Project Course in Electrical Engineering with Emphasis on IoT

ET1544 H22

Project Report

Canny Fishbowl

KUMMATHI JOTHEESH REDDY
SAVITRI VENKATA TEJASWAR

Lecturers: Saleh Javadi and Alper Idrisoglu

Department of Mathematics and Natural Sciences
Blekinge Institute of Technology (BTH)
Karlskrona, Sweden

Abstract

Context: IoT, where real-world things are built into intelligent things that are controlled virtually. In order to provide us control over the things around us and to keep us updated on their status, the IoT aspires to bring everything in our environment under one common infrastructure. These days IoT can be seen in very common, day-to-day things. One such thing is an aquarium. With the help of IoT, aquariums have developed a lot. One of their few functionalities is being monitored by the user remotely.
Objectives: The main objective of the project is to build an IoT based smart aquarium along with a web application and a mobile application where the aquarium can be monitored remotely. The web application and the mobile application will show the readings precisely of the monitored aquarium and will have options like feed, pump, etc. The readings are taken from the sensors and the options provided are actuators.
Methods: We developed a model in which sensor readings are transmitted from an Arduino to an MQTT broker, and then used those readings to create an interactive web and mobile application.
Results:. Our project's prototype is successfully acquiring data from the sensors and sending it to an MQTT broker. The data is then displayed on a custom web and mobile app that we developed, allowing users to view the data in real time and interact with the actuators.
Conclusions: The development of our IoT-based smart aquarium has demonstrated the potential for the Internet of Things to improve the lives of aquarium owners and the health of their aquatic pets. The system allows for remote monitoring and control of various parameters, such as temperature and water level, using MQTT protocols for efficient and reliable communication. We believe that this system has the potential to change the way aquariums are managed and maintained, and we hope that our work will inspire others to utilize the power of the IoT in innovative ways to solve real-world problems. There is also room for further advancement in this field, and we are excited to see what the future holds for the development of smart aquariums and other IoT-based solutions.

Keywords: actuators, HTML,Internet of Things (IoT), monitoring, MQTT, sensors, smart aquarium.

Acknowledgments

We are deeply grateful to our professors, Mr. Alper Idrisoglu and Mr. Saleh Javadi, for their exceptional guidance and support throughout the course. Their expertise in the field and wealth of knowledge have been invaluable resources, and we have greatly benefited from their insights and suggestions. We appreciate their patience and encouragement as we worked on this project. Their contribution to our skill development has been invaluable, and we are truly grateful for their mentorship.

Kummathi Jotheesh Reddy
Venkata Tejeswar Savithri

Contents

Abstract i

Acknowledgments ii

List of Figures v

List of Tables vi

List of Acronyms vii

1 Introduction 1

2 Related Work 2

3 Methodology 4
3.1 Modeling 6
3.2 Implementation 12
3.2.1 Hardware Implementation 12
3.2.2 Software Implementation 23

4 Results and Discussion 34

5 Conclusions and Future Work 38

References 39

Appendices 41
.1 Appendix #1 42
.1.1 Arduino code for the Canny fish bowl 42
.1.2 Web application HTML code. 50

List of Figures

3.1 Schematic block diagram 4
3.2 Proposed Model 5
3.3 sensors and actuators connected to Arduino 6
3.4 Relation between analog output and millimeters 7
3.5 Relationship between turbidity and voltage 8
3.6 Relationship between NTU and voltage 8
3.7 Relationship between power ans the volume 10
3.8 Flow chart of the proposed model 11
3.9 Arduino Uno WiFi rev2 13
3.10 Temperature sensor(DS18B20) 14
3.11 Water Level sensor 15
3.12 convertion of sensorvalue>voltage>NTU 16
3.13 Analog Turbidity Sensor 16
3.14 MG90S micro servo 17
3.15 MG90S micro servo 18
3.16 MG90S micro servo 19
3.17 MG90S micro servo 20
3.18 Fish food dispenser 21
3.19 Working Circuit 21
3.20 Digital circuit 22
3.21 Prototype 22
3.22 Flow chart to dump the code in arduino 24
3.23 compilation and uploading to Arduino board 24
3.24 Flow chart of the Arduino after dumping the code 25
3.25 HTML, CSS, JS 26
3.26 Developed web interface 27
3.27 MIT app inventor 28
3.28 Designer window 29
3.29 Blocks window 29
3.30 Developed Blocks for App interface 30
3.31 Developed App interface 31
3.32 Flow chart of the actuators functionality 33

4.1 Working Circuit 34

4.2 Prototype 35
4.3 Prototype 35
4.4 Developed App interface 36
4.5 Developed web interface 37

List of Tables

3.1 Analog water level sensor reading. 7
3.2 Power reading for given volume 9

4.1 Readings from the prototype. 37

List of Acronyms

CSS Cascading Style Sheets. iv, 26

DC Direct current. 18

HTML Hypertext Markup Language. i, iv, 23, 26

IDE integrated development environment. 23

IoT Internet of Things. i, 1–4, 13, 38

JS Java Script. iv, 26

JSON JavaScript Object Notation. 26, 30

LED Light emitting Diode. 14, 26, 32

MIT Massachusetts Institute of Technology. 23, 28

MQTT Message Queuing Telemetry Transport. i, 2–5, 10, 23, 25, 26, 30, 32, 36, 38

NTU Nephelometric Turbidity unit. iv, 5, 8, 9, 16, 37

TCP Transmission Control Protocol. 3

TSS Total suspended solids. 16

V Voltage. 9

Wi-Fi Wireless Fidelity. 10, 23, 25, 32, 36

Chapter 1

Introduction

Recent studies have shown people are more drawn toward fish and want them as pets. One such way is to store them in a container with water to provide them with oxygen. The water needed to be changed every day, as there was no other way to find out how clean the aquarium was other than seeing it with the bare eye. As technology evolved, people wanted more than just a box with water. Thus, the concept of an aquarium came to light. In the beginning, these aquariums were built with just water filters, so the water needed to be changed less frequently. The filters built into the aquarium remove the toxicity in the water, clean the water, and filter the leftover food particles in the water.

As aquariums became more popular, their capabilities expanded, allowing people to go for longer periods without changing or filtering the water. However, all of these functions necessitate frequent monitoring and human intervention. And going out of the station for too long caused concern for the fish, who need food and fresh water for oxygen. [1] [2]

To tackle the situation, a solution is required where continuous monitoring is available without users being at the aquarium 24/7. a system in which the aquarium will be continuously monitored using an IoT. With these kinds of aquariums, humans can monitor the readings remotely. There is no need to physically check the aquarium. This makes humans not worry about the aquarium if they are gone for too long.

The aquarium will have a web application and a mobile application where users can see the readings of the water level, turbidity, and temperature. They can even feed the fish and pump water into the aquarium by reading the water level remotely. The web application and mobile application display all the readings and give the user the option to act according to the readings. For example, if the web page and mobile application show the water level is below a certain level, then with the option "pump" provided, the human can pump the water into the aquarium through the web page or mobile application without being physically present.

Chapter 2

Related Work

The history of aquariums can be traced back to ancient civilizations such as China, Egypt, and Rome, where fish keeping was popular as a pastime. However, it wasn't until the 19th century, with the advancement of glass-making technology, that transparent glass containers were created and used as modern aquariums. This sparked the growth of the aquarium hobby, which gained popularity among both amateur and professional aquarists.

As technology continued to advance, so too did the capabilities of aquariums. The development of artificial lighting and filtration technologies in the early 20th century allowed for the creation of increasingly complex and sophisticated aquariums. In the 1950s and 1960s, the home computer revolution and the growth of the internet led to the development of new technologies for the aquarium industry, such as automated feeding systems and digital water quality monitors. [3]

Today, the aquarium hobby continues to thrive and evolve, with new technologies being developed to make it easier for aquarium owners to monitor and care for their tanks. One such technology is the Internet of Things (IoT), which allows for the remote monitoring and control of various aspects of the tank through the use of sensors and smart devices. IoT technology has also made it possible to automate tasks such as water changes and feeding through the use of smart pumps and automatic feeders.

The aim of our project was to create an IoT-based smart aquarium and a corresponding web and mobile application that allows the user to remotely monitor the condition of their aquarium in real-time. To gain a deeper understanding of this, we conducted research on related projects that utilized both the IoT and MQTT protocols. These resources provided valuable insights on the essential components and technologies that we need to include in our project in order to successfully achieve our objectives.

- Article "An IoT-Based Smart Aquarium Monitoring System" presents a smart aquarium system that uses Internet of Things (IoT) technologies to enable remote monitoring and control of the aquarium environment. The system includes sensors for measuring various parameters, such as temperature, pH, and dissolved oxygen levels, and actuators for adjusting these parameters as needed. The system uses MQTT (Message Queuing Telemetry Transport) as the communication protocol to transmit data between the sensors and actuators and a cloud server. The authors conducted experiments to evaluate the performance of the system and found that it was effective at improving the health and well-being of the aquatic life in the aquarium. [4]

- Article "Smart Agriculture Using the Internet of Things and MQTT Protocol" discussed the use of the MQTT (Message Queuing Telemetry Transport) protocol in the field of agriculture. This lightweight protocol, the publish-subscribe messaging protocol that works on the TCP protocol to ensure data security and transfer, is suitable for IoT applications because of its low overhead and ability to function in low-bandwidth and high-latency environments. The authors discuss the various ways in which MQTT is applied in agriculture, such as for remote monitoring and control of irrigation systems, crop health, and soil conditions, and outline the benefits of using MQTT in agriculture, including its reliability and scalability. They also emphasize the need for further research and development in this area to fully utilize the potential of MQTT in agriculture. [5]

MIT App Inventor is a user-friendly software program that allows for the easy development of mobile applications. Its drag-and-drop interface makes it simple to use. MQTT is a useful protocol for connecting devices and servers in IoT systems because it is efficient and works well in resource-constrained environments. When combined with MIT App Inventor, it has the potential to create user-friendly IoT applications. You will learn more about the impact of these related works on our proposed model further down.

Chapter 3

Methodology

A schematic plan and the approach of trial and error may be used to address any issue. To comprehend the answer to our project, we have drawn a schematic block diagram (figure 3.1). A very crucial component of every IoT prototype project is an Arduino board. For this project, we have used an Arduino WiFi Rev 2 model.

To develop a smart IoT-based aquarium, we must feed the fish, manage the temperature of the water, monitor the cleanliness and level of the water, and often replace the water using a water pump. All of this should be under the user's control, which means we've created a mobile and web application MQTT broker via which the user can monitor the aquarium remotely.

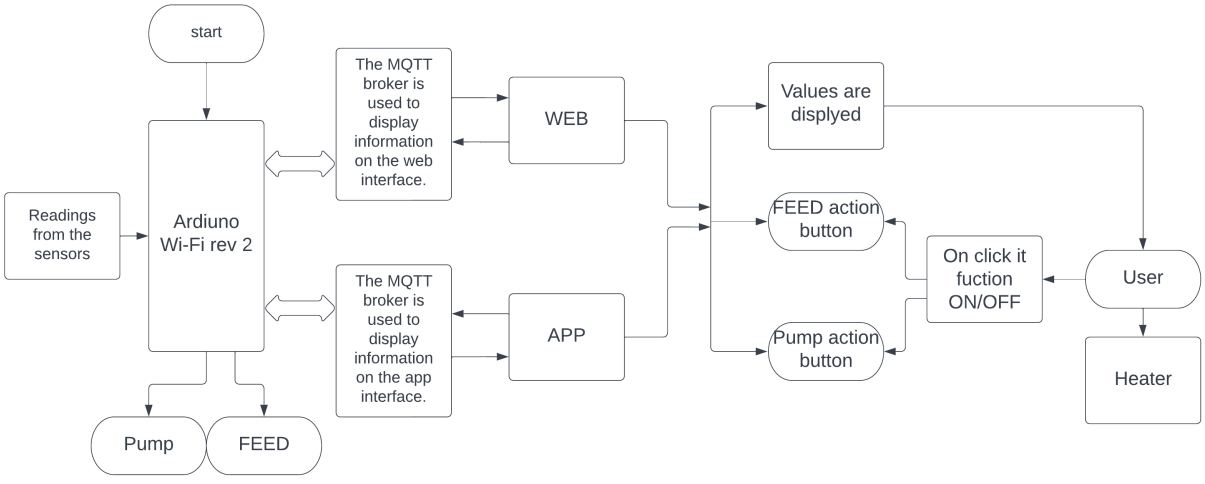

Figure 3.1: Schematic block diagram

Let's look at the block diagram. A temperature sensor that measures the temperature in the aquarium is connected to the Arduino board, allowing the user-installed heater to operate automatically based on its temperature settings.

We designed a module that is attached to a servo motor to feed the fish. When the user interacts with the feed, the servo begins to rotate, and food is released. Water is extremely important in an aquarium. For a fish to thrive in an aquarium, the cleanliness of the water and the amount of water must be maintained. We attached an analog turbidity sensor to check the water purity, which gives a number that should be converted into NTU (a user-understandable value) using the sensor value. We've also added a water level sensor, which tells us how much water is in the tank. We put a water pump whose job is to change the water, and the user has control over it.

Everything is within the user's control. He only needs to connect to NTU from anywhere in the globe to monitor the aquarium. Using the MQTT broker, he can also remotely feed the fish and turn on and off the pump.

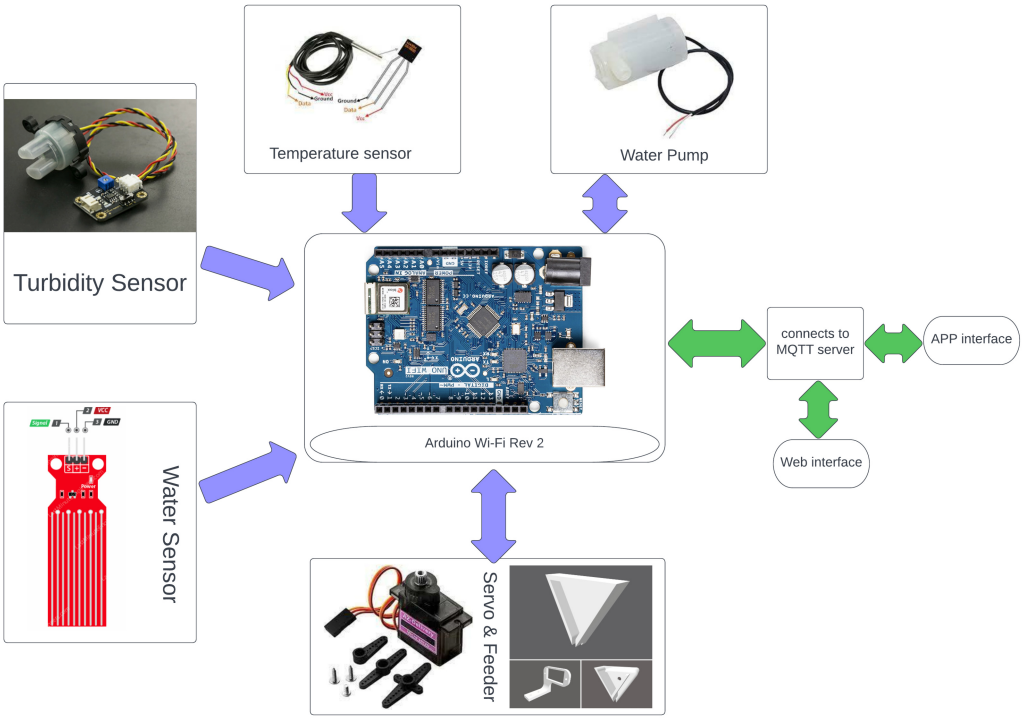

Figure 3.2: Proposed Model

Following the description of the schematic flow and proposed model required for our project above, the chapter moves on to modeling and various implementations.

