- ADAFRUITSSD1306
- ULTRASONIC DISTANCE SENSOR
- ACE BUTTONS
- LED

ESP32:

- The ESP32 kit is a hardware platform that features the ESP32 microcontroller, a powerful and versatile microcontroller unit developed by Espressif Systems. It is often used in IoT and embedded systems projects due to its capabilities, which include dual-core processing, built-in Wi-Fi and Bluetooth connectivity, and a wide range of input/output options.

## ADAFRUIT SSD1306:

- The Adafruit SSD1306 is a popular OLED (Organic Light-Emitting Diode) display module that can be used in a water monitoring project. In this context, it serves as a visual output interface, displaying important information and data related to water quality, level, or other parameters.
- You can use it to show real-time measurements, alerts, or any relevant information from the monitoring system, making it easier for users or operators to access and understand the data.
- It provides a compact and clear display, which is particularly useful for quick on-site analysis and decision-making in water monitoring applications.

## PURPOSE :

- The ESP32 kit serves as the core component in water detection by providing wireless connectivity and data processing, allowing for real-time monitoring and remote alerts, ensuring the efficient management and protection of water resources.

## HC-SR04 ULTRASONIC DISTANCE SENSOR:

- The HC-SR04 ultrasonic distance sensor is a key component in water monitoring systems. It utilizes ultrasonic sound waves to measure the distance between the sensor and the water surface.

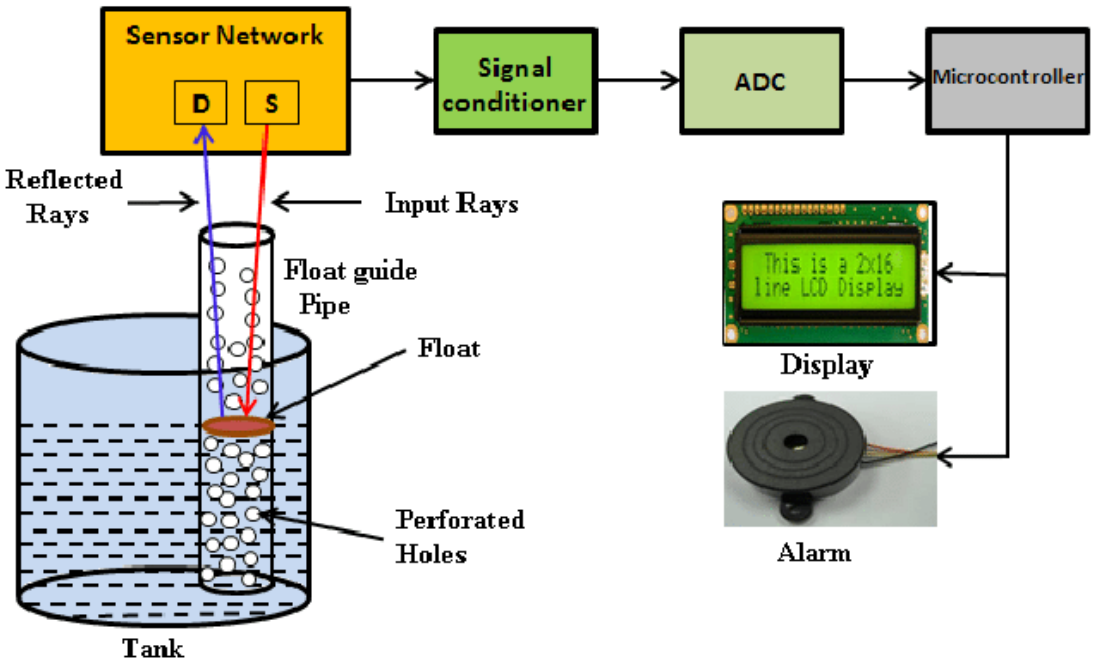
- When deployed above a water body, it calculates the water level by timing the sound wave's round trip.
- This data is crucial for monitoring water levels in reservoirs, rivers, or tanks, aiding in flood prediction, water resource management, and ensuring optimal water usage. Its non-contact nature and accuracy make it a valuable tool for real-time water level measurements, enabling timely responses to water level fluctuations and ensuring efficient water resource utilization.

