MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

B.Sc. Computer Science Major Project Report

AQUA CARE: SMART FISH FARMING MONITORING SYSTEM USING IOT

A report submitted in partial fulfilment of the requirements for the Sixth Semester B.Sc. Computer Science Degree

SUBMITTED BY

LIYA THOMAS	**(2220807)**
KARTHIKEYAN K.B	**(2220821)**
PRANAV P	**(2220823)**

Under the guidance of

Dr. RESMI V

DEPARTMENT OF COMPUTER SCIENCE

B.Sc. Computer Science

2025

MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled **"AQUACARE: SMART FISH FARMING MONITORING SYSTEM USING IOT"** is a bonafide record of the work done by **LIYA THOMAS** (2220807), **KARTHIKEYAN K.B** (2220821), **PRANAV P** (2220823), in partial fulfilment of the requirements for the award of the Degree of Bachelor of Science in Computer Science by the University of Kerala.

INTERNAL GUIDE **HEAD OF THE DEPARTMENT**

EXTERNAL EXAMINERS

1.

2.

ACKNOWLEDGEMENT

We begin by expressing our deepest gratitude to the **ALMIGHTY** for guiding us throughout this project. We would also like to extend our heartfelt thanks to those individuals who have contributed significantly to our success.

We thank our **PARENTS**, whose unwavering support and encouragement have been a constant source of motivation for us. Their faith in us has been a driving force, and we are forever grateful.

We are thankful to our Principal, **Dr. MEERA GEORGE,** who provided us with a platform to grow and excel. We appreciate the opportunities we've had to develop our skills and knowledge.

We thank our Director, **Dr. K. OOMMACHAN**, whose visionary leadership and encouragement have inspired us to strive for excellence. We thank him for providing us with the necessary resources and facilities.

We express our sincere gratitude to **Ms. TINU C. PHILIP**, Head of the Department of Computer Science, who offered valuable guidance and support throughout our learning journey. Her expertise and feedback have been invaluable.

We are grateful to our project guide, **Dr. RESMI V**, Assistant Professor, Department of Computer Science, who provided us with exceptional guidance, support, and encouragement throughout the project. Her suggestions and feedback have helped us refine our work.

We also appreciate the support and feedback from all the faculty members of the Department of Computer Science, who took the time to review our work and offer valuable comments.

Last but not least, we would like to thank our friends and everyone who has contributed to the success of this project, whether directly or indirectly. Your support and encouragement have meant the world to us.

ABSTRACT

Aquaculture, or fish farming, is a rapidly growing sector that plays a crucial role in meeting the global demand for seafood. Effective management of water quality and environmental conditions is essential for the health and productivity of aquatic life. Traditional methods of monitoring and maintaining these conditions are labour-intensive and prone to human error, which can lead to suboptimal conditions for fish growth and increased mortality rates. To address these challenges, automation and remote monitoring have become increasingly important in modern aquaculture practices. This project describes the creation of an automated fish farming system that monitors and regulates the aquatic environment by integrating an ESP32 microcontroller with a number of sensors and actuators. The system continuously measures the aquarium's pH, dissolved oxygen concentration, water temperature, and water level using a pH sensor, Dissolved Oxygen (DO) sensor, DHT11 temperature sensor, and an ultrasonic sensor. Through the ESP32's Wi-Fi capabilities, real-time data from these sensors is sent to a Firebase database. A mobile app is designed to display this data, allowing users to monitor the conditions of the fish farm remotely.

The system also has control capabilities for a water pump and an oxygen pump, which can be operated automatically based on sensor readings or manually via the mobile app. For example, the water pump can be adjusted to maintain ideal water levels. The mobile app facilitates this by sending control commands to the Firebase database, which the ESP32 retrieves and executes.

This automated technology promotes sustainable aquaculture practices by lowering human labour, enabling remote management, and maintaining ideal water conditions, all of which increase the efficiency of fish farming operations.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
MySQL	My Structured Query Language
SMS	Short Message Service
LED	Light Emitting Diode
Wi-Fi	Wireless Fidelity
PHP	Hypertext Preprocessor
LDR	Light Dependent Resistor

1 INTRODUCTION

Fish farming, often known as aquaculture, is a quickly expanding industry that is essential to supplying the world's seafood needs. The production and well-being of aquatic life depend on the efficient control of environmental factors and water quality. The labour-intensive and human error-prone nature of traditional monitoring and maintenance techniques can result in less-than-ideal circumstances for fish growth and higher mortality rates. Automation and remote monitoring have grown in significance in contemporary aquaculture methods to meet these issues.

Using an ESP32 microcontroller, this project seeks to create an automated fish farming system that incorporates multiple sensors to continuously monitor important water conditions. The system uses an ultrasonic sensor to determine the aquarium's water level, a DHT11 sensor to track water temperature, a Dissolved Oxygen (DO) sensor to keep an eye on the oxygen levels required for fish respiration, and a pH sensor to determine how acidic or alkaline the water is.

By continuously gathering data from various sensors, the ESP32 may use its integrated Wi-Fi capabilities to send the data to a Firebase database. Users can then access this data via a smartphone application, which gives them a real-time picture of the water conditions in their fish farm. A water pump and an oxygen pump can also be controlled via automation elements in the system. With the help of the app, these pumps can be controlled manually or automatically in response to preset criteria for sensor readings, guaranteeing that the aquatic environment is always maintained in ideal conditions.

By combining these technologies into a unified system, fish farming operations become more accurate and efficient while also enabling remote management and monitoring. By eliminating the need for continuous manual intervention, this automated approach lowers labor costs and the possibility of human error while fostering scalable and sustainable aquaculture methods. This research represents a major breakthrough in automated fish farming systems by utilizing contemporary IoT technologies.

1.1 PURPOSE OF THE PROJECT

- **Providing Fish with the Best Living Conditions:** The system makes sure the aquarium environment is safe and healthy for the fish by keeping an eye on important variables including pH, temperature, light, and water level.
- **Automation for Ease of Maintenance:** This makes aquarium care more effective and time-efficient by automating the monitoring of important elements, reducing the need for manual checks.
- **Real-time Monitoring and Alerts:** If any parameter deviates from the optimal range, users can act right away thanks to the system's real-time data and notifications via an Android app.
- **User-friendly Application:** The Kotlin-developed Android application provides a practical and easy-to-use interface for users to obtain data and manage the system.
- **Encouraging Fish Health and Longevity**: Effective water quality control reduces stress and wards off illnesses, improving the general
- **Water Level Management:** Ensures adequate water levels in the aquarium to prevent issues like pump damage or fish stress due to low water levels.

1.2 OBJECTIVE

By putting in place an automated system that makes use of Internet of Things technology to maximize water quality and aquatic life's environmental conditions, this initiative aims to completely transform conventional fish farming methods. By regularly checking important water parameters including pH, dissolved oxygen, temperature, and water level—all of which are essential for preserving a proper aquatic environment—it aims to protect fish health and production. The system can gather data in real time thanks to the employment of sophisticated sensors, guaranteeing accurate and trustworthy monitoring. Through the ESP32 microcontroller's integrated Wi-Fi, this data is sent to a Firebase database, allowing for smooth and immediate access to the information.

The project includes a specific Android application created in Kotlin to improve user ease. Fish farmers can use this software as a central interface to manually operate vital equipment like water pumps, check water conditions in real time, and get warnings when parameters depart from permitted ranges. The system's automatic control features, which activate these devices based on preset thresholds for improved efficiency, guarantee ideal water quality without the need for continual human supervision.

In addition to increasing convenience, the technology tackles important aquaculture issues such the inaccuracy and inefficiencies of conventional manual monitoring techniques. The project lowers labour needs, decreases human error, and guarantees consistent maintenance of water conditions by automating regular chores. Consequently, this lowers operating expenses It increases the accessibility of fish farming, particularly for farmers with limited resources or those operating on a small scale. The system's dependability and usefulness are further increased by its capacity to maintain proper water levels using ultrasonic sensors to avoid damaging equipment and stressing out the fish.

The project's main goal is to advance scalable and sustainable aquaculture methods. In addition to improving fish health and lowering mortality rates, the system promotes ecologically friendly agricultural practices by providing optimal living circumstances for fish. Its automation and remote monitoring features enable effective resource use, reducing energy and water waste and increasing output. Additionally, because of its scalability, the system can be used in a variety of agricultural environments, ranging from little aquariums to expansive fish farms.

In summary, this project represents a significant advancement in modern fish farming technologies. By integrating IoT, automation, and user-centric design, it offers a comprehensive solution to the challenges of traditional aquaculture. The result is a more sustainable, cost-effective, and efficient approach to fish farming that benefits both farmers and the environment while setting a new standard for the industry.

2 LITERATURE SURVEY

The Smart Aquarium Management System eliminates manual aquarium care and integrates an efficient, cost-effective solution for aquarium maintenance. It feeds, lights, and controls the oxygen levels as well which ensures an effective and reliable aquarium care. A system with the ability to operate in real time and integrate real time data is proposed. It conducts an aeration system, temperature monitoring, and water pH level detection. The implemented IoT based system allows users to track the status of their aquarium. This way the system prevents over or underfeeding of the fish and makes the aquarium easier to maintain. [1]

[2] shows the development, construction and operation of an automatic fish feeder. It is a microcontroller-based system. It is intended to deliver a specific quantity of fish luncheons at a predetermined time every day into an aquarium. The system includes a cylindrical can, distributing tubes and a stand. A stepper motor that is attached to the canister controls the dispensing of food. A timer-controlled switch permits rotation of the motor at the time set up by a program inside the microcontroller. The device automatically dispenses fish feed at preset times.

Out of all the pets, keeping a fish as a pet is proven to be the least effort consuming. With that being said, one does have to keep in mind that the maintenance of a fish aquarium can prove to be challenging. These include changing the water of the aquarium, feeding the fish, maintaining the temperature, and controlling the brightness of the lights. However, the plan is to move most of these problems from the manual control to an automatic one in order to facilitate the fish keepers. Because of this, aquarists will no longer have to be concerned with constantly checking their aquariums or their fish. This software is designed to automatically control a number of aquarium environmental aspects, including water temperature, aquarium lighting and fish feeding, as well as water draining and restoring. The end user does not require skills. We expect that the efficiency of the system provided by this project will be much higher than that of the currently available solutions.[3]

The product developers from [4] emphasize the implementation of IoT technology to replace the traditional manual approach in maintaining aquariums where the IoT gadget will be configured to control, check, and communicate about the temperature, water pressure, light intensity, and other vital aquarium parameters to mobile phone applications while allowing for water management regarding overflow and underflow. It will also allow for automatic control of the aquarium’s lighting.

Proper aquaculture systems monitoring and management practices are crucial for proper functioning and increased output. This work describes an aquaculture system with an efficient fish tank monitoring solution to help with the efficiency of operations and the welfare of the fish. At the centre of this system is the Arduino Uno microcontroller which acted as the central processing unit of the unit, controlling all functions. Among its key components are pH sensor, turbidity sensor and water level sensor. In cases where the pH level is l outside preset bounds or water is too muddy, the system sends out automated warnings via SMS to a specified owner. Also, data is sent to an IoT website that can be accessed remotely for monitoring purposes. In order to maintain steady environmental conditions two pump motors are used: an automatic one activates in the event of a drop in water level while the other hose removes water in the event of particles getting deposited. In addition, the system is supplemented by a servo motor mechanism that allows the fish feed to be dispensed at set times so as not to overfeed the fish. [5]

This study utilized the LinkIt 7697 module along with the BlocklyDuino editor to create a control system for a smart aquarium. The system is designed to monitor temperature, light intensity, and water level in the aquarium, while also providing alerts for any intruders detected by various sensors. Additionally, it includes an automatic feeder operated by a servo motor. Data is transmitted to a back-end computer through Wi-Fi and is displayed in real-time on the Cloud Sandbox platform. [6].

The authors of [7] aim to create an IoT system designed for freshwater fish aquarium owners. This system will monitor the pH and temperature of the aquarium water and automatically feed the fish twice daily according to the owner's schedule via a smartphone app. It employs an ESP32 microcontroller, a motor for automatic feeding, a waterproof temperature sensor to measure the water temperature, and a colour sensor paired with a universal pH strip to assess the water's pH level. The colour sensor detects the strip's colour, which changes when placed in the aquarium water, and the program then categorizes it into the corresponding pH value.

Aquaculture is an expanding industry that plays a vital role in sustainable food production and the economic development of countries. Technology-driven aquafarming offers effective solutions for monitoring water quality and achieving high yields. Integrating the Internet of Things (IoT) into aquaculture can meet these needs. This research article presents a detailed approach to seamlessly incorporate IoT sensors into aquafarming settings, using Arduino boards and communication modules. The proposed method accurately measures key water quality parameters, including temperature, pH levels, and Dissolved Oxygen (DO), which are crucial for creating optimal conditions in aquaculture. This approach allows for the real-time collection of important data points, helping to prevent fish diseases and reduce mortality rates with minimal human intervention and lower maintenance costs.[8]

3 SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Existing System: Challenges and Limitations in Traditional Fish Farming

Monitoring and managing the aquatic environment in conventional fish farming systems primarily depends on manual procedures and simple instruments, which presents a number of accuracy and efficiency issues. These systems frequently lack the sophisticated features required for reliable and efficient management of water quality. A thorough explanation of the typical characteristics and constraints of current systems may be found below:

1. **Manual Monitoring**
    - Critical water parameters such as pH, temperature, dissolved oxygen, and water level must be physically checked by farmers using physical test kits or other tools.
    - This procedure is labour-intensive, time-consuming, and prone to human error, which increases the possibility of suboptimal circumstances and produces contradictory data.
2. **Lack of Automation**
    - The majority of conventional systems are not automated, necessitating human involvement to handle parameter deviations or changes in water quality.
    - The farmer must carry out corrective steps, including turning on pumps or correcting pH levels, which increases workload and the likelihood of delays.
3. **No Remote Monitoring**
    - Farmers must be physically there to verify the tank's condition because these systems usually do not offer remote monitoring.
    - This reduces adaptability and raises the possibility that problems may be overlooked while absent.
4. **No Alerts or Notifications**
    - Until they physically examine the tank or notice obvious symptoms of illness in the fish, users are typically oblivious of issues.
    - Lack of real-time alerts makes it more difficult to take prompt action to stop possible harm.