3.1 Modeling

In this section, we'll go through the readings from the sensors and how we'll use them, as well as how we'll interact with the actuators and the sequential flow of our proposed model.

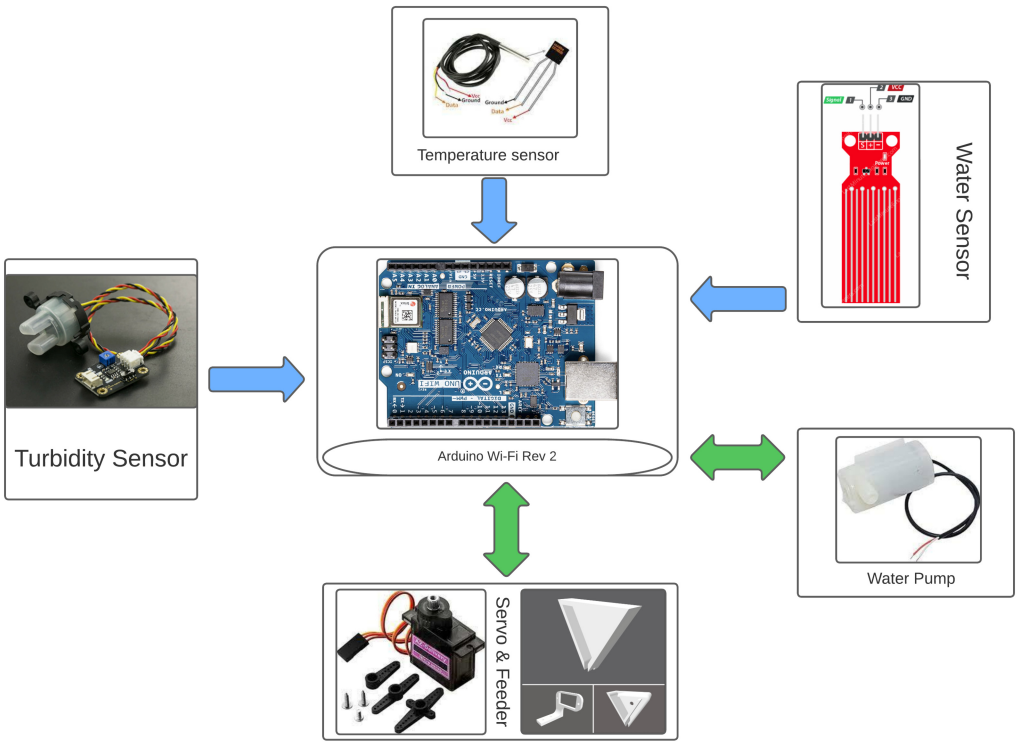

Figure 3.3: sensors and actuators connected to Arduino

We begin with readings from the sensors connected to the Arduino, and we go into depth about these sensors in the next sections. We begin with the temperature sensor (DS18B20), which provides a value in degrees Celsius. If you want your temperature in Fahrenheit, simply use this formula 3.1 to convert Celsius to Fahrenheit.

$$Fahrenheit(^{\circ}F) = [(Temperature\ in\ celsius\ (^{\circ}C) \times (9/5)) + 32] \tag{3.1}$$

The second sensor is the water level sensor. Because it is an analog sensor, its measurements are in the range of 0 to 1023 rather than millimeters or centimeters. So, in order to convert the analog value to a centimeter, we must do a small experiment, the results of which are as follows [6]:

Table 3.1: Analog water level sensor reading.

The sensor's length when immersed in water	Analog readings
0 mm	480
5 mm	530
10 mm	615
15 mm	660
20 mm	680
25 mm	690
30 mm	700
35 mm	705
40 mm	710

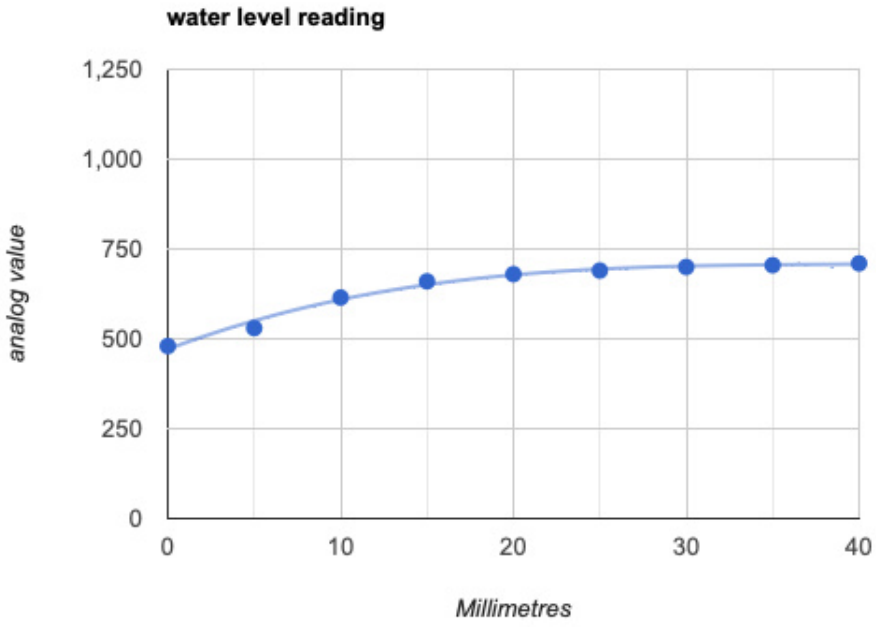

Figure 3.4: Relation between analog output and millimeters

The last sensor in our project, the analog turbidity sensor, informs us about the purity of the water, but as the name implies, it provides us data in analog values ranging from 0 to 1023. I was able to convert these analog values into voltage thanks to this article i got to know about the relationship between the turbidity reading and the voltage. [7]

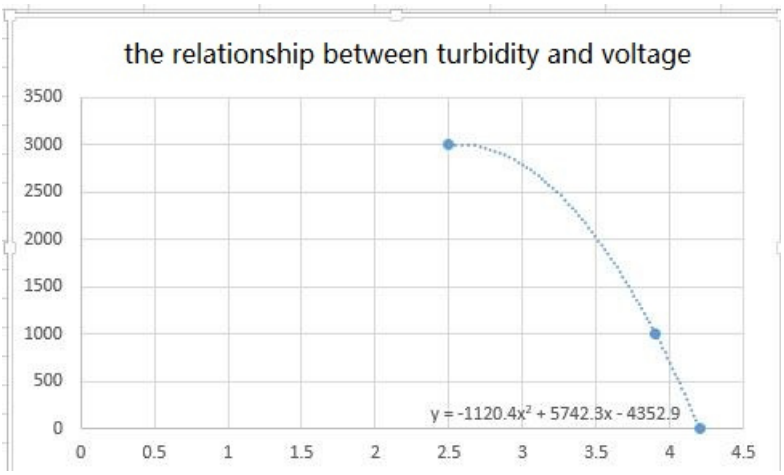

Figure 3.5: Relationship between turbidity and voltage

However, turbidity is measured in Nephelometric Turbidity unit (NTU) rather than voltage, therefore these are the findings of a 2018 study conducted by W L Hakim to determine the relationship between voltage and NTU. [8]

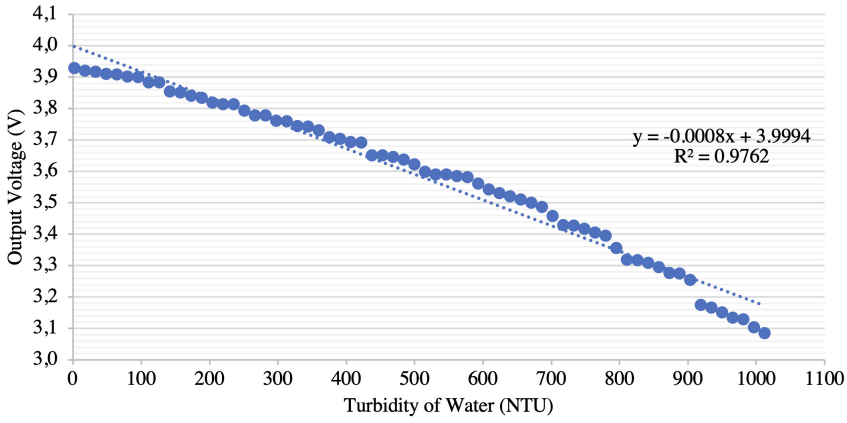

Figure 3.6: Relationship between NTU and voltage

From the graph above, we can obtain an equation for the NTU in terms of voltage.

$$Output\,Voltage\,(V) = [-0.0008\,(Turbidity\,of\,Water\,(NTU))\,+\,3.9994] \quad (3.2)$$

from equation 3.2...

$$Turbidity\,of\,Water\,(NTU) = [\frac{(Output\,Voltage\,(V)) - 3.994}{(-0.0008)}] \quad (3.3)$$

Now that the sensors have been calibrated, we can proceed to calculate the energy requirements. To do this, we need to know the power and time. The time can be obtained from the Arduino, which has a built-in timer. Once we have these two parameters, we can use them to accurately calculate the energy.

$$Energy(E) = Power(P) \times Time(T) \quad (3.4)$$

To determine the length of time that the heater was in operation, you need to subtract the time at which the water temperature falls below the required temperature (T1) from the time at which the temperature returns to the required temperature after falling (T2). This will give you the total time that the heater was in operation.

$$t = T2 - T1 \quad (3.5)$$

To calculate the power generated by a heater, you need to determine the heater's heat output and the time taken for the operation. You can do this by analyzing the data provided with the heater, taking into account factors such as the volume of water being heated. We prepared a table and graph using this data.

Table 3.2: Power reading for given volume

Volume in litres	power in watts
10 L	25 W
15 L	50 W
35 L	75 W
60 L	100 W
90 L	150 W
130 L	200 W
180 L	250 W
230 L	350 W

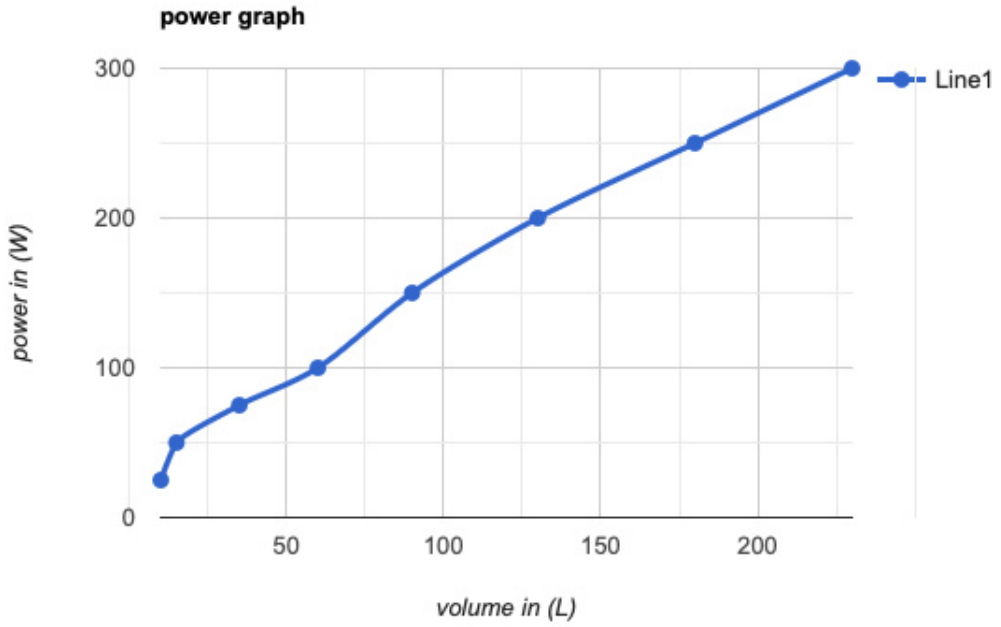

Figure 3.7: Relationship between power ans the volume

From the graph, we have interpolated an equation for power (p(x)) in terms of volume (x). This equation allows us to determine the power generated by the heater at different volumes.

$$p(x) \approx -0.00158892x^2 + 1.59028x + 17.5911 \tag{3.6}$$

Equation 3.4, energy in joules, can be calculated using equations 3.5 and 3.6.

Let us now describe the flow of our proposed model before proceeding to the actuators. When we turn on the Arduino, it attempts to connect to Wi-Fi, and if it fails, it tries again, while the reading begins to flood into the Arduino, as shown in Figure 3.7. We introduced a MQTT broker for the user to monitor the aquarium, which Arduino tries to connect to and communicate data to appropriate dictionaries every second. Using the MQTT broker, a web application and mobile application must be created in which the sensor readings are displayed and the actuator buttons are configured to fulfill their functions.

