Table of Contents

[**1** Team Report: Main Page 3](#_Toc197365742)

[**2** Links to Any Data Gathered or Data Analysis Planned 4](#_Toc197365743)

[**2.1** Water Quality and Contamination: 5](#_Toc197365744)

[**2.2** Sensor-Based Water Monitoring: 5](#_Toc197365745)

[**2.3** Blynk for IoT Water Monitoring: 5](#_Toc197365746)

[**3** Outline of the Problem (Supported by Research) 6](#_Toc197365747)

[**3.1** Ensuring Access to Clean Drinking Water (UN Sustainable Development Goal 6 and 3) 6](#_Toc197365748)

[**3.2** IoT-Based Water Quality Monitoring as a Solution 7](#_Toc197365749)

[**3.3** Our Project's Contribution 7](#_Toc197365750)

[**3.4** References 7](#_Toc197365751)

[**4** Summary of the Project solution 8](#_Toc197365752)

[**4.1** Hardware and Component Overview 8](#_Toc197365753)

[**4.2** Software and Data Transmission via Blynk.io 9](#_Toc197365754)

[**4.2.1** Temperature Sensor Sample Code: 9](#_Toc197365755)

[**4.2.2** Turbidity Sensor Sample Code: 10](#_Toc197365756)

[**4.3** Implementation Workflow 11](#_Toc197365757)

[**4.4** Conclusion and Future Extensions 11](#_Toc197365758)

[**5** List of Project Requirements 12](#_Toc197365759)

[**6** Initial Design - Code Templates, Photos of the Setup, Data Processing through Blynk 15](#_Toc197365760)

[**6.1** Code Templates 16](#_Toc197365761)

[**6.1.1** Code for Blynk 16](#_Toc197365762)

[**6.1.2** Code for Temperature sensor 17](#_Toc197365763)

[**6.2** Setup photos 17](#_Toc197365764)

[**7** Description of data generated, datasets or APIs used and how you plan to store and manage data 18](#_Toc197365765)

[**7.1** Description of Data Generated 18](#_Toc197365766)

[**7.2** Datasets or APIs Used 18](#_Toc197365767)

[**8** Implementation Plan for Portable Water Quality Monitoring Device 19](#_Toc197365768)

[**8.1** Introduction 19](#_Toc197365769)

[**8.2** Equipment and Parts 19](#_Toc197365770)

[**8.3** APIs and Software 20](#_Toc197365771)

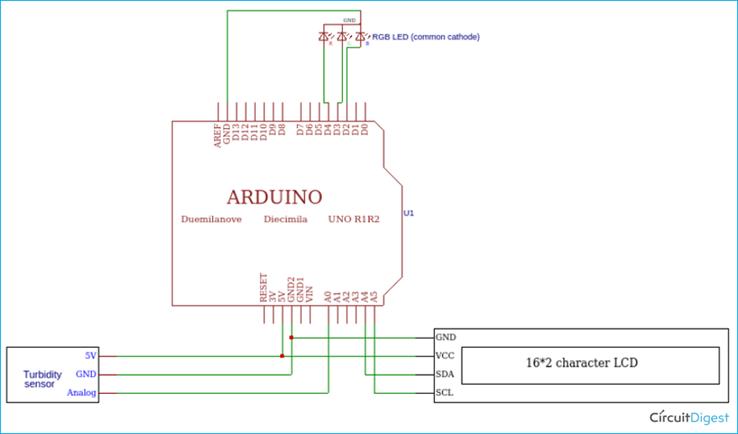
[**8.4** Code Samples 20](#_Toc197365772)

[**8.4.1** Turbidity Sensor Code 20](#_Toc197365773)

[**8.4.2** Temperature Sensor Code 21](#_Toc197365774)

[**8.4.3** Integration with Blynk.io 22](#_Toc197365775)

[**8.5** Connection Diagrams 24](#_Toc197365776)

[**8.5.1** Turbidity Sensor Connection 24](#_Toc197365777)

[**8.5.2** Temperature Sensor Connection 24](#_Toc197365778)

[**8.6** Putting It All Together 25](#_Toc197365779)

[**8.6.1** Final Set Up 26](#_Toc197365780)

[**9** Testing approach 26](#_Toc197365781)

[**9.1** Testing & Evidence 26](#_Toc197365782)

[**9.1.1** Testing Methods 26](#_Toc197365783)

[**9.1.2** Turbidity Sensor Testing 27](#_Toc197365784)

[**9.1.3** Combination of Parts Together 27](#_Toc197365785)

[**9.2** Results 28](#_Toc197365786)

[**9.3** Challenges met during the testing 28](#_Toc197365787)

[**10** Security Analysis to prevent security holes 29](#_Toc197365788)

[**10.1** Security 29](#_Toc197365789)

[**11** Future improvements planned and potential next steps in developing the idea further 29](#_Toc197365790)

[**11.1** Future Steps 29](#_Toc197365791)

[**12** Smart Water Quality Monitoring System: Final Project Report Update 30](#_Toc197365792)

[**12.1** Internet-functionality planned and implemented 30](#_Toc197365793)

[**12.1.1** Google Sheets 31](#_Toc197365794)

[**12.1.2** AQICN API 32](#_Toc197365795)

[**12.1.3** Code Samples: Arduino IDE 34](#_Toc197365796)

[**12.1.4** Code Samples: Google Apps Script 39](#_Toc197365797)

[**12.2** Secondary research on similar internet projects found online 43](#_Toc197365798)

[**12.2.1** Research Findings on Similar IoT Water Quality Projects 43](#_Toc197365799)

[**12.2.2** Implementation Insights from Research 44](#_Toc197365800)

[**12.2.3** References for Research part: 44](#_Toc197365801)

[**12.3** Ideas for the potential future application of Machine Learning to enhance the system’s functionality: ChatGPT API, Simple ML with sensor readings data. 45](#_Toc197365802)

[**12.3.1** ChatGPT API 45](#_Toc197365803)

[**12.3.2** Simple ML 46](#_Toc197365804)

[**13** Academic-Style Poster 49](#_Toc197365805)

[**14** Critical Analysis 49](#_Toc197365806)

[**14.1** System Impact 50](#_Toc197365807)

[**14.2** Key Assumptions & What We Learned 50](#_Toc197365808)

[**14.3** New Perspective 50](#_Toc197365809)

[**14.4** What We’d Do Differently 50](#_Toc197365810)

[**14.5** Final Reflection 51](#_Toc197365811)

[**15** Ethical Risk Analysis 52](#_Toc197365812)

[**16** Video Demonstration 53](#_Toc197365813)

[**16.1** Link to the video 53](#_Toc197365814)

# Team Report: Main Page

* + Project Title: Portable Water Quality Monitoring Device
  + Your name(s) & email addresses:

Adriana Prepeshniuk [S00273868@atu.ie](mailto:S00273868@atu.ie)

Roman Ilkevych [S00274252@atu.ie](mailto:S00274252@atu.ie)

Yelyzaveta Kareieva [S00267553@atu.ie](mailto:S00267553@atu.ie)

Valerii Navalnyi [S00274450@atu.ie](mailto:S00274450@atu.ie)

Darragh Cawley [S00270878@atu.ie](mailto:S00270878@atu.ie)

* + Links to:

Trello Page  [Water Quality Checker Device | Trello](https://trello.com/b/AU3PxAXM/arduino-project-10)

GitHub Team Page [Valeryschka/ArduinoWaterQualitySystem](https://github.com/Valeryschka/ArduinoWaterQualitySystem)

# Links to Any Data Gathered or Data Analysis Planned

To support our project on water quality monitoring, we have reviewed academic literature on water contamination, sensor-based water monitoring, and IoT applications for real-time water quality assessment. The following sources provide background on the problem and possible solutions:

## Water Quality and Contamination:

* WHO. (2017). "Guidelines for Drinking-water Quality." World Health Organization. Available at: <https://www.who.int/publications/i/item/9789241549950>
  + Sankhla, M.S., Kumari, M., Nandan, M., Kumar, R. and Agrawal, P., 2016. Heavy metals contamination in water and their hazardous effect on human health-a review. *Int. J. Curr. Microbiol. App. Sci*, *5*(10), pp.759-766. Available at: <https://www.ijcmas.com/abstractview.php?ID=1020&vol=5-10-2016&SNo=82>

## Sensor-Based Water Monitoring:

* + Lakshmikantha, V., Hiriyannagowda, A., Manjunath, A., Patted, A., Basavaiah, J. and Anthony, A.A., 2021. IoT based smart water quality monitoring system. *Global Transitions Proceedings*, *2*(2), pp.181-186. Available at:
  + Koditala, N.K. and Pandey, P.S., 2018, August. Water quality monitoring system using IoT and machine learning. In *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)* (pp. 1-5). IEEE. Available at: <https://ieeexplore.ieee.org/abstract/document/8509050>
  + Kelechi, A.H., Alsharif, M.H., Anya, A.C.E., Bonet, M.U., Uyi, S.A., Uthansakul, P., Nebhen, J. and Aly, A.A., 2021. Design and Implementation of a Low-Cost Portable Water Quality Monitoring System. *Computers, Materials & Continua*, *69*(2). Available at: <https://cdn.techscience.cn/ueditor/files/cmc/TSP_CMC_69-2/TSP_CMC_18686/TSP_CMC_18686.pdf>
  + Forhad, H.M., Uddin, M.R., Chakrovorty, R.S., Ruhul, A.M., Faruk, H.M., Kamruzzaman, S., Sharmin, N., Jamal, A.S.I.M., Haque, M.M.U. and Morshed, A.M., 2024. IoT based real-time water quality monitoring system in water treatment plants (WTPs). *Heliyon*, *10*(23). Available at: <https://www.sciencedirect.com/science/article/pii/S2405844024167771>
  + Hong, W.J., Shamsuddin, N., Abas, E., Apong, R.A., Masri, Z., Suhaimi, H., Gödeke, S.H. and Noh, M.N.A., 2021. Water quality monitoring with arduino based sensors. *Environments*, *8*(1), p.6. Available at: <https://www.mdpi.com/2076-3298/8/1/6>

## Blynk for IoT Water Monitoring:

* + Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future Generation Computer Systems*, 29(7), 1645-1660. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S0167739X13000241?via%3Dihub>
  + Bayborodin, P. (2014) *Manual device activation*, *Blynk Documentation*. Available at: <https://docs.blynk.io/en/getting-started/activating-devices/manual-device-activation>

# Outline of the Problem (Supported by Research)

## Ensuring Access to Clean Drinking Water (UN Sustainable Development Goal 6 and 3)

Access to clean and safe drinking water is a fundamental human right, yet millions of people worldwide lack consistent access to potable water. According to the World Health Organization (WHO, 2017), approximately 785 million people lack access to basic drinking water services, and over 2 billion people use contaminated water sources that pose severe health risks.