5. **Standalone Devices**
    - Measurement devices for things like pH, temperature, water level, and dissolved oxygen don't work together; they work separately.
    - It can be inefficient for users to examine each metric independently, which makes it challenging to evaluate tank conditions holistically.

3.2 PROPOSED SYSTEM

Proposed System: Advancing Fish Farming with Automation and IoT Integration

The proposed system offers a clever, integrated, and automated solution for contemporary fish farming by utilizing Internet of Things technologies, an embedded system (ESP32), and an intuitive Android mobile application. The system, which minimizes manual involvement and allows for continuous, real-time monitoring and management of critical water parameters, was created to improve efficiency, accuracy, and convenience.

Key Features of the Proposed System

1. **Real-Time Monitoring**
    - IoT sensors are used to monitor critical parameters including pH, dissolved oxygen, temperature, and water level in continuous mode.
    - An ESP32 microcontroller is used to send the data wirelessly to the Firebase database, enabling users to obtain live updates wherever and whenever they please through the Android app.
2. **Automation and Alerts**
    - It automatically monitors the environment and sends alerts when any parameter exceeds or falls below the defined parameters.
    - Mobile app notifications also allow users to take action quickly to avoid harming aquatic life.
3. **Remote Monitoring**
    - The combined Android app allows users to remotely check their fish farming environment, providing flexibility and enabling them to maintain water conditions even in their absence.

4. **Integrated System**
    - In contrast with the standalone tool, the suggested system unifies multiple sensors (i.e., pH, dissolved oxygen, temperature, water level) into one platform.
    - Using the app, we can have a complete view of the aquatic environment, simplifying management processes.
5. **Customizable Thresholds**
    - Users are granted the flexibility to establish custom thresholds for every parameter that is being monitored, tailoring the ranges to the physiological needs of the from which aquatic species they're working.
    - Automated control systems vary the oxygen pump and water pump as necessary to ensure perfect conditions in the water.
6. **User-Friendly Mobile Application**
    - Kotlin The app is based on Kotlin, providing a user-friendly interface for real-time data viewing, alerts management, historical trends analysis, and settings customization.
7. **Energy Efficiency**
    - The tank network also promotes energy-efficient operation and prevents resource wastage by using real data to optimize resource usage (such as water pumps and oxygen systems) according to needs.
8. **Improved Fish Health and Sustainability**
    - Regular monitoring and immediate adjustments help maintain a healthy aquatic ecosystem, minimising stress to the fish and promoting their health.
    - This system reduces potential risks like TEMP SHOCKS, or lack of oxygen, allowing for sustainable species for aquaculture development.

3.3 FEASIBILITY STUDY

A feasibility study is the evaluation of a project to determine its viability from technical, economic, operational, and schedule perspective. A comprehensive feasibility study for Aqua Care project, which is to track and manage farming parameters (pH, water temperature, ambient light and water level) with IoT and mobile application include the following key areas

3.3.1 Technical Feasibility

Hardware Components:

ESP32 Microcontroller: The ESP32 is a low-cost, low-power system on a chip with a microcontroller and embedded Wi-Fi for IoT applications. It can be used with numerous sensors and is suitable for continuous data monitoring.

Sensors:

pH Sensor: Available and widely used for water quality monitoring.

Temperature Sensor: Used to monitor water temperature.

Ambient Light Sensor: Used to measure light intensity.

Water Level Sensor: Used to monitor the water level in the aquarium.

Software:

Mobile Application: We have developed an android based mobile application using Kotlin. Kotlin is the document-supported language and community-supported language for Android development.

Wi-Fi Communication: The ESP32 communicates with the mobile app for transmitting data remotely over Wi-Fi.

Software Development:

Requires Embedded C for the ESP32 to interact with the sensors and send data to the mobile app over Wi-Fi.

Android App Development: An app will be written in Kotlin, giving us a more contemporary and efficient development environment with tools such as Android Studio for building the mobile front-end and linking the backend through wi-fi. Telemetry Integration via ESP32 and Android wireless– The ESP32 and Android application work together to allow hardware sensors to communicate with the mobile interface in a way that is technically feasible.

Data Management and Cloud Storage: The mobile app can either save data locally on the mobile device, or in the cloud (for historical data, trends, and user preferences e.g. Firebase).

3.3.2 Social Feasibility

By examining the proposed automated fish farming system's effects on society, including its acceptability, advantages, and conformity to societal norms and values, its social viability can be assessed. The main ideas that demonstrate its social viability are as follows:

1. Improved Accessibility for Small-Scale Farmers

   - By lowering the need for physical labour, the system enables small-scale farmers to embrace contemporary aquaculture techniques, potentially increasing their standard of living.

   - It streamlines fish farming procedures, making it accessible even for individuals with little technical know-how or resources.

2. Promotion of Sustainable Practices

   - By optimizing resource utilization and minimizing energy and water waste, the system promotes acceptability among environmentally concerned people and communities and is in line with the growing emphasis on environmental conservation and sustainable farming in society.

3. Enhancement of Food Security

   - The technology increases aquaculture yields by promoting better fish health and lower mortality rates, which helps meet the growing demand for fish as a source of protein and supports global food security initiatives.

4. Reduction of Labor-Intensive Processes

   - The technology lessens the mental and physical burden on fish farmers by automating monitoring and control, which increases aquaculture's attractiveness to a wider range of people, including the elderly and those with impaired physical capabilities.

5**.** Support for Education and Skill Development

- The use of IoT and smart technology in aquaculture promotes skill development and knowledge exchange among farming communities.
- By raising awareness of the advantages of technology in agriculture, training programs pertaining to its application could close the gap between conventional and contemporary methods.

6. Alignment with Digital Transformation Trends

- The system's utilization of IoT and mobile applications is in line with the public's increasing comfort level with smart technology as society becomes more digitally connected.
- As a result, the solution is more palatable and simpler to incorporate into current process.

7. Contribution to Public Health and Nutrition

- The technology indirectly promotes improved consumer nutrition by guaranteeing healthier fish production.
- This supports public health campaigns, particularly in areas where fish is a staple food.

8. Encouragement of Innovation in Agriculture

- The approach encourages more innovation in the fields of agriculture and aquaculture by acting as a model for incorporating technology into conventional farming methods.

9. Reduction of Inequality in Aquaculture

- The system provides equal chances for small-scale and large-scale fish farmers to benefit from contemporary technologies by bridging the gap with reasonably priced and scalable solutions.

10. Social Acceptance and Scalability

- The simplicity and tangible benefits of the system make it socially acceptable to a wide audience, from hobbyist aquarium owners to professional fish farmers.
- Its scalability ensures that it can adapt to diverse cultural and societal contexts, further enhancing its social feasibility.

3.3.3 Operational Feasibility

1. Ease of Implementation

- Simple Setup Process:
  System is designed for easy installation. Sensors, ESP32 microcontroller, and Android mobile application are easy to integrate and require less technical expertise. The system can be easily deployed on a farm or aquarium based on straightforward instructions.

- Plug-and-Play Design:
  Sensors (pH, Temperature, water level, and dissolved oxygen) are pre-calibrated and require basic setup to place in aquariums or fish farming systems, which reduces time and complications.

2. Reliability and Accuracy of Sensors

- Consistent Performance:
  The system uses IoT sensors to give real-time data of important water parameters. These sensors are robust and capable of consistent operation in a water environment.
- Data Transmission and Storage:
  Data is transmitted wirelessly to the Firebase database via an ESP32 microcontroller, allowing for real-time monitoring and access to the data. The information is then stored in a secure fashion, in addition you will be accessible from anywhere through the mobile app.

3. Automation and Control

- Automation of Critical Processes:
  It automates core tasks like monitoring the water parameters and activating actions (like turning on the pumps when oxygen remains low or adjusting the water level values). This minimizes the need for continuous manual monitoring, allowing fish farming operations to be more efficient.
- Customizable Thresholds:
  The system is highly adaptable to individual operational needs with adjustable thresholds for every parameter to meet the different demands of different fish species.

4. Mobile Application Usability

- User-Friendly Interface:
  For the mobile application in Android, ease of use and simplicity were top of mind during the development process. Users can easily set thresholds, search against historical trends, visualize real-time data, and receive alerts. Due to the simplicity of the app, users with very little technical literacy can use the system without any hassle.
- Remote Access:
  With the mobile app, users can manage and monitor their own fish farm or aquarium remotely, providing them with flexibility and control over it even though they are not on site. This function provides for constant monitoring of the aquatic environment.

5. Maintenance and Support

- Low Maintenance:
  The system, with limited maintenance, is designed for long durability. For example, IoT sensors are far more rugged, and therefore do not need to be recalibrated so frequently. ESP32 microcontroller and mobile app on both platforms are frequently updated for performance improvements and bug fixes.
- Troubleshooting:
  In case of problems, the system uses the app to send out alerts and messages. This allows users to detect and resolve issues easily and quickly. In addition, online documentation and support are available to assist with troubleshooting.

6. Scalability and Flexibility

- Adaptability to Different Scale Operations:
  The approach is applicable to different tank proportions and fish farming businesses, from large commercial farms to small hobby aquariums. Thanks to its versatility, the system can meet various user needs
- Integration with Existing Systems:
  The proposed approach can therefore readily fit into existing fish farming systems without necessitating any modification, thus assuring seamless adoption without interrupting normal business as usual.

7. Energy Efficiency

- Optimized Resource Use:
  By automating actions like controlling water pumps and changing lighting levels based on about what's happening in the building at a given moment, the technology aims to lower power consumption. Aquarium and fish farm owners benefit from reduced operational costs and prevent energy wastage.

8. Continuous Monitoring

- Real-Time Monitoring:
  It keeps track of the water quality parameters in real time, so any change, deviance, etc can be detected instantly. Real-time alerts that are sent to the mobile app make it easy for users to take corrective action as and when required.

9. Integration of Automation and Control Features

- Automated Water Pumps and Oxygen Pumps:
  This means that depending on the real-time data, the system automatically controls the water pumps so that the conditions of the water are optimal with minimal manual intervention. It decreases the chances of human error and allows for increased operational efficiency.
- Threshold-Based Actions:
  Each parameter has its own threshold, and based on that, the system sends automated alerts or actions. So that only such treatment is performed as is necessary; and in that which is superfluous, let it rest.

10. Cost-Effectiveness

- Reduction of Operational Costs:
  This helps the system reduce operational costs in fish farming by minimizing human error, reducing manual labour and optimizing resource utilization. In the long-term, this increases profitability for fish farmers, and targets small scale operations who may be more resource-poor.

3.3.4 Economic Feasibility

1. Initial Investment and Setup Costs

- Cost of IoT Sensors and Equipment:
  The main parts of the system are IoT sensors (pH, dissolved oxygen, temperature, and water level), the ESP32 microcontroller, and the Android application interface. Sub-components are widely available and cheap due to cheap sensors in market and open-source microcontrollers.
- Mobile Application Development:
  The mobile application development (in Kotlin) will require an upfront fee such as software development and testing. The downside is that it is a one-time cost, and once the app is developed, it can be used by an unlimited number of users with no further cost.
- Installation Costs:
  Installation is a streamlined process that needs low labour costs due to the fact that the sensors and devices are designed for easy integration into environments like aquariums or fish farms that already exist. Because of simple setup, the system is economical for even smaller operations.

2. Operational and Maintenance Costs

- Low Maintenance Requirements:
  It is a low-maintenance system. Generally, sensors need to be calibrated just a few times a year, while the mobile app is updated regularly to enhance performance. Over time, this makes the system economical, with very few recurring costs.
- Power Consumption:
  The system uses the least amount of energy possible. Electricity expenses can be decreased by using low-power Internet of Things sensors and automating lighting and pumps based on real-time data. Farmers benefit from cheaper operating expenses as a result, especially in large-scale enterprises.
- Cloud Services (Firebase Database):
  There may be minor, ongoing costs associated with storing data on cloud services like Firebase, depending on usage and storage capacity. But typically speaking, this

expense is minimal, particularly when the system is utilized for small- to medium-sized tasks.

3. Long-Term Savings

- Reduced Labor Costs:
  The necessity for manual labour is greatly decreased by automation. Fish growers no longer have to manually adjust pumps and systems or do routine water parameter checks. Over time, this labour cost decrease might result in significant savings, particularly for small-scale farmers who may rely significantly on human resources.
- Lower Risk of Fish Mortality:
  The technology lowers the risk of fish mortality from low water quality, temperature swings, or oxygen deprivation by guaranteeing ideal conditions and automating parameter monitoring and control. Reduced operating losses and increased profitability are the results of fewer stock losses.
- Efficient Resource Management:
  Using real-time data, the automation system optimizes the use of resources like energy and water. Significant savings may result from this, especially in large-scale aquaculture operations with heavy resource usage. Costs and waste can be decreased by adjusting lighting, oxygen levels, and water temperature according to the fish's actual needs.

4. Increased Yields and Revenue

- Improved Fish Health and Growth:
  The method encourages healthier fish by preserving ideal water quality and lowering stress levels in fish, which improves growth rates and yields. The profitability of the farm will increase as healthier fish have a higher chance of reaching market size.
- Higher Fish Production:
  By reducing losses from illnesses, stress, or poor water quality, consistent and precise management of environmental factors helps to enhance output. Revenue for fish farmers can be greatly increased by higher yields.

5. Social and Environmental Benefits

- Positive Environmental Impact:
  The system's resource optimization and energy efficiency lessen the environmental impact of fish farming. The system encourages sustainable activities that are in line with international environmental goals by consuming less energy, water, and chemicals.
- Social Impact:
  The approach boosts local economies, sustains livelihoods, and opens up new prospects in coastal and rural communities by increasing the sustainability and profitability of fish farming. It contributes to food security and poverty reduction by assisting small-scale farmers in achieving better business results.

