VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND DEVELOP AN IOT APPLICATION THAT AUTOMATICALLY CONTROLS THE ENVIRONMENT AFFECTING SHRIMP POND QUALITY BASED ON ABNORMAL SIGNS

By
Tran Nguyen Hoang Hai

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Information Technology/Computer Science/Computer Engineering

Ho Chi Minh City, Vietnam
2024

DESIGN AND DEVELOP AN IOT APPLICATION THAT AUTOMATICALLY CONTROLS THE ENVIRONMENT AFFECTING SHRIMP POND QUALITY BASED ON ABNORMAL SIGNS

APPROVED BY:

_____ ,
Assoc. Prof. Dinh Duc Anh Vu

_____
Mrs. Vo Thi Luu Phuong

THESIS COMMITTEE
(Whichever applies)

ACKNOWLEGMENTS

Words cannot express my gratitude to Assoc. Prof. Dinh Duc Anh Vu and Mr. Nguyen Duy Xuan Bach for your invaluable patience and feedback. I also could not have undertaken this journey without Mr. Bach, who generously provided knowledge and expertise. His wide-ranged knowledge and enthusiasm help me answers a lot of concern during my works.

I am especially appreciative of the editing assistance, feedback sessions, and moral support I received from my classmates and friends. Their assistance streamlines and improves my task.

Lastly, I am deeply thankful to the faculty of the School of Computer Science at the International University, especially MSc. Nguyen Quang Phu, for their constant support from the start of my academic journey. Furthermore, I sincerely appreciate the esteemed members of my reading and examination committee.

Ho Chi Minh city, June 19th, 2024

Student

Tran Nguyen Hoang Hai

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This thesis presents the development and implementation of a comprehensive mobile application designed specifically for shrimp farmers. The application aims to enhance farm management efficiency and optimize yield through data-driven decision-making. Key features of the application include real-time water quality monitoring and the integration of IoT sensors to provide actionable insights, enabling farmers to respond promptly to environmental changes. Field tests and user feedback from pilot deployments in several shrimp farms indicate significant improvements in operational efficiency and overall productivity. The study concludes that the application streamlines farm management practices and contributes to sustainable aquaculture by promoting best practices and reducing the risk of adverse environmental impacts. This innovative tool represents a significant advancement in the digital transformation of aquaculture, offering a scalable solution to meet the evolving needs of shrimp farmers worldwide.

CHAPTER 1

INTRODUCTION

1.1. Background

Shrimp farming is one type of aquaculture business that produces a lot of profits and product for human consumption. Therefore the difficult and complication during the process is understandable. It need constant, close monitoring and observing to immediately provide actions toward shrimp's ponds cause shrimps are very sensitive with environments around them. In that case, the need for an online application to manage and keep track on the shrimp's environments details are very high demand in the shrimp farming field.

1.2. Problem Statement

Shrimp farming faces challenges in operational efficiency and productivity due to limited tools for real-time monitoring and data-driven decision-making. Current methods lack integrated solutions for comprehensive farm management, particularly in monitoring water quality and responding promptly to environmental changes. As a result, farmers often struggle to optimize yield, mitigate risks, and adhere to sustainable practices effectively. There is a critical need for a specialized mobile application that integrates real-time monitoring capabilities and actionable insights, empowering shrimp farmers to enhance farm management practices and achieve sustainable aquaculture outcomes.

1.3. Scope and Objectives

- **Scope:**

The scope of this thesis is to develop and implement a specialized application tailored for shrimp farmers. The application will focus on enhancing farm management practices through real-time monitoring of water quality parameters. It will integrate IoT sensors to collect and analyze data, providing actionable insights for farmers to optimize shrimp health and

maximize yield. The scope also includes usability testing and validation through field trials in real shrimp farming environments.

- **Objectives:**
  - ✓ Develop an easy-to-use application: Design and develop a user-friendly application specifically for shrimp farmers. Emphasize ease of use and ensure accessibility on various platforms, including smartphones, tablets, and desktop computers.
  - ✓ Real-Time Monitoring: Implement features for real-time monitoring of crucial water quality parameters, such as temperature, using IoT sensors. These sensors will provide continuous data, enabling shrimp farmers to make timely and informed decisions to maintain optimal conditions. The system will alert farmers to any critical changes, ensuring a proactive approach to farm management.
  - ✓ Data Integration: Integrate IoT sensor data to analyze and visualize environmental conditions in shrimp ponds. This will provide farmers with actionable insights, helping them to make informed decisions based on real-time data. By visualizing trends and patterns, farmers can optimize their operations and enhance the overall health and productivity of their shrimp ponds..
  - ✓ Develops a robust system tailored for shrimp farming applications, employing appropriate programming languages and technologies, such as:
    - ReactJS: A JavaScript library that supports the creation of user interfaces and enhances the user experience.
    - NodeJS: A JavaScript runtime environment used to develop the back-end system, including APIs.
    - MySQL: A database management system employed to store user information.
    - Arduino: Enables the incorporation of IoT devices into the system.

- Socket.io (optional): Facilitates real-time data transmission between the back-end and front-end components of the application.

By achieving these objectives, the application aims to bridge existing technological gaps in shrimp farming, offering a comprehensive solution for improved farm management and sustainable production practices.

1.4. Assumption and Solution

This thesis assumes that:

- Regarding the data, temperature is the only factor that can be obtained directly from the device. Implementing retrieval of other data requires additional investment of both money and time due to the need for different devices.
- The oxygen bubbler serves as the only automation device. It operates by turning on when the some parameter level reaches a warning threshold and automatically turns off otherwise.
- Due to the cost and time-intensive nature of certain devices to establish, the automation device and the data it received to function might not be the same.

Given the aforementioned conditions:

- Develop a user-friendly website equipped with essential management functionalities.
- Implement remote data collection and automated equipment control in ponds to alleviate manual labor for farmers.

CHAPTER 2

LITURATURE REVIEW/RELATED WORK

2.1 Related application

At the present, there are already some application for farmers and investors keep track and manage their ponds conditions. Here is the list of some bold apps with its feature and disadvantage.

