



The present work was submitted to
the German-Mongolian Institute of Resources and Technology

DESIGN AND IMPLEMENTATION OF A SENSOR-BASED WATER QUALITY MONITORING DEVICE

Bachelor's Thesis

By

Batnyam Baatarchuluun

Study program: Mechatronics Engineering

Student ID: B2100091

1st Supervisor/Examiner: Odbileg Norovrinchen

2nd Supervisor/Examiner: Young Suk Kim

Ulaanbaatar/Nalaikh

2025

Statutory Declaration

Batnyam Baatarchuluun

Last Name, First Name

B2100091

Student ID Number

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

DESIGN AND IMPLEMENTATION OF A SENSOR-BASED WATER QUALITY
MONITORING DEVICE

I did not use any sources other than those stated. In case that the work is additionally submitted on a data medium, I declare that the written and the electronic form are completely identical. The work was not submitted in the same or similar form to any examination authority.

2025/11/13

Place, Date

Signature

A handwritten signature in black ink, consisting of a large, stylized 'B' with a diagonal stroke through it, positioned over a horizontal line.

I hereby affirm in lieu of an oath that I provided the submitted bachelor thesis

Table of Contents

Acknowledgement	6
Abstract	7
1. Introduction	8
1.1 Problem Statement	8
1.2 Research Question	10
2. Literature review	10
2.1 Overview of Water Quality Standard	10
2.2 Water Quality Parameters and Standards	12
2.3 Importance of Water Quality Monitoring	13
2.4 Technological Developments	13
2.5 Comparison with Previous Studies	14
3. Methodology	15
3.1 Equipments	15
3.1.1 Oxidation-Reduction Potential (ORP) Sensor	15
3.1.2 Turbidity Sensor	16
3.1.3 pH Sensor	17
3.1.4 TDS Sensor	19
3.2 Sensor Calibration	20
3.2.1 pH Sensor	22
3.2.2 TDS Sensor	22
3.2.3 Turbidity Sensor	23
3.2.4 ORP Sensor	24
4. System Design and Methodology	24
4.1 Control Function	24
4.2 Sensor Selection	28
4.2.1 pH Sensor (Ph4502c)	28
4.2.2 ORP (Oxidation-Reduction Potential) Sensor (ORP board v1.1)	29
4.2.3 TDS (Total Dissolved Solids) Sensor (GravityTDS)	29
4.2.4 Turbidity Sensor (TDS V1)	30
4.3 Component Selection	31
4.4 Required Hardware List	31
4.5 Software Design	35
4.5.1. Embedded Firmware(ESP32)	35
4.5.2 Backend Server Implementation (Node.js + JavaScript + Arduino)	36
4.5.3 Frontend Dashboard (Web Interface)	36
4.5.4 System Integration Overview	37
5. Implementation and Results	39
5.1 Hardware Assembly	39
5.2 Software Deployment	41

5.3 Testing and Troubleshooting	43
6. Result and conclusion	46
7. Discussion	48
8. References	49
9.APPENDICES	50
Appendix A – Schematic Diagram	50
Appendix B – Arduino Code	51
Appendix C – Website Code	52
Appendix D – Website Preview	57

List of figures

Figure 1. Technical scheme	12
Figure 2. Connection diagram between the ORP sensor and the ESP32 microcontroller	15
Figure 3. Connection diagram between the turbidity sensor and the ESP32 microcontroller	16
Figure 4. Connection diagram between the pH sensor and the ESP32 microcontroller	17
Figure 5. Connection diagram between the TDS sensor and the ESP32 microcontroller	18
Figure 6. Definition of ADC parameters and calibration constants in Arduino code	20
Figure 7. Implementation of software-based calibration for sensor voltage readings	20
Figure 8. pH sensor calibration code implementation	21
Figure 9. Implementation of the TDS calibration equation in ESP32 code	22
Figure 10. Implementation of the turbidity calibration equation in ESP32 code	22
Figure 11. Implementation of the ORP calibration equation in ESP32 code	23
Figure 12. Flowchart of the data acquisition and processing sequence	24
Figure 13. Schematic diagram of water quality monitoring system circuit	32
Figure 14. Circuit schematic showing connections between ESP32 and sensors	33
Figure 15. Web-based dashboard interface for real-time water quality monitoring	26
Figure 16. System end-to-end data-flow architecture	37
Figure 17. Assembled prototype showing OLED display output and internal wiring	40
Figure 18. System architecture diagram showing data flow between hardware and UI	41
Figure 19. pH comparison of river, bottled, and tap water samples	42
Figure 20. Total dissolved solids comparison of river, bottled, and tap water samples	44
Figure 21. Oxidation-reduction potential comparison of water samples	45

List of table

Table 1. Sensor types	11
Table 2. List of Hardware Components Used	31

List of equation

Equation 1. Linear calibration model for sensor voltage adjustment	23
Equation 2. Formula for calculating pH from $[H^+]$	27
Equation 3. Electrode potential dependence on pH from the Nernst equation	27
Equation 4. Simplified Nernst equation for pH calculation	27
Equation 5. Nernst equation describing redox equilibrium	28
Equation 6. Formula converting EC to TDS	29
Equation 7. Voltage-based turbidity calculation formula	29

Acknowledgement

First and foremost, I would like to extend my heartfelt appreciation to my primary supervisor, Professor Odbileg Norovryenchin, for his unwavering guidance, encouragement, and invaluable technical expertise throughout the entire development of this thesis. His mentorship has been instrumental in shaping both the direction and the overall quality of this work.

I am also sincerely grateful to my second supervisor, Mr. Young Suk Kim, for his insightful feedback, continuous support, and readiness to assist whenever needed. His contributions significantly strengthened various components of the research, including the system design and implementation.

Special acknowledgement is due to my teacher, Mrs. Bayardulam J., whose support and expertise during the calibration phase were essential. Her assistance greatly enhanced the precision of the setup and contributed to the successful operation of the system.

Finally, I would like to express my deep gratitude to the German-Mongolian Institute for Resources and Technology (GMIT) for fostering a resourceful, inspiring, and supportive academic environment. Conducting this thesis within such an institution has been a meaningful milestone in my educational and professional journey. To everyone who has supported and encouraged me directly or indirectly thank you.

Abstract

This thesis presents the design, integration, and development of a sensor-based water quality monitoring device utilizing ESP32 microcontroller, aimed at enabling real-time, accessible, and cost-effective water quality assessment. The system integrates four key sensors such as pH, turbidity, oxidation-reduction potential (ORP), and total dissolved solids (TDS) and all managed through the ESP32 microcontroller. These sensors continuously measure fundamental water quality parameters, which are displayed on a LCD screen and transmitted via a Wi-Fi module for remote monitoring through computers or other smart devices.

The system's primary objective is to provide an efficient, portable, and affordable tool that allows users to easily evaluate water quality without the need for laboratory equipment. In the future, the system can be extended with additional sensors such as temperature, dissolved oxygen, or conductivity sensors to further enhance measurement precision and environmental adaptability.

Overall, this research successfully demonstrates a compact, modular, and practical prototype that contributes to improving water quality monitoring practices and supports sustainable water management solutions.

1. Introduction

1.1 Problem Statement

Water is an essential resource that sustains life, economic development, and ecosystem. However, the rapid urbanization, industrialization, and environmental degradation in many regions are exerting increasing pressure on water quality. This is especially true in urban centers like Ulaanbaatar, the capital of Mongolia, where water sources face a unique set of challenges. Ulaanbaatar is located in a semi-arid region characterized by extreme temperature variations, pollution from industrial and domestic sources, and seasonal fluctuations in water availability (Batjargal et al., 2020). These factors not only affect the quantity of available water but also degrade its quality, posing a significant threat to public health and the environment.

In Ulaanbaatar, rapid urbanization, industrial activities, and air pollution from coal-burning contribute significantly to the degradation of water quality (Bat-Erdene & Tsogt, 2019). The Tuul River, a primary water source for the city, has been increasingly contaminated by industrial effluents, sewage discharge, and runoff from surrounding mining activities (Baatar & Tumenjargal, 2021). Meanwhile, the city's water treatment infrastructure is often inadequate to address these challenges, leaving many areas without access to clean water (Tserenbat et al., 2022). Given these circumstances, effective and real-time monitoring of water quality is essential to address these challenges and inform public health decisions.

Access to clean and safe water is critical for public health and environmental sustainability, yet traditional water quality monitoring methods are often expensive, time-consuming, and require specialized equipment and expertise. These limitations make real-time monitoring challenging, particularly in remote or resource-constrained areas. There is a growing need for an affordable, portable, and user-friendly solution that can provide accurate, real-time water quality data to support timely decision-making and intervention.

The quality of water is a complex interplay of physical, chemical, and biological parameters, and turbidity, pH level, Oxidation-Reduction Potential (ORP), and Total Dissolved Solids (TDS) are widely recognized as fundamental indicators for assessing water quality (Jin et al., 2019). Turbidity refers to the cloudiness of water caused by suspended particles and is a key indicator of contamination by microorganisms, heavy metals, or organic pollutants. The pH level provides insights into the acidity or alkalinity of water, which influences chemical reactions and biological processes within aquatic ecosystems. ORP measures the oxidative capacity of water, providing an indication of its ability to neutralize pollutants, while TDS quantifies the concentration of

dissolved substances, such as salts, minerals, and metals, which can impact water's taste, safety, and suitability for irrigation.

This project presents the design, development, and testing of a multifunctional device that integrates sensors for turbidity, pH, ORP, and TDS measurement. The proposed device aims to address the limitations of current monitoring technologies by providing a portable, cost-effective, and user-friendly tool for water quality monitoring in both urban and rural settings. By utilizing advanced sensor technologies and a simple user interface, this device seeks to make water quality monitoring more accessible, empowering local communities, water management authorities, and environmental organizations to take proactive measures in addressing water contamination.

1.2 Research Question

The main objective of this research is to design and implement a low-cost, real-time, and modular sensor-based water quality monitoring device using the ESP32 microcontroller. This system aims to provide accessible and reliable water quality data for users in both urban and rural regions of Mongolia.

The specific objectives of this research are as follows:

- To integrate multiple water quality sensors (pH, TDS, turbidity, and ORP) into a single ESP32-based platform.
- To design and implement a data acquisition and processing system capable of real-time monitoring.
- To develop a user-friendly interface for displaying sensor readings and transmitting data via Wi-Fi.
- To test and evaluate the performance of the prototype under different water conditions.
- To propose future enhancements such as temperature compensation and mobile application integration.

By addressing these questions, this research aims to develop a practical, portable, and modular prototype that supports improved water quality monitoring and provides a foundation for future development of water quality solutions in Mongolia.

2. Literature review

2.1 Overview of Water Quality Standard

Globally, the World Health Organization (WHO) provides recognized guidelines for drinking-water quality, which serve as a foundation for national standards and public health policies. These guidelines emphasize a risk-based approach through Water Safety Plans, ensuring safe water from catchment to consumer. WHO establishes health-based targets and guideline values for key chemical, microbial, and radiological parameters, derived from toxicological and epidemiological studies. The latest edition, published in 2022, includes updated values for contaminants such as arsenic, lead, nitrate, and cyanobacterial toxins, alongside enhanced guidance for small water supplies and climate-related emergencies. These standards aim to protect human health by setting permissible limits for parameters like pH, turbidity, and dissolved solids, while promoting proactive monitoring and independent surveillance. Despite widespread adoption—over 98% of countries reference WHO guidelines—challenges remain in ensuring compliance, particularly in rural and resource-limited areas. By aligning with Sustainable Development Goal 6.1, WHO's framework underscores the importance of universal access to safe drinking water and the need for affordable, real-time monitoring solutions (WHO, 2022).

Global standards for drinking water quality, as outlined by the World Health Organization (WHO) and widely adopted by national agencies, set permissible limits for key parameters such as pH, total dissolved solids (TDS), turbidity, and oxidation-reduction potential (ORP) to ensure safety and palatability. WHO recommends a pH range of 6.5 to 8.5, as extreme acidity or alkalinity can cause corrosion, scaling, and health issues. For TDS, the guideline suggests a palatability threshold of 600 mg/L, with an upper acceptable limit near 1000 mg/L, since excessive dissolved solids may affect taste and indicate contamination. Turbidity, which reflects suspended particles and microbial risk, should remain below 5 NTU for aesthetic quality and ideally under 1 NTU to ensure effective disinfection. Although WHO does not prescribe a strict ORP value, safe drinking water typically exhibits an ORP between +650 and +750 mV, signaling adequate oxidation for pathogen control. These standards collectively safeguard public health, maintain water treatment efficiency, and support global efforts toward universal access to safe drinking water.

In Mongolia, water quality management is governed by several national standards that define the permissible limits and testing methodologies for various types of water sources. Among

these, MNS 4586:1998 and MNS 0900:2018 are two of the most significant regulatory documents used to evaluate and maintain water quality within the country.

The standard MNS 4586:1998, titled *“Surface Water Quality Standard of Mongolia,”* establishes the requirements for maintaining the chemical, physical, and biological quality of surface water bodies such as rivers, lakes, and reservoirs. Its main objective is to ensure a healthy and sustainable environment by preserving the ecological balance of aquatic ecosystems. This standard classifies surface waters according to their intended use — for instance, for human consumption, agricultural irrigation, fisheries, or industrial purposes — and specifies the maximum allowable concentrations of various contaminants for each category. By setting these limits, MNS 4586:1998 serves as a foundation for environmental monitoring, pollution control, and sustainable water resource management across the country.

On the other hand, MNS 0900:2018, titled *“Drinking Water Standard of Ulaanbaatar,”* defines the quality requirements for potable water supplied to households and public facilities in the capital city. This standard is comprehensive, covering 105 specific quality indicators, which are divided into four main categories: 55 organic compounds, 20 chemical parameters, 23 elemental indicators, and 7 microbiological indicators. These parameters collectively ensure that drinking water meets safety, health, and environmental standards, protecting the population from waterborne diseases and toxic substances. Laboratory testing is performed to verify compliance, and the results are used for quality assurance and regulatory certification.

2.2 Water Quality Parameters and Standards

The four selected sensors—pH, TDS, Turbidity, and ORP—collectively provide a comprehensive approach to assessing the fundamental physicochemical characteristics of water. These parameters are critical because they directly influence health safety, taste, and treatment efficiency. Among them, pH and turbidity are often prioritized for immediate safety and treatment control, while TDS and ORP provide broader indicators of chemical balance and disinfection. The pH sensor ensures the evaluation of acidity and alkalinity. The TDS sensor quantifies the total ionic content, representing the degree of mineralization and overall purity of the sample. The turbidity sensor measures optical clarity, which reflects filtration efficiency and the presence of suspended solids or particulate matter. Meanwhile, the ORP sensor indicates the oxidation–reduction potential, providing insight into the redox state and disinfection efficiency of water.

Although they do not directly measure specific contaminants such as heavy metals or microbial components, they effectively represent the core physical and chemical parameters essential for evaluating overall water quality and treatment performance.

