SMART WATER MANAGEMENT

PROBLEM DEFINITION:

The project's overarching goal is to advance water management practices by deploying IoT sensors in public areas like parks and gardens, facilitating real-time monitoring of water consumption. It encompasses several critical phases, beginning with the definition of clear project objectives. Subsequently, it involves the meticulous design and deployment of an IoT sensor system tailored for outdoor environments, followed by the development of a secure and user-friendly data-sharing platform in the cloud. The final pivotal step involves seamless integration of these components using Python-based IoT technology. By making real-time water consumption data accessible to the public, this comprehensive water management system empowers communities and local authorities to implement informed conservation strategies, contributing to sustainable and responsible water resource management.

PROJECT OBJETIVES:

The project's primary objectives encompass real-time water consumption monitoring in public spaces like parks and gardens, thereby enabling data-driven insights into water use patterns. It aims to enhance public awareness of water conservation by making this real-time consumption data readily accessible. Through these efforts, the project seeks to encourage responsible water usage and contribute to sustainable resource management practices, ultimately promoting the efficient and environmentally conscious stewardship of this vital natural resource.

IOT SENSORS DESIGN:

To design and deploy IoT sensors for monitoring water consumption in public places within the water management system, the approach should focus on selecting rugged, weatherproof sensors capable of measuring flow rates and related data accurately. Strategic sensor placement in proximity to water sources should be considered, along with a reliable power supply strategy, potentially including solar panels for sustainability. The sensors should use secure communication protocols to transmit data to the central platform, and robustdata validation mechanisms should be implemented to ensure data accuracy. Durable enclosures will protect sensors from environmental factors, and remote monitoring capabilities will facilitate timely maintenance. Scalability should be built into the design to accommodate future expansion, while compliance with local regulations and standards ensures the system's legality and effectiveness in promoting water conservation and sustainable resource management.

Sensor Types:

Common sensor types for smart water management include:

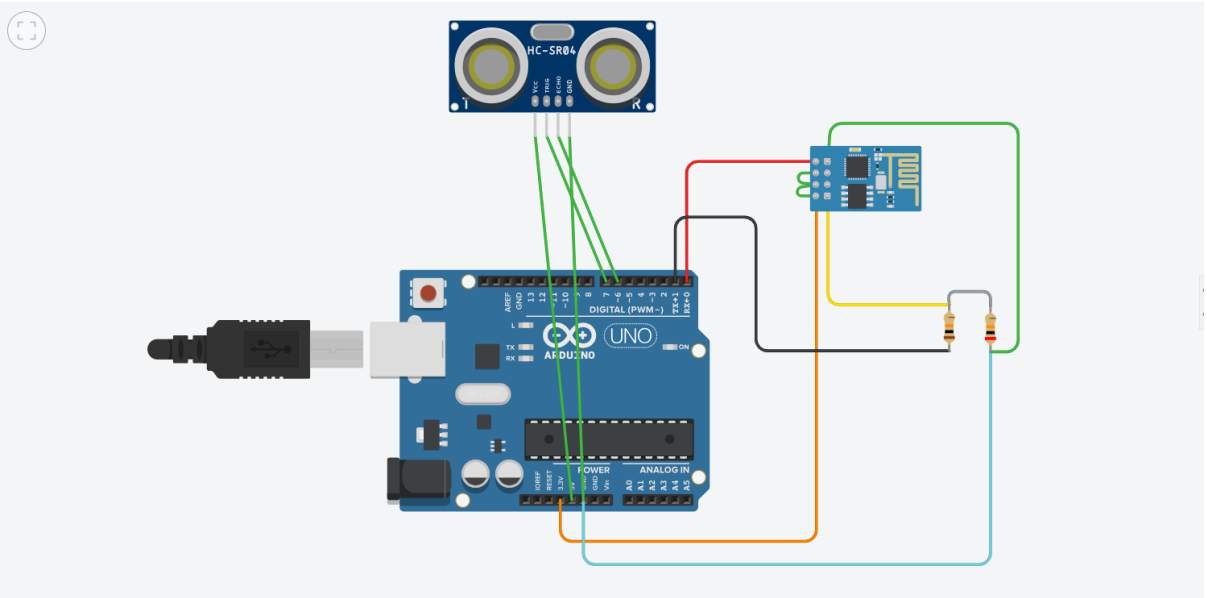
- Water quality sensors (for parameters like pH, turbidity, conductivity, dissolved oxygen, and contaminants).
- Flow meters (ultrasonic, electromagnetic, or mechanical) for measuring water flow rates.
- Level sensors for monitoring water levels in reservoirs, tanks, and pipes.
- Pressure sensors to monitor water pressure in distribution systems.
- Temperature sensors for tracking temperature variations.