## PURPOSE:

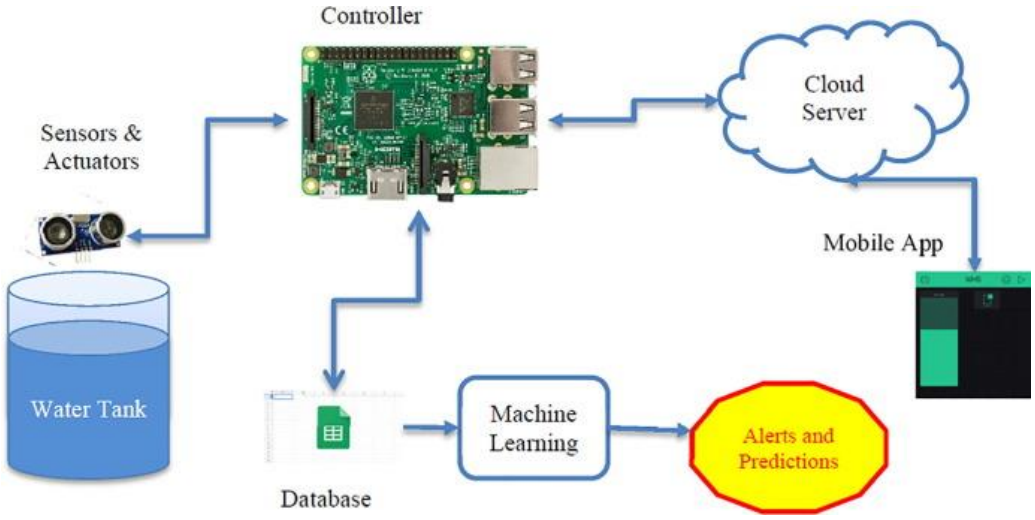
- This sensor emits ultrasonic pulses and calculates the time it takes for the signals to bounce back from the water's surface.

- By converting this time into distance, it provides accurate water level data, essential for monitoring and managing water resources

## BLOCK DIAGRAM:



# CONNECTING MOBILE APP WITH SMART WATER MANAGEMENT IOT PROJECT:




## 1.Set Up a Server:

- ✓ Ensure your Smart Water Management Iot project has a server that can receive and process data from IoT devices.
- ✓ Implement APIs on the server to provide data to the mobileapp.
- ✓ These APIs should handle incoming HTTP requests and return data in a format that the mobile app can understand

(usually JSON).

## 2. Mobile App Development:

- ✓ Use a mobile app development framework like Flutter or React Native to create your mobile app. In this guide, I'll use Flutter.

- ✓ Create the app's user interface and layout, including buttons and widgets to display data.

3. HTTP Requests from Mobile App:

- ✓ Use the http package (or equivalent) in Flutter to send HTTP requests to the server.
- ✓ For example, you can use the http.get method to request data.

PROGRAM

```
/* Fill-in your Template ID (only if using Blynk.Cloud) */
#define BLYNK_TEMPLATE_ID "TMPLIcLQu4bQ"
#define BLYNK_TEMPLATE_NAME "water monitor"
#define BLYNK_AUTH_TOKEN "OgvenxCWu9sG7-9deFGLFCLE4rWCGW7N"
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "Wokwi-GUEST";   //WiFi Name
char pass[] = "";   //WiFi Password
//Set Water Level Distance in CM
int emptyTankDistance = 150 ;  //Distance when tank is empty
int fullTankDistance =  40 ;  //Distance when tank is full (must be greater than 25cm)
//Set trigger value in percentage
int triggerPer =   10 ;  //alarm/pump will start when water level drop below triggerPer
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
```

```
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <AceButton.h>
using namespace ace_button;
// Define connections to sensor
#define TRIGPIN    27  //D6
#define ECHOPIN    26  //D7
#define wifiLed    2  //D0
#define BuzzerPin  13  //D3
#define RelayPin   14 //D5
#define ButtonPin1 12   //RX   //Mode
#define ButtonPin2 33  //SD3  //Relay
#define ButtonPin3 32  //D4   //STOP Buzzer
#define fullpin    25
//Change the virtual pins according the rooms
#define VPIN_BUTTON_1    V1
#define VPIN_BUTTON_2    V2
#define VPIN_BUTTON_3    V3
#define VPIN_BUTTON_4    V4
#define VPIN_BUTTON_5    V5
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

```
float duration;
float distance;
int   waterLevelPer;
bool  toggleBuzzer = HIGH; //Define to remember the toggle state
bool toggleRelay = false; //Define the toggle state for relay
bool modeFlag = true;
bool conection = true;
String currMode;
char auth[] = BLYNK_AUTH_TOKEN;
ButtonConfig config1;
AceButton button1(&config1);
ButtonConfig config2;
AceButton button2(&config2);
ButtonConfig config3;
AceButton button3(&config3);
void handleEvent1(AceButton*, uint8_t, uint8_t);
void handleEvent2(AceButton*, uint8_t, uint8_t);
void handleEvent3(AceButton*, uint8_t, uint8_t);
BlynkTimer timer;
void checkBlynkStatus() { // called every 3 seconds by SimpleTimer

  bool isconnected = Blynk.connected();
  if (isconnected == false) {
    //Serial.println("Blynk Not Connected");
    digitalWrite(wifiLed, LOW);
```

