

MSc Project Report

**School of Physics, Engineering and Computer Science
University of Hertfordshire**

WATER QUALITY MONITORING SYSTEM

Report by
SAYAJI PRAKASH SHIRKE

Supervisor
Dr. Georgios Pissanidis

Date
17/05/2021

DECLARATION STATEMENT

I certify that the work submitted is my own and that any material derived or quoted from the published or unpublished work of other persons has been duly acknowledged (ref. UPR AS14 Appendix III - version 14.1, sections 2 to 2.9– Section on cheating and plagiarism)

Student Full Name: SAYAJI PRAKASH SHIRKE

Student Registration Number: 18023684

Date: 17/05/2021

ABSTRACT

This project aims to develop a remote IoT-based device that determines water quality by using measurement within and against defined set parameters of water quality. The obtained water quality data have to be stored in a database for possible post-processing. The communication between these devices is established using MQTT over WiFi. MQTT is already integrated as a network protocol/broker within the ESP32 device and the inbuilt WiFi, both of which are essential components in facilitating the connection between the devices (e.g. the server and ESP32). Mosquitto is the broker who will implement the MQTT protocol and, therefore, enable establishing this connection, creating a UI to check the communication between devices. Node-RED will be implemented to create the server in the system, communicating with the Mosquitto.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank almighty God for the abundance of blessings He has showered on me in granting me the opportunity and privilege of pursuing my Master's degree at such a well-respected and resourceful institution. I extend my deepest love and gratitude to my dear parents, who have helped and supported me to become independent and make a better future for myself, having the courage to overcome any problems I have faced along the way. I also take immense pleasure in thanking my guide Professor Amit Pujari and the University for allowing me to choose this topic of great personal interest to myself, and also one that has the potential to make such a powerful impact on society, bettering it in helping us all to keep our water bodies safe and clean. I am very thankful to Dr Georgios Pissanidis for his constant support and guidance, always standing by me and helping me resolve all my doubts and tackle any issues with his wisdom and intelligence. Finally, I would also like to thank all of my friends who have helped me, both directly and indirectly, to complete my project and achieve this task.

Table of Contents

Table of Contents.....	1
LIST OF FIGURES	Error! Bookmark not defined.
List of Tables	3
GLOSSARY	4
1 Introduction.....	1
1.1 Feasibility	1
1.2 Project Aim and Objectives.....	3
2 Literature Review.....	5
3 Project Work	10
3.1 System Overview	10
3.2 Experimental Setup	11
3.2.1 Hardware Setup.....	11
3.2.2 IDE Software Setup	12
3.3 Sensor Software Development	12
3.4 Connection Details	15
3.5 Data Handling.....	17
4 Results.....	19
5 Conclusion and Discussion	21
5.1 Future Scope	21
6 References.....	22
7 APPENDICES	23

List of Figures

Figure 1 System Overview.....	11
Figure 2 Power Supply.....	14
Figure 3 Sensors Connection	15
Figure 4 Block Diagram Data Flow	16
Figure 5 Data Flow in Database.....	17
Figure 6 Data Flow-chart	18
Figure 7 MQTT Connection	19
Figure 8 Node-red Flow	19
Figure 9 Node-red Output with Dial	20

List of Tables

Table 1. 1 Cost of Hardware	1
-----------------------------------	---

GLOSSARY

- **ADC:-** Analog to Digital Converter
- **AI:-** Artificial Intelligence
- **DAC:-** Digital to Analog Converter
- **DC:-** Direct Current
- **EH:-** Energy Harvesting
- **GPRS:-** General Packet Radio Service
- **GSM:-** Global System for Mobile Communication
- **I2C:-** Inter-Integrated Circuit
- **ide:-** Integrated development environment
- **IoT:-** Internet of Things
- **MBps:-** Megabytes per Second
- **MQTT:-** Message Queuing telemetry transport
- **pH:-** potential of Hydrogen
- **RFID:-** Radio Frequency Identification
- **RS232:-** Recommended Standard 232
- **SD:-** Secure Digital
- **SMS:-** Short Message Service
- **SPI:-** Serial Peripheral Interface
- **SQL:-** Structured Query Language
- **UI:-** User Interface
- **URL:-** Uniform Resource Locator
- **USB:-** Universal Serial Bus
- **WSN:-** Wireless Sensor Network

1 Introduction

All living life forms on Earth depend, undeniably, in one way or another, on the water to sustain their existences, which is the primary reason for which ensures that it is of acceptable quality is a matter of such crucial importance. As intellectual beings with the capacity to endeavour to solve such problems, our moral responsibility is to ensure that a high quality of water is maintained for healthy consumption by all living organisms. The exponential rate at which industrialisation has taken and is still taking place across the globe has brought with it the ever-increasing threat of water pollution, which is becoming more and more damaging to the health of our precious ecosystems and the organisms which constitute them. The brief discussion of industrialisation here highlights just one example in a vast range of other contributing socio-economic factors and influences that collectively are behind an overall deterioration in global water quality by introducing harmful contaminants. In this way, more must be done to combat the decline in water quality in the fight against its pollution.

Despite the best efforts by national governments and international organisations to prevent and deter contamination with the imposition of various laws and measures, and strategies, monitoring remains an indispensable tool in addressing the fact that these mitigative interventions cannot entirely eliminate the infiltration of all contaminants into every water source. However, the act of monitoring water quality brings with it various practical challenges, such as the difficulty of getting to bodies of water in both remote and geographically inaccessible locations, as well as those enveloped in political conflicts. This is the context in which my project finds its purpose, seeking to produce a device that can be deployed into water bodies to record water quality measurements, which can later be retrieved via an Internet connection. These measurements are taken across several different parameters to collect data on the corresponding qualities of the water, which are later used to make an overall assessment of its cleanliness by transfer to a database where it is analysed.

1.1 Feasibility

To fulfil its intended objective of being deployed as a remote in-water monitoring device, the proposed device needs to float on the surface of water successfully. Therefore, a solid and lightweight device design ensures that it can do so whilst also securely encompassing the components it needs to function. In consideration of the importance of these design features to the efficiency of the device's performance in accomplishing its purpose, acrylic has been chosen as the ideal material to develop its body, given its well-renowned properties of high durability, ability to withstand a broad range of different temperatures and also its weather resistance, which facilitates its operation in the diverse weather conditions which it will have to endure in a range of distinct environments and climates.

In terms of how the device is powered, a lithium-ion battery serves as its source, which is charged itself by solar power, as one of how the device's design incorporates an environmentally friendly feature, in alignment with the eco-conscious motives which inspired its creation. Charging the battery via connection to a solar panel also means that the margin for human error in not manually charging it, thereby rendering the device unworkable, is significantly reduced, therefore increasing its potential productivity. The lithium-ion battery is also a very cost-effective and readily available choice, which singularities it as an ideal power source for this device, able to satisfy its power demand, as well as that of the other sensors connected to it.

The use of MQTT establishes the connection between the device (microcontroller). It is for this reason that ESP32 is the optimal choice of microcontroller, as it has the inbuilt WiFi and compatibility with MQTT necessary to make possible this communication between the device and the database created to store the data.

