CITS5506 Internet of Things, Semester 2, 2024

Smart Pet Water Feeder
Group 21

Project Report

THE UNIVERSITY OF WESTERN AUSTRALIA

Group Members:
Hanxun Xu (23885505)
Jean Wang (24137024)
Jiashuai Chai (24590681)
Mike Bernal (23952058)
Zongqi Wu (23957505)

TABLE OF CONTENTS

ABSTRACT

IoT technology, known for its accessibility, user-friendly interface, and efficient monitoring, has become a major trend. Pet ownership can be a source of joy in many people's lives, but ensuring pets have access to clean water can be challenging when owners are away. Besides, it is difficult to monitor the pet's water intake feeders, which may cause us to miss the opportunity to detect kidney diseases, diabetes etc. Our project allows owners to provide water to their pets at scheduled times or manually provide water by using our website. The system comprises reservoir, water bowl, and waste tank, controlled by a Raspberry Pi. Additionally, there are weight sensor, and a turbidity sensor trigger automatic water refill or changes when the amount or quality of water drops below acceptable standards. Additionally, the system will track individual water consumption for multiple pets using RFID technology and send alert notifications to their owners when the amount of water intake is abnormal. During the test, the RFID reader identified all tags, the turbidity sensor effectively monitored, water level sensors worked as expected. But the precision of weight sensor needs to be improved. Valve response delays, caused by power fluctuations, may require improve. In summary, our system improves pet water consumption management through automation, individualized tracking, and health monitoring, which can provide convenience and peace of mind to their owners considerably.

Keywords: automictically water refill, Raspberry Pi, water quality sensor, weight sensor, RFID

1. INTRODUCTION AND PROBLEM

1.1. INTRODUCTION

Nowadays, almost all people want to keep a pet in their home as they believe that a pet in house can decrease daily stress, boredom, and loneliness from daily life [1]. From the pie chart in Image 1 below, most of the pet ownership make a positive impact. However, making sure that pets always have access to clean and appropriate water can be a challenge. For instance, when pet owners are away from home, it may be significantly different for them to provide water to their pets. Besides, it is quite inconvenient to monitor the amount and the quality of water when using a traditional water feeder, which might cause some health problems.

Although there are already some automatic pet water feeders, these products usually ignore that monitor the amount of water intake and the quality of water. This project plan to use IoT technique, helping pet owners easily access this kind of data by using our online system. Meanwhile, if the system think that the certain pets' water intake is abnormal, it will send notifications to their owner. Besides, a turbidity sensor has been embedded to our product, if the system believe that the quality of water is not good enough, it will drop the water in the water bowl into waste tank and refill automatically.

This report will be separated into 4 parts, introduction, implement, result and conclusion. First, the current problems in pet water feeding will be illustrated, and then the hardware and software solution, including sensors, cost, and accuracy. After that the testing results and some recommendations for future improvements will be presented. Finally, the system's advantages in providing convenience and health monitoring for pet owners will be emphasized.

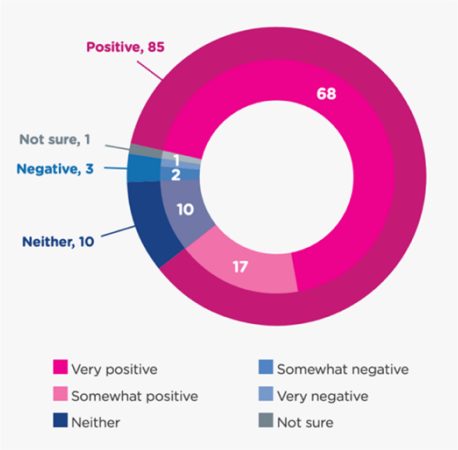


Image 1: Impact of pet ownership on life(%)

1.2. PRIMARY PROBLEM

From the pets' perspective, when their owners are busy or must away their house due to some reasons like working, travelling and so on, pets might lose the access to clean and sufficient water. So that pets tend to be sick, and even suffer from some diseases, such as parasitic diseases, gastroenteritis, etc. Besides, it is difficult for owners to monitor the amount of water intake, which means they might miss the opportunities to detect certain diseases early, such as kidney diseases and diabetes. Once the diseases become serious, pets tend to suffer more severe pain.

1.3. SECONDARY PROBLEM

From the owners' perspective, they need to manually refill their pets' water bowl frequently, which might be time-consuming and inconvenient, especially when trying to keep water clean and fresh all the time. Over time, this repetitive task may cause owners to feel anxiety and stress. Additionally, owners tend to ignore early signs of health problems which may result in severe illnesses. Once their pets become ill, they must pay expensive medical treatment fees, which could affect the owner's budget and the quality of life.

1.4. PROJECT OBJECTIVES

By using IoT techniques, this project aims to facilitate the life of both pets and pets' owners. Smart pet water feeder system can monitor the weight of the water bowl. When the weight falls below the threshold, the system will refill the water automatically, which helps owners avoid inconvenient and time-consuming routine.

Additionally, by using a turbidity sensor, it is possible to monitor the quality of water. If the system determined that the water is not clean enough, it will automatically drop the water into waste tank and refill clean water again, ensuing that pets always have access to clean water.

The multiple pet households are also be considered; the pets' owner can allocate a unique RFID tag to each pet. Additionally, the system can personalize the appropriate water intake for each pet. If any pet's water intake is detected as abnormal, the system will send a notification to their owners. These strategies can not only help owners prevent potential future financial stress by addressing health problems early but also prevent pets from suffering more pain in the future.

2. SOLUTION AND IMPLEMENTATION

To clearly present the hardware solution, software solution and their relationships, the system block diagram is prepared (See image 2 for block diagram).

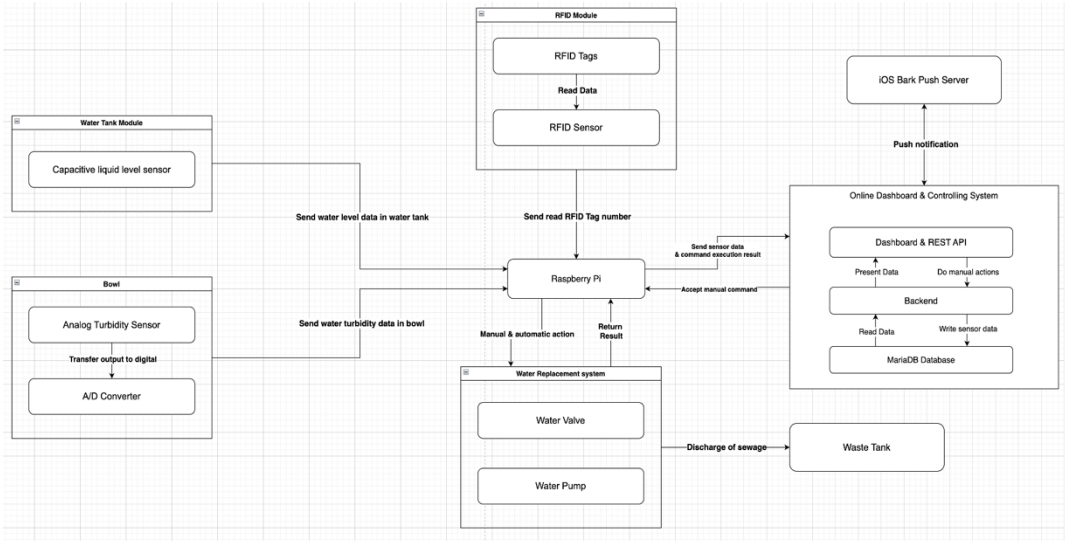


Image 2: System block diagram

2.1. HARDWARE SOLUTION

The water feeder system consists of various water and electrical components that automate the process of refilling a bowl and managing wastewater. It operates based on water level, weight, and water quality sensors, ensuring efficient water supply for pets.

2.1.1 PROTOTYPE OVERVIEW

Multiple components were carefully assembled to ensure seamless integration, allowing sensors to collect data without interruptions. The water bowl, waste tank, and reservoir are made from transparent, non-metallic bottles, enhancing the turbidity and non-contact capacitive liquid level sensor's ability to accurately measure water levels and quality. The clear containers improve visibility, making it easier to monitor water levels and quickly detect potential issues with the water flow or contamination. Additionally, the solenoid valves feature built-in filters to block debris, protecting the water pump. The pump itself includes a built-in check valve that prevents backflow, ensuring that the water remains clean.