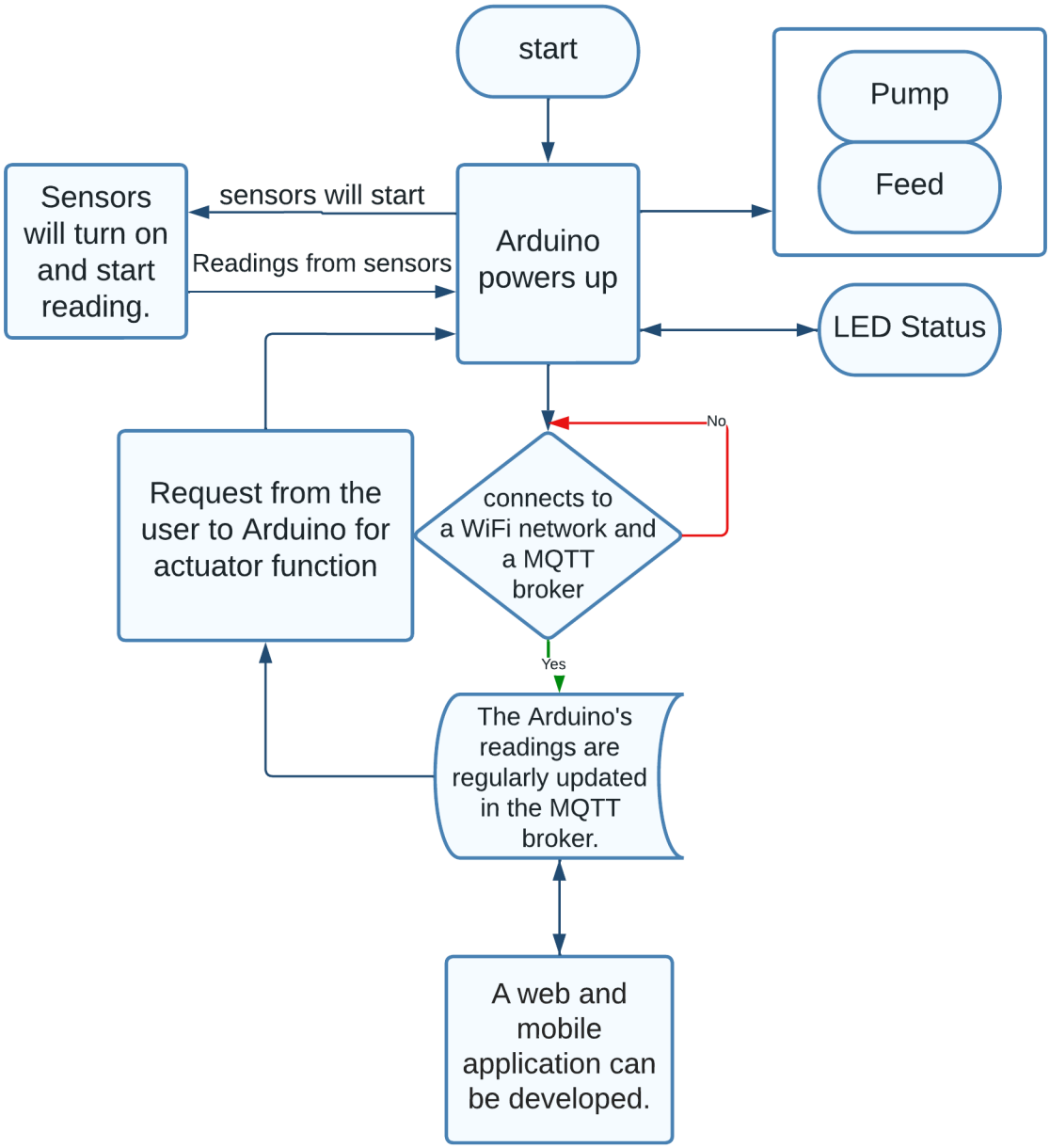

Figure 3.8: Flow chart of the proposed model

In this project, the actuators are a servo motor and a water pump. The servo motor feeds the fish by rotating 180 degrees and returning to its starting position when triggered . When the water pump is enabled, it begins to function and must be turned off by the user.

3.2 Implementation

This chapter's section is separated into two subsections, one for hardware implementation and one for software implementation. In the hardware implementation, you'll learn about component specs and how we used them in this project and set up those components, while in the software implementation, you'll learn about user access to the components through a interface.subsection are:

1. Hardware Implementation.
2. Software Implementation.

3.2.1 Hardware Implementation

As earlier said, we will use a variety of components to keep the aquarium functioning well. components involve:

1. Arduino Uno WiFi rev2.
2. Temperature sensor(DS18B20).
3. Water Level sensor.
4. Analog Turbidity Sensor.
5. Servo motor(MG90S).
6. Water pump (5v).
7. Relay module (5v).
8. Water heater (75W).
9. Fish food dispenser.

1. Arduino Uno WiFi rev2

The Arduino UNO WiFi Rev.2 is the most common device used for the prototype projects on IoT. The Arduino UNO WiFi Rev.2 is your best option for many of the fundamental IoT and other application scenarios, whether you want to build a sensor, collect the data from the sensor and control the sensor.

The Arduino UNO WiFi Rev 2 includes the secure ATECC608 crypto chip accelerator, as well as a Microchip ATmega4809 8-bit micro controller. It also has an onboard IMU (inertial measurement unit), LSM6DS3TR, and u-NINA-W102 Blox's Wi-Fi and Bluetooth® modules. [9]

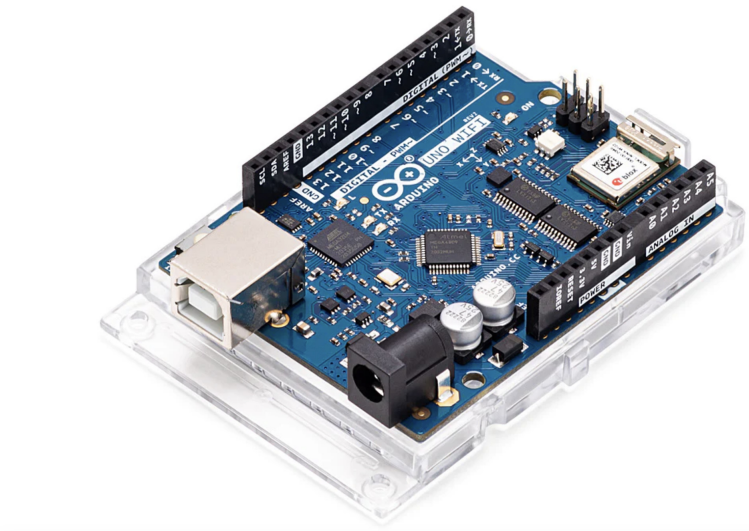

Figure 3.9: Arduino Uno WiFi rev2

Technical specifications of Arduino Uno WiFi rev2:

- **Micro controller:**ATmega4809
- **Pins:** 25 (Digital I/O Pins:14, Analog input pins:6, PWM pins:5)
- **Connectivity:** Bluetooth®, Wi-Fi
- **Supply Voltage:** +5 V
- **Memory:** ATmega4809 (6KB SRAM, 48KB flash, 256 bytes EEPROM)

2. Temperature sensor(DS18B20)

There are various temperature sensors available; we chose this DS18B20 since it is waterproof and sturdy in comparison to other temperature sensors. The data from the DS18B20 is 9-12 bits long. The DS18B20 is the appropriate device for our application because it must be submerged. [10]

Figure 3.10: Temperature sensor(DS18B20)

Technical specifications of Temperature sensor(DS18B20):

- **Supply voltage:** MIN:+3.0 MAX:+5.5
- **Thermometer Error :** ±0.5°C (-10°C to +85°C), ±1°C(-30°C to +100°C), ±2°C (-55°C to +125°C)
- **Programmable Resolution:** 9 Bits to 12 Bits
- **Temperature Conversion Time:** MAX: 750ms (12bits)
- **Pins:** 3 (GND,DQ,Vcc)

3. Water Level sensor:

We utilize this sensor to read the aquarium's water level, as the name implies. In terms of hardware, the sensor contains 10 exposed copper traces, five of which are power traces and the remaining five are sensing traces. These traces are interwoven so that one sensory trace is connected to every two power traces. Power and sensation traces are normally not related, but when immersed in water, they become connected. When the board is powered, a power LED on the board will illuminate. [11]

The power and sensor traces combine to generate a variable resistor whose resistance fluctuates depending on how much water is present. This resistance changes inversely with the sensor's depth of immersion in water:

- The sensor's conductivity and resistance improve when it is submerged in more water.
- The lower the conductivity and greater the resistance, the less water the sensor is submerged in.

The sensor provides an output voltage proportionate to the resistance; the water level may be measured by monitoring this voltage.

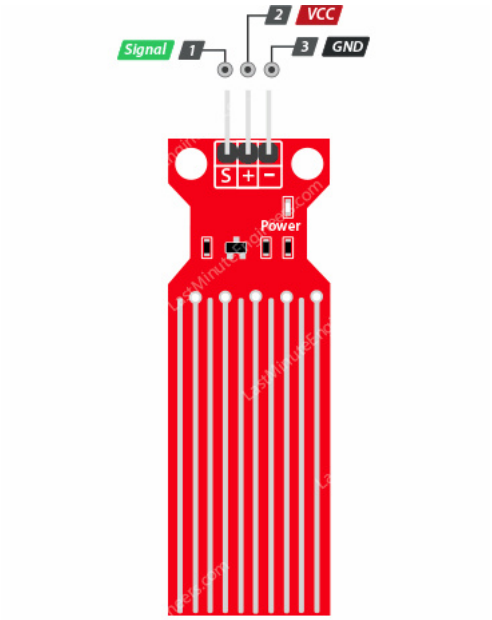

Figure 3.11: Water Level sensor

Technical specifications of Water Level sensor [12]:

- **Working Voltage:** DC 3-5V
- **Working Current:** <20mA
- **Detection Area:** 40 mm x 16 mm
- **Working Temperature:** 10 °C to 30 °C
- **Pins:** 3 (Signal,GND,Vcc)

4. Analog Turbidity Sensor:

The Arduino turbidity sensor detects water quality by measuring the level of turbidity. It is able to detect suspended particles in water by measuring the light transmittance and scattering rate, which change with the amount of Total suspended solids (TSS) in the water. As the TSS increases, the liquid turbidity level increases. [7]

The water quality rating in terms of Nephelometric Turbidity unit (NTU) was required. So we must first convert the sensor value to voltage, and then use the voltage to determine the NTU as discussed above in modeling. [8]

```
219  float tur(){
220    int sensorValue = analogRead(A0);// read the input on analog pin 0:
221    float voltage = sensorValue * (5.0 / 1023.0); // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
222    float ntu = (voltage-3.994)/(-0.0008); //converting the voltage to NTU
223    Serial.print("voltage: ");
224    Serial.println(voltage); // print out the value you read:
225    delay(500);
226    Serial.print("NTU: ");
227    Serial.println(ntu);
228    return ntu;
229  }
```

Figure 3.12: convertion of sensorvalue>voltage>NTU

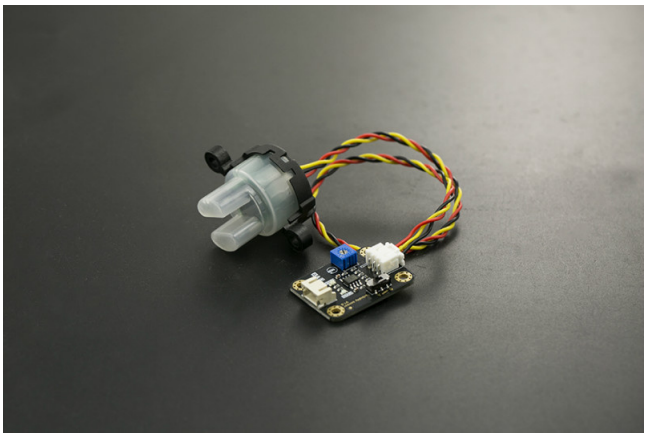

Figure 3.13: Analog Turbidity Sensor

Technical specifications of Water Level sensor [13]:

- **Operating Voltage:** 5V DC
- **Working Current:** 40mA (MAX)

- **Response Time:** <500ms
- **Insulation Resistance:** 100M (Min)
- **Output Method:** Analog and Digital

5. Servo motor(MG90S):

This MG90S servo can spin about 180° (90° in each direction) and comes with three servo horns (arms) and hardware—position 0° (1.5 ms pulse) is in the middle. The overall concept is straightforward. We regard 1.5 ms to be the "neutral" position. Increasing the 1-2 ms pulse width causes the servo to move in one direction, while reducing the pulse width causes the servo to move in the opposite direction.The servo's primary function is to store and feed the fish. [14] [15]

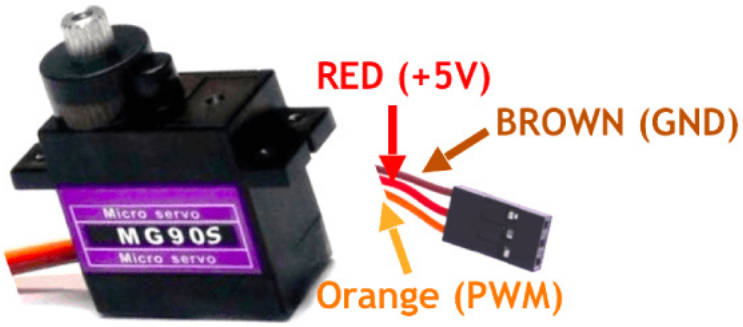

Figure 3.14: MG90S micro servo

Technical specifications of MG90S micro servo:

- **Operating Voltage:** 4.8V to 6V (Typically 5V)
- **Stall Torque:** 1.8 kg/cm (4.8V)
- **Max Stall Torque:** 2.2 kg/cm (6V)
- **Gear Type:** Metal
- **Rotation :** 0°-180°

6. Water pump (5v):

A water pump is as important in every aquarium as lungs are in the human body since the water in an aquarium must be changed often due to the leftover food and waste emitted by the fish. In this project, a 5 volt DC micro submersible water pump fulfilled this function. [16]

Figure 3.15: MG90S micro servo

Technical specifications of Water pump:

- **Operating Voltage:** 5V DC
- **Flow Rate:** 2.5L/Min
- **Pump Type:** Submersible
- **current consumption:** 300mA to 1A max

7. Relay module (5v):

A relay, as we all know, is an electromechanical device that is used for switching and connecting. The relay module in this project serves as an intermediary between the Arduino and the water pump relay, allowing the Arduino to regulate the function of the water pump, either starting or stopping it. [17]

Figure 3.16: MG90S micro servo

Technical specifications of Single channel relay module:

- **Supply voltage:** 3.75V to 6V
- **Quiescent current:** 12mA
- **Current when the relay is active:** 70mA
- **Relay maximum contact voltage:** 250VAC or 30VDC
- **Relay maximum current:** – 10A
- **Pins:** 6

8. Water heater (75W):

We used the platinum glass heater for the aquarium in our project; it includes a high-precision electronic thermostat. The temperature may be set between 20 and 33 degrees Celsius and maintained with a precision of +/- 0.4 degrees Celsius or greater. The heater includes a ceramic heat sink and an electronic thermometer built in. Because of the straightforward temperature adjustment, it is simple to operate. [18]

Figure 3.17: MG90S micro servo

Technical specifications of Aquael Platinum Heater 75 W [19]:

- **Effect:** 75 watts
- **Temperature range:** 20 - 33° C
- **Accuracy:** + / -0.4 ° C
- **Fits aquarium:** About 35 - 75 liters
- **Length:** 225 mm°

9. Fish food dispenser:

It's a 3D-printed module attached to the servo's head that dispenses food into the aquarium when it revolves 180°.