4 SYSTEM REQUIREMENTS SPECIFICATION

4.1 SOFTWARE REQUIREMENTS

The software for the automated fish farming system consists of two main components: the firmware for the ESP32 microcontroller and the mobile application for monitoring and controlling the system. Additionally, Firebase is used for data storage and real-time communication.

1. **Firmware for ESP32 Microcontroller**

Development Environment:

- **Arduino IDE**: Used for developing and uploading firmware to the ESP32 microcontroller.
- **Operating System**: Compatible with Windows, macOS, and Linux.

Programming Language:

- **Embedded C**: The primary language for programming the ESP32.

2. **Mobile Application**

Development Environment:

- **Android Studio**: Used for developing android mobile applications.
- **Operating System**: Compatible with Windows, macOS, and Linux for development; Android for deployment.

Programming Languages:

- **Kotlin**: For native Android development.
- **Embedded C**: For native iOS development.

Core Features:

- **User Interface**
- **Real-Time Data Visualization**: Displays current sensor readings (pH, dissolved oxygen, temperature, water level) in a user-friendly format.
- **Control Interface**: Provides manual controls for turning on/off the oxygen and water pumps.
- **Threshold Settings**: Allows users to set sensor thresholds for automatic pump operation.
- **Alerts and Notifications**: Sends push notifications for critical events, such as high pH level.
- **Firebase Integration**: Fetches real-time data from Firebase and sends control commands.
- **User Authentication**: Secure login and access control using Firebase Authentication (email/password, Google sign-in, etc.).

3. **Backend (Firebase Real-Time Database)**

Platform:

- **Google Firebase**: A cloud-based platform providing a real-time NoSQL database, authentication, and hosting services.

4. **Additional Requirements**

Internet Connectivity:

- **Wi-Fi Access**: The ESP32 requires a stable Wi-Fi connection for data transmission to Firebase.
- **Data Bandwidth**: Sufficient bandwidth to support real-time updates and remote-control operations, typically at least 2 Mbps.

4.2 HARDWARE REQUIREMENTS

1. **Microcontroller**

ESP32 Microcontroller

2. **Sensors**

- **pH Sensor**:
- **Type**: Analog pH sensor
- **Measurement Range**: 0 to 14 pH
- **Accuracy**: ±0.1 pH

- **Dissolved Oxygen (DO) Sensor**:
- **Type**: Analog dissolved oxygen sensor
- **Measurement Range**: 0 to 20 mg/L
- **Accuracy**: ±0.5 mg/L

- **DHT11 Temperature and Humidity Sensor**:
- **Type**: Digital temperature and humidity sensor
- **Temperature Range**: 0°C to 50°C
- **Temperature Accuracy**: ±2°C
- **Humidity Range**: 20% to 90% RH
- **Humidity Accuracy**: ±5% RH

- **Ultrasonic Sensor**:
- **Type**: Ultrasonic distance sensor
- **Measurement Range**: 2 cm to 400 cm
- **Accuracy**: ±3 mm

3. **Actuators**

- **Oxygen Pump**:
- **Type**: Electric air pump for aquariums
- **Control**: Operated via relay module controlled by the ESP32

- **Water Pump**:
- **Type**: Submersible water pump for aquariums
- **Control**: Operated via relay module controlled by the ESP32

4. **Relay Modules**

- **Relay Module**:
- **Use**: Controls high voltage pumps (oxygen and water pumps) based on ESP32 commands.

4.3 LANGUAGE DESCRIPTION

4.3.1. Embedded C

Embedded C is a variant of the C programming language used for developing firmware for microcontrollers, such as the ESP32 microcontroller in the Automated Fish Farming System.

Usage:

- Firmware Development**:** Embedded C is essential for programming the firmware of microcontrollers. It provides direct access to hardware resources and allows for real-time processing, which is crucial for managing sensors, actuators, and other components in the system.

Description:

- Efficiency**:** Embedded C is known for its efficiency, which is important in environments where memory and processing power are limited. It is used for tasks such as controlling the actuators (like oxygen and water pumps) based on sensor data (pH, temperature, dissolved oxygen).
- Real-time Processing**:** Embedded C is suitable for real-time applications because it allows precise control over timing and interrupts, necessary for handling sensor data and ensuring timely responses to environmental changes.

Key Features:

- **Direct Hardware Manipulation**: Allows for the direct control of hardware peripherals, which is essential for interacting with sensors and actuators.
- **Memory Management**: Provides low-level access to memory, enabling efficient use of resources in embedded systems.
- **Portability**: While specific to the hardware architecture, the basic syntax of C makes it relatively portable across different microcontroller platforms.

4.3.2. Kotlin (Android Studio)

Kotlin is a modern programming language used for Android app development and is widely utilized in Android Studio for developing mobile applications.

Usage:

- Mobile Application Development**:** Kotlin is used for developing the mobile application that interfaces with the ESP32 microcontroller in the Automated Fish Farming System. This application provides real-time data monitoring, alerts, and control capabilities to the users.

Description:

- Interoperability: Kotlin is fully compatible with Java, which makes it easy to integrate into existing Android projects. It also provides features such as null safety and extension functions that improve code readability and maintenance.
- Android Studio Integration**:** Kotlin is the preferred language for Android development because of its seamless integration with Android Studio, the official IDE for Android development. This provides access to the latest Android APIs and tools for building robust, efficient mobile applications.

Key Features:

- **Concise Syntax**: Kotlin's syntax is designed to be more concise and expressive than Java, making it easier to write and understand code. This reduces development time and the likelihood of errors.

- **Null Safety**: Kotlin offers a powerful null safety mechanism, which helps prevent null pointer exceptions, a common source of bugs in Java-based Android apps.
- **Coroutines**: Kotlin Coroutines simplify asynchronous programming by allowing developers to write code that looks sequential but runs concurrently, which is useful for handling real-time data updates from the fish farming system.

4.3.3. NoSQL (Firebase Database)

NoSQL databases are designed to manage large volumes of unstructured, semi-structured, and structured data. Firebase is a popular NoSQL cloud database used for real-time data synchronization in mobile and web applications.

Usage:

- Backend Data Management: In the Automated Fish Farming System, Firebase is used as the backend database to store and manage sensor data, such as water temperature, pH levels, and dissolved oxygen. It allows real-time data updates and monitoring through the mobile app.

Description:

- Real-time Synchronization**:** Firebase provides real-time data synchronization across all devices connected to the application. Any change in the data (such as a new sensor reading) is instantly updated in the database, which is crucial for providing accurate and timely information to the users.
- Scalability: Firebase is highly scalable and can handle a large number of users and data operations simultaneously without compromising performance. This makes it suitable for applications that need to scale quickly.

4.3.4. MySQL

MySQL is an open-source relational database management system (RDBMS) widely used for storing, organizing, and retrieving structured data. It is a crucial component of many web-based applications and serves as the backend for numerous systems.

Usage:

- Data Storage and Management: MySQL stores data in tables and allows efficient management through CRUD (Create, Read, Update, Delete) operations.
- Database Integration in Applications: Used as the backend for websites, mobile apps, and IoT systems to manage user data, sensor readings, or any other structured information.
- Data Analytics: Facilitates complex queries for generating reports, trends, and insights from the stored data.

Description:

- Relational Database Model: MySQL organizes data into tables with predefined relationships, enabling structured storage and easy access to interconnected datasets.
- Open Source and Community-Driven: MySQL is free to use, with a large community that continuously improves its features and provides support.
- Client-Server Architecture: MySQL operates using a server-client model where the database server handles requests sent by clients via SQL commands.

Key Features:

- **Scalability:** Suitable for applications ranging from small-scale systems to large, enterprise-level solutions.
- **High Performance:** Optimized for high-speed operations, making it efficient for handling large datasets.
- **Cross-Platform Support:** Compatible with multiple operating systems, including Windows, Linux, and macOS.
- **Security Features:** Provides robust security measures like user authentication, data encryption, and access control.
- **Integration:** Easily integrates with various programming languages (e.g., PHP, Python, Java) and frameworks, making it versatile for developers.

4.3.5. PHP (Hypertext Preprocessor)

PHP is an open-source server-side scripting language designed for web development. It is widely used to create dynamic and interactive web applications.

Usage:

- Backend Web Development: PHP powers the server-side logic of web applications, handling tasks like user authentication, form handling, and database interactions.
- Dynamic Content Creation: It enables the generation of dynamic content by embedding scripts into HTML.

Description:

- Server-Side Processing: PHP runs on the server, generating HTML, JSON, or XML responses for the client browser.
- Database Integration: It works seamlessly with databases like MySQL and PostgreSQL, making it a preferred choice for creating data-driven applications.

Key Features:

- **Cross-Platform Support:** PHP runs on multiple platforms, including Windows, macOS, and Linux, and is compatible with most web servers like Apache and Nginx.
- **Ease of Use:** Its simple syntax and extensive library support make PHP beginner-friendly and highly flexible for developers.
- **Dynamic and Interactive Pages:** PHP scripts can generate content based on user input, ensuring a personalized experience.
- **Integration:** PHP integrates well with HTML, JavaScript, and CSS to create full-stack web solutions.

4.4 COMPONENTS REQUIRED

Hardware Components

1. ESP32 Microcontroller:

- A powerful microcontroller with Wi-Fi and Bluetooth capabilities.
- Acts as the central control unit for processing data from sensors and managing actuators.

2. pH Sensor:

- Measures the acidity or alkalinity of the water.
- Crucial for maintaining optimal water conditions for fish health.

3. Temperature Sensor:

- Monitors the water temperature.
- Helps in ensuring suitable thermal conditions for aquatic life.

4. Ultrasonic Sensor:

- Measures the water level in the tank.
- Ensures water levels are maintained within a safe range.

5. Water Pump:

- Controls water circulation or replacement.
- Keeps the water clean and oxygenated.

6. Relay Module:

- A switching device used to control high-power components (like pumps) with the microcontroller's low-power output.

7. Oxygen Pump:

- Adds dissolved oxygen to the water.
- Essential for fish survival and maintaining a healthy aquatic environment.

8. Power Supply:

- Provides necessary electrical power to the entire system.

9. Breadboard:

- A tool for prototyping and testing circuit designs without soldering.

10. Jumper Wires:

- Connect various components on the breadboard and microcontroller.

11. LDR (Light Dependent Resistor):

- Detects light intensity.
- May be used for adjusting operations based on ambient lighting conditions.

Software Components

1. Arduino IDE:

- Used to write, compile, and upload code to the ESP32 microcontroller.
- Provides a platform for programming the system's behaviour.

2. Android Studio:

- Used for developing the mobile application.
- Provides a user-friendly interface for monitoring and controlling the fish farming system.

5 SYSTEM DESIGN

5.1 ARCHITECTURE DIAGRAM

The architecture diagram outlines a high-level understanding of the system, detailing the components and interactions. It presents an image of the system itself and the relationships, constraints and boundaries between the components.

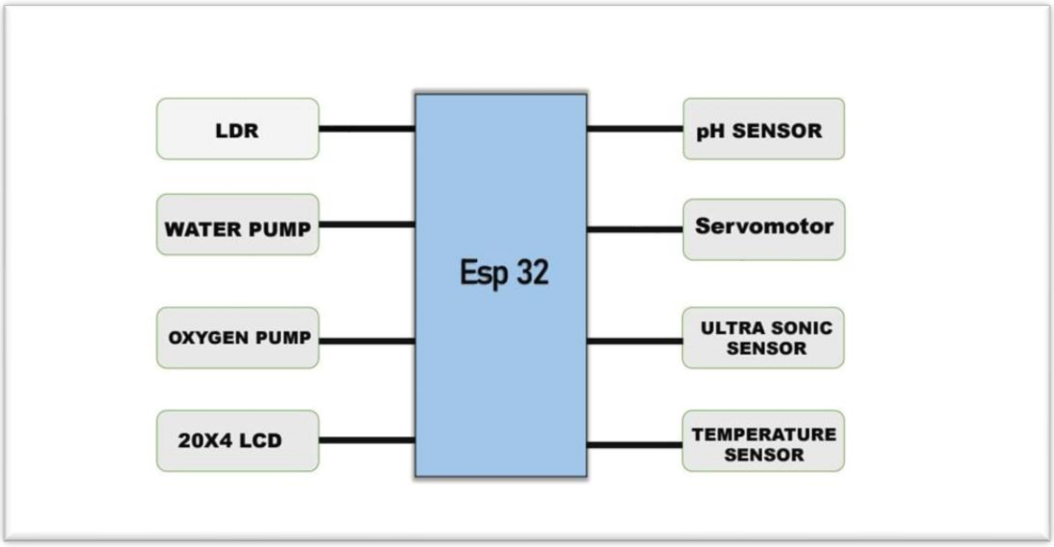


Fig 5.1 Architecture Diagram

5.2 CIRCUIT DIAGRAM

In other words, Circuit diagram is the graphical representation of an electrical circuit. An actual wiring is seen in a circuit diagram.