Table 1. Sensor types

Sensor type	Measured Parameter	Working Principle	Standard Limit (MNS 0900:2018)	Water Quality Significance
pH Sensor	Acidity / Alkalinity (Hydrogen ion concentration)	Measures potential difference between a glass electrode and Ag/AgCl reference electrode, based on the Nernst equation	6.5 – 8.5 pH units	Indicates chemical stability, corrosion or scaling potential, and disinfection efficiency
TDS Sensor	Total Dissolved Solids (inorganic and organic ions)	Measures electrical conductivity and converts it to TDS (mg/L or ppm)	≤ 1000 mg/L	Reflects mineral content, ionic strength, and overall purity of water
Turbidity Sensor	Clarity / Suspended particles	Uses a light beam and photodiode to detect light scattering caused by suspended solids	≤ 5 NTU	Indicates filtration effectiveness, presence of solids or microbial contamination
ORP Sensor	Oxidation–Reduction Potential (redox state)	Measures voltage difference between platinum electrode and Ag/AgCl reference electrode	No fixed limit (indicative parameter)	Assesses oxidation capacity, disinfection potential, and chemical reactivity

2.3 Importance of Water Quality Monitoring

Water quality monitoring is essential for safeguarding public health, protecting ecosystems, and ensuring sustainable water resource management. Contaminated water can transmit harmful pathogens, heavy metals, and chemical pollutants, leading to severe health risks such as gastrointestinal diseases, neurological disorders, and long-term chronic conditions. Monitoring provides critical data for detecting pollution sources, assessing compliance with national and international standards, and guiding treatment processes to maintain safe drinking water. It also supports early warning systems for contamination events, enabling timely interventions to prevent outbreaks and environmental degradation. In addition, continuous monitoring is vital for

industrial and agricultural sectors, where water quality directly influences production efficiency and food safety. With growing pressures from urbanization, climate change, and industrialization, real-time and cost-effective monitoring technologies—such as sensor-based systems—are increasingly important for achieving global water security and meeting Sustainable Development Goals.

2.4 Technological Developments

Technological advancements have also facilitated wireless communication and Internet of Things (IoT) integration. Microcontrollers connected to Wi-Fi modules or cloud platforms provide real-time data visualization and remote access, allowing users, water authorities, and environmental organizations to monitor water quality from a distance. LCD displays, simple interfaces, and modular designs improve usability and accessibility, bridging the gap between laboratory testing and field-level monitoring.

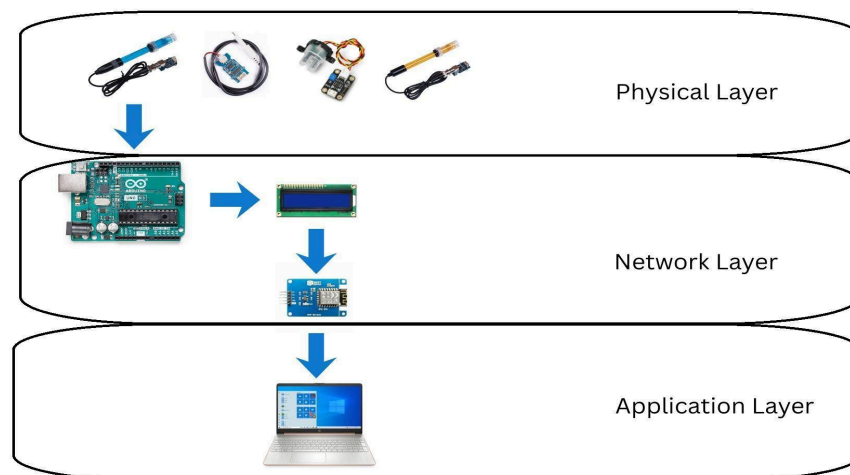


figure 1. Technical scheme

Despite these developments, several gaps remain. Many existing systems are too expensive or technically complex for household or small-scale applications. Few devices are adapted to the specific environmental, economic, and social contexts of Mongolia, where water quality challenges are exacerbated by urbanization, industrial activity, and seasonal water fluctuations. Current monitoring approaches are limited by infrequent testing and lack of real-time data.

In conclusion, sensor-based water quality monitoring provides a practical, cost-effective, and real-time alternative to traditional laboratory methods. This thesis addresses the gaps by designing a multifunctional device integrating pH, turbidity, ORP, and TDS sensors with an

ESP32 microcontroller, featuring LCD display and Wi-Fi connectivity. The proposed system aims to provide a locally relevant solution for water quality monitoring while supporting further technological development and environmental management initiatives.

2.5 Comparison with Previous Studies

Several previous studies have proposed sensor-based water quality monitoring systems using microcontrollers such as Arduino Uno and Raspberry Pi. Jin et al. (2019) developed a low-cost monitoring system that achieved real-time measurement but lacked long-distance communication capabilities. Bat-Erdene and Tsogt (2019) reported on locally developed systems in Mongolia, which were limited by high costs and maintenance requirements.

Compared to these systems, the proposed ESP32-based device offers built-in Wi-Fi connectivity, lower power consumption, and higher integration efficiency, allowing real-time data transmission without additional modules. This improvement makes the system more suitable for Mongolia's diverse environmental conditions, where affordability and portability are crucial.

3. Methodology

3.1 Water quality sensors

3.1.1 Oxidation-Reduction Potential (ORP) Sensor

The Oxidation-Reduction Potential (ORP) sensor is an electrochemical device designed to measure the redox potential of a liquid sample, which indicates its ability to either gain or lose electrons. This parameter serves as an important indicator of the chemical reactivity, water quality, and oxidative capacity of the solution. In environmental monitoring and water treatment applications, ORP measurement provides valuable insight into the balance between oxidizing and reducing substances present in the water.

The ORP sensor operates based on the potential difference generated between two electrodes — a measuring electrode, typically made of platinum or gold, and a reference electrode, usually composed of silver/silver chloride (Ag/AgCl). When both electrodes are immersed in the sample solution, a voltage (measured in millivolts, mV) is produced as a result of electron exchange between the redox-active species in the water and the electrode surface. This voltage represents the oxidation-reduction potential of the sample.

A high positive ORP value indicates that the solution has a strong oxidizing potential, meaning it contains substances capable of accepting electrons — commonly associated with effective disinfection and low organic contamination. Conversely, a negative ORP value reflects a reducing environment where electron-donating substances, such as organic compounds or pollutants, dominate.

In practical terms, ORP values above +300 mV usually represent water with strong disinfectant properties (such as chlorinated drinking water), while values between 0 and +300 mV are typical for natural water bodies. Values below 0 mV suggest anaerobic or polluted conditions.

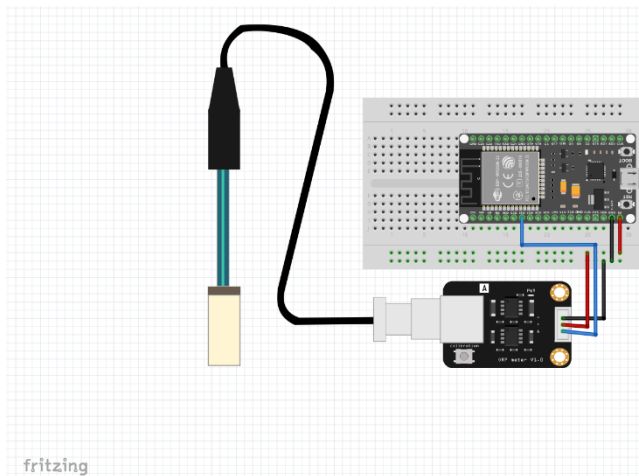


figure 2.Connection diagram between the ORP sensor and the ESP32 microcontroller.

Due to its sensitivity and reliability, the ORP sensor is extensively used in drinking water treatment, wastewater management, aquaculture systems, and industrial process control to monitor redox balance and ensure proper water quality standards. When integrated with digital system ESP32 microcontrollers, ORP sensors enable real-time monitoring and automated data collection, contributing to smarter and more sustainable water management practices.

3.1.2 Turbidity Sensor

The Turbidity sensor is an optical device used to measure the clarity or cloudiness of water, which is directly related to the amount of suspended particles such as silt, clay, organic matter, and microorganisms present in the sample. Turbidity is a key indicator of water quality, as it reflects the degree to which light passing through water is scattered or absorbed by particles. The principle of operation of a turbidity sensor is based on light scattering. The sensor typically consists of a photodiode or phototransistor detector that measures the intensity of light after it

has interacted with the particles in the water.

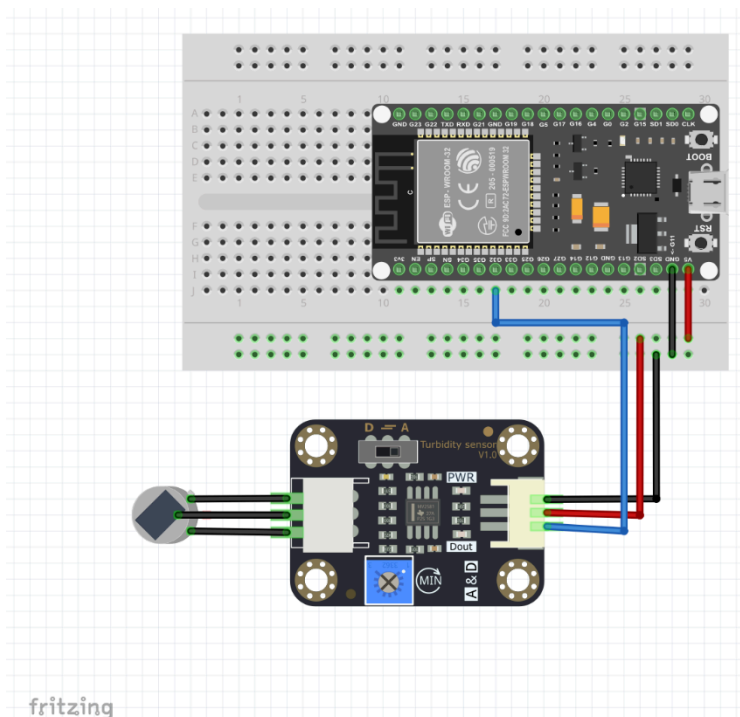


Figure 3. Connection diagram between the turbidity sensor and the ESP32 microcontroller.

When the water is clear, most of the light passes straight through the sample and only a small amount is scattered, resulting in a low turbidity value. As the concentration of suspended solids increases, more light is scattered and less reaches the detector, leading to a higher turbidity reading. The measurement is usually expressed in Nephelometric Turbidity Units (NTU), in accordance with ISO 7027 or EPA Method 180.1 standards.

Turbidity plays a critical role in assessing the aesthetic quality and safety of drinking water, as high turbidity can reduce the effectiveness of disinfection and indicate the presence of harmful microorganisms. In environmental contexts, it also serves as an indicator of soil erosion, industrial discharge, and sediment pollution in natural water bodies.

3.1.3 pH Sensor

The pH sensor is an electrochemical instrument used to determine the acidity or alkalinity of an aqueous solution by measuring the concentration of hydrogen ions. The pH value, expressed on

a logarithmic scale ranging from 0 to 14, represents the chemical reactivity and balance of the solution, where lower values indicate acidity, higher values indicate alkalinity, and a value of 7 denotes neutrality.

The working principle of a pH sensor is based on the potential difference generated between a glass measuring electrode and a reference electrode, which is usually composed of silver/silver chloride (Ag/AgCl). When both electrodes are immersed in the water sample, a voltage difference develops as a result of the interaction between hydrogen ions and the glass membrane of the measuring electrode. This potential difference follows the Nernst equation, establishing a logarithmic relationship between the measured voltage and the hydrogen ion concentration. The resulting electrical potential is then converted into a pH value by the measurement circuit or a connected microcontroller.

pH measurement is a fundamental parameter in evaluating water quality, chemical stability, and biological suitability. Acidic conditions ($\text{pH} < 7$) may indicate industrial contamination, acid rain, or chemical discharge, while alkaline conditions ($\text{pH} > 7$) can result from mineral imbalances or excess chemical additives. Neutral pH values typically reflect clean and balanced water suitable for drinking and biological processes. Maintaining an appropriate pH level is essential for ensuring the effectiveness of water treatment processes and the health of aquatic environments. The pH sensor is widely applied in water treatment plants, environmental monitoring, aquaculture, agriculture, industrial processing, and laboratory analysis. In modern applications, pH sensors are often integrated with microcontroller ESP32 to enable real-time data acquisition, calibration, and digital monitoring. This integration enhances the efficiency, precision, and automation of water quality management systems, making pH measurement a critical component in both environmental and industrial research.

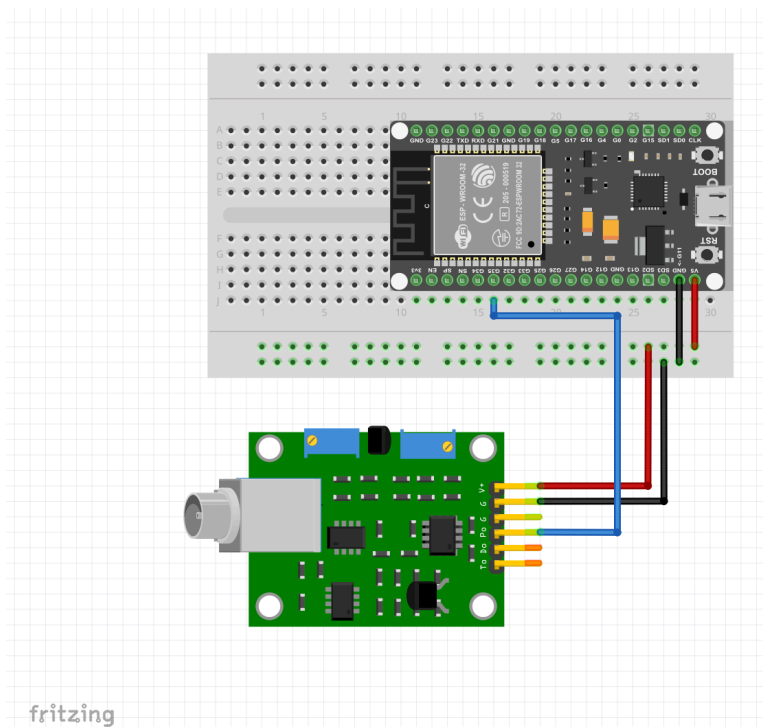


figure 4.Connection diagram between the pH sensor and the ESP32 microcontroller.

3.1.4 TDS Sensor

The Total Dissolved Solids (TDS) sensor is an electronic device used to measure the concentration of dissolved substances in water, including minerals, salts, and other inorganic or organic compounds. TDS is an essential indicator of water purity and overall chemical composition, as it represents the total amount of mobile charged ions, typically measured in milligrams per liter (mg/L) or parts per million (ppm). High TDS values generally indicate the presence of impurities, while low values suggest clean and low-mineral water.