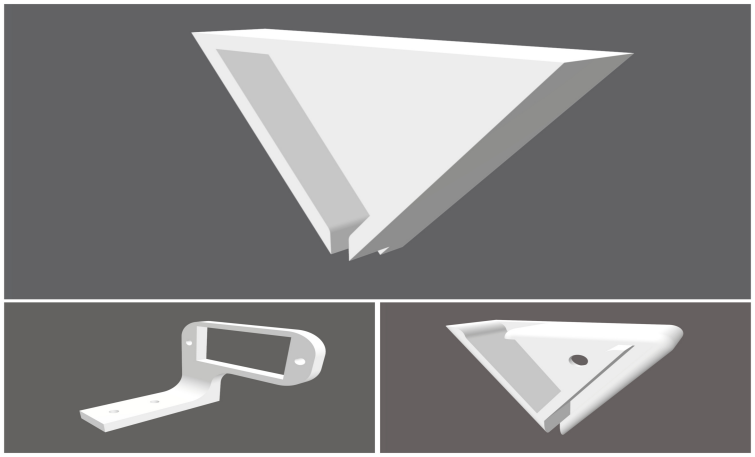

Figure 3.18: Fish food dispenser

We created a circuit using the components listed above (Figure 3.2). We began with a breadboard and then mounted it on an aquarium box.

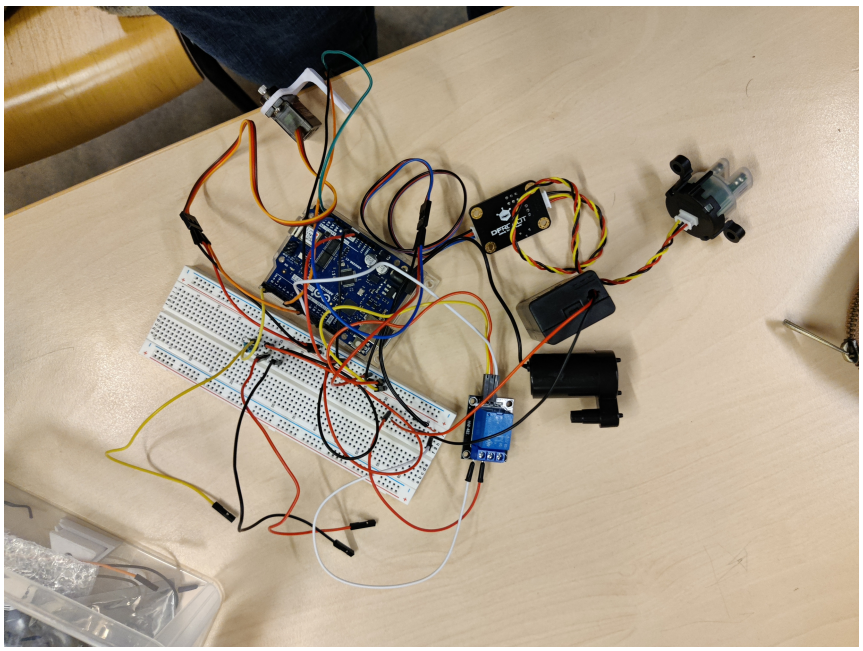

Figure 3.19: Working Circuit

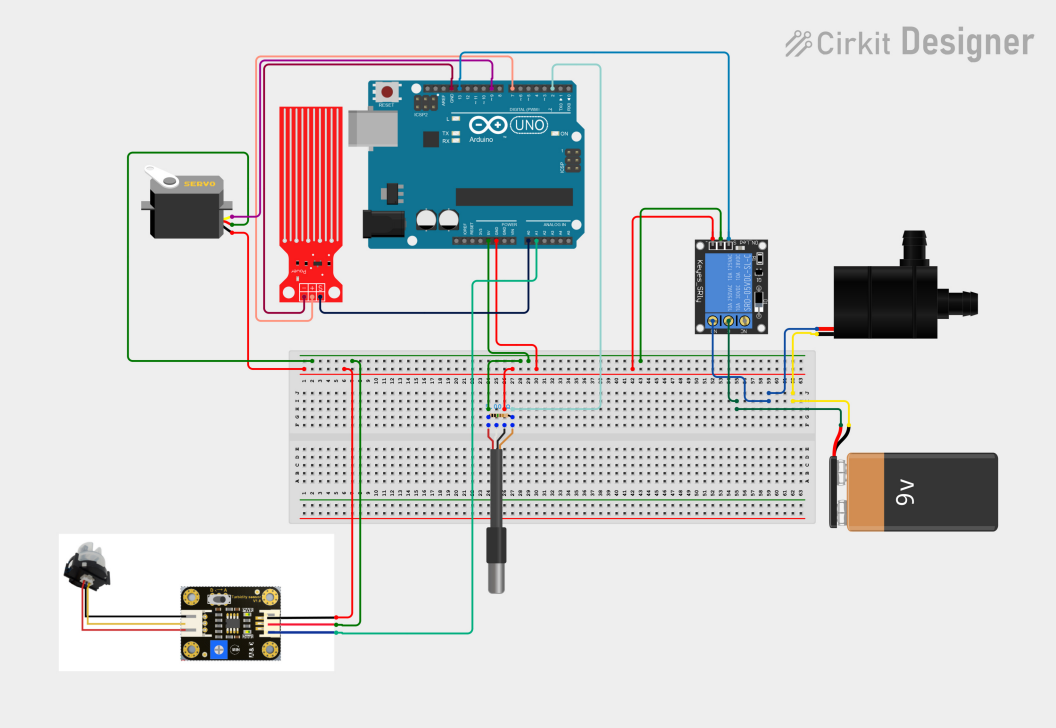

Figure 3.20: Digital circuit

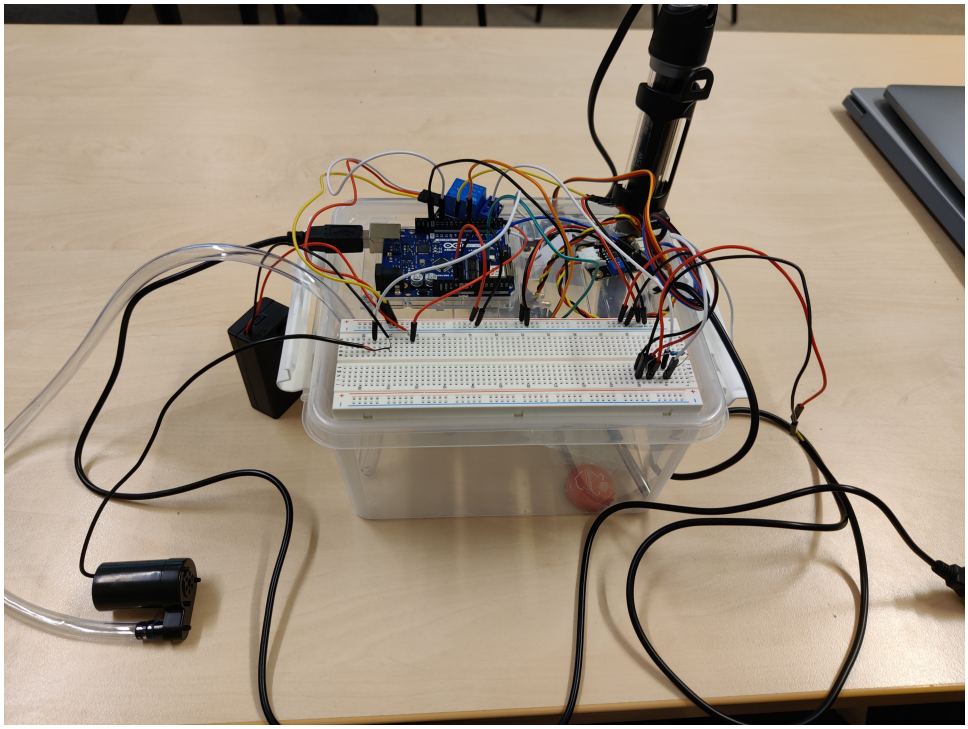

Figure 3.21: Prototype

3.2.2 Software Implementation

So far, we've gone through the hardware components and their connections in great detail, but we'll need software to make the gear function. So, according to our model, we need three things:

- To begin reading the components, an Arduino code is necessary, and the Arduino should connect to Wi-Fi and link to a MQTT broker.

  1. Developing and dumping the Arduino program.

- We built a web interface with HTML and inline CSS to allow the user to monitor the aquarium as proposed.

  2. Developing web application using HTML.

- We also recommended a user interface for the mobile, which we built using MIT App Inventor.

  3. Developing mobile application using MIT app inventor.

1. Developing and dumping the Arduino program:

Arduino, our project's preferred micro-controller, also provides open-source software that is simple to use for beginners and that expert programmers may explore [9]. The Arduino project makes available the Arduino integrated development environment (IDE). Because the components, as we all know, cannot understand standard language, this environment converts the Arduino programming language into binary machine code for the components.C/C++ is used to write Arduino code.

A "sketch" is a program developed for the Arduino in the Arduino IDE. The Arduino IDE includes a "Wiring" software library from the Wiring project, which supports many common input and output processes. A basic Arduino C/C++ sketch consists of two functions that are compiled and linked into an executable cyclic executive program through a program stub main(). [20]

- **setup():** is a function that runs once at the beginning of a program and can be used to initialize settings.
- **loop():** is a function that is called continuously until the board shuts down.

First, we have to dump the developed code onto the Arduino board after it has been compiled successfully without any errors (Figure 3.22).

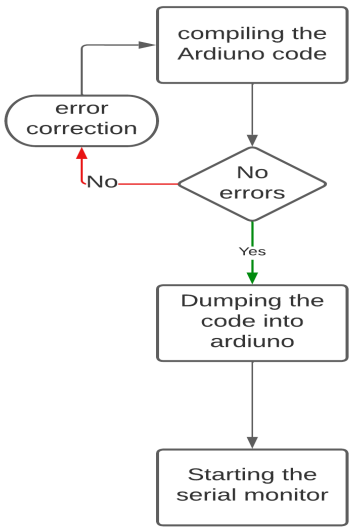

Figure 3.22: Flow chart to dump the code in arduino

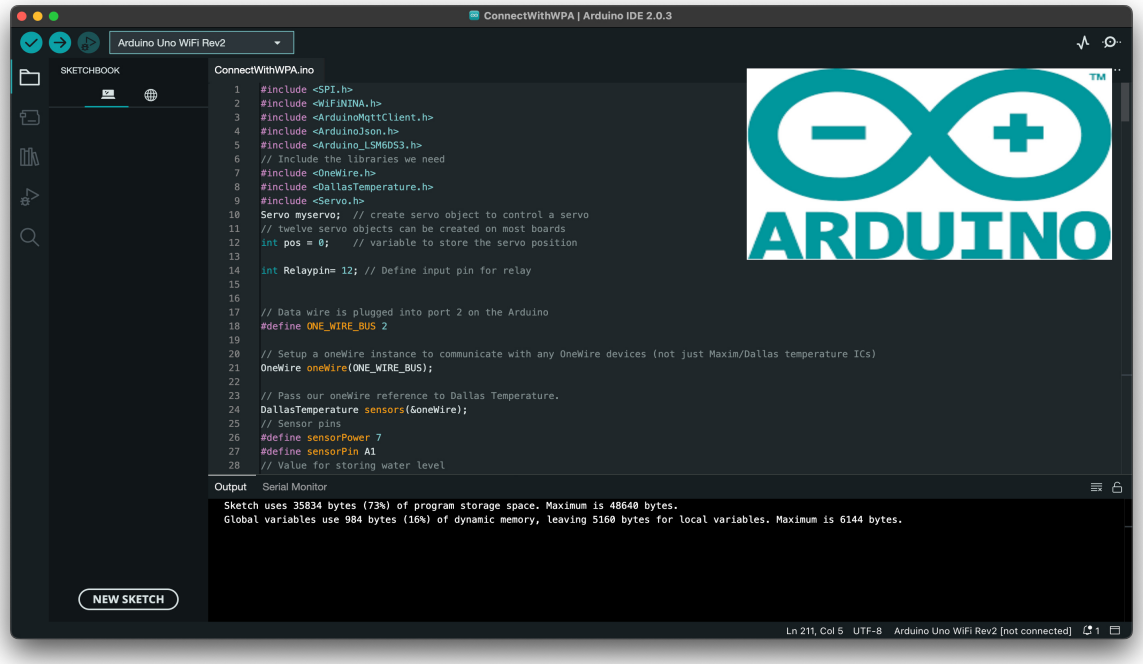

Figure 3.23: compilation and uploading to Arduino board

The serial monitor is then opened to allow the Arduino to try to connect to the defined Wi-Fi; if it fails, it tries again and again. When the Arduino connects to WiFi, it attempts to connect to an MQTT broker so that data from sensors and messages from the MQTT broker can be transmitted.Messages are transmitted from the MQTT broker to Arduino when the user interacts with a buttons in the interface.

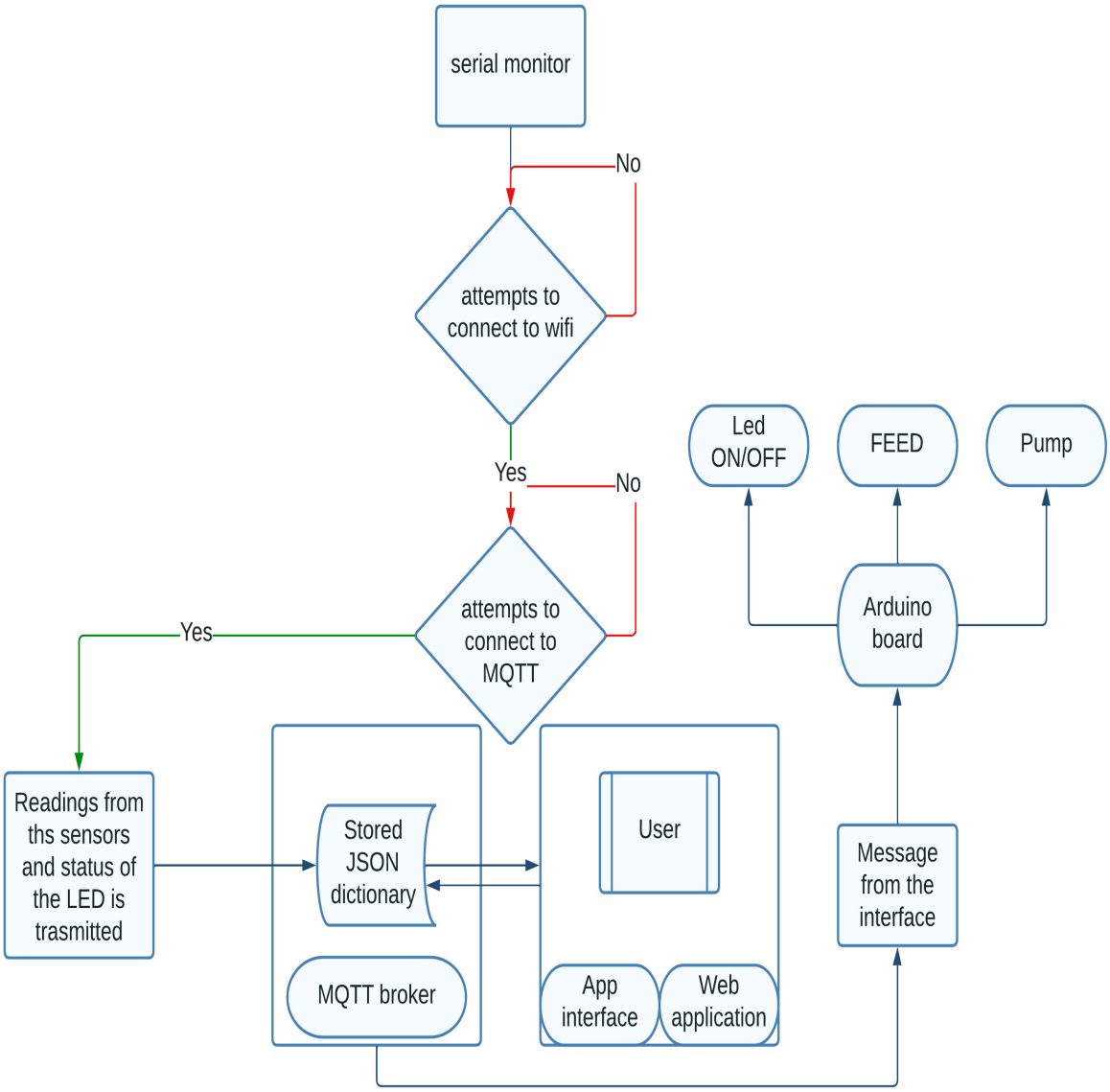

Figure 3.24: Flow chart of the Arduino after dumping the code

2. Developing web application using HTML:

We created a web interface using simple HTML and inline CSS styling. HTML does not require an introduction because it is a widely used technology for developing web applications.

