# **Implementation Plan for Portable Water Quality Monitoring Device**

## **Introduction**

Clean water is something we often take for granted, but for many people around the world, it is a daily challenge. That is why I am excited about this project: a **Portable Water Quality Monitoring Device** built with Arduino. It is designed to measure water quality - specifically turbidity (how clear the water is) and temperature - and share that info in real time using the Blynk app. This ties directly into **UN Sustainable Development Goal 6: Clean Water and Sanitation and 3: Good Health & Well-being**, offering a low-cost, portable tool to help communities ensure their water is safe. Over the next few pages, I will walk you through the equipment, parts, software setup, code samples, and diagrams we used to bring this idea to life.

## **Equipment and Parts**

Building this device required some trial and error to get the right components working together. Here’s what we ended up with:

* **Arduino Boards:**
  + We started with the **Arduino MKR WIFI 1010**, which seemed promising because of its built-in Wi-Fi. But here’s the catch: it runs at 3.3V, and our turbidity sensor needs 5V to operate properly. After scratching our heads a bit, we switched to the **Arduino Uno R4** and **Arduino Yun R2**, both of which deliver a solid 5V output - perfect for our needs.
* **Sensors:**
  + **SEN0189 Turbidity Sensor:** This little gem measures how murky or clear the water is. Turbidity is a big deal because cloudy water can signal contaminants like dirt or bacteria.
  + **Waterproof DS18B20 Digital Temperature Sensor:** This sensor gives us accurate water temperature readings. Temperature matters because it can influence how pollutants behave in water.
  + **Jopto TSW-30:** As we thought the previous Turbidity sensor was faulty, we decided to purchase another one, which we thought was more reliable. Although it was easier to install, but did not produce any real readings.
* **Other Bits and Pieces:**
  + **Breadboard:** Great for prototyping without soldering - makes life easier when you are testing things out.
  + **Jumper Wires:** These connect everything together, keeping the setup flexible.
  + **4.7K Ohm Resistors:** Essential for the DS18B20 temperature sensor. It needs a pull-up resistor on its data line to communicate reliably with the Arduino.

## **APIs and Software**

To make this device more than just a box with wires, we hooked it up to **Blynk.io** for real-time data sharing. Blynk is an IoT platform that provides a library for Arduino, letting it send sensor data to their servers and then to your phone. That library acts like an API because it gives us a structured way to push data from the Arduino to the app. We will use it to display turbidity and temperature readings and even send alerts if the water quality dips. For the full Blynk setup, we would need extra libraries and an auth token, but it is rather future improvement.

## **Code Samples**

Now, here is the coding part. Below are two basic code snippets to read data from our sensors. Those were our starting point to test the sensors.

### **Turbidity Sensor Code**

*const int turbidityPin = A0; // Analog input for turbidity sensor*

*void setup() {*

*Serial.begin(9600);*

*}*

*void loop() {*

*int sensorValue = analogRead(turbidityPin);*

*// Convert the analog value to a voltage (assumes 5V operating voltage)*

*float voltage = sensorValue \* (5.0 / 1023.0);*

*Serial.print("Turbidity Voltage: ");*

*Serial.println(voltage);*

*delay(2000);*

*}*

*A screenshot of a computer program

AI-generated content may be incorrect.*

This code reads the analog signal from the SEN0189, converts it to a voltage, and maps it to a turbidity scale. Some tweaking to the mapping might be needed based on real-world testing, but it is a solid starting point.

### **Temperature Sensor Code**

#include <OneWire.h>

#include <DallasTemperature.h>

// Data wire is plugged into digital pin 2 on the Arduino R4 Uno

#define ONE\_WIRE\_BUS 2

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

void setup() {

Serial.begin(9600);

sensors.begin();

}

void loop() {

sensors.requestTemperatures();

float tempC = sensors.getTempCByIndex(0);

Serial.print("Temperature: ");

Serial.print(tempC);

Serial.println(" °C");

delay(2000);

}

A screenshot of a computer program

AI-generated content may be incorrect.