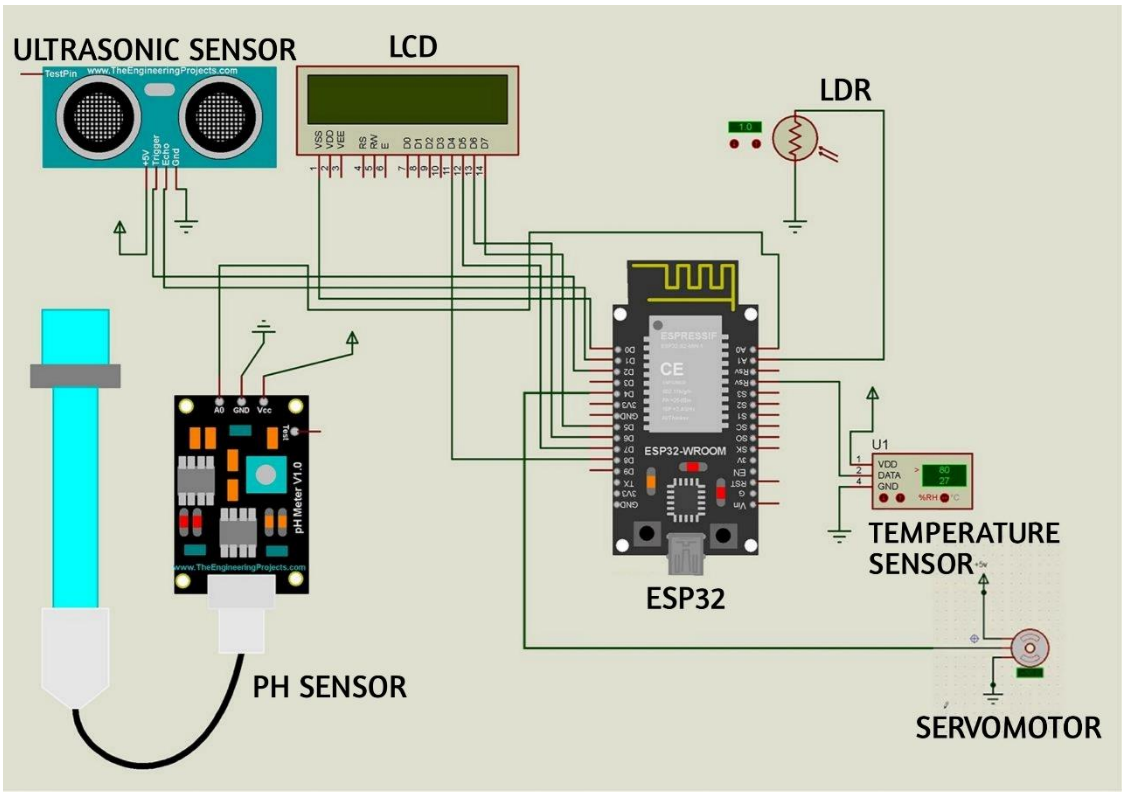


Fig 5.2 Circuit Diagram

5.3 TABLE DESIGN

Table design lets you fine-tune your table definition.

5.3.1 User details

User_details		
field name	Type	constraints
Userid	Int	Primary Key
first_name	varchar (20)	NOT NULL
last_name	varchar (20)	NOT NULL
Email	varchar (40)	NOT NULL
phone_no	varchar (12)	NOT NULL
username	varchar (15)	NOT NULL
password	varchar (10)	NOT NULL

5.3.2 Aquatic details

Aquatic_ details		
Field name	Type	Constraints
adid	Int	Primary Key
value	double (10,2)	NOT NULL
type	varchar (30)	NOT NULL
date	Date	

5.3.3 Fish adaptability

Fish_adaptability		
Field name	Type	Constraints
faid	int	Primary Key
ph_start_range	double (10,2)	NOT NULL
ph_end_range	double (10,2)	NOT NULL
temp_start_range	double (10,2)	NOT NULL
temp_end_range	double (10,2)	NOT NULL
oxygen_start_range	double (10,2)	NOT NULL
oxygen_end_range	double (10,2)	NOT NULL
species	varchar (30)	NOT NULL

5.4 DATA-FLOW DIAGRAM

A flow of data through a process or a system is represented in a data flow diagram.

Level 0 DFD: A high-level view of how the system would work, the user interacts with the system, which collects data from sensors, processes that data using ESP32 and updates it to Firebase Data is fetched and controlled via the mobile application.

Level 1 DFD: Enhances functionalities such as collecting sensor data, processing the data and user commands.

Level 2 DFD: It focused on certain processes e.g. handling sensor data and controlling actuators based on threshold values.

LEVEL 0

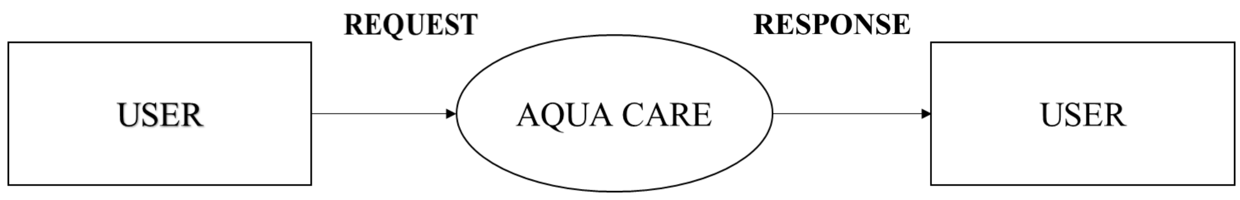


Fig 5.4.1 Data flow diagram Level 0

LEVEL 1

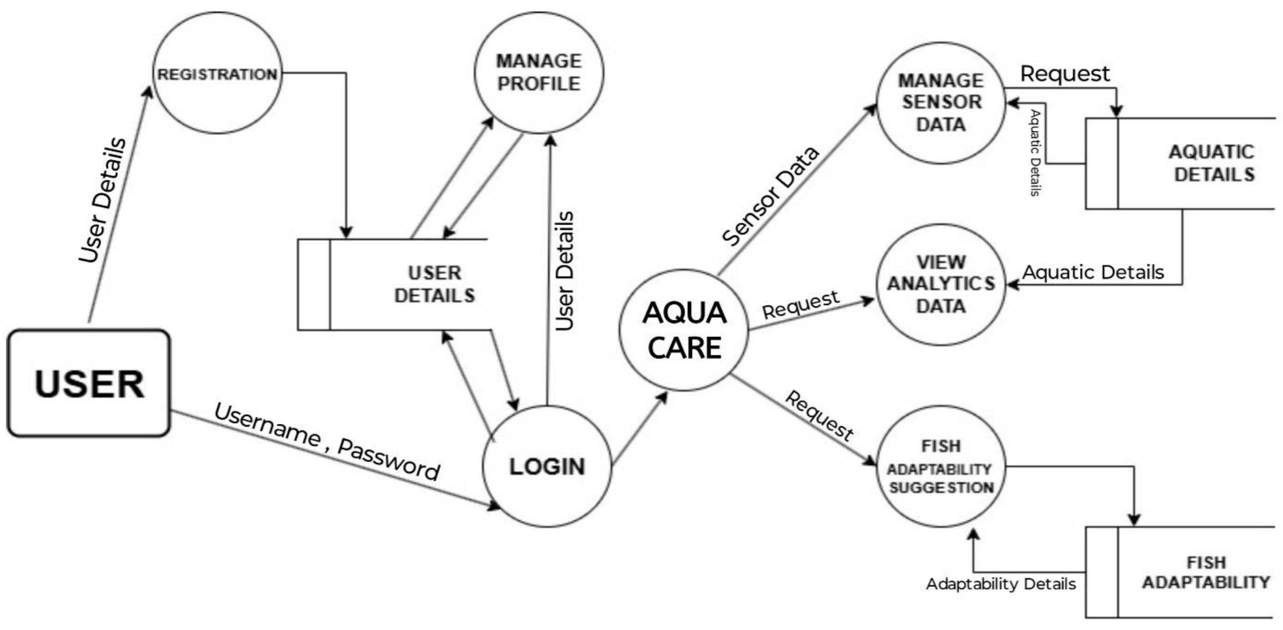


Fig 5.4.2 Data flow diagram Level 1

LEVEL 2

MANAGE SENSOR DATA

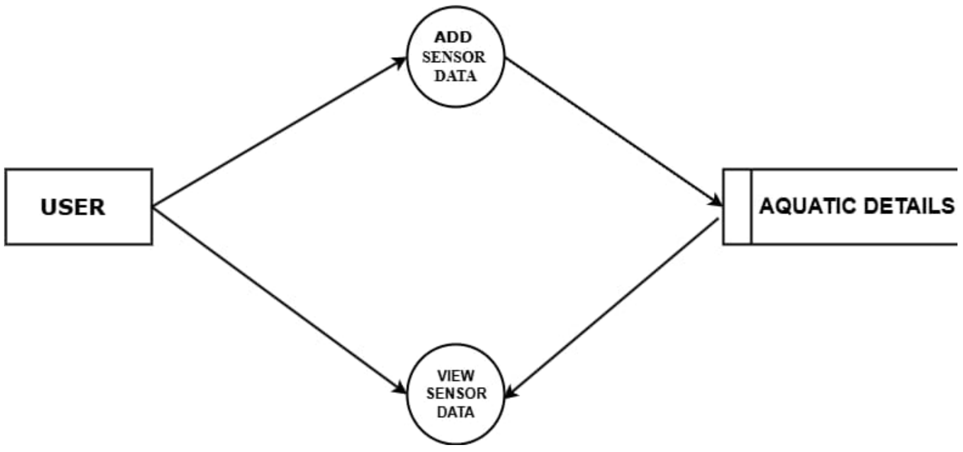


Fig 5.4.3 Data flow diagram Level 2

5.5 USE CASE DIAGRAM

Use Case diagrams are a graphical way of showing a user's potential interactions with a system. A use case diagram will depict multiple use cases and different types of users which your system has and it is often accompanied with another diagram too. Use cases are either circles or ellipses.

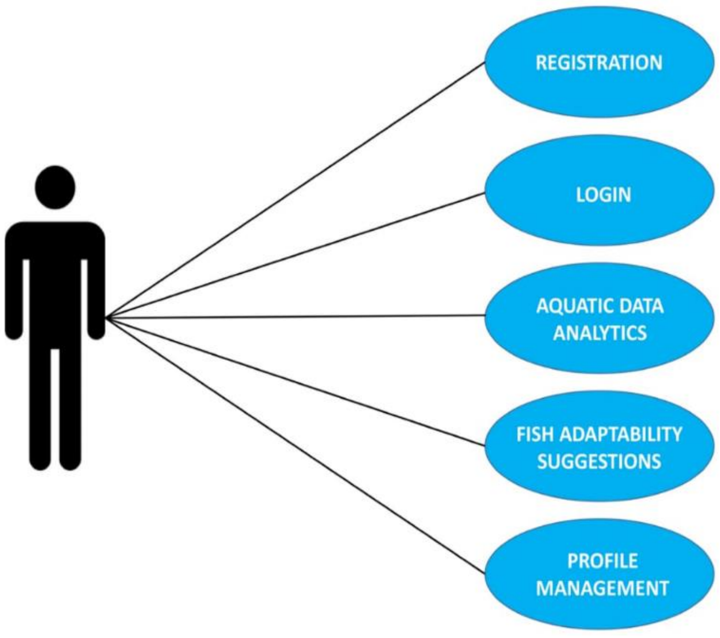


Fig 5.5 Use case Diagram

6 MODULE DESCRIPTION

1. ESP32 Microcontroller Module

- Type: ESP32 Wi-Fi/Bluetooth Microcontroller
- Purpose: Serves as the central control unit of the system.
- Functionality:
    i. Collect sensor data from pH, temperature, water level (ultrasonic), and dissolved oxygen sensors.
    ii. Controls the water pump based on sensor readings and user inputs.
    iii. Communicates with the Firebase cloud to upload sensor data and receive control commands from the mobile app.
    iv. Provides real-time monitoring and control of the aquarium system.

2. pH Sensor Module

- Sensor Type: Analog pH Sensor
- Purpose: Measures the acidity or alkalinity of the aquarium water.
- Functionality:
    i. Continuously monitors the pH level of the water to ensure it is suitable for aquatic life.
    ii. Sends pH readings to the ESP32, which transmits them to Firebase for real-time monitoring via the mobile app.
    iii. Alerts users if pH levels exceed predefined thresholds.

3. Temperature Sensor Module

- Sensor Type: Digital Temperature Sensor
- Purpose: Monitors the water temperature.
- Functionality:
    i. Ensures the aquarium maintains an optimal temperature for the aquatic species.
    ii. Continuously sends temperature data to the ESP32, which uploads it to Firebase for monitoring.

4. Ultrasonic Sensor Module

- Sensor Type: Ultrasonic Distance Sensor (e.g., HC-SR04)

- Purpose: Measures the water level in the aquarium.
- Functionality:
    1. Sends ultrasonic waves and measures the time taken for them to bounce back to determine the water level.
    2. When the water level drops below a specified threshold, it triggers the ESP32 to activate the water pump to refill the aquarium.
    3. Data is sent to Firebase to allow remote monitoring of the water level through the mobile app.

5. Dissolved Oxygen Sensor Module

- Sensor Type: Dissolved Oxygen (DO) Sensor
- Purpose: Monitors the dissolved oxygen level in the aquarium water.
- Functionality:
    1. Continuously measures the concentration of dissolved oxygen to ensure it stays within the required range for the aquatic life.
    2. Sends oxygen level data to the ESP32, which uploads it to Firebase for real-time monitoring.

6. Water Pump Actuator Module

- Actuator Type: Electric Water Pump
- Purpose: Maintains the water level in the aquarium.
- Functionality:
    1. Controlled by the ESP32 based on input from the ultrasonic sensor.
    2. Automatically activates when the water level drops below the preset threshold and deactivates when the level reaches the optimal point.
    3. The user can also control the pump manually via the mobile app, with commands being sent through Firebase to the ESP32.

7. Firebase Cloud Module

- Platform: Firebase Real time Database
- Purpose: Cloud-based real-time database that stores sensor data and facilitates communication between the ESP32 and the mobile app.
- Functionality:

i. Stores all sensor data, including pH, temperature, water level, and dissolved oxygen readings, allowing users to access it from anywhere.
ii. Receives user commands from the mobile app and sends them to the ESP32 for execution.
iii. Supports real-time updates, so any changes in sensor data or commands are instantly reflected in the mobile app.