Figure 3.25: HTML, CSS, JS

As previously stated, the sensor data are sent by Arduino to a MQTT broker are saved in JSON dictionaries in a Java script file. The values from the Arduino, which include temperature, water level, turbidity, and timer, must be shown in the web interface, and the user must be able to interact with the water pump, feeder, and LED on the Arduino. After importing the Java script file into the LED, we call them and assign them to their corresponding categories.

You can view the web interface that we developed using HTML, CSS, and JS on the following page, and the complete code is included in Appendix 1.2.

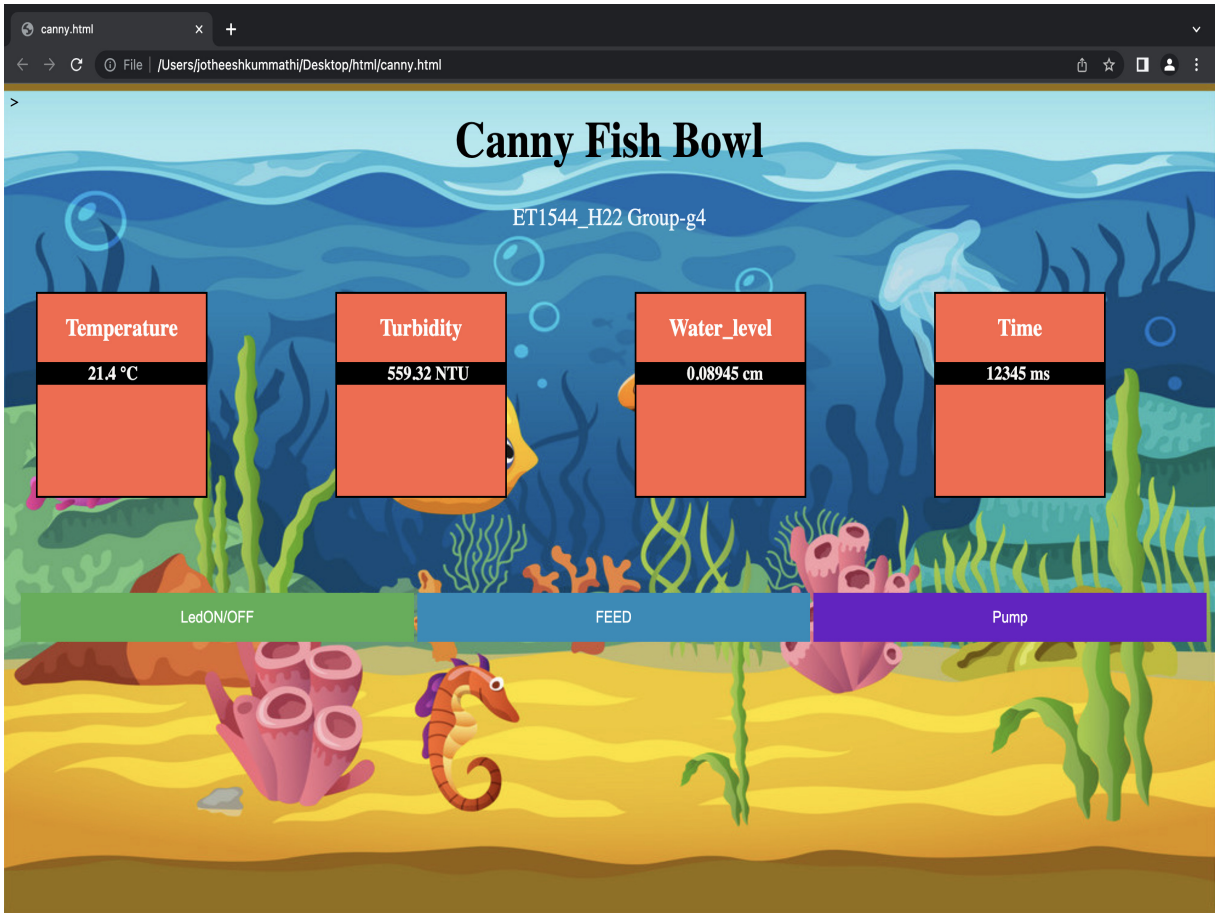

Figure 3.26: Developed web interface

3. Developing mobile application using MIT app inventor:

The Massachusetts Institute of Technology (MIT) App Inventor, a web application integrated development environment initially offered by Google MIT. It is a drag-and-drop visual programming tool for creating and developing fully complete Android mobile apps. It is an excellent open source software for students who want to develop, build, and employ personally relevant mobile technology solutions in their everyday lives and projects. [21]

Figure 3.27: MIT app inventor

This software has two windows for a project: one called "Designer," which shows the screen of our app and can be edited, and another called "Blocks," which allows you to perform the app's logic. The figures 3.26 and 3.27 below are the windows, respectively.

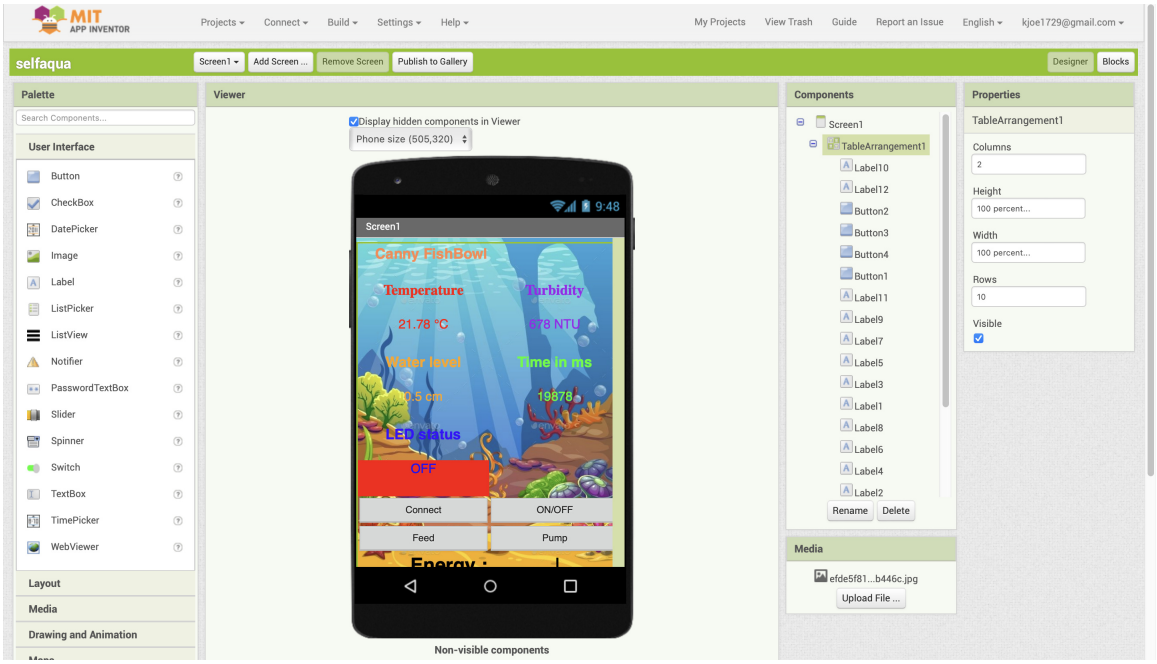

Figure 3.28: Designer window

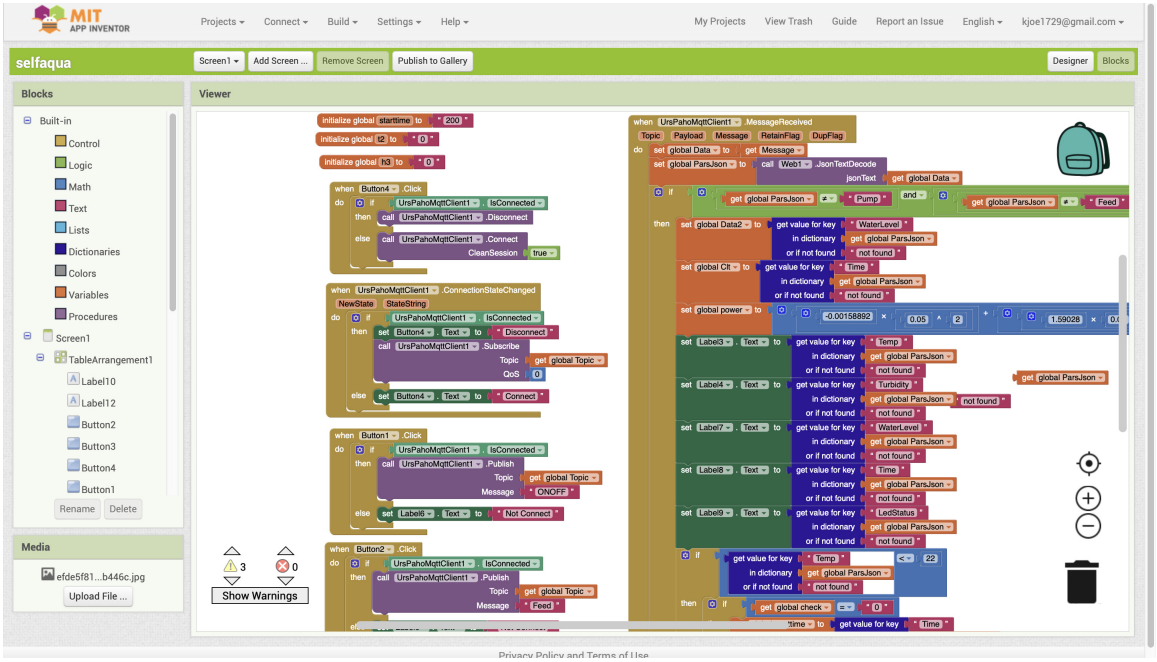

Figure 3.29: Blocks window

So we created the Mobile interface by connecting to the MQTT broker, importing data from JSON dictionaries, adding logic to blocks, and categorizing sensor readings. Figure 3.29 shows the generated mobile interface, whereas Figure 3.28 shows the logic that was constructed between the blocks.