App	Features	Advantage	Disadvantage
Farmext	➢ Keep a pond diary ➢ Expense Management ➢ Storage Management ➢ Decentralized of pond management ➢ Remind the farming progress and the work schedule ➢ Remote technical consultant ➢ Harvest time forecasts, report and FCR calculator ➢ Buy materials online ➢ Connect to control cabinet, environmental monitoring meter	➢ Does not required much technique ➢ Understand revenue and expenditure without documents ➢ Control disease ➢ Reduce management burden	➢ Almost everything from figures, images, changes in nature must be manually entered ➢ No EC measurement ➢ Cannot predict disease ➢ Does not give advice on shrimp foods ➢ Must paid for full features

Jala	➢ Expense Management ➢ Create camps, ponds, lakes, warehouses ➢ Has prices, pictures, simulation chart	➢ Much information to controls	➢ Almost everything from figures, images, changes in nature must be manually entered ➢ Cannot predict disease ➢ Does not give advice on shrimp foods
CIBA Shrimpapp, Cherry Shrimp care	➢ Similar as above apps	➢ Use to received shrimp's information. ➢ Use to calculate pH, radius, number of foods, etc.	➢ No stored information, only use for calculation and diagnosis.

Table 2.1 Related Application

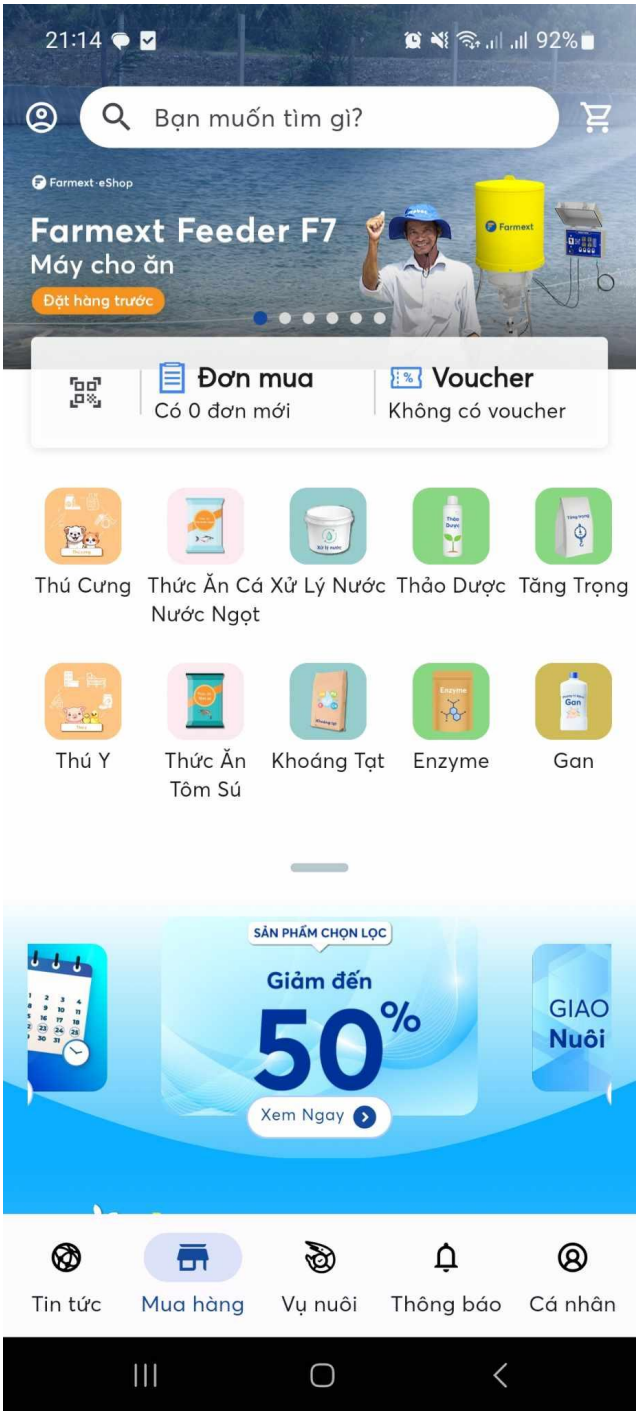


Figure 2.1. Farmext [1]

Farmext is a feature-rich aquaculture farm management tool that lets you keep track of safety information in your pond, calculate production expenses, and figure out earnings.

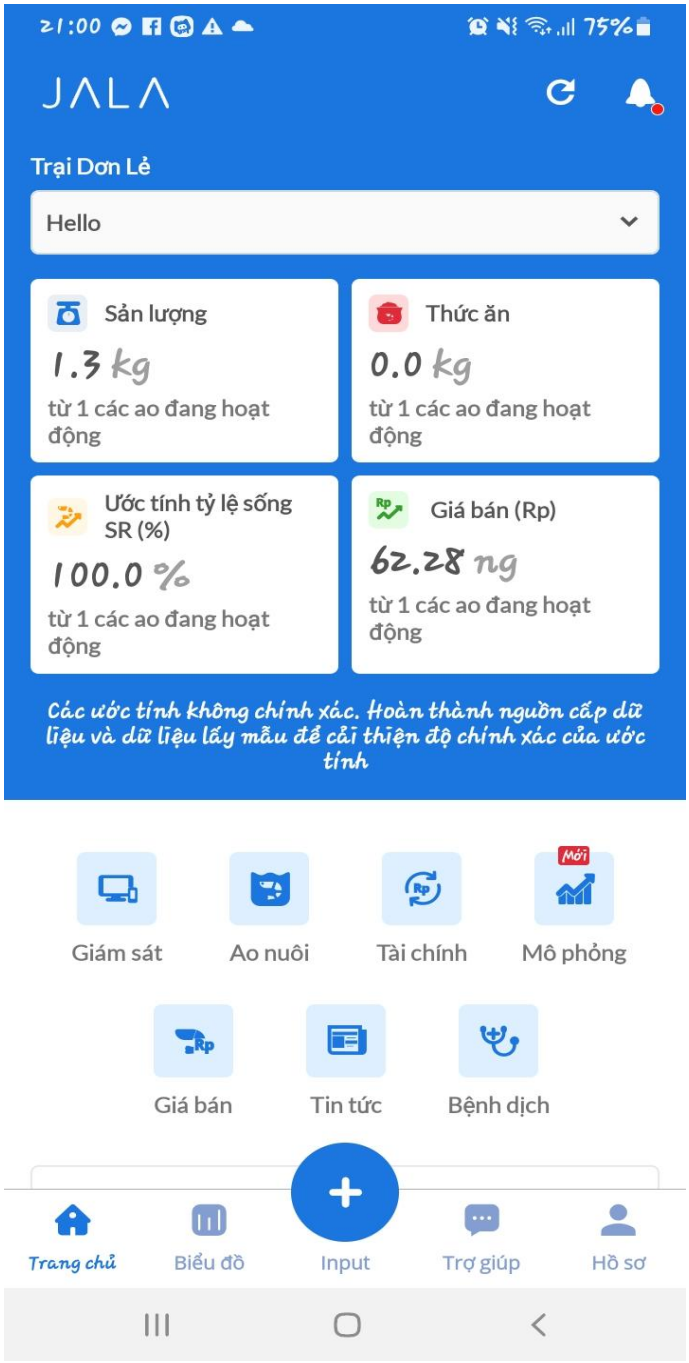


Figure 2.2. Jala [2]

The JALA App puts the power to record, track, and evaluate every step you take in real-time to make the best decision possible for your shrimp farming operation.