The different parameters intended to serve as the determining indicators of water quality in this device is measured by the microcontroller, turbidity, temperature, and pH sensors of which it is composed. All of these electrical components are both cost-effective and readily available in the market meaning that it is relatively easy to source them, mainly online. These parts are also compatible for use in conjunction with one another and can be connected easily using a breadboard. Below is a table in which the estimated cost split based on the most competitive prices available for the purchase of each component required is represented:

Table 1. 1 Cost of Hardware

Material	Cost in GBP (£)
Acrylic sheet	£24.00
Temperature Sensor	£3.60
pH Sensor	£29.30
Turbidity Sensor	£9.072
ESP32	£10.00
Solar Panel	£10.00
Total:	£85.972

At a total material cost of £85.972 per unit, the device is highly cost-efficient to produce. This cost-effectiveness of production will be increased further due to the economies of scale generated upon mass production. Such mass production can be considered feasible to realise given the potential the product has to appeal to a huge market, ranging from national governments to private sector water industry involved companies who all share the all-encompassing stakeholder interest of the maintenance

of a specific, desired quality of water. It is this universality of the product's appeal that allows it to attract such a wide-ranging interest, as well as the relevance of its function, which is only set to increase in consideration of the high impact of the intersection of current and future social, economic and environmental issues on the water.

1.2 Project Aim and Objectives

The accomplishment of this project is not only a challenging task but a complex one too, as it is made up of many different stages, equally demanding in their rights. Splitting these stages into smaller, more specific sub-tasks is therefore a more suitable approach to take with regards to a more efficient method to tackle its management. These task objectives are as follows below:

The first step in laying the foundations to develop this device is to write the codes for the pH, temperature, and turbidity sensors. Once the codes for each sensor have been written, they need to be combined into a single code to facilitate simultaneous data obtention of the different parameters when the device is in operation. This allows for this data to be sent to the database as coherent datasets, which can later be more efficiently analysed to assess the overall quality of the water at a specific point in time, or over a while. The data recorded for each parameter by its corresponding sensor, however, is still (Pitt, et al., 1975) (Anon., 2019) (Anon., 2019) accessible as independent measurements, given that the functions of their respective individual codes are maintained.

The next step is to establish the connection between the device and the server using the MQTT. This communication network is set up by the use of Mosquitto and the I2C and SPI communication protocols integrated into the ESP32 microcontroller. The main difficulty which is encountered here is understanding the Node-RED and MQTT communication methods.

The coding for the device is written using Arduino IDE, where the time interval between successive datasets being transferred to the database is determined to control the quantity of data being transmitted at any one time. This coding also allows for the size of the database to be controlled. Furthermore, the coding for the communication between the device and the server is also written using the same software, which fulfils the function of the 'brain' of the device.

The last main objective concerns database handling, where the quantities, types of data, and the periods of time for which they have stored all need to be taken into careful consideration. This is of critical importance as not creating files of such a size with the potential to crash the system, but at the same time, retain a database of an appropriate size, which has the necessary scope to be used by scientists in future analysis a historical

record. Maximising data efficiency to strike a balance between meeting the two aims above is a crucial goal at this stage.

2 Literature Review

In spite of the abundance of water on Earth, with around 71% of its surface covered with the life-sustaining liquid (Anon., 8), only around 3% is actually available for consumption. Take into consideration on top of that the threat of contamination by industrial pollution, over-abstraction for commercial use, as well as climate change to name but a few, and it becomes clear that the threat faced to both the quantity and quality of these already limited reserves is severe. Therefore, it is unsurprising that preserving these precious water resources has rocketed in its importance as a priority, not only in the agenda of national governments and international organisations but also for individuals worldwide preoccupied with the potential consequences of such an impactful environmental affair. Given the aforementioned socio-economic and environmental risks to which water resources across the globe find themselves exposed without exception, amongst many others, maintaining the equilibrium between the retention of a sufficient and acceptable quality of water supply and its ever-increasing consumption, is one of the key reasons as to why its monitoring is of such prominence. The concept of water security, as defined by UN-Water (Anon., 2019), in which this consideration of both the quantity and quality of water available hold importance, is also a significant influence in the pursuit to attain equality in securing this basic necessity for people in diverse socio-economic, geographical and political environments across the globe (Anon., 2019).

In the case of those disadvantaged countries which lack access to their own freshwater supply, such as Yemen and Djibouti (Anon., 2018), the water which they can use has to undergo a significant process of refinement before it can even be deemed suitable, including for non-drinking purposes. This highlights the increased importance of ensuring that these critical stores of water are protected, which aims to be achieved by this type of stringent monitoring. In the already tense and overburdened climate which enshrouds the world today, facing issues including the Covid-19 pandemic, a global economic recession, and increasing geopolitical conflict, it is the moral responsibility of engineers, such as myself, to help lessen the cruelty of the water crisis in their technological contributions. (Bakker, 2012)

With the rapid advance of technology, IoT has emerged as one of the significant leading technologies in the world of engineering. Varsha discusses the potential impact that such an innovative technology could have for the evolution of the smart water system, where the decline in water quality in the distribution system is a factor which affects both biological and non-biological contents, and thereby damage to the ecosystem as a whole (e.g., changes in water colour and odour). Rooted in a combination of the different fields of electronics, software development, and communication networking, the collaboration of such a diverse variety of distinct domains guarantees the successful development of a relevant IoT device in this context. Wireless Sensor Network (WSN),

Radio Frequency Identification (RFID), Energy Harvesting (EH) and Artificial Intelligence (AI) more specifically, all make up the different components which work together to achieve IoT development. This takes place on three different layers: the Perception, Network and User Interface (UI) Layers. In the context of water monitoring, this developmental model can be observed in the following examples: the collection of data from different sources or sensors on the Perception layer, the establishment of the connection between the cloud, monitoring device, and UI on the Network and UI layers. There are a variety of different means through which connections can be established using GSM or GPRS protocol, with the help of 3G and 4G technology. The fourth and final layer on which IoT development can be done, the Application layer, can perhaps be considered the most intuitive, as this is the stage in which the data and computing parts become visible and available for human interface.

With regards to finding a solution to tackle the issue of monitoring water quality, many writers are unanimous in their proposal of an IoT based water parameter monitoring device, which is based on three major parameters, specifically: quality factor, quantity factor and technological doctor. The quality factor would seek to assess the acidity or alkalinity of the water, determined using its pH, and also the level of dissolved oxygen, as well as its turbidity to establish the degree of opacity/cloudiness of the water. Similarly, the quantity factor uses the level of pressure of flow of the water, as measured by ultrasonic sensors, as the basis of its judgement. The final factor, the technological doctor, involves a selection of OS stations, data fusion and forecasting to provide overall information pertaining to overall water quality from any emergent or urgencies in the decision-making of an essential smart water system.