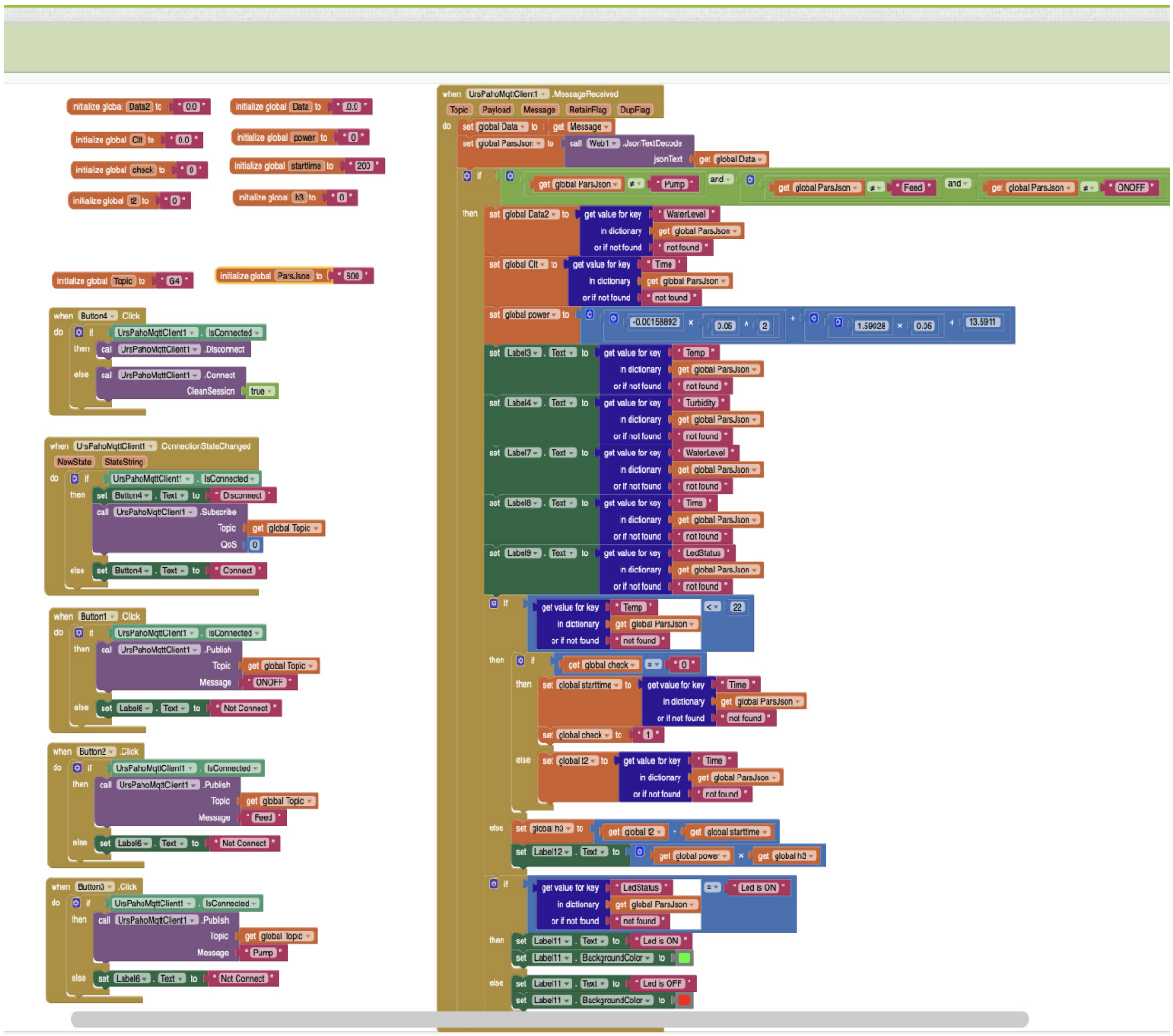

Figure 3.30: Developed Blocks for App interface

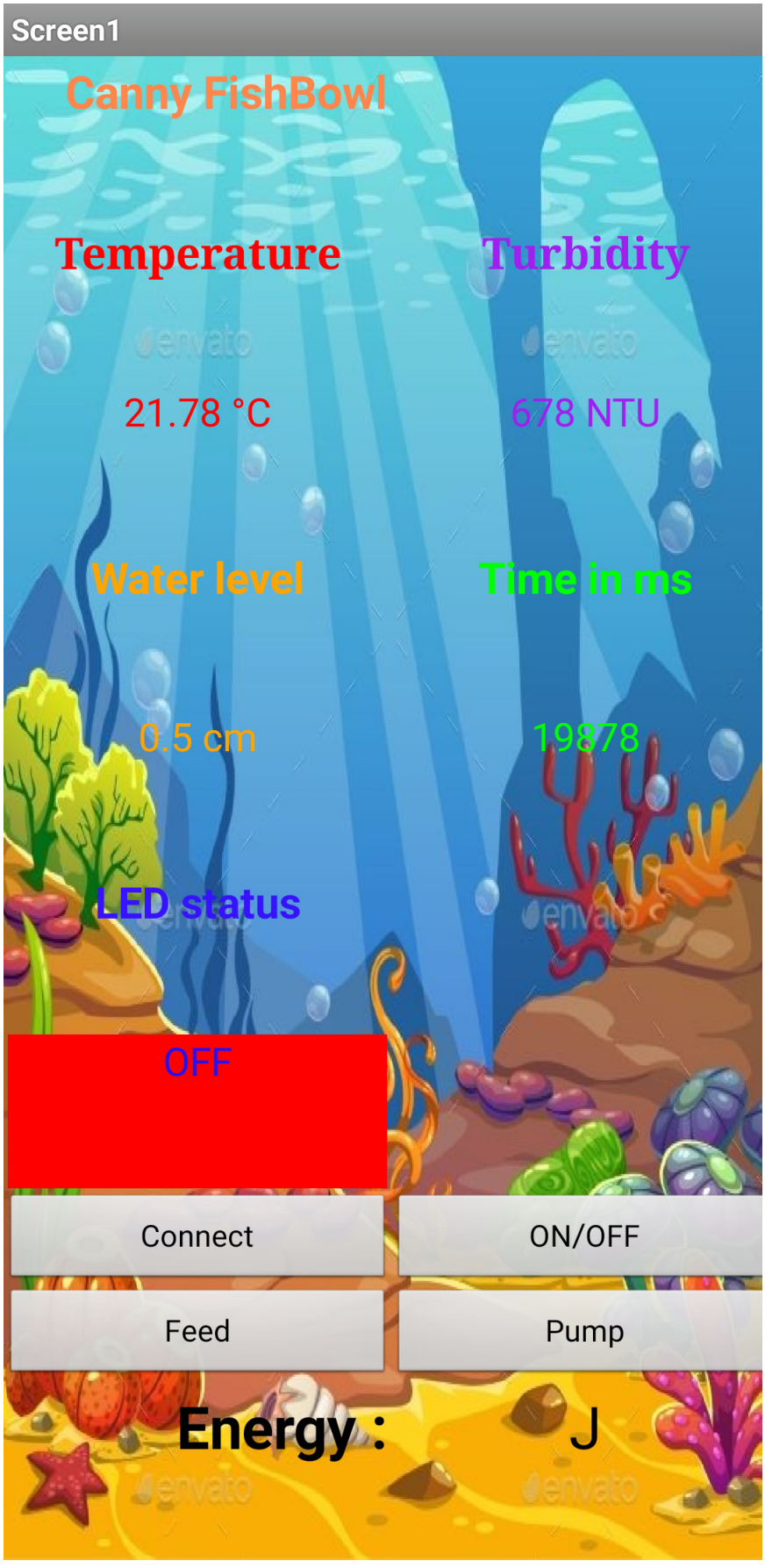

Figure 3.31: Developed App interface

This section concludes with an explanation of the functionality of the buttons Pump, Feed, and LED.We made a flowchart to help you understand.

The user has control over both the mobile application and the web interface interface, as shown in the flow chart. He may perform the function of any of the three actuators by just clicking on the appropriate button.Arduino communicates to the MQTT broker after connecting to Wi-Fi, as previously described. This connection enables Arduino to send sensor readings and receive requests from MQTT broker, which are initiated by the user.

There are three buttons in our design, each representing a different function:

1. LED on/off.
2. Feed.
3. Pump.

1. LED on/off:

When the LED on/off button in any interface is clicked, the MQTT broker receives the request and sends it to Arduino. If the Arduino's led is already turned on, the request turns it off; if the led is already off, the request turns it on.

2. Feed:

When the feed button in any interface is clicked, the MQTT broker receives the request and sends it to Arduino. Arduino gets the request and passes it on to the servo motor, which then starts rotating 180° degrees clockwise and comes back to its initial position of 0°.

3. Pump:

When a pump button is clicked in any interface, the MQTT broker recognizes the request and sends it to Arduino. If the pump is already operational, the request switches it off; if the pump is not operational, the request activates it.

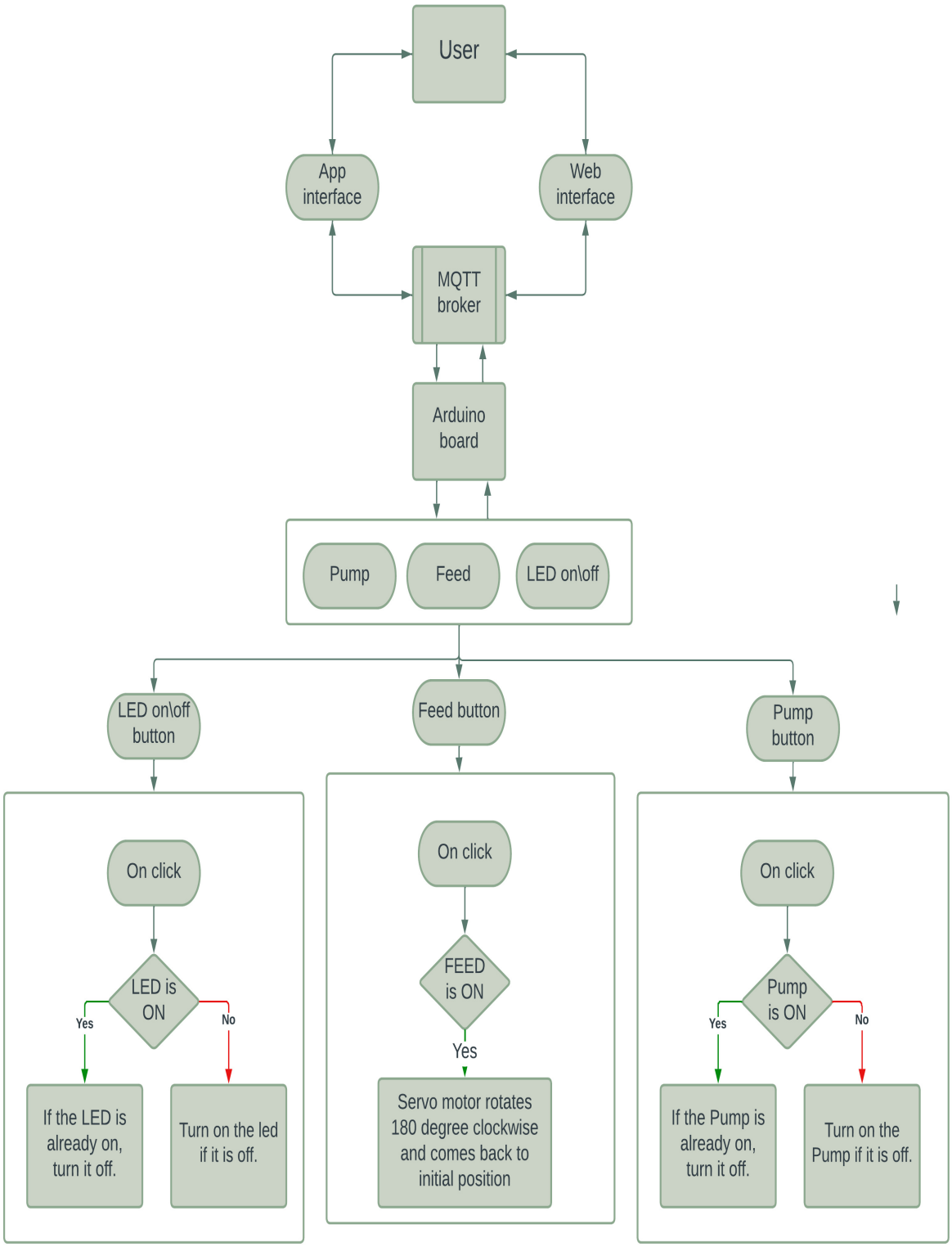

Figure 3.32: Flow chart of the actuators functionality

Chapter 4

Results and Discussion

We began by working with each component independently, and after the success of one component, we moved on to the next; we conducted this experiment on a breadboard. Once we have developed the web application and app interface, We have arranged the components most efficient manner possible for the prototype aquarium.

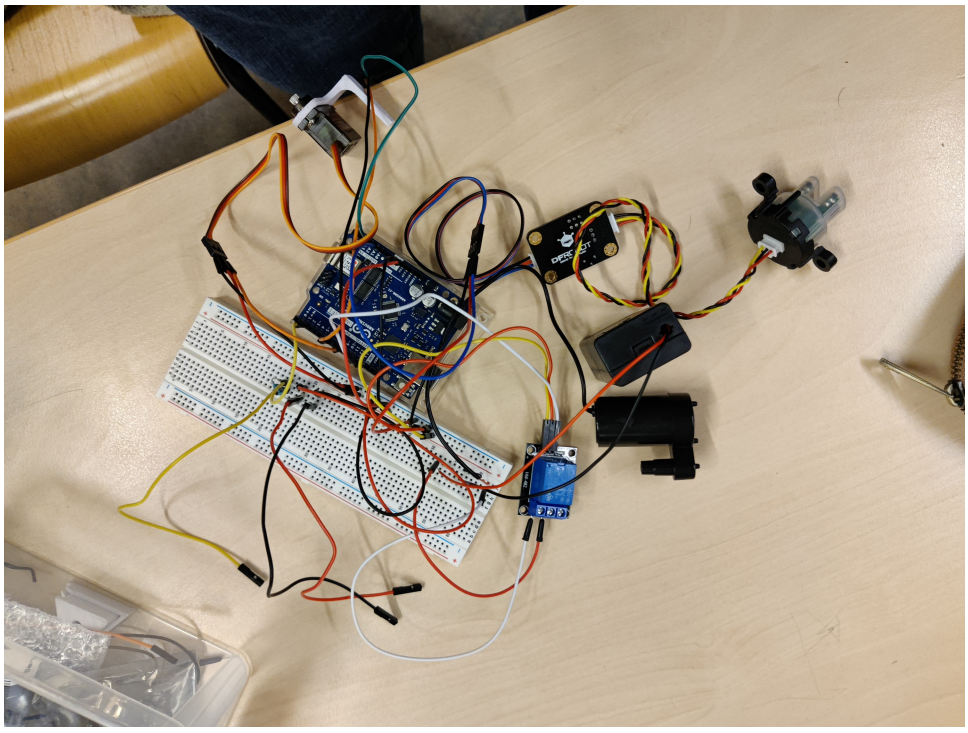

Figure 4.1: Working Circuit

The heater, turbidity sensor, water level sensor, and temperature sensor have been installed on the prototype, and the servomotor is mounted to the inside wall; the Arduino and breadboard are mounted on the top of the lid; the battery is located outside the prototype, but the water pump pipe is placed inside.