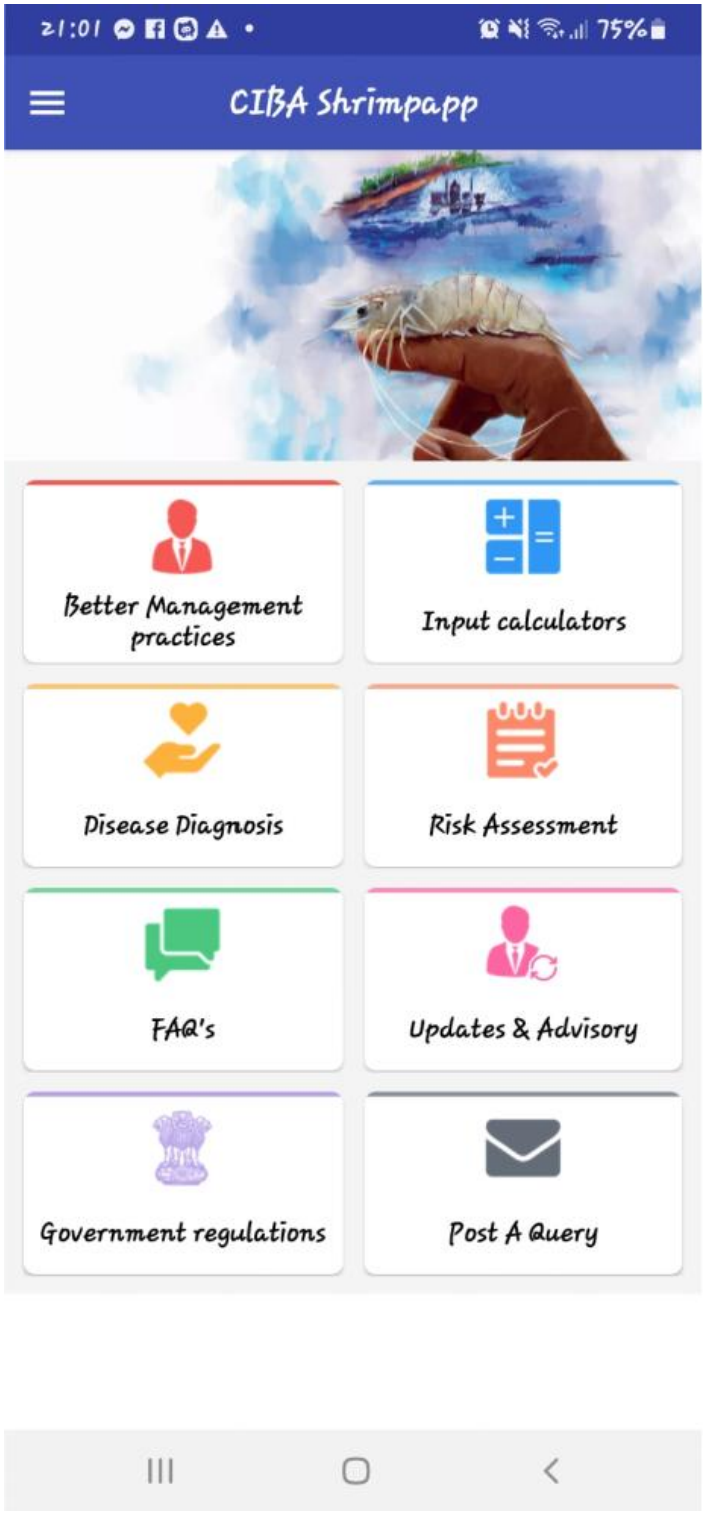


Figure 2.3. CIBA Shrimp [3]

The Central Institute of Brackishwater Aquaculture (CIBA) created the CIBA Shrimp App, a smartphone application, to help shrimp growers. It offers useful tools and materials including illness detection, farming techniques, and control of water quality.

2.2 Critical Index Research

2.2.1. Temperature

Temperature in the context of shrimp farming refers to the measurement of water hotness or coldness in the pond, typically quantified in degrees Celsius (°C) or Fahrenheit (°F). Various shrimp species exhibit specific temperature preferences that influence their growth and overall health. Typically, a temperature range of 25°C to 30°C (77°F to 86°F) is generally considered suitable for many species of shrimp commonly used in aquaculture [5]. The optimal conditions are usually between 31°C to 33°C (87.8°F to 91.4°F).

Scenarios:

- High Temperature: Elevated water temperatures in the shrimp pond can be caused by a number of factors, including high outside temperatures, too much sun exposure, a lack of cover or shade, and insufficient aeration.
- Low Temperature: Water temperatures might fall below the acceptable range due to cold weather, poor heating or circulation systems, and inadequate insulation or protection.

Solutions:

- High Temperature:
  - Use aeration equipment like air pumps and diffusers, or create more water movement with fountains, waterfalls, or other water features to enhance oxygen levels and promote evaporative cooling. Through evaporation, these techniques encourage cooling and improve oxygenation.
  - Provide Shade: Installing shade nets or other structures will shorten the period that water is exposed to the sun. This lessens the temperature of the water and stops it from evaporating too much.

- Use Water Exchange: Use cooler water from a different source to carry out partial water exchanges in order to reduce the temperature overall. This lessens heat buildup and keeps the atmosphere colder.
- Evaporative Cooling: Utilize techniques like fogging, misting, or sprinklers to lower the temperature of the air around you. By encouraging evaporative cooling, these methods successfully reduce the amount of heat in the space.
- Insulation: During periods of extreme heat, cover the water's surface and insulate the pond's edge to minimize temperature fluctuations. This guards against sharp variations in temperature and helps to maintain a more constant temperature.