The TDS sensor operates based on the principle of electrical conductivity. Dissolved ions in water enable the flow of electric current, and the degree of conductivity is directly proportional to the concentration of dissolved solids. The sensor applies a small alternating current through the water sample and measures the resulting voltage to determine its conductivity. This conductivity value is then converted into a TDS reading using an empirical conversion factor that accounts for temperature and the ionic characteristics of the water. Since temperature significantly affects conductivity, many TDS sensors include automatic temperature compensation to ensure measurement accuracy.

TDS measurement provides crucial information for evaluating water quality in both environmental and industrial contexts. In drinking water analysis, excessive TDS can affect

taste, odor, and health, while in industrial applications, it can influence corrosion rates, scaling, and process efficiency. A TDS value below 300 ppm is generally considered good for drinking purposes, while higher values may indicate contamination or the need for filtration and treatment.

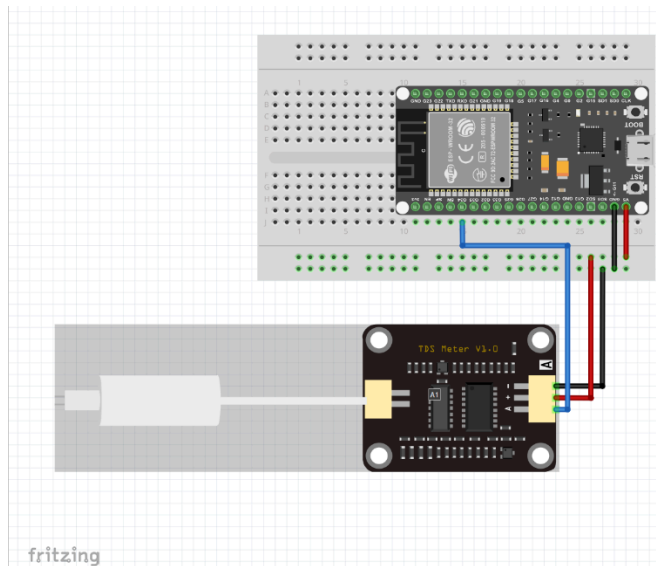


Figure 5. Connection diagram between the TDS sensor and the ESP32 microcontroller.

Water quality is a crucial factor for human health, environmental sustainability, and economic development. Monitoring involves measuring physical, chemical, and biological parameters such as pH, turbidity, oxidation-reduction potential (ORP), total dissolved solids (TDS), temperature, and conductivity. Traditional laboratory-based methods are accurate but often costly, time-consuming, and unsuitable for real-time monitoring, particularly in remote or underserved areas. These limitations have led to growing interest in sensor-based, portable, and cost-effective water quality monitoring systems.

Sensor-based systems commonly use pH, turbidity, ORP, and TDS sensors to measure water acidity, suspended solids, oxidative potential, and dissolved substances. Integration with microcontroller ESP32 enables compact, automated, and real-time monitoring. Studies by Jin et al. (2019) and Bat-Erdene & Tsogt (2019) demonstrate that such systems can provide accurate water quality data with minimal human intervention. Portable and multifunctional devices allow simultaneous measurement of multiple parameters, making them suitable for both urban and rural water sources.

3.2 Sensor Calibration

Sensor calibration was performed using several different approaches depending on the characteristics and requirements of each sensor. Certain devices, such as the TDS sensor, did not require extensive calibration because they are pre-calibrated or rely on fixed conversion factors supplied by the manufacturer. However, other sensors—particularly those measuring pH, turbidity, and ORP—required more complex calibration procedures.

Calibration of the ORP sensor was especially challenging due to the limited availability of standard ORP calibration solutions. Despite this constraint, calibration was carried out using known reference values and controlled test conditions to ensure acceptable accuracy. For pH and turbidity, standard buffer solutions and empirical voltage–value mapping were used.

During implementation, several calibration libraries available in the Arduino IDE were tested. Many of these libraries were outdated, intended for older Arduino devices, or incompatible with the ESP32 platform. As a result, portions of these libraries had to be manually modified, rewritten, or fully integrated directly into the main program without external library dependencies. Custom calibration routines were also incorporated so that manual calibration could be performed when necessary.

```
13
14  const int SAMPLE_COUNT = 10;    // Average N readings
15  const float VREF = 5;           // Reference voltage (ESP32 = 5V)
16  const int ADC_RES = 4095;       // 12-bit ADC = 0-4095
17
18
19  // Calibration factors (you can adjust later)
20  float scale_pH = 1.0, offset_pH = 0.0;
21  float scale_ORP = 1.0, offset_ORP = 0.0;
22  float scale_TDS = 1.0, offset_TDS = 0.0;
23  float scale_TUR = 1.0, offset_TUR = 0.0;
24
```

figure 6. Definition of ADC parameters and calibration constants in Arduino code

```

80 // Apply calibration (optional)
81 float cal_pH = v_pH * scale_pH + offset_pH;
82 float cal_ORP = v_ORP * scale_ORP + offset_ORP;
83 float cal_TDS = v_TDS * scale_TDS + offset_TDS;
84 float cal_TUR = v_TUR * scale_TUR + offset_TUR;
85

```

figure 7. Implementation of software-based calibration for sensor voltage readings

3.2.1 pH Sensor

The pH sensor was calibrated using a two-point calibration method based on standard buffer solutions of pH 4.30 and pH 7.25. During testing, the sensor produced a voltage output of 3.30 V when immersed in the pH 4.30 solution, and 2.518 V when immersed in the pH 7.25 solution. These two reference points were used to derive a linear calibration equation that maps the sensor's analog voltage to an estimated pH value. The calculated slope and offset were then incorporated directly into the program code, enabling the ESP32 to compute corrected pH readings in real time. This approach ensured more accurate and stable measurements compared to relying solely on default or library-based calibration parameters.

```

45 const float CAL_pH1 = 4.30f; const float CAL_V1 = 3.300f;
46 const float CAL_pH2 = 7.25f; const float CAL_V2 = 2.518f;
47 float PH_SLOPE = -2.744f;
48 float PH_INTERCEPT = 14.679f;
49

```

figure 8. Ph sensor calibration code Implementation

3.2.2 TDS Sensor

Although the TDS sensor did not require a formal calibration procedure, adjustments were made to improve its compatibility and accuracy within the system. The DFRobot TDS sensor library for Arduino (figure 8.) was reviewed and adapted to match the electrical characteristics of the sensor used in this project. Instead of relying on the default library configuration, the original calculation and filtering routines were modified and integrated directly into the code. This allowed the TDS conversion formula to be tailored to the specific voltage range and sampling behavior of the connected hardware, resulting in more stable and reliable TDS readings. By embedding the calibration and computation logic into the main program, the implementation became both simpler and more consistent across different devices.

```

1  #define sensorPin A0 // Analog pin connected to the TDS sensor data output
2
3  int sensorValue = 0;
4  float voltage = 0;
5  float tdsValue = 0;
6  float temperature = 25.0; // Assume a constant temperature for simplicity, or use a temperature sensor
7
8  void setup() {
9      Serial.begin(9600);
10 }
11
12 void loop() {
13     sensorValue = analogRead(sensorPin);
14     voltage = sensorValue * (5.0 / 1024.0); // Convert analog reading to voltage (assuming 5V Arduino)
15
16     // Apply temperature compensation if a temperature sensor is used
17     // float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0);
18     // float compensatedVoltage = voltage / compensationCoefficient;
19
20     // Conversion formula (adjust based on your sensor's datasheet or calibration)
21     // This is a common example, but might need modification
22     tdsValue = (133.42 * voltage * voltage * voltage - 255.86 * voltage * voltage + 857.39 * voltage) * 0.5;
23
24     Serial.print(" Loading...");
25     Serial.print(tdsValue);
26     Serial.println(" ppm");
27
28     delay(1000);
29 }

```

figure 9. Implementation of the TDS calibration equation in esp32 code

3.2.3 Turbidity Sensor

The turbidity sensor was calibrated using a one-point calibration method, in which distilled water was used as the 0 NTU reference. When immersed in distilled water, the sensor produced an output of approximately 2.0 V, and this value was defined as the baseline corresponding to 0 NTU. All subsequent turbidity measurements were interpreted relative to this baseline, enabling the detection of increased turbidity through deviations in measured voltage.

Arduino-based turbidity libraries were also utilized to assist with voltage reading, filtering, and basic NTU conversion. However, due to variations in sensor behavior, portions of the library logic were adapted and incorporated directly into the system's code to ensure accurate operation with the specific hardware used in this project. The implementation and calibration workflow are shown in (figure 9.), where the corresponding code snippet and hardware configuration are provided.

```

50 // ----- TURBIDITY (simple linear) -----
51 const float V_TURB_CLEAR = 2.00f;
52 const float TURB_V_DIRTY = 1.00f;
53 const float TURB_NTU_DIRTY = 1000.0f;
54 float TURB_SLOPE;
55 float TURB_INTERCEPT;
56

```

Figure 10. Implementation of the Turbidity calibration equation in esp32 code

3.2.4 ORP Sensor

The ORP sensor was calibrated by assigning the sensor's electrical midpoint (ORP center) to 2.5 V. Distilled water was used as the reference solution and was assumed to have an approximate oxidation–reduction potential of 250 mV. Using these reference points, the sensor output was adjusted so that a measured voltage of 2.5 V corresponds to 250 mV, and all subsequent ORP values were derived relative to this calibrated center. The conversion procedure and the specific calculation used for this mapping are illustrated in (figure 10.) and expressed formally in Equation 1, which was applied in the code to compute calibrated ORP readings throughout the system.

```
56 |  
57 | // ----- ORP (DFRobot board) -----  
58 | const float ORP_CENTER_V = 2.5f;  
59 | const float ORP_SCALE_MV_PER_V = 1000.0f;  
60 | float ORP_OFFSET_MV = -1418.0f; // Calibrated for 1.332V => 250 mV in distilled water  
61 |
```

figure 11. Implementation of the ORP calibration equation in esp32 code

$$v_{\text{Calibrated}} = v_{\text{measured}} \times \text{scale} + \text{scale} + \text{offset}$$

Equation 1. Linear calibration model for sensor voltage adjustment.

4. System Design and Methodology

4.1 Control Function

The control function of the water quality monitoring system plays a vital role in ensuring the continuous and accurate measurement of various water parameters. It acts as the central operational process that manages data acquisition, signal conversion, data processing, and wireless communication between the sensors, the ESP32 microcontroller, and the user interface. The control logic implemented in the ESP32 determines how the system responds to the inputs from the sensors and delivers output data to the web dashboard for monitoring and analysis.

The system employs four major sensors: pH, TDS, Turbidity, and ORP sensors. Each of these sensors measures a specific characteristic of water quality. The pH sensor measures the acidity or alkalinity of the water, while the TDS sensor determines the total dissolved solids (measured in ppm). The Turbidity sensor measures the cloudiness or presence of suspended particles in the water (measured in NTU), and the ORP sensor measures the oxidation-reduction potential (in mV), which indicates the water's ability to break down contaminants. These parameters together provide a comprehensive evaluation of water quality.

The ESP32 microcontroller serves as the central control unit of the system. It receives analog inputs from each sensor through its built-in 12-bit Analog-to-Digital Converter (ADC). The raw voltage signals are first read and then converted into digital values. To ensure precision, each sensor is calibrated using predefined mathematical equations that convert voltage readings into accurate measurement units. For example, the pH sensor output voltage is converted into a pH value using a linear calibration curve, while the TDS sensor reading is calculated using a conductivity-based conversion formula. This ensures that all sensor outputs reflect the true physical conditions of the water being tested.

After data acquisition and conversion, the data processing phase takes place. The ESP32 executes the main control algorithm written in INO (ESP32 framework). This algorithm continuously reads data from the sensors at regular intervals, filters out noise, and performs real-time computation to interpret the data correctly. Threshold values for each water parameter are predefined according to international water quality standards such as those provided by the World Health Organization (WHO). The system continuously compares the measured data against these threshold values to determine the condition of the water.

For example:

- If $\text{pH} < 6.5$ or $\text{pH} > 8.5$, the water is considered unsafe and a pH alert is triggered.
- If $\text{TDS} > 1000$ ppm, the system indicates high mineral content and poor water purity.
- If $\text{Turbidity} > 5$ NTU, it suggests high particle concentration or contamination.
- If $\text{ORP} < 250$ mV, it indicates poor oxidation potential and possible bacterial presence.

Once the decision-making process is complete, the ESP32 performs the control and communication functions. It transmits the processed data wirelessly to a web server or IoT dashboard using Wi-Fi. The web interface allows users to visualize water quality in real time through charts, numeric displays. The use of IoT technology enables remote monitoring, which is highly beneficial for water management in rural and industrial areas. In case the analog-to-digital conversion fails or produces invalid data, the system identifies the issue and displays a “Sensor Error” message, prompting the user to check the connection or recalibrate the sensor. Similarly, if the Wi-Fi data transmission fails, the ESP32 temporarily stores the measured data locally and retries sending once the connection is re-established. This ensures data integrity and prevents data loss during network interruptions.