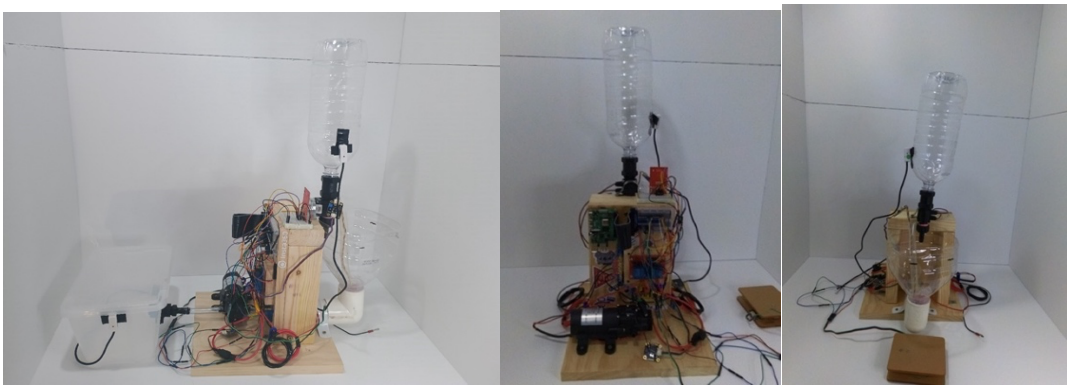


Image 3, 4 and 5: Water Feeder Prototype (side, back, front)

2.1.2. COST

The hardware list table includes components that are already available for the project, including the Raspberry Pi 4B, Micro SD card, plywood, water bottles, PVC accessories such as Teflon Tape, Silicone, Screws, various wires, brackets, the water pump, PVC cement, hose, and various power management accessories including a 12V 10A DC adapter, 5.5V 3.3A DC Adapter, an HDMI Cable, various cables (cobbler and 18-22 AWG wires) and a 5A fuse.

ID	Component	Quantity	Unit Price	Supplier/Source
1	Raspberry Pi 4B	1	$62.00	Supplied by M. Bernal
2	12V 10A adapter	1	$10.00	Supplied by M. Bernal
3	5.5V 3.3A adapter	1	$10.00	Supplied by M. Bernal
4	4 Channel Relay module	1	$12.95	Altronics
5	Bus bar	2	$10.00	Supplied by M. Bernal
6	Turbidity sensor	1	$15.75	Core Electronics
7	Capacitive Liquid Level Sensor	2	$17.40	Core Electronics
8	HX711 Beam Load Cell	1	$6.15	Core Electronics
9	Waste Tank Container	1	$4.20	Bunnings
10	Solenoid Valve (Pressurized)	1	$18.65	Core Electronics
11	RFID Module	1	$15.95	Altronics
12	ADC Converter	1	$25.70	Core Electronics
13	Gravity-fed Solenoid Valve	1	$27.00 (17.95 USD)	eBay
14	Cloud Server	1	$15.00	Binarylane.com.au
	Total Cost:	**$ 278.15**		

Table 1: Hardware List

2.1.3. AVAILABILITY

The availability of component prototypes is listed below, with a scale of 1 to 3 indicating the ease of procurement, where 1 is the easiest and 3 is the hardest.

ID	Component	Description	Availability
1	Water Bottles (Reservoir/Bowl)	Highly available in local grocery stores.	1
2	Pressurized Solenoid Valve	Nationally available at some local reticulation or major electronic stores.	2
3	PVC Fittings (overall)	Hard to procure; the adapter style needed for the prototype is either not commonly used or infrequently stocked.	3
4	Sensors	Highly available from major electronic stores.	2
5	Gravity-fed Solenoid Valves	The hardest to procure due to being a specialized component, which requires outsourcing from overseas (USA).	3

Table 2: Ease of procurement

2.1.4. SIZE

The following table shows the key components of prototype, and their capacity. The components are classified as Liquid Containers and PVC Fittings.

ID	Category	Components	Dimension
1	Liquid Containers	Reservoir	1.5 litres capacity; 100mm(W) x 100mm (L) x 280mm(H)
2		Bowl	2 litres capacity; 150mm(W) x 150mm (L) x 160mm(H)
3		Waste tank	10 litres capacity; 195mm(W) x 280mm (L) x 145mm(H)
4	PVC Fittings	Reservoir solenoid valve	½ "or < 15mm male (threaded)
5		Reservoir adapter	¾ "or < 20mm female (threaded) to female (threaded)
6		Bowl adapter	1.57 "or 40mm female (non-threaded) to female (non-threaded)
7		Bowl reducer	40mm to 20mm female (non-threaded) to female (threaded)
8		Bowl reducer / hose adapter	20mm female (threaded) to 10mm male
9		Waste tank adapter	½ "or < 15mm female (threaded) to female (threaded)
10		Waste tank/ hose adapter	14mm male to male (threaded)
11		Vinyl tubing	10mm
12		Overall Dimension:	290mm(W) x 550mm (L) x 680mm(H)

Table 3: Components dimensions

2.1.5. POWER REQUIREMENT

The following table shows the power consumption of each component in the prototype, and its power requirements.

ID	Electronic Component	Voltage (V)	Current (A)	Power (W)	Description
1	Raspberry Pi 4B	5V	3A	15W	Requires a 5V 3A power supply for stable connection.
2	4-Channel Relay Module	5-12V	-	Low	Control and routes 12v components input/output. Requires 12v DC adapter.
3	Water Pump Motor	12.6V	3.5A	44.1W **87W** (peak)	Controlled via Relay module, used for pressurizing water to the bowl valve.
4	Reservoir Solenoid Valve	12V	0.33A	4W	Gravity-fed system, controlled via relay module.
5	Bowl Solenoid Valve	12V	0.33A	4W	Pressurized by the water pump, controlled via relay module.
6	Non-contact Liquid Sensors (x2)	5-12V	0.02A (each)	0.2W (total)	Requires low power for liquid level detection.
7	RFID Module	2.5V-3.6V	0.1A	0.2W	Average current is below 100mA.
8	Turbidity Sensor	5V	0.04A	0.2W	Requires low power for water quality monitoring.
9	Weight Sensor	3.3V-5V	<0.02A	<0.1W	Requires low power for measuring bowl weight.
		Total Power:		**67.88W**	
		Total Current:	**7.36A**		

Table 4: System Power Consumption

The power is calculated using the formula in **Equation 3** below.

$$P = V * I$$

Equation 3: Power consumption formula

Where:

- **P** is the power in watts,
- **V** is the voltage in volts,
- **I** refer to the current in amperes. [2]

The total power consumption of **67.88W** and current drawn of **7.36A** are within the limits of 12V 10A power supply, which can provide up to 120W and 10A which does not overload the system. The water pump, which draws 3.5A, is protected by a 5A fuse, allowing for safe operation while providing protection against overcurrent.

Residential pricing Show less

All prices below are current as 1 July 2023.

Residential electricity tariff: A1 and A2

Tariff	Charges	1 July 2023	Current 1 July 2024
A1 & A2	Supply charge - per day	$1.1046	$1.1322
	Supply charge for additional homes - cents per day	43.9179	45.0158
	Electricity charge - cents per unit	30.812	31.5823

Image 6:Household electricity pricing (Residential) [3]

The daily cost is calculated using the formula:

$$Cost\ per\ hour\ =\ Power(kWh)\ *(cents\ per\ kWh)$$
$$Cost\ per\ hour\ =\ 0.06788 * 31.5823$$
$$Cost\ per\ hour\ =\ 0.021438$$

The cost of operating the water feeder is **$0.021438** or 2.1438 cents per hour or 51.45 cents per day.

2.1.6. MEASUREMENT ACCURACY

A. HX711 weight sensor

The HX711 weight sensor by DFRobot provides accurate weight measurements up to **10 kilograms**, with a synthesis deviation of less than **±0.2%** g. To ensure precise measurements, the module was calibrated using the built-in program available in the sensor's signal adapter board, which can be accessed at DFRobot's GitHub repository [4]. To further improve accuracy, we collect multiple weight readings and use a median to filter outliers. Any readings that fall outside the specified range are discarded, and the remaining valid readings are averaged to produce a more accurate result. This method helps eliminate outliers caused by the design constraints of our prototype and the sensor's limitations, resulting in more consistent and reliable measurements.

2.2. COMPONENT EXPLANATION

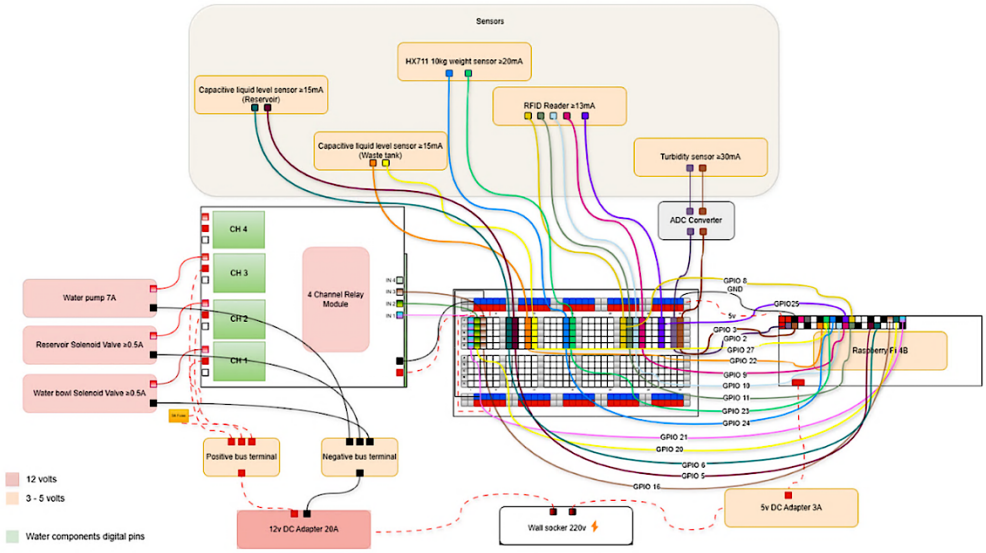


Image 7: Circuit Diagram

The system operates using 5V and 12V power supplies. The 12V supply is routed through bus terminals to distribute power to the pump and solenoid valves. A 4-channel relay module, controlled by the system, manages the routing of control signals to these components, allowing for water flow regulations.

The 5V power supply is dedicated to the sensors and control functions, keeping signals clear by isolating them from the higher voltage components. By keeping the power sources separate and directing signal routing, the system ensures stable and efficient operation for both low and high-voltage components.

A 400-tie point breadboard is utilized to provide flexible and temporary connections during the prototyping phase, allowing for easy integration, testing, and modification of components.

2.2.1. SENSORS

The Smart Pet Water Feeder utilizes sensors to monitor and control the water supply:

1. **Turbidity Sensor (SEN0189)**: Measures if water is contaminated by detecting suspended particles, like dust and soil. It helps assess water quality and triggers water changes when necessary. It is an analog sensor; hence A/D converter ADS1115 is required to read the data.
2. **Non-contact Capacitive Liquid Level Sensors (2 are used)**: Monitor water levels in the reservoir and waste tank without direct contact, ensuring accurate measurements and timely refills or emptying.
3. **HX711 Beam Load Cell**: Measures the weight of the water bowl, allowing precise monitoring of water consumption.
4. **RFID Module**: Identifies individual pets, enabling personalized water consumption tracking for multiple pets.

2.2.2. CONTROL AND COMMUNICATION HUB

Raspberry Pi 4B serves as the central control unit, orchestrating the operation of all components in the system. The one-to-many connection model is adopted. It collects data from multiple sensors, controls the water flow through a relay module, and processes the RFID readings to track individual pet water intake. The RPi also uses its GPIO pins to interface with various sensors mentioned above and control actuators (water pump, solenoid valves).

The Raspberry Pi utilises Wi-Fi since it is typically located at home. The prototype uses direct current with a power adaptor, which is steady and unlikely to cause power issues [5]. In contrast, cellular data plans will bring an ongoing expense to the pet owner [6], and not all pet owners are willing to pay this for the prototype. Besides, Cellular fits outdoor-use products [7]. Most households have internet and Wi-Fi at home, so utilising Wi-Fi is reliable and cost-effective.

2.2.3. HARDWARE IMPLEMENTATION AND CONFIGURATION

The hardware components are integrated as follows:

1. **Water Management**: The reservoir and bowl are connected via PVC pipes, with solenoid valves controlling water flow. The waste tank collects excess or dirty water.
2. **Sensor Integration**: Sensors are strategically placed to monitor water levels, quality, and consumption. The RFID reader is positioned near the bowl for pet identification.
3. **Power Distribution**: A 12V 10A DC adapter powers the system, with appropriate voltage regulation for different components.
4. **Connectivity**: The Raspberry Pi's built-in Wi-Fi module enables cloud connectivity for remote monitoring and control.
5. **Safety Features**: A 5A fuse protects the water pump from overcurrent, while built-in filters in solenoid valves prevent debris from entering the system.

2.3. SOFTWARE IMPLEMENTATION

2.3.1 FRONTEND & BACKEND

The entire software structure could be separated to client-side and server-side. HTTP and MQTT are adopted to connect between them (see Image 8 for system architecture block diagram).

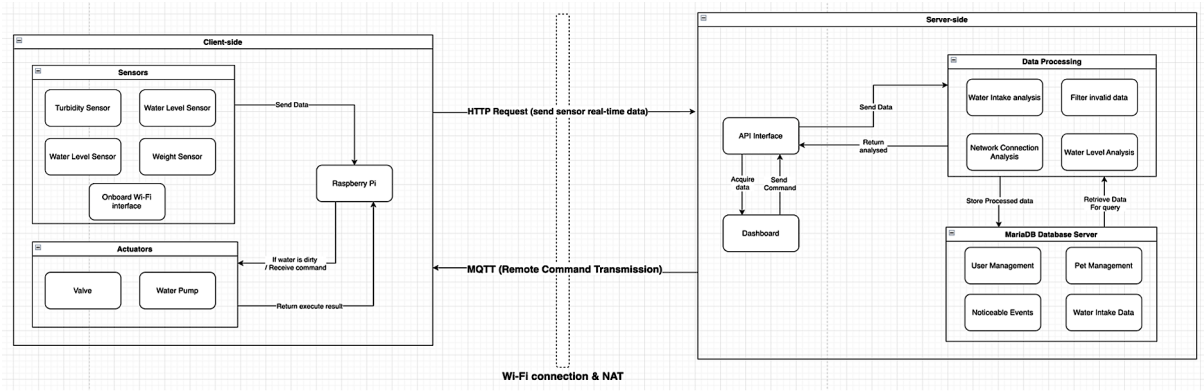


Image 8: System Architecture Block Diagram

2.3.1.1 CLIENT-SIDE PROGRAM OVERVIEW

Client programme is defined in the project as the application that runs on Raspberry Pi. This application manages sensors, remote commands, and local data processing. The one-to-many model is adopted to link between Pi and sensors. Raspberry Pi communicates with several sensors and actuators simultaneously. Python is used in the client application since it works well on Linux, particularly Raspberry Pi. Due to plentiful libraries, Raspberry Pi can easily interact with the sensor. The primary client application entry is in **Appendix A**.

2.3.1.2 SERVER-SIDE PROGRAM OVERVIEW

The server-side program could be separated into frontend and backend. The frontend and backend adopt the MVC architecture (see image 9). The server-side system is built using Flask, Python, SQLAlchemy and Jinja Template.

First, Jinja Template is responsible for the construction of the User Interface (including presenting API data and sending instructions to the prototype); Flask and Python perform data processing and build REST API. When the sensor sends any data through HTTP, Flask and Python will receive and process the data, and finally present the processing results in JSON.

Second, SQLAlchemy builds the database model (DB Model). The DB Model represents the full database's data structure, including tables and columns. DBModel will allow Flask to interact with the database server in OOB. Flask's scalability and security when processing data will improve because the DB Model can be freely updated to meet the newer version software, and because it can act as an intermediary to sanitise data, Flask cannot interact directly with the database server, which reduces SQL Data Injection attacks [8].

What is more, for the database server, we use MariaDB as the backend database server. MariaDB allows fast processing of more data with less resources and provides extensive security [9]. In contrast, MySQL will consume more resources when process data [9]. Therefore, MariaDB becomes a reliable choice, which providing wide compatibility and saving resources on the backend server.

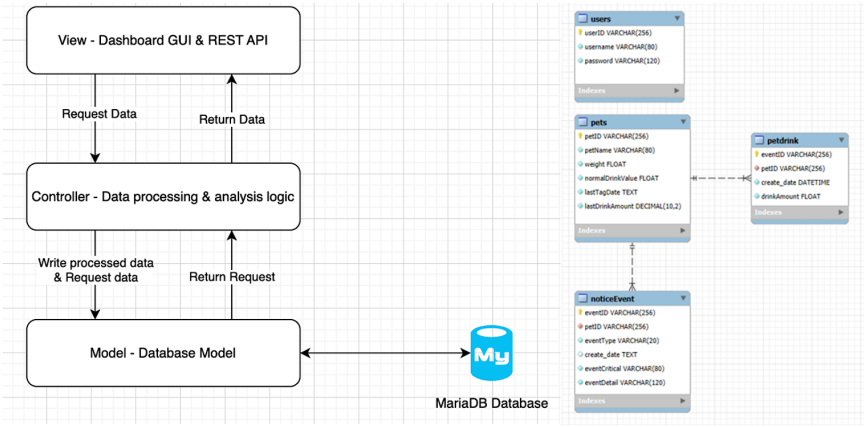
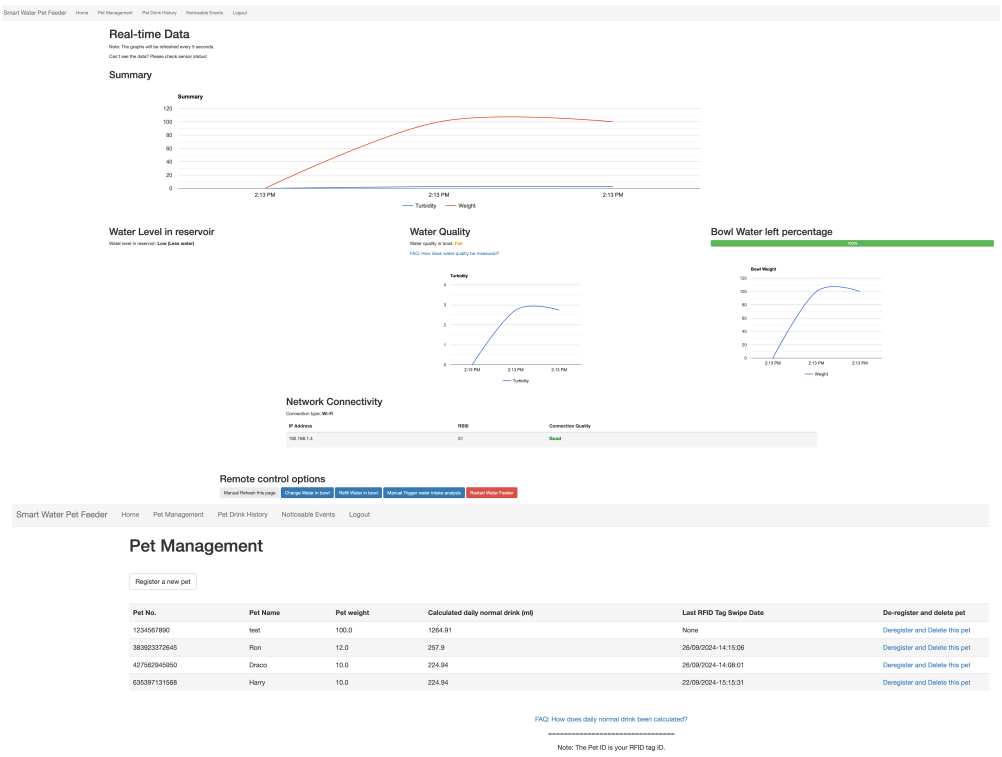


Image 9: MVC System Structure and DB ERD

2.3.1.2 FRONTEND USER INTERFACE

A GUI dashboard appears when users visit the server address in a browser. Water quality, water tank remaining amount, water bowl remaining amount, and prototype's real-time network connection data are analysed in the GUI. Users may also command the water feeder. The water feeder may replenish or replace the bowl's water. The GUI dashboard allows users to restart the water feeder if it malfunctions (see image 10). In addition, the prototype uses RFID to identify pets, so users may add RFID tags via the GUI Interface to identify water-drinking dogs and record their water consumption for study (see image 11). Users may also see pet water consumption (image 12). Also, the programme offers user authentication for safety. The dashboard requires the valid login and password to log in, preventing unauthorised prototype actions (see image 13). Finally, the system gives FAQs (including how to evaluate water quality and compute typical pet consumption) to help pet owners understand how the platform reaches the result.



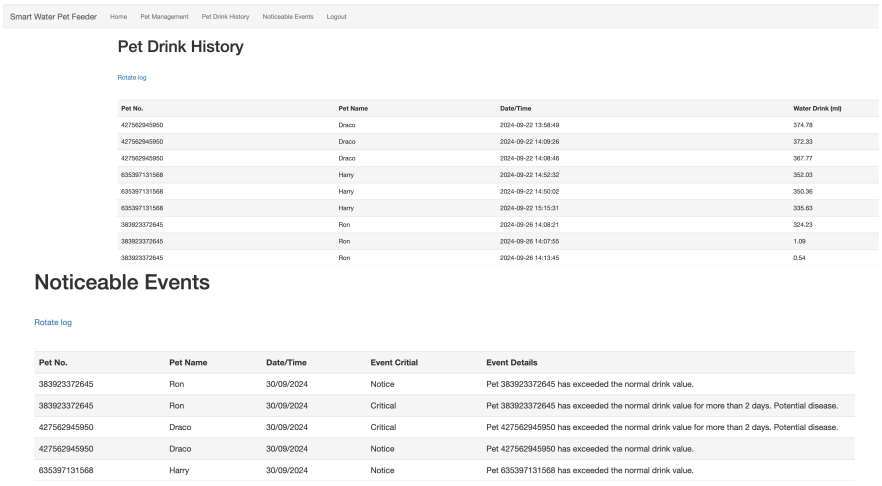


Image 10, 11, 12 and 13: GUI dashboards with example data

2.3.1.3 BACKEND HANDLING LOGIC

As for the backend, all are written in Python and Flask framework. The backend program defines all routes, that is, request URLs. All routes are divided into three parts: API Handling and GUI Handling.

A. API Handling

This Raspberry Pi prototype API Handling component uploads data. All API operations, including receiving and presenting sensor data, will be handled by API Handling. Session dictionary stores all received data. Data is automatically erased when the server is restarted. It helps decrease database server load and old data storage. The Raspberry Pi client will get a JSON http response with a status code regardless of data processing success. Raspberry Pi will need less bandwidth and processing resources for client-server interactions. The status code will also allow the prototype client software identify data processing success and notify the log or user. When data processing is successful, API handling sends **status=True** and status **code=200** to Raspberry Pi (see image 14), which may offer status code or status to assess data processing success. However, if data processing unsuccessful, **status=False** and status **code=400** will be given and allowing the Raspberry Pi application to conduct an exception function, record an error log, or warn the user.

Pretty print ☑


```
{
  "turbiditysensor": "3.34",
  "waterlevelbowlpercentage": "1.85",
  "waterlevelreservoir": "True",
  "waterlevelwaste": "True",
  "weightBowl": 92.676924314158
}
```

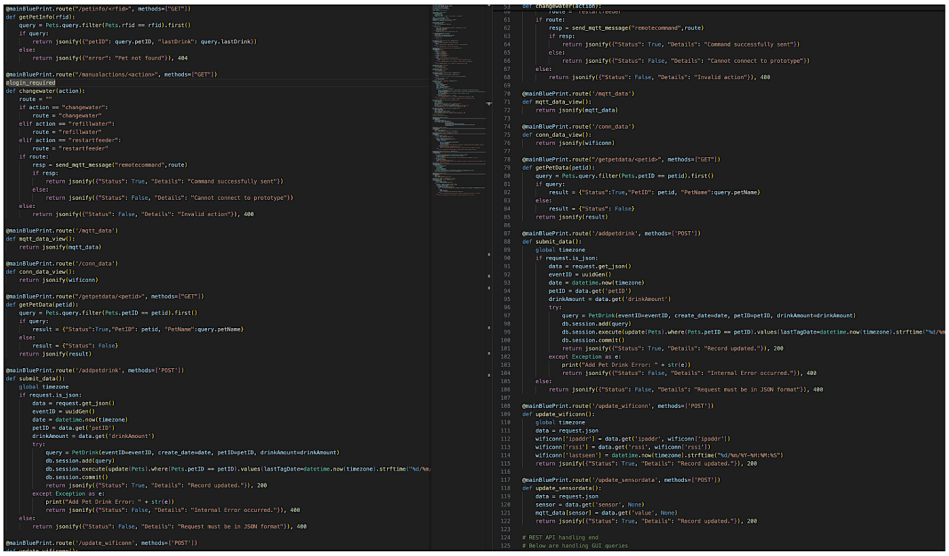



Image 14: server-side backend program and REST API Example

B. GUI Handling

For GUI Handling, this is used to read the latest information, render the Jinja template, and display the data. In other words, before rendering the GUI dashboard, the backend will automatically obtain the latest data in the database through the Model and add it when rendering the Jinja template, so that the user can automatically see all the data after opening the web page. In addition, the front-end dashboard uses JavaScript and AJAX asynchronous processing to allow real-time refresh. In other words, JavaScript and ajax will automatically fetch the latest sensor and Wi-Fi connection data from the API when the user browses the dashboard, and update the dashboard in real time, so that the user can get the latest data and graph without refreshing.

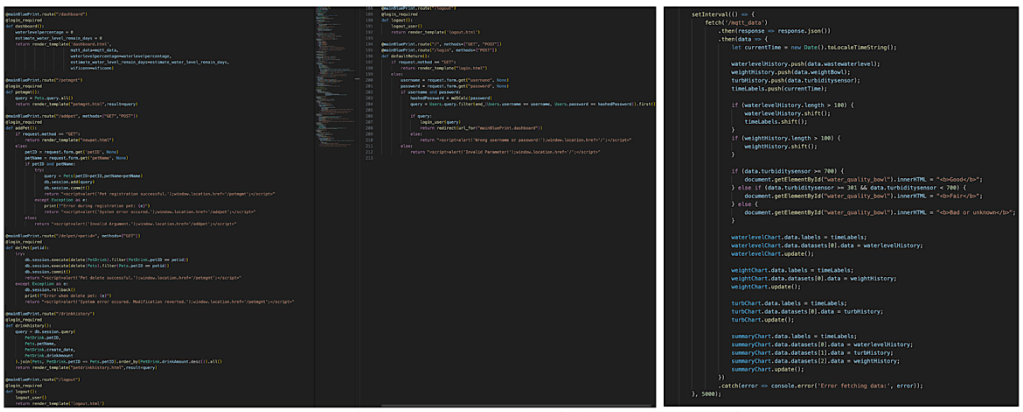


Image 15: GUI handling logics & AJAX API data fetch

2.3.1.4 REMOTE COMMAND TRANSMISSION

In the remote command transmission, the server-side program uses MQTT to send data to the prototype. Therefore, in the API Handling, the MQTT publisher function is added. It implies the server-side program can automatically connect to the MQTT Broker Server and subscribe to the "**remotecommand**" topic (see image 16 working flow). For example, when the user clicks the **reboot water feeder** button in the GUI dashboard, the frontend JavaScript will automatically send the "**restartfeeder**" data to the API "**manualactions**". When the API obtains the data, it will send the "**reboot**" command through the MQTT publisher function. When the prototype obtains the command, it will automatically execute the operation and return “0” when it finished (see image 17 and 18).

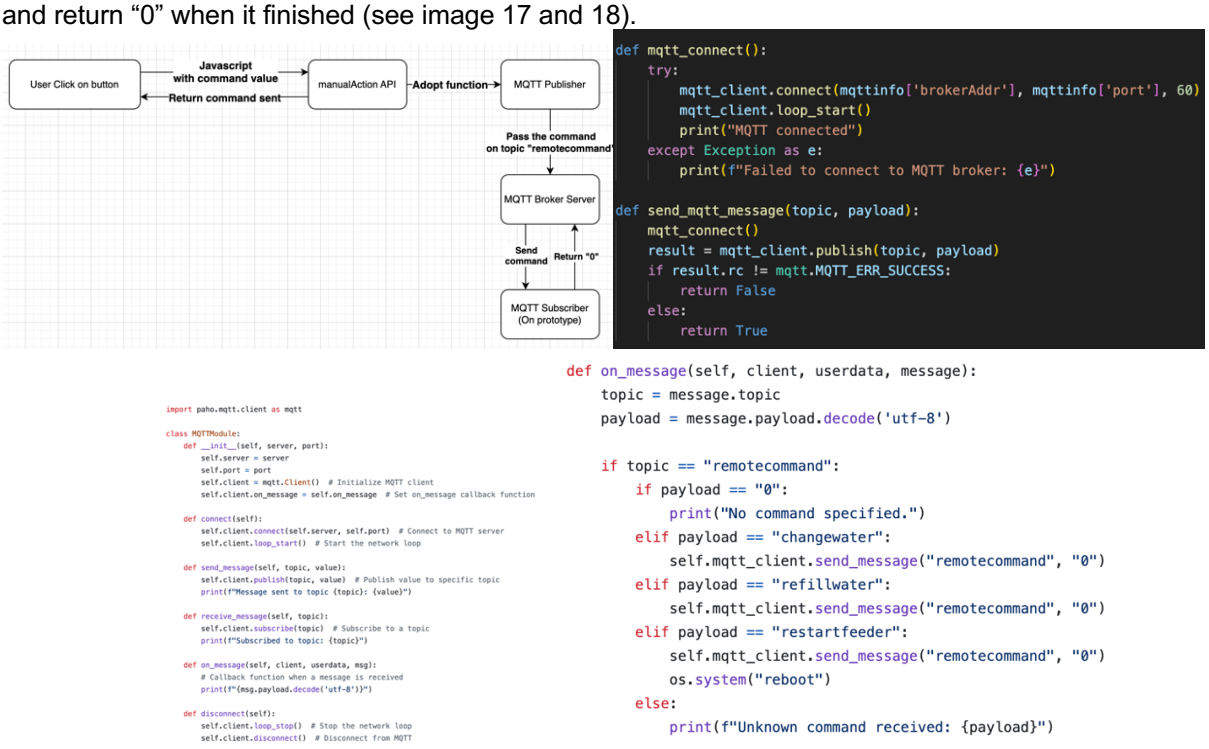


Image 16, 17 and 18: Backend interact with MQTT workflow; Codes on API backend; Codes on raspberry Pi.

2.3.1.5 DATABASE MODEL

For the database model, as mentioned above, it is written in SQLAlchemy (see image 19). The purpose is to allow Flask Program to sanitize the received data through the model and finally store it in the database through OOB. It is presented in the form of class and allows Flask backend to call it directly. This will also help to against False Data Injection (FDI) attack.

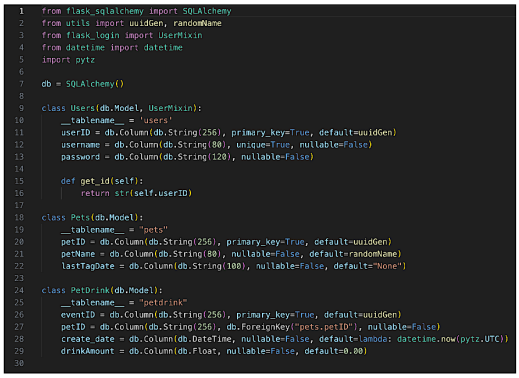


Image 19: Database Model

2.3.1.6 IOS BARK PUSH SERVICE

Bark is an open-source iOS notification push toolkit and broker server. It links the IoT Platform with Apple Push Unit (APU). The REST API receives server messages and push tokens and sends them to the Apple Push Unit. The Apple Push Unit delivers the final message via each device's virtual queue (see image 20 for procedure) [10]. MQTT and it are similar, but also has differences:

- Bark has two push broker servers in each message delivery chain, while MQTT only have one.
- For Bark, each step needs to return acknowledge to the previous step.
- A virtual queue is used for the final message delivery. This ensures that users can receive messages within the offline period when they recover from a network outage (like when get off a plane and turn on cellular data or connect to Wi-Fi).

Smart Pet Water Feeder uses Bark Push Service to provide alerts, such as enrolling or removing a new pet with erratic water consumption (see image 21). Push Tokens are easy to get by download and open Bark from application store. Users must also register push tokens in the dashboard config file. **[Bark_Broker_Server, Token]** is the registration syntax (see image 22 for samples). Users may host Broker Servers for enhance security and privacy. But the Broker address must be specified in the config file.

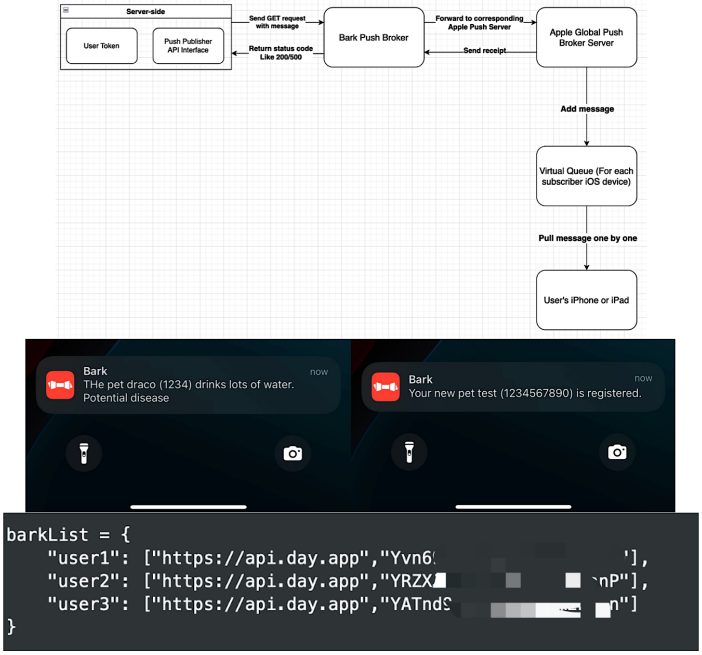


Image 20, 21 and 22: Bark workflow, example of bark, and registered push token in dashboard config

2.3.2 CONNECTION

For connection between prototype and cloud platform, the HTTP and MQTT is adopted (see image 24). Because the environment where the prototype is located (like the pet owner's home) does not necessarily have a public routable IP address due to IPv4 runout from Regional Internet Registry (RIR), it is very likely that a CGNAT address will be assigned to prototype [11] (see image 23 for NAT workflow). Although this helps save IP addresses, it will result in the backend server unable to directly talk to the prototype over HTTP requests. However, one-way transmission (i.e., the prototype sends data to the backend server via HTTP) is possible. Also, after the MQTT connection is established, the backend server and the prototype can communicate directly, regardless of the presence of NAT [12].

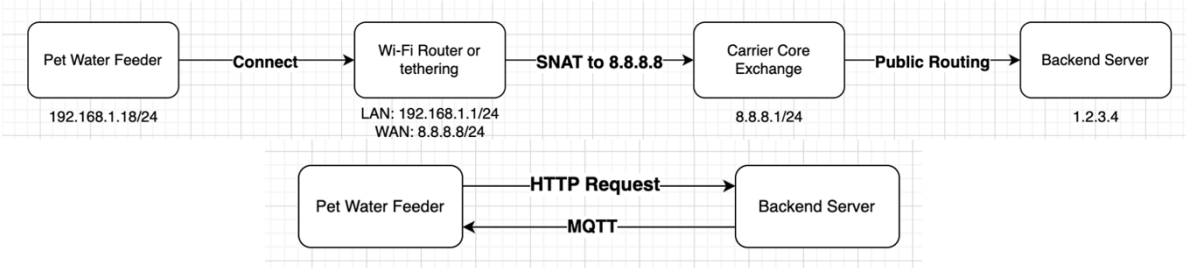


Image 23 and 24: Typical NAT Workflow; Data transmit between prototype and server.

2.3.3 DATA ANALYSIS STRATEGY

2.3.3.1 TURBIDITY ANALYSIS (WATER QUALITY ANALYSIS)

Turbidity, like smoke in the air, makes a liquid cloudy or hazy owing to numerous unseen particles. An important water quality test is turbidity intensity, which measures liquid clarity. It measures the quantity of light dispersed by aqueous substances when illuminated. Higher dispersed light intensity increases turbidity. Nephelometric Turbidity Unit (NTU) measures water turbidity by light [13]. This prototype employs the SEN0189 sensor to adjust input and output voltages depending on water turbidity. Turbidity and voltage formula (see equation 1) [14]:

$$\mathbf{Y = -1120.4x^2 + 5742.3x - 4352.9}$$

Equation 1: NTU calculation formula

Where **x** is equal to the voltage of turbidity sensor, and **Y** is NTU value. Based on WHO guide [15], if NTU value is lower than 1, it can be considered as clean water and safe for pets; The value between 1 and 5 could be considered as fair; if NTU value is higher than 5, which means not suitable for pet to drink (see table 5). If in this situation, the cloud platform will automatically send command to change the water in bowl.

NTU Value	Water quality
x <= 1	Safe to drink
1 < x <= 5	Can drink
x > 5	Not safe to drink

Table 5: NTU value and water quality (Adapt from WHO) [13]

Note: **x** in the **table 5** refers to NTU value.

2.3.3.2 PET WATER INTAKE ANALYSIS

Water intake is very important for pets. Drinking too much or too little water can indicate the presence of disease [16]. Therefore, RFID sensors and weight sensors are installed around the water bowl. When the pet drinks water, the RFID sensor will sense the RFID tag on the pet and report the RFID tag ID and water intake to the cloud platform after drinking. After the cloud platform receives the data, the first step is to store the data in the database for dashboard viewing. Then, the pet weight is read from the database, and their normal daily water intake is calculated using the following formula (see equation 2) [17]:

$$y = 40 * (x)^{0.75}$$

Equation 2: Normal water intake calculation

Where **x** is the pet weight in kilogram, and **y** is referring to normal daily water intake (in ml unit). The platform will calculate the sum of water intake per day per pet (see image 25). If the value higher and lower than **±10%** of normal value more than consecutive 3 days, it can be considered as potential disease for pet [17]. Hence, if the value keeps far higher or far lower (exceeds ±**10%** of normal value) in consecutive 3 days, the platform will notify pet owner related to this concern.

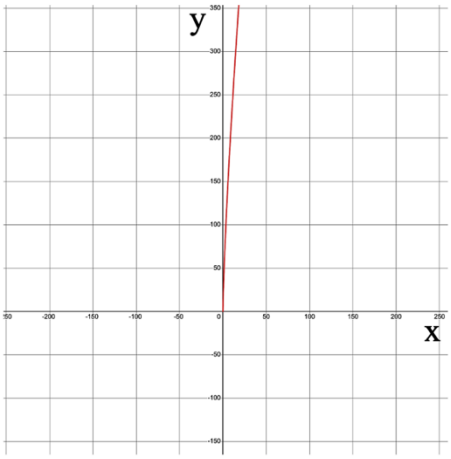


Image 25: Water intake equation graph (relationship between pet weight and water intake)

2.3.3.3 WIRELESS NETWORK CONNECTIVITY ANALYSIS

Wireless Network Connectivity analysis evaluates network signal strength using the RSSI value obtained while connecting to the Wi-Fi network and advises pet owners on water feeder position. This will ensure real-time dashboard updates. The client application receives the real-time RSSI via the Raspberry Pi OS library **iw** and transfers it to the cloud platform via the API. The cloud platform will analyse the information using the threads (in table 6) and classify as Good, Fair, and Poor.

RSSI Value Threads hold	Network Quality Level
x = None or x => 0	Invalid value / Using cable connection
x < 0 and x >= -67	Good
x < -67 and x >= -80	Fair
x < -80	Poor

Table 6: RSSI threads hold and the levels (adapt from Juniper Network) [18]

In the table above, **x** means the latest RSSI value that was reported by client program. Two circumstances occur if Wi-Fi network quality is Poor in three consecutive reports. First, the user may be using the Raspberry Pi's on-board network connector to provide the network, instead of Wi-Fi. The other circumstance is Water feeder site is badly covered. When reporting connection quality, the water feeder IP address is also given (see image 26). In the first scenario, an IP address with odd RSSI (like **-99** or **None**) would exist. As a network cable connection, this does not need to be moved. However, the RSSI value will be **invalid** or **Unknown**. The second circumstance will inform the pet owner to move the water feeder to a Good or Fair signal area.

```
@mainBluePrint.route('/update_wificonn', methods=['POST'])
def update_wificonn():
    global timezone
    data = request.json
    wificonn['ipaddr'] = data.get('ipaddr', wificonn['ipaddr'])
    wificonn['rssi'] = data.get('rssi', wificonn['rssi'])
    wificonn['lastseen'] = datetime.now(timezone).strftime("%d/%m/%Y-%H:%M:%S")
    return jsonify({"Status": True, "Details": "Record updated."}), 200
```

Image 26: API interface of Wi-Fi connection quality update

3. RESULTS AND DISCUSSION

3.1. CALIBRATION

3.1.1 WEIGHT SENSOR CALIBRATION

To accurately measure the water weight, the sensor was calibrated using multiple known weights and the scaling factor is calculated using the following formula (See equation 3):

$$\text{Calibration Factor} = \frac{Sensor\ Output (Raw\ Data)}{Known\ Weight}$$

Equation 3: Calibration Factor calculation formula [19]

The scaling factor of **223.7383270263672** was then used to convert the raw output from the HX711 sensor into real-world weight measurements in grams. The median-based outlier filtering and thresholding is applied to clean the weight sensor readings. This process involves averaging valid readings, skipping invalid values, and removing extreme outliers. The steps are as follows:

1. Collect multiple readings. Gather **n** weight measurements.
2. Sort the readings in ascending order.
3. Calculate the median to set a safe reference point, where the median **M** is (See equation 4):

$$\text{Median} = \begin{cases} X\left[\frac{n+1}{2}\right] & \text{if n is odd} \\ \frac{X\left[\frac{n}{2}\right] + X\left[\frac{n}{2}+1\right]}{2} & \text{if n is even} \end{cases}$$

where
- **n** is number of observations in a data set
- **X** is the ordered/sorted list of values in the data set

Equation 4: Median Formula [20]

4. Set a threshold **T** to filter our readings that are too far from the central value in range **[M – T, M + T]**.
5. Exclude readings that fall outside the threshold range caused by design constraints or sensor limitations.
6. Average the remaining valid readings using the average formula (See equation 5).

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i$$

Equation 5: Arithmetic Mean Formula [21]

3.2. TESTING CASES AND RESULTS

The system components were individually calibrated and then integrated to test and ensure overall functionality and reliability.

- **RFID Reader**: Three different tags were logged in the database for testing and the RFID reader successfully identified all tags. The drinking histories were sent to the database and displayed properly in the dashboard. The average detection range was 2 cm and average detection latency was around 2 seconds, which is considered acceptable for the project prototype.
- **Weight Sensor**: The weight sensor was able to measure the water bowl's weight accurately under stable conditions. However, due to the limitations of the current prototype design, the placement of the weight sensor significantly affects the weight distribution, precision and accuracy of measurements.
- **Turbidity Sensor:** The sensor was tested with water samples mixed with varying amounts of flour to simulate different levels of turbidity. The sensor successfully detected different water quality levels ranging from 0 to 6. The results demonstrated the sensor's effectiveness for real-time water quality monitoring.
- **Pump:** The pump operated as expected, initiating the draining process when the turbidity reading exceeded the threshold of 5 NTU, or receiving the remote control to drain the bowl.
- **Water Level Sensors:** Both the reservoir and waste tank water level sensors accurately detected water levels. The sensor data was transmitted to the database and properly displayed on the system dashboard.