Author Anuradha is amongst those who have suggested an IoT-based, low-cost system to monitor data quality using real-time data to do so. The emphasis of this research is on the accuracy of the analysis of the water quality intended to be achieved by use of this efficient, cost-effective and dynamic system in real-time. The proposed methodology of the study utilises Raspberry Pi and multiple sensors to measure total dissolved solids (TDS), temperature, turbidity, and the pH value of the water. The author also plans to use open-source IoT applications and APIs to store and retrieve the data, using HTTP protocols over the Internet, or LAN. Such a system is built on various IoT applications, including that of ThingSpeak API, for instance, the tool which acts as the gateway amongst the various touchpoints of the system. The use of this platform has successfully facilitated the creation of sensor logging with location-tracking applications. (Pitt, et al., 1975)

The paper also sets out to explore the sensor-based measurement of physical and chemical parameters of water in its discussion. The sensors being employed in the research are as follows: pH sensor (SKU: SEN0161), turbidity sensor (SEN0189), temperature sensor (DS18B20), and a TDS probe compatible with IoT. The values

collected from these sensors for pH value, turbidity, temperature, and total dissolved solids counts respectively, are further intended to be processed by the controller, assisted by the Internet and tools mentioned above. One efficient option of a microcontroller with incorporated Ethernet is that of the Raspberry Pi, which has been used as a core controller in this study. The performance of the system has been assessed taking into consideration its success in both commercial and domestic settings, where the authors have concluded that it demonstrates a high level of performance, accuracy and robustness in its water quality analysis by use of real-time data. (ChungJae & Yoo, 2015)

The focus of a second paper investigated, on a water quality monitoring system using IoT by Ashwini Doni, promotes the choice of pH, turbidity and temperature sensors as parameters for the monitoring. The server module consists of an Internet-enabled PC. Data is transferred to the server using GPRS, where the IP address of the localhost can access this server. A key aspect of this paper's aim is its focus on the development of a water quality monitoring system over a web server, in which the results obtained from the data sensing modules are stored, and then accessed with an IP address. The setup of this device as a whole advocates communication using the server and GPRS technology. (Radhakrishnan & Wu, 2018)

In the past, there have been a diverse variety of different approaches taken in the attempt to create this device. One such example which embodies a very unique in the approach it adopted, is that of Wan-Young Chung's development of a remote water-based monitoring system, in which he connected the different nodes of the device, and then used a base station with a computer to access the data. A pH sensor, dissolved oxygen level checker, electrical conductivity and turbidity sensors were deployed to gauge the quality of the water. The devices were placed in different locations, with an RS-232 protocol used to connect the server with the PC. They were also connected to a network of cables. JavaScript was then used to represent the data collected in the form of a graph. (Pitt, et al., 1975)

In April 2019, Demetillo and his team also engineered a device to monitor and calculate water quality, which utilised a solar-powered battery source as its source of power, as I also propose to do. The data was transmitted by means of a GSM module. A SIM card with Internet connectivity provided the connection to the database. A SIM card was also used to send SMS messages containing the computed data, whilst an SD card was added to the device, to store this data as a database in SQL. The data obtained from the different sensors of the device was then published in the form of a graph on its dedicated website. (Demetillo, et al., 2019)

Given the vast range of distinctive methodologies which have been adopted in the pursuit of the development of different devices which share the same common general

aim summarised here, it is evident that the advance of technology has brought with it a trend of evolution in their designs. Supported by the breadth of research which increasingly urges the growing necessity of the advancement of these types of devices, the new microcontrollers and different types of modems available in the era of accelerating technology inspires new hope of further progress in this field. (Pitt, et al., 1975)

The main focus of this project is the electronic development of the proposed device in terms of increasing the efficiency of the electronic workings of similar future devices in a broader sense to fulfil the requirements of the device's function in the most optimum way possible. ESP32 has been chosen as the microcontroller of this device, playing the role of its 'brain', for the six critical reasons listed below:

1. WiFi

ESP 32 comes included with integrated WiFi that transmits data at a rate of 150.0 Mbps. It can perform as a standalone system or as a slave device to a host MCU. This will facilitate the establishment of the connection between the device and the server. It works on its communication channel through its SPI/ SDIO or I²C.**Invalid source specified.**

2. Processor

ESP 32 has a Dual-Core 32-bit LX6 Tensilica Xtensa processor, which can process data at a rate of 160-240 MHz. It is highly integrated with in-built antenna switches, RF balun, power amplifier, filters and power management modules. ESP32 adds priceless functionality and versatility to the applications with minimal printed circuit board requirements.**Invalid source specified.**

3. Input and Output

It has an inbuilt Analog to Digital Converter (ADC) and Digital to analog converter(DAC). There are 18 pins for ADC and 2 for DAC. 10 pins can be used for data sensing through the sensors. It has a rich peripheral interface with DMA that includes capacitive touch; it has CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I²S (Integrated Inter-IC Sound). This wide range of input and output pins allows a flexible and easy communication of data in and out without any barrier**Invalid source specified..**

4. Arduino IDE Compatibility

It can be programmed using Arduino IDE, which is free and readily available software and can be easily downloaded to fulfil this purpose at no additional cost. The coding and operations of ESP32 are easy to perform. The board manager tool in the IDE changes to ESP32 and its models by adding the ESP32 boards to the system **Invalid source specified..**

5. MQTT Communication Channel

It supports MQTT, the publish-subscribe network protocol that will be used to transport messages between devices. It is lightweight and efficient, which have a small message header to optimise network bandwidth. It has Bi-directional communications, which allows messaging between the device to the cloud and vice-versa. This can be used to connect to plenty of devices. For different works, MQTT uses different tools depending on the users choice of the web, mobile platforms, desktop tools, command-line tools, commercial applications, desktop notification tools, gateways and Misc **Invalid source specified..**

3 Project Work

3.1 System Overview

The temperature, turbidity and pH sensors in use here to take readings for the stipulated parameters is connected to ESP32 via a breadboard by the use of jump wires. It is of vital importance that these sensors are connected in such a way that the length of the wires can be varied depending on the distance between the ports on the breadboard and ESP32 through which the sensors are connected. It is also crucial that whilst connecting the device with the sensors, the current flowing out of ESP32 to the sensors doesn't overload and therefore damage them. In order to prevent such damage from occurring, the datasheet for each sensor should be referenced carefully to ascertain the (maximum) current flow (permitted) to/of each sensor. Resistors should then be added to the breadboard in accordance with the respective current intakes of each sensor, which helps to maintain the durability and longevity of the performance of these sensors, thereby contributing to the overall longevity of the device's function.

The current flowing in the device as a whole constitutes a second, major concern, which also needs to be addressed. A rechargeable lithium-ion battery has been selected as the power source of this device, the Adafruit Universal USB/DC/Solar Lithium Ion/Polymer Charger - bq24074, which can boost up to 5V of current into ESP32 in accordance with the device's requirements. This power setup comes included with the solar panel responsible for charging the lithium-ion battery by placement on top of the device to ensure its continuous operation. A current stabiliser has also been incorporated as part of the battery setup to ensure that the current flowing from the battery to ESP32 remains constant and therefore doesn't run the risk of damaging the setup.