◆ Low Temperature:

- Heating Systems: To maintain comfortable water temperatures during the winter, install heating devices, such as water heaters or heat pumps. In spite of the colder temperatures, this maintains a steady and appropriate climate.
- Pond Insulation: Use insulating materials to keep the pond warm and protected from the cold outdoors. This keeps the pond's temperature steady and warmer.
- Water Circulation and Aeration: Enhance water circulation and aeration to distribute heat evenly and prevent temperature fluctuations.
- Solar Heating: Use solar heating methods, such as solar collectors or passive solar architecture, to naturally raise the temperature of water. This keeps the environment steady and comfortable by using renewable energy.

2.2.2. pH

Changes in pH in aquaculture can have an impact on the species' health as well as the chemistry, biology, and physical aspects of the aquatic environment. The ideal pH range for shrimp pond water is 7.2 to 8.8 [7]. The range of 7.8 to 8.5 is ideal. The daily variation in pH shouldn't be more than 0.5. Significant pH fluctuations have the potential to shock, weaken,

and cease feeding shrimp and fish. Shrimp will develop slowly, become stunted, and become more prone to illness if high or low pH persists over an extended period of time. Pond pH often rises throughout the day and falls at night. It is advised to take a pH reading at least twice per day.

Causes of increased and decreased pH:

- Ground foundation:
  - Acidic soil: pH decreases.
  - Heavy rain: pH decreases.
- Algae and pond organisms

The pH is influenced by microbes and algae's consumption of CO2. Because of photosynthesis, the quantity of algae will change during the day, and in the afternoon, the pH will be high (8.8 to 9.1). When algae die, the pond's pH drops. Areas used for shrimp aquaculture are low in salinity, and heavy rainfall causes algae to bloom fast. To keep the pH steady, microbes and algae must coexist in equilibrium.

Solutions:

Using slaked or quicklime to repair the bottoms of ponds. It help increase the pH level.

2.2.3. Dissolved oxygen

For shrimp and their surroundings, oxygen is essential. The organisms and metabolic processes that are essential for shrimp aquaculture are impacted by low oxygen levels. Shrimp themselves develop more slowly, have fewer functions, and engage in less foraging. A lack of oxygen can also result in overfeeding and the synthesis of toxic materials. Sustaining appropriate oxygen levels is essential for shrimp aquaculture to be effective.

For shrimp farming, an oxygen level of five parts per million (ppm) or higher is often advised. It is imperative to take prompt measures to augment the oxygen supply in the water if the oxygen level drops below 5 parts per million.

Causes:

- Poor Aeration: Inadequate oxygen transport from the atmosphere to the water in the pond may result from insufficient or inefficient aeration devices.
- High Water Temperature: Compared to colder water, warm water has less dissolved oxygen. Higher water temperatures in the summer or during hot weather can reduce oxygen solubility and cause oxygen depletion.
- Overabundant Organic Matter: As organic matter breaks down, it uses up oxygen. Examples of this include uneaten feed, excrement, and dead creatures. An overabundance of organic materials accumulated in the pond may cause the oxygen content to drop.
- Insufficient Water Exchange: Low water exchange or stagnant circumstances can hinder the body's ability to replace dissolved oxygen from outside sources, which in turn can cause oxygen depletion.

Solutions:

- Use a blower to improve the oxygen supply if the levels are low. To avoid overfeeding, use an aerator, replace the water, and modify the feed amount when the oxygen level falls below 4 ppm.
- Lakes that have a lot of phytoplankton during the daytime may have high oxygen levels. But at night, the density of phytoplankton drops dramatically, necessitating aeration. Keep an eye on the weather since it may impact plants and change the amount of oxygen in the water. This is especially true on rainy and stormy days.
- Adopt a daily aeration schedule to provide enough oxygen for optimum shrimp development. Encourage surface ventilation of the water to enhance oxygen

interaction. To enhance the area of contact, collect any duckweed that may be there in a corner.

2.2.4. Electrical conductivity

In aquaculture systems, electrical conductivity (EC) is commonly measured and used as a proxy for the water's total ion concentration, or the degree of mineralization. It also shows how salinized the water is. The relationship between electrical conductivity and salinity is exactly proportional. In a shrimp pond, electrical conductivity should be measured in the range of 1.0 to 5.0 millisiemens per centimeter (mS/cm).

Effect (The higher the EC, the higher the salinity):

- Low salinity causes the shrimp to molt unevenly, leaving their fragile shell after GDS; it also raises the loss rate repeatedly, which weakens the shrimp's resilience.
- If the salt is too high for the shrimp to withstand, the shrimp will develop slowly and stunted; they may even die in large numbers. Acute hepatopancreatic necrosis and white feces illness will progress in a highly convoluted manner.

Solution:

Only twenty to thirty percent of the pond's water is changed three times a day. The bottom oxygen system and fan are operating at maximum capacity, giving shrimp a robust supply of oxygen. Treat shrimp ponds right away by using microorganisms and controlling algae.