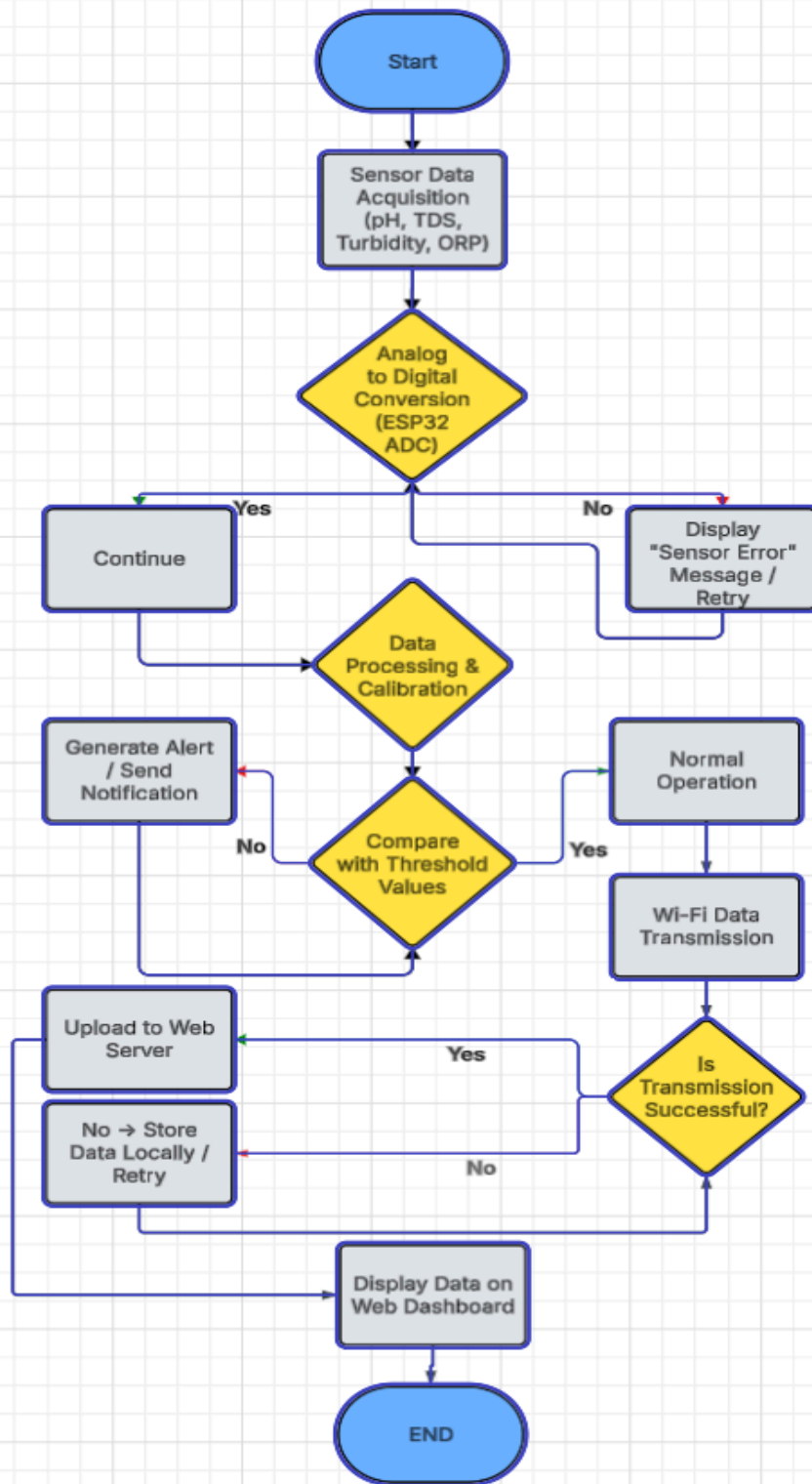


figure 12. Flowchart of the data acquisition and processing sequence in the water quality monitoring system

4.2 Sensor Selection

4.2.1 pH Sensor (Ph4502c)

Purpose:The pH sensor is used to measure the hydrogen ion concentration in water, which indicates whether the water is acidic, neutral, or alkaline. pH is one of the most critical parameters in determining water quality because it affects chemical reactions, biological processes, and the solubility of metals.

Reason for Selection:A pH sensor was selected due to its reliability, accuracy, and wide range of measurement (typically 0–14 pH). It provides real-time monitoring of water acidity or alkalinity, which is essential for maintaining safe water quality standards. It is also compatible with microcontrollers like ESP32 for IoT-based monitoring.

Formula

$$pH = -\log_{10}[H^+]$$

Equation 2. Formula for calculating pH from $[H^+]$.

Explanation:

The sensor measures the potential difference between two electrodes (the measuring and reference electrodes). This relationship is defined by the **Nernst equation**:

$$E = E_0 - \frac{2.303RT}{nF} \log_{10}[H^+]$$

Equation 3. Electrode potential dependence on pH from the Nernst equation.

From this, the pH can be derived as:

$$pH = \frac{E_0 - E}{0.0591}$$

Equation 4. Simplified Nernst equation for pH calculation.

(at approximately 25°C)

Where:

- E – Measured electrode potential (V)
- E_0 – Reference potential (V)

- R – Gas constant (8.314 J/mol·K)
- T – Temperature (Kelvin)
- F – Faraday constant (96485 C/mol)
- n – Number of electrons involved in the reaction

4.2.2 ORP (Oxidation-Reduction Potential) Sensor (ORP board v1.1)

Purpose: The ORP sensor measures the oxidation-reduction potential of water, indicating its ability to oxidize or reduce contaminants. It helps determine the sanitizing strength of water, especially in processes involving disinfection, wastewater treatment, and environmental monitoring.

Reason for Selection: The ORP sensor was chosen because it gives insight into water's chemical stability and contamination level. It complements the pH sensor to provide a complete electrochemical profile of water. It's also durable, easy to calibrate, and suitable for long-term continuous measurement in harsh environments.

Formula:

$$E = E_0 + \frac{RT}{nF} \ln \frac{[\text{Oxidized species}]}{[\text{Reduced species}]}$$

Equation 5. Nernst equation describing redox equilibrium.

Explanation:

The ORP sensor measures the ability of water to oxidize or reduce contaminants. It indicates the oxidation-reduction balance of a solution, which reflects the water's sanitizing strength or contamination level. The sensor outputs a potential value (in millivolts) corresponding to the electron exchange between oxidized and reduced ions in the solution.

4.2.3 TDS (Total Dissolved Solids) Sensor (GravityTDS)

Purpose:The TDS sensor measures the concentration of dissolved solids such as salts, minerals, and metals in the water. High TDS values can indicate contamination or poor water quality, affecting both industrial and domestic water applications.

Reason for Selection:The TDS sensor was selected because it provides a quick and practical way to assess water purity. It is widely used, cost-effective, and easy to interface with microcontrollers. It helps determine water suitability for drinking, irrigation, or industrial processes.

Formula:

$$TDS(ppm) = k \times EC(\mu S/cm)$$

Equation 6. Formula converting EC to TDS.

Explanation:

The TDS sensor measures the total concentration of dissolved solids such as minerals, salts, and metals in water. It is derived from the electrical conductivity (EC) of the solution using a conversion factor k , typically between 0.5 and 0.7. Higher TDS values indicate higher concentrations of dissolved substances and lower water purity.

Where:

- EC – Electrical conductivity of the water sample ($\mu S/cm$)
- k – Conversion factor (usually 0.5–0.7 depending on water type)

4.2.4 Turbidity Sensor (TDS V1)

Purpose: The turbidity sensor measures the cloudiness or haziness of water caused by suspended solids. It is a key parameter for detecting physical contamination and evaluating filtration or sedimentation efficiency.

Reason for Selection: The turbidity sensor was selected because it offers an optical method to continuously monitor water clarity. It provides immediate feedback on changes in water quality due to pollution or sediment disturbance. The sensor is robust, low-cost, and easy to integrate into automated systems.

Formula:

$$Turbidity(NTU) = \alpha \times \frac{V_{ref} - V_{out}}{V_{out}} + b$$

Equation 7. Voltage-based turbidity calculation formula.

Explanation:

The turbidity sensor measures water cloudiness or the amount of suspended particles by analyzing the scattering of light. When the water becomes more turbid, less light reaches the detector, resulting in a change in output voltage (V_{out}). The relationship between V_{out} and turbidity (in NTU) is established during calibration using standard solutions.

Where:

- V_{ref} – Reference voltage (clean water)
- V_o – Measured output voltage
- a, b – Calibration constants determined experimentally

4.3 Component Selection

The main reason for selecting these sensors (pH, ORP, TDS, and Turbidity) is that they are low-cost, easy to use, and reliable for general applications. Although these components may not provide the same precision and durability as high-end industrial-grade devices, they are highly suitable for academic projects, prototype systems, and small-scale water quality monitoring applications.

These sensors offer an excellent balance between price, performance, and simplicity. They are widely available, easy to interface with microcontrollers like the ESP32, and require minimal calibration and maintenance. This makes them accessible and practical for users with limited technical background or budget constraints.

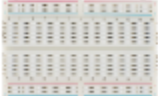
Moreover, to maintain a low-cost system design, the number of components and complex circuits were minimized without compromising the essential functions of the system. As a result,

the overall setup remains affordable, compact, and efficient, allowing real-time water quality monitoring at a fraction of the cost of commercial systems.

4.4 Required Hardware List

The following table provides a summary of the hardware components employed in the indoor smart farm prototype, along with their respective quantities and functional roles.

Table 2. List of Hardware Components Used

No	Component	Quantity	Purpose	Picture
1	ESP32	1	Main controller for sensor reading and actuator control	
2	ORP sensor	1	Oxidation Reduction Potential (ORP)	
3	Ph sensor	1	A pH sensor is a probe that measures the acidity or alkalinity	
4	TDS sensor	1	A TDS sensor measures the Total Dissolved Solids	
5	Turbidity	1	A turbidity sensor measures the cloudiness or haziness	
6	Breadboard / PCB	1	Prototyping platform for hardware assembly	
7	Type-C Cable (ESP32)	1	Upload programs and provide power during testing	

8	Connecting Wires	Several	Circuit connections between components	
---	------------------	---------	--	---

Future planned improvements to the water quality monitoring system include the integration of a temperature sensor to enhance the accuracy of measurements through temperature compensation and the development of a mobile application to enable real-time data monitoring, user notifications, and remote system management.

The complete hardware wiring and sensor configuration are illustrated in Figure X. This diagram presents the interconnections between the ESP32 microcontroller and the water quality sensors, including the pH, ORP, TDS, and turbidity sensors. Additionally, an OLED display is integrated to provide real-time on-site monitoring of the measured parameters.

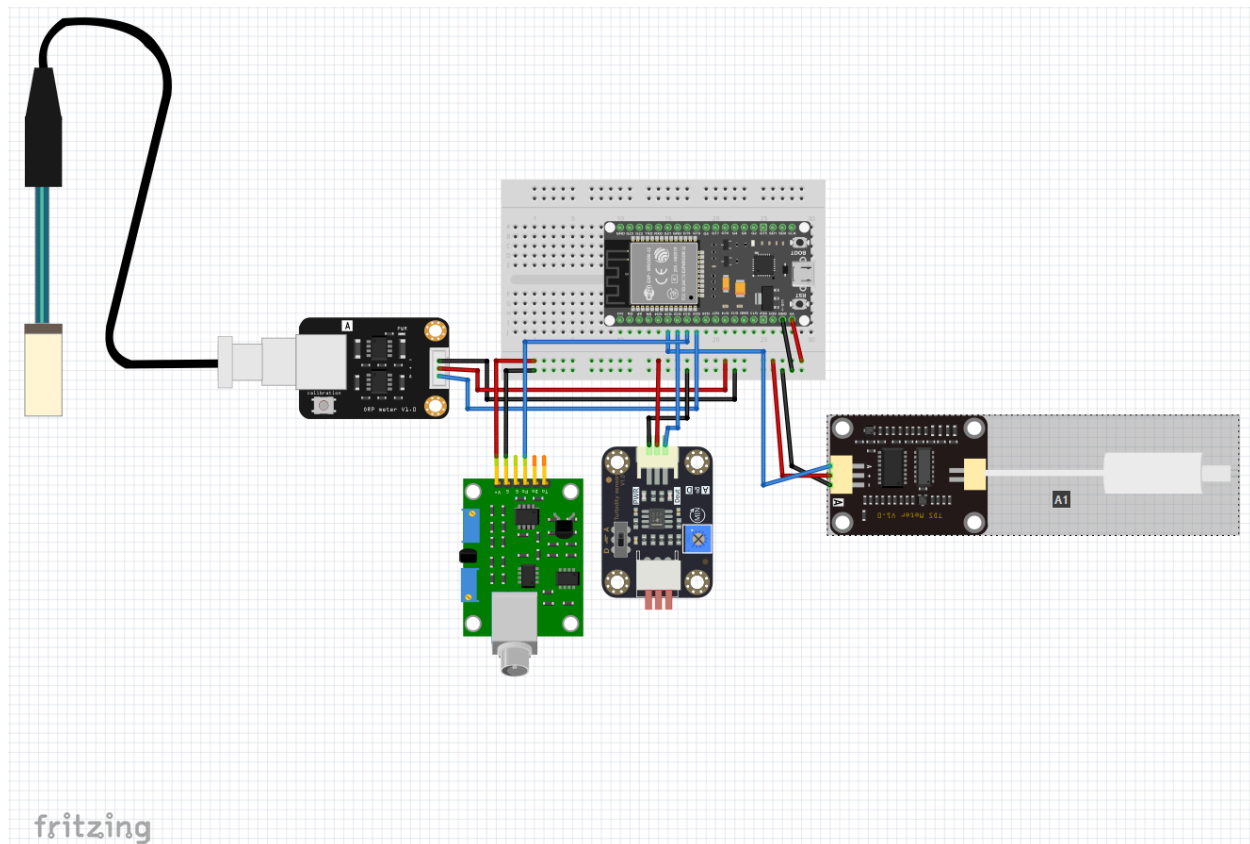


Figure 13. .Schematic diagram of water quality monitoring system circuit, showing full wiring of sensors and power connections using Fritzing.

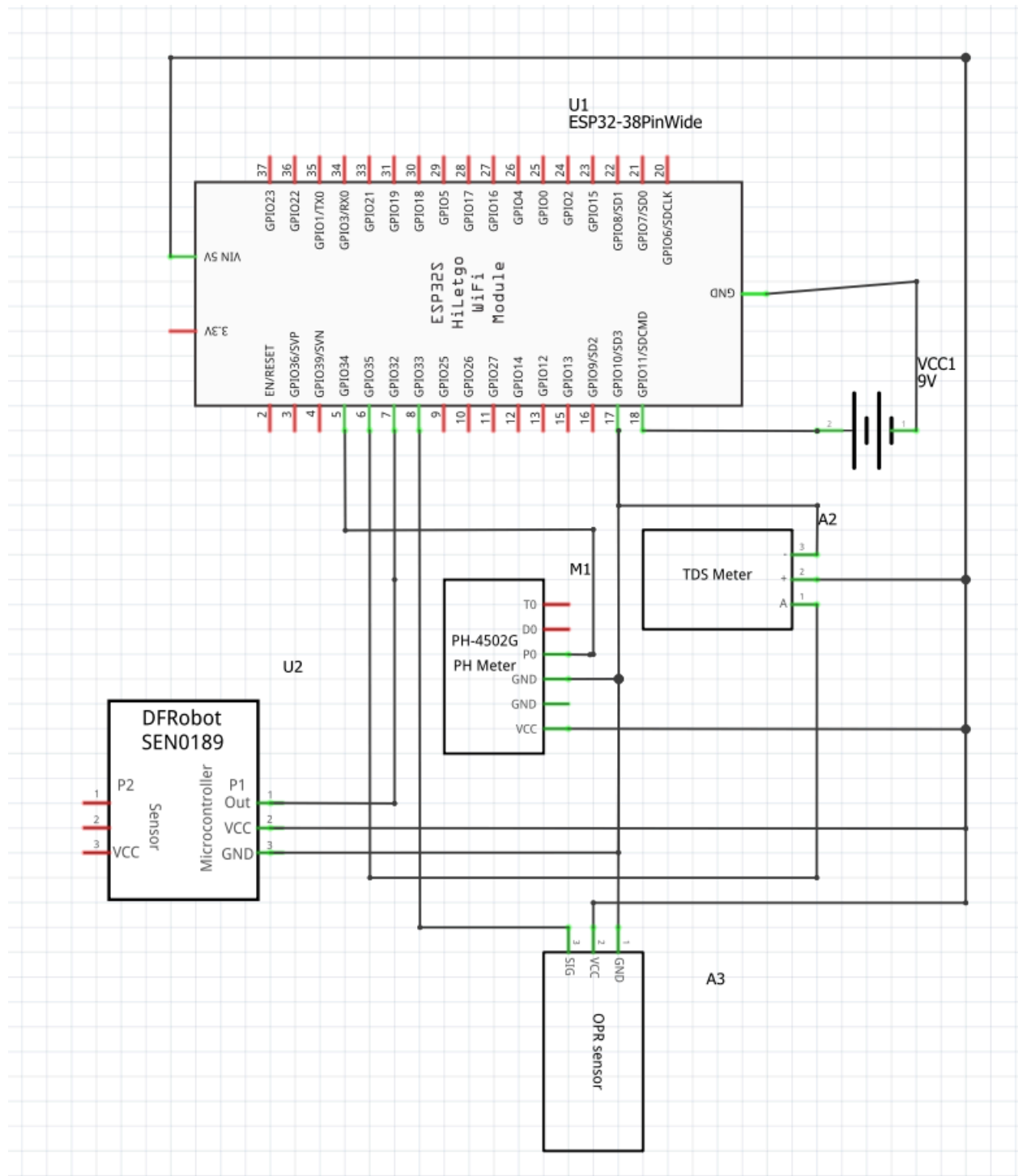


figure 14. Circuit schematic showing the connection between the ESP32 microcontroller and the water quality sensors (pH, ORP, and TDS).