The third and final element which completes the overview of the system as a whole, is that of the network connectivity of the device with the server. This is established wirelessly by ESP32 (with the server) through MQTT using its inbuilt WiFi domain. MQTT publishes and subscribes the messages from one node to the other. Setting up this network by using MQTT circumvents the need to involve additional intermediate hardware, like GSM modules, to establish the connection to the cloud. Node-RED is the tool on the Node.js platform used to build the framework that enables the connection of MQTT. It aims to do so by constructing and defining the network connectivity between the nodes, ESP32 and the database in this context. The nodes simply need to be dragged and dropped into the framework, and then the connection between them can be defined, as well as the parameters of the nodes in the framework described. The definition of this connection between the nodes is carried out using the MQTT Explorer tool, where it does so by providing a server host ID, which effectively describes the parameters for connection. It is also used to describe the connection links and setup

between the server (nodes) and the database. With the nodes and its parameters correctly defined, the path of the data flow will have been successfully set up. The publishing of this connection is facilitated by use of this modem, where the portal requests the details of the network used to be added, such as the URL, username, password, and port number. Once these details have been published in the network, the connection can then be established. In this portal it is also possible to adjust and set the details of the connection, as well as that of any other parameters or details which are desired to be established in the network. Messages on and from ESP32 can be both communicated and published upon completing the setup and connection.

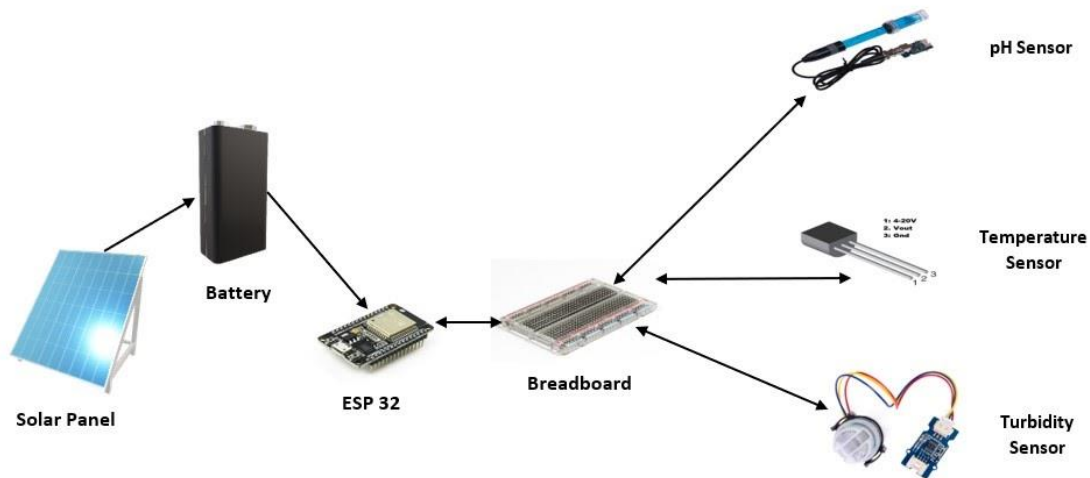


Figure 1 System Overview

3.2 Experimental Setup

3.2.1 Hardware Setup

The first step in building the hardware setup of this device involves the wiring/engineering of the connection between ESP32 and the device's lithium-ion battery which serves as its power source. The microcontroller is responsible for powering the device's sensors, which individually consume between 2-3V of energy, whilst it requires 5V of electrical energy to sustain its own functioning. For this reason, it is critical that the battery can provide sufficient current to support the operation of the device as a whole singular and comprehensive unit. The jumper cables which adjoin the different components of the circuit, through which they are powered, must be connected to a legitimate power pin to access their power supply, either VIN pin or 3.3V, and GND to manage the removal of any excess voltage. (Anon., 2021) Doing so enables an adequate current flow which is capable of powering the device's sensors. In the microcontroller there are 18 ADC and 2 DAC pins, all of which can be used for input and output of the device. Of these 18 ADC pins, 2 contain I2C, and 3 SPI

interface, which provides the opportunity of a greater flexibility in terms of connections that are possible with ESP32 and other devices. In this device there are 3 different sensors that are involved, none of which pose any issues in terms of their connections, which is especially true in the case of the temperature sensor, as its output can be formatted in either analogue or digital form, both of which can be easily read by the microcontroller. The connection of these sensors must be done very carefully as the pins that the sensors are being connected with need to be updated or specified in the code to trigger the sensor's task.

3.2.2 IDE Software Setup

The coding for the microcontroller is written using the Arduino IDE software, which can be installed in the system directly from the Arduino website. The Arduino IDE software then needs to be updated with the dependencies and libraries required for the ESP32 microcontroller in use here. Once the necessary dependencies and libraries have been imported, the setting of the board can be changed to ESP32's details, as opposed to the default Arduino details, in order to support its corresponding required libraries. The establishment of the connection between the physical board and the software code is fostered by using the Mosquitto broker. Mosquitto and MQTT Explorer are the main methods by which communication is established. These open-source software are available to be downloaded from the MQTT portals. The following prerequisite software dependencies must have already been installed in the system in order to support the main software and their functions:

1. Git
2. Mercurial
3. Mbed CLI
4. Node.js

The download of Node.js is the first step required to install Node-RED, given that it is one of its tools. To enable the installation of Node.js, the importance of the software dependencies in facilitating the operation of the leading software to engineer the software development of this device. Git, Mercurial and Mbed CLI are the main dependencies used to create the necessary environment to capacitate the functioning of this software.

3.3 Sensor Software Development

The coding for the sensors comprises one of the most pivotal tasks with regards to the scope of the software development of this project. In order to write the coding for each sensor, it is of vital importance that the necessary corresponding libraries required for the specific sensor in question are added in the header of the code.

With respect to the turbidity sensor, once submerged in water, it is triggered to shoot a beam of light in that water, which will then scatter due to the presence of particles. The

ejected beam is then re-received by the sensor's beam receiver, which determines the extent to which its particles have been scattered. The more the light scatters, the higher the concentration of particles in the water, which is typically indicative of water in which particles in addition to those which constitute the water itself are present and can therefore potentially denote a lower quality/purity of water. In the coding section for this sensor, the appropriate header file, `liquidcrystal_I2C.h`, needs to be first downloaded and added in the Arduino IDE, before it can be added in the header section of the code in order to enable the correct functioning of the sensor and thereby the generation of reading for the turbidity measurements of the water. The turbidity sensor is also small in size, which contributes to the lightweight and buoyant nature of the device.

The pH sensor measures the hydrogen-ion activity in the water, which provides an indication as to the acidity or alkalinity of the water and is expressed as its pH value - another one of the device's fundamental determinants of water quality in this project. In order to compute the pH reading/output generated by this sensor, the Nernst equation, as given below, is employed to obtain the actual pH value:

Equation 3.1

$$E = E_0 - 2.3\left(\frac{RT}{nF}\right)\ln Q$$

where,

Q= Reaction coefficient, Q

E = mV output from the electrode

E0 = Zero offset for the electrode

R = Ideal gas constant= 8.314 J/mol-K

T = Temperature in °K

F = Faraday constant = 95,484.56 C/mol

N = Ionic Charge, C

The computation of this actual pH value of the water, ascertained by use of the above equation, takes place in the background, embedded in the code itself, meaning that only the final value is received by the database, with no traces of the calculation that had been done to achieve it being brought with it. No additional header files and/or libraries other than `liquidcrystal_I2C.h` need to be added in the header of the code for the pH sensor, as this header file includes all the necessary base libraries imperative to facilitate its operation.