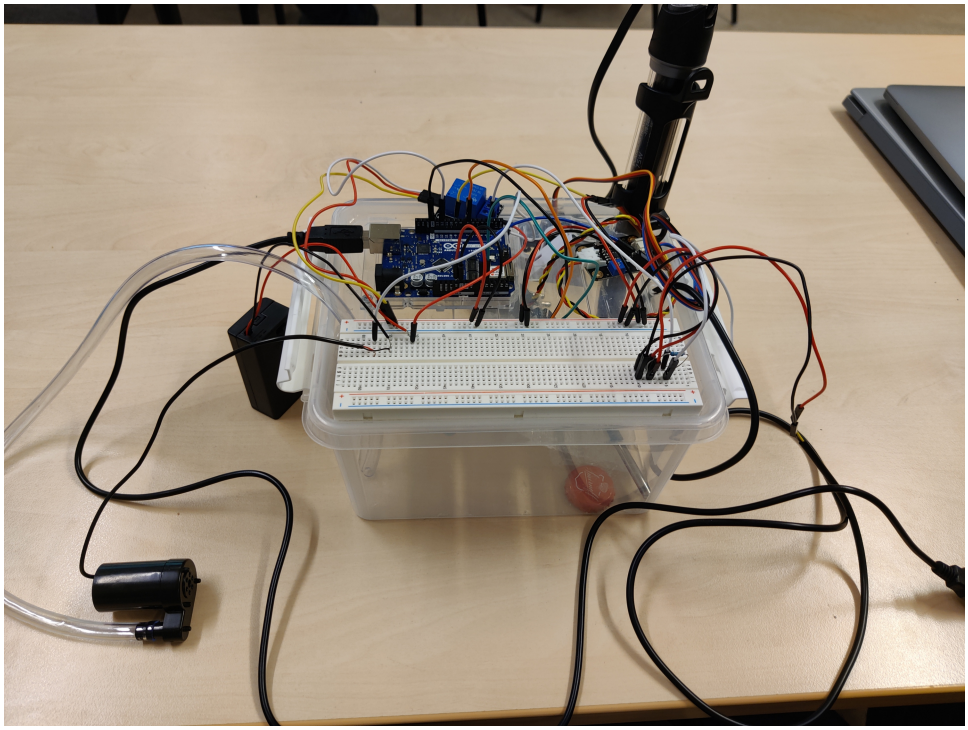

Figure 4.2: Prototype

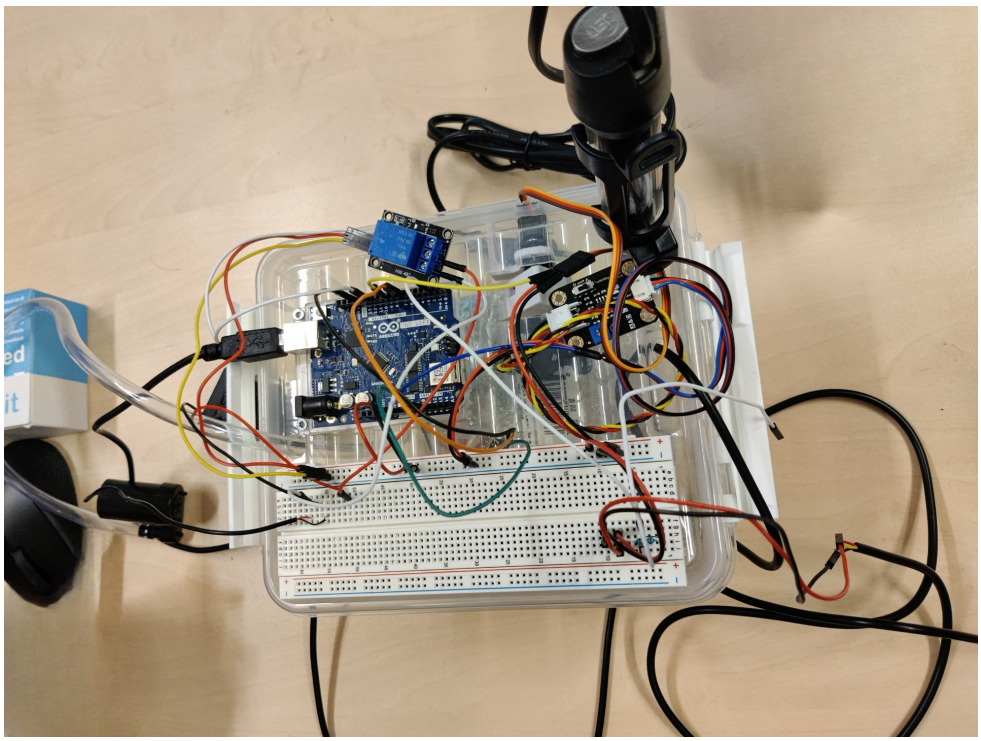

Figure 4.3: Prototype

The Arduino board powers up and connects to the specified Wi-Fi network after the Arduino code is successfully uploaded. Our web interface works flawlessly, and when the user clicks the connect button on the app interface, it connects to the MQTT broker and begins showing the information.

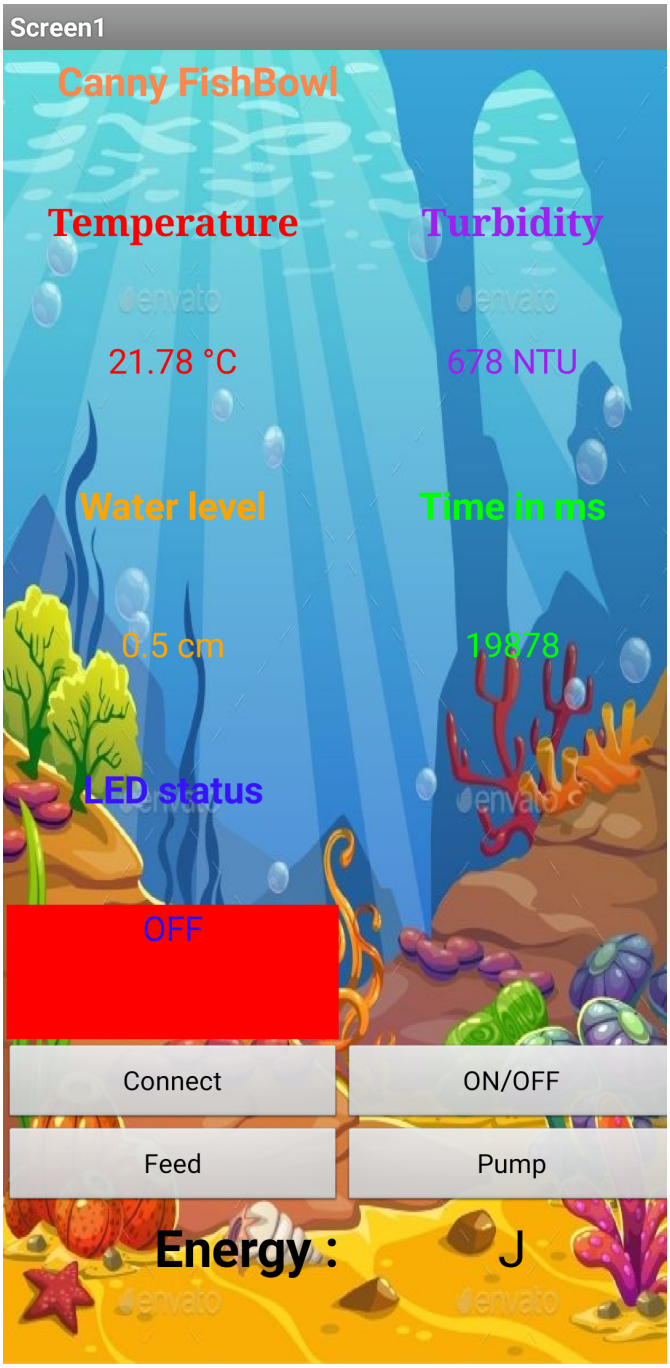

Figure 4.4: Developed App interface

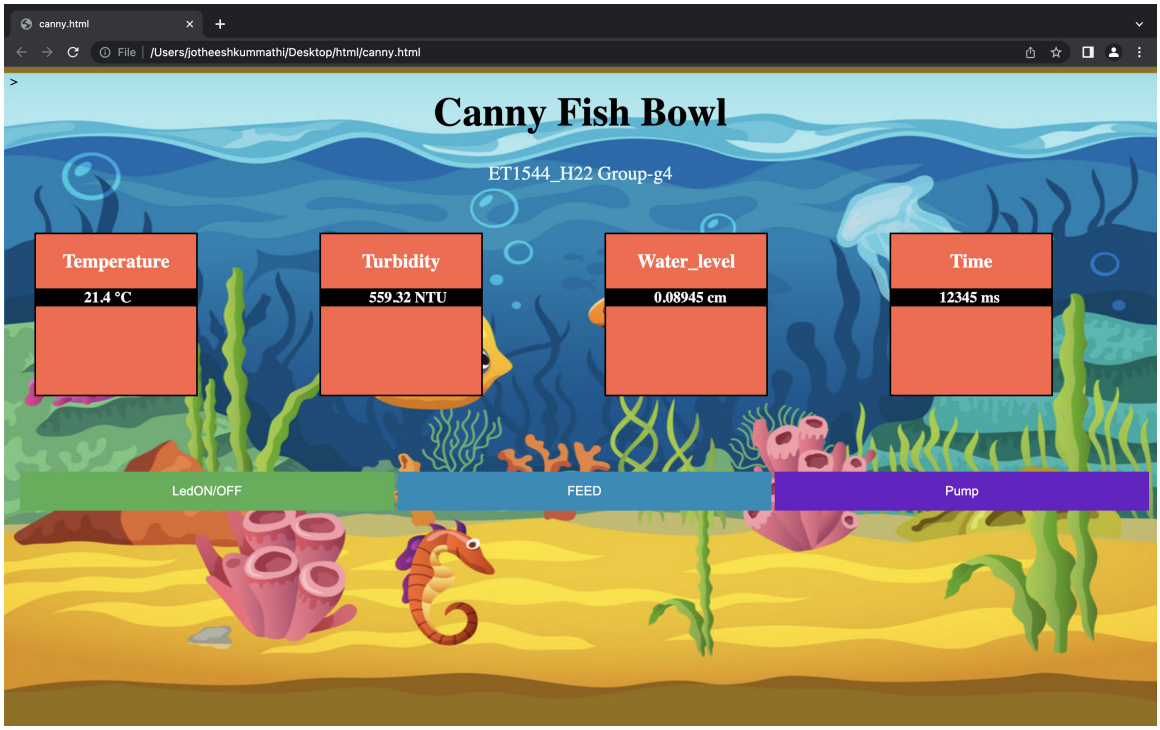

Figure 4.5: Developed web interface

These are the readings we obtained when testing our prototype with 0.3 liters of water.

Table 4.1: Readings from the prototype.

Sensors	Readings
Temperature sensor in $°C$	21.6
Turbidity sensor in NTU	688.54
water level sensor in centimeter	0.5
Energy in joules	1512.7449

Note:

1. We observed a delay in the functioning of the pump and servo when they were requested.
2. Temperature rise from $21°C$ to $25°C$
3. Power = 18.066 Watts
4. Time = 83.73 sec

Chapter 5

Conclusions and Future Work

The development of our IoT-based smart aquarium has demonstrated the potential for the Internet of Things to improve the lives of aquarium owners and the health of their aquatic pets. The system allows for remote monitoring and control of various parameters, such as temperature and water level, using MQTT protocols for efficient and reliable communication. We believe that this system has the potential to change the way aquariums are managed and maintained, and we hope that our work will inspire others to utilize the power of the IoT in innovative ways to solve real-world problems.

There are many potential directions for future work on the smart aquarium system. One possibility is to integrate additional sensors and actuators to provide even more detailed and precise control over the aquarium environment. This could include support for parameters such as oxygen levels and feeding mechanisms. Another option is to add advanced automation and control algorithms, such as machine learning models that can adapt to the needs of the aquarium over time. Additionally, integrating the system with other smart home devices and systems, such as smart thermostats, lighting, and security systems, could create a seamless smart home experience.

Expanding the system to support Koi ponds is another potential area of future work [22]. This would involve adapting the system to work with the larger scale and unique needs of koi ponds, such as filtration and aeration. It would also be valuable to continue refining the user interface and experience to make it even more user-friendly and intuitive for users of all skill levels.

Overall, the IoT-based smart aquarium system has great potential for further development and refinement, and we are excited to see what the future holds for this innovative technology.

References

[1] Benefits, problems, and characteristics of home aquarium owners - aline h. kidd, robert m. kidd, 1999. [Online]. Available: https://journals-sagepub-com.miman.bib.bth.se/doi/abs/10.2466/pr0.1999.84.3.998

[2] aquarium - maintenance problems | britannica. [Online]. Available: https://www.britannica.com/science/aquarium/Maintenance-problems

[3] algone. History of the aquarium & fish keeping. [Online]. Available: https://www.algone.com/history-of-the-aquarium-fish-keeping

[4] Realization of IoT based fish farm control using mobile app | IEEE conference publication | IEEE xplore. [Online]. Available: https://ieeexplore-ieee-org.miman.bib.bth.se/abstract/document/8644854

[5] S. V. Mukherji, R. Sinha, S. Basak, and S. P. Kar, "Smart agriculture using internet of things and MQTT protocol," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pp. 14–16.

[6] Water level monitor - arduino project hub. [Online]. Available: https://create.arduino.cc/projecthub/NewMC/water-level-monitor-b42be9

[7] Turbidity_sensor_sku__sen0189-DFRobot. [Online]. Available: https://wiki.dfrobot.com/Turbidity_sensor_SKU__SEN0189

[8] W. L. Hakim, L. Hasanah, B. Mulyanti, and A. Aminudin, "Characterization of turbidity water sensor SEN0189 on the changes of total suspended solids in the water," vol. 1280, no. 2, p. 022064. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1280/2/022064

[9] ARDUINO UNO WiFi REV2. [Online]. Available: https://store.arduino.cc/products/arduino-uno-wifi-rev2

[10] DS18b20 temperature sensor tutorial with arduino and ESP8266. [Online]. Available: https://create.arduino.cc/projecthub/akarsh98/ds18b20-temperature-sensor-tutorial-with-arduino-and-esp8266-db31aa

[11] In-depth: How water level sensor works and interface it with arduino. [Online]. Available: https://lastminuteengineers.com/water-level-sensor-arduino-tutorial/

[12] Water level sensor liquid water droplet depth detection [water level sensor] - US $1.50 : HAOYU electronics : Make engineers job easier. [Online]. Available: https://www.hotmcu.com/water-level-sensor-liquid-water-droplet-depth-detection-p-113.html

[13] Gravity: Analog turbidity sensor for arduino - DFRobot. [Online]. Available: https://www.dfrobot.com/product-1394.html

[14] MG90s – metal gear micro servo motor. [Online]. Available: https://components101.com/motors/mg90s-metal-gear-servo-motor

[15] Servo motor basics with arduino | arduino documentation. [Online]. Available: https://docs.arduino.cc/learn/electronics/servo-motors

[16] MACFOS. Buy 5v DC noiseless mini submersible pump with 5mm female jack. [Online]. Available: https://robu.in/product/5v-noiseless-mini-submersible-pump-with-dc-5mm-female-jack/

[17] 5v single-channel relay module. [Online]. Available: https://components101.com/switches/5v-single-channel-relay-module-pinout-features-applications-working-datasheet

[18] PLATINIUM HEATER - aquael_com. [Online]. Available: https://www.aquael.com/produkty/aquaristics-us/heaters-us/platinium-heater/

[19] Aquael platinium heater 75 w - doppvärmare. [Online]. Available: https://www.cyberzoo.se/sv/articles/2.48.896/aquael-platinium-heater-75-w-doppvarmare

[20] N. Zlatanov, *Arduino and Open Source Computer Hardware and Software.*

[21] S. C. Pokress and J. J. D. Veiga, "MIT app inventor: Enabling personal mobile computing." [Online]. Available: http://arxiv.org/abs/1310.2830

[22] Z. A. Firmansyah and D. Hirawan, "WATER QUALITY MONITORING ON THE KOI FISH HATCHERY BASED ON INTERNET OF THINGS."