4.5 Software Design

The developed software is designed for remote use via mobile devices, enabling users to monitor water quality data from any location. From a software architecture perspective, the system follows a layered design approach, consisting of four main components: the embedded system (microcontroller firmware), the backend server infrastructure, the database management system, and the frontend user dashboard. The software was implemented with a minimal and user-friendly interface, ensuring simplicity, efficiency, and accessibility for all users.

4.5.1. Embedded Firmware(ESP32)

The ESP32 microcontroller was programmed using the Arduino IDE, utilizing C++ as the primary language for embedded system development. The firmware is designed to handle all essential operations of the water quality monitoring system, ensuring accurate data acquisition, processing, and communication.

Sensor Data Acquisition: The ESP32 periodically reads analog inputs from the pH, ORP, TDS, and Turbidity sensors, converting the raw voltage signals into meaningful measurement values through calibration equations.

Control Logic Execution: The microcontroller compares each sensor reading against predefined threshold values based on the Mongolian National Drinking Water Standard (MNS 0900:2018). When abnormal readings are detected, the system flags alerts for unsafe water quality conditions.

Data Transmission: The ESP32's built-in Wi-Fi module enables real-time data transmission to the backend server using HTTP POST or WebSocket protocols, supporting both reliability and low-latency communication for web-based monitoring.

OLED Display Updates: The latest sensor readings (pH, ORP, TDS, and Turbidity) are displayed locally on an OLED screen, allowing immediate visualization of water quality status even without an internet connection.

The firmware implements interrupt handling, timing control, and basic error detection to maintain stable operation and prevent data loss during continuous monitoring. It also includes modular

functions for sensor calibration, data filtering, and network reconnection, ensuring robust and long-term system reliability.

4.5.2 Backend Server Implementation (Node.js + JavaScript + Arduino)

The backend server is implemented using Node.js and functions as the central communication layer between the sensing hardware and the frontend dashboard. It receives sensor data from the microcontroller through HTTP requests, validates the incoming measurements, and stores them in a persistent JSON file for later retrieval. The server also provides real-time updates to the dashboard through Socket.io, ensuring that new readings are displayed immediately without requiring page refresh. Additionally, the backend exposes endpoints for retrieving historical data, exporting measurements as CSV files, and relaying user-issued control commands back to the microcontroller. This implementation enables reliable data acquisition, storage, visualization, and bidirectional control within the overall monitoring system.

4.5.3 Frontend Dashboard (Web Interface)

The frontend dashboard serves as the main user interface of the water-quality monitoring system. Built with HTML, CSS, and JavaScript, it communicates with the backend through RESTful API calls and Socket.io to provide real-time visualization of water-quality parameters, including pH, TDS, Turbidity, and ORP.

Live sensor readings are displayed through numeric indicators and graphical charts, with Socket.io ensuring that values update instantly without requiring page refreshes. In addition to real-time monitoring, the dashboard also supports data management. Each reading is stored with a timestamp, and users can retrieve historical data through the `/data` API endpoint for tabular or graphical viewing.

Several interactive features enhance usability:

- **Save:** Stores the current sensor values and timestamps in the backend.
- **Download:** Exports all stored data as a CSV file for offline analysis.
- **Clear:** Removes all stored records from the backend when they are no longer required.

Visual alerts notify users of unsafe water conditions or communication issues with the ESP32 device. The interface follows a minimalist, responsive design to ensure clarity on both desktop and mobile devices. Overall, the dashboard provides an efficient and accessible user

experience, linking the embedded hardware with the backend system for real-time and historical water-quality monitoring.

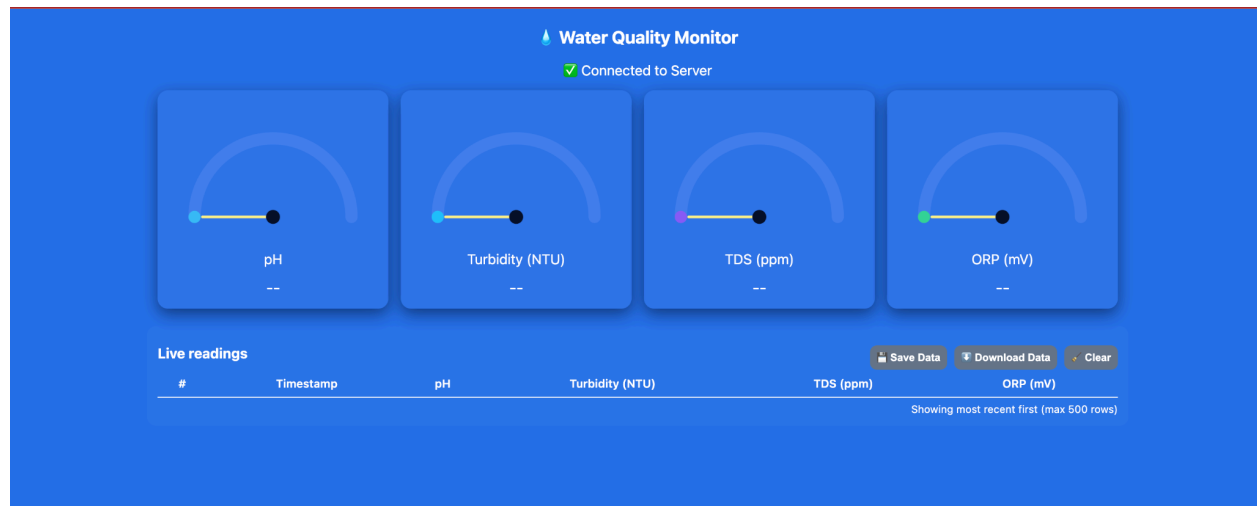


figure 15. Web-based dashboard interface for real time water quality monitoring

4.5.4 System Integration Overview

- **Sensing → Backend:**

The ESP32/Arduino collects water-quality measurements and transmits them to the backend via HTTP POST requests (`/api/data`), including sensor values and an ISO 8601 timestamp.

- **Backend → Storage:**

The backend validates each reading and stores it in a persistent JSON file (`data.json`), which functions as the system's primary data repository during prototyping.

- **Backend → Frontend (Real-Time):**

Newly received measurements are broadcast to the frontend dashboard through Socket.io events (e.g., `sensorData`), enabling immediate real-time updates.

- **Frontend → Backend (Historical & Export):**

The dashboard retrieves stored measurements from `data.json` via `GET /data`, while `/save` and `/download` enable CSV generation and file export.

- **User → Device (Control):**

User-initiated control commands are sent from the dashboard to the backend, which then relays them to the ESP32/Arduino through HTTP or an available serial interface.

- **Optional Serial Integration:**

When serial port support is enabled, the backend may directly read from or write to a USB/COM device; otherwise, this feature remains inactive.

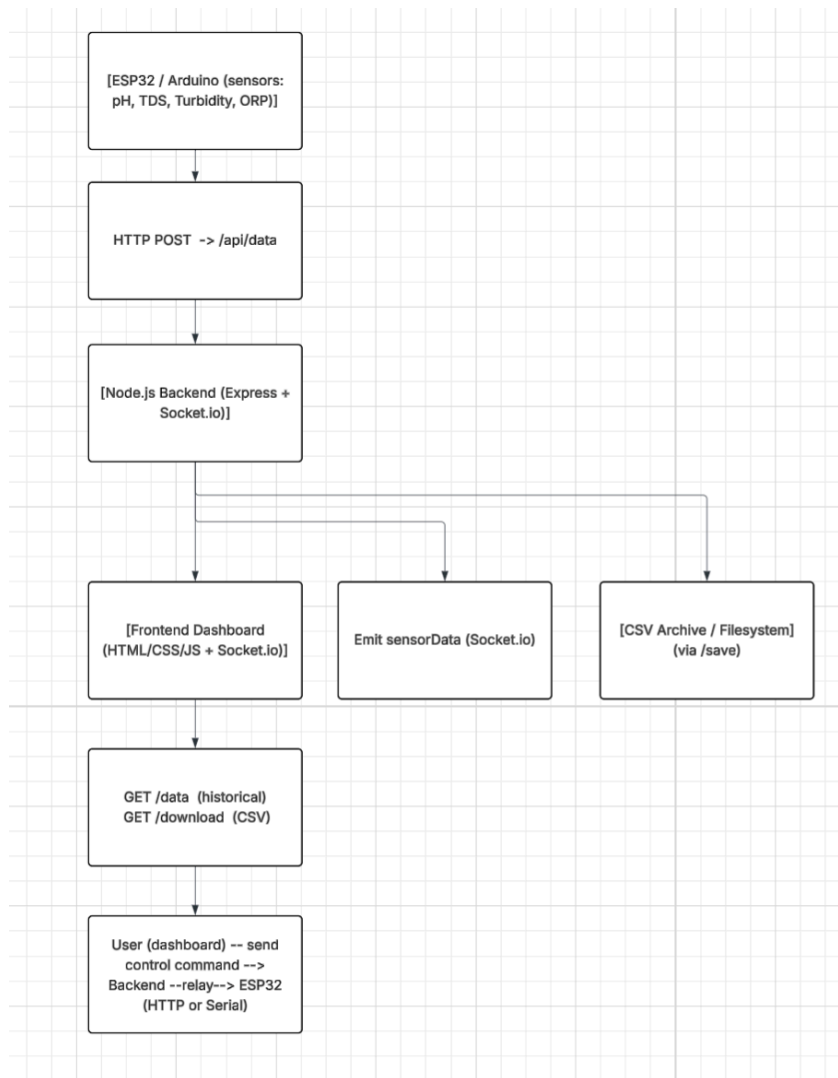


figure 16. presents the system's end-to-end data flow. Sensor measurements from the ESP32/Arduino are sent to the backend server, which broadcasts them in real time to the frontend dashboard and saves them for CSV export. The dashboard retrieves both live and historical data and allows users to issue control commands. These commands are returned to the ESP32 via HTTP or serial communication, completing the control cycle.

5. Implementation and Results

The implementation phase involved integrating both hardware and software components into a fully functional ESP32-based water quality monitoring prototype. The system combines four analog sensors — pH, ORP, TDS, and turbidity — connected to the ESP32 microcontroller for continuous environmental data collection. An OLED display was incorporated to provide real-time on-site visualization, while a Wi-Fi connection enables data transmission to the Node.js-based backend server.

The firmware was programmed using the Arduino IDE in C++, where the ESP32 was configured to read analog voltage signals from each sensor, apply calibration formulas, and convert these into meaningful water quality values. Each reading cycle was time-stamped and transmitted to the backend server via HTTP POST requests in JSON format.

On the software side, the backend server was implemented using Node.js, Express, and Socket.io, while the frontend dashboard was developed using HTML, CSS, and JavaScript. The backend receives sensor data from the ESP32, stores it temporarily in memory (or optionally in MongoDB), and broadcasts real-time updates to all connected clients. The frontend displays this information as interactive charts and tables for user analysis.

Figure X illustrates the overall system integration, showing the connection between sensors, the ESP32 microcontroller, the backend server, and the frontend dashboard. The hardware assembly and wiring layout are shown in Figure Y.

The prototype system was tested using three different water samples:

- Tap Water (Ulaanbaatar city)
- Bottled Drinking Water
- River Water (Tuul River)

The sensors collected continuous data over multiple readings for each sample, and the measured values were compared against the Mongolian National Drinking Water Standard.

5.1 Hardware Assembly

The hardware assembly of the water-quality monitoring system integrates the DOIT ESP32 DevKit V1 microcontroller with four analog sensing modules—pH (PH4502C), ORP (ORP Board v1.1), TDS (Gravity TDS and TDS V1), and a turbidity module—alongside a 1.3-inch OLED display for local visualization. Each sensor and peripheral component was connected to the ESP32 according to its electrical interface requirements to ensure stable and accurate data acquisition.

The **PH4502C pH sensor module** includes an onboard signal conditioning circuit that converts the probe's low-level electrochemical signal into a stable analog voltage readable by the ESP32 ADC. This module was powered through the 5 V rail while its signal output was connected to one of the ESP32's analog input pins. Proper grounding was maintained to reduce noise and drift during measurement.

The **ORP Board v1.1** operates similarly, providing a regulated amplification stage for the ORP probe. Its analog output, centered around 2.5 V, was connected to an ESP32 ADC pin, enabling millivolt-level redox potential conversion through software calibration. This module also required stable 5 V power and a shared ground reference.

The system utilizes two TDS sensing modules: the **Gravity TDS sensor** and the **TDS V1 module**. Both include electrical isolation and voltage regulation to protect the ESP32 from conductivity-related interference in the water sample. Their analog outputs were connected to separate ADC channels, while the isolation circuits were powered by 5 V to maintain safe and stable readings.

A **1.3-inch OLED display** (I2C interface) was integrated to provide real-time local feedback. The display was connected to the ESP32's I2C pins (SDA and SCL) and powered through the 3.3 V rail to ensure compatibility with the microcontroller's logic level. This arrangement allowed the system to show live pH, TDS, turbidity, and ORP values without requiring a computer connection.

All sensors were mounted on a stable platform, and the waterproof probes were immersed in the test solution while keeping the electronics isolated from moisture. Common grounding practices and short cable lengths were used to minimize noise and ensure reliable analog signal acquisition. The ESP32 also handled Wi-Fi communication, allowing sensed data to be transmitted to the backend server for processing and visualization.



figure 17. Assembled prototype showing OLED display output and internal wiring

5.2 Software Deployment

The software for the water-quality monitoring system was deployed locally on a laptop, which hosted both the backend server and the frontend dashboard. The ESP32 firmware was developed using the **Arduino IDE** and uploaded directly to the **DOIT ESP32 DevKit V1** via USB. All required sensor libraries and calibration routines were embedded into the firmware prior to deployment, enabling the microcontroller to automatically collect and transmit measurements once powered on.