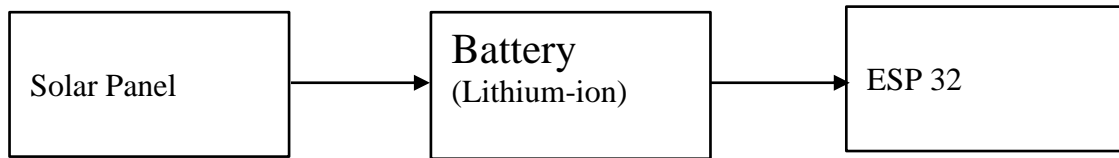


Figure 2 Power Supply

The final sensor incorporated as part of the design and function of this device is that of the temperature sensor, which is the easiest of the three to handle and work with. As indicated by its name, the temperature sensor takes a reading of the temperature of the water in which it is situated, which can warn of issues with water quality. Such issues include the risk of an increased surface evaporation rate that an increased water temperature causes, and therefore a potentially dwindling supply of water available from that resource. The sudden trend of an increasing water temperature can also be a symptom of potential contamination by industrial waste, which would dramatically impact the decline of the water's quality. The temperature sensor which has been chosen to be used in this device is a sensor manufactured by Dallas. The company provides its own special, dedicated libraries to accompany the product used in the coding for this sensor. Adding `dallastemperature.h` in header files enables the addition of the dependencies and the libraries needed for the sensors to work properly, as has been done so far for the other sensors.

With all sensors coded for, enabled and working correctly, the device as a whole is in working condition and ready to be operated. However, it is important to bear in mind the importance of ensuring that no dependencies are maintained between the different functions of the individual codes of each sensor, whilst coding, as every sensor's functionality needs to be defined individually and independent of the other sensors. Apart from the header files and libraries necessitated for the coding of the device's sensors, further additional libraries with their headers also need to be added, in order to fully configure the software setup with the tools necessary.

The libraries for ESP32, from the Arduino IDE libraries, still need to be installed, which supplies the basic features of the ESP32, which can then start to be worked with. After the installation of the ESP32's libraries has been completed, it is possible to add other necessary libraries from the Arduino IDE's library manager feature. This addition of ESP32's library now enables it to be added to the header section of the code as `ESP32.h`, thereby permitting access to all of those features in the code. Once these features have been added to the code, other, additional features (libraries?) which are essential to the code, such as `WiFi.h`, which permits WiFi access, can also be added. This specific library, `WiFi.h`, allows for the initiation and facilitation of all the functions which are required for WiFi to connect to and access the server, from which point the communication between the devices can be set up in turn. `publicclient.h` is another

principal library which needs to be added to the main framework, because of the role it plays as a prerequisite for the successful and final formulation of the connection to the server. This, in essence, inaugurates (sets up) the communication with the broker and the client, and also/thereby allows the server's IP address to be put into the framework. DHT.h is a further important library, which empowers the functioning of the temperature and humidity sensors in conjunction with one another. This brings to a close the discussion of the basic libraries required to bring the fundamental features into the framework.

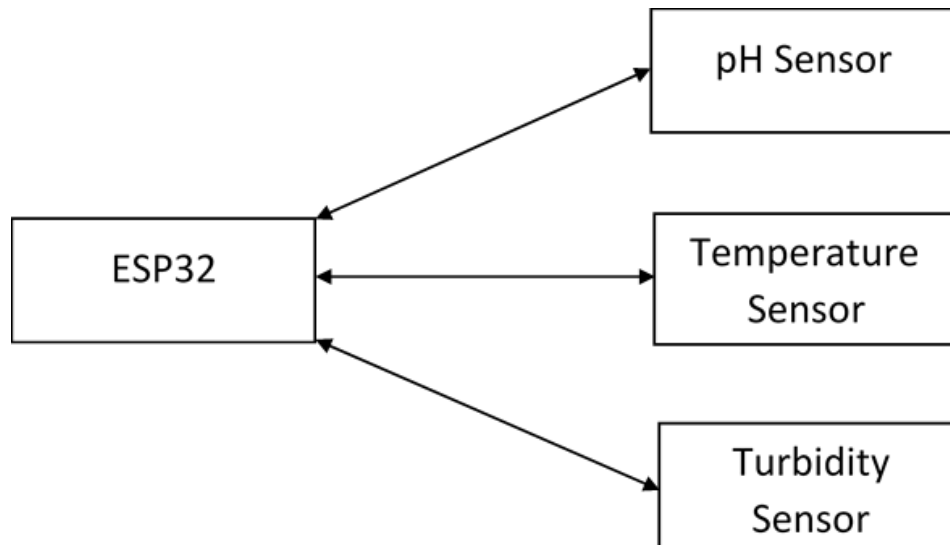


Figure 3 Sensors Connection

All of the codings are present in Arduino IDE, where it is possible to modify and update any code in the microcontroller. With the inclusion of all the necessary header files and libraries in the framework, all the functionalities have been added. The next stage in the process is that of the institution of the setup for the communication between ESP32 and the server. Here, the description of the port and the call for the connection link are established. If the connection is done via use of the port name and the password, it needs to be declared and called here to ensure that the code can enter the secure network. Following the successful establishment of the connection between the server and the node, the subsequent step comprises the computation of the sensors. Whilst coding for the sensors at this stage, it is vital to ensure that the variables don't overlap with one another, in order to assure that proper dataflow can take place. After each sensor's code has been written, it needs to be programmed to trigger the data collected to be sent to the database at specific, determined time intervals, which guarantees a constant data flow in the system. In order to successfully repeat this trigger at the chosen time intervals, the delay function in the code is used. Time also has to be added to this function, so that the code is aware of when exactly to prompt the data to be sent, i.e., how long it should delay for until doing so.

3.4 Connection Details

The establishment of the connection between two devices is set up by use of an MQTT Client, MQTT Explorer, and its corresponding server, the latter of which is a necessary component in the institution of this connection. Node.js, the platform on which the tool Node-RED is built, is used here to create a local server for and within the system with/in the form of an IP address. Once this IP address has been generated, MQTT Explorer can fulfil its intended function of managing the server's connection, more specifically manipulating the type of connection and the access of details like username and password. The name of the server and its details with regards to its connectivity and ports, amongst other such technicalities, can also be adjusted. Following the initial setting up of this connection, its description can then take place, the process of doing so a crucial step in enabling the flow of data. This connection is described through the use of the interface of Node-RED, which provides the means by which a user is able to drag and drop different nodes in accordance with their personal requirements, and also define the connection of each distinct and individual node as per the desired/required flow of data. Node-RED also affords the opportunity of the management of the different parameters of the different variables/combinations of nodes in this setup in its use. It is also fundamental in the role it plays in this project in implementing the connection between ESP32 and the server.

