**PHASE – 5 - IOT PROJECT**

**SMART WATER MANAGEMENT SYSTEM**

**CHAPTER 1 :**

**INTRODUCTION:**

Water conservation is a pressing global concern, and the implementation of smart water management systems has emerged as a crucial step toward addressing this issue. These systems leverage technology to monitor and manage water consumption in real time, offering valuable insights for individuals, communities, and governments. The Python code presented below illustrates a simulated smart water management system. It simulates the operation of smart water meters and water quality sensors, offering a glimpse into how real-time water consumption monitoring and data analysis can play a pivotal role in water conservation efforts.

In this introductory code, we set the stage for understanding the various components and functionalities of a smart water management system. We simulate the behavior of smart water meters and water quality sensors, highlighting the critical role of data processing and analysis in detecting water leaks and ensuring water quality. By adopting such systems and technologies, we can take significant steps toward responsible water usage and conservation.

The necessity of a smart water management system within the Internet of Things (IoT) domain is underpinned by several compelling factors. First and foremost, these systems provide invaluable data-driven insights into water consumption, quality, and distribution, enabling well-informed decision-making and the optimization of water management practices. They play a pivotal role in identifying leaks, wasteful consumption patterns, and conservation opportunities, significantly reducing water wastage and fostering responsible water use.

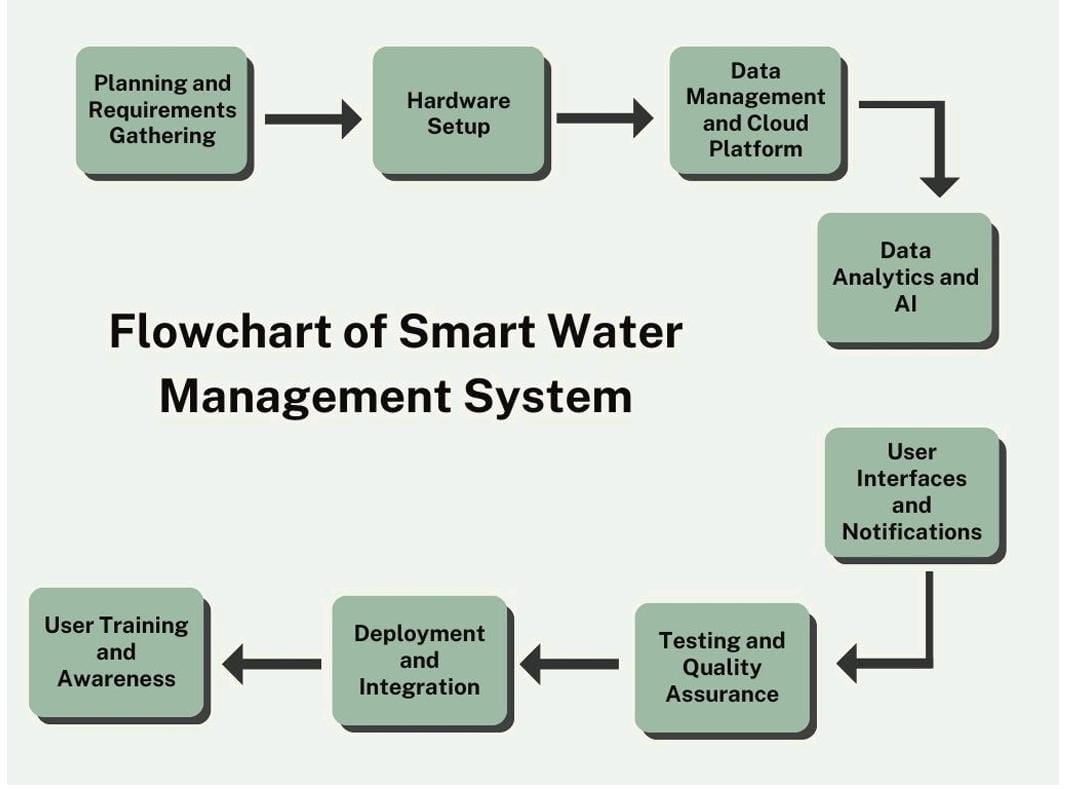
Additionally, IoT-based systems monitor aging water distribution infrastructure, proactively detecting leaks and inefficiencies, which is instrumental in extending the longevity and efficiency of infrastructure. Moreover, these systems contribute to environmental sustainability by enabling the monitoring and mitigation of the environmental impact of water management activities. They facilitate the adaptive measures needed to combat climate change and its impact on water resources.

Furthermore, IoT systems offer a means to raise public awareness about water consumption through real-time data provision, motivating individuals and communities to adopt water-saving behaviors. They also aid utility companies in resource allocation, minimizing operational costs, and increasing overall efficiency. By providing early drought warnings, these systems help communities prepare and implement water-saving strategies during water shortages.

In a broader context, IoT-driven water management supports remote monitoring and control, reducing the necessity for physical inspections and interventions. This integration with other IoT systems, such as weather monitoring and agriculture, enables a holistic approach to resource management. In conclusion, IoT-powered smart water management systems are indispensable for promoting responsible water use, ensuring water quality, optimizing resource allocation, and adapting to environmental challenges, making them a critical step toward sustainable and efficient water management practices.