Contaminated drinking water is a leading cause of waterborne diseases such as cholera, typhoid, and diarrhea. The presence of turbidity, chemical pollutants, and microbial contaminants significantly reduces water quality. According to Sankhla et al. (2016), heavy metals such as lead and arsenic, commonly found in untreated water, pose serious health threats, including neurological and developmental disorders.

To address this challenge, the United Nations Sustainable Development Goal 6 (Clean Water and Sanitation) aims to ensure universal and equitable access to safe drinking water by 2030, as well as 3 (Good Health and Well-being) to ensure safe drinking water as part of it. One effective approach to achieving this goal is through real-time water quality monitoring using smart sensors and IoT technology.

## IoT-Based Water Quality Monitoring as a Solution

Traditional methods of water testing involve manual sample collection and laboratory analysis, which are time-consuming and not feasible for continuous monitoring. However, advancements in IoT (Internet of Things) technology allow for real-time data collection and remote monitoring of water quality.

Studies such as those by Lakshmikantha et al. (2021) and Koditala & Pandey (2018) highlight the benefits of using low-cost sensors integrated with IoT platforms like Blynk to measure turbidity, temperature, and other key parameters. These systems enable users to receive real-time alerts if water quality falls below safety standards.

## Our Project's Contribution

Our project utilizes an Arduino UNO R4 Wi-Fi, Jopto TSW-30 & SEN0189 Turbidity Sensor, and DS18B20 Waterproof Temperature Sensorto continuously monitor water quality and send data to the Blynk application. This system allows individuals to check the condition of their drinking water remotely, ensuring that it meets safety standards before consumption.

Beyond individual health benefits, our system can be scaled for community-level water monitoring, helping municipalities and organizations ensure water quality in real-time. The collected data can be analysed for long-term trends, identifying pollution sources and enabling proactive measures to improve water safety. By providing accessible and cost-effective water quality monitoring, our project contributes to achieving SDG 6, reducing health risks, and promoting sustainable water management.

## References

* + Lakshmikantha, V., Hiriyannagowda, A., Manjunath, A., Patted, A., Basavaiah, J. and Anthony, A.A., 2021. IoT based smart water quality monitoring system. *Global Transitions Proceedings*, *2*(2), pp.181-186. Available at: <https://www.sciencedirect.com/science/article/pii/S2666285X2100090X>
  + Sankhla, M.S., Kumari, M., Nandan, M., Kumar, R. and Agrawal, P., 2016. Heavy metals contamination in water and their hazardous effect on human health-a review. *Int. J. Curr. Microbiol. App. Sci*, *5*(10), pp.759-766. Available at: <https://www.ijcmas.com/abstractview.php?ID=1020&vol=5-10-2016&SNo=82>
  + WHO. (2017). "Guidelines for Drinking-water Quality." World Health Organization. Available at: <https://www.who.int/publications/i/item/9789241549950>
  + Koditala, N.K. and Pandey, P.S., 2018, August. Water quality monitoring system using IoT and machine learning. In *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)* (pp. 1-5). IEEE. Available at: <https://ieeexplore.ieee.org/abstract/document/8509050>

# Summary of the Project solution

The Portable Water Quality Monitoring Device is an Arduino‐based system designed to provide an affordable and accessible means to assess water quality in real time. This solution aligns with the United Nations Sustainable Development Goal 6, which emphasizes clean water and sanitation for all. The device focuses on two primary water quality parameters: turbidity and temperature. By measuring the clarity of water with a turbidity sensor and its temperature using a Waterproof DS18B20 Digital Temperature Sensor, the system offers a simple, yet effective method for communities to monitor water safety.

## Hardware and Component Overview

Initially, the project was attempted using an Arduino MKR WIFI 1010; however, this board was found to be incompatible with the SEN0189 Turbidity Sensor due to its 3.3V operating voltage. To resolve this issue, the project was reconfigured to use the Arduino R4 Uno and R2 Yun, both of which operate at 5V and are thus compatible with the sensor specifications. The hardware components include:

* Arduino R4 Uno / R2 Yun: These boards ensure proper voltage levels for the sensors.
* Waterproof DS18B20 Digital Temperature Sensor: **Provides accurate temperature readings.**
* SEN0189 Turbidity Sensor (and a backup, Jopto Turbidity Sensor Detection): **Measures water clarity.**
* Breadboard, Jumper Wires, and 4.7K Resistors: These components are used for prototyping and ensuring reliable connections.

A computer chip connected to wires

AI-generated content may be incorrect.

## Software and Data Transmission via Blynk.io

The device’s sensor data is processed by the Arduino board and transmitted in real time to a mobile application via the Blynk.io platform. Blynk serves as an API that facilitates communication between the hardware and the user interface, allowing users to view live readings of water temperature and turbidity. The integration of Blynk.io not only simplifies the display of sensor data but also enables remote monitoring, which is critical in community health applications.

A brief overview of the code is as follows:

### Temperature Sensor Sample Code:

#include <OneWire.h>

#include <DallasTemperature.h>

#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);}

### Turbidity Sensor Sample Code:

const int turbidityPin = A0; // Analog pin for turbidity sensor

void setup() {

Serial.begin(9600);

}

void loop() {

int sensorValue = analogRead(turbidityPin);

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

Serial.print("Turbidity Voltage: ");

Serial.println(voltage);

delay(2000);

}

Additionally, Blynk.io integration is achieved through code similar to the following:

#include <BlynkSimpleEsp8266.h>

char auth[] = "YourBlynkAuthToken";

char ssid[] = "YourWiFiSSID";

char pass[] = "YourWiFiPassword";

void setup() {

Serial.begin(9600);

Blynk.begin(auth, ssid, pass);

}

void loop() {

Blynk.run();

float temperature = getTemperature(); // Function to read DS18B20 data

int turbidityValue = getTurbidity(); // Function to read SEN0189 data

Blynk.virtualWrite(V1, temperature);

Blynk.virtualWrite(V2, turbidityValue);

delay(2000);}

## Implementation Workflow

The overall workflow of the project is straightforward:

1. User Interaction:

The user immerses the sensors into a water sample.

1. Data Acquisition:

The DS18B20 and SEN0189 sensors measure the water temperature and turbidity, respectively.

1. Data Processing:

The Arduino board reads sensor data, calculates necessary parameters (such as converting analog values to voltage), and determines whether the water is safe to drink based on predefined thresholds.

1. Data Transmission:

Processed data is sent to the Blynk.io server, which then updates the mobile application interface, providing immediate feedback to the user.

1. User Feedback:

The device communicates a simple status - such as “Safe,” “Moderate,” or “Unsafe” - to help the user make an informed decision about water consumption.

## Conclusion and Future Extensions

In summary, the Arduino Water Quality System represents an effective, low-cost approach to water quality monitoring. By integrating reliable sensors with Arduino hardware and leveraging the Blynk.io platform for real-time data display, the project addresses a critical need for accessible water testing methods. Future enhancements could include adding additional sensors (such as a TDS sensor for measuring dissolved solids), further refining the data analysis algorithms, and implementing cloud connectivity for remote monitoring and data logging.

This project not only meets the immediate goal of providing a portable water quality assessment tool but also lays the groundwork for future scalability and integration in resource-limited environments. The design is both practical and adaptable, ensuring that communities can access vital information about their water quality, ultimately contributing to improved public health outcomes.

A circuit board with wires

AI-generated content may be incorrect.

# List of Project Requirements

1.The system uses Blynk.io to send data to a user’s phone:

There will be connection with a mobile Blynk-based API for the sake of data being able to be more easily legible. The connection between an Arduino and a Blynk based application should make it more convenient to interpret the data.

2.The system uses Jopto TSW-30 OR SEN0189 Turbidity Sensor to measure the transparency of water.

A turbidity sensor will be attached to the Arduino for the sake of determining the turbidity of a water sample. This is central to the project as the turbidity of water influences its quality noticeably.

3.System uses DS18B20 Waterproof Temperature Sensor to measure the temperature of the water.

The waterproof temperature sensor will be used to determine the temperature of the water sample. This is relevant as temperature is another factor that influences the water’s quality greatly.

4.Detects High/Low Turbidity

The turbidity sensor will determine whether the turbidity of the sample is high or low and it should be able to relay this information to the user.

5.Detects Too High/Low Temperature

The waterproof temperature sensor will determine the temperature of the sample and will be able to determine if the temperature is too high or too low for use. It will be able to relay this information back to the user.