This snippet uses the OneWire and DallasTemperature libraries (grab them from the Arduino IDE Library Manager) to pull temperature data from the DS18B20. It’s super reliable and easy to expand.

**Integration with Blynk.io**

Below is an example code we used to send sensor data to the Blynk platform:

#include <BlynkSimpleEsp8266.h>

// Authentication token from Blynk

char auth[] = "YourBlynkAuthToken";

char ssid[] = "YourWiFiSSID";

char pass[] = "YourWiFiPassword";

void setup() {

Serial.begin(9600);

Blynk.begin(auth, ssid, pass);

}

void loop() {

Blynk.run();

// Assume we have functions getTemperature() and getTurbidity()

float temperature = getTemperature();

int turbidityValue = getTurbidity();

// Send values to Blynk app

Blynk.virtualWrite(V1, temperature);

Blynk.virtualWrite(V2, turbidityValue);

delay(2000);

}

A screenshot of a computer program

AI-generated content may be incorrect.

## **Connection Diagrams**

### **Turbidity Sensor Connection**

Image Source: [here](https://circuitdigest.com/microcontroller-projects/measuring-turbidity-of-water-to-determine-water-quality-using-arduino-turbidity-sensor#:~:text=Interfacing%20Turbidity%20Sensor%20with%20Arduino%20%E2%80%93%20Circuit%20Diagram&text=This%20is%20a%20very%20simple,D2%2C%20D3%2C%20and%20D4.)

* **VCC:** Hook this to the 5V pin on the Arduino.
* **GND:** Connect to any GND pin on the Arduino.
* **Signal:** Plug this into analog pin A0. That’s where the Arduino reads the turbidity data.

### **Temperature Sensor Connection**

A circuit board with wires

AI-generated content may be incorrect.

Image Source: [here](https://lastminuteengineers.com/ds18b20-arduino-tutorial/#:~:text=To%20avoid%20overheating%20and%20damage,the%20DS18B20%20is%20connected%20properly.&text=If%20you're%20using%20the,the%20data%20and%20the%205V.)

* **VCC:** Connect to 5V on the Arduino.
* **GND:** Goes to GND.
* **Data:** Attach to digital pin 2. Add a 4.7K resistor between the data pin and VCC as a pull-up—it keeps the signal stable.

## **Putting It All Together**

Here’s how the system comes together:

1. **Assembly:**
   1. Wire up the sensors to the Arduino Uno R4 or Yun R2 as described. Use the breadboard to keep things organised.
2. **Blynk Setup:**
   1. Download the Blynk app, create a project, and pick your Arduino model.
   2. Grab the auth token it gives us.
   3. In the Arduino IDE, install the Blynk library,
   4. Add virtual pins (e.g., Blynk.virtualWrite(V1, turbidity)) to send data to the app.
3. **App Interface:**
   1. In Blynk, add **Value Display** widgets for turbidity and temperature.
   2. Toss in a **Gauge** widget for a cool visual.
   3. Set up a **Notification** widget to ping your phone if readings go off (e.g., turbidity > 10).
4. **Feedback Rules:**
   1. **Turbidity:** < 5 = “Safe,” 5-10 = “Moderate,” > 10 = “Unsafe.”
   2. **Temperature:** < 25°C = “Good,” 25-35°C = “OK,” > 35°C = “Check Quality.”
5. **Testing:**
   1. Dip the sensors in water in a transparent glass, because the photons should be able to go through it.
   2. Check the readings in the Blynk app and tweak the code or thresholds if needed.

A circuit board with wires and a computer

AI-generated content may be incorrect.A circuit board with wires

AI-generated content may be incorrect.

### **Summary**

The Arduino Water Quality System is designed to be a low-cost, portable device for monitoring water quality, aligning with the goals of sustainable development. The project combines embedded systems expertise with modern IoT platforms, leveraging sensors such as the Waterproof DS18B20 for temperature and SEN0189 for turbidity. Initial challenges, such as voltage mismatches with the Arduino MKR WIFI 1010, led to the adoption of boards like the Arduino R4 Uno and R2 Yun, which better suit the sensors' operating requirements.