2.3 Hardware

2.3.1 Arduino Uno R3

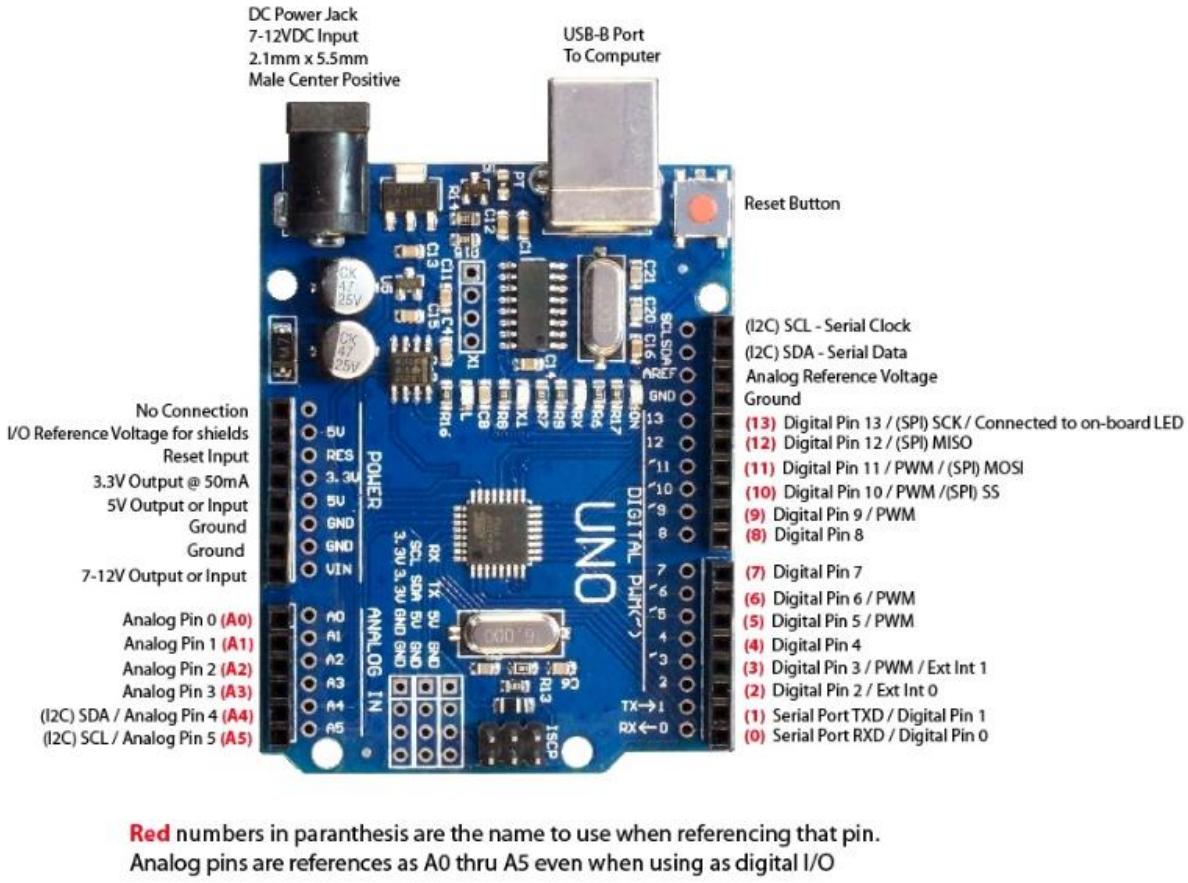


Figure 2.4. Arduino Uno R3

The Arduino Uno R3 [10] is a well-liked and extensively utilized Arduino development board variant. The Arduino circuit is utilized in this project to link and interact with all the other hardware elements.

Micro controllers	ATmega328 8-bit
Operating voltage	5 VDC (supplied via port only USB)
Operating frequency	16 MHz
Consumption stream	About 30 mA
Recommended input voltage	7 – 12 VDC
Limit input voltage	6 – 20 VDC

Number of Digital I/O pins	14 (6 hardware PWM pins)
Analog pin number	6 (10-bit resolution)
Maximum current per I/O pin	30 mA
Maximum output current (5V)	500 mA
Maximum output current (3.3V)	50 mA
Flash Memory	32 KB (ATmega328) with 0.5KB used by bootloader
SRAM	2KB (ATmega328)
EEPROM	1 KB (ATmega328)

Table 1.2. Specifications of Arduino UNO R3

Specific purposes for certain pins:

- **Digital**: Through the pinMode(), digitalWrite(), and digitalRead() primary functions, digital pins 2 through 13 (also known as digital I/O pins) are utilized as input and output pins for digital signals. The microcontroller will be damaged if more than 40mA is supplied through these pins in normal mode, which has an operating voltage of 5V and a current of 20mA.
- **Analog:** includes six analog input pins (A0 through A5) with a resolution of 10 bits (0 to 1023) for each pin. Using the analogRead() method, these pins are utilized to read voltage signals between 0 and 5V (default), or 1024 values.
- **PWM:** pins 3, 5, 6, 9, 10, and 11; these pins are used to supply 8-bit PWM pulses using the analogWrite() method.
- **UART:** Atmega328P allows data transmission through pin 0 (RX) and pin 1 (TX).

In particular, Arduino UNO has 2 pins A4 (SDA) and A5 (SCL) that support I2C/TWI communication with other devices.

2.3.2 DS18B20

The DS18B20 [9] is a small temperature sensor with a built in 12bit ADC. It is simple to connect to a digital input on an Arduino board. The sensor requires few extra components and communicates via a single-wire bus.

Figure 2.5. DS18B20

Temperature measurement range	-55°C to +125°C (-67°F to +257°F)
Accuracy	±0.5°C within measuring range -10°C to +85°C
Resolution	9 to 12 bits
Unique identifier	64-bit for each device

Table 2.3. Functions of DB18B20

2.3.3 ESP8266

ESP8266 NodeMCU Lua CP2102 [8] is a module designed around the ESP8266 microcontroller, with built-in Wi-Fi and a USB-to-serial chip (CP2102) for computer connectivity.