6.Displays some advice for the user, based on Turbidity and Temperature data

The project ultimately is designed for giving the user more information about a sample of water and thus will give easily understood advice based on the finding of the sensors.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ​  Test Case # ​ | Test Case Title ​ | Requi  rement No ​ | Test Steps ​ | Test Data / Screenshot ​ | Expected Result ​ | Test Pass/Fail ​ | Tester ​ | Date ​ |
| 1 ​ | Get data sent to Blynk.​ | 1 | 1.Run the code​  2.Receive data​ | ​A screenshot of a web page  AI-generated content may be incorrect. | Display of turbidity​ | PASS​ | Roman, Adriana, Lisa​ | 13.03.2025​ |
| 2 ​ | Measures water’s turbidity​ | 2 ​ | 1. Connect the sensor to Arduino​  2. Run the code ​  3. Soak the operating sensor in water ​ | ​A circuit board with wires  AI-generated content may be incorrect. | To get a measure of water’s turbidity ​ | PASS​ | Dream Team​ | 4.03.2025​ |
| 3 ​ | Measures water’s temperature​ | 3​ | 1. Connect the sensor to Arduino​  2. Run the code​  3. Soak the operating sensor in water ​ | ​Изображение выглядит как кабель, Электрическая проводка, электроника, Электронная техника  Содержимое, созданное ИИ, может быть неверным. | To get a measure of water’s temperature​  ​ | PASS​ | Dream Team​ | 25.02.2025​ |
| 4 ​ | Detects High/Low Turbidity.​ | 4​ | 1. Soak the sensor in water, then coffee 2. Get different values ​ | ​A circuit board with wires on a table  AI-generated content may be incorrect. | Detect difference in high and low turbidity ​ | FAIL​ | Dream Team​ | 25.02.2025​ |
| 5 ​ | Detects Too High/Low Temperature.​ | 5​ | 1. Soak the sensor in water, then coffee 2. Get different values  ​ | Изображение выглядит как текст, снимок экрана, Шрифт, число  Содержимое, созданное ИИ, может быть неверным. ​ | Detect difference in high and low temperature​ | FAIL​ | Dream Team​ | 28.02.2025​ |
| 6 ​ | Display advice for the user​ | 6​ | 1.Turn on System ​  2. Set up Blynk ​ | ​A screenshot of a computer  AI-generated content may be incorrect. | To display advice for the user​ | FAIL​ | Roman, Adriana​ | 10.03.2025 ​ |

# Initial Design - Code Templates, Photos of the Setup, Data Processing through Blynk

The core of a project is ***Arduino Uno R4 Wi-Fi*** which controls the sensors and communicates with the Blynk platform for real-time data processing and output.

***Jopto TSW-30 & SEN0189 Turbidity Sensors***: These sensors measure the turbidity (cloudiness) of the water. The turbidity sensor outputs a value proportional to how clear or polluted the water is.

***DS18B20 Waterproof Temperature Sensor:*** This sensor is used to measure the water's temperature, which is essential for assessing water quality.

The Turbidity sensor and temperature sensor can be connected to the analog or digital pins of the Arduino. Arduino sends data to Blynk via Wi-Fi, where users can monitor it on a smartphone in real-time.

* Data Processing through Blynk

Blynk API gives us a real-time communication between the Arduino device and the app.

The app displays:

* Turbidity level:

Real-time water clarity reading.

* Temperature:

Real-time water temperature reading.

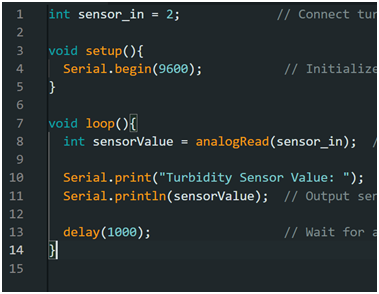
* Safety status:

A message like "Safe to Drink" or "Unsafe to Drink"

The app visualises this data through widgets like Value Display for temperature and turbidity, and Label for the safety status.

## Code Templates

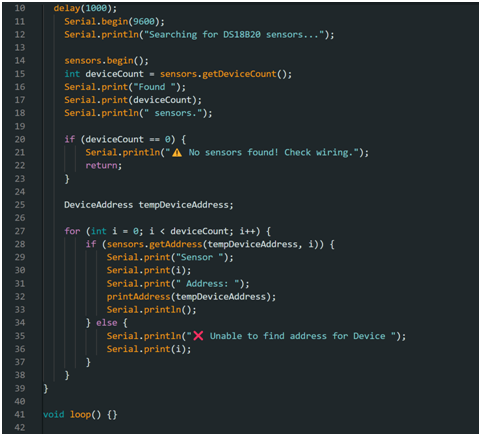
Code for Turbidity sensors



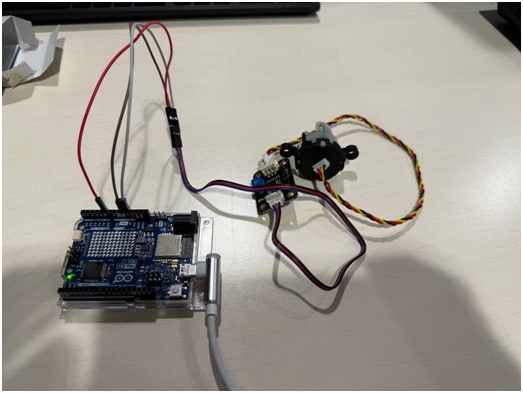
### Code for Blynk



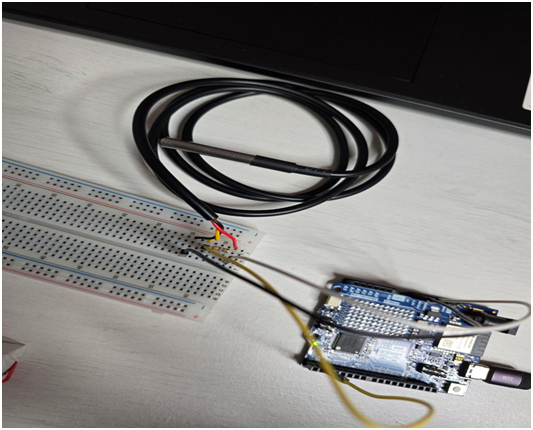
### Code for Temperature sensor



## Setup photos

(Arduino and SEN0189)

( Arduino and Jopto TSW-30)

(Arduino and DS18B20 connected via Breadboard)

# Description of data generated, datasets or APIs used and how you plan to store and manage data

## Description of Data Generated

The main data generated by the system includes:

* Turbidity Level:

A numeric value that represents the clarity of the water (higher values indicate murkier water).

Measurement: Analog reading from the turbidity sensor.

* Temperature:

A floating-point value that represents the water's temperature in °C.

Measurement: DS18B20 sensor provides temperature readings.

* Safety Status:

A message that indicates whether the water is "Safe to Drink" or "Unsafe to Drink" based on turbidity and temperature readings.

## Datasets or APIs Used

**Blynk API**: For real-time data transfer between the sensors and the app. It helps visualize the sensor values and allows for status updates.

**Blynk Cloud:** Data will be stored and processed in the cloud, ensuring it's always accessible from the Blynk app.

**Arduino Data Processing:**Local data processing on the Arduino involves reading sensor values and sending them to Blynk for further analysis.

# 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 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:

* + Wire up the sensors to the Arduino Uno R4 or Yun R2 as described. Use the breadboard to keep things organised.

2. Blynk Setup:

* + Download the Blynk app, create a project, and pick your Arduino model.
  + Grab the auth token it gives us. In the Arduino IDE, install the Blynk library.
  + Add virtual pins (e.g., Blynk.virtualWrite(V1, turbidity)) to send data to the app.

3. App Interface:

* + In Blynk, add Value Display widgets for turbidity and temperature.
  + Toss in a Gauge widget for a cool visual.
  + Set up a Notification widget to ping your phone if readings go off (e.g., turbidity > 10).

4. Feedback Rules:

* + Turbidity: < 5 = “Safe,” 5-10 = “Moderate,” > 10 = “Unsafe.”
  + Temperature: < 25°C = “Good,” 25-35°C = “OK,” > 35°C = “Check Quality.”
  + Dip the sensors in water in a transparent glass, because the photons should be able to go through it.
  + Check the readings in the Blynk app and tweak the code or thresholds if needed.

5. The Set Up:

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.

### Final Set Up

A close-up of a circuit board

AI-generated content may be incorrect.

# Testing approach

## Testing & Evidence

### Testing Methods

The goal is to determine how effectively the sensors can measure different water conditions and provide accurate real-time data.

We initially planned to use an Arduino MKR WiFi 1010 as our microcontroller. However, we quickly discovered that this board only supports 3.3V, whereas our sensors require 5V to function correctly. Due to this voltage incompatibility, we had to switch to the Arduino Uno R4 WiFi, which supports 5V and was expected to work better with our sensors.

Once we had the appropriate microcontroller, we started testing the turbidity sensor (SEN0189) while postponing the setup of the temperature sensor, as it required a resistor that we did not initially have.

For our initial tests, we selected two water samples with significantly different clarity levels:

Tap water – expected to be clean with minimal turbidity.

Coffee – a high-turbidity liquid used to simulate dirty or contaminated water.

These contrasting samples allowed us to evaluate how well the sensor could differentiate between clear and turbid water.

### Turbidity Sensor Testing

To begin, we researched online tutorials and documentation for the SEN0189 turbidity sensor. Most sources suggested connecting it to an analog input on the Arduino. However, when we followed this method, our readings were unexpectedly binary (only 1s and 0s) rather than providing a range of values.

Suspecting a connection issue, we tried switching the sensor to a digital input, which led to slightly better results (0 – 1023). However, the values still did not vary as expected between different water samples. The value stayed the same despite we tried it in different substances. This prompted further troubleshooting, where we investigated how the sensor calculates turbidity values.