**CHAPTER 2 :**

**BLOCK DIAGRAM EXPLANATION :**



**Project Planning and Requirement Gathering**: Specify the project's scope, considering the intended user base (residential, commercial, or industrial). Gather specifications for user interfaces (web and mobile applications), software (cloud platform, data analytics), and hardware (smart meters, sensors).

**Hardware Installation**: For smart water meters, purchase meters suitable for different sectors (industrial, commercial, residential), install them at the respective locations, set them up for precise water usage measurement, ensure secure connectivity, check their operation, and establish maintenance procedures. For sensors, choose the right ones for measuring flow, water quality, pressure, and temperature, install them at strategic points, calibrate as needed, set up data transmission to the data gateway, implement error-checking, and establish regular sensor maintenance.

**Setting up a Cloud Platform and Managing Data**: Choose a cloud computing platform (e.g., AWS, Azure, Google Cloud) for data processing and storage. Create a central database, establish secure data transfer methods, create interfaces and APIs for data ingestion and retrieval, ensure data security and privacy compliance, and process incoming data in real-time.

**Implementing Data Analytics and AI:** Develop data analytics models to process and evaluate water usage data, create machine learning models to detect consumption patterns and anomalies, implement leak detection methods, and offer water-saving insights based on usage data. Regularly upgrade and refine AI algorithms for improved detection accuracy.

**User Interfaces and Notifications**: Develop consumer, utility, and local government-friendly online and mobile applications, create real-time dashboards, set up notification systems for alerts and notifications via push, SMS, and email, provide user profiles and alert customization, and offer access to previous usage information and analysis.

**Testing and Quality Control**: Perform comprehensive testing on all system components, including user interfaces, data processing, and software. Test the system's performance under varying loads, assess alerting and leak detection precision, and verify user privacy safeguards and data security measures.

**Integration and Deployment:** Install the system in designated locations, connect it to existing utilities and water infrastructure, and ensure seamless data synchronization between hardware and the cloud platform.

**User Education and Awareness Programs**: Conduct training for users, utility suppliers, and municipal authorities on system usage. Launch public awareness campaigns to inform the public about water conservation benefits and smart water management systems.

**Monitoring and Upkeep**: Create a performance tracking method and maintain routine sensor, smart meter, and software upgrade plans. Continuously analyze system data for areas that can be enhanced and optimized.

**Scaling and Evaluation**: Analyze the impact of the smart water management system on water conservation, cost savings, and environmental benefits. Consider expanding the system to more communities or regions based on the initial deployment's success.

**Reporting and Documentation**: Create thorough documentation for system administrators, users, and maintenance staff. Generate recurring reports on water usage trends, leak findings, and conservation achievements.

**Improvement and Feedback Loop**: Establish a feedback loop with users and stakeholders to collect input for system improvements. Continuously enhance the system based on user feedback and technological advancements.

**CHAPTER 3 :**

**PROGRAM FOR SMART WATER MANAGEMENT SYSTEM :**

import random

import time

class SmartWaterMeter:

def \_init\_(self, location, pin):

self.location = location

self.water\_usage = 0

self.pin = pin # Pin used for simulating activity (optional)

def measure\_water\_usage(self):

# Simulate water usage data

simulated\_data = random.randint(1, 50)

self.water\_usage += simulated\_data

# Simulate water meter activity using a pin (optional)

# digitalWrite(self.pin, HIGH) # Simulate the meter is active

# Simulate the time taken to measure water usage (optional)

# delay(100)

# digitalWrite(self.pin, LOW) # Simulate the meter is inactive (optional)

return self.water\_usage

class WaterQualitySensor:

def \_init\_(self, location, pin):

self.location = location

self.water\_quality = "Good"

self.pin = pin # Pin used for simulating activity (optional)

def measure\_water\_quality(self):

# Simulate water quality data

simulated\_data = "Good"

# Simulate sensor activity using a pin (optional)

# digitalWrite(self.pin, HIGH) # Simulate the sensor is active

# Simulate the time taken to measure water quality (optional)

# delay(100)

# digitalWrite(self.pin, LOW) # Simulate the sensor is inactive (optional)

return simulated\_data

class CloudPlatform:

def \_init\_(self):

self.water\_meters = {}

self.water\_quality\_sensors = {}

def store\_water\_meter\_data(self, meter\_location, data):

self.water\_meters[meter\_location] = data

def store\_water\_quality\_data(self, sensor\_location, data):

self.water\_quality\_sensors[sensor\_location] = data

def data\_processing(data):

# Analyze and process incoming data here

for location, usage in data.water\_meters.items():

if usage > 40:

print(f"Potential water leak detected at {location}. Alert sent.")

for location, quality in data.water\_quality\_sensors.items():

if quality != "Good":

print(f"Water quality issue detected at {location}. Alert sent.")