8. Mobile App Module

- Type: Mobile Application
- Purpose: Serves as the user interface for monitoring and controlling the aquarium system.
- Functionality:
  i. Displays real-time data from all sensors, including pH, temperature, water level, and dissolved oxygen readings.
  ii. Allows the user to control the water pump and potentially other actuators.
  iii. Sends commands to Firebase, which are retrieved and executed by the ESP32.
  iv. Provides notifications or alerts if any sensor readings exceed predefined thresholds.

7 SYSTEM IMPLEMENTATION

SOURCE CODE

7.1. Arduino code

```
#include <WiFi.h>
#include <LiquidCrystal_I2C.h>
#include <FirebaseESP32.h>
#include <DHT.h>
#include <ESP32Servo.h>
#define WIFI_SSID "kiranks"
#define WIFI_PASSWORD "10341069876"
#define FIREBASE_HOST "https://aquacare-63a47-default-rtdb.asia-
southeast1.firebasedatabase.app/"
#define FIREBASE_AUTH "AIzaSyDvFNeQTYpYzrhHd-o1dE3n2gViab7pqMk"
FirebaseConfig config;
FirebaseAuth auth;
FirebaseData firebaseData;
#define TRIG_PIN 22
#define ECHO_PIN 23
#define PH_SENSOR_PIN 34
#define LDR_PIN 32
#define DHT_PIN 5
#define DHT_TYPE DHT11
DHT dht(DHT_PIN, DHT_TYPE);
#define WATER_PUMP_PIN 25
#define OXYGEN_PUMP_PIN 26
#define FEEDING_SERVO_PIN 15
LiquidCrystal_I2C lcd(0x27, 20, 4);
Servo feedingServo;
```

```
void setup() {
  Serial.begin(115200);
  Wire.begin(21, 19);
  lcd.begin(20, 4);
  lcd.backlight();
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi...");
  lcd.setCursor(3, 3);
  lcd.print("Connecting to Wi-Fi.");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    lcd.setCursor(6, 1);
    lcd.print("AQUA-FARM");
  }
  Serial.println("Connected to Wi-Fi.");
  config.host = FIREBASE_HOST;
  config.signer.tokens.legacy_token = FIREBASE_AUTH;
  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(PH_SENSOR_PIN, INPUT);
  pinMode(LDR_PIN, INPUT);
  pinMode(WATER_PUMP_PIN, OUTPUT);
  pinMode(OXYGEN_PUMP_PIN, OUTPUT);
  digitalWrite(WATER_PUMP_PIN, LOW);
  digitalWrite(OXYGEN_PUMP_PIN, LOW);
  dht.begin();
  feedingServo.attach(FEEDING_SERVO_PIN);
  feedingServo.write(0);
  lcd.clear();
}
void loop() {
  String automatic = "off";
  String waterPump = "off";
  String feedingSwitch = "off";
  if (Firebase.getString(firebaseData, "/settings/automatic")) {
    automatic = firebaseData.stringData();
    Serial.print("Automatic mode from Firebase: ");
    Serial.println(automatic);
  } else {
    Serial.print("Failed to read '/settings/automatic', reason: ");
    Serial.println(firebaseData.errorReason());
  }
  if (Firebase.getString(firebaseData, "/settings/feedingSwitch")) {
    feedingSwitch = firebaseData.stringData();
    Serial.print("Feeding Switch value from Firebase: ");
    Serial.println(feedingSwitch);
  } else {
    Serial.print("Failed to read '/settings/feedingSwitch', reason: ");
    Serial.println(firebaseData.errorReason());
  }
  if (Firebase.getString(firebaseData, "/settings/waterPump")) {
    waterPump = firebaseData.stringData();
    Serial.print("Water pump setting from Firebase: ");
    Serial.println(waterPump);
  } else {
    Serial.print("Failed to read '/settings/waterPump', reason: ");
    Serial.println(firebaseData.errorReason());
  }
  if (feedingSwitch == "on") {
    feedingServo.write(90);
    Serial.println("Feeding switch is ON: Servo turned to 90 degrees.");
  } else {
    feedingServo.write(0);
    Serial.println("Feeding switch is OFF: Servo remains at 0 degrees.");
  }
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2;
  int phValue = (analogRead(PH_SENSOR_PIN));
  float phLevel = map(phValue, 0, 4095, 0, 14);
  int ldrValue = analogRead(LDR_PIN);
  int lightIntensity = map(ldrValue, 0, 4095, 0, 100);
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  lcd.setCursor(0, 1);
  lcd.print("Dist:");
  lcd.print(int(distance));
  lcd.setCursor(0, 2);
  lcd.print("pH:");
  lcd.print(int(phLevel-7));
  lcd.setCursor(10, 1);
  lcd.print("Light:");
  lcd.print(lightIntensity);
  lcd.print("%");
  lcd.setCursor(10, 2);
  lcd.print("Temp:");
  lcd.print(temperature);
  lcd.setCursor(0, 3);
  lcd.print("Hum:");
  lcd.print(int(humidity));
  lcd.print("%");
  Firebase.setFloat(firebaseData, "/Sensor/distance", distance);
  Firebase.setFloat(firebaseData, "/Sensor/phLevel", phLevel-7);
  Firebase.setFloat(firebaseData, "/Sensor/light", lightIntensity);
  Firebase.setFloat(firebaseData, "/Sensor/temperature", temperature);
  Firebase.setFloat(firebaseData, "/Sensor/humidity", humidity);
  if (automatic == "on") {
    Serial.println("Running in Automatic Mode");
    lcd.setCursor(0, 0);
    lcd.print("Automatic mode ON  ");
    lcd.print(automatic);
    if (distance > 8) {
      digitalWrite(WATER_PUMP_PIN, HIGH);
      digitalWrite(OXYGEN_PUMP_PIN, LOW);
    }
    if (lightIntensity <= 50) {
      digitalWrite(OXYGEN_PUMP_PIN, HIGH);
      digitalWrite(WATER_PUMP_PIN, LOW);
    }
  } else if (automatic == "off") {
    Serial.println("Running in Manual Mode");
    lcd.setCursor(0, 0);
    lcd.print("Automatic mode OFF ");
```

```
    lcd.print(automatic);
    if (waterPump == "on") {
      digitalWrite(WATER_PUMP_PIN, HIGH);
      Serial.println("Water pump turned on in manual mode.");
    } else {
      digitalWrite(WATER_PUMP_PIN, LOW);
      Serial.println("Water pump turned off in manual mode.");
    }
    digitalWrite(OXYGEN_PUMP_PIN, LOW);
  }
  delay(2000);
}
```

7.2 PHP CODE

7.2.1 Connection Code

```
<?php
$con=new mysqli("localhost","root","","aqua_farm");
if(!$con)
{
  die("sql connection error");
}
$server_path="../upload";
$base_path="../upload/";
?>
```

7.2.2 Registration Page

```
<?php
require 'connection.php';
$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
```

```
$phone = $_REQUEST['phone'];
$email = $_REQUEST['email'];
$password = $_REQUEST['password'];
$image = $_FILES['image']['name'];
$data = array();
$post = array();
$sql = "SELECT * FROM users WHERE phone_number='$phone' OR email='$email'";
$result = $con->query($sql);
$count = $result->num_rows;
if ($count > 0) {
    $sql_phone = "SELECT * FROM users WHERE phone_number='$phone'";
    $result_phone = $con->query($sql_phone);
    $count_phone = $result_phone->num_rows;
    $sql_email = "SELECT * FROM users WHERE email='$email'";
    $result_email = $con->query($sql_email);
    $count_email = $result_email->num_rows;
    if ($count_phone > 0 && $count_email > 0) {
        $post = array(
            "status" => false,
            "message" => "Phone Number and Email Already Exist",
            "userData" => $data
        );
    } elseif ($count_phone > 0) {
        $post = array(
            "status" => false,
            "message" => "Phone Number Already Exists",
            "userData" => $data
        );
    } elseif ($count_email > 0) {
        $post = array(
            "status" => false,
            "message" => "Email Already Exists",
            "userData" => $data
        );
    }
} else {
    $random_name = rand(1000,1000000) . "-" . $image;
    $image_tmp_name = $_FILES["image"]["tmp_name"];
    $upload_name = strtolower($random_name);
    $upload_name = preg_replace('/\s+/', '-', $upload_name);
    $upload_name = $server_path . "/" . $upload_name;
    if(move_uploaded_file($image_tmp_name , $upload_name)){
        $photo = basename($upload_name);
    } else {
        $photo = "";
    }
    $sql = "INSERT INTO `users`(`first_name`, `last_name`, `email`, `phone_number`,
`password`, `photo`)
            VALUES ('$first_name', '$last_name', '$email', '$phone', '$password', '$photo')";
    $result = $con->query($sql);
    $count = $con->affected_rows;
    if($count > 0){
        $last_id = $con->insert_id;
        $sq = "SELECT * FROM users WHERE userid=$last_id";
        $res = $con->query($sq);
        $row = $res->fetch_assoc();

        $data[] = array(
            "userid" => ($row['userid'] == null ? "" : $row['userid']),
            "first_name" => ($row['first_name'] == null ? "" : $row['first_name']),
            "last_name" => ($row['last_name'] == null ? "" : $row['last_name']),
            "phone" => ($row['phone_number'] == null ? "" : $row['phone_number']),
            "email" => ($row['email'] == null ? "" : $row['email']),
            "password" => ($row['password'] == null ? "" : $row['password']),
            "photo" => ($row['photo'] == null ? "" : $base_path.$row['photo'])
        );
        $post = array(
            "status" => true,
            "message" => "Registration Successful",
            "userData" => $data
        );
    } else {
        $post = array(
            "status" => false,
            "message" => "Registration Failed",
            "userData" => $data
        );
    }
}
echo json_encode($post);
?>
```

7.2.3 Login Page

```
<?php
require 'connection.php';
$phone = $_REQUEST['phone'];
$password = $_REQUEST['password'];
$data = array();
```

```
$post = array();
$sql = "SELECT * FROM users WHERE phone_number='$phone' AND
password='$password'";
$result = $con->query($sql);
$count = $result->num_rows;
if($count > 0) {
   $row = $result->fetch_assoc();
   $data[] = array(
      "userid" => ($row['userid'] == null ? "" : $row['userid']),
      "first_name" => ($row['first_name'] == null ? "" : $row['first_name']),
      "last_name" => ($row['last_name'] == null ? "" : $row['last_name']),
      "phone" => ($row['phone_number'] == null ? "" : $row['phone_number']),
      "email" => ($row['email'] == null ? "" : $row['email']),
      "photo" => ($row['photo'] == null ? "" : $base_path.$row['photo'])
   );
   $post = array(
      "status" => true,
      "message" => "Login Successful",
      "userData" => $data
   );
} else {
   $post = array(
      "status" => false,
      "message" => "Invalid phone number or password",
      "userData" => $data
   );
}
echo json_encode($post);
?>
```

7.2.4 View Analysis

```
<?php
ini_set('display_errors', 1);
error_reporting(E_ALL);
require 'connection.php';
$date=$_REQUEST['date'];
$sql = "SELECT * FROM sensor_data WHERE date='$date'";
$result = $con->query($sql);
if ($result->num_rows > 0) {
   $data = [];
   while ($row = $result->fetch_assoc()) {
      $data[] = $row;
   }
   echo json_encode(["status" => true, "message" => "Data retrieved successfully", "data" =>
$data]);
} else {
   echo json_encode(["status" => false, "message" => "No data found"]);
}
$con->close();
?>
```

7.2.5 Edit Profile

```
<?php
require 'connection.php';
$userid = $_REQUEST['userid'];
$data = array();
$first_name = $_REQUEST['first_name'];
$last_name = $_REQUEST['last_name'];
$phone = $_REQUEST['phone'];
```

```
$email = $_REQUEST['email'];
$sql_check_email = "SELECT * FROM users WHERE email = ? AND userid != ?";
$stmt_check_email = $con->prepare($sql_check_email);
$stmt_check_email->bind_param("si", $email, $userid);
$stmt_check_email->execute();
$res_check_email = $stmt_check_email->get_result();
if ($res_check_email->num_rows > 0) {
    $data = array("status" => false, "message" => "The email address is already exists.");
    echo json_encode($data);
    exit; // Stop further execution if email is already in use
}
$stmt_check_email->close();
$sql_check_phone = "SELECT * FROM users WHERE phone_number = ? AND userid != ?";
$stmt_check_phone = $con->prepare($sql_check_phone);
$stmt_check_phone->bind_param("si", $phone, $userid);
$stmt_check_phone->execute();
$res_check_phone = $stmt_check_phone->get_result();
if ($res_check_phone->num_rows > 0) {
    $data = array("status" => false, "message" => "The phone number is already exists.");
    echo json_encode($data);
    exit;
}
$stmt_check_phone->close();
$sql_update = "UPDATE users SET first_name = ?, last_name = ?, email = ?, phone_number
= ? WHERE userid = ?";
$stmt_update = $con->prepare($sql_update);
$stmt_update->bind_param("ssssi", $first_name, $last_name, $email, $phone, $userid);

if ($stmt_update->execute()) {
    $data = array("status" => true, "message" => "Update successful");
} else {
    $data = array("status" => false, "message" => "Update failed: " . $stmt_update->error);
}
$stmt_update->close(); // Close statement
$con->close(); // Close connection
echo json_encode($data);
?>
```