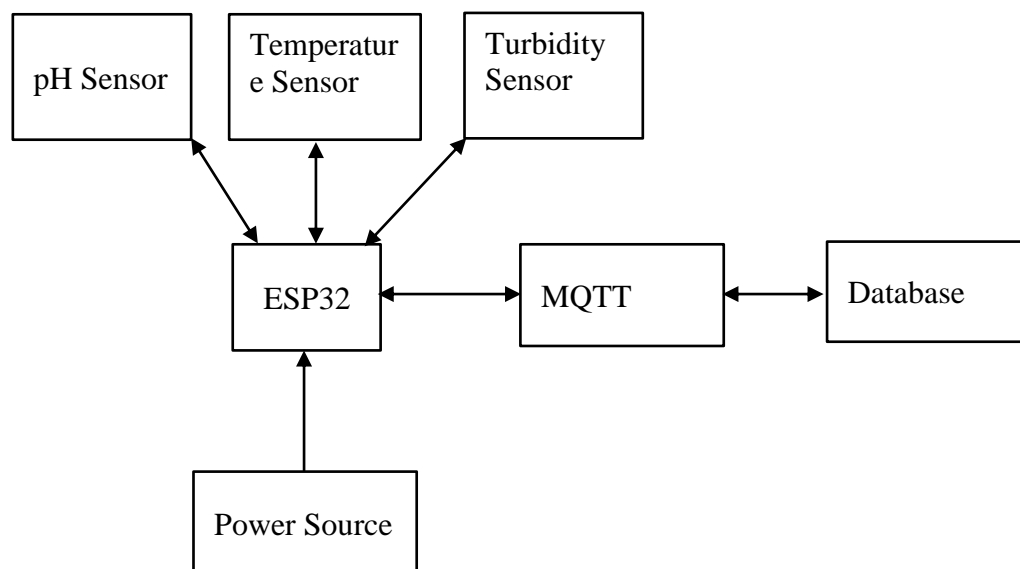


Figure 4 Block Diagram Data Flow

3.5 Data Handling

With the connection between the server and ESP32 established, and the sensors connected with the microcontroller (ESP32), the data which these sensors collect, obtained from the real-time scenario in which the device designed performs its function, can then be transferred to the database. This data consists of the temperature, pH value and turbidity readings as measured by the device in action and compiled in order to later provide and substantiate an assessment of the quality of the water in that specific environment from which it was collected. However, the obtention of this data from the device at regular and constant intervals of a high frequency over an extended period requires a storage space of an increasingly high volume to support its continued retention. Such a large influx of data coming into the database system from multiple devices and at such a high frequency (every 30 minutes from each operating device) poses the risk of an overflow of data in the server. Furthermore, the availability of a sizable server is necessary to safely accommodate this high volume of data that also needs to be retained over a prolonged period of time (forever), in itself entailing expensive additional ongoing costs. For this reason, exclusively the minimum and maximum readings collected each day for each determining variable being measured by the sensors are saved in the database, with the rest of the data being eliminated, on a weekly basis.

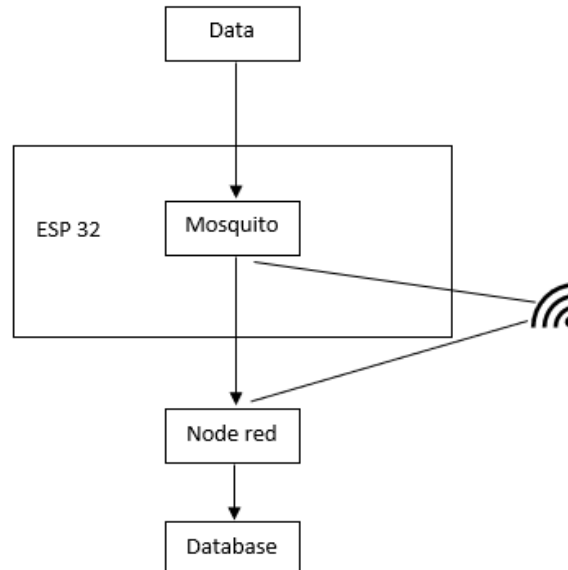


Figure 5 Data Flow in Database

Those daily minimum and maximum values are then be refined further, with just the lowest and highest for the corresponding week retained at the end of each month. Following such a rigorous and systematic process of data, refinement ensures that it is managed in the most efficient way, with the most scientifically relevant and

representative samples being kept for analysis, whilst overcoming the necessity to upgrade the size of the server by keeping the data being stored in it to an appropriate minimum.

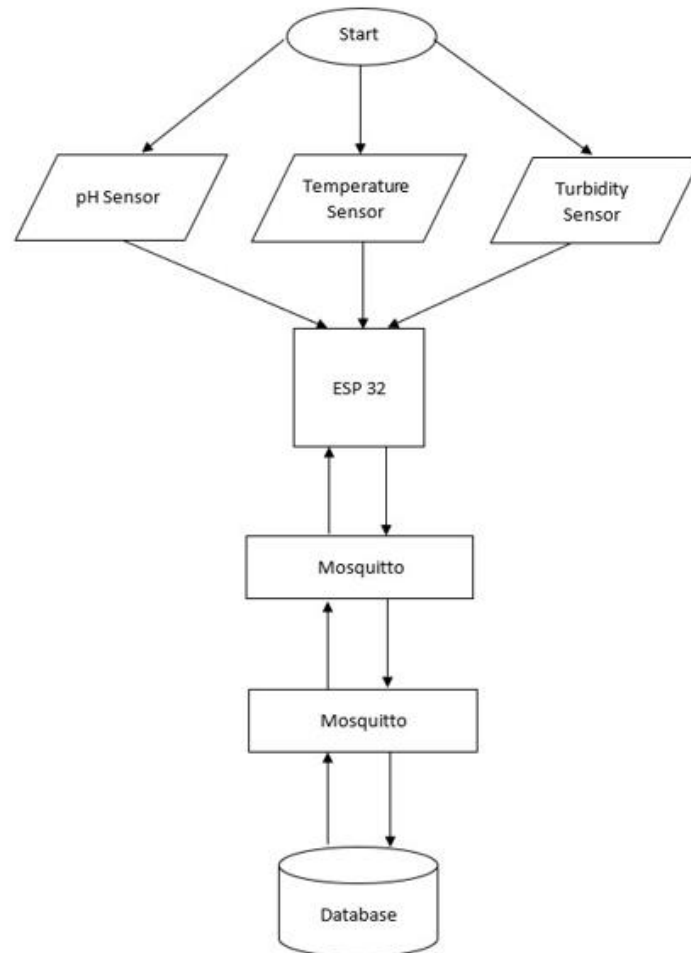
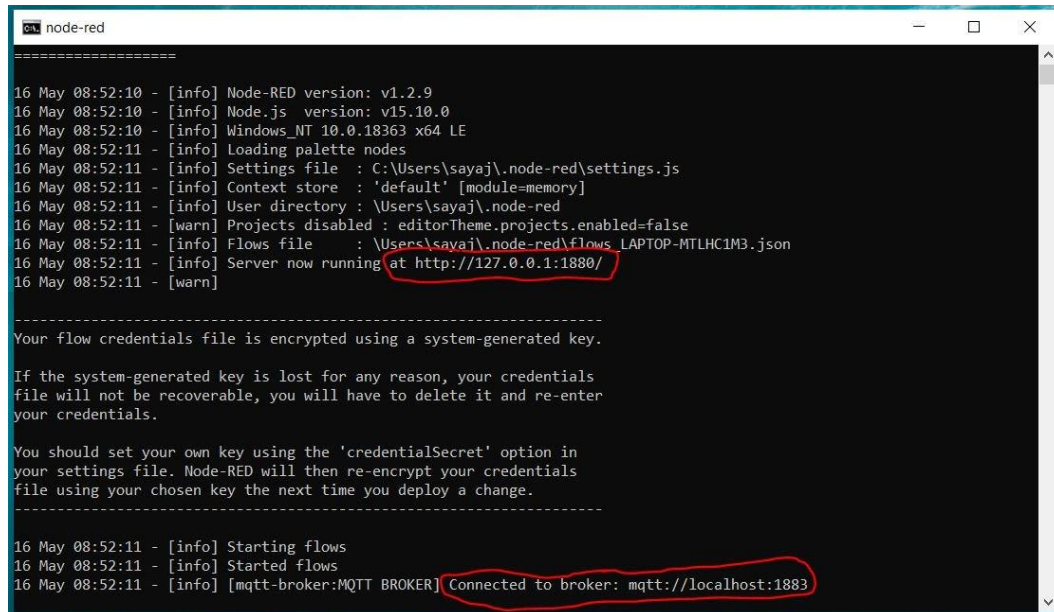