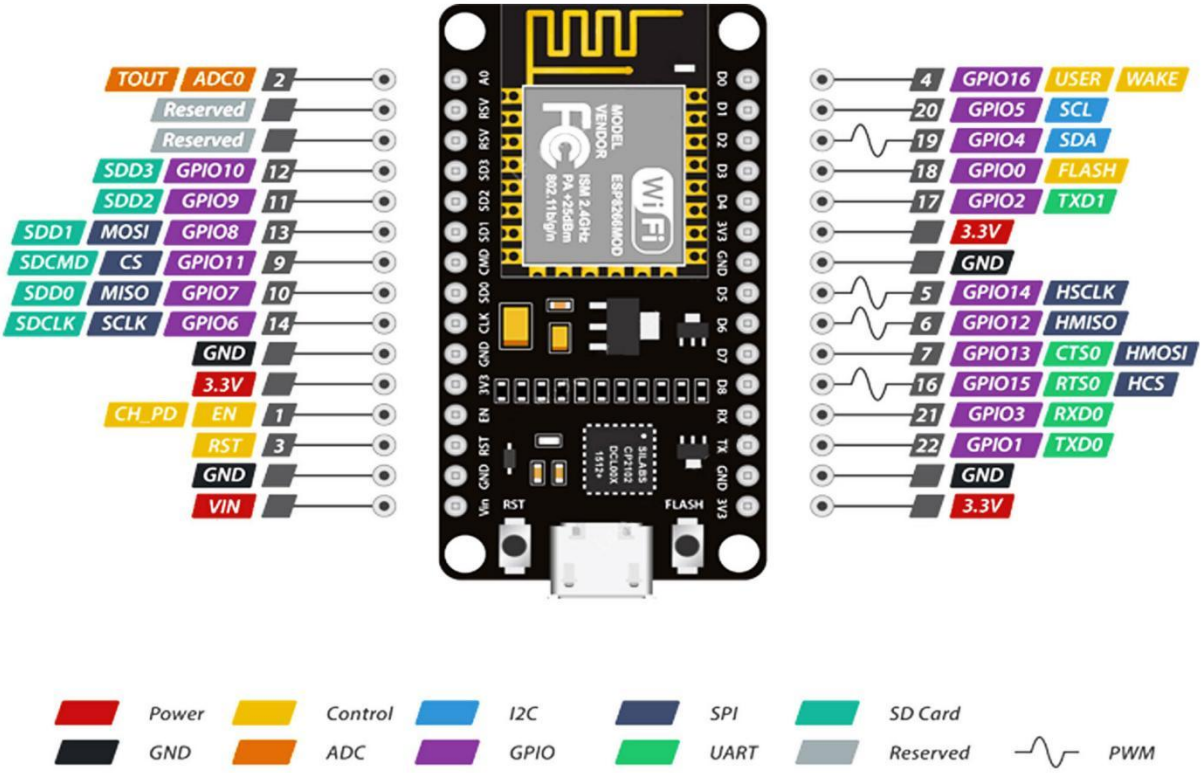


Figure 2.6. ESP8266

Chip	ESP8266EX
WiFi	2.4 GHz supports 802.11 b/g/n standard
Operating voltage	3.3V
Input voltage	5V through USB port
Number of I/O pins	11
Number of ADC pins	1 (maximum input voltage 3.3V)
Flash Memory	4MB
Communicate	Micro USB Cable

Security support	WPA/WPA2
Protocol integration	TCP/IP

Table 2.4. Specifications of ESP8266

2.3.4 JQC-3FF-S-Z

A small electromagnetic relay, the JQC-3FF-S-Z has contact ratings of up to 10A at 250V AC or 30V DC, and coil voltages ranging from 5V to 24V DC. It is extensively utilized in industrial controls, automobile electronics, and home appliances.

Figure 2.7. JQC-3FF-S-Z

Maximum load	AC 250V/10A, DC 30V/10A
Trigger current	5mA
Working voltage	5V
Module size	50 x 26 x 18.5mm
Mounting bolt holes diameter	3.1mm

Table 2.5. Specification of JQC-3FF-S-Z

2.3.5 Small Oxygen Bubbler

A small oxygen bubbler is a device that increases oxygen levels in water by producing tiny bubbles. It's commonly used in aquariums, hydroponics, and aquaculture to ensure adequate oxygen for aquatic life.

Figure 2.8. Oxygen Bubbler

Connection	USB charging port, 5V
Size	3x6cm
Power	5W

Table 2.6. Specification of oxygen bubbler

2.4 Software

2.4.1 ReactJS

Facebook developed the open-source JavaScript framework and library known as React.js. Compared to conventional JavaScript techniques, it allows for the quick building of

interactive user interfaces and online apps with less code.

For the purpose of creating dynamic and responsive user interfaces for single-page web apps, or SPAs, ReactJS is used. One remarkable feature of ReactJS is its flexible re-rendering, which updates only the necessary portions of the interface when a component's state changes. Both performance and the user experience are improved by this. Because of the abundance of development tools and the strong community support it has received, ReactJS has become one of the most sought-after technologies in the current online application development space.

2.4.2 NodeJS

Node.js is an open-source, server-side runtime environment built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript code outside of a web browser, enabling server-side scripting to create dynamic web pages and applications. Node.js is known for its event-driven, non-blocking I/O model, which makes it lightweight and efficient for handling concurrent connections and real-time applications. It has a rich ecosystem of libraries and frameworks that facilitate building scalable and fast network applications.

2.4.3 MySQL

MySQL is an open-source relational database management system (RDBMS) that utilizes Structured Query Language (SQL). It enables users to store, manage, and retrieve data efficiently. MySQL is widely recognized for its scalability, reliability, and ease of integration with web-based applications, making it a popular choice among developers for building dynamic websites and applications that require robust data management capabilities.

2.4.4 SocketIO

Socket.IO is a JavaScript library that enables real-time, bidirectional communication between web clients and servers. It allows for seamless, event-driven communication by establishing a WebSocket connection when possible, and falling back to other techniques

such as long polling when necessary. Socket.IO is commonly used in web applications for features like chat applications, real-time analytics, collaborative tools, and multiplayer gaming, where instant updates and interaction between clients and servers are crucial. Its versatility and cross-browser compatibility make it a powerful tool for building interactive and responsive web experiences.

2.4.5 Arduino

The open-source electronics platform Arduino is built on user-friendly hardware and software. It is composed of a programming environment that enables users to create code and upload it to the programmable circuit board, also known as a microcontroller. Professionals, students, and enthusiasts utilize Arduino boards extensively to build electronic gadgets, prototypes, and interactive projects.

Based on Wiring, the Arduino programming language is comparable to C/C++. Users may interact with sensors, actuators, and other linked components by writing sketches, or programs, that they upload to the Arduino board. Suitable for a wide range of projects, from basic LED experiments to intricate robotics and Internet of Things applications, Arduino boards are available in several sizes and configurations.