We suspected that the SEN0189 turbidity sensor was faulty. To verify this, we ordered a different turbidity sensor, the Jopto TSW-30 Turbidity Sensor, expecting better results. Unfortunately, this new sensor also failed to provide valid readings under different conditions.

After further investigation, we assumed that these turbidity sensors are not compatible with the Arduino Uno R4, so that’s why we decided to switch this sensor to Arduino Yun which we had in our lecture’s cabinet. Unfortunately, this didn’t give us results either.

Temperature Sensor Testing

Once we received the required resistor, we attempted to set up the DS18B20 waterproof temperature sensor. We used multiple wiring configurations and different code variations from online sources.

Despite our efforts, the sensor failed to function correctly. The main errors encountered were:

* "Sensor not detected" – indicating a connection or code issue.
* A constant reading of -127°C – which typically occurs when the sensor is not being detected at all.

At one point, a wiring mistake caused the sensor to become extremely hot, likely due to the wires being placed too close together on the breadboard. This highlighted the importance of precise wiring when working with temperature sensor.

Since the DS18B20 requires a 4.7kΩ pull-upresistor between VCC and the data pin, any deviation from this setup can result in errors. Additionally, ensuring the correct OneWire and DallasTemperature libraries are installed is crucial for the sensor to function.

### Combination of Parts Together

During this phase of the project, we used a completely new setup. We created a grounded circuit on a breadboard to collect data from both the turbidity and temperature sensors.

On our first attempt, incorrect wiring caused the resistor to become extremely hot. After troubleshooting, we identified the issue, repositioned the resistor, and replaced it with a new one. Once the wiring was corrected, we successfully obtained data from both sensors and even managed to send and receive it using the Blynk system.

A row of resistors on a grey surface

AI-generated content may be incorrect.

## Results

As the successful part of our project, we integrated the system with Blynk (an IoT platform) to enable real-time monitoring and remote access to water quality data. This will allow us to display turbidity levels dynamically, improving usability. The sensors were connected but the calibration failed, and they do not match the required values.

## Challenges met during the testing

Throughout the testing phase, we encountered several challenges and areas for improvement:

Microcontroller Compatibility Issues: The Arduino MKR WiFi 1010 only supports 3.3V, making it incompatible with our sensors, which require 5V. The Arduino Uno R4 WiFi, while resolving the voltage issue, turned out to be incompatible with our turbidity sensors, causing ongoing sensor malfunctions.

Turbidity Sensor Difficulties: Initially, the sensor output was just 1s and 0s in analog mode. Switching to digital mode improved results but still lacked proper variation. Introducing a resistor to fix the 0V issue led to fluctuating readings (1-10 and 500-1000), which were incorrect. The Jopto TSW-30 Turbidity Sensor was also tested but did not work correctly.

Temperature Sensor Setup Failure: Despite trying multiple configurations, the DS18B20 sensor was either not detected or displayed -127°C. One incorrect setup caused the sensor to become extremely hot due to wiring mistakes on the breadboard. The sensor requires a 4.7kΩ pull-up resistor to function correctly.

Multiple sources and wiring setups were tested, but none provided a working solution for the temperature sensor.

# Security Analysis to prevent security holes

## Security

As this project does not involve sensitive personal information, the amount of security required is relatively low. For the sake of device security**,** the Arduino IDE was updated to its most recent version. As this project does not contain sensitive information, there is no need for authentication before a user can use the Arduino. We will add an additional layer of verification on the Blynk-based application. For the sake of network security, the number of networks the device was connected to was limited. XOR data encryption will be added to prevent any third party altering the gathered data. Additionally, a checksum function will be added to add a further layer of encryption. In order to have a **secure user interface** and accommodate for the use of Wi-Fi being used to transfer information from the Arduino to the application, HTTPS was used as it is notably more secure than the default HTTP.

# Future improvements planned and potential next steps in developing the idea further

## Future Steps

To improve the accuracy and reliability of our water quality monitoring system, we will attempt to connect a fully functional turbidity sensor. Given our previous challenges with sensor compatibility, we will carefully test different configurations and ensure proper calibration. Additionally, we will test a wider range of water samples beyond just tap water and coffee, including filtered water, rainwater, and potentially water from natural sources such as rivers or lakes. This will allow us to assess the effectiveness of our system under varying conditions.

For the temperature measurement component, we will double-check the DS18B20 sensor wiring and verify that the correct pull-up resistor is used. Since incorrect wiring previously led to sensor malfunctions and overheating, ensuring proper setup is crucial. If necessary, we will explore alternative temperature sensors that may be more compatible with our system.

To enhance the project's functionality, we will continue refining data logging and visualization. One potential improvement is integrating the system with Blynk or a similar IoT platform, allowing for remote monitoring of water quality parameters. This would make our project more practical for real-world applications, enabling users to receive real-time updates on water conditions.

Furthermore, we aim to expand our project by integrating a pH sensor, providing additional insight into water quality. Measuring pH levels alongside turbidity and temperature will give a more comprehensive assessment, making our system more valuable for applications such as environmental monitoring and drinking water safety.

We also plan to enhance the physical design of our project to improve stability and portability. Soldering the wires will ensure more stable connections, reducing the chances of loose or unreliable contacts. Additionally, gluing or securing all the components together in a compact structure will make the device easier to handle and transport. These improvements will contribute to a more durable and professional-looking final product.

Incorporating AI is another step we plan to take. By utilizing machine learning, we can analyse sensor data and improve the accuracy of our assessments based on user needs and environmental conditions. AI-driven algorithms could help detect patterns, predict potential water quality issues, and provide recommendations for corrective actions.

Overall, this project highlights the importance of hardware compatibility, calibration, and troubleshooting in sensor-based systems. By refining our methods, improving the physical design, and expanding our testing conditions, we aim to develop a reliable and practical water quality monitoring tool. Our goal is to create a system that can be used in real-world scenarios, contributing to better water safety and environmental awareness.

# Smart Water Quality Monitoring System: Final Project Report Update

## Internet-functionality planned and implemented

When planning an advancement of the project it was essential to consider the options for its scalability, Internet-part implementation for practicality and in the end – marketability.

“*Modern problems require modern solutions.*” — Dave Chapelle

For these reasons the DreamTeam decided to use Google Sheets for gathering, using, and analysing data. Moreover, it has an easy way of implementing Machine Learning (Simple ML extenstion), which you will see later in the chapter.

### Google Sheets

A green and white logo

AI-generated content may be incorrect.

Figure 1: Google Sheets logo.

Source: [Klipfolio](https://www.klipfolio.com/blog/google-sheets)

Google Spreadsheet was undoubtedly the main choice here, with thorough documentation and tutorials, a reputable and reliable tool for data organisation, analysation, and visualisation, which is a perfect fit for **Smart Water Quality Monitoring System** to gather, analyse and draw conclusions from the data received. The code samples will be shown in the end of this chapter can be accessed from our [Github page](https://github.com/valerkahere/SmartWaterQualityMonitoringSystem/blob/Project2/Code.gs).

A screenshot of a spreadsheet

AI-generated content may be incorrect.

Figure 2: an example of values sent from [Arduino® UNO R4 WiFi every 30 seconds.](https://store.arduino.cc/en-ie/products/uno-r4-wifi?srsltid=AfmBOooJrRuyPMS-K5mGoZAIZgfoANnsmX4ctES2N-qc3Ox0ED19nIpU)

***Note:*** Random values were used as part of the testing because of the hardware issues mainly with the sensors.

This would allow for future application of tools like Simple Machine Learning in Google Sheets extensions to enhance the project functionality. However, for now the focus was on gathering more data from different sources. It is where API comes in place.

### AQICN API

At the very start of the semester, when we as a team were deciding on the project idea itself, the choice was lying between the two options: Air Quality Monitoring Device or Smart Water Quality Monitoring System.­­­­ Well, we sticked to Water Monitoring as the main feature, which would be a more relevant decision in the context of the device being used in underdeveloped countries on a global scale — it is simpler for an individual to change their source of drinking water rather than moving to another less polluted city. Anyway, the Air Quality Monitoring has been a valuable idea and potentially useful device. Therefore, as a way of complementing the project with a worthwhile extension, the Team Representative decided to embed [AQICN](https://aqicn.org/contact/)’s API — **The World Air Quality Index Project** (also operating in Sligo town). It is a non-profit initiative that provides live air quality data and information for more than 130 countries, where the main point is to raise air pollution awareness and offer a unified global view of air quality.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Example of values sent to the spreadsheet by Arduino, utilising AQICN API.

***Note:*** AQI stands for U.S. Air Quality Index. [More information](https://www.airnow.gov/aqi/aqi-basics/#:~:text=How%20does%20the%20AQI%20work,the%20greater%20the%20health%20concern.).

"*Trust, but v­erify.*" — a Russian proverb

The great thing about an external API is that you can always check the actual real-time data on a website.

­­ A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Live time data on [AQICN’s website](https://aqicn.org/city/ireland/sligo-town/)

As a result, external API was a great addition to our project, where users now are not only able to get relevant Water Quality data, but also Air Quality in Sligo town, and this can be changed to any destination point.

### Code Samples: Arduino IDE

Here you can see the full Arduino script, which is available on our [Github repository](https://github.com/valerkahere/ArduinoWaterQualitySystem/blob/Project2/ArduinoSpreadsheet.ino), too.