Figure 6 Data Flow-chart

4 Results

The following image displays the server connection which has been established in our localhost. The first red circle shows the server's IP address, which will establish a primary connection. Using the same IP address, the connection in MQTT has been successfully established, and communication between the nodes and the payload can be achieved.



```
node-red
-----
16 May 08:52:10 - [info] Node-RED version: v1.2.9
16 May 08:52:10 - [info] Node.js version: v15.10.0
16 May 08:52:10 - [info] Windows_NT 10.0.18363 x64 LE
16 May 08:52:11 - [info] Loading palette nodes
16 May 08:52:11 - [info] Settings file : C:\Users\sayaj\.node-red\settings.js
16 May 08:52:11 - [info] Context store : 'default' [module=memory]
16 May 08:52:11 - [info] User directory : \Users\sayaj\.node-red
16 May 08:52:11 - [warn] Projects disabled : editorTheme.projects.enabled=false
16 May 08:52:11 - [info] Flows file : \Users\sayaj\.node-red\flows_LAPTOP-MTLHC1M3.json
16 May 08:52:11 - [info] Server now running at http://127.0.0.1:1880/
16 May 08:52:11 - [warn]

-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
16 May 08:52:11 - [info] Starting flows
16 May 08:52:11 - [info] Started flows
16 May 08:52:11 - [info] [mqtt-broker:MQTT BROKER] Connected to broker: mqtt://localhost:1883
```

Figure 7 MQTT Connection

Connection explanation of the node-red flow

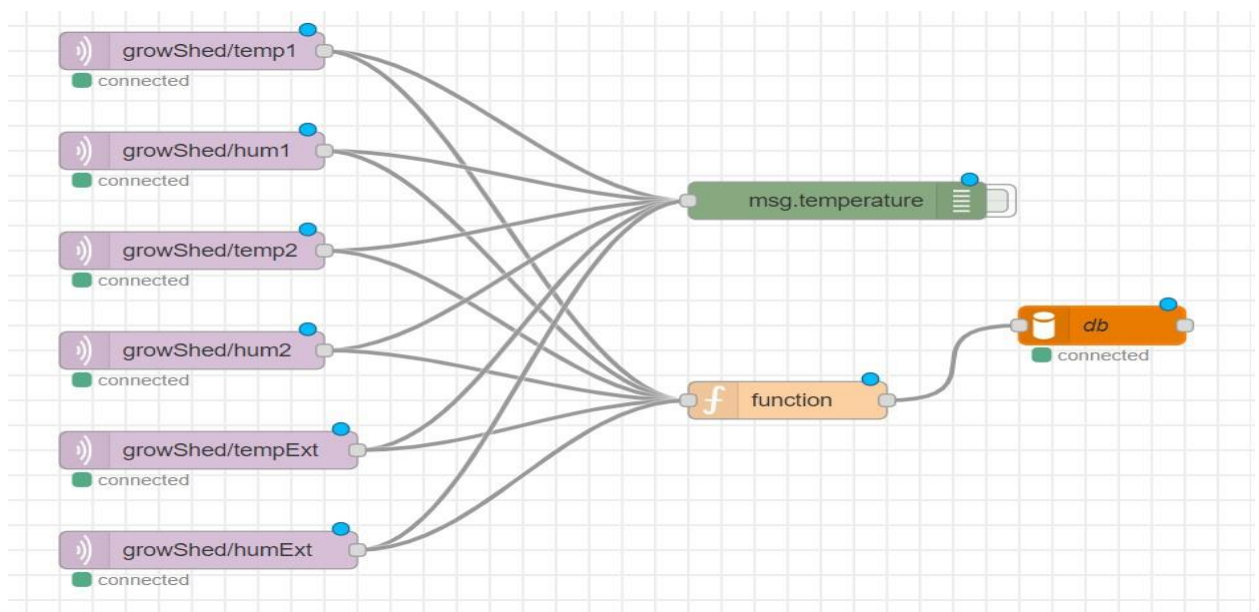


Figure 8 Node-red Flow

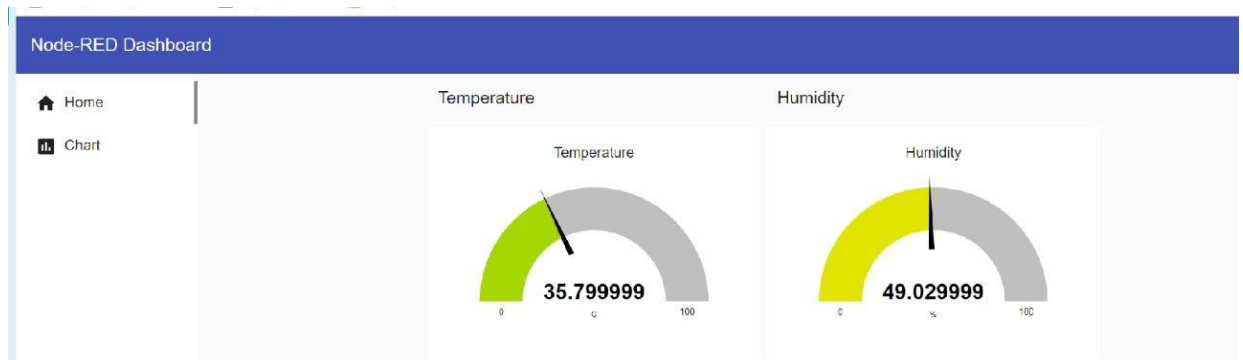


Figure 9 Node-red Output with Dial

Here are the two sample dials output of the code created, which provide an indication of the values of temperature and humidity in Node-RED.

5 Conclusion and Discussion

The formulation of the connection between the proposed device and the sensors in this project is relatively easy to achieve with the aid of the chosen microcontroller (ESP32), given that it is integrated with the pre-installed ADC and DAC which helps in its contribution to increasing the feasibility of the connection between the various sensors and ESP32. Inbuilt MQTT in this microcontroller also assists the communication and setting up of said communication between the server and the device, making it as easily done as of that which is created between a website and a database. The selection of lithium-ion battery with included solar panel Adafruit Universal USB/DC/Solar Lithium Ion/Polymer Charger - bq24074 to power the device has been made in consideration of its capacity to provide a sufficient supply of electricity to the device. The establishment of the correct connection with the database within the system using the right host ID enables it to receive the data being collected by the device accurately and in full, thereby perpetuating a constant and consistent dataflow. This data obtained is then efficiently managed in the database by the practice of a data refinement process and the use of a filter in storage to prevent an overflow of data and the retention of excess data, therefore removing the obligation to upgrade the size of the server unnecessarily.

5.1 Future Scope

Mechanical improvements, i.e., to the physical body of the device, will constitute a large part of its improved sustainability and thereby an increased efficacy in its functioning. The use of a more efficient, lightweight battery will also help in encouraging the improvement of output from the device.

6 References

- Anon., 2018. 5 Countries Most Threatened by Water Shortages.
- Anon., 2019. *operations Management Strategies for the Elimination of Waste in Manufacturing Industry.*, s.l.: UKDiss.
- Anon., 2021. In: *ESP32 series V3.6.* s.l.:s.n.
- Anon., 8. What is Water Security. *What is Water Security*, 2013 5.
- Bakker, K., 2012. Water Security: Research Challenges and Opportunities. *WATER MANAGEMENT*, 24 8, pp. 914-915.
- ChungJae, W.-Y. & Yoo, J.-H., 2015. Remote water quality monitoring in wide area. Volume 217, pp. 51-57.
- Demetillo, A. . T., Japitana, M. . V. & Taboada , E. B., 2019. A system for monitoring water quality in a large aquatic area using wireless sensor network technology. *Springer Nature*.
- Pitt, C. W., Gfeller, F. R. & Stevens, R. J., 1975. *R.F. sputtered thin films for integrated optical components.* s.l.:ScienceDirect.
- Radhakrishnan, V. & Wu, W., 2018. *IoT technology for Smart water system.* Birmingham City, IEEE , p. 6.

7 Appendices

```
#include <Adafruit_BME280.h>
#include <DHT.h>
#include <WiFi.h>

const char* ssid    = "<your wifi name>"; // ESP32 and ESP8266 uses 2.4GHZ wifi
only
const char* password = "<your wifi password>";

//MQTT Setup Start
#include <PubSubClient.h>
#define mqtt_server "192.168.0.205"
WiFiClient espClient;
PubSubClient client(espClient);
#define mqttTemp1 "growShed/temp1"
#define mqttHum1 "growShed/hum1"
#define mqttTemp2 "growShed/temp2"
#define mqttHum2 "growShed/hum2"
#define mqttTempExt "growShed/tempExt"
#define mqttHumExt "growShed/humExt"
//MQTT Setup End

Adafruit_BME280 bme1; // I2C
Adafruit_BME280 bme2; // I2C
#define DHTPIN 33
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

float temp1, temp2,tempExt, hum1, hum2, humExt;

void setup() {
  Serial.begin(9600);
  Serial.println();

  // begin Wifi connect
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(2000);
  WiFi.begin(ssid, password);
```

```

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
//end Wifi connect

client.setServer(mqtt_server, 1883);

dht.begin();
delay(5000);
unsigned status;
status = bme1.begin(0x76);
if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring, address,
sensor ID!");
    Serial.print("SensorID was: 0x"); Serial.println(bme1.sensorID(),16);
    Serial.print("    ID of 0xFF probably means a bad address, a BMP 180 or BMP
085\n");
    Serial.print("    ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("    ID of 0x60 represents a BME 280.\n");
    Serial.print("    ID of 0x61 represents a BME 680.\n");
    while (1);
}
status = bme2.begin(0x77);
if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring, address,
sensor ID!");
    Serial.print("SensorID was: 0x"); Serial.println(bme2.sensorID(),16);
    Serial.print("    ID of 0xFF probably means a bad address, a BMP 180 or BMP
085\n");
    Serial.print("    ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("    ID of 0x60 represents a BME 280.\n");
    Serial.print("    ID of 0x61 represents a BME 680.\n");
    while (1);
}

```



```
}
```

```
void getValues() {
```

```
    temp1 = bme1.readTemperature();  
    temp2 = bme2.readTemperature();  
    tempExt = dht.readTemperature();  
    hum1 = bme1.readHumidity();  
    hum2 = bme2.readHumidity();  
    humExt = dht.readHumidity();
```

```
    Serial.print("BME 1 Temperature = ");  
    Serial.print(temp1);  
    Serial.println(" *C");
```

```
    Serial.print("BME 1 Pressure = ");  
    Serial.print(bme1.readPressure() / 100.0F);  
    Serial.println(" hPa");
```

```
    Serial.print("BME 1 Humidity = ");  
    Serial.print(hum1);  
    Serial.println(" %");
```

```
    Serial.print("BME 2 Temperature = ");  
    Serial.print(temp2);  
    Serial.println(" *C");
```

```
    Serial.print("BME 2 Pressure = ");  
    Serial.print(bme2.readPressure() / 100.0F);  
    Serial.println(" hPa");
```

```
    Serial.print("BME 2 Humidity = ");  
    Serial.print(hum2);  
    Serial.println(" %");
```

```
    Serial.print("Ext Temp = ");  
    Serial.print(tempExt);  
    Serial.println(" *C");
```

```
    Serial.print("Ext Humidity = ");  
    Serial.print(humExt);  
    Serial.println(" %");
```

```
Serial.println("^C");
pHValue = ph.readPH(voltage, temperature); // convert voltage to pH with temperature compensation
Serial.print("pH:");
Serial.println(pHValue, 4);

Serial.println();

}
```

```
Void turbidity()
{
    var s = data.split('-');
    var t = parseInt(s);
    switch(t){
        case 1:
            $('#turbidity').css('background-color','white');
            break;
        case 2:
            $('#turbidity').css('background-color','#EEE1DF');
            break;
        case 3:
            $('#turbidity').css('background-color','D5BDB8');
            break;
        case 4:
            $('#turbidity').css('background-color','#886058');
            break;
        case 5:
            $('#turbidity').css('background-color','#4D2820');
            break;
    }
}
```

```
void reconnect() {
    // Loop until we're reconnected
    int counter = 0;
    while (!client.connected()) {
        if (counter==5){
            ESP.restart();
        }
        counter+=1;
        Serial.print("Attempting MQTT connection...");
    }
}
```

```
// Attempt to connect

if (client.connect("growTentController")) {
    Serial.println("connected");
} else {
    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");
    // Wait 5 seconds before retrying
    delay(5000);
}
}
}

void loop() {
    if (!client.connected()){
        reconnect();
    }
    getValues();

    client.publish(mqttTemp1, String(temp1).c_str(),true);
    client.publish(mqttHum1, String(hum1).c_str(),true);
    client.publish(mqttTemp2, String(temp2).c_str(),true);
    client.publish(mqttHum2, String(hum2).c_str(),true);
    client.publish(mqttTempExt, String(tempExt).c_str(),true);
    client.publish(mqttHumExt, String(humExt).c_str(),true);

    delay(5000);
}
```