```
    conection = true;


  }
  if (isconnected == true) {
    digitalWrite(wifiLed, HIGH);
    //Serial.println("Blynk Connected");
    conection = false;
  }
}
// When App button is pushed - switch the state
BLYNK_WRITE(VPIN_BUTTON_3) {
  modeFlag = param.asInt();
  if(!modeFlag && toggleRelay){
      digitalWrite(RelayPin, LOW);  //turn off the pump
      toggleRelay = false;
    }
    controlBuzzer(500);
    currMode = modeFlag ? "AUTO" : "MANUAL";
}
BLYNK_WRITE(VPIN_BUTTON_4) {
  if(!modeFlag){
    toggleRelay = param.asInt();
    digitalWrite(RelayPin, toggleRelay);
    controlBuzzer(500);
  }
```

```
  else{
    Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  }
}
BLYNK_WRITE(VPIN_BUTTON_5) {
  toggleBuzzer = param.asInt();
  digitalWrite(BuzzerPin, toggleBuzzer);
}
BLYNK_CONNECTED() {
  Blynk.syncVirtual(VPIN_BUTTON_1);
  Blynk.syncVirtual(VPIN_BUTTON_2);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
}
void displayData(){
  display.clearDisplay();
  display.setTextSize(3);
  display.setCursor(30,0);
  display.print(waterLevelPer);
  display.print("");
  display.print("%");
  display.setTextSize(1);
  display.setCursor(0,25);
  display.print(conection ? "OFFLINE" : "ONLINE");
```

```
  display.setCursor(60,25);
  display.print(currMode);
  display.setCursor(110,25);
  display.print(toggleRelay ? "! ON" : "OFF");
  display.display();
}
void measureDistance(){
  // Set the trigger pin LOW for 2uS
  digitalWrite(TRIGPIN, LOW);
  delayMicroseconds(2);
  // Set the trigger pin HIGH for 20us to send pulse
  digitalWrite(TRIGPIN, HIGH);
  delayMicroseconds(20);
  // Return the trigger pin to LOW
  digitalWrite(TRIGPIN, LOW);
  // Measure the width of the incoming pulse
  duration = pulseIn(ECHOPIN, HIGH);
  // Determine distance from duration
  // Use 343 metres per second as speed of sound
  // Divide by 1000 as we want millimeters
  distance = ((duration / 2) * 0.343)/10;
  if (distance > (fullTankDistance - 10)  && distance < emptyTankDistance ){
    waterLevelPer = map((int)distance ,emptyTankDistance, fullTankDistance, 0, 100);
    Blynk.virtualWrite(VPIN_BUTTON_1, waterLevelPer);
    Blynk.virtualWrite(VPIN_BUTTON_2, (String(distance) + " cm"));
```

```
    // Print result to serial monitor
//    Serial.print("Distance: ");
//    Serial.print(distance);
//    Serial.println(" cm");
    if (waterLevelPer < triggerPer){
      if(modeFlag){
        if(!toggleRelay){
          controlBuzzer(500);
          digitalWrite(RelayPin, HIGH); //turn on relay
          toggleRelay = true;
          Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
        }
      }
      else{
        if (toggleBuzzer == HIGH){
          digitalWrite(BuzzerPin, HIGH);
          Serial.println(" BuzzerPin high");
        }
      }
    }
    if (distance < fullTankDistance){
      digitalWrite(fullpin, HIGH);
      if(modeFlag){
        if(toggleRelay){
          digitalWrite(RelayPin, LOW); //turn off relay
```

```
        toggleRelay = false;
        Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
        controlBuzzer(500);
      }
    }
    else{
      if (toggleBuzzer == HIGH){
      digitalWrite(BuzzerPin, HIGH);
      }
    }
  }
  if (distance > (fullTankDistance + 5) && waterLevelPer > (triggerPer + 5)){
    toggleBuzzer = HIGH;
    Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
    digitalWrite(BuzzerPin, LOW);
  }
  if (distance = fullTankDistance){
  Serial.println(" udh bang ");
  }
 }
 displayData();
 delay(100);
}
void controlBuzzer(int duration){
 digitalWrite(BuzzerPin, HIGH);
```