PYTHON CODE:

```
String ssid     = "Simulator Wifi";  // SSID to connect to
String password = ""; // Our virtual wifi has no password
String host     = "api.thingspeak.com"; // Open Weather Map API
const int httpPort   = 80;
String uri     = "/update?api_key=ZL866TD4PE5NEJV6&field1=";

int setupESP8266(void) {
  // Start our ESP8266 Serial Communication
  Serial.begin(115200);   // Serial connection over USB to computer
  Serial.println("AT");   // Serial connection on Tx / Rx port to ESP8266
  delay(10);        // Wait a little for the ESP to respond
  if (!Serial.find("OK")) return 1;

  // Connect to 123D Circuits Simulator Wifi
  Serial.println("AT+CWJAP=\"" + ssid + "\",\"" + password + "\"");
  delay(10);        // Wait a little for the ESP to respond
  if (!Serial.find("OK")) return 2;

  // Open TCP connection to the host:
  Serial.println("AT+CIPSTART=\"TCP\",\"" + host + "\"," + httpPort);
  delay(50);        // Wait a little for the ESP to respond
  if (!Serial.find("OK")) return 3;

  return 0;
}

int distanceThreshold = 0;
int cm = 0;
int inches = 0;
long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time in microseconds
  return pulseIn(echoPin, HIGH);
```

```
}
void loop()
{
  // set threshold distance to activate LEDs
  distanceThreshold = 350;
  // measure the ping time in cm
  cm = 0.01723 * readUltrasonicDistance(7, 6);
  // convert to inches by dividing by 2.54
  inches = (cm / 2.54);
  Serial.print(cm);
  Serial.print("cm, ");
  Serial.print(inches);
  Serial.println("in");

  // Construct our HTTP call
  String httpPacket = "GET " + uri + String(temp) + " HTTP/1.1\r\nHost: " +
host + "\r\n\r\n";
  int length = httpPacket.length();

  // Send our message length
  Serial.print("AT+CIPSEND=");
  Serial.println(length);
  delay(10); // Wait a little for the ESP to respond if (!Serial.find(">")) return -1;

  // Send our http request
  Serial.print(httpPacket);
  delay(10); // Wait a little for the ESP to respond
  if (!Serial.find("SEND OK\r\n")) return;
}

void loop() {

 anydata();

  delay(10000);
}
  delay(100); // Wait for 100 millisecond(s)
}
```

TINKERCAD SIMULATION:



OUTPUT:

Text

1 (Arduino Uno R3)

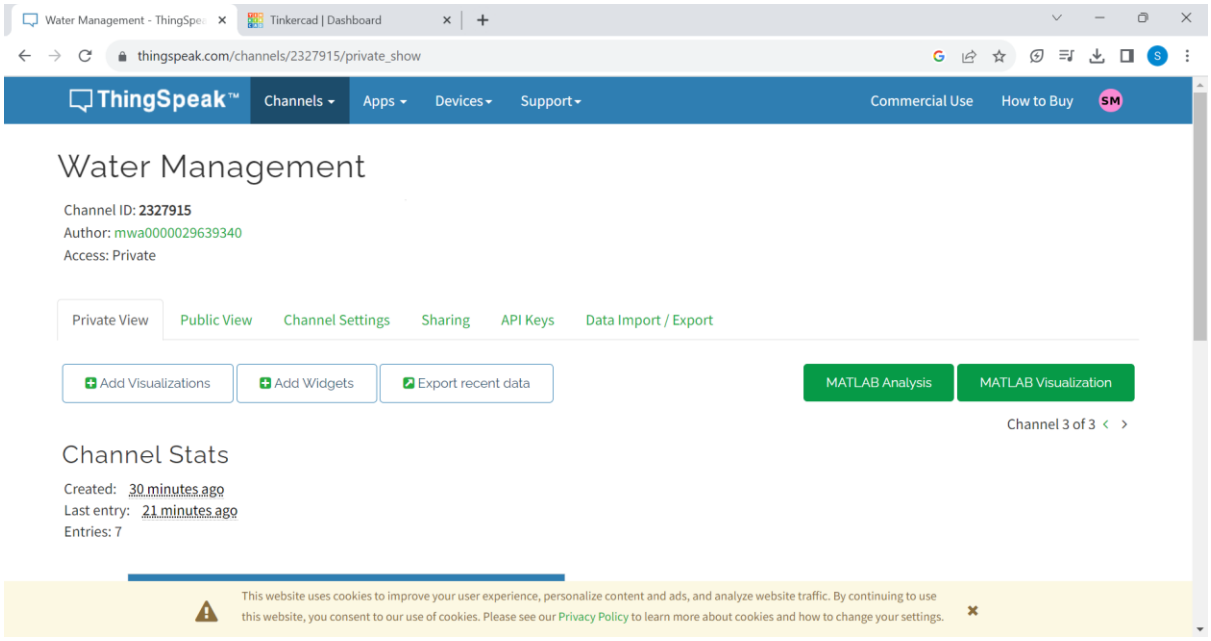
```
1  int distanceThreshold = 0;
2  int cm = 0;
3  int inches = 0;
4  long readUltrasonicDistance(int triggerPin, int echoPin)
5  {
6    pinMode(triggerPin, OUTPUT);  // Clear the trigger
7    digitalWrite(triggerPin, LOW);
8    delayMicroseconds(2);
9    // Sets the trigger pin to HIGH state for 10 microseconds
10   digitalWrite(triggerPin, HIGH);
11   delayMicroseconds(10);
12   digitalWrite(triggerPin, LOW);
13   pinMode(echoPin, INPUT);
14   // Reads the echo pin, and returns the sound wave travel time i
15   return pulseIn(echoPin, HIGH);
```

Serial Monitor

```
146cm, 57in
146cm, 57in
146cm, 57in
146cm, 57in
146cm, 57in
146cm, 57in
```

Send Clear

THINKSPEAK CHANNEL :



OUTPUT: