# Design Manual

AQUASENSE

Group 06:

- E/20/016 Amarakeerthi H.K.K.G.
- E/20/055 De Silva H.D.S.
- E/20/231 Madhura T.W.K.J.
- E/20/404 Ukwaththa U.A.N.T.

04th July 2025

# Table Of Content

# 1. Introduction

## 1.1 Project Overview

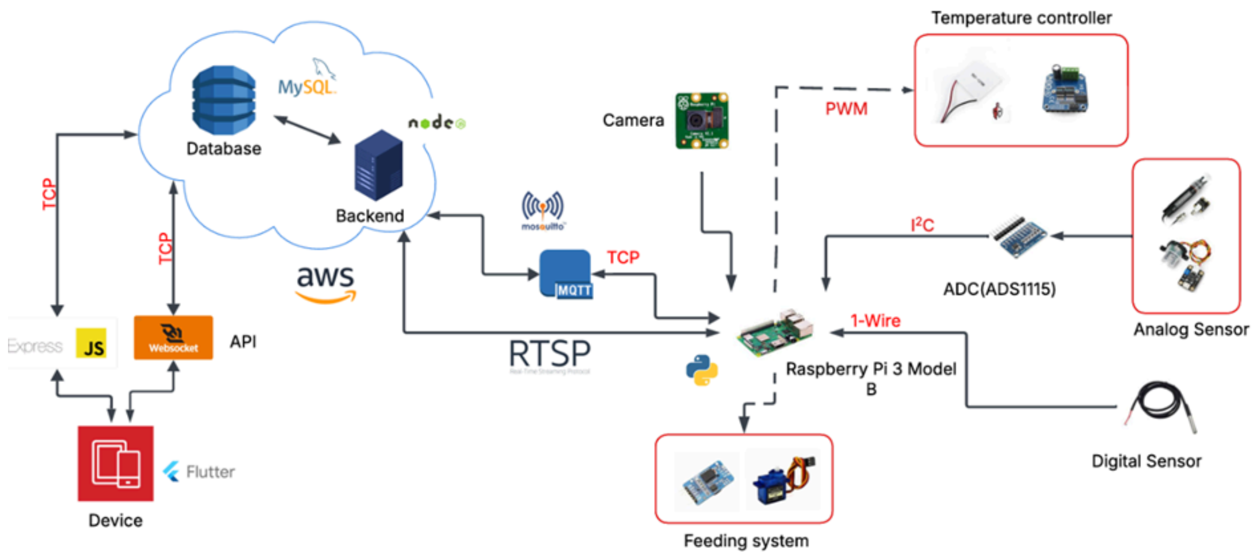
AQUASENSE is a smart aquarium management system designed to automate and optimize the care of home aquariums. The system integrates real-time water quality monitoring, automated feeding, temperature regulation, low oxygen level and abnormal fish behavior detection through fish behavior analysis and alert system. It leverages a Raspberry Pi-based Central Control Unit (CCU), a network of sensors and actuators, and a modern web-based frontend for real-time monitoring, analytics, and control.

## 1.2 Objective and Scope

The objective of this design manual is to guide users, developers, and stakeholders in implementing, deploying, and utilizing the AQUASENSE system. It covers all system components, including architecture, data flow, security, software/hardware infrastructure, calibration, and cloud deployment. The scope includes configuration, deployment, and effective use of the system for reliable, automated aquarium management.

## 1.3 Solution Architecture

The AQUASENSE Smart Aquarium System is designed as a modular, distributed IoT solution that integrates hardware, firmware, and cloud-based software to automate and optimize aquarium management. The architecture is centered around a Central Control Unit (CCU) based on a Raspberry Pi 3B, which interfaces with a network of sensors and actuators, and communicates with a modern web frontend and backend services for real-time monitoring, analytics, and control.



### 1. Device Layer

- **User Devices:**
    - Mobile and web applications (built with Flutter) allow users to interact with the system.
    - Communication with the backend is established via APIs and WebSockets for real-time updates.

### 2. API & Backend Layer

- **API Gateway:**
    - Built using Express.js and JavaScript, it exposes RESTful endpoints and WebSocket channels.

    - Handles TCP connections from user devices for both synchronous and asynchronous communication.
- **Backend Server:**
    - Developed with Node.js, it manages business logic, device control, and data processing.
    - Interfaces with the database (MySQL) for persistent storage of sensor data, user settings, and logs.
    - Hosted on AWS EC2 for scalability and reliability.
- **Database:**
    - MySQL is used for structured data storage, supporting queries for analytics and historical data.

### 3. Cloud & Messaging Layer

- **MQTT Broker (Mosquitto):**
    - Facilitates lightweight, real-time messaging between the backend and the Raspberry Pi (edge device).
    - Ensures reliable delivery of sensor data, actuator commands, and system status updates.

### 4. Edge Device Layer

- **Raspberry Pi 3 Model B:**
    - Acts as the Central Control Unit (CCU) within the aquarium environment.
    - Runs Python scripts to interface with sensors, actuators, and the camera.
    - Communicates with the backend via MQTT (for control/data) and RTSP (for video streaming).
- **Camera Module:**
    - Captures live video of the aquarium, streaming via RTSP for remote monitoring and fish behavior analysis.

**5. Sensor & Actuator Layer**

- **Temperature Controller:**
  - Connected via PWM (Pulse Width Modulation) to the Raspberry Pi for precise temperature regulation.
- **Analog Sensors:**
  - Connected through an ADC (ADS1115) using the $I^2C$ protocol to measure parameters pH and turbidity.
- **Digital Sensor:**
  - Communicates with the Raspberry Pi using the 1-Wire protocol, typically for temperature sensing.
- **Feeding System:**
  - Controlled by the Raspberry Pi, automates fish feeding using a servo motor.

**6. Data Flow & Communication Protocols**

- **Sensor Data Acquisition:**
  - Sensors (analog and digital) send real-time data to the Raspberry Pi.
  - Data is processed and sent to the backend via MQTT.
- **Actuator Control:**
  - The backend or user commands trigger actuators (feeder, temperature controller) through the Raspberry Pi.
- **Video Streaming:**
  - The camera streams live video using RTSP, accessible via the backend or frontend.
- **User Interaction:**
  - Users receive real-time updates and can send commands through the frontend, which are relayed to the backend and then to the Raspberry Pi.

**7. Cloud Integration**

- **AWS Hosting:**
  - The backend and database are deployed on AWS EC2, ensuring high availability and scalability.

### 1.3.1 Communication

AQUASENSE employs MQTT as the primary communication protocol for efficient, real-time, and reliable data exchange between the CCU, sensors, actuators, and the backend. The system's communication design ensures scalability, modularity, and robust event handling.

**Session Management**

- Monitoring sessions are initiated from the frontend, synchronizing time and configuration with the CCU.
- The CCU manages session states, ensuring that data is published and logged only during active sessions.
- Time synchronization at session start guarantees consistent timestamping for all events and logs.