```
  Serial.println(" BuzzerPin HIT");
  delay(duration);
  digitalWrite(BuzzerPin, LOW);
}
void setup() {
  // Set up serial monitor
  Serial.begin(9600);
  // Set pinmodes for sensor connections
  pinMode(ECHOPIN, INPUT);
  pinMode(TRIGPIN, OUTPUT);
  pinMode(wifiLed, OUTPUT);
  pinMode(RelayPin, OUTPUT);
  pinMode(BuzzerPin, OUTPUT);
  pinMode(fullpin, OUTPUT);
  pinMode(ButtonPin1, INPUT_PULLUP);
  pinMode(ButtonPin2, INPUT_PULLUP);
  pinMode(ButtonPin3, INPUT_PULLUP);
  digitalWrite(wifiLed, HIGH);
  digitalWrite(RelayPin, LOW);
  digitalWrite(BuzzerPin, LOW);
  config1.setEventHandler(button1Handler);
  config2.setEventHandler(button2Handler);
  config3.setEventHandler(button3Handler);
  button1.init(ButtonPin1);
  button2.init(ButtonPin2);
```

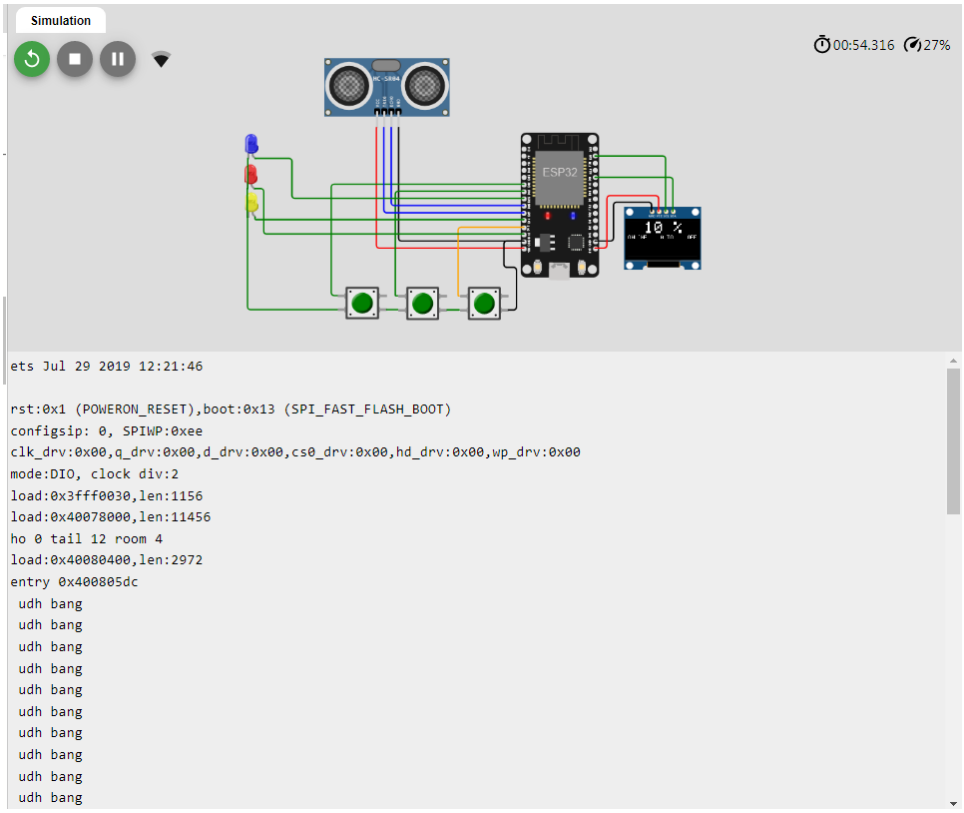
```
  button3.init(ButtonPin3);
  currMode = modeFlag ? "AUTO" : "MANUAL";
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  delay(1000);
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.clearDisplay();
  WiFi.begin(ssid, pass);
  timer.setInterval(2000L, checkBlynkStatus); // check if Blynk server is connected every 2 seconds
  timer.setInterval(1000L,  measureDistance); // measure water level every 1 seconds
  Blynk.config(auth);
  delay(1000);
  Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
  Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
  Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
  delay(500);
}
void loop() {
  Blynk.run();
  timer.run(); // Initiates SimpleTimer
  button1.check(); //mode change
```