7.3 KOTLIN CODE

7.3.1 Dashboard Activity

```
package com.project.aquafarm.dashboard

import android.animation.ObjectAnimator
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.view.View
import android.view.animation.CycleInterpolator
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.AppCompatButton
import androidx.core.content.ContextCompat
import com.google.firebase.database.*
import com.project.aquafarm.R
import com.project.aquafarm.analysis.AnalysisActivity
import com.project.aquafarm.databinding.ActivityDashBoardBinding
import com.project.aquafarm.profile.ProfileActivity
import com.project.aquafarm.suggestion.SuggestionActivity
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.OkHttpClient
import okhttp3.Request
import okhttp3.RequestBody.Companion.toRequestBody
import org.json.JSONObject
import java.io.IOException
import java.text.SimpleDateFormat
import java.util.*
import java.util.concurrent.TimeUnit

class DashBoardActivity : AppCompatActivity(), View.OnClickListener {
   private lateinit var binding: ActivityDashBoardBinding
   private lateinit var database: DatabaseReference

   private val previousValues = mutableMapOf<String, String>()
   private var lastWaterPumpState: String? = null

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      enableEdgeToEdge()
      binding = ActivityDashBoardBinding.inflate(layoutInflater)
      setContentView(binding.root)


      binding.profileIcon.setOnClickListener {
         val intent = Intent(applicationContext, ProfileActivity::class.java)
         startActivity(intent)
      }
      initializeDatabaseListener()

      val analysisBtn = findViewById<AppCompatButton>(R.id.analysisBtn)
      shakeButton(analysisBtn)

      analysisBtn.setOnClickListener {
         navigateToAnalysis()
      }

      binding.autoControlSwitch.setOnCheckedChangeListener { _, isChecked ->
         val state = if (isChecked) "on" else "off"
         updateSwitchState("automatic", state)

         binding.waterPumpSwitch.isEnabled = !isChecked
         if (isChecked) {
            binding.waterPumpSwitch.isChecked = false
            updateSwitchState("waterPump", "off")
         }
      }

      binding.waterPumpSwitch.setOnCheckedChangeListener { _, isChecked ->
         val state = if (isChecked) "on" else "off"
```

```
            updateSwitchState("waterPump", state)

            binding.autoControlSwitch.isEnabled = !isChecked
            if (isChecked) {
                binding.autoControlSwitch.isChecked = false
                updateSwitchState("automatic", "off")
            }
        }

        binding.feedingSwitch.setOnCheckedChangeListener { _, isChecked ->
            val state = if (isChecked) "on" else "off"
            updateSwitchState("feedingSwitch", state)
        }
        binding.suggestionBtn.setOnClickListener(this)
    }

    private fun initializeDatabaseListener() {
        database = FirebaseDatabase.getInstance().reference

        database.addValueEventListener(object : ValueEventListener {
            override fun onDataChange(snapshot: DataSnapshot) {
                try {
                    val phValue = snapshot.child("Sensor/phLevel").value.toString()
                    val waterLevel = snapshot.child("Sensor/distance").value.toString()
                    val ambienceLight = snapshot.child("Sensor/light").value.toString()
                    val temperature = snapshot.child("Sensor/temperature").value.toString()
                    val oxygen = snapshot.child("Sensor/humidity").value.toString()
                    val waterPumpState = snapshot.child("settings/waterPump").value.toString()
                    val currentDate =
                        SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).format(Date())
                    val currentTime = SimpleDateFormat("HH:mm",
Locale.getDefault()).format(Date())
                    binding.tempValue.text = temperature

                    val isCurrentlyOn = waterPumpState == "on"
                    when {
                        waterLevel > 8.toString() && !isCurrentlyOn -> {
                            updateSwitchState("waterPump", "on")
                            showWaterPumpAlert(
                                "Water Pump ON",
                                "Water level is high. Pump is turned ON."
                            )
                        }
                        waterLevel < 5.toString() && isCurrentlyOn -> {
                            updateSwitchState("waterPump", "off")
                            showWaterPumpAlert(
                                "Water Pump OFF",
                                "Water level is low. Pump is turned OFF."
                            )
                        }
                    }
                    when {
                        phValue < 6.5.toString() -> {
                            showPhAlert(
                                "Low pH Alert",
                                "pH level is too low ($phValue). Consider adding alkaline solutions."
                            )
                        }

                        phValue > 8.5.toString() -> {
                            showPhAlert(
                                "High pH Alert",
                                "pH level is too high ($phValue). Consider adding acidic solutions."
                            )
                        }
                    }

                    updateProgressBarWithValue(
                        "phLevel", phValue,
                        binding.progressPH,
                        binding.phValue
                    )
                    updateProgressBarWithValue(
                        "temperature",
                        temperature, binding.progressTemp, binding.tempValue
                    )

                    updateProgressBarWithValue(
                        "waterLevel",
                        waterLevel,
                        binding.progressWater,
                        binding.waterValue
                    )
                    updateProgressBarWithValue(
                        "light",
                        ambienceLight,
                        binding.progressLight,
                                    binding.lightValue
                                )
                                updateProgressBarWithValue(
                                    "oxygen",
                                    oxygen,
                                    binding.progressOxygen,
                                    binding.o2value
                                )

                                if (shouldSendData(
                                        phValue,
                                        temperature,
                                        ambienceLight,
                                        oxygen,
                                        waterLevel
                                    )
                                ) {
                                    sendDataToPhpServer(
                                        phValue,
                                        temperature,
                                        ambienceLight,
                                        oxygen,
                                        waterLevel,
                                        currentDate,
                                        currentTime
                                    )
                                }
                            } catch (e: Exception) {
                                Log.e("FirebaseError", "Error fetching data: ${e.message}")
                            }
                        }

                        override fun onCancelled(error: DatabaseError) {
                            Toast.makeText(
                                applicationContext,
                                "Failed to read sensor data: ${error.message}",
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    })
                    }

                    private fun shouldSendData(vararg values: String): Boolean {
                        return values.any { it != "N/A" }
```

```
}

private fun sendDataToPhpServer(
    phLevel: String,
    temperature: String,
    ambientLight: String,
    oxygen: String,
    waterLevel: String,
    currentDate: String,
    currentTime: String
) {
    val json = JSONObject().apply {
        put("ph_value", phLevel)
        put("temperature", temperature)
        put("ambience_light", ambientLight)
        put("oxygen", oxygen)
        put("water_level", waterLevel)
        put("date", currentDate)
        put("time", currentTime)
    }

    val jsonString = json.toString()
    val requestBody = jsonString.toRequestBody("application/json".toMediaTypeOrNull())
    val request = Request.Builder()
        .url("http://campus.sicsglobal.co.in/Project/Aqua_farm/api/store_sensor_data.php")
        .post(requestBody)
        .addHeader("Content-Type", "application/json")
        .build()

    val client = OkHttpClient.Builder()
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        .writeTimeout(30, TimeUnit.SECONDS)
        .build()

    client.newCall(request).enqueue(object : okhttp3.Callback {
        override fun onFailure(call: okhttp3.Call, e: IOException) {
            Log.e("PHP Server", "Failed to send data: ${e.message}")
            runOnUiThread {
                Toast.makeText(
                    this@DashBoardActivity,
                    "Failed to send data to server.",
                    Toast.LENGTH_SHORT
                ).show()
            }
        }

        override fun onResponse(call: okhttp3.Call, response: okhttp3.Response) {
            val responseBody = response.body?.string()
            if (response.isSuccessful) {
                Log.d("PHP Server", "Data sent successfully: $responseBody")
            } else {
                Log.e("PHP Server", "Server Error: ${response.code}")
            }
        }
    })
}

private fun updateProgressBarWithValue(
    key: String,
    value: String?,
    progressBar: ProgressBar,
    textView: TextView
) {
    if (value.isNullOrBlank() || value == "N/A") {
        return
    }

    val numericValue = value.toFloatOrNull() ?: 0f
    when (progressBar.id) {
        R.id.progressTemp -> progressBar.max = 100
        R.id.progressWater -> progressBar.max = 100
        R.id.progressPH -> progressBar.max = 14
    }
    val normalizedValue = when (progressBar.id) {
        R.id.progressTemp -> (numericValue / 100 * progressBar.max).toInt()
        R.id.progressWater -> (numericValue / 20 * progressBar.max).toInt()
        else -> numericValue.toInt()
    }

    ObjectAnimator.ofInt(progressBar, "progress", normalizedValue).apply {
        duration = 500
        start()
    }

    textView.text = value

    val textColor: Int = when {
        normalizedValue < 30 -> ContextCompat.getColor(this, R.color.color_low)
        normalizedValue in 30..70 -> ContextCompat.getColor(this, R.color.color_normal)
        else -> ContextCompat.getColor(this, R.color.color_high)
    }

    textView.setTextColor(textColor)
}

private fun updateSwitchState(switch: String, state: String) {
    database.child("settings").child(switch).setValue(state)
        .addOnSuccessListener {
            Log.d("Firebase", "$switch switch state updated to $state")
        }
        .addOnFailureListener {
            Log.e("FirebaseError", "Failed to update $switch state: ${it.message}")
        }
}

private fun shakeButton(button: AppCompatButton) {
    val shake = ObjectAnimator.ofFloat(
        button,
        "translationX",
        0f,
        25f,
        -25f,
        20f,
        -20f,
        10f,
        -10f,
        5f,
        -5f,
        0f
    )
    shake.duration = 5000
    shake.interpolator = CycleInterpolator(1f)
    shake.start()
}

private fun navigateToAnalysis() {
    val intent = Intent(this, AnalysisActivity::class.java)
    startActivity(intent)
}

override fun onClick(v: View?) {
    when (v?.id) {
        R.id.suggestionBtn -> {
            suggestionClick()
        }
    }
}

private fun suggestionClick() {

    val intent = Intent(this@DashBoardActivity, SuggestionActivity::class.java)
    startActivity(intent)
}

private fun showWaterPumpAlert(title: String, message: String) {
    if (lastWaterPumpState != title) {
        runOnUiThread {
            AlertDialog.Builder(this)
                .setTitle(title)
                .setMessage(message)
                .setPositiveButton("OK") { dialog, _ -> dialog.dismiss() }
                .show()
        }
        lastWaterPumpState = title
    }
}

private fun showPhAlert(title: String, message: String) {
    AlertDialog.Builder(this)
        .setTitle(title)
        .setMessage(message)
        .setPositiveButton("OK") { dialog, _ -> dialog.dismiss() }
        .show()
}
}
```

8 SYSTEM TESTING

Methods of Testing

Testing is the process of finding bugs in a program. It helps improve the quality of the software. It must be done thoroughly and with the help of specialist testers. System testing is the process of checking whether the developed system functions according to the objectives and requirements.

8.1. Unit Testing

Unit testing of software applications is performed during the development (coding) phase of an application. The objective of unit testing is to isolate a section of code and verify its correctness. In procedural programming, a unit may be an individual function or procedure. The goal of unit testing is to isolate each part of the program and ensure that the individual components function correctly.

Unit testing is usually performed by the developer. It aims to discover errors in the individual modules of the system, whereas integration testing focuses on the decision logic, control flow, recovery procedures, throughput, capacity, and timing characteristics of the entire system.

8.2. Integration Testing

Integration testing verifies that different modules of the system interact correctly when integrated. This is done after unit testing to ensure that all units work together as expected.

Integration testing is performed to:

- Detect errors in data flow between modules.
- Ensure smooth communication between different system components.
- Validate the correctness of interfaces and interactions between units.

8.3. System Testing

This testing ensures that all the system components work together properly. After completing the project, system testing was performed based on control flow and correct output. This process verified that the project functioned correctly with system configurations and other dependencies to produce the desired results.

8.4. Validation Testing

Validation testing ensures that the product meets the client's needs and expectations. It verifies that the software functions correctly in its intended environment.

Validation testing can be best demonstrated using the V-Model, where each phase of development is tested to ensure compliance with requirements.

8.5. Black Box Testing

Black box testing focuses on testing the system without knowledge of its internal workings. The tester evaluates the functionality of the application based on the expected inputs and outputs.

- Ensures that all functionalities work as per requirements.
- Helps detect incorrect or missing functionalities.
- Focuses on user experience and external behavior.

8.6. White Box Testing

White box testing, also known as glass-box testing, tests the internal structure, design, and implementation of the system. It involves examining the code to verify logical flows, error handling, and performance optimization.

- Tests the internal logic and structure of the code.
- Identifies hidden errors within the application's source code.
- Requires knowledge of programming and system architecture.

8.7. TEST CASES

Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
IOT_001	Powering on ESP32	ESP32 should turn on and connect to Wi-Fi	ESP32 successfully powered on and connected to Wi-Fi	Pass
IOT_002	Connecting ESP32 to Firebase	Data should be sent to Firebase Realtime Database	Data successfully transmitted to Firebase	Pass
IOT_003	pH Sensor Data Transmission	pH values should be displayed in the database	pH data updated in Firebase and displayed correctly	Pass
IOT_004	Temperature Sensor Data Transmission	Temperature readings should update in the database	Temperature data updated in Firebase and displayed correctly	Pass
IOT_005	Ultrasonic Sensor Water Level Detection	Correct water level should be displayed	Water level data displayed accurately in Firebase	Pass
IOT_006	Oxygen Sensor Data Transmission	Oxygen level should update in the database	Oxygen level data displayed correctly in Firebase	Pass
IOT_007	Servo Motor Activation (Fish Feeder)	Feeder should rotate and dispense food	Feeder successfully dispensed food on command	Pass
IOT_008	Automatic Water Pump Activation	Water pump should turn on when water level is low	Pump activated automatically at low water level	Pass

Table 8.7.1 Test case: System

Test Case ID	Description	Expected Result	Actual Result	Pass/Fail
APP_001	User Registration/Login	User should be able to register and log in	Registration and login functions working correctly	Pass
APP_002	Dashboard Display	Real-time sensor data should be visible	Dashboard correctly displays real-time data	Pass
APP_003	Manual Fish Feeder Control	Pressing the button should activate the servo motor	Fish feeder activated via app command	Pass
APP_004	Manual Water Pump Control	Pump should turn on/off through the app	Water pump controlled successfully via app	Pass
APP_005	Automation Rule for Water Pump	Pump should activate when water is low	Water pump activated automatically as per rule	Pass
APP_006	App Connectivity with Firebase	Data should update in real-time	Real-time updates reflected in Firebase and app	Pass

Table 8.7.2 Test case: Mobile App

9 CONCLUSION

The development of an automated fish farming system using the ESP32 microcontroller successfully addresses the challenges associated with traditional aquaculture monitoring. It guarantees accurate monitoring of vital factors including pH, temperature, water level, and dissolved oxygen by integrating the ESP32 microcontroller with a variety of sensors. Real-time data transmission from the system to Firebase enables remote monitoring and control via an intuitive smartphone application. By automating crucial procedures like water level control and sending out timely alerts for any anomalies, this project not only makes maintenance easier but also improves the stability and health of the aquatic ecosystem. This project's integration of IoT and mobile technology demonstrates the potential for smart systems in daily life, encouraging increased awareness and concern for aquatic creatures' welfare. This project significantly reduces human effort, improves accuracy, and promotes sustainable fish farming practices. With its efficient, data-driven approach, the system offers a scalable and cost-effective solution for modern aquaculture management.