- **Valves:** Both the reservoir and bowl valves performed as expected during individual testing and the bowl valve operated correctly during integrated testing. However, the gravity-fed solenoid valve encountered instability due to power fluctuations. Only 50% of the refilling commands were executed properly as intended.
- **Alert Notifications:** During the testing phase, the alert notification system functioned effectively. All notifications were sent promptly and accurately following the triggering of specified events.

3.3. LIMITATIONS

There are some limitations found during building prototype and testing.

- **Weight Sensor**: current max capacity is only 10kg, if go beyond that, the sensor will return inaccurate data. The reason is that the weight sensor is installed at the bottom of the bowl and the PVC pipe, and the weight sensor cannot distinguish and exclude the weight of the bowl and the PVC pipe. The weight of the bowl and the PVC pipe is relatively heavy, which is about 7kg. Therefore, when water is added to the bowl, it may exceed the upper limit of 10kg and cause inaccurate readings.
- **Turbidity Sensor**: this sensor is very sensitive to light. In other words, the light in the room will affect its reading. The stronger the light, the lower the raw value it produces, which lead to the water is clearer. Therefore, if the light in the room is very strong (such as direct sunlight), it will affect its judgment of water quality and may lead to less accuracy.
- **RFID Reader**: the RFID reader is using on-board RFID antenna. The coverage of RFID sensor through on-board RFID antenna is low. This means that all RFID tags must be placed against the reader to be read, rather than close to it. This results in a slower response and places higher requirements on pets, as they must stand in a specified position to be triggered, which is hard to guarantee.
- **Gravity-fed solenoid valve:** Power fluctuations affect the performance of the gravity-fed solenoid valve. When the power supply to the solenoid coil is inconsistent, it leads to variations in the electromagnetic force that controls the valve, resulting in the valve not opening or closing properly.

3.4. FUTURE WORK

For RFID Sensor, utilising an external antenna instead of a built-in antenna makes the RFID Reader more sensitive and enables for close-range reading, rather than requiring the reader to lean against it. Alternative designs such as integrating a camera with facial recognition technology, can also be considered robust solutions. For Weight Sensor, the range utilised in this prototype is 10kg, although the PVC pipe's weight may alter the reading. To get a more accurate weight reading, use a weight sensor that can handle 20-30kg and tare (subtract the PVC pipe and bowl from the computed total weight). Water flow sensor can be utilized and integrated between a reservoir, bowl, or pump to measure actual water consumption instead of the weight sensor in the future iteration of the water feeder. Besides, an upgrade to a higher-capacity power supply is necessary to ensure the consistent performance of the gravity-fed solenoid valve while minimizing electrical noise and interference. Lastly, for software applications, data transfer security is a problem. API data sent to the cloud is not encrypted using TLS. Plain HTTP is utilised. Although quicker and more convenient than HTTPS or TLS, HTTP is vulnerable to man-in-the-middle attacks [22]. Middlemen may intercept and leak data [22]. Thus, TLS will become a mandated data encryption standard to prevent intermediaries from decrypting and eavesdropping.

4. CONCLUSION

This project successfully developed a smart pet water feeder using IoT technology, achieving the goals set out in our initial design. The system effectively solves the problem of pets not being able to access water when their owners are away and addresses the limitation of existing pet water feeders by monitoring water intake. The system integrates Raspberry Pi, water level sensors, weight sensors,

turbidity sensors, and RFID technology. It can automatically refill or replace water based on the water bowl’s level and quality, and it can notify users to manually refill the reservoir or empty the waste tank based on sensor data. Additionally, using RFID tags, the system accurately tracks the drinking habits of multiple pets, allowing owners to monitor each pet's water intake.

However, the current installation of the weight sensor limits its accuracy, the RFID has a small detection range and slow identification speed, and power fluctuations sometimes affect the valve response time. These issues need to be optimized in future updates.

Overall, despite some areas needing improvement, the system successfully automates water refilling and quality monitoring. When abnormal drinking patterns are detected, it sends alerts to the owner via the Bark app. The system also provides an easy-to-use interface that visualizes the pets' drinking data and supports both automatic and manual refilling via the website. This reduces the owner's burden of constantly refilling water and helps prevent potential health issues in pets, which could lead to expensive treatments if not caught early.

5. REFERENCES

[1] S. Koley, S. Srimani, D. Nandy, P. Pal, S. Biswas, and Dr. I. Sarkar, “Smart Pet Feeder,” Journal of Physics: Conference Series, vol. 1797, no. 1, p. 012018, Feb. 2021, doi: https://doi.org/10.1088/1742-6596/1797/1/012018.

[2] Electrical units, fact sheet, ast1396 | Electrical circuits 5: Electrical units (fact sheet), Department of Education WA, © The University of Western Australia, version 2.0 revised August 2015. Available: https://www.uwa.edu.au/study/-/media/Faculties/Science/Docs/Electrical-units.pdf

[3] “Household Electricity Pricing”, Energy Policy WA, [online]. Available: https://www.wa.gov.au/organisation/energy-policy-wa/household-electricity-pricing. [Accessed: 6 Oct 2024].

[4] DFRobot, "DFRobot_HX711_I2C," GitHub. https://github.com/DFRobot/DFRobot_HX711_I2C (accessed Oct. 14, 2024).

[5]J. Misra, “Powering the IoT System,” Bridgera, Jul. 14, 2017. https://bridgera.com/powering-the-iot-system/ (accessed Sep. 20, 2024).

[6] Lasya K, “Wi-Fi vs Cellular: Which is Better for IoT?,” Vegavid Technology, Aug. 28, 2023. https://vegavid.com/blog/wifi-vs-cellular-which-is-better-for-iot/ (accessed Sep. 20, 2024).

[7] “Wi-Fi vs Cellular Data Connectivity in IoT Deployments,” Caburn Telecom, 2024. https://caburntelecom.com/wi-fi-vs-cellular-data/ (accessed Sep. 20, 2024).

[8] “10 Reasons to love SQLAlchemy,” Pajhome.org.uk, 2015. https://pajhome.org.uk/blog/10_reasons_to_love_sqlalchemy.html (accessed Sep. 20, 2024).

[9]“MariaDB vs MySQL - Difference Between Open Source Relational Databases - AWS,” Amazon Web Services, Inc. https://aws.amazon.com/compare/the-difference-between-mariadb-vs-mysql

[10] “Setting up a remote notification server | Apple Developer Documentation,” Apple Developer Documentation, 2024. https://developer.apple.com/documentation/UserNotifications/setting-up-a-remote-notification-server (accessed Oct. 16, 2024).

[11] rachelle.oconnor, “What is CGNAT - NodeOne Internet,” NodeOne Internet, May 22, 2024. https://nodeone.com.au/news/what-is-cgnat/ (accessed Sep. 20, 2024).

[12] The HiveMQ Team, “MQTT Essentials Part 3: Client, Broker and Connection Establishment,” Hivemq.com, 2015. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/

[13]yida, “What is Turbidity and How to measure Turbidity in water with the Arduino? - Latest Open Tech From Seeed,” Latest Open Tech From Seeed, Jan. 21, 2020. https://www.seeedstudio.com/blog/2020/01/22/what-is-turbidity-and-how-to-measure-turbidity-in-water-with-the-arduino/?srsltid=AfmBOooPd9LaZI2XBmvqdlAZHKQBO2eXoSjkEi2vGqsiFtt3VA03XSAS (accessed Sep. 20, 2024).

[14]“Turbidity_sensor_SKU__SEN0189-DFRobot,” wiki.dfrobot.com. https://wiki.dfrobot.com/Turbidity_sensor_SKU__SEN0189

[15] J. Byrd, “How to Measure Turbidity of Water (5 Methods Explained),” https://waterfilterguru.com/. https://waterfilterguru.com/how-to-measure-turbidity-of-water/

[16] “Reasons why your dog or cat is drinking so much water,” 608vets.com, 2023. https://www.608vets.com/pet-help-advice/general-pet-advice/129-reasons-why-your-dog-or-cat-is-drinking-so-much-water (accessed Sep. 20, 2024).

[17] “How to Measure Your Dog’s Water Intake: A Key Indicator of Health | Companion Animal Veterinary Hospital,” Companionanimalvet.com.au, 2018. https://www.companionanimalvet.com.au/askthevet/dogs/how-measure-your-dogs-water-intake-key-indicator-health (accessed Sep. 20, 2024).