CHAPTER 3

METHODOLOGY

3.1. Overview

To properly answer the research topic, I have merged these technologies for the development of vital website features, data categorization, and data storage. The result is a comprehensive solution. A solid and comprehensive answer to effectively address the current research challenge is what the combination of these technologies seeks to provide:

- Front-end: ReactJS
- Back-end: Node
- Database: MySQL
- IoT: Arduino

3.2. System Design

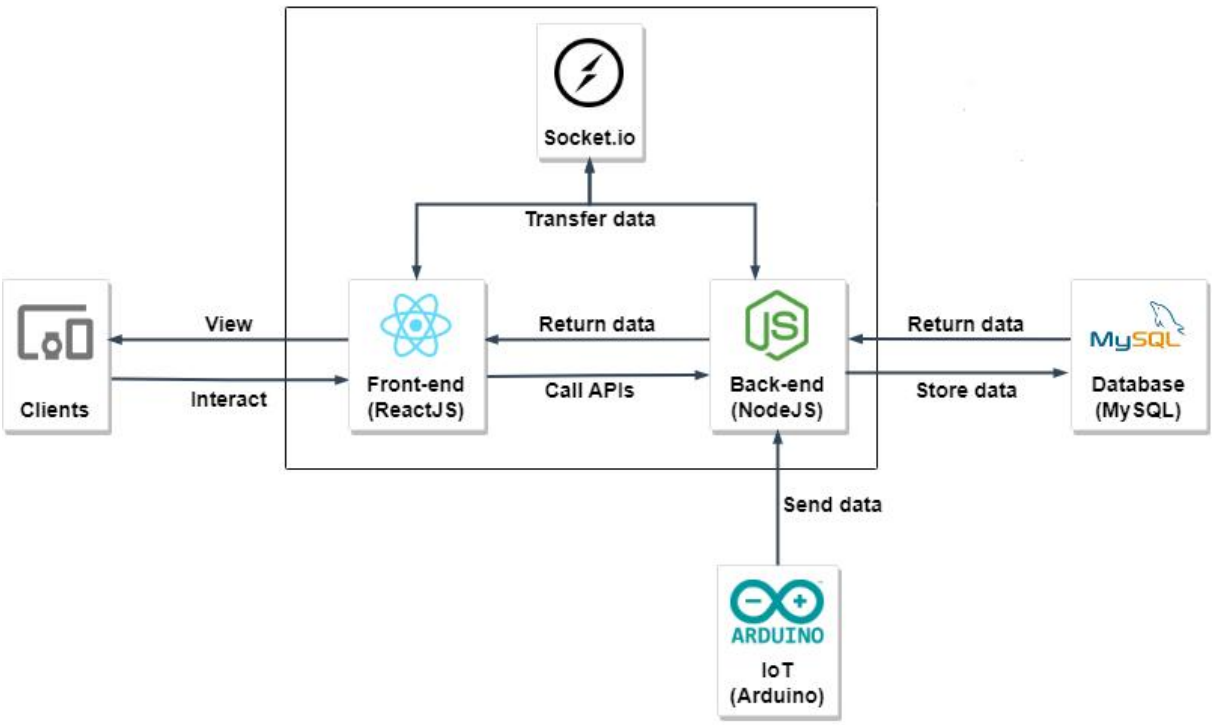


Figure 3.1. System Design

This is the project's process. All data from the IoT is processed using NodeJS and then sent to consumers via Arduino and MySQL. Data will be received by the front-end side and

sent to users for viewing and request-making. Socket will be used to transfer certain data between the front-end and back-end in real time.

3.2.1. Database design

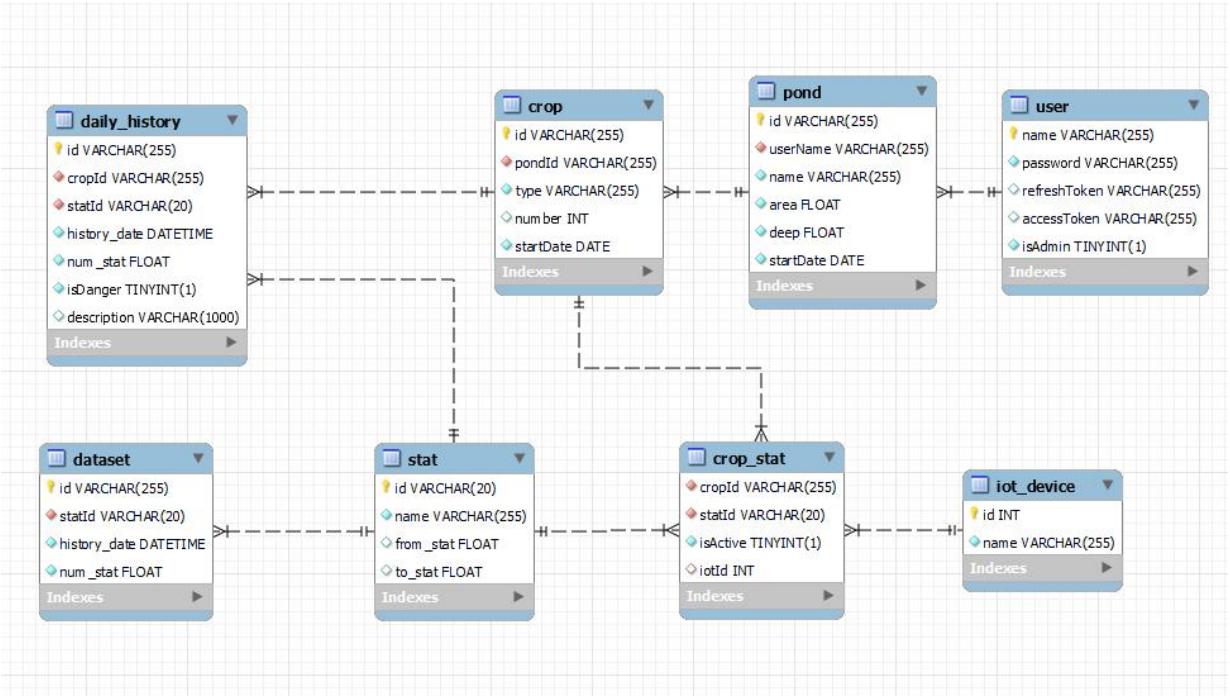


Figure 3.2. Database

An overview of database system which was used in this project. This contains several tables for the application to function properly, which will be explain in the latter part.

3.2.2. User interface design

ReactJS is a JavaScript library that was used to construct the user interface and user experience (UI/UX) for this thesis project. For this project, ReactJS has many advantages that are helpful, such as:

- Component-Based Architecture: UI components that may be reused for effective maintenance.
- Virtual DOM: Performance-enhancing rendering optimized.
- One-Way Data Flow: A coding structure that is predictable and easy to maintain.
- JSX Syntax: Integration of legible and declarative code.