10 FUTURE ENHANCEMENTS

Future enhancements for this automated fish farming system aim to further improve efficiency, sustainability, and scalability. Integrating AI-based predictive analysis can help anticipate water quality trends, allowing proactive measures to be taken before conditions become critical. Implementing IoT edge computing will enable faster decision-making by processing data locally on the ESP32, reducing cloud dependency. Additional sensors, such as ammonia level detectors and salinity sensors, can enhance monitoring capabilities, ensuring more precise control of water quality. An automated feeding system can be introduced to dispense food based on fish activity and predefined schedules, optimizing feeding efficiency and reducing waste. To promote sustainability, a solar-powered version of the system can be developed, making it suitable for remote fish farms with limited electricity access. Expanding the system to support multi-farm monitoring through a centralized dashboard will allow users to manage multiple fish farms seamlessly. Lastly, integrating voice and SMS alerts can enhance real-time communication, ensuring that farmers receive critical updates even in areas with limited internet access. These advancements will make the system more intelligent, cost-effective, and adaptable to the evolving needs of modern aquaculture.

11 PUBLICATIONS

Liya Thomas, Karthikeyan K.B, Pranav P. (2025, January 24). Aqua care: Smart Aquarium Monitoring System Using IoT. Advanced Information Science and Computing Systems, International Conference on Advanced Information Science and Computing Systems, 10

12 BIBLIOGRAPHY

[1] A. S., "Smart Aquarium Management System," *Advances in Parallel Computing*, vol. 37, pp. 523–527, 2020. doi: 10.3233/APC200196.

[2] L. Onwuka, A. Adejo, and I. Joseph, "Design and Construction of a Microcontroller-based Automatic Fish Feeding Device," 2011.

[3] A. A., A. A. J., G. B. R., R. S. Nair, M. Sreekumar, and S. P., "Pisces Bot," in *2024 International Conference on E-mobility, Power Control and Smart Systems (ICEMPS)*, Thiruvananthapuram, India, 2024, pp. 01–05. doi: 10.1109/ICEMPS60684.2024.10559334.

[4] N. Ramaiah, D. T. P., S. K. Sridhar, N. Chatterjee, R. S. N., and B. Khadka, "Fish Tank Monitoring System Using IoT," *International Journal of Scientific Research in Science and Technology (IJSRST)*, vol. 7, no. 3, pp. 298–304, May–June 2020.

[5] P. Arthi, M. N., S. H., and J. B., "IoT based Smart Aquaculture Monitoring System for Fish Tank Management," *Journal of Ubiquitous Computing and Communication Technologies*, vol. 6, no. 3, pp. 214–227, 2024.

[6] W.-T. Sung, S.-C. Tasi, and S.-J. Hsiao, "Aquarium Monitoring System Based on Internet of Things," 2022.

[7] T. E. Suherman, M. H. Widianto, and Z. Athalia, "Internet of Things System for Freshwater Fish Aquarium Monitoring and Automation Using Iterative Waterfall," in *2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*, Prapat, Indonesia, 2022, pp. 1–6. doi: 10.1109/ICORIS56080.2022.10031310.

[8] R. P. Shete, A. M. Bongale, and D. Dharrao, "IoT-enabled effective real-time water quality monitoring method for aquaculture," *MethodsX*, vol. 13, p. 102906, 2024. doi: 10.1016/j.mex.2024.102906.

13 APPENDIX

13.1 UI SCREENSHOTS

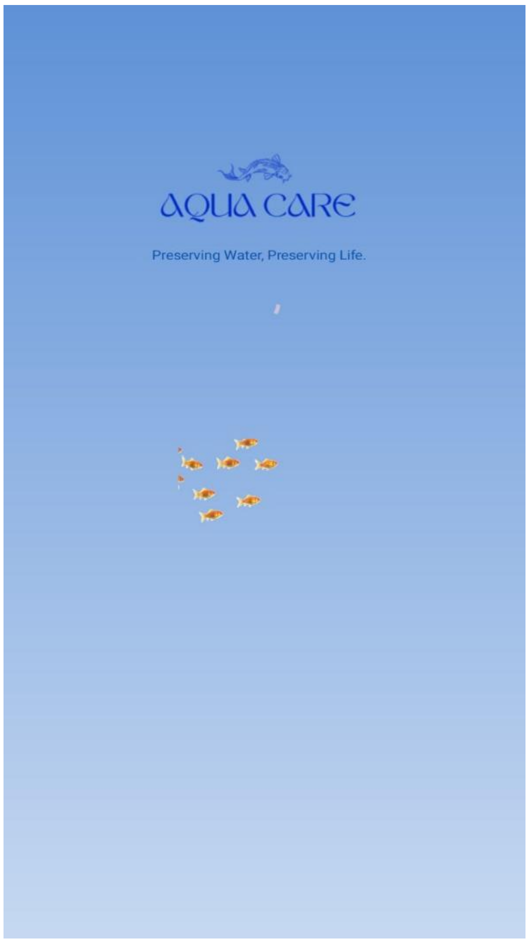


Fig 9.1.1 Splash page

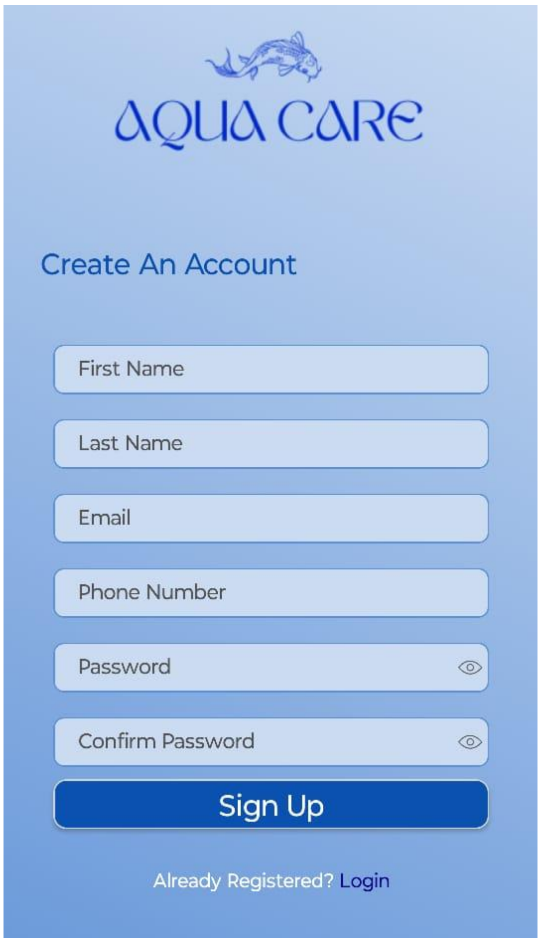


Fig 9.1.2 Sign up page

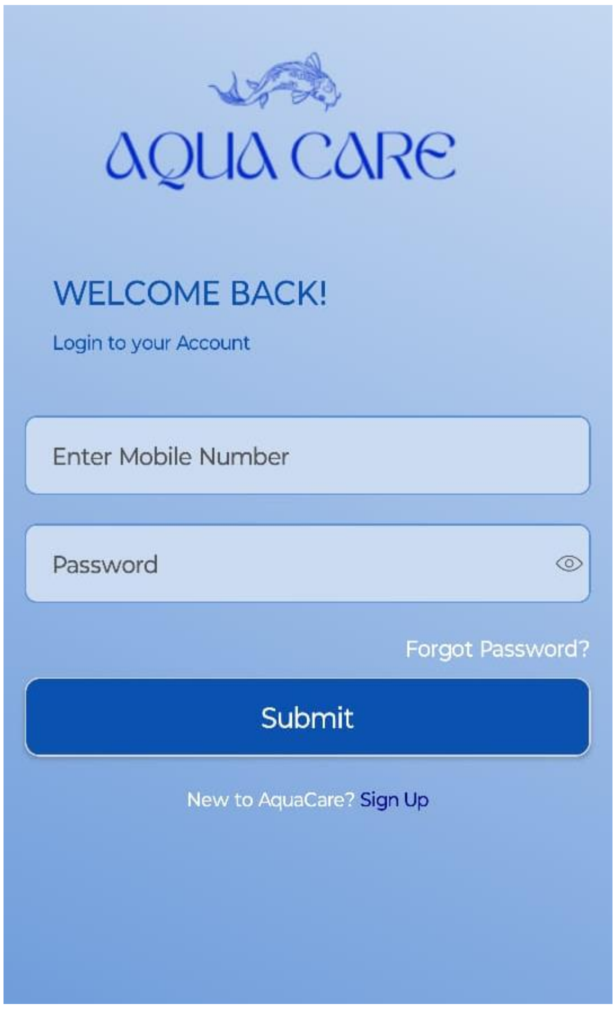
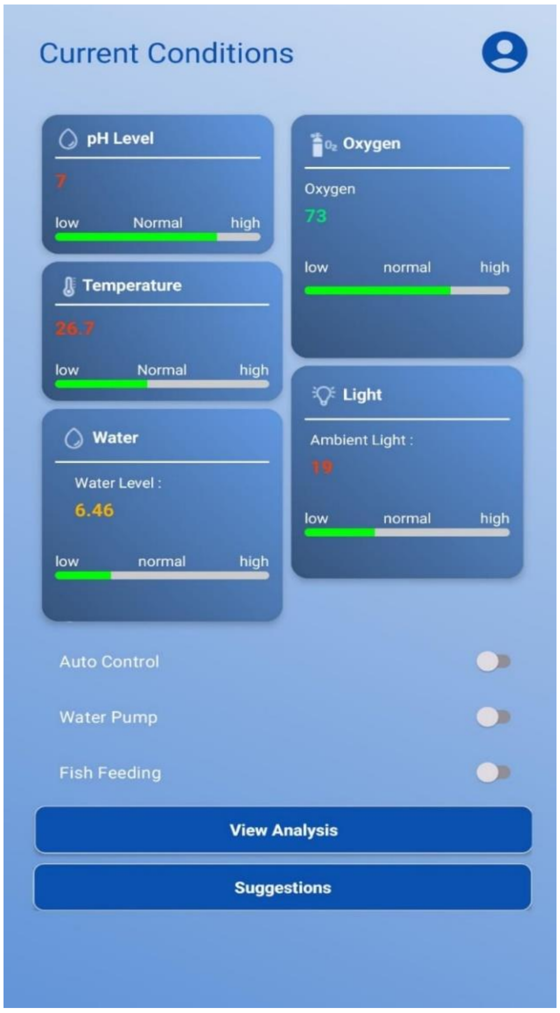


Fig 9.1.3 Login page


Current Conditions

pH Level
7
low Normal high

Temperature
26.7
low Normal high

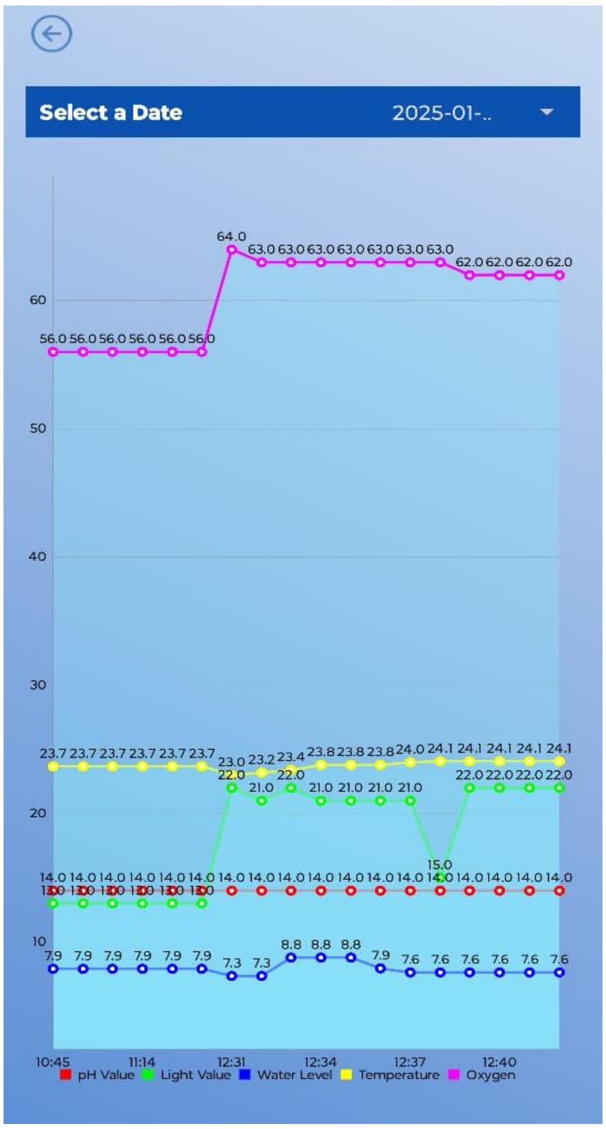
Water
Water Level :
6.46
low normal high