# Specify the pins for each component

meter1 = SmartWaterMeter("Residential", 2) # Pin 2 used for simulating activity (optional)

meter2 = SmartWaterMeter("Commercial", 3) # Pin 3 used for simulating activity (optional)

quality\_sensor1 = WaterQualitySensor("Reservoir", 4) # Pin 4 used for simulating activity (optional)

quality\_sensor2 = WaterQualitySensor("Treatment Plant", 5) # Pin 5 used for simulating activity (optional)

# Create a cloud platform instance

cloud\_platform = CloudPlatform()

while True:

# Simulate water consumption and quality data

meter1\_data = meter1.measure\_water\_usage()

meter2\_data = meter2.measure\_water\_usage()

quality\_sensor1\_data = quality\_sensor1.measure\_water\_quality()

quality\_sensor2\_data = quality\_sensor2.measure\_water\_quality()

# Store data in the cloud platform

cloud\_platform.store\_water\_meter\_data(meter1.location, meter1\_data)

cloud\_platform.store\_water\_meter\_data(meter2.location, meter2\_data)

cloud\_platform.store\_water\_quality\_data(quality\_sensor1.location, quality\_sensor1\_data)

cloud\_platform.store\_water\_quality\_data(quality\_sensor2.location, quality\_sensor2\_data)

# Process the stored data (simplified)

data\_processing(cloud\_platform)

# Simulate data collection at regular intervals (e.g., every 5 minutes)

time.sleep(300) # Sleep for 5 minutes

**CODE FOR IMPLEMENTING:**

#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);

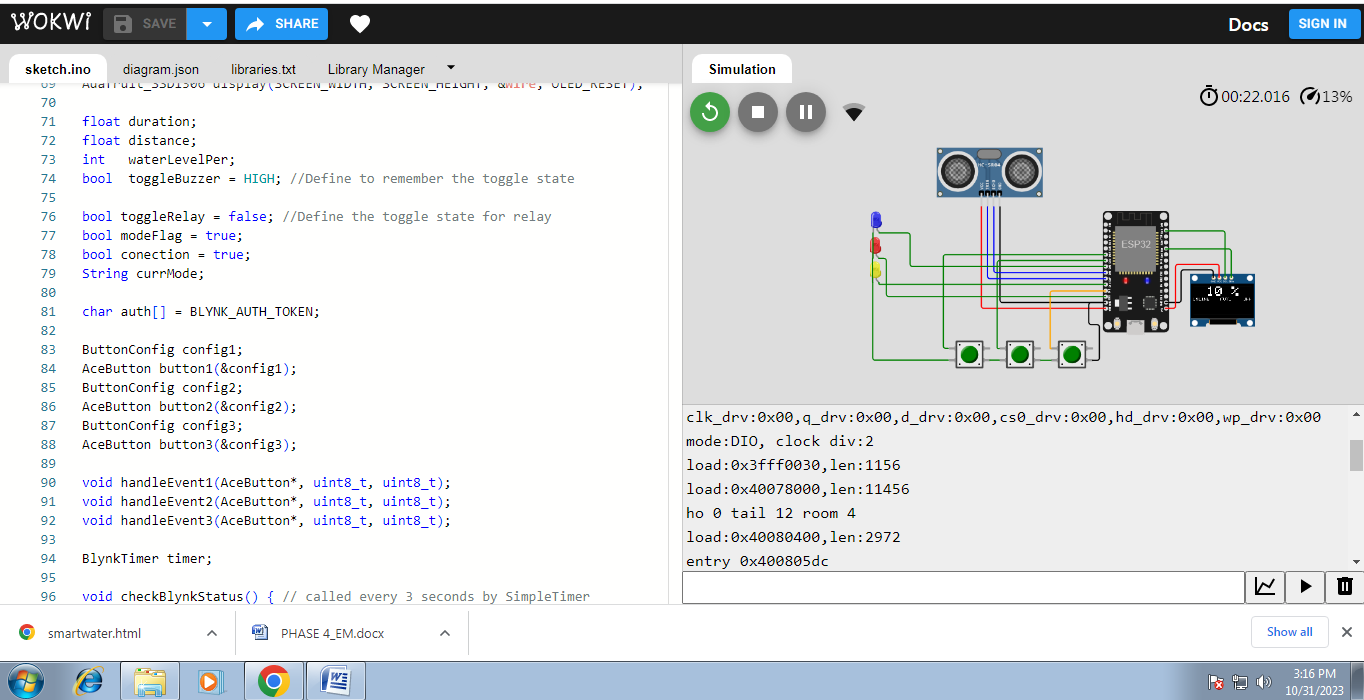
break;

}

}

**CHAPTER 4 :**

**RESULT:**

****

**Conclusion:**

In conclusion, a smart water management system is a crucial and innovative solution for addressing the growing challenges of water scarcity and sustainability. By leveraging advanced technologies like IoT, data analytics, and automation, such a system can optimize water usage, detect leaks, and improve overall efficiency in water distribution. This not only conserves a precious resource but also reduces costs and enhances the quality of life for communities. As we continue to face water-related issues, investing in smart water management systems is a forward-looking and sustainable choice that can benefit both the environment and society.