- Rich Ecosystem: Community support and a large number of libraries.
- Cross-Platform Compatibility: Reusing code while developing native mobile apps.
- SEO-Friendly: Utilizing server-side rendering to improve exposure in search results

An extensive feature set has been created as part of this project to help shrimp farmers manage their ponds, crops, and data efficiently. These features come with a variety of functionality designed to improve productivity and streamline processes. The following are the main features that have been implemented:

✓ **Authorization:** Strong permission procedures are incorporated into the project to guarantee secure access and safeguard sensitive data. Before getting access to the system, users must authenticate themselves by logging in and confirming their credentials. In addition, new users may make accounts using a registration tool, which gives them access to all of the platform's features.

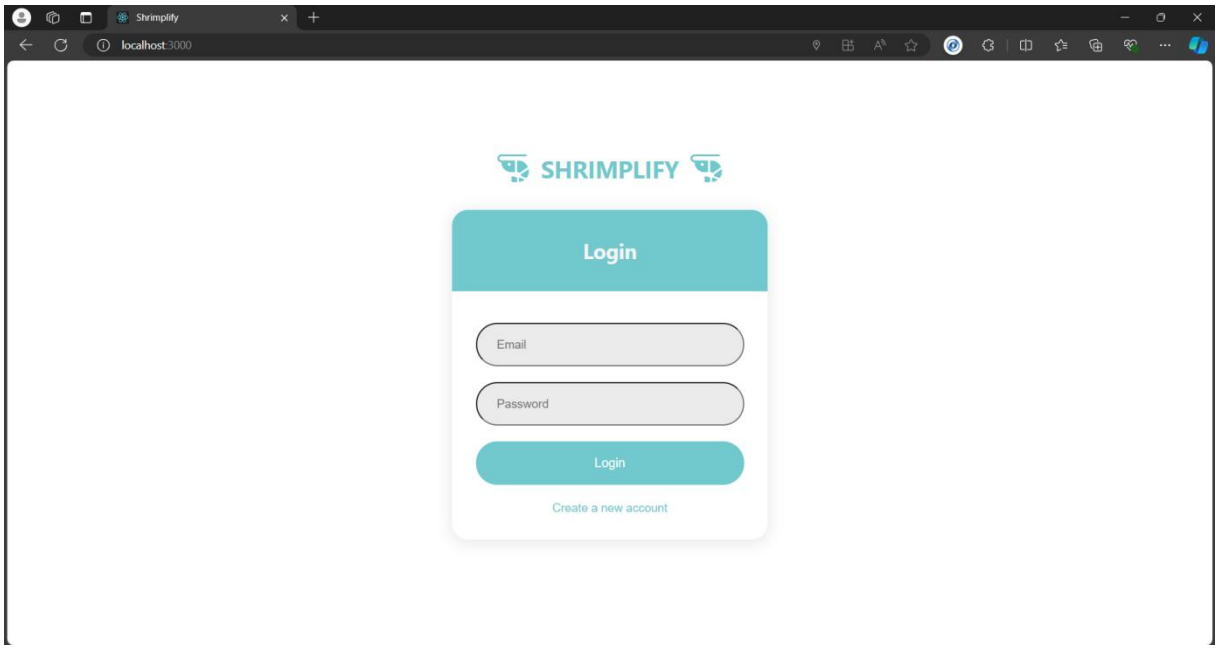


Figure 3.3. Login screen

A login screen include the application name, along with two input fields for user to fill their email and password.

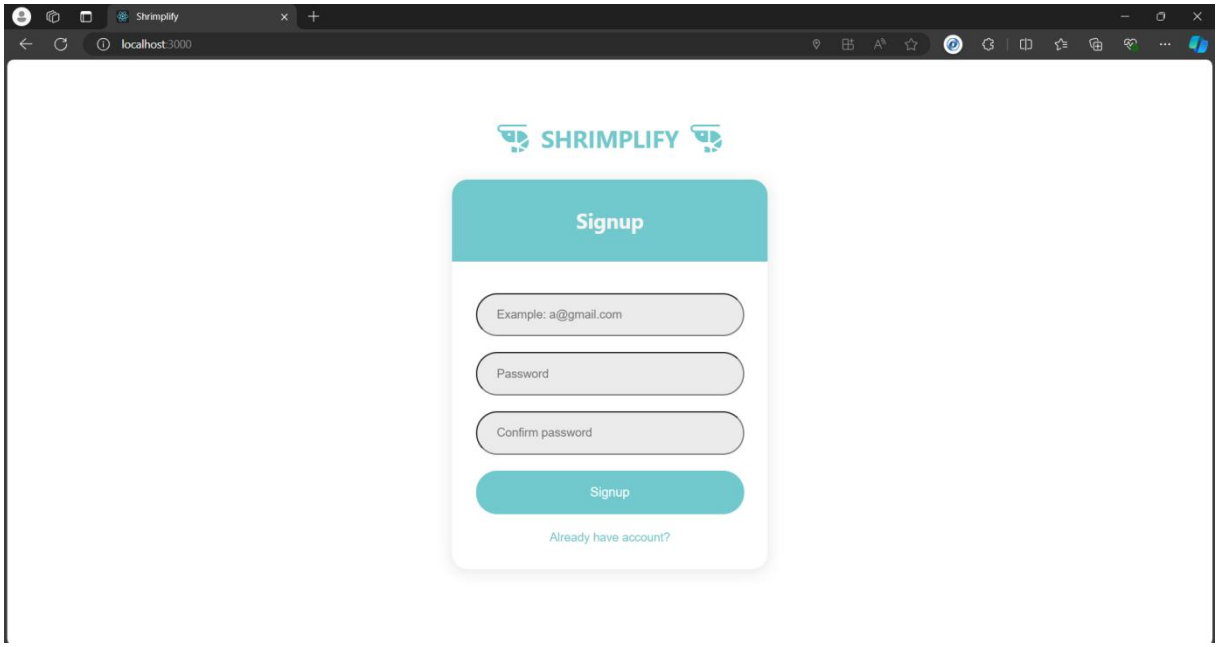


Figure 3.4. Sign up screen

A sign up screen is similar to login screen, with additional confirm password field for validation.

- ✓ **Basic Actions:** The system provides essential features like adding new ponds, removing old ones, and updating data, making it easy for farmers to manage their ponds, crops, and related information. With a user-friendly interface, these tasks can be completed without any technical difficulties. This ensures that farmers can efficiently organize and maintain their agricultural activities without getting bogged down by complex processes, allowing them to focus on improving productivity and crop management. The intuitive design helps streamline their workflow, making daily operations smoother and more effective.