The backend server was implemented using Node.js and executed with the npm start command. This process initialized the Express-based API endpoints, WebSocket communication through Socket.io, and the file-based data storage system. Deployment relied on several key project files, including server.js for backend logic, script.js for frontend interaction, and the public/ directory containing the user interface. The backend operated entirely on **localhost**, meaning both the server and dashboard were accessible only from the same machine during testing and development.

Sensor data was stored in a local data .json file, while CSV export files were generated on demand to support data archiving and offline analysis. All data handling logic, including calibration adjustments and numerical conversions, was integrated into the backend and firmware code during deployment to ensure consistent behavior across devices.

Although the overall deployment process was straightforward, several challenges arose during sensor calibration and firmware coding. These included adapting outdated Arduino libraries, modifying analog-to-value conversion formulas, and ensuring correct communication timing between the ESP32 and the backend. Once resolved, the system operated reliably in a fully local environment, providing real-time monitoring and control capabilities for all connected sensors.

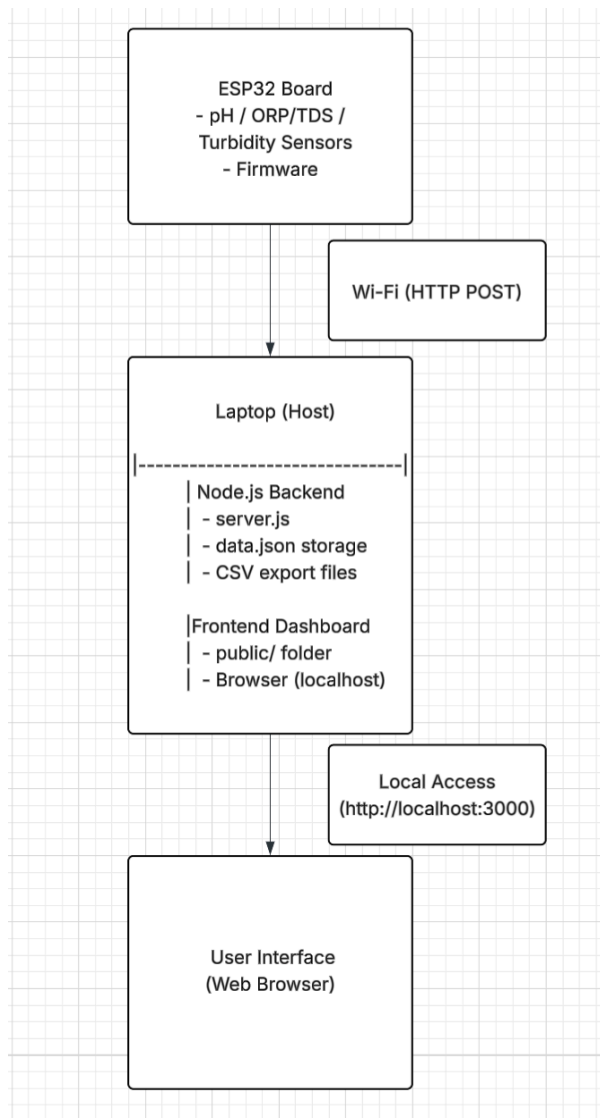


figure 18. System architecture diagram showing data flow between hardware, backend and user interface

5.3 Testing and Troubleshooting

To evaluate the performance of the system under real conditions, several water samples were collected and tested using the assembled device. Tap water, bottled water, and river water were used as representative samples. All measurements were taken at a controlled temperature of approximately 25 °C to ensure consistency across tests. The system recorded pH, TDS, turbidity, and ORP values for each sample, and the results were compared to observe differences in water quality. These comparisons are presented in the graphs below, where the variations between the three water sources can be clearly seen.

The pH graph shows noticeable variation among the three samples. The river water measured pH 6.1, indicating slightly acidic characteristics, which is typical for natural surface water. The bottled water measured pH 6.5, still mildly acidic but more balanced than the river sample. In contrast, the tap water displayed a significantly higher value of pH 7.2, which is slightly alkaline and within the expected range for treated municipal water. Overall, the system successfully detected clear differences in pH levels across the three water sources.

pH Experiment results

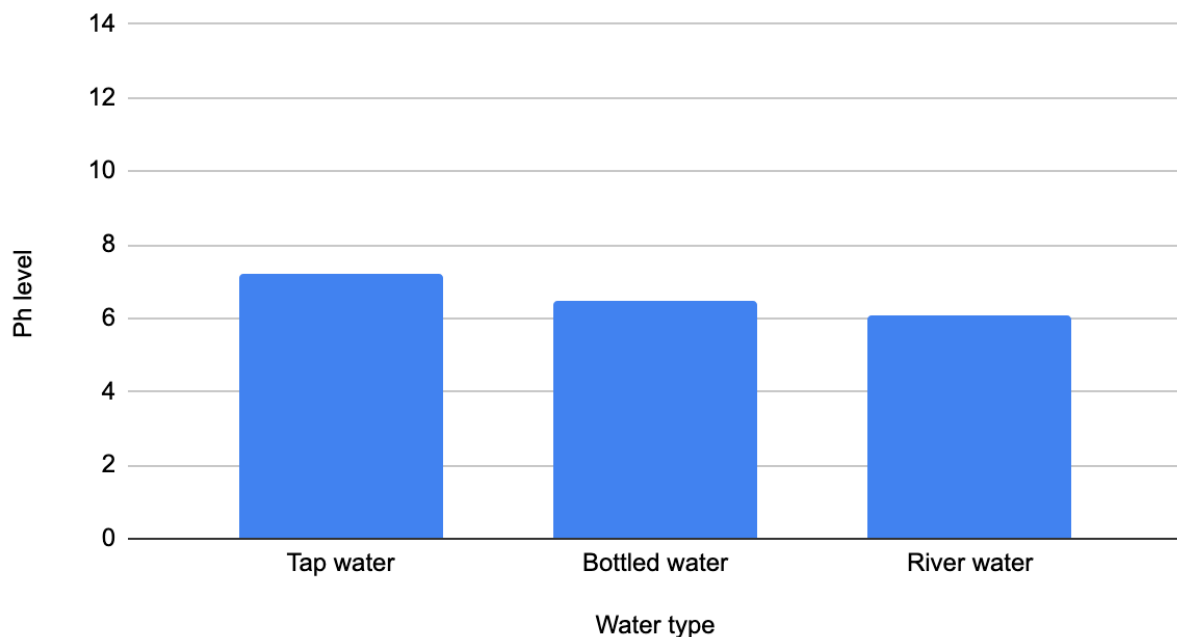


figure 19. pH Comparison of River, Bottled, and Tap Water Samples

The TDS results show significant variation among the three tested water sources. The bottled water sample recorded the lowest TDS level at 57 mg/L, indicating low mineral content and effective filtration. Tap water measured 213 mg/L, which reflects a moderate concentration of dissolved minerals typically found in municipal water supplies. In contrast, the river water sample exhibited the highest TDS value at 263 mg/L, suggesting a greater presence of natural sediments, organic materials, and dissolved solids. Overall, the data show that bottled water is the least mineralized, while river water contains the highest concentration of dissolved substances.

Total Dissolved Solids Experiment result

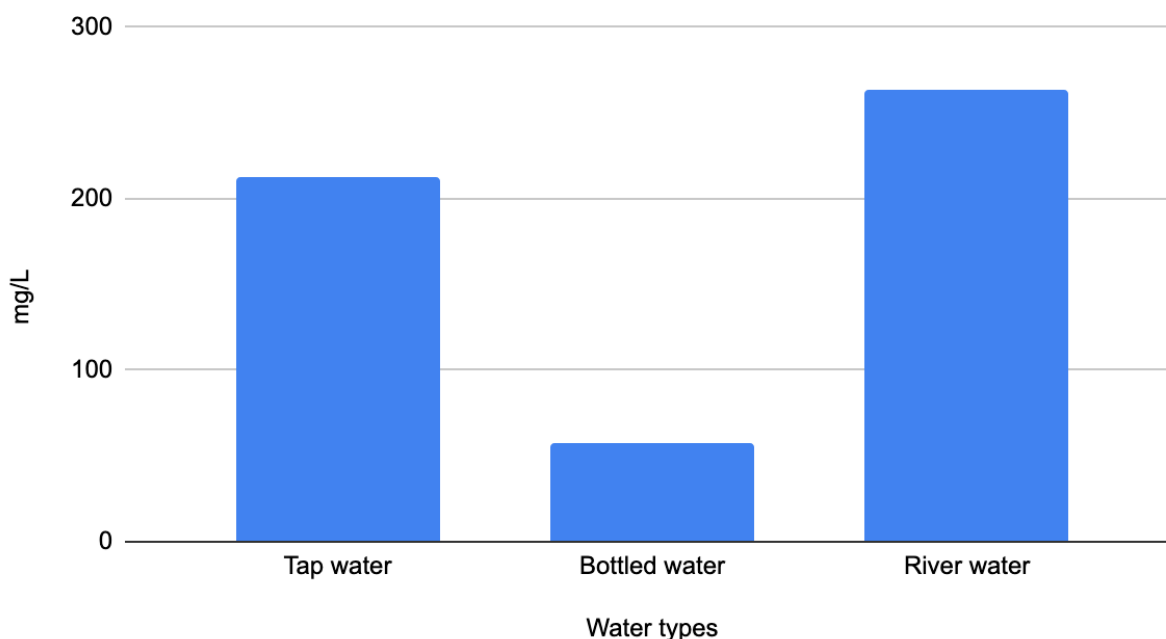


figure 20. Total Dissolved Solids Comparison of River, Bottled, and Tap Water Samples

The turbidity results show a clear difference between treated/filtered water and natural surface water. Both tap water and bottled water recorded turbidity levels of less than 1 NTU, indicating very clear water with minimal suspended particles. In contrast, the river water sample measured approximately 9 NTU, which is significantly higher. This increased turbidity is expected, as river water naturally contains suspended sediments, organic matter, and particles from the surrounding environment. Overall, the results confirm that untreated river water has noticeably higher turbidity compared to tap and bottled water.

The ORP results follow the expected inverse relationship with pH. The river water sample showed the highest ORP value at 314 mV, indicating stronger oxidizing conditions, which corresponds with its lower pH level. Bottled water recorded a moderate ORP of 299 mV, consistent with its slightly acidic pH. Tap water exhibited the lowest ORP value at 260 mV, matching its higher pH of the three samples. Since ORP decreases as pH increases, these measurements align well with theoretical behavior. Overall, the results demonstrate that the system accurately captured the inverse correlation between pH and ORP across the tested water samples.

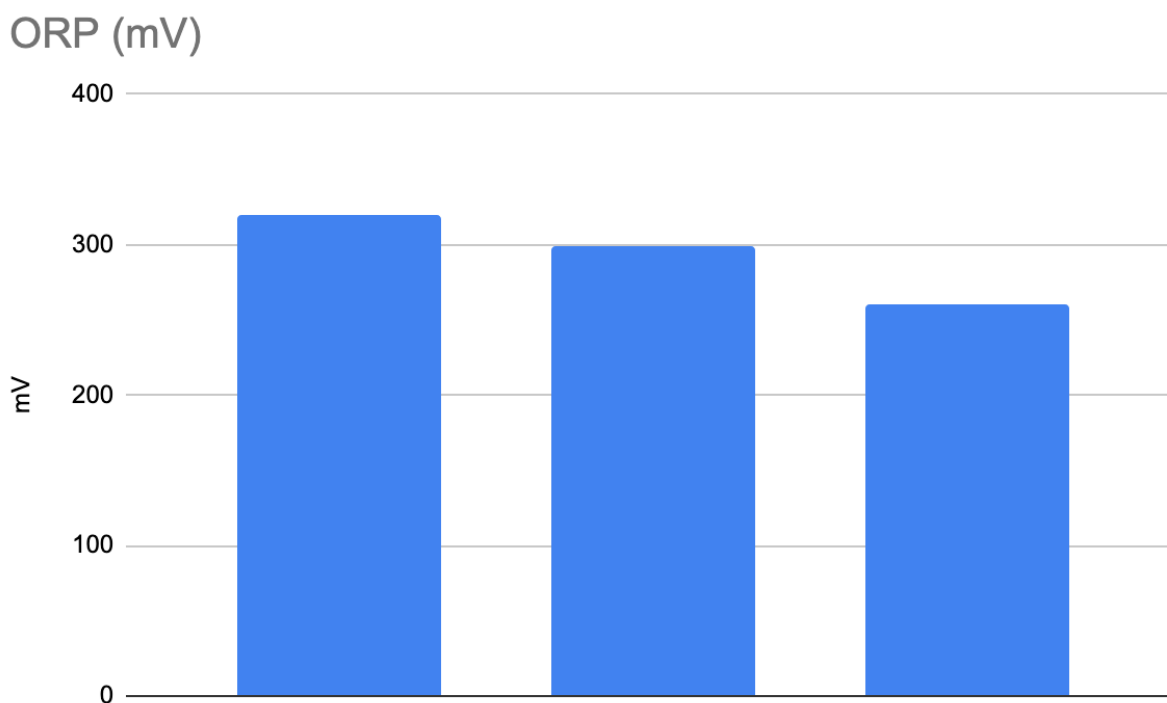


figure 21. Oxidation-Reduction Potential Comparison of River, Bottled, and Tap Water Samples

The results from all measured parameters—pH, TDS, turbidity, and ORP—show clear distinctions between the three tested water sources. Tap water demonstrated the highest pH level (slightly alkaline) and the lowest ORP value, indicating weaker oxidizing conditions. Its TDS level was moderate, reflecting typical mineral content found in treated municipal water, while turbidity remained very low, confirming that it is well-filtered and clear.

Bottled water showed a near-neutral pH and the lowest TDS value among the samples, suggesting effective filtration and low mineral content. Its ORP value was moderate and turbidity remained below 1 NTU, indicating that bottled water is clean and stable with minimal suspended particles.

River water, in contrast, showed the lowest pH (slightly acidic) and the highest ORP value, reflecting stronger oxidizing conditions. It also recorded the highest turbidity (around 9 NTU) due to suspended sediments common in natural water sources. Its TDS level was also the highest among the samples, indicating a greater presence of dissolved solids, organic materials, and minerals.

Overall, the comparison confirms that bottled water is the cleanest and most balanced, tap water is safe but mineralized, and river water contains the highest level of impurities, which is expected for untreated natural water. These results also demonstrate that the developed system successfully detects meaningful differences in water quality across varying environmental conditions.

6. Result and conclusion

The results of this project demonstrate that a low-cost, microcontroller-based water-quality monitoring system can reliably measure four essential parameters—ORP, TDS, turbidity, and pH—at two-second intervals and transmit these measurements to both a computer interface and an integrated display. The system performs continuous real-time analysis of water conditions, offering an affordable and practical alternative to conventional laboratory-grade equipment. Through iterative calibration, firmware refinement, and hardware optimization, the device achieved stable and responsive measurements, confirming that accurate environmental sensing is achievable even with budget-friendly components.