#include "WiFiS3.h"

#include <WiFiSSLClient.h> // Use WiFiSSLClient for HTTPS

// WiFi Credentials

char ssid[] = "YOUR\_WIFI\_NAME"; // <-- Put your WiFi name here

char pass[] = "YOUR\_WIFI\_PASSWORD"; // <-- Put your WiFi password here

int status = WL\_IDLE\_STATUS; // the WiFi radio's status

// Google Apps Script Web App URL details

const char\* host = "script.google.com";

const int httpsPort = 443;

// MAKE SURE this is your deployed script URL (after new deployment)

String scriptPath = "/macros/s/[YOUR DEPLOYED URL]/exec"; // <-- PASTE SCRIPT URL

// --- DEFINE YOUR THRESHOLDS HERE ---

const long TURBIDITY\_CLEAR\_THRESHOLD = 30; // Example: Turbidity < 50 is "clear"

const float TEMP\_COLD\_THRESHOLD = 15.0;   // Example: Temp < 15C is "cold"

const float TEMP\_GOOD\_MAX\_THRESHOLD = 25.0; // Example: Temp <= 25C (and >=15C) is "good"

// Temperatures > TEMP\_GOOD\_MAX\_THRESHOLD will be "warm"

// --- END OF THRESHOLDS ---

void setup() {

  Serial.begin(9600);

  delay(1000);  // Short delay to stabilize

  // Check for the WiFi module:

  if (WiFi.status() == WL\_NO\_MODULE) {

    Serial.println("Communication with WiFi module failed!");

    while (true);

  }

  // Attempt to connect to WiFi network:

  Serial.print("Connecting to WiFi SSID: ");

  Serial.println(ssid);

  while (status != WL\_CONNECTED) {

    status = WiFi.begin(ssid, pass);

    delay(5000);

  }

  Serial.println("Connected to WiFi!");

  Serial.print("IP Address: ");

  Serial.println(WiFi.localIP());

}

// MODIFIED: Added 'finalResult' parameter

void sendDataToSheet(long turbidity, float temperature, String safetyStatus) {

  WiFiSSLClient client;

  Serial.println("\n------------------------------");

  Serial.print("Connecting to ");

  Serial.println(host);

  if (!client.connect(host, httpsPort)) {

    Serial.println("Connection failed");

    return;

  }

  // MODIFIED: Added 'safetyStatus' parameter to the URL

  String url = scriptPath + "?turbidity=" + String(turbidity) +

               "&temperature=" + String(temperature) +

               "&safetyStatus=" + safetyStatus; // Added safetyStatus parameter

  // Quick URL encoding for safetyStatus (handles potential spaces, though unlikely for SAFE/UNSAFE)

  url.replace(" ", "%20");

  Serial.println("Request URL:");

  Serial.println("https://" + String(host) + url);

  // Improved HTTP request with more headers

  client.print(String("GET ") + url + " HTTP/1.1\r\n" +

               "Host: " + host + "\r\n" +

               "User-Agent: ArduinoWiFi/1.1\r\n" +

               "Connection: close\r\n" +

               "\r\n");

  Serial.println("Request sent. Waiting for response...");

  // Wait for server response with better timeout handling

  unsigned long timeout = millis();

  while (client.available() == 0) {

    if (millis() - timeout > 10000) {

      Serial.println(">>> Response timeout!");

      client.stop();

      return;

    }

  }

  // Process and display response headers

  Serial.println("Response received:");

  String responseStatus = "";

  bool headersDone = false;

  while (client.available()) {

    String line = client.readStringUntil('\n');

    line.trim();

    if (responseStatus == "" && line.startsWith("HTTP/1.")) {

      responseStatus = line;

      Serial.println(responseStatus);

    }

    if (line == "") { headersDone = true; }

    if (headersDone && line.length() > 0) {

      Serial.println("BODY: " + line);

    }

  }

  Serial.println("Connection closed");

  client.stop();

}

void loop() {

  // Generate sensor data (fake values)

  long turbidity = random(95, 100); // Range 0 to 1000

  // clear water 0 to 10,

  // 30 to 70 cloudy

  // coffee 70 to 100

  float temperature = random(70, 72); // Range 0 to 100 C

  Serial.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

  Serial.println("New sensor data:");

  Serial.print("Turbidity: ");

  Serial.print(turbidity);

  Serial.print(", Temperature: ");

  Serial.println(temperature);

  // --- ADDED: Calculate Status Strings ---

  String turbidityStatus;

  String temperatureStatus;

  // Determine Turbidity Status

  if (turbidity < TURBIDITY\_CLEAR\_THRESHOLD) {

    turbidityStatus = "clear";

  } else {

    turbidityStatus = "cloudy"; // Assuming anything not clear is cloudy

  }

  Serial.print("Turbidity Status: ");

  Serial.println(turbidityStatus);

  // Determine Temperature Status

  if (temperature < TEMP\_COLD\_THRESHOLD) {

    temperatureStatus = "cold";

  } else if (temperature <= TEMP\_GOOD\_MAX\_THRESHOLD) {

    temperatureStatus = "good";

  } else {

    temperatureStatus = "warm";

  }

  Serial.print("Temperature Status: ");

  Serial.println(temperatureStatus);

  // --- END OF Status Calculation ---

  // --- IMPLEMENTED: Your Analysis Snippet ---

  String finalResult; // This will hold SAFE, NOT IDEAL, or UNSAFE

  Serial.print("Overall Analysis: ");

  if (turbidityStatus == "clear" && temperatureStatus == "good") {

    finalResult = "SAFE";

    Serial.println("Water is SAFE to drink.");

  } else if (turbidityStatus == "cloudy" && (temperatureStatus == "cold" || temperatureStatus == "good")) {

    finalResult = "NOT IDEAL";

    Serial.println("Water is drinkable but temperature and turbidity are not ideal.");

  } else { // Covers cloudy turbidity OR clear but not good/cold/warm (though that last case is unlikely with current logic)

    finalResult = "UNSAFE";

    Serial.println("Water is UNSAFE to drink.");

  }

  // --- END OF Analysis Snippet ---

  // Send data using Apps Script method

  if (WiFi.status() == WL\_CONNECTED) {

    // MODIFIED: Pass the 'finalResult' to the function

    sendDataToSheet(turbidity, temperature, finalResult);

  } else {

    Serial.println("WiFi Disconnected. Attempting to reconnect...");

    status = WiFi.begin(ssid, pass);

  }

  delay(30000); // Send data every 30 seconds

}

### Code Samples: Google Apps Script

Now, that is the Apps Script code sample, that only works in conjunction with Arduino IDE and connected device itself. See the sample on our [Github](https://github.com/valerkahere/SmartWaterQualityMonitoringSystem/blob/Project2/Code.gs).

// === CONFIGURATION ===

const AQICN\_TOKEN = "79dbdddb9819eeaacc4e4c09a9ba7ac5a9b8ecae"; // Your token

const SPREADSHEET\_ID = '1se7rWs55vOttLGmBw\_8x33rYeCIS2SM\_FHJAbLdqBNM'; // Your Sheet ID

function doGet(e) {

  Logger.log("Received request: " + JSON.stringify(e)); // Log incoming request

  var result = 'Ok';

  var sheet = SpreadsheetApp.openById(SPREADSHEET\_ID).getActiveSheet();

  // MODIFIED: Add "Water Safety" header and adjust column count/check

  const expectedHeaders = ["Date", "Turbidity", "Water Temperature", "Water Safety", "AQI Sligo", "PM2.5 Sligo", "PM10 Sligo", "Temperature Sligo", "Humidity level"]; // Added "Water Safety"

  const headerRange = "A1:I1"; // Now 9 columns

  const headerColumnCount = expectedHeaders.length;

  if (sheet.getLastRow() === 0 || sheet.getRange("A1").getValue() !== expectedHeaders[0] || sheet.getRange("D1").getValue() !== expectedHeaders[3]) {

    // Add headers if first row is empty or first/fourth header doesn't match

    sheet.getRange(headerRange).setValues([expectedHeaders]);

    sheet.getRange(headerRange).setFontWeight("bold").setHorizontalAlignment("center");

    sheet.setFrozenRows(1);

    Logger.log("Headers set/updated.");

  }

  var newRow = sheet.getLastRow() + 1;

  var rowData = [];

  // Column A: Timestamp

  rowData[0] = new Date();

  // Arduino data

  if (e && e.parameter) {

    // Column B: Turbidity

    rowData[1] = e.parameter.turbidity || "";

    // Column C: Temperature (Assuming this was meant to be Temp, header says pH?) - Mapped to pH column based on header

    rowData[2] = e.parameter.temperature || "";

    // --- ADDED: Column D: Water Safety Status ---

    rowData[3] = e.parameter.safetyStatus || ""; // Get the safetyStatus parameter

    // --- END ---

  } else {

    rowData[1] = ""; // Turbidity

    rowData[2] = ""; // Temperature/pH

    rowData[3] = ""; // Water Safety

  }

  // AQICN Data - Indices shifted due to added column

  var aqData = tryMultipleAqiEndpoints();

  // Column E: AQI Sligo

  rowData[4] = aqData.aqi;

  // Column F: PM2.5 Sligo

  rowData[5] = aqData.pm25;

  // Column G: PM10 Sligo

  rowData[6] = aqData.pm10;

  // Column H: Temperature Sligo

  rowData[7] = aqData.temperature;

  // Column I: Humidity level

  rowData[8] = aqData.humidity;

  Logger.log("Row data to write: " + JSON.stringify(rowData));

  // Write new row to spreadsheet - MODIFIED: Use headerColumnCount

  var newRange = sheet.getRange(newRow, 1, 1, headerColumnCount);

  newRange.setValues([rowData]);

  Logger.log("Data written successfully to row " + newRow);

  return ContentService.createTextOutput("Success: Data logged including Safety Status and AQI");

}

// REVISED: Try multiple possible endpoints for Sligo, including sligo-town, with enhanced logging

function tryMultipleAqiEndpoints() {

  const endpoints = [

    // ADDED: The specific endpoint likely tested successfully in the browser

    `https://api.waqi.info/feed/ireland/sligo-town/?token=${AQICN\_TOKEN}`,

    // Existing endpoints

    `https://api.waqi.info/feed/sligo/?token=${AQICN\_TOKEN}`,

    `https://api.waqi.info/feed/ireland/sligo/?token=${AQICN\_TOKEN}`,

    `https://api.waqi.info/feed/geo:54.277;-8.474/?token=${AQICN\_TOKEN}`

  ];

  var lastError = "No endpoints attempted."; // Default error message

  Logger.log("Attempting to fetch AQI data using token: " + AQICN\_TOKEN); // Log the token being used (verify it looks right)

  for (let i = 0; i < endpoints.length; i++) {

    let endpoint = endpoints[i];

    Logger.log(`Attempting endpoint #${i + 1}: ${endpoint}`); // Log the exact URL

    try {

      // Fetch response WITH status code

      const response = UrlFetchApp.fetch(endpoint, {muteHttpExceptions: true});

      const responseCode = response.getResponseCode();

      const responseText = response.getContentText();

      Logger.log(`Endpoint #${i + 1} responded with HTTP status code: ${responseCode}`); // Log the status code

      // Optional: Log the raw response text for debugging, but can be noisy

      // Logger.log(`Endpoint #${i + 1} raw response: ${responseText}`);

      // Check for successful HTTP status (200 OK) before parsing JSON

      if (responseCode === 200) {

        const data = JSON.parse(responseText);

        // Check if the API's internal status is "ok"

        if (data.status === "ok") {

          Logger.log("Successful API call with endpoint: " + endpoint);

          // Return the extracted data

          return {

            aqi: data.data.aqi || "N/A",

            pm25: data.data.iaqi && data.data.iaqi.pm25 ? data.data.iaqi.pm25.v : "N/A",

            pm10: data.data.iaqi && data.data.iaqi.pm10 ? data.data.iaqi.pm10.v : "N/A",

            temperature: data.data.iaqi && data.data.iaqi.t ? data.data.iaqi.t.v : "N/A",

            humidity: data.data.iaqi && data.data.iaqi.h ? data.data.iaqi.h.v : "N/A"

          };

        } else {

          lastError = `API Error (Status ${data.status}) from endpoint: ${endpoint}`;

          Logger.log(`Endpoint #${i + 1} failed - API Status: ${data.status}`);

        }

      } else {

        // Handle non-200 HTTP responses (like 403 Forbidden, 404 Not Found, 429 Too Many Requests, or 5xx Server Errors)

        lastError = `HTTP Error ${responseCode} from endpoint: ${endpoint}`;

        Logger.log(`Endpoint #${i + 1} failed - HTTP Status: ${responseCode}. Response: ${responseText}`);

        // Specifically check for "Invalid key" in the response if it's a 4xx error

        if (responseCode >= 400 && responseCode < 500 && responseText.includes("Invalid key")) {

           Logger.log(">>> Detected 'Invalid key' response from server for this endpoint.");

           // You could decide to stop trying other endpoints if you get an explicit Invalid Key

           // break;

        }

      }

    } catch (err) {

      // Catch errors during the fetch itself (e.g., network timeout, DNS error)

      lastError = `Fetch Exception for endpoint ${endpoint}: ${err.message}`;

      Logger.log(`Endpoint #${i + 1} failed - Exception: ${err}`);

    }

  } // End of loop

  // If all endpoints fail, log the last encountered error and return the error object

  Logger.log("All endpoints failed. Last error: " + lastError);

  return {aqi:"API Error", pm25:"N/A", pm10:"N/A", temperature:"N/A", humidity:"N/A"};

}

// Test function to check API endpoints (No changes needed here)

function testEndpoints() {

  const result = tryMultipleAqiEndpoints();

  Logger.log(JSON.stringify(result));

}

## Secondary research on similar internet projects found online

To validate our implementation approach and ensure we are aligned with best practices in the field, we have examined several similar IoT-based water quality monitoring systems developed in recent years. This secondary research has provided valuable insights into effective approaches and potential enhancements for our project.

### Research Findings on Similar IoT Water Quality Projects

**IoT-based Water Quality Monitoring for Koi Fish**

A 2023 study by researchers developing a water quality monitoring system for Koi fish quarantine presents a remarkably similar approach to ours. Their system utilises NodeMCU ESP32 connected to temperature, pH, and TDS sensors, transmitting data via IoT to the Blynk application. Like our system, they focus on key parameters that affect water quality-temperature, pH, and turbidity. The researchers achieved impressive sensor accuracy levels: 98.88% for pH, 98.77% for temperature, and 94.92% for TDS readings. These findings validate our hardware selection and approach, whilst suggesting that high levels of accuracy are achievable with similar components.

**Google Sheets Integration in Water Quality Monitoring**

Another project published in 2023 specifically implemented IoT technology based on Google Sheets for water quality monitoring, directly mirroring our approach. Their system demonstrated excellent precision values above 95% across temperature, pH, and TDS sensors. Crucially, they concluded that "Google Sheets can be utilised in IoT technology and can operate on more than one distinct computer device at the same time" and that this approach is "highly effective, flexible, and user-friendly". This strongly validates our choice of Google Sheets as a data platform.

**IoT with Arduino for Catfish Pond Monitoring**

A 2024 study utilised an ESP32 microcontroller with Arduino IDE connected to pH, temperature, and ultrasonic sensors for monitoring catfish pond conditions. Their system also stored data in a real-time database accessible via Blynk. During 27 days of continuous operation, they achieved an impressive average accuracy of 99.45% with high precision (0.01% relative standard deviation). This demonstrates the long-term reliability potential of Arduino-based water monitoring systems like ours.

**Portable Water Quality Analysis with Open-Source Technology**

Particularly relevant to our portable device concept is a 2024 project that developed a proof-of-concept using Arduino sensors and IoT to compute a water quality index (WQI) in real-time. The researchers concluded that "water quality monitoring can be done by using modern-day open-source technology" and that "this new method for tracking water quality could be a game changer in terms of how and where we can track water quality". This directly supports our implementation approach and validates the potential impact of our device.

### Implementation Insights from Research

From this research, several key implementation insights emerge that support our approach:

1. **Sensor Selection Validation**: The consistent use of temperature, turbidity, and pH sensors across multiple successful projects validates our hardware choices.
2. **Data Transmission Methods**: While Blynk is commonly used, our Google Sheets approach is also validated by successful implementations showing high flexibility and reliability.
3. **Arduino as a Reliable Platform**: Multiple studies demonstrate Arduino's effectiveness for water quality monitoring, with systems achieving >95% accuracy.
4. **Real-Time Monitoring Value**: All studies emphasise the importance of real-time monitoring capabilities, confirming the value of our internet-connected approach.
5. **Portability Potential**: Research supports that portable, low-cost monitoring devices can provide valuable water quality insights, aligning with our project goals.

This secondary research firmly establishes that our implementation approach is well-founded in current IoT water quality monitoring practices, whilst offering unique advantages through our specific combination of technologies and focus on portability.

### References for Research part:

Author(s) unknown (2023) *Design of Water Quality Monitoring System for Koi Fish Farming Using NodeMCU ESP32 and Blynk Application Based on Internet of Things*. Publication details not provided in search results. Available at: <https://www.semanticscholar.org/paper/b03ee28e2974ee26de7d2986eede8eb6972a6a7b> [Accessed: 02 May 2025].

Author(s) unknown (2024) *Design and Implementation of IoT-based Water Quality and Leakage Monitoring System for Urban Water Systems Using Machine Learning Algorithms*. Publication details not provided in search results.

Author(s) unknown (2024) *Dynamic Water Quality Monitoring via IoT Sensor Networks and Machine Learning Technique*. Publication details not provided in search results.

Author(s) unknown (2023) *IMPLEMENTATION OF INTERNET OF THINGS (IoT) BASED ON GOOGLE SHEETS FOR WATER QUALITY MONITORING SYSTEM*. Publication details not provided in search results.

Author(s) unknown (2024) *Internet of Things (IoT) and Arduino IDE as a Smart Water Quality Control for Monitoring in Catfish Ponds*. Publication details not provided in search results.

Author(s) unknown (2024) *IoT-Based Water Quality Monitoring System for Koi Fish Quarantine*. Publication details not provided in search results. Available at: <https://www.semanticscholar.org/paper/6f0a052ea01f35ad22da04f2360a22fa5030b8d7> [Accessed: 02 May 2025].

Author(s) unknown (2024) *Modeling Continuous Water Quality in Real-Time with IoT and an Arduino Kit*. Publication details not provided in search results.

Author(s) unknown (2024) *Water quality level estimation using IoT sensors and probabilistic machine learning model*. Publication details not provided in search results.

## Ideas for the potential future application of Machine Learning to enhance the system’s functionality: ChatGPT API, Simple ML with sensor readings data.

There were several considerations on utilisation of ML in the project: AI or ML itself in any way. Up to date, both ways were explored and investigated as for implementation.

### ChatGPT API

Team Representative, as they worked on technical side of the project, had an idea on complementing the data analysis with ChatGPT API, utilised by open-source program, however ultimately it resulted in failure as the script was not compatible with Arduino board.

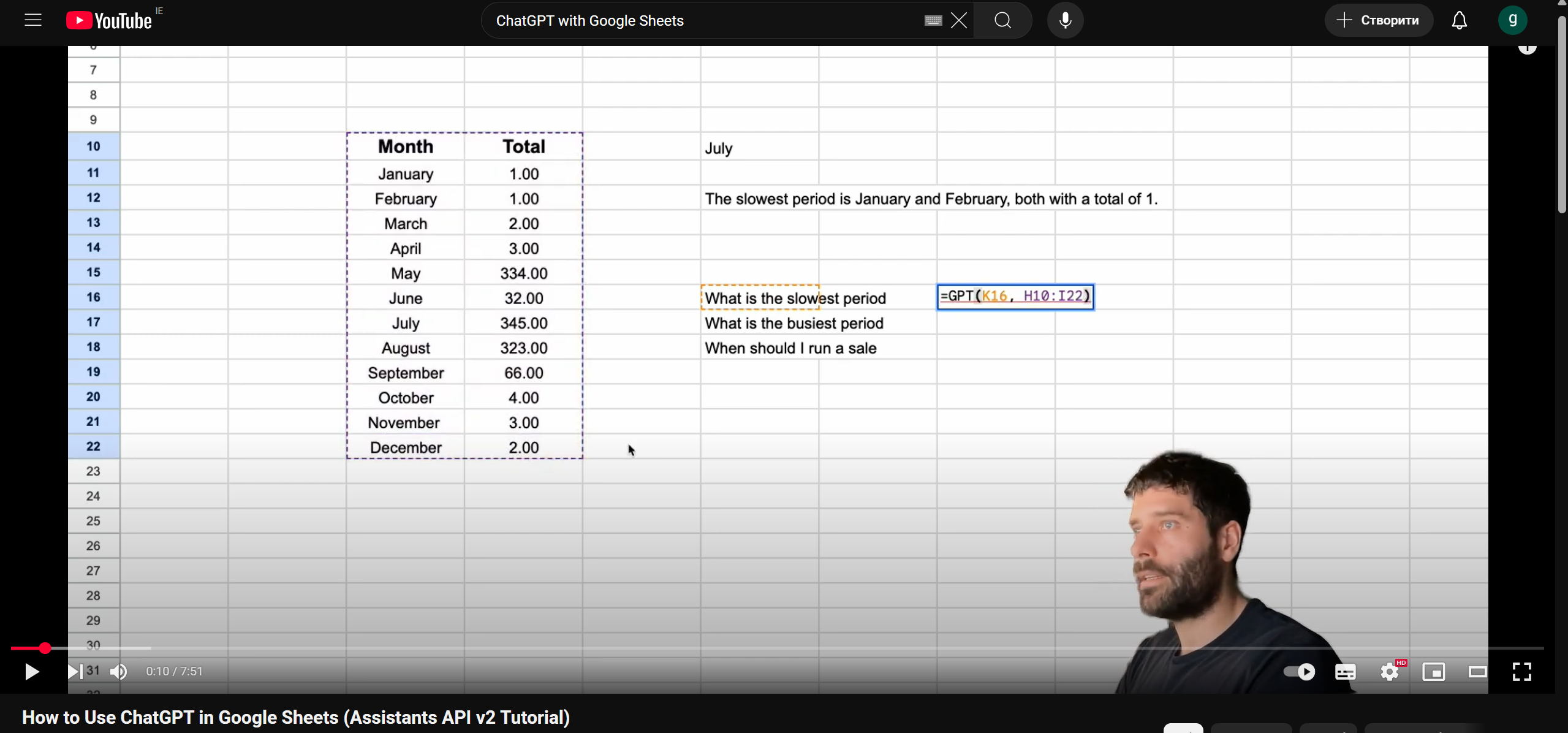


Figure 5: The intended use of ChatGPT in Google Sheets.

Source: [Bart Slodyczka](https://www.youtube.com/watch?v=QJDj0UUcoDM)

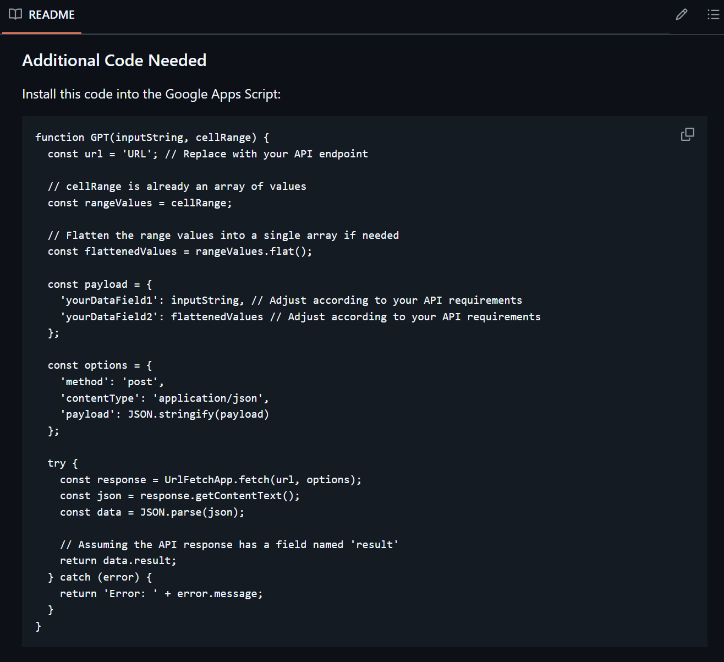


Figure 6: Open-source Google Apps Script code

Source: [Barty-Bart](https://github.com/Barty-Bart/google-sheets-gpt)

### Simple ML

Consequently, the decision has been made to stick with **Simple ML for Sheets** as ultimately the easiest way to access Machine Learning for data analysis for beginners, without even needing prior code knowledge.

A close-up of a logo

AI-generated content may be incorrect.

Figure 7: Simple ML logo.

Source: [Simple ML](https://simplemlforsheets.com/)

After you install the extension in Google Sheets, the functionality is quite extensive for a complicated program like Machine Learning algorithm:

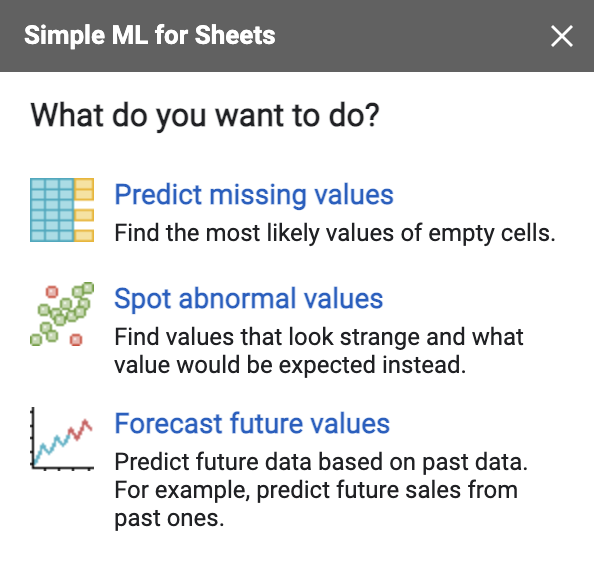


Figure 8: Simple ML functionality

All the above were successfully tested by DreamTeam

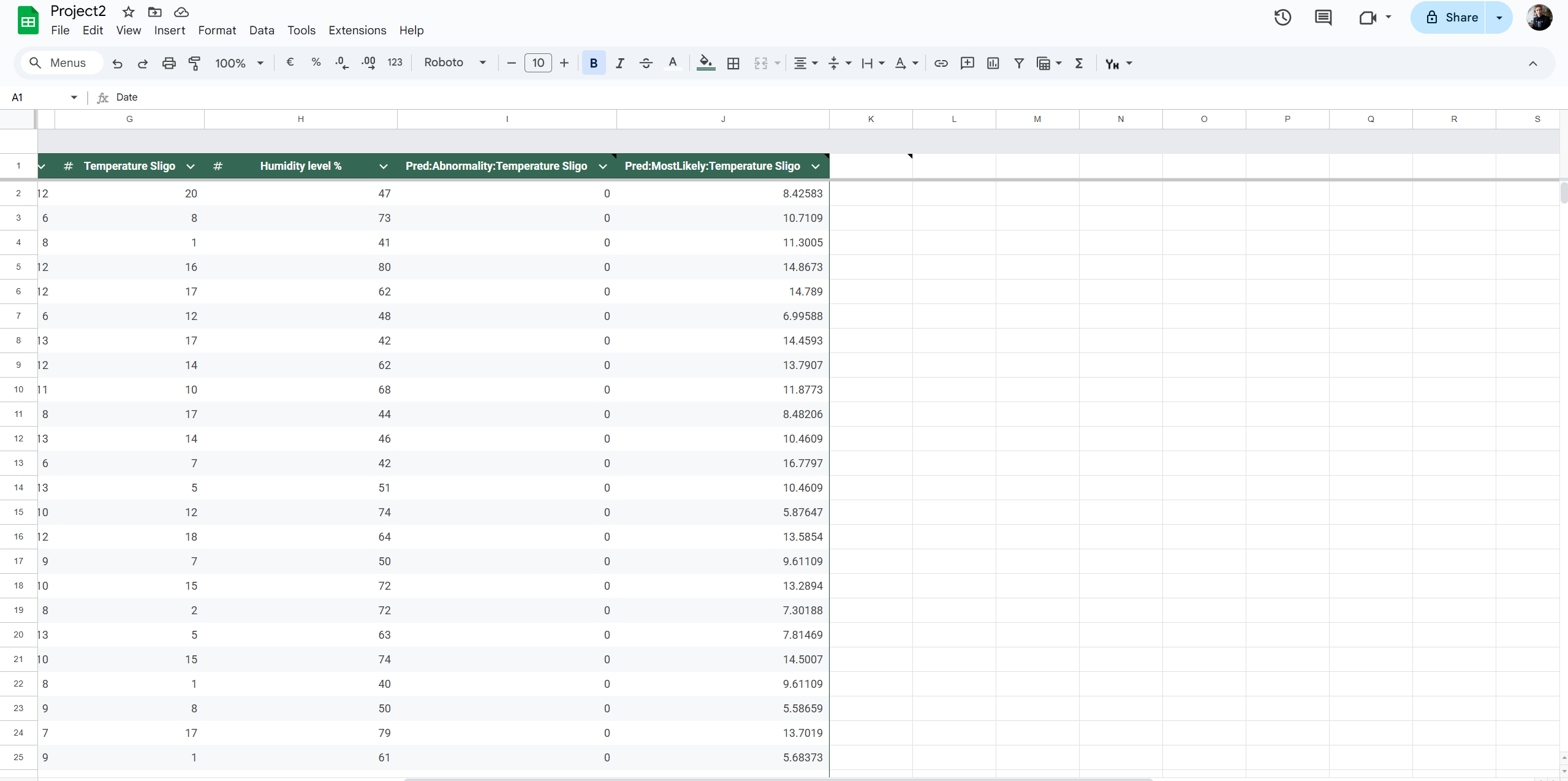


Figure 9: Example usage of “Spot abnormal values”. See the “Abnormality” column, where 0 is normal, and 1 is abnormal (in relation to the value), and the “Most Likely”, which lists the most likely value.

It is important to bear in mind that the whole Simple ML data analysis utilises the data you have in a spreadsheet. Therefore, the more data you have — the better. That is why during the lifetime on the project the data analysation would become more effective the more data the user gathers. Of course, the potential advancement here is advanced ML.

A screenshot of a computer

AI-generated content may be incorrect.

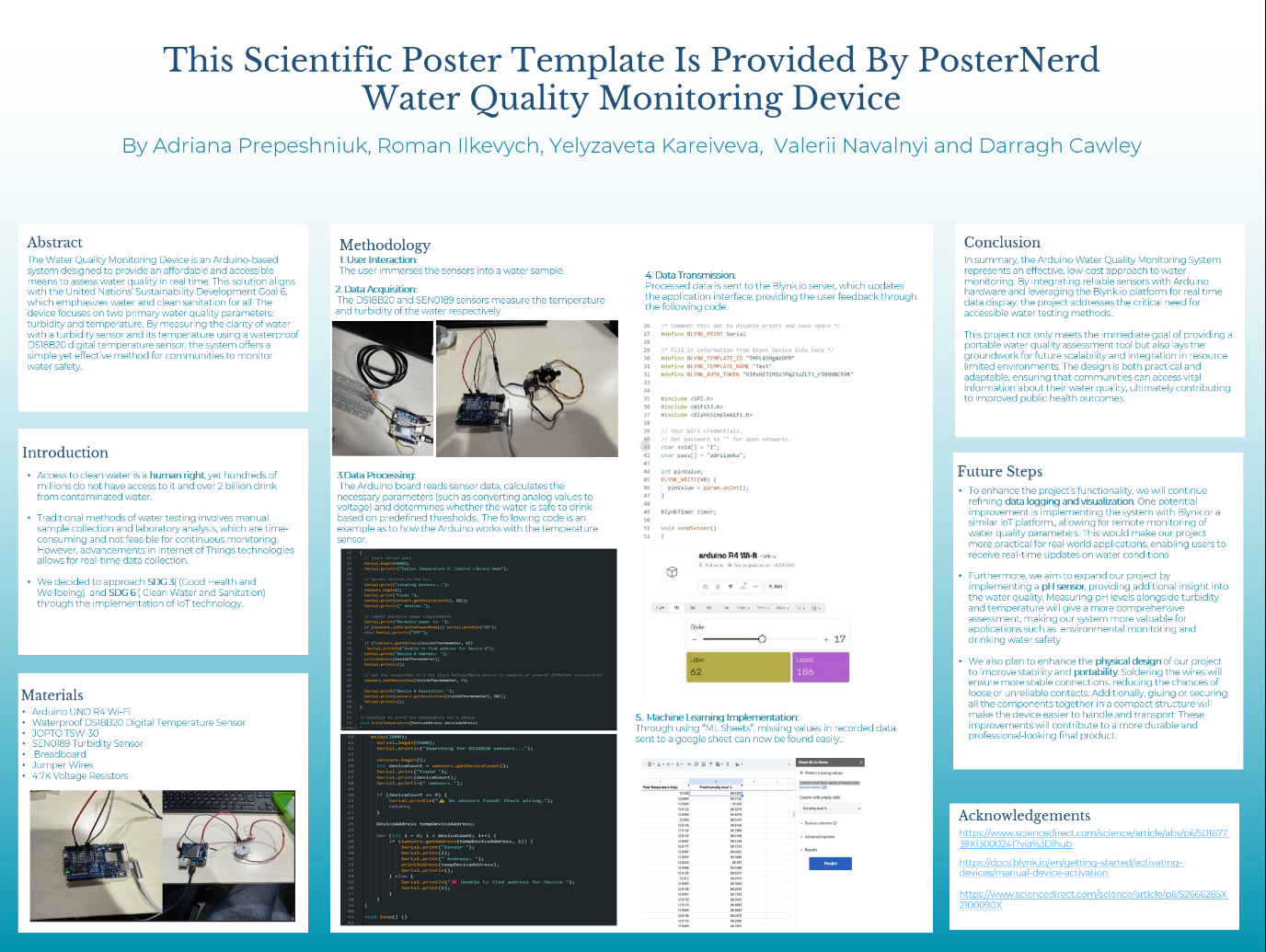
Figure 10: Advanced Simple ML functionalities.

Source: [Google](https://www.youtube.com/watch?v=GYCORMNr_IU)

Next steps for the development of the system:

* pH sensor.
* Improved portability, as had previously been planned, by running on a battery.
* Exploring test or sound alerts.

# Academic-Style Poster



# Critical Analysis

As part of our Lab 7 task, we looked back at our Portable Water Quality Monitoring Device project and tried to critically think about what’s working, what assumptions we made, and what we’ve learned so far. This helped us see our work from different perspectives and consider real improvements.

## System Impact

We originally designed the system to help people monitor if their water was safe to drink, especially in places where clean water isn’t guaranteed. While the idea is promising, we realised it could have more impact than we thought — supporting health, education, and sustainability (UN SDGs 3 & 6). People could use it at home, while travelling, or in schools and farms. But to make that happen, the system would need to be truly portable, affordable, and easy to use.

## Key Assumptions & What We Learned

We assumed the system would be cheap, easy to build, and useful to anyone. But testing showed us that sensors often don’t work as expected, and getting everything to connect —especially between Arduino, APIs, and cloud platforms — is harder than it looks. We also assumed people would easily understand and use the data, but that's not always the case, especially for older or visually impaired users.

One big lesson was to verify everything early — power requirements, sensor compatibility, and code examples. We wasted time on sensors that never worked properly. We also learned to question our own thinking. Just because we would use this system doesn’t mean everyone else would.

We realised that some of our assumptions were based on bias — like assuming that if something makes sense to us, then it will work for everyone. But not everyone has the same access to technology or understanding of how APIs and spreadsheets work. The lack of early testing with real users limited the design choices we made.

## New Perspective

We now think more broadly about who our system is really for. Not everyone will find an API-based setup easy to use. Not everyone has Wi-Fi. And some people might need audio feedback, battery power, or even alerts in different formats. Including these perspectives helped us see what features really matter.

We also started to think about inclusion. For example, the current version doesn’t support people with visual impairments and might be confusing for older users. This highlighted the need to think more about accessibility — not just functionality.

## What We’d Do Differently

This reflection made it clear we need to:

* Simplify or get help with technical parts like sensor setup.
* Use Google Sheets and Apps Script instead of Blynk for more control.
* Focus on portability — power the device with a battery and remove the need for a PC.
* Include backups for sensors or data logging to avoid future obstacles.
* Think more about the end user — who they are, where they are, and how they will use it.

We also now understand that we need a clearer plan from the start, especially when working with technology we’re not yet confident with. Too much time was spent solving problems that could’ve been avoided with better planning or choosing simpler components.

## Final Reflection

Lab 7 was a wake-up call. It showed us that making a useful IoT project is about more than building — it’s about understanding your users, questioning your assumptions, and being flexible when things don’t go to plan. We’re not giving up on the idea, but we’re approaching it smarter now, with a clearer focus on what works, what doesn’t, and what we still need to learn.

Even though the original plan didn’t work out exactly as we hoped, we’ve learned important lessons about inclusive design, testing early, and managing complexity. These lessons will help us in future projects—and maybe even lead us to a better version of this one.

# Ethical Risk Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Risk Zone | Description | Likelihood | Impact | Score | Risk Remediation Action |
| Zone 1 – Truth, Disinformation, Propaganda | Risk of influencing elections | 1 | 1 | 1 | N/A |
| Zone 2 – Addiction & the Dopamine Economy | Risk of harming the user’s life | 1 | 2 | 2 | Inform the user of the accuracy of the data produced historically. |
| Zone 3 – Economic & Asset Inequality | Risk of contributing to societal inequality | 2 | 2 | 4 | Keeping a lower price point, keeping outcomes of the project public. |
| Zone 4 – Machine Ethics & Algorithmic Biases | Risk of discrimination | 1 | 1 | 1 | N/A |
| Zone 5 – Surveillance State | Risk of government overreach | 1 | 1 | 1 | N/A |
| Zone 6 – Data Control & Monetization | Risk of monetizing user personal data | 2 | 1 | 2 | Make the user aware that recorded data is recorded. |
| Zone 7 – Implicit Trust & User Understanding | Risk of the misuse of data | 2 | 1 | 2 | Inform the users of what the recorded data will be used for directly. |
| Zone 8 – Hateful & Criminal Actors | Risk of increasing rates of violent crime | 1 | 3 | 3 | Inform the user of methods of peaceful protest if they consistently have complaints regarding water quality in a sector. |

# Video Demonstration

## Link to the video

[Video Project.mp4](https://atlantictu-my.sharepoint.com/:v:/g/personal/s00273868_atu_ie/ERwE7ofIXW9HjFHY7hdNvT4Bq1ArrgUsvb0llGcEs-hpaQ?e=vkDEbr)