**Topic Subscription**

- **Sensor Data Topics:**

| | | |
|---|---|---|
| - aquasense/sensor/temperature | - | water temperature readings |
| - aquasense/sensor/ph | - | pH level readings |
| - aquasense/sensor/turbidity | - | water turbidity readings |

- **Actuator Command Topics:**

| | | |
|---|---|---|
| - aquasense/actuator/feeder | - | feeding commands |
| - aquasense/actuator/heater | - | temperature control commands |

- **Status and Alert Topics:**

| | | |
|---|---|---|
| - aquasense/status | - | system status updates |
| - aquasense/alerts | - | alert notifications |

- **Camera Stream Topic:**

| | | |
|---|---|---|
| - aquasense/camera/stream | - | live video feed |

**Session Management**

- The frontend initiates monitoring sessions, synchronizing time with the CCU for consistent data logging.
- The CCU manages session states, publishing sensor data and actuator statuses only during active sessions.
- Time synchronization is performed at session start.

**Event Handling**

- Event handling is implemented using MQTT topics:
    - **Sensor Events:** Sensors publish data at regular intervals; the CCU processes incoming data and checks against thresholds.
    - **Actuator Events:** The CCU or frontend issues commands via MQTT; actuators execute actions and report status.
    - **Alert Events:** Abnormal readings or detected fish distress generate alerts, published to the alert topic and pushed to the frontend.
    - **System Events:** Status updates, device errors, and session changes are published to the status topic for system health monitoring.
- The event handling logic is distributed between the firmware (Python scripts on the Raspberry Pi) and the backend (Node.js/Express), with each event type mapped to a specific MQTT topic.

## 1.4 Data Flow

1. Sensors monitor water parameters and publish data via MQTT.
2. The CCU subscribes to sensor topics, processes data, and triggers actuators (temperature control) if parameters exceed thresholds.
3. Actuator commands are sent via MQTT for feeding, temperature.
4. The camera module streams fish behavior data - irregularities and low Oxygen levels generate alerts.
5. The frontend visualizes real-time and historical data, live video stream and manages notifications.
6. Data is optionally backed up to the cloud for remote access and long-term storage.

## 1.5 Security

**1. Authentication and Authorization**

- User Authentication:
  Only registered users can access the system's frontend and control features. This prevents unauthorized individuals from viewing or manipulating aquarium data and controls.
- Authorization:
  Access to sensitive operations (such as changing device settings or triggering actuators) is restricted based on user roles or permissions, ensuring that only authorized users can perform critical actions.

**2. Unique User Identification**

- Personalized Experience:
  Each user is assigned a unique identifier upon registration. This ID is used to associate sensor data, device settings, and historical logs with the correct user account.
- Secure Data Segregation:
  The unique identifier ensures that users can only access their own aquarium data and controls, preventing cross-user data leaks.

This system is designed for domestic users, extensive security measures are not a primary concern.

# 2. Software Infrastructure

## 2.1 Frontend

### 2.1.2 Key Features

- Real-Time Monitoring: Live display of water parameters (temperature, pH, turbidity) with instant updates.
- Automated and Manual Control: Users can trigger feeding, control temperature.
- Live aquarium Visualization: Integration with the camera module allows users to view live video streams and receive alerts for abnormal fish activity and low oxygen levels.
- Alert Notifications: Immediate notifications for unsafe water conditions or system errors.
- Historical Data Visualization: Interactive charts for parameter trends.

### 2.1.3 Technology Stack

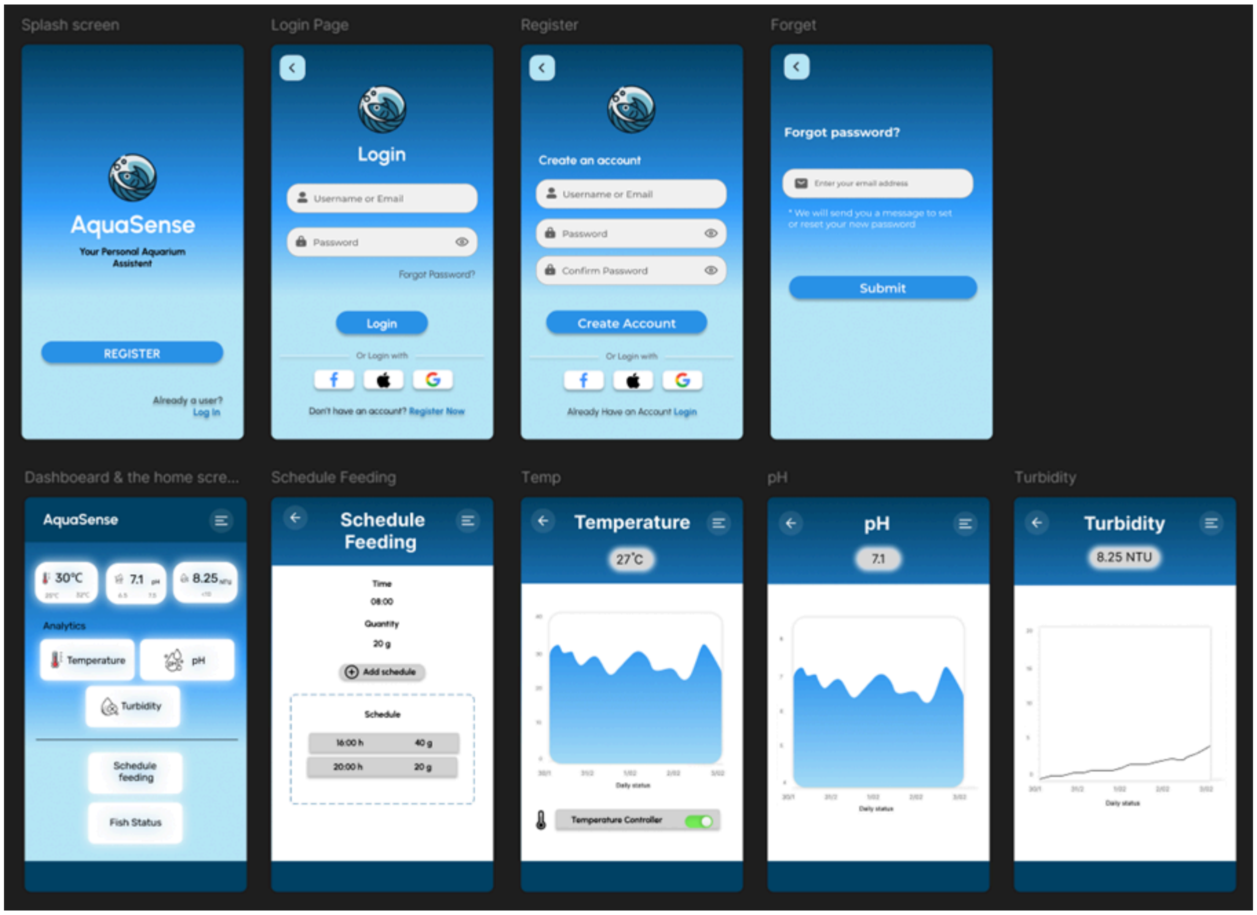
| **Component** | **Technology/Package** | **Purpose/Role** |
|---|---|---|
| Language | Dart | Main language for frontend logic |
| Framework | Flutter | UI for Android |
| Visualization | Flutter charts/widgets | Data display and analytics |

- The use of Flutter and Dart is confirmed by the repository's language statistics and structure.

## 2.1.4 UI Design

**Figma Design:**

The design is structured to provide a seamless user experience, guiding both new and returning users from onboarding through daily aquarium monitoring and control. Additionally, profile screen, Live Video, etc screens are developed during the development.



## 2.2 Backend

### 2.2.1 Introduction

The AQUASENSE backend is responsible for managing data storage, device communication, business logic, and user authentication. It acts as the bridge between the frontend, the Central Control Unit (CCU), and the cloud, ensuring reliable and secure operation of the smart aquarium system.

### 2.2.2 Key Features

- Data Storage: Persistent storage of sensor readings, actuator logs, and user profiles.
- Real-Time Data Processing: Handles incoming data from the CCU and sensors, processes it, and relays updates to the frontend.
- Authentication and Authorization: Ensures only registered users can access and control the system.
- Alert Management: Generates and manages alerts for abnormal conditions or system errors.
- Cloud Integration: Supports deployment on AWS EC2 for scalability and reliability.

### 2.2.3 Technology Stack

| **Component** | **Technology** | **Purpose/Role** |
|---|---|---|
| Runtime | Node.js | Backend server environment |
| Framework | Express.js | RESTful API and routing |
| Language | JavaScript | Main backend logic |
| Database | MYSQL | Flexible, high-performance data storage |

| | | |
|---|---|---|
| Cloud Hosting | AWS EC2 | Scalable backend deployment |

- The backend is primarily implemented in JavaScript (Node.js/Express), with MYSQL for data storage.

### 2.2.4 Modular Architecture and Data Flow

- Route Layer: Handles incoming HTTP and WebSocket requests from the frontend.
- Controller Layer: Validates and processes requests, invoking business logic as needed.
- Service Layer: Implements core logic for device management, data processing, and alerting.
- Database Layer: Stores and retrieves sensor data, user profiles, and system logs.
- MQTT Integration: Subscribes to and publishes messages for real-time communication with the CCU and devices.
- Data Flow:
    1. Sensor data is published via MQTT to the backend.
    2. Backend processes and stores the data, checks for threshold violations.
    3. Alerts are generated if needed and pushed to the frontend.
    4. User commands from the frontend are relayed to actuators via MQTT.

### 2.2.5 RESTful API Endpoints

**Key Endpoint Categories**

1. Sensor Data Retrieval

- Purpose: Fetch the latest readings from all connected sensors, including temperature, pH, and turbidity.
- Usage: Enables the frontend to display real-time water quality data and update dashboards instantly.

2. Actuator Control

- Purpose: Allow users or automated routines to trigger actuators such as the fish feeder, heater/cooler.
- Usage: Supports both manual and scheduled control of aquarium devices, ensuring optimal conditions and timely feeding.

3. Alerts and Notifications

- Purpose: Retrieve recent alerts generated by the system, such as unsafe water conditions, abnormal fish behavior, sensor actuator failures, etc.
- Usage: Keeps users informed of critical events, enabling prompt intervention to maintain fish health and system reliability.

4. Settings and Configuration

- Purpose: Update user preferences or system settings, such as feeding schedules.
- Usage: Allows for personalized and adaptive aquarium management tailored to user needs and specific tank requirements.

5. Historical Data and Analytics

- Purpose: Access historical sensor readings and system logs for trend analysis and long-term monitoring.
- Usage: Facilitates data-driven insights, helping users identify patterns, optimize maintenance, and ensure consistent aquarium health.

6. User Management

- Purpose: Handle user registration, authentication, profile management.
- Usage: Ensures secure access, personalized dashboards, and data isolation for each user.

**Main Backend API Endpoints**

| **Endpoint** | **Method** | **Description** |
|---|---|---|
| /api/sensors | GET | Retrieve the latest sensor readings (temperature, pH, turbidity) |
| /api/actuators/feeder | POST | Trigger the fish feeder actuator |
| /api/actuators/heater | POST | Trigger the heater/cooler actuator |
| /api/alerts | GET | Fetch recent system alerts |
| /api/settings | PUT | Update user or system settings |
| /api/history | GET | Retrieve historical sensor data |
| /api/users/register | POST | Register a new user |
| /api/users/login | POST | User login and authentication |
| /api/users/logout | POST | User logout |
| /api/users/profile | GET | Retrieve user profile and preferences |

# 3. Cloud Deployment Architecture and Procedure

## 3.1 Architecture Overview

The system is a multi-tier application designed to collect and display real-time sensor data.

- Data Source: A Raspberry Pi device reads sensor data (temperature, pH, turbidity) and sends it to the backend via HTTP POST requests.
- Backend Server: A Node.js API, hosted on an AWS EC2 instance, serves as the central backend. It receives data from the Raspberry Pi and connects to the database.
- Database: A MySQL server, running on the same EC2 instance, is used for data persistence.
- Frontend Client: A Flutter mobile application connects to the Node.js API to fetch and display data for the end-user.

## 3.2 AWS EC2 Server Setup

The cloud environment is provisioned on an AWS EC2 instance with the following specifications.

- **Instance Configuration:**
    - AMI (Amazon Machine Image): The instance is launched using the Ubuntu 22.04 LTS image.
    - Instance Type: The t2.micro instance type is used, which is eligible for the AWS Free Tier.
    - Key Pair: A `.pem` key file must be generated and used for secure SSH access.
- **Security Group Inbound Rules:**
    - SSH traffic on port 22 is permitted exclusively from your personal IP address for secure management.
    - HTTP traffic on port 80 is open to all sources (`0.0.0.0/0`) to allow standard web access.
    - Custom TCP traffic on port 3000 is open to all sources (`0.0.0.0/0`) to allow the Flutter app and Raspberry Pi to connect to the Node.js API.
    - MySQL traffic on port 3306 is optionally opened only to your IP address for remote database access if needed.

## 3.3 Server Environment Configuration

Once the EC2 instance is running, it must be configured by connecting via SSH and installing the necessary software dependencies.

1. **System Update**: All system packages are updated to their latest versions.
2. **Core Dependencies**: Node.js (version 18.x), npm, and git are installed on the server.
3. **Database Installation**: The MySQL server package is installed.
4. **Database Security**: The default MySQL installation is secured by running the `mysql_secure_installation` script, which involves setting a root password and applying security defaults.

## 3.4 Database and Backend Deployment

With the environment ready, the database and Node.js application can be deployed.

- **Database Schema Setup**:
    1. After logging into the MySQL server, a new database named `sensordata` is created.
    2. A dedicated user, `sensoruser`, is created for the application and granted all privileges on the `sensordata` database.
- **Backend Code Deployment**:
    1. The application's source code is cloned from its GitHub repository onto the server.
    2. After navigating into the project directory, all Node.js package dependencies are installed using the `npm install` command.
    3. A `.env` file is created to hold sensitive environment variables, such as the database host, user, password, and the server port.

## 3.5 Service Management

To ensure the Node.js API runs continuously, the PM2 process manager is used.

1. **PM2 Installation**: The PM2 package is installed globally using npm.
2. **Application Startup**: The Node.js server is started with PM2, which runs it as a background process.
3. **Persistence**: The PM2 process list is saved, and a startup script is configured to ensure the API automatically restarts if the server reboots.

## 3.6 Client Configuration

Both the IoT device and the frontend application must be configured to point to the deployed server.

- **Raspberry Pi**: The device's script is set to send HTTP POST requests to the server's public IP address on port 3000.
- **Flutter Application**: The API base URL within the Flutter project's source code is set to the server's public IP address on port 3000.

# 4. Hardware infrastructure

## 4.1 Central Control unit

### 4.1.1 Introduction

The CCU of the Smart Aquarium system is built around a **Raspberry Pi 3 Model B**, which functions as the central processing unit and communication gateway. It is responsible for collecting data from various sensors including pH, temperature, turbidity and a camera module and transmitting this data to the cloud or a backend server.

Additionally, the CCU manages control commands for aquarium actuators such as the feeder and Peltier-based temperature control. By using the Raspberry Pi, the system benefits from higher processing power, flexible connectivity options like Wi-Fi and Bluetooth.

### 4.1.2 Hardware Components

- **Raspberry Pi 3 Model B**

Serves as the central controller for the system. It features a 1.4 GHz 64-bit quad-core processor, built-in Wi-Fi, Bluetooth 4.2, and multiple GPIO pins for connecting sensors and actuators. It provides sufficient processing power for data acquisition, communication tasks.

- **Power Supply (5V, 2.5A)**

Provides stable and sufficient power to the Raspberry Pi. A regulated 5V supply with at least 2.5A output ensures reliable operation, especially when peripherals such as USB devices or sensors are connected.

- **Analog-to-Digital Converter (ADC) Module (ADS1115)**

Required for interfacing analog sensors (such as the pH sensor and turbidity sensor) with the Raspberry Pi, which supports only digital inputs. The ADS1115 is a high-resolution 16-bit ADC that communicates via $I^2C$, offering excellent precision for measuring low-voltage signals. It supports four single-ended or two differential input channels, making it suitable for connecting multiple analog sensors.

- **Raspberry Pi Camera Module V2 Original PI 3 Official camera V2 8MP 1080P30**

An official camera for Raspberry Pi with an 8-megapixel sensor, capable of capturing high-resolution still images and 1080p video at 30 frames per second. Used for fish activity tracking and visual monitoring.

- **DS18B20 Waterproof Digital Temperature Sensor**

A digital temperature sensor enclosed in a waterproof casing, ideal for submersion in aquarium water. Provides accurate temperature readings via a digital 1-Wire interface.

- **Gravity Analog pH Sensor**

Measures the pH level of water, providing an analog voltage signal proportional to acidity or alkalinity. Essential for monitoring water chemistry and fish health.

- **DFRobot Gravity: Analog Turbidity Sensor**

Detects the cloudiness or suspended particles in water, providing an analog voltage output that correlates to turbidity levels.

- **Peltier Module TEC1-12706**

A thermoelectric module capable of heating or cooling when voltage is applied, used for precise water temperature regulation in the aquarium.

- **Double BTS7960B DC 43A Motor Driver H-Bridge PWM (MD0012)**

A high-current motor driver module for controlling Peltier modules with PWM signals, enabling bidirectional control and high power handling.

- **Servo Motor Plastic Wheel SG90 Full Set Normal**

A small servo motor used to control the feeding mechanism in the aquarium system, providing precise movement to dispense food accurately.

- **Cooling Fan 4010 Axial 40x40x10mm 5V**

A compact cooling fan used specifically with the Peltier module to dissipate heat and ensure stable temperature regulation in the aquarium system.

- **Aluminum Heatsink**

A metal block used with the Peltier module to absorb and disperse heat, helping maintain efficient temperature control in the aquarium system.

- **DS3231 Precision RTC Real time Clock Memory Module AT24C32 IIC I2C ZS-042**

A highly accurate real-time clock module that maintains timekeeping even during power loss, used for scheduling feeding times

- **DC 12V 5A Power Supply Unit**

Provides stable 12V power for operating the Peltier modules and the Motor Driver H-Bridge in the aquarium system.

### 4.1.3 Firmware

The firmware for the Raspberry Pi 3 B is developed primarily in Python, leveraging various libraries to interface with sensors and actuators. The key functions of the firmware include:

**System Initialization**

- Loads configuration files (e.g., MQTT broker settings, sensor thresholds, scheduled feeding times).
- Initializes GPIO pins and I2C/SPI communication interfaces.
- Establishes communication with connected hardware devices (sensors, actuators, camera).
- Starts separate threads or processes for sensors, actuators, communication, and video streaming tasks.

**Sensor Data Acquisition**

- Reads values from:
    - pH sensor via ADS1115 (analog input).
    - Temperature sensor (DS18B20) via 1-Wire interface.

- Turbidity sensor via ADS1115.
- Real-time clock (DS3231) via I2C for precise timekeeping.

· Captures images or short videos from the Raspberry Pi Camera Module.

· Converts camera frames into an RTSP video stream using the mediaMTX server running locally on the Raspberry Pi.

**Data Processing**

· Converts raw sensor readings into real-world units (e.g., voltage to pH values).

· Applies calibration factors and noise filtering to improve measurement accuracy.

· Compares readings against predefined thresholds to detect abnormal conditions

(e.g., high turbidity, incorrect temperature).

**Actuator Control**

· Controls the servo motor to operate the feeding mechanism precisely.

· Activates or deactivates the Peltier module via the BTS7960B motor driver

· based on temperature readings.

**Communication (MQTT Protocol)**

· Publishes sensor data and system status to MQTT topics at defined intervals.

· Subscribes to MQTT topics to receive remote commands (e.g., trigger feeding, adjust temperature settings).

· Exchanges data in structured JSON format,

e.g.: { "temperature": 26.4, "pH": 7.2, "turbidity": 150 }

**Safety and Error Handling**

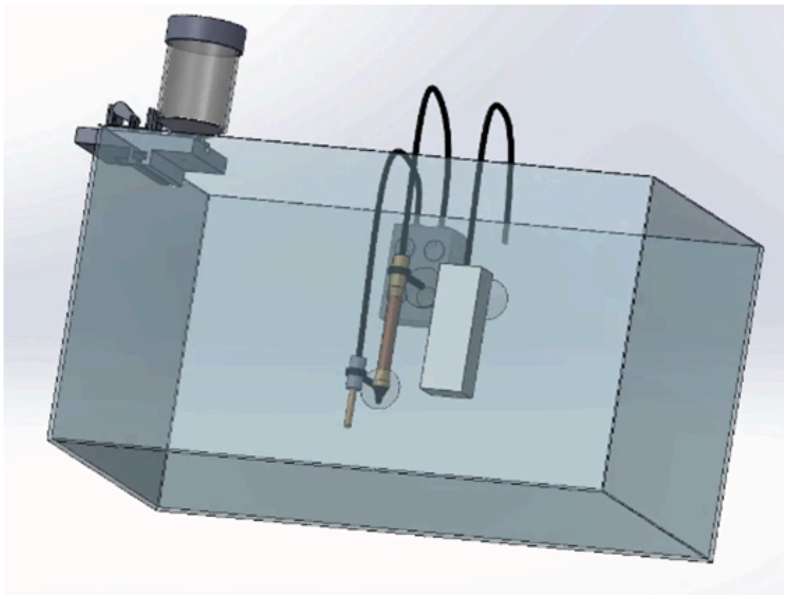
- Monitors for hardware faults, sensor disconnections, or abnormal sensor values.
- Disables relevant actuators automatically if critical faults occur (e.g., over-temperature conditions).
- Sends alerts or error notifications via MQTT to inform users of issues requiring attention.

**Scheduled Tasks**

- Automatically triggers the feeding mechanism at predefined times based on RTC scheduling.
- Initiates periodic sensor readings and data uploads (e.g., every 10 seconds).

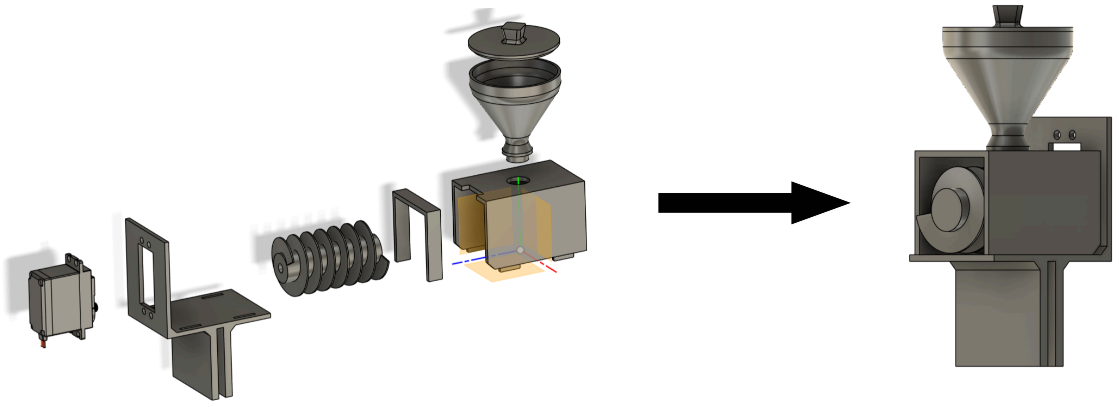
The Raspberry Pi runs a Linux-based OS which provides a robust and flexible environment for running the firmware. This allows easy updates and integration of new features, such as advanced analytics or machine learning algorithms for fish activity recognition.

## 4.2 3D-Designs



Covers the mechanical design and housing for the Smart Aquarium Unit. It includes CAD models, 3D-printed enclosures, and protective casings to safely mount sensors, electronics, and actuators while ensuring waterproofing and easy maintenance.

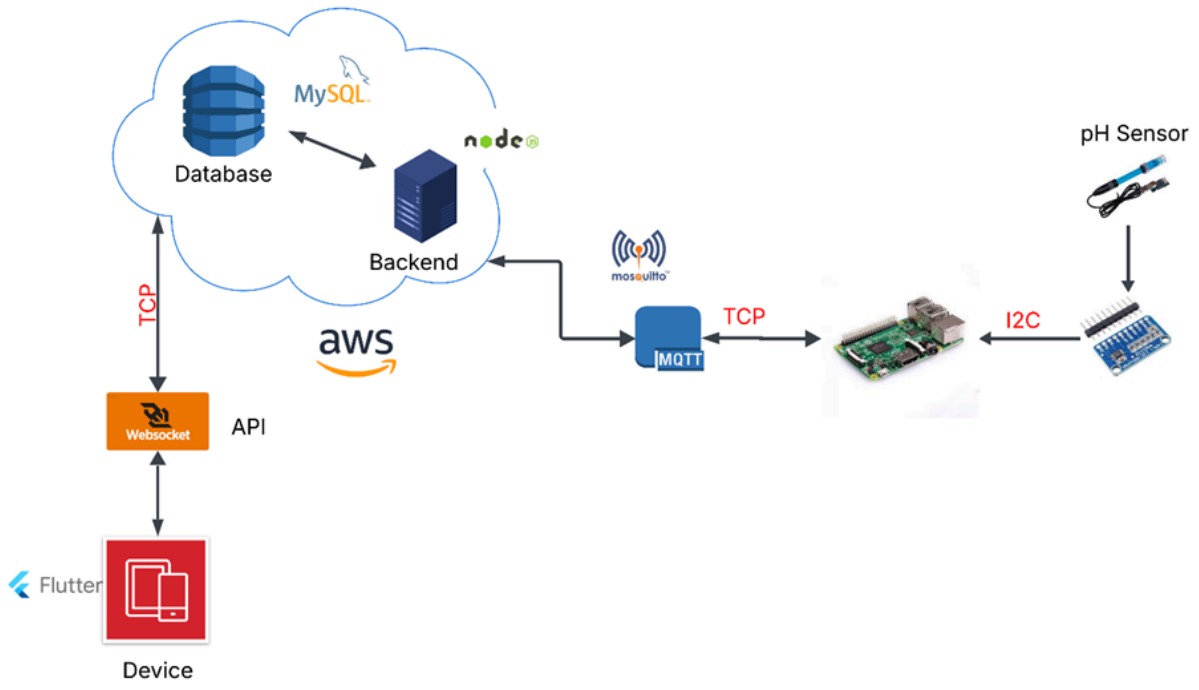
### 4.2.1 Fish feeding dispenser



This image displays a 3D model of the automatic Fish Feeding Dispenser, shown in both an exploded component view and a final assembled view.

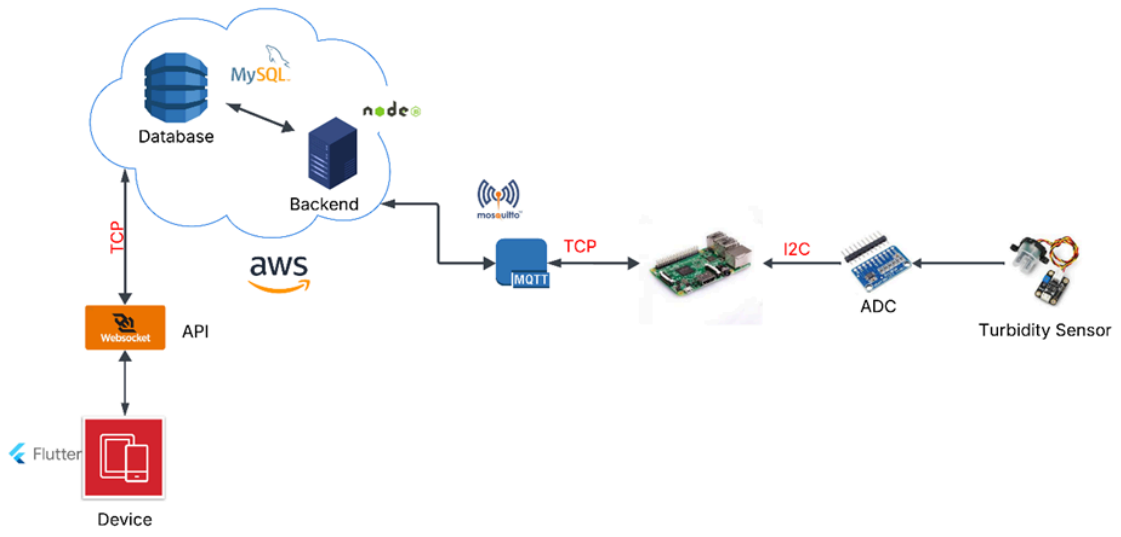
## 4.2.2 Sensors

### 4.2.2.1. pH Sensor



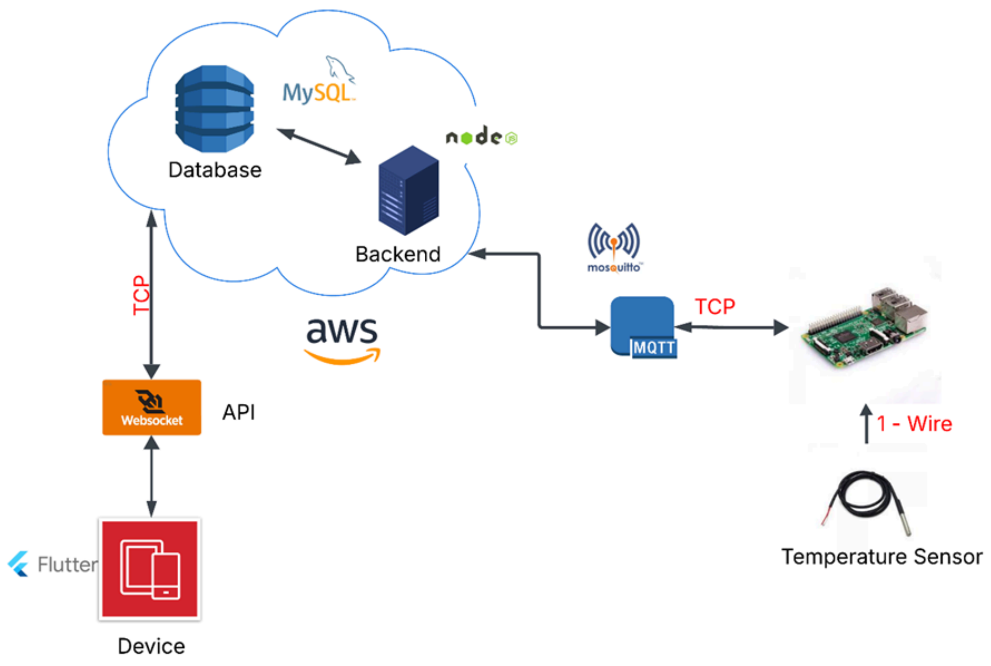
- The pH monitoring system begins with a pH sensor that measures the acidity or alkalinity of a solution and sends analog signals to an ADC (Analog-to-Digital Converter).
- The ADC converts these signals into digital data, which is transmitted to a Raspberry Pi via the I2C communication protocol.
- The Raspberry Pi processes the data and publishes it to an MQTT broker (Mosquitto) using TCP.
- A backend server, developed with Node.js and hosted on AWS, subscribes to the MQTT broker to receive the pH data and stores it in a MySQL database.
- A Flutter application on the user's device connects to the backend via WebSocket over TCP to display the real-time pH readings.

### 4.2.2.2 Turbidity Sensor



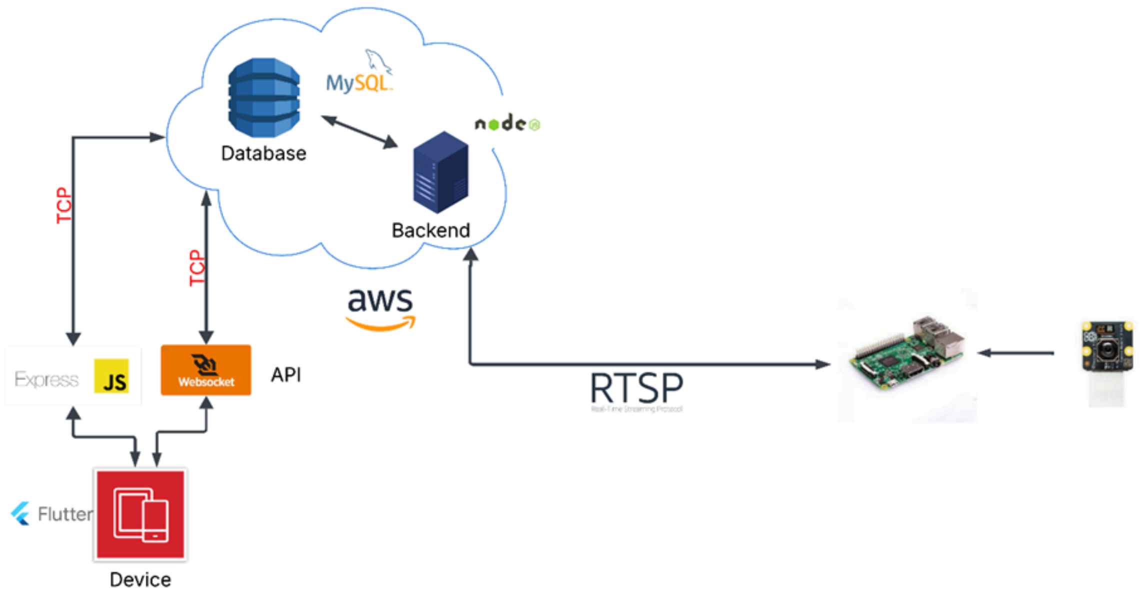
- The turbidity monitoring system begins with a turbidity sensor that measures the clarity of water and sends analog signals to an ADC (Analog-to-Digital Converter).
- The ADC converts these analog signals into digital data, which is transmitted to a Raspberry Pi via the I2C communication protocol.
- The Raspberry Pi processes the data and publishes it to an MQTT broker (Mosquitto) using TCP.
- A backend server, developed with Node.js and hosted on AWS, subscribes to the MQTT broker to receive the turbidity data and stores it in a MySQL database.
- A Flutter application on the user's device connects to the backend via WebSocket over TCP to display the real-time turbidity readings.