A major strength of the system is its flexibility and openness. The use of open-source hardware and software ensures that the device is easy to maintain, repair, and modify, enabling future users to expand its capabilities without significant redesign. Its compact design and low power consumption make the system suitable for small-scale field investigations, educational demonstrations, and community-based water monitoring programs. When compared with commercial multiparameter analyzers, this prototype is several times more cost-effective while maintaining a reliable level of operational stability.

From an environmental perspective, the system's simplicity and accessibility make it a valuable tool for everyday water assessment. Since it requires only a power source to begin monitoring, it can be used by non-experts, community members, or field researchers who need rapid information without specialized laboratory equipment. Its affordability reduces financial barriers for schools, rural communities, and institutions that lack access to expensive testing devices.

This is especially relevant in Mongolia, where water quality can vary significantly across regions, and where early detection of contamination in rivers, wells, and local water supplies is crucial for public safety.

The system also provides strong educational value. By combining sensor calibration, data acquisition, electronics, and programming, it creates a hands-on learning platform for students to explore environmental science and IoT-based measurement systems. Students can interact with real data, adjust calibration parameters, investigate trends, and understand how modern monitoring technologies are engineered and deployed. This strengthens both theoretical understanding and practical technical skills.

Overall, the project demonstrates that accessible technologies—supported by structured calibration procedures and thoughtful system integration—can deliver meaningful environmental data at a fraction of the cost of professional instruments. The system's affordability, ease of use, and adaptability make it well-positioned for educational use, public awareness initiatives, and environmental monitoring efforts in diverse contexts. This work contributes to the growing field of low-cost IoT environmental sensing and highlights the potential for scalable, user-friendly, and customizable solutions for water-quality evaluation.

7. Discussion

While the current water-quality monitoring system meets its core objectives, several opportunities exist for improvement and expansion in the future:

1. Improved Calibration Accuracy

Future versions should include multi-point calibration for all sensors—pH, ORP, TDS, and turbidity. Using a wider range of reference solutions will increase measurement precision and reduce long-term drift.

2. Redesign of the Equipment Base

The physical base of the device can be redesigned to provide better stability, improved cable management, and stronger protection against environmental interference. A more robust mounting frame would also help reduce sensor noise.

3. Larger and More Readable Display

Upgrading to a larger, higher-contrast display will make data easier to read during field use. A touchscreen interface could also improve user interaction and menu navigation.

4. Increased Portability

A compact, lightweight enclosure would make the system more portable and suitable for on-site water testing. A rechargeable battery module could also be added for mobile operation.

5. Enhanced Firmware and Software Features

Future software improvements may include automated data validation, adaptive sensor filtering, and more detailed error reporting. Additional calibration tools could be built into the firmware for quick on-site sensor adjustment.

6. Expanded Data Logging and Analytics

Long-term data storage in a database, combined with online dashboards or cloud services, would enable trend analysis, comparisons across locations, and remote monitoring capabilities.

8. References

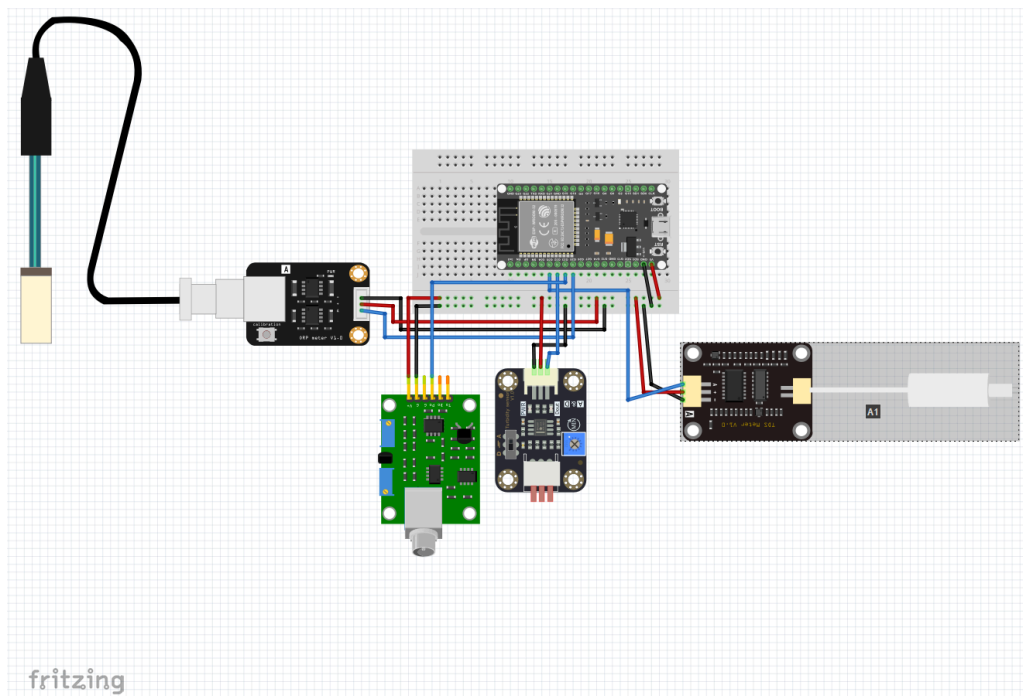
1. Batjargal, Z., Tsogt, T., & Enkhbaatar, E. (2020). Water pollution in Ulaanbaatar: Causes, challenges, and solutions. *Environmental Science and Pollution Research*, 27(18), 225-234.
2. Mongolian Agency for Standardization and Metrology. (1998). *MNS 4586:1998. Water environment quality indicators. General requirements*. Ulaanbaatar.
3. Jin, J., Zhang, W., & Liu, X. (2019). Review of water quality parameters and monitoring techniques. *Water Science and Technology*, 80(12), 2307-2322.
4. Mongolian Standards Agency. (2018). *MNS 0900:2018 Environment. Protecting health. Safety. Drinking water. Hygiene requirements, quality and safety assessment* (MNS 0900:2018). Ulaanbaatar: Mongolian Standards Agency.
5. MNS 0900:2005. Environment. Health protection. Safety. Drinking water. Hygienic requirements, assessment of the quality and safety. Ulaanbaatar: Mongolian Agency for Standardization and Metrology, 2005.
6. Hong, W. J., Shamsuddin, N., Abas, E., Apong, R. A., Masri, Z., Suhaimi, H., Gödeke, S. H., & Noh, M. N. A. (2021). *Water Quality Monitoring with Arduino Based Sensors. Environments*, 8(1), 6. <https://doi.org/10.3390/environments8010006>
7. Hakimi, I. M., & Jamil, Z. (2021). *Development of a water quality monitoring device using Arduino UNO*. IOP Conference Series: Materials Science and Engineering, 1144 (1), Article 012064. <https://doi.org/10.1088/1757-899X/1144/1/012064> [ResearchGate+1](#)
8. Gokulanathan, S., Manivasagam, P., Prabu, N., & Venkatesh, T. (2019). *A GSM Based Water Quality Monitoring System Using Arduino*. Shanlax International Journal of Arts, Science and Humanities, 6(4), 22-26. <https://doi.org/10.34293/sijash.v6i4.341>
9. M. Cantelon MHTHNR. *Node.js in Action*. 2nd ed. Shelter Island, NY: Manning Publications; 2013.
10. DFRobot. (n.d.). *Gravity: Analog TDS Sensor/Meter for Arduino SKU SEN0244*. Retrieved November 13, 2025, from https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor__Meter_For_Arduino_SKU__SEN0244
11. Strellok163. (2023, September 17). *pH, TDS/EC, ORP, temperature – metr*. Fritzing Forum. <https://forum.fritzing.org/t/ph-tds-ec-orp-temperature-metr/20558>
12. Fritzing. (n.d.). *Fritzing user manual: Full reference of all menus, widgets, keyboard and mouse shortcuts*. Retrieved November 13, 2025, from https://fritzing.org/learning/full_reference

13. Huynh, D. (n.d.). *Turbidity_Sensor – DTH_Turbidity_Sensor library for Arduino* [Computer software]. GitHub. https://github.com/duyhuynh/Turbidity_Sensor
14. Node.js Documentation. [Online].; 2023. Available from: <https://nodejs.org>.

9.APPENDICES

Appendix A – Schematic Diagram

- Full circuit diagram showing connection between DOIT ESP32 Dev v1,pH sensor, ORP sensor, TDS, Turbidity and OLED display.
- Label each pin connection clearly.



Appendix B – Arduino Code

Appendix B Arduino Code part 1

```
1
2 #include <U8g2Lib.h>
3 > /*****
11 *****/
12 #include <Arduino.h>
13 #include <Wire.h>
14 #include <WiFi.h>
15 #include <HttpClient.h>
16 #include <GravityTDS.h>
17 #include <ph4502c_sensor.h>
18
19
20 // ----- USER SETTINGS -----
21 const char* ssid = "Pasha";
22 const char* password = "negeesteg";
23 const char* apiUrl = "http://172.20.10.2:3000/api/data";
24 const unsigned long postIntervalMs = 30000;
25
26 // ----- HARDWARE PINS -----
27 #define PIN_PH 32
28 #define PIN_ORP 33
29 #define PIN_TDS 35
30 #define PIN_TURB 34
31 #define OLED_SDA 21
32 #define OLED_SCL 22
33
34 // ----- ADC SETTINGS -----
35 const float VREF = 3.3f;
36 const int ADC_BITS = 12;
37 const int ADC_MAX = (1 << ADC_BITS) - 1;
38 const int SAMPLES = 15;
39
40 // ----- TEMPERATURE (TDS compensation) -----
41 float waterTempC = 25.0f;
42
43 // ----- pH CALIBRATION (your measured points) -----
44 const float CAL_pH1 = 4.30f; const float CAL_V1 = 3.300f;
45 const float CAL_pH2 = 7.25f; const float CAL_V2 = 2.518f;
46 float PH_SLOPE = -2.744f;
47 float PH_INTERCEPT = 14.679f;
```

Appendix B Arduino Code part 2

```
48
49 // ----- TURBIDITY (simple linear) -----
50 const float V_TURB_CLEAR = 2.00f;
51 const float TURB_V_DIRTY = 1.00f;
52 const float TURB_NTU_DIRTY = 1000.0f;
53 float TURB_SLOPE;
54 float TURB_INTERCEPT;
55
56 // ----- ORP (DFRobot board) -----
57 const float ORP_CENTER_V = 2.5f;
58 const float ORP_SCALE_MV_PER_V = 1000.0f;
59 float ORP_OFFSET_MV = -1418.0f; // Calibrated for 1.332V => 250 mV in distilled water
60
61 // ----- OBJECTS -----
62 GravityTDS tdsSensor;
63 U8G2_SH1106_128X64_NONAME_F_HW_I2C display(U8G2_R0, /* reset= */ U8X8_PIN_NONE, OLED_SDA, OLED_SCL);
64 HttpClient http;
65
66 // ----- HELPERS -----
67 int readAverageRaw(int pin, int samples) {
68     long sum = 0;
69     for (int i = 0; i < samples; ++i) {
70         sum += analogRead(pin);
71         delay(3);
72     }
73     return (int)(sum / samples);
74 }
75
76 float rawToVoltage(int raw) {
77     return ((float)raw / ADC_MAX) * VREF;
78 }
79
80 void computePhCalibration() {
81     if (fabs(CAL_V2 - CAL_V1) < 1e-6) return;
82     PH_SLOPE = (CAL_pH2 - CAL_pH1) / (CAL_V2 - CAL_V1);
83     PH_INTERCEPT = CAL_pH1 - PH_SLOPE * CAL_V1;
84 }
85
86 void computeTurbidityCoeffs() {
87     TURB_SLOPE = (TURB_NTU_DIRTY - 0.0f) / (TURB_V_DIRTY - V_TURB_CLEAR);
88     TURB_INTERCEPT = -TURB_SLOPE * V_TURB_CLEAR;
```

```

89 }
90
91 // ----- SETUP -----
92 > void setup() {--
140 }
141
142 // ----- LOOP -----
143 unsigned long lastPost = 0;
144
145 > void loop() {--
246 }

```

Appendix C – Website Code

Node.js backend code: Server initialization, database communication routes, API for frontend.

```

Users > pashooxx > Downloads > water > public > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8" />
5 <meta name="viewport" content="width=device-width,initial-scale=1" />
6 <title>Water Monitor</title>
7 <style>
8 body{background:#2771e9;color:#ffffff;font-family:system-ui,Arial;margin:0;padding:20px;text-align:center}
9 h1{margin:0 0 14px;font-size:20px}
10 #status{color:#ffffff;margin-bottom:12px}
11 .grid{display:grid;grid-template-columns:repeat(auto-fit,minmax(220px,1fr));gap:14px;max-width:1100px;margin:0 auto}
12 .card{background:rgba(255,255,255,0.03);padding:14px;border-radius:12px;box-shadow:0 6px 18px rgba(2,6,23,0.6)}
13 canvas{width:100%;height:160px;display:block}
14 .label{margin-top:10px;color:#fcfddf}
15 .value{font-size:18px;margin-top:12px;color:#ffffff}
16 .table-wrap{max-width:1100px;margin:20px auto 0;background:rgba(255,255,255,0.02);padding:12px;border-radius:10px}
17 table{width:100%;border-collapse:collapse;font-size:13px;color:#e6eef8}
18 th,td{padding:8px 6px;border-bottom:1px solid rgb(255, 255, 255);text-align:center}
19 th{color:#ffffff;font-weight:600}
20 .controls{display:flex;gap:8px;justify-content:center;margin-top:10px}
21 button{background:#64748b;border:none;padding:8px 12px;border-radius:8px;color:#ffffff;font-weight:600;cursor:pointer}
22 button.secondary{background:#64748b;color:hsl(0, 0%, 100%)}
23 .small{font-size:12px;padding:6px 8px}
24 #errorMsg{color:#ff6b6b;font-size:14px;margin-top:10px;display:none}
25 </style>
26 </head>
27 <body>
28 <h1>▲ Water Quality Monitor</h1>
29 <div id="status">Connecting...</div>
30 <div id="errorMsg"></div>
31 <div class="grid">
32 <div class="card">
33 <canvas id="phGauge"></canvas>
34 <div class="label">pH</div>
35 <div class="value" id="phValue">--</div>
36 </div>
37 <div class="card">
38 <canvas id="turbGauge"></canvas>
39 <div class="label">Turbidity (NTU)</div>
40 <div class="value" id="turbValue">--</div>
41 </div>
42 <div class="card">
43 <canvas id="tdsGauge"></canvas>