[18] “RSSI values for good/bad signal strength,” Mist, Jul. 09, 2019. https://www.mist.com/documentation/rssi-values-good-bad-signal-strength/

[19] Random Nerd Tutorials, “Arduino with Load Cell and HX711 Amplifier (Digital Scale)” https://randomnerdtutorials.com/arduino-load-cell-hx711/#calibrate-load-cell (accessed Oct. 14, 2024).

[20] M. Farooq, "Median – Definition," ITFeature, [online]. Available: https://itfeature.com/statistics/averages/median-definition/. [Accessed: 14-Oct-2024].

[21] Average Formula," Cuemath. https://www.cuemath.com/average-formula/ (accessed Oct. 14, 2024).

[22] Cloudflare, “Why is HTTP not secure? | HTTP vs. HTTPS,” Cloudflare, 2021. Available: https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/

6. APPENDICES

APPENDIX A: MAIN ENTRY CODE FOR CLIENT PROGRAM

```
import os
from time import sleep, time
import threading
from modules import WaterLevelModule, TurbidityModule, mqttModule, ValveModule, RFIDModule, readWeight, PumpModule
from modules import wificonn, httpModule

class WaterFeeder:
    def __init__(self, waste_water_level_sensor_arg, turbidity_sensor, reservoir_valve, rfid_module, mqtt_client, wifi_conn, httpmodule, readWeight, pump_arg, bowl_valve_arg, reservoir_water_level_sensor_arg):
        self.waste_water_level_sensor = waste_water_level_sensor_arg
        self.turbidity_sensor = turbidity_sensor
        self.reservoir_valve = reservoir_valve
        self.rfid_module = rfid_module
        self.mqtt_client = mqtt_client
        self.wifi_conn = wifi_conn
        self.httpmodule = httpmodule
        self.readWeight = readWeight
        self.monitoring = True
        self.pump = pump_arg
        self.bowl_valve = bowl_valve_arg
        self.reservoir_water_level_sensor = reservoir_water_level_sensor_arg

        # Stop event for controlling the pump flow
        self.stop_event = threading.Event()
        self.valve_lock = threading.Lock()

        # Subscribe to remote command via MQTT
        self.mqtt_client.client.subscribe("remotecommand")
        self.mqtt_client.client.on_message = self.on_message

    def monitor_waste_water_level(self):
        while self.monitoring and not self.stop_event.is_set():
            try:
                print("Checking waste water sensor...")
                waste_water_level = self.waste_water_level_sensor.is_low()
                print(f"Monitoring - Waste water level: {self.waste_water_level_sensor.get_water_level()}")
                waste_tank_sensor_location = self.waste_water_level_sensor.sensor_location
                print(f"Uploading waste water level: {waste_water_level}")
                self.httpmodule.uploadSensorData(f"{waste_tank_sensor_location}", str(waste_water_level))
                sleep(5)
            except Exception as e:
                print(f"Error in monitor_waste_water_level: {e}")

    def monitor_reservoir_water_level(self):
        while self.monitoring and not self.stop_event.is_set():
            try:
                print("Checking reservoir water sensor...")
                reservoir_water_level = self.reservoir_water_level_sensor.is_low()
                print(f"Monitoring - Reservoir water level: {self.reservoir_water_level_sensor.get_water_level()}")
                reservoir_sensor_location = self.reservoir_water_level_sensor.sensor_location
                print(f"Uploading reservoir water level: {reservoir_water_level}")
                self.httpmodule.uploadSensorData(f"{reservoir_sensor_location}", str(reservoir_water_level))
                sleep(5)
            except Exception as e:
                print(f"Error in monitor_reservoir_water_level: {e}")

```

```
                    return

                print("Refilling bowl...")
                self.reservoir_valve.open()  # Open reservoir valve

                start_time = time()
                # Refill for 40 seconds or until threshold is reached
                while time() - start_time < 40:
                    current_weight = self.readWeight.read_weight()
                    print(f"Current bowl weight: {current_weight} grams")

                    if current_weight >= 600:  # Threshold for bowl weight
                        print(f"Bowl refilled to {current_weight} grams.")
                        break
                    sleep(1)

                print("Stopping refill and closing reservoir valve.")
                self.reservoir_valve.close()
                sleep(2)
                print(f"Reservoir valve status: {self.reservoir_valve.get_status()}")

    def on_message(self, client, userdata, message):
        topic = message.topic
        payload = message.payload.decode('utf-8')

        if topic == "remotecommand":
            if payload == "0":
                print("No command specified.")
            elif payload == "changewater":
                self.start_drain_bowl_thread()
                sleep(2)
                self.mqtt_client.send_message("remotecommand", "0") # Drain bowl
            elif payload == "refillwater":
                print("Received command to refill water in the bowl.")
                self.start_refill_bowl_thread()
                sleep(2)
                self.mqtt_client.send_message("remotecommand", "0")
            elif payload == "closevalve":
                print("Received command to close the valve.")
                self.reservoir_valve.close()
                self.mqtt_client.send_message("remotecommand", "0")
            elif payload == "restartfeeder":
                self.mqtt_client.send_message("remotecommand", "0")
                os.system("reboot")
            else:
                print(f"Unknown command received: {payload}")

    def drain_bowl(self):
        print("Draining water...")
        start_time = time()

        self.pump.start()
        self.bowl_valve.open()
```

```
            while time() - start_time < 40:
                if self.stop_event.is_set():
                    print("Stopping pump and closing valve due to interruption...")
                    self.pump.stop()
                    self.bowl_valve.close()
                    return

                if not self.waste_water_level_sensor.is_low():
                    print("Waste tank is full! Please empty the tank.")
                    print("Stopping pump and closing valve...")
                    self.pump.stop()
                    self.bowl_valve.close()
                    return

                sleep(1)

            print("Stopping pump and closing valve...")
            self.pump.stop()
            self.bowl_valve.close()

    def cleanup(self):
            self.monitoring = False
            self.stop_event.set()

            if hasattr(self, 'turbidity_thread') and self.turbidity_thread.is_alive():
                self.turbidity_thread.join()
            if hasattr(self, 'weight_thread') and self.weight_thread.is_alive():
                self.weight_thread.join()
            if hasattr(self, 'rfid_thread') and self.rfid_thread.is_alive():
                self.rfid_thread.join()
            if hasattr(self, 'waste_water_thread') and self.waste_water_thread.is_alive():
                self.waste_water_thread.join()
            if hasattr(self, 'reservoir_thread') and self.reservoir_thread.is_alive():
                self.reservoir_thread.join()
            if hasattr(self, 'drain_bowl_thread') and self.drain_bowl_thread.is_alive():
                self.drain_bowl_thread.join()

            self.waste_water_level_sensor.cleanup()
            self.rfid_module.cleanup()
            self.pump.cleanup()
            self.wifi_conn.stop_real_time_update()

if __name__ == "__main__":
    backendAddr = "203.29.240.135"

    mqtt_client = mqttModule.MQTTModule(server=backendAddr, port=1883)
    mqtt_client.connect()
    turbidity_sensor = TurbidityModule(id="turbiditysensor", sensor_channel=0)
    reservoir_valve = ValveModule(pin=20)
    httpmodule = httpModule.HTTPModule(server=backendAddr)
    wifi_conn = wificonn.WiFiConn(update_interval=5, api_url=f'http://{backendAddr}:5000/update_wificonn')
    waste_water_level_sensor = WaterLevelModule(in_pin=22, mode_pin=27, sensor_location="waterlevelwaste")
    weight_bowl = readWeight(iic_mode=0x03, iic_address=0x64, calibration_value=223.7383270263672)
```

```
        print("Getting 10 weight readings, please wait...")
        weight_bowl.begin()
        rfid_module = RFIDModule(server=backendAddr, water_weight=weight_bowl)
        pump = PumpModule(pin=16)
        bowl_valve = ValveModule(pin=21)
        reservoir_water_level_sensor = WaterLevelModule(in_pin=6, mode_pin=5, sensor_location="waterlevelreservoir")

        try:
            water_feeder = WaterFeeder(
                pump_arg=pump,
                mqtt_client=mqtt_client,
                waste_water_level_sensor_arg=waste_water_level_sensor,
                turbidity_sensor=turbidity_sensor,
                reservoir_valve=reservoir_valve,
                rfid_module=rfid_module,
                wifi_conn=wifi_conn,
                httpmodule=httpmodule,
                readWeight=weight_bowl,
                bowl_valve_arg=bowl_valve,
                reservoir_water_level_sensor_arg=reservoir_water_level_sensor
            )
            water_feeder.start_monitoring()
            while True:
                sleep(1)

        except KeyboardInterrupt:
            print("Exiting program...")

        finally:
            water_feeder.cleanup()
            mqtt_client.disconnect()
```

APPENDIX B: PROJECT RESOURCES

- **Smart Water Feeder -** *Hardware Interface:* https://github.com/mbernal-git/water_feeder_pi
- **Smart Water Feeder –** *Backend and API:* https://github.com/xosadmin/cits5506