Oxygen
Oxygen
73
low normal high

Light
Ambient Light :
19
low normal high

Auto Control
Water Pump
Fish Feeding

Fig 9.1.4 Home page



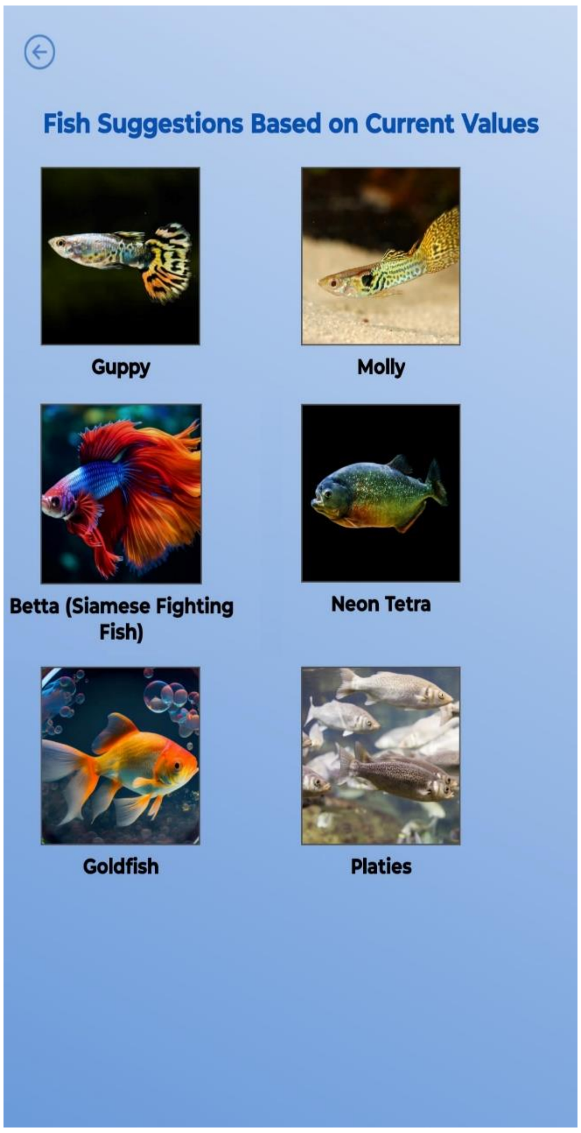
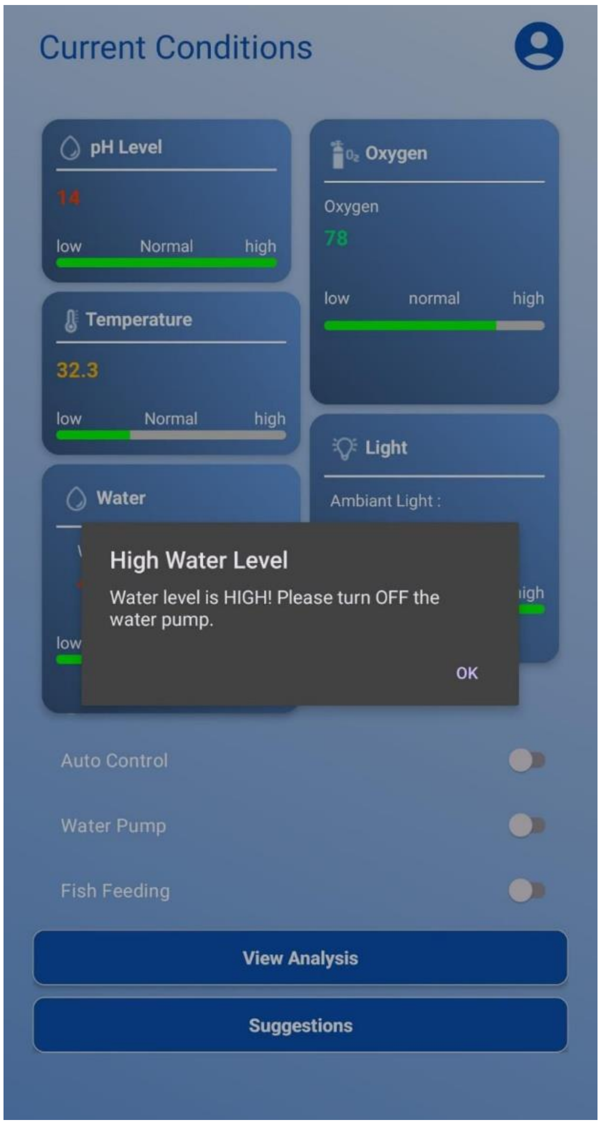

Fish Suggestions Based on Current Values
Guppy
Molly
Betta (Siamese Fighting Fish)
Neon Tetra
Goldfish
Platies

Fig 9.1.5 Data analysis page

Fig 9.1.6 Suggestion page



Current Conditions

pH Level

14

low Normal high

Oxygen

Oxygen

78

low normal high

Temperature

32.3

low Normal high

Light

Ambiant Light :

Water

High Water Level

Water level is HIGH! Please turn OFF the water pump.

OK

Auto Control

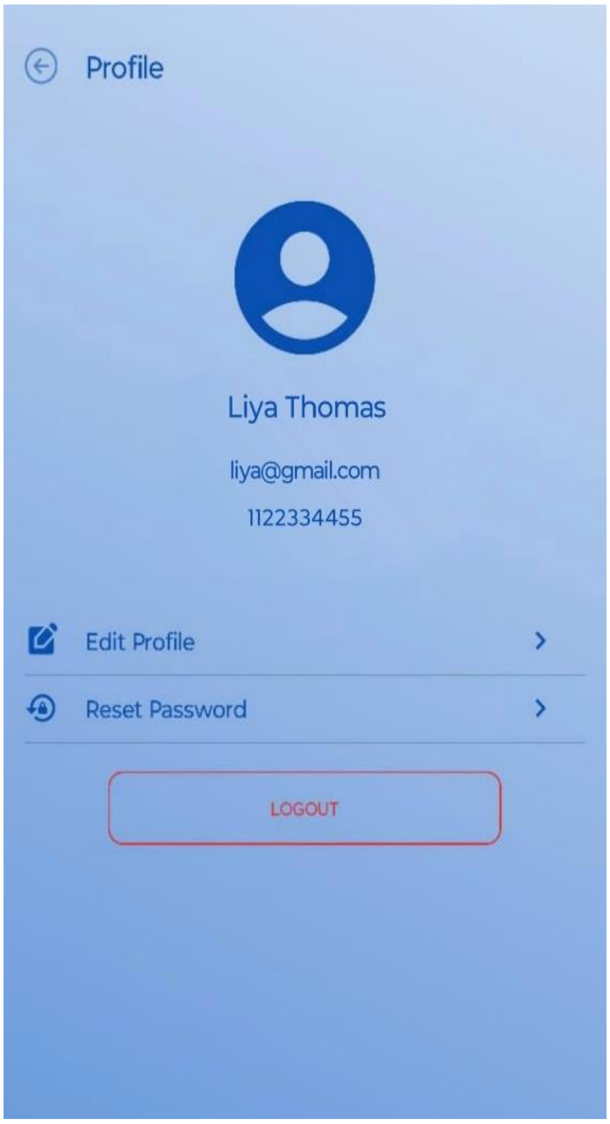
Water Pump

Fish Feeding

View Analysis

Suggestions

Fig 9.1.7 Alert Notification



Profile

Liya Thomas

liya@gmail.com

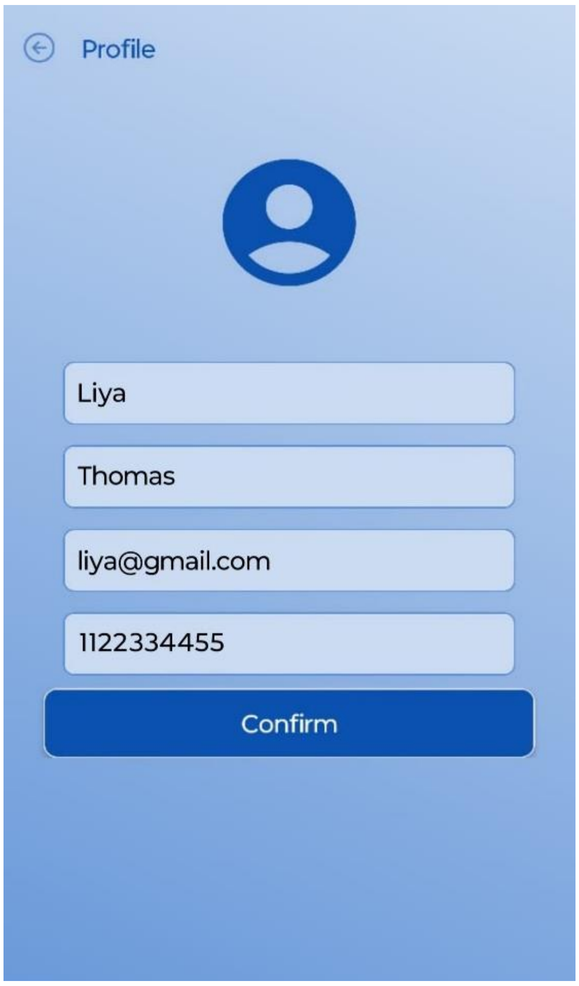
1122334455

Edit Profile

Reset Password

LOGOUT

Fig 9.1.8 Profile page



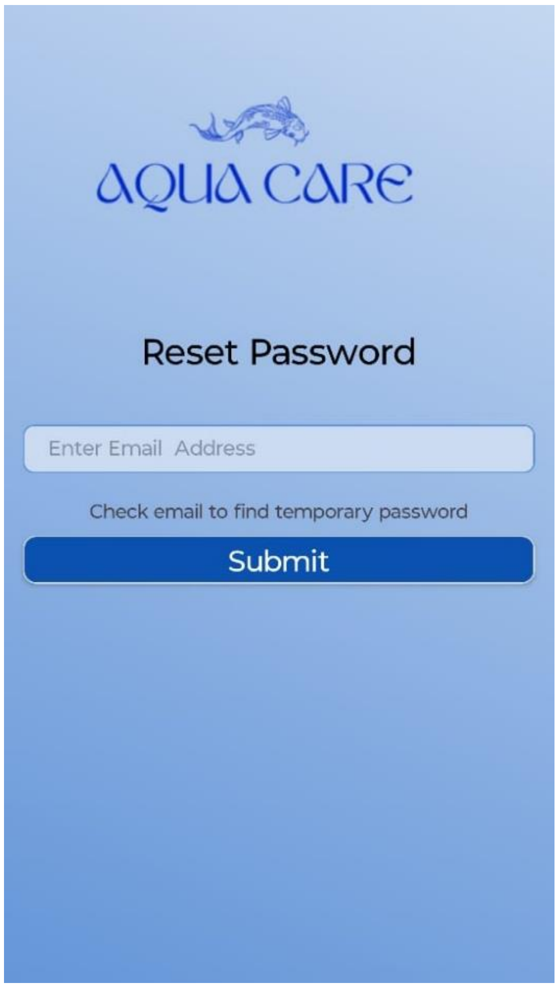


Fig 9.1.9 Edit profile page

Fig 9.1.10 Password reset page

13.2 USER MANUAL

1. Introduction

The Smart Fish Farming Monitoring System is designed to automate and monitor key water quality parameters to ensure a healthy environment for aquatic life. It continuously monitors key water parameters such as pH, temperature, oxygen levels, and water level. The system automates water and oxygen pumps, controls ambient lighting, and provides manual control through a mobile application.

2. Installation Steps

1. Connect the ESP32 microcontroller to a stable power source.
2. Properly install and calibrate the sensors:
   - **pH sensor** – Monitors water acidity/alkalinity.
   - **Temperature sensor** – Tracks water temperature.
   - **Ultrasonic sensor** – Measures water levels.
   - **Oxygen sensor** – Detects oxygen concentration in water.
3. Connect the water pump and oxygen pump for automated control.
4. Ensure the ambient light is properly connected for controlled illumination.

3. System Overview

The system continuously monitors water parameters and automates actions to maintain optimal conditions. The ESP32 collects sensor data and sends it to a Firebase Realtime Database, which can be viewed in a mobile app. The app allows users to set automation rules and manually control devices like the water pump, oxygen pump, and fish feeder.

4. Turning On/Off the Devices

Turning On:

1. Connect the ESP32 microcontroller to a stable power source.
2. Ensure all sensors, motors, and pumps are properly installed and powered.
3. Turn on the mobile app to monitor real-time data.

Turning Off:

1. Use the mobile app to disable pumps and feeder if needed.
2. Disconnect the ESP32 from the power source.
3. Turn off the mobile app if not in use.

5. Using The Devices

Connecting to Wi-Fi and Firebase

1. Power on the ESP32 and connect it to a Wi-Fi network.
2. Verify successful data transmission to Firebase Realtime Database.

Mobile App Installation and Usage

1. Download and install the Fish Farm Monitoring App.
2. Register or log in to the app.
3. View real-time water parameters on the dashboard.
4. Manually control pumps and fish feeder when needed.

6. Trouble Shooting

If data is not updating or devices are not functioning properly:

- **Check Wi-Fi connectivity:** Ensure ESP32 is connected to a stable network.
- **Verify sensor connections:** Make sure all sensors are properly placed and secured.
- **Restart the system:** Power off and on the ESP32 and the mobile app.
- **Recalibrate sensors:** Regularly clean and recalibrate pH and DO sensors for accuracy.
- **Check power supply:** Ensure all components are receiving adequate power.

7. Safety And Maintenance

Safety Guidelines

- Avoid exposing electrical components to water.
- Handle sensors and motors carefully to prevent damage.
- Keep the power supply stable to avoid sudden shutdowns.

Maintenance Tips

- Clean sensors regularly to prevent debris buildup.
- Check and refill the fish feeder as needed.
- Inspect wires and connections for any wear or damage.
- Update the mobile app and firmware when necessary for optimal performance.

8. Technical Information

- **Microcontroller:** ESP32
- **Wireless Connectivity:** Wi-Fi (2.4 GHz)
- **Database:** Firebase Realtime Database
- **Sensors Used:** pH, DO, DHT11, Ultrasonic, Ambient Light
- **Actuators:** Water pump, Oxygen pump, Servo motor (fish feeder)
- **Power Supply:** 5V/12V (depending on components)