```

Appendix C Arduino Code part 2

```

43 <canvas id="tdsGauge"></canvas>
44 <div class="label">TDS (ppm)</div>
45 <div class="value" id="tdsValue">---</div>
46 </div>
47 <div class="card">
48 <canvas id="orpGauge"></canvas>
49 <div class="label">ORP (mV)</div>
50 <div class="value" id="orpValue">---</div>
51 </div>
52 </div>
53 <div class="table-wrap">
54 <div style="display:flex;justify-content:space-between;align-items:center">
55 <div style="font-weight:700">Live readings</div>
56 <div class="controls">
57 <button id="saveBtn" class="small">💾 Save Data </button>
58 <button id="downloadBtn" class="small secondary">📄 Download Data </button>
59 <button id="clearBtn" class="small secondary">🗑️ Clear</button>
60 </div>
61 </div>
62 <table id="dataTable" aria-label="Sensor data table">
63 <thead>
64 <tr><th>#</th><th>Timestamp</th><th>pH</th><th>Turbidity (NTU)</th><th>TDS (ppm)</th><th>ORP (mV)</th></tr>
65 </thead>
66 <tbody></tbody>
67 </table>
68 <div style="text-align:right;color:#ffffff;font-size:12px;margin-top:6px">Showing most recent first (max 500 rows)</div>
69 </div>
70 <script src="/socket.io/socket.io.js"></script>
71 <script>
72 const socket = io({ reconnection: true, reconnectionAttempts: 10, reconnectionDelay: 1000 });
73 const statusEl = document.getElementById('status');
74 const errorEl = document.getElementById('errorMsg');
75 socket.on('connect', () => {
76   statusEl.textContent = '✅ Connected to Server';
77   errorEl.style.display = 'none';
78 });
79 socket.on('disconnect', () => {

```

Appendix C Arduino Code part 3

```

79 socket.on('disconnect', () => {
80   statusEl.textContent = '❌ Disconnected';
81   errorEl.textContent = 'Lost connection to server. Retrying...';
82   errorEl.style.display = 'block';
83 });
84 socket.on('connect_error', (err) => {
85   errorEl.textContent = 'Connection error: ${err.message}';
86   errorEl.style.display = 'block';
87 });
88 const dataList = [];
89 const MAX_ROWS = 500;
90 const tbody = document.querySelector('#dataTable tbody');
91 function renderTable(){
92   tbody.innerHTML = '';
93   for (let i = 0; i < dataList.length; i++){
94     const r = dataList[i];
95     const tr = document.createElement('tr');
96     tr.innerHTML = `<td>${i+1}</td>
97                   <td>${new Date(r.timestamp).toLocaleString()}</td>
98                   <td>${r.pH != null ? r.pH.toFixed(2) : '—'}</td>
99                   <td>${r.turbidity != null ? r.turbidity.toFixed(1) : '—'}</td>
100                  <td>${r.tds != null ? r.tds.toFixed(0) : '—'}</td>
101                  <td>${r.orp != null ? r.orp.toFixed(0) : '—'}</td>`;
102     tbody.appendChild(tr);
103   }
104 }
105 function pushReading(obj){
106   dataList.unshift(obj);
107   if (dataList.length > MAX_ROWS) dataList.pop();
108   renderTable();
109 }
110 function createGauge(canvas, min, max, color){
111   const ctx = canvas.getContext('2d');
112   canvas.width = canvas.clientWidth * devicePixelRatio;
113   canvas.height = canvas.clientHeight * devicePixelRatio;
114   const w = canvas.width, h = canvas.height;
115   let cur = min;
116   function draw(v){
117     ctx.clearRect(0,0,w,h);

```

Appendix C Arduino Code part 4

```

118 const cx = w/2, cy = h*0.82, r = Math.min(w*0.38, h*0.68);
119 ctx.lineCap = 'round';
120 ctx.beginPath(); ctx.lineWidth = 14*devicePixelRatio; ctx.strokeStyle='rgba(255,255,255,0.08)'; ctx.arc(cx,cy,r,Math.PI,2*Math.PI); ctx.stroke();
121 const end = Math.PI + (Math.max(min,Math.min(max,v)) - min)/(max-min) * Math.PI;
122 ctx.beginPath(); ctx.lineWidth = 14*devicePixelRatio; ctx.strokeStyle = color; ctx.arc(cx,cy,r,Math.PI,end); ctx.stroke();
123 const ang = Math.PI + (Math.max(min,Math.min(max,v)) - min)/(max-min) * Math.PI;
124 ctx.beginPath(); ctx.moveTo(cx,cy); ctx.lineTo(cx + Math.cos(ang)*(r-18), cy + Math.sin(ang)*(r-18));
125 ctx.strokeStyle = '#fef08a'; ctx.lineWidth = 3*devicePixelRatio; ctx.stroke();
126 ctx.beginPath(); ctx.fillStyle='#071029'; ctx.arc(cx,cy,8*devicePixelRatio,0,Math.PI*2); ctx.fill();
127 }
128 draw(cur);
129 return (v)>={ cur = v; draw(v); };
130 }
131 const phGauge = createGauge(document.getElementById('phGauge'), 0, 14, '#38bdf8');
132 const turbGauge = createGauge(document.getElementById('turbGauge'), 0, 100, '#22c1ff');
133 const tdsGauge = createGauge(document.getElementById('tdsGauge'), 0, 2000, '#8b5cf6');
134 const orpGauge = createGauge(document.getElementById('orpGauge'), -1000, 1000, '#34d399');
135 socket.on('sensorData', data => {
136   const now = new Date().toISOString();
137   if (data.pH !== null && data.pH !== undefined) {
138     phGauge(data.pH); document.getElementById('phValue').textContent = data.pH.toFixed(2);
139   }
140   if (data.turbidity !== null && data.turbidity !== undefined){
141     turbGauge(data.turbidity); document.getElementById('turbValue').textContent = data.turbidity.toFixed(1);
142   }
143   if (data.tds !== null && data.tds !== undefined){
144     tdsGauge(data.tds); document.getElementById('tdsValue').textContent = data.tds.toFixed(0);
145   }
146   if (data.orp !== null && data.orp !== undefined){
147     orpGauge(data.orp); document.getElementById('orpValue').textContent = data.orp.toFixed(0) + ' mV';
148   }
149   pushReading({
150     timestamp: data.timestamp || now,
151     pH: data.pH ?? null,
152     turbidity: data.turbidity ?? null,
153     tds: data.tds ?? null,
154     orp: data.orp ?? null
155   });

```

Frontend EJS code: Real-time data display, manual control page, graph generation logic.

Appendix C Arduino Code part 5

```

1 const express = require('express');
2 const http = require('http');
3 const socketio = require('socket.io');
4 const fs = require('fs');
5 const path = require('path');
6 const os = require('os');
7
8 // Optional: SerialPort (only if you want to read from USB serial)-
9
10 let SerialPort, ReadlineParser;
11 try {
12   SerialPort = require('serialport').SerialPort || require('serialport');
13   ReadlineParser = require('@serialport/parser-readline').ReadlineParser;
14 } catch (e) {
15   // serialport not installed or not available; server will still work but serial reading disabled
16   SerialPort = null;
17   ReadlineParser = null;
18   console.warn('serialport not available - serial input disabled (install @serialport if needed).');
19 }
20
21 const app = express();
22 const server = http.createServer(app);
23 const io = socketio(server);
24
25 // == USER-SPECIFIC HOST/PORT ==
26 const PORT = process.env.PORT || 3000;
27 const HOST = process.env.HOST || ''; // <- bind explicitly to your Mac IP
28 // =====
29
30 const MAX_STORE = 10000; // keep up to 10k rows in memory
31 const dataStore = []; // newest first
32
33 app.use(express.static(path.join(__dirname, 'public')));
34 app.use(express.json({ limit: '5mb' }));
35
36 // Helper to push entry (normalized) into store + emit
37 function pushEntry(entry) {
38   // ensure ISO timestamp string
39   const ts = entry.timestamp ? new Date(entry.timestamp).toISOString() : new Date().toISOString();
40   const row = {
41     timestamp: ts,
42     pH: (entry.pH !== undefined && entry.pH !== null) ? Number(entry.pH) : null,
43     turbidity: (entry.turbidity !== undefined && entry.turbidity !== null) ? Number(entry.turbidity) : null,
44     tds: (entry.tds !== undefined && entry.tds !== null) ? Number(entry.tds) : null,

```

Appendix C Arduino Code part 5

```

45 |   orp: (entry.orp !== undefined && entry.orp !== null) ? Number(entry.orp) : null
46 |   });
47 |   datastore.unshift(row);
48 |   if (datastore.length > MAX_STORE) datastore.pop();
49 |   io.emit('sensorData', row);
50 |   console.log('Stored ->', row);
51 |   return row;
52 | }
53 |
54 | // Endpoint ESP32 posts to (keeps compatibility with your existing /api/data)
55 | app.post('/api/data', (req, res) => {
56 |   const body = req.body || {};
57 |   const entry = pushEntry({
58 |     pH: body.pH,
59 |     turbidity: body.turbidity,
60 |     tds: body.tds,
61 |     orp: body.orp,
62 |     timestamp: body.timestamp || Date.now()
63 |   });
64 |   return res.status(200).json({ ok: true, received: entry });
65 | });
66 |
67 | // Provide history to clients (frontend fetchInitial expects an array)
68 | app.get('/data', (req, res) => {
69 |   // Return copy (most recent first)
70 |   res.json(datastore.slice(0, MAX_STORE));
71 | });
72 |
73 | > // Save endpoint: writes CSV on server. Accepts optional JSON body { data: [...] }--
74 |
75 | app.post('/save', (req, res) => {
76 |   let rows = [];
77 |
78 |   if (req.body && Array.isArray(req.body.data) && req.body.data.length > 0) {
79 |     rows = req.body.data.slice();
80 |   } else {
81 |     rows = datastore.slice().reverse(); // chronological oldest->newest
82 |   }
83 |
84 |   if (rows.length === 0) {
85 |     return res.status(400).json({ ok: false, message: 'No data to save' });

```

Appendix C Arduino Code part 6

```

86 |   }
87 |
88 |   let csv = 'timestamp,pH,turbidity,tds,orp\n';
89 |   rows.forEach(r => {
90 |     const ts = r.timestamp || '';
91 |     const ph = (r.pH !== null && r.pH !== undefined) ? r.pH : '';
92 |     const turb = (r.turbidity !== null && r.turbidity !== undefined) ? r.turbidity : '';
93 |     const tds = (r.tds !== null && r.tds !== undefined) ? r.tds : '';
94 |     const orp = (r.orp !== null && r.orp !== undefined) ? r.orp : '';
95 |     csv += `${ts},${ph},${turb},${tds},${orp}\n`;
96 |   });
97 |
98 |   const now = new Date();
99 |   const fname = `water_data_${now.getFullYear()}${('0'+(now.getMonth()+1)).slice(-2)}${('0'+now.getDate()).slice(-2)}_${('0'+now.getHours()).slice(-2)}${('0'+now.getMinutes()).slice(-2)}.csv`;
100 |   const outPath = path.join(__dirname, fname);
101 |
102 |   try {
103 |     fs.writeFileSync(outPath, csv, { encoding: 'utf8' });
104 |     console.log('Saved ${rows.length} rows to ${outPath}');
105 |     return res.status(200).json({ ok: true, file: fname, rows: rows.length });
106 |   } catch (err) {
107 |     console.error('Failed to write CSV', err);
108 |     return res.status(500).json({ ok: false, message: 'Failed to write file', error: err.message });
109 |   }
110 | });
111 |
112 | // optional: list saved CSV files in project root
113 | app.get('/saved-files', (req, res) => {
114 |   const files = fs.readdirSync(__dirname).filter(f => f.startsWith('water_data_') && f.endsWith('.csv'));
115 |   files.sort().reverse();
116 |   res.json(files);
117 | });
118 |
119 | // socket.io connection logging
120 | io.on('connection', (socket) => {
121 |   console.log('Client connected:', socket.id);
122 |   socket.on('disconnect', () => console.log('Client disconnected:', socket.id));
123 | });
124 |
125 | > /* =====
126 |
127 | async function findSerialPort() {

```

Appendix C Arduino Code part 7

```

132 async function findSerialPort() {
133   if (!SerialPort) return null;
134   const override = process.env.SERIAL_PORT || null;
135   if (override) return override;
136   try {
137     const ports = await (require('serialport').list());
138     const candidate = ports.find(p => {
139       const info = `${p.vendorId}|${p.productId}|${p.manufacturer}|${p.path}`.toLowerCase();
140       return info.includes('arduino') || info.includes('usb') || info.includes('tty') || info.includes('com');
141     });
142     return candidate ? candidate.path : (ports[0] ? ports[0].path : null);
143   } catch (e) {
144     console.warn('Serial port list error', e.message);
145     return null;
146   }
147 }
148
149 (async () => {
150   // Try to attach to serial port (non-blocking)
151   const portPath = await findSerialPort();
152   if (portPath && SerialPort && ReadlineParser) {
153     try {
154       const port = new SerialPort({ path: portPath, baudRate: Number(process.env.SERIAL_BAUD || 115200), autoOpen: true });
155       const parser = port.pipe(new ReadlineParser({ delimiter: '\n' }));
156       console.log('Serial port opened:', portPath);
157       parser.on('data', line => {
158         const parts = line.trim().split(',').map(s => s.trim()).filter(s => s.length > 0);
159         if (parts.length >= 3) {
160           const pH = parseFloat(parts[0]);
161           const turbidity = parseFloat(parts[1]);
162           const tds = parseFloat(parts[2]);
163           const orp = parts.length >= 4 ? parseFloat(parts[3]) : null;
164           pushEntry({ pH, turbidity, tds, orp, timestamp: Date.now() });
165         } else {
166           // optionally ignore or log malformed lines
167           // console.log('Serial bad line:', line);
168         }
169       });
170       parser.on('error', e => console.error('Parser error', e));
171       port.on('error', e => console.error('Serial port error', e));
172     } catch (e) {
173       console.warn('Failed to open serial port:', e.message);
174     }
175   }
176 }

```

Appendix C Arduino Code part 8

```

168   }
169   });
170   parser.on('error', e => console.error('Parser error', e));
171   port.on('error', e => console.error('Serial port error', e));
172 } catch (e) {
173   console.warn('Failed to open serial port:', e.message);
174 }
175 } else {
176   console.log('No serial port attached or serialport module missing. Set SERIAL_PORT env var to enable.');
```

```

177 }
178 }());
179
180 /* Start server */
181 function getLocalIPv4() {
182   const ifaces = os.networkInterfaces();
183   for (const name of Object.keys(ifaces)) {
184     for (const iface of ifaces[name]) {
185       if (iface.family === 'IPv4' && !iface.internal) {
186         return iface.address; // j: "192.168.1.126"
187       }
188     }
189   }
190   return '127.0.0.1';
191 }
192
193 server.listen(PORT, HOST, () => {
194   const localIP = getLocalIPv4();
195   console.log(`Server listening (bound host=${HOST}) port=${PORT}`);
196   console.log(`Reachable on: http://\${localIP}:\${PORT} (or http://localhost:\${PORT})`);
197 });
198

```

Appendix D – Website Preview