### 4.2.2.3 Temperature sensor



- The temperature monitoring system uses a temperature sensor that measures environmental temperature and sends data to a Raspberry Pi using the 1-Wire communication protocol.
- The Raspberry Pi processes the temperature data and publishes it to an MQTT broker (Mosquitto) via TCP.
- A backend server, developed with Node.js and hosted on AWS, subscribes to the MQTT broker to receive the temperature readings and stores the data in a MySQL database.
- A Flutter application running on the user's device connects to the backend through WebSocket over TCP to retrieve and display the real-time temperature readings.

### 4.2.3 Camera Module (for fish activity tracking)

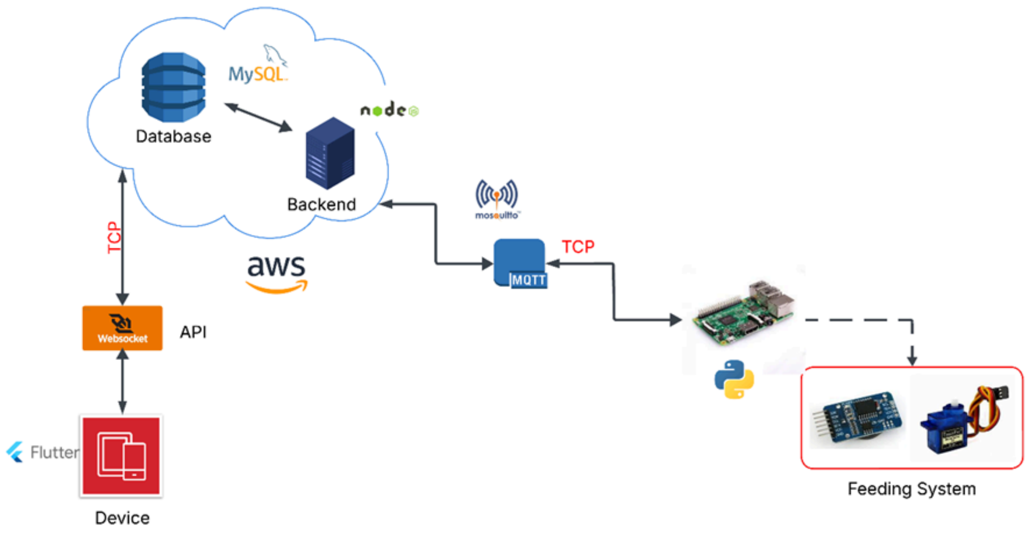


- The smart aquarium system uses a Raspberry Pi Camera Module V2 for real-time visual monitoring of fish activity and environmental conditions.
- The camera connects to the Raspberry Pi via the CSI (Camera Serial Interface) and captures high-resolution video frames at configurable settings (e.g., 30 FPS at 720p).
- These frames are encoded and streamed using the mediaMTX server (formerly rtsp-simple-server), which runs on the Raspberry Pi and generates an RTSP stream URL (e.g., `rtsp://<pi_ip>:8554/stream`).
- This live video stream is accessed by the backend or user applications such as mobile apps or dashboards.
- The backend can optionally record footage or analyze the stream using computer vision techniques for motion tracking or fish health monitoring.

The camera system is initialized by the firmware at startup and runs alongside other processes like sensor data collection and actuator control. The setup allows users to remotely observe the aquarium, verify fish behavior and feeding, and detect mechanical or biological issues, with potential future support for AI-based alerts and behavior analysis.

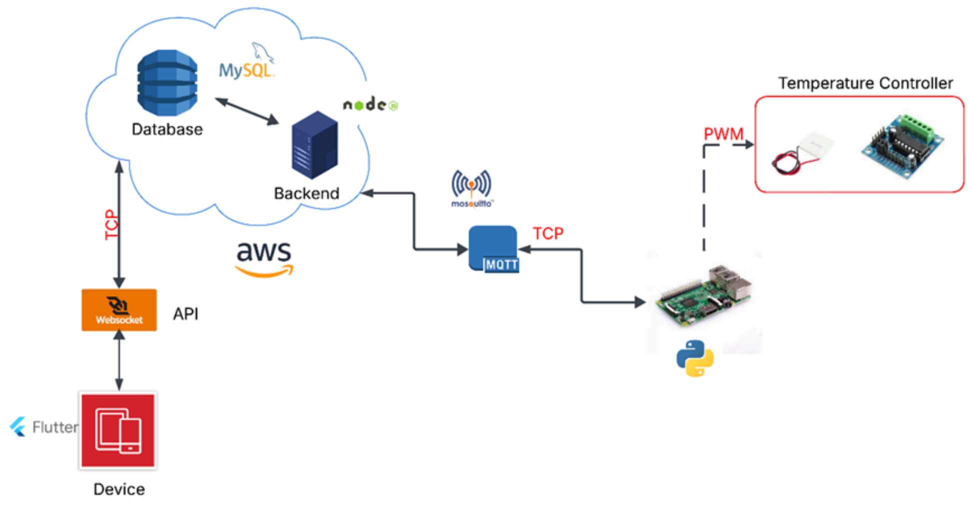
## 4.3 Actuators

### 4.3.1 Servo Motor



- The feeding system is controlled by a Raspberry Pi, which receives feeding commands through the MQTT protocol over TCP. These commands are published by the backend server via a Mosquitto MQTT broker.
- On the Raspberry Pi, a Python script subscribes to the MQTT topic and listens for feed instructions. When a feed command is received, the script activates the servo motor, which is part of the feeding mechanism.
- The servo motor is connected to the Pi via GPIO pins and is responsible for mechanically dispensing fish food.
- This system allows automated or remote feeding of the fish based on user input from the mobile app or a present schedule defined in the backend.

### 4.3.2 Peltier Cooling/Heating Unit

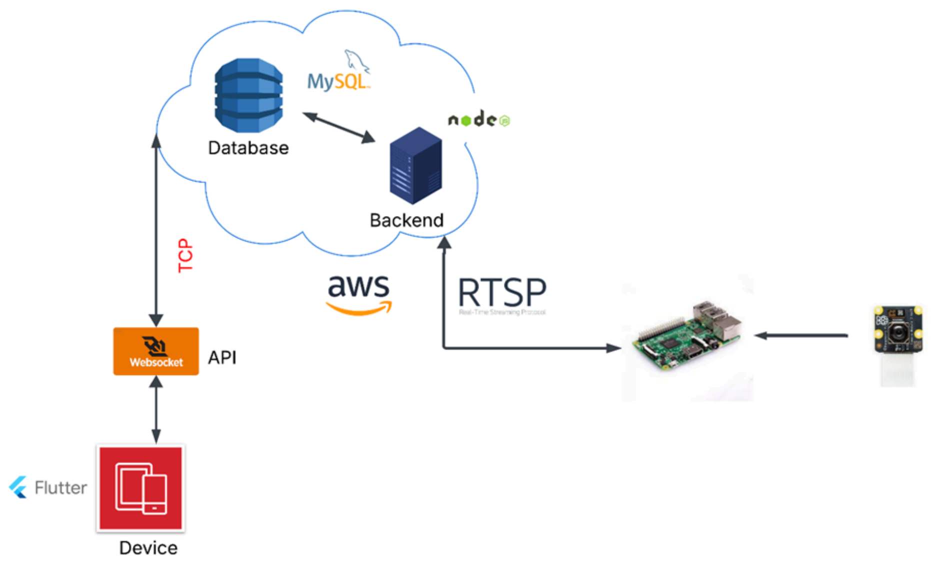


- The temperature control system automatically maintains the water temperature using a Peltier module, controlled by a motor driver and a Raspberry Pi.
- A temperature sensor continuously monitors the water temperature. This sensor is connected to the Raspberry Pi, where a Python script runs locally to process the temperature data.
- Based on the sensor readings, the Python script decides whether to heat or cool the aquarium. It generates appropriate PWM (Pulse Width Modulation) signals to control the motor driver, which adjusts the power sent to the Peltier module.
- The Peltier module then either heats or cools the water depending on the requirement. This control loop runs entirely on the Raspberry Pi, without needing instructions from the backend or user app. This ensures real-time, autonomous temperature regulation for optimal fish health.

