```
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>

// Initialize SoftwareSerial and LiquidCrystal
SoftwareSerial sserial(12, 13);
LiquidCrystal lcd(9, 8, 7, 6, 5, 4);

// Define pin constants
#define turbidityPin A2
#define valvePin 11
#define flowPin 2

// Global variables for tracking measurements and states
float turbidity = 0, flowRate = 0;
bool leak = 0, valve = 0;
int waterLimit = 10, waterUsed = 0;
volatile byte pulseCount;

// Function to handle pulse count from the flow sensor
void pulseCounter() {
  pulseCount++;
}

// Function to set up the initial state of the system
void setup() {
  Serial.begin(115200);
  sserial.begin(9600);

  pinMode(valvePin, OUTPUT);
```

```
  digitalWrite(valvePin, 0);

  lcd.begin(16, 2);
  lcd.setCursor(0, 0);
  lcd.print("  Water Meter   ");
  delay(5000);
  lcd.clear();

  digitalWrite(valvePin, 1);

  pinMode(flowPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(flowPin), pulseCounter, FALLING);
}

// Main loop to continuously check and update the system state
void loop() {
  Turbidity();
  lcdScr();
  flow();
  leakage();
  sendData();
}

// Function to detect and handle leakage
void leakage() {
  static uint32_t time3;

  if (millis() - time3 > 60000) {
    digitalWrite(valvePin, 1);
```

```
   if (millis() - time3 > 65000) {
     bool leak_ = (flowRate > 0.00f) ? 1 : 0;

     if (millis() - time3 > 65000) {
       leak = leak_;
       time3 = millis();
     }
   }
  } else {
   valve = (waterUsed >= waterLimit || turbidity >= 3000 || leak) ? 0 : 1;
   digitalWrite(valvePin, (valve) ? 0 : 1);
  }
}

// Function to calculate and display flow rate and water usage
void flow() {
  static uint32_t previousMillis, totalMilliLitres;
  static float totalLitres;

  if (millis() - previousMillis > 1000) {
   uint16_t pulse1Sec = pulseCount;
   pulseCount = 0;

   flowRate = ((1000.0 / (millis() - previousMillis)) * pulse1Sec) / 4.5;
   previousMillis = millis();

   uint32_t flowMilliLitres = (flowRate / 60) * 1000;
   float flowLitres = (flowRate / 60);
```

```
    totalMilliLitres += flowMilliLitres;
    totalLitres += flowLitres;
    waterUsed = totalLitres;

    Serial.println(flowRate);
    Serial.println(waterUsed);
  }
}

// Function to measure and calculate turbidity
void Turbidity() {
  float volt = 0;
  for (int i = 0; i < 100; i++) {
    volt += ((float)analogRead(turbidityPin) / 1023) * 5;
    delay(5);
  }
  volt = volt / 100;
  volt = round(volt * 100) / 100; // rounding to 2 decimal places

  if (volt < 2.5) turbidity = 3000;
  else turbidity = -1120.4 * sq(volt) + 5742.3 * volt - 4353.8;
}

// Function to update the LCD screen with current data
void lcdScr() {
  static uint32_t time1;
  static byte screen = 0;
```

```
if (millis() - time1 > 3000) {
  lcd.clear();
  time1 = millis();
  screen++;
  if (screen > 2) screen = 0;
}

switch (screen) {
  case 0:
    lcd.setCursor(0, 0);
    lcd.print("   Turbidity    ");
    lcd.setCursor(0, 1);
    lcd.print("  ");
    lcd.print(turbidity);
    lcd.print(" NTU");
    break;

  case 1:
    lcd.setCursor(0, 0);
    lcd.print((leak) ? "Leakage Detected" : "   No Leakage   ");
    lcd.setCursor(0, 1);
    lcd.print((valve) ? "  Valve Opened  " : "  Valve Closed  ");
    break;

  case 2:
    lcd.setCursor(0, 0);
    lcd.print("H2O limit: ");
    lcd.print(waterLimit);
    lcd.print("L ");
```

```
    lcd.setCursor(0, 1);
    lcd.print("H2O Used: ");
    lcd.print(waterUsed);
    lcd.print("L ");
    break;
  }
}

// Placeholder function to send data, implement as needed
void sendData() {
  // Implement data sending logic here
```