```
  button3.check(); //buzzer reset
  if(!modeFlag){  //if in manual mode
    button2.check();
  }
}
void button1Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT1");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(modeFlag && toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
        controlBuzzer(500);
      }
      modeFlag = !modeFlag;
      currMode = modeFlag ? "AUTO" : "MANUAL";
      Blynk.virtualWrite(VPIN_BUTTON_3, modeFlag);
      controlBuzzer(200);
      break;
  }
}
void button2Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT2");
  switch (eventType) {
```

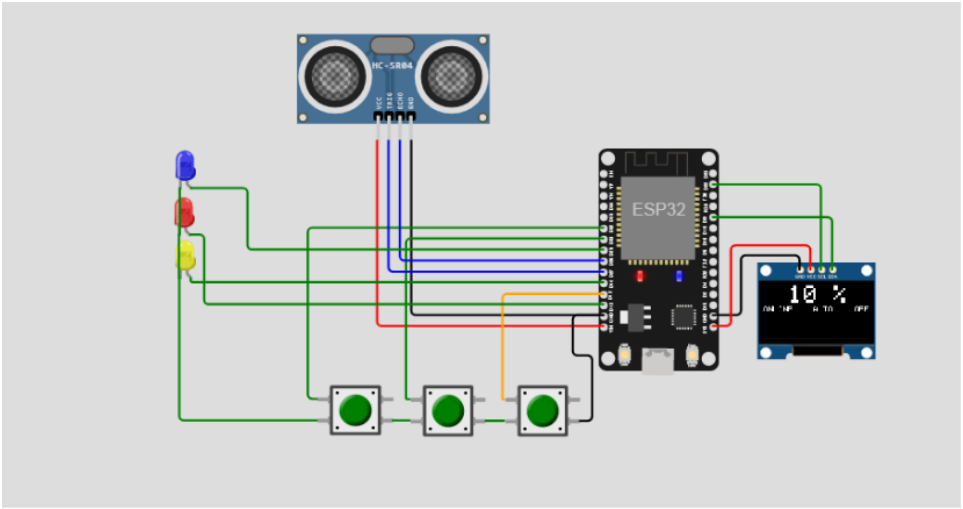
```
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      if(toggleRelay){
        digitalWrite(RelayPin, LOW);  //turn off the pump
        toggleRelay = false;
      }
      else{
        digitalWrite(RelayPin, HIGH);  //turn on the pump
        toggleRelay = true;
      }
      Blynk.virtualWrite(VPIN_BUTTON_4, toggleRelay);
      controlBuzzer(500);
      delay(1000);
      break;
  }
}
void button3Handler(AceButton* button, uint8_t eventType, uint8_t buttonState) {
  Serial.println("EVENT3");
  switch (eventType) {
    case AceButton::kEventReleased:
      //Serial.println("kEventReleased");
      digitalWrite(BuzzerPin, LOW);
      toggleBuzzer = LOW;
      Blynk.virtualWrite(VPIN_BUTTON_5, toggleBuzzer);
      break;
```

```
  }
}
```

## OUTPUT:




## CIRCUIT DIAGRAM:



## APP INTERFACE:

To connect your mobile device to this system, you can follow these steps:

- Download the Blynk app: If you haven't already, download the Blynk app from your device's app store. Blynk is available for both Android and iOS.
- Create a Blynk account: Open the Blynk app and create a Blynk account if you don't have one.
- Create a new Blynk project.

a. Click on the "+" button to create a new project.

b. Select your hardware: Choose "ESP32" as your hardware model.

c. Select the connection type: Choose the appropriate connection type, either Wi-Fi or Cellular.

- Set up your project:

a. Customize your project by adding widgets like buttons, displays, and sliders.

b. Assign virtual pins to these widgets, and make sure they match the virtual pins in your Arduino code (e.g., VPIN_BUTTON_1, VPIN_BUTTON_2, etc.).

- Configure the Auth Token:

a. In your Arduino code, you've defined the BLYNK_AUTH_TOKEN. Replace it with the auth token generated for your project in the Blynk app.

- Connect your ESP32 to Wi-Fi:

a. Make sure your ESP32 is connected to the same Wi-Fi network as your mobile device. You've set your Wi-Fi credentials in the code.

- Upload the code to your ESP32:

a. Upload the modified Arduino code to your ESP32 board using the Arduino IDE.

- Run the project:

a. After uploading the code, open the Serial Monitor to monitor the ESP32's status.

b. Once the ESP32 is running, it should connect to the Blynk server and show the water level data on your mobile app.

c. You can interact with the widgets in the Blynk app to control the system.

## CONCLUSION

- In conclusion, water level detection based on IoT offers significant advantages in terms of real-time monitoring and remote control of water levels in various applications. It enhances efficiency, reduces the risk of water-related disasters, and provides valuable data for informed decision-making. By leveraging IoT technology, we can better manage water resources and improve overall water infrastructure, contributing to environmental sustainability and human safety. As IoT continues to evolve, the potential for even more advanced and sophisticated water level detection systems is promising.