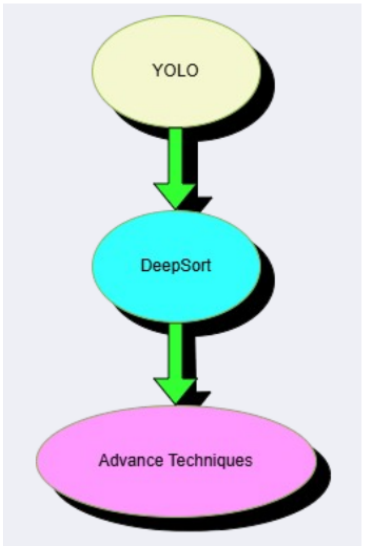
# 5. Low Oxygen Level Detection & Behavior Monitoring System Using Fish Tracking

## 5.1 Introduction

A trained Fish Tracking Model is integrated into the Smart Aquarium platform to enable real-time behavior analysis of fish. This system leverages advanced machine learning and computer vision techniques to detect, identify, and track individual fish.



## 5.2 Core Technologies



· **YOLO (You Only Look Once)**
Used for real-time object detection to identify fish in each video frame captured from the aquarium.

· **DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric)**
Assigns unique IDs to each fish and maintains these IDs across frames, enabling continuous tracking of individual fish.

· **Kalman Filters & Optical Flow**
Used to smooth tracking trajectories, reduce noise, and handle complex scenarios such as fish overlapping, partial occlusion, and variable lighting.

Fish tracking model code repository : -
https://github.com/cepdnaclk/e20-3yp-Smart-Aquarium/tree/main/code/object-tracking-yolov8-deep-sort-master

## 5.3 Behavior Monitoring and Abnormality Detection

The system detects and flags abnormal behaviours based on real-time movement patterns. In this product, there are two main patterns that can be identified from the system.

- **Extended Stationary Behaviour:**
  May indicate illness, stress, or reflex impairment.
- **Frequent Surfacing:**
  A potential sign of low dissolved oxygen in water (hypoxia).

These behavioural flags are supported by marine biology studies and research on fish health indicators.

Reference:-

1. Reebs, S. G. (2009). Oxygen and fish behaviour. Journal of Experimental Biology, 218(11), 1777–1786

2. Davis (2007) demonstrated that reflex impairment, such as a lack of escape response or immobility, predicts stress and mortality in fish.

"Reflex impairment is predictive of stress and mortality in fish."— Davis, M. W. (2007). Fisheries Research, 86(1), 253–264.

## 5.4 Data Flow and Integration

**Video Input:**
Real-time video is captured via the Raspberry Pi camera and streamed as an RTSP feed using a local mediaMTX server.

**Backend Processing:**
The RTSP stream is received by a Node.js backend, which sends video frames to a machine learning model deployed on an AWS server. This model performs fish detection and tracking (using YOLO and DeepSORT) on each frame.

**Data Storage:**
Detected fish activity, alert logs, and metadata are stored in a MySQL database for historical analysis and system diagnostics.

**Live Monitoring:**
Processed insights, behavioral alerts, and real-time tracked positions are pushed to a Flutter-based mobile app using WebSockets over TCP, enabling low-latency updates and continuous health monitoring.

# 6. Testing

## 6.1 Software testing

### Tools & Tests

Postman

- Use to test backend
- Check if the URLs and endpoints are correct
- Inspect request and response data

wscat

- Use to test WebSocket api
- Send and receive messages in real-time and it Debug WebSocket endpoints

MQTT Explorer

- Use to test connection between raspberry pi and backend
- Subscribe to topics and view real-time messages
- Publish messages to test how your system reacts

### Functionality Testing

- Ensure all sensors correctly transmit data to the backend.
- Test actuator control (feeding, cooling, fan) via software commands.
- Validate MQTT topic publishing/subscription flows.
- Check mobile app receive real-time updates and control responses.

## Integration Testing

- Sensor-to-MQTT-to-Node.js backend.
- Node.js backend to MySQL database logging.
- YOLO + DeepSORT integration with RTSP video streams.
- WebSocket connection to the Flutter app.

## Model Accuracy Testing

- Evaluate fish detection accuracy using ground-truth annotated videos.
- Test DeepSORT tracking consistency during fish occlusion.
- Trigger and validate alerts for abnormal behavior (e.g., surfacing too long).

## User Interface Testing (Flutter App)

- Test real-time updates on the dashboard.
- Validate command buttons (manual feed, toggle cooling).
- Test alert notifications and data visualization.

## Communication Protocol Testing

- Verify reliability of MQTT message delivery.
- Check JSON payload format and error handling.
- Simulate network interruptions and test reconnections.

## Data Logging Testing

- Ensure correct timestamping from RTC (DS3231).
- Check data persistence in local storage and database.
- Validate CSV export if implemented.

## 6.2 Hardware testing

### Sensor Calibration Testing

- pH & Turbidity: Compare sensor outputs with standard solutions.
- Temperature: Compare DS18B20 output with a calibrated thermometer.
- Real-Time Clock (DS3231): Test accuracy over time.

### Actuator Testing

- Servo Motor: Check smooth and precise operation for feeding.
- Peltier Module: Test cooling/heating functionality with BTS7960B.
- Fan: Ensure proper response when Peltier is active.

### Power Supply Testing

- Verify stable voltage to Raspberry Pi and all modules under full load.
- Test for overheating or voltage drop in long-duration operations.

### Camera Module Testing

- Test resolution, frame rate, and night/day visibility.
- Ensure continuous video stream without lag or frame drops.
- Verify correct capture and storage of frames for analysis.

## System Stability & Uptime

- Run the system for long hours to test memory usage, crashes, or sensor disconnects.
- Monitor heat buildup and test effectiveness of heat dissipation setup (heatsink + fan).

## Environmental Testing

- Place the entire system in a simulated aquarium environment and test response to real water conditions.

## Source Code Repository

GitHub- https://github.com/cepdnaclk/e20-3yp-Smart-Aquarium