Appendices

.1 Appendix #1

.1.1 Arduino code for the Canny fish bowl

```
#include <SPI.h>
#include <WiFiNINA.h>
#include <ArduinoMqttClient.h>
#include <ArduinoJson.h>
#include <Arduino_LSM6DS3.h>
// Include the libraries we need
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Servo.h>
Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards
int pos = 0;    // variable to store the servo position

int Relaypin= 12; // Define input pin for relay


// Data wire is plugged into port 2 on the Arduino
#define ONE_WIRE_BUS 2

// Setup a oneWire instance to communicate with any
   OneWire devices (not just Maxim/Dallas temperature ICs
   )
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
// Sensor pins
#define sensorPower 7
#define sensorPin A1
// Value for storing water level
int val = 0;
/*
 * The setup function. We only start the sensors here
 */
///////please enter your SSID and Password
char ssid[] = "Qwe";        // your network SSID (name)
char pass[] = "123456789";    // your network password (
   use for WPA, or use as key for WEP)
```

```
36 int status = WL_IDLE_STATUS;     // the WiFi radio's
      status
37
38 WiFiClient wifiClient;
39 MqttClient mqttClient(wifiClient);
40
41 // MQTT server setup
42 const char broker[] = "212.237.204.254";
43 int        port     = 1883;
44 const char topic[]  = "G4";
45
46 //Test varaibles
47
48 String PoParam="";
49
50 const long interval = 1000; //use to change the
      transmition speed
51 unsigned long Millis = 0;
52 String subMessage = "";
53 boolean stat=0;
54 boolean s=0;
55 String subString ="Led␣is␣OFF";
56
57 float x, y, z;
58
59 void setup() {
60
61   //Do not change this part
62   //Initialize serial and wait for port to open:
63   Serial.begin(9600);
64   while (!Serial) {
65     ; // wait for serial port to connect. Needed for
         native USB port only
66   }
67
68   // check for the WiFi module:
69   if (WiFi.status() == WL_NO_MODULE) {
70     Serial.println("Communication␣with␣WiFi␣module␣failed
         !");
71     // don't continue
72     while (true);
73   }
74
```

```
75  // attempt to connect to WiFi network:
76  while (status != WL_CONNECTED) {
77    Serial.print("Attempting to connect to WPA SSID: ");
78    Serial.println(ssid);
79    // Connect to WPA/WPA2 network:
80    status = WiFi.begin(ssid, pass);
81    // wait 10 seconds for connection:
82    delay(10000);
83
84  }
85
86    while (!mqttClient.connect(broker, port)) {
87    Serial.print("MQTT connection failed! Error code = ")
        ;
88    Serial.println(mqttClient.connectError());
89  }
90
91  // subscribe to a topic
92  mqttClient.subscribe(topic);
93
94  // you're connected now, so print out the data:
95  Serial.println("You're connected to the network");
96  Serial.print("SSID: ");
97  Serial.println(WiFi.SSID());
98
99
100   if (!IMU.begin()) {
101    Serial.println("Failed to initialize IMU!");
102
103    while (1);
104  }
105  Serial.print(IMU.accelerationSampleRate());
106
107  //Start Coding
108
109  //End Coding
110  sensors.begin();
111  Serial.begin(9600); //Baud rate: 9600
112  // Set D7 as an OUTPUT
113  pinMode(sensorPower, OUTPUT);
114
115  // Set to LOW so no power flows through the sensor
116  digitalWrite(sensorPower, LOW);
```

```
117   myservo.attach(9);  // attaches the servo on pin 9 to
          the servo object
118   // put your setup code here, to run once:
119   pinMode(Relaypin, OUTPUT); // Define the Relaypin as
          output pin
120 }
121
122 void loop() {
123   //
124   StaticJsonDocument<200> OutMes;
125   StaticJsonDocument<200> inMes;
126   //Utelization of own defined function bottom.
127   getIMU();
128   //Data Serialization example.
129     OutMes["ID"]=("Device_2");
130     OutMes["Temp"]=temp();
131     OutMes["Turbidity"]=tur();
132     OutMes["WaterLevel"]=wl();
133     OutMes["Time"]=get_time();
134     OutMes["LedStatus"]=(subString);
135    serializeJson(OutMes, PoParam);
136    Serial.println(PoParam);
137
138    //Read incoming message
139   int messageSize = mqttClient.parseMessage();
140   if (messageSize) {
141     subMessage = "";
142
143     // Use the Stream interface to print the contents
144     while (mqttClient.available()) {
145       subMessage = subMessage + (char)mqttClient.read();
146     }
147     //Serial.println(subMessage);
148
149     DeserializationError error = deserializeMsgPack(inMes
          , subMessage);
150
151     //Taking Action acording to subscribed message
152     if(subMessage == "Pump") {
153        if(s==1){
154         digitalWrite(Relaypin, LOW);
155         s=0;
156         }
```

```
157       else if(s==0){
158            digitalWrite(Relaypin, HIGH);
159            s=1;
160           }
161
162     }
163     if(subMessage == "ONOFF") {
164       if(stat==1){
165         digitalWrite(LED_BUILTIN, LOW);
166         stat=0;
167         subString = "Led is OFF";
168         }
169         else if(stat==0){
170            digitalWrite(LED_BUILTIN, HIGH);
171            stat=1;
172            subString = "Led is ON";
173           }
174
175     }
176     if(subMessage == "Feed") {
177       rotateservo(180);
178
179     }
180
181   }
182
183   //Publishing measured data through MQTT
184    if(millis()-Millis>interval){
185        mqttClient.beginMessage(topic);
186         mqttClient.print(PoParam);
187         mqttClient.endMessage();
188         Millis=millis();
189   }
190
191   delay(10);
192   PoParam="";
193
194 }
195 float temp(){
196   // call sensors.requestTemperatures() to issue a global
        temperature
197   // request to all devices on the bus
198   //Serial.print("Requesting temperatures...");
```

```
199   sensors.requestTemperatures(); // Send the command to
        get temperatures
200   //Serial.println("DONE");
201   // After we got the temperatures, we can print them
        here.
202   // We use the function ByIndex, and as an example get
        the temperature from the first sensor only.
203   float tempC = sensors.getTempCByIndex(0);
204
205   // Check if reading was successful
206   if(tempC != DEVICE_DISCONNECTED_C)
207   {
208     //Serial.print("Temperature for the device 1 (index
          0) is: ");
209     Serial.print("Temperature: ");
210     Serial.println(tempC);
211   }
212   else
213   {
214     Serial.println("Error: Could not read temperature
          data");
215   }
216   return tempC;
217 }
218
219 float tur(){
220   int sensorValue = analogRead(A0);// read the input on
        analog pin 0:
221   float voltage = sensorValue * (5.0 / 1023.0); //
        Convert the analog reading (which goes from 0 -
        1023) to a voltage (0 - 5V):
222   float ntu = (voltage-3.994)/(-0.0008); //converting the
         voltage to NTU
223   Serial.print("voltage: ");
224   Serial.println(voltage); // print out the value you
        read:
225   delay(500);
226   Serial.print("NTU: ");
227   Serial.println(ntu);
228   return ntu;
229 }
230 int wl(){
231   //get the reading from the function below and print it
```

```
232   int level = readSensor();
233
234   Serial.print("Water level: ");
235   Serial.println(level);
236
237   delay(100);
238   return level;
239 }
240
241 //This is a function used to get the reading
242 int readSensor() {
243   digitalWrite(sensorPower, HIGH);  // Turn the sensor ON
244   delay(10);                        // wait 10 milliseconds
245   val = analogRead(sensorPin);      // Read the analog
        value form sensor
246   digitalWrite(sensorPower, LOW);   // Turn the sensor
        OFF
247   return val;                       // send current reading
248 }
249 float get_time(){
250   Serial.print("TIME: ");
251   Serial.println(millis());
252   return millis();
253 }
254 void rotateservo(int pos1)
255 {
256   for (pos = 0; pos <= pos1; pos += 1) { // goes from 0
        degrees to 180 degrees
257     // in steps of 1 degree
258     myservo.write(pos);              // tell servo to go
          to position in variable 'pos'
259     delay(15);                       // waits 15ms for
          the servo to reach the position
260   }
261   for (pos = pos1; pos >= 0; pos -= 1) { // goes from 180
        degrees to 0 degrees
262     myservo.write(pos);              // tell servo to go
          to position in variable 'pos'
263     delay(15);                       // waits 15ms for
          the servo to reach the position
264   }
265 }
266 void water_pump(){
```

```
267   // put your main code here, to run repeatedly:
268   digitalWrite(Relaypin, HIGH); // Sends high signal
269   delay(1000); // Waits for 1 second
270   digitalWrite(Relaypin, LOW); // Makes the signal low
271   delay(1000); // Waits for 1 second
272 }
273
274 // This is an example how to create your ovn functiond
      feel free to create your own functions for each sensor
      /actuator.
275 void getIMU(){
276   // This is en example of measuring acceleration from
        inertial measurement unit (IMU)
277   if (IMU.accelerationAvailable()) {
278     IMU.readAcceleration(x, y, z);
279   }
280 }
```

.1.2 Web application HTML code.

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in
   Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
  <head>
      <script type="text/javascript" src="https://gc.kis.
         v2.scr.kaspersky-labs.com/FD126C42-EBFA-4E12-
         B309-BB3FDD723AC1/main.js?attr=
         HYvv4K4KNewbB4X0PGAcuv2LEMSdd3-
         Qntk0P5tg8jOy_rSHBSFjc9iqqb42vHSCtknvU40T6UEnU3DfzlFOj0OD44qml
         -pue2jYzYuCdjvI39Q-
         lMpMHp9UFbKdNLVjYCvFJolqaytdX_vBfmJgB-1
         rmKJE2qbsGwN4bzDzn4SdJuT01K22V0up3FZ25DTvirXg0Lgqrc9Wq5uKhMjCy
         -LQK8zYKnPrgUBJ3yPwAZdvmmkytf_BDt-19
         U1cexmnWyHQdY5bGzpeAoXEMrd3IH2k8Lyu8XDCsV51Bd1E9ckwAYnrfBCWLM3
         -
         of5iqxNjJcDOaZBydhYupEGjHs9JNudbZ6CY8SIzROlgnYdP1poNgeu7_owElp
         -GMGN3q-HsU6g32BxOj2d8lp7VhP7Dd1PBQP-
         FIHRb7eOQHkyvXLJQj7RQ" charset="UTF-8"></script
         ><link rel="stylesheet" crossorigin="anonymous"
         href="https://gc.kis.v2.scr.kaspersky-labs.com/
         E3E8934C-235A-4B0E-825A-35A08381A191/abn/main.
         css?attr=
         aHR0cHM6Ly9jZG4uaW5zdC1mcy1kdWItcHJvZC5pbnNjbG91ZGdhdGUubmV0Lz
         "/>
      <style>
        body {
          background-image: url('web.jpg');
          background-repeat: no-repeat;
          background-attachment: fixed;
          background-size: 100% 100%;
        }
        .title{
            color: #020101;
            text-align: center;
            font-size: 50px;
            margin-top: 1px;
```

```
        }
        .subtitle{
            font-weight: lighter;
            text-align: center;
            color: aliceblue;
        }
        .city {
            background-color: tomato;
            color: white;
            border: 2px solid black;
            margin: 30px;
        }
        .box {
            height: 200px;
            width:  200px
        }
        .container {
            display: grid;
            grid-template-columns: 1fr 1fr 1fr 1fr;
            margin: 30px 0px 0px 0px;
        }
        .container2 {
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            margin: 30px 0px 0px 340px;
        }
        .container3 {
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            margin: 60px 0px 0px 10px;
        }
        .butt {
            border: none;
            color: white;
            padding: 15px 32px;
            text-align: center;
            text-decoration: none;
            display: inline-block;
            font-size: 16px;
            margin: 4px 2px;
            cursor: pointer;
        }
        .text-block2 {
```

```
65                      /* position: absolute; */
66                     /* font-weight: bold; */
67                     font-size: 19px;
68                     bottom: 100px;
69                     right: 220px;
70                     background-color: rgb(0, 0, 0);
71                     color: rgb(249, 245, 245);
72                     padding-left: 60px;
73                     padding-right: 20px;
74                     /* margin: 40px 10px 0px 10px; */
75                     opacity: 1;
76           }
77           .butt1 {background-color: #4CAF50;} /* Green */
78           .butt2 {background-color: #008CBA;} /* Blue */
79           .butt3 {background-color: #6a1ec7;} /* orange */
80       </style>
81
82           <title>ET1544 Students</title>
83           <meta charset="windows-1252">
84           <meta name="viewport" content="width=device-width
                ,␣initial-scale=1.0">
85           <script src="mqttws31.js" type="text/javascript"
                ></script>
86           <script type="text/javascript">
87           var client;
88           var reconnectTimeout;
89           var host="212.237.204.254";
90           var port=9001;
91           var topic="G4";
92           var jasonData;
93
94
95           client = new Paho.MQTT.Client(host, port, "");
96
97
98           // set callback handlers
99           client.onConnectionLost = onConnectionLost;
100          client.onMessageArrived = onMessageArrived;
101
102          // connect the client
103          client.connect({onSuccess:onConnect});
104
105
```

```
106         // called when the client connects
107         function onConnect() {
108           // Once a connection has been made, make a
                subscription and send a message.
109           console.log("onConnect");
110           client.subscribe(topic);
111
112         }
113
114         // called when the client loses its connection
115         function onConnectionLost(responseObject) {
116           if (responseObject.errorCode !== 0) {
117             console.log("onConnectionLost:"+
                  responseObject.errorMessage);
118
119           }
120         }
121
122         // called when a message arrives
123         function onMessageArrived(message) {
124           console.log("onMessageArrived:"+message.
                payloadString);
125           try{
126             jasonData=JSON.parse(message.payloadString);
127             document.getElementById("temp").innerHTML=
                  jasonData.Temp;
128             document.getElementById("turbinfo").innerHTML
                  =jasonData.Turbidity;
129             document.getElementById("wl").innerHTML=
                  jasonData.WaterLevel;
130             document.getElementById("clk").innerHTML=
                  jasonData.Time;
131           }
132           catch(e){}
133
134
135           document.getElementById("button").onclick =
                function() {SendOnOFF()};
136           document.getElementById("button2").onclick =
                function() {SendFeed()};
137           document.getElementById("button3").onclick =
                function() {Sendpump()};
138
```

```
139
140
141
142             function SendOnOFF(){
143                 client.subscribe(topic);
144                 message = new Paho.MQTT.Message("ONOFF");
145                 message.destinationName = topic;
146                 client.send(message);
147             }
148             function SendFeed(){
149                client.subscribe(topic);
150                message = new Paho.MQTT.Message("Feed");
151                message.destinationName = topic;
152                client.send(message);
153             }
154             function Sendpump(){
155                 client.subscribe(topic);
156                 message = new Paho.MQTT.Message("Pump");
157                 message.destinationName = topic;
158                 client.send(message);
159             }
160         }
161     </script>
162     </head>>
163         <body>
164            <h1 class="title">Canny Fish Bowl</h1>
165            <h2 class="subtitle">ET1544_H22 Group-g4</h2>
166            <div class="container">
167               <span class="city␣box">
168                  <h2 style="text-align:center;">
                        Temperature in (C) </h2>
169                  <div class="text-block2" id="temp"></
                        div>
170               </span>
171
172               <span class="city␣box">
173                  <h2 style="text-align:center;">
                        Turbidity in NTU</h2>
174                  <div  class="text-block2" id="turbinfo"
                        ></div>
175               </span>
176
177               <span class="city␣box">
```

```
178                     <h2 style="text-align:center;">Water
                           level in cm</h2>
179                     <div  class="text-block2" id="wl"></div
                           >
180                 </span>
181                 <span class="city␣box">
182                     <h2 style="text-align:center;">Time in
                           ms</h2>
183                     <div class="text-block2" id="clk"></div
                           >
184                 </span>
185              </div>
186             <div class="container3">
187                 <button class="butt␣butt1" id="button">
                       LedON/OFF</button>
188                 <button class="butt␣butt2" id="button2">
                       FEED</button>
189                 <button class="butt␣butt3" id="button3">
                       Pump</button>
190             </div>
191         </body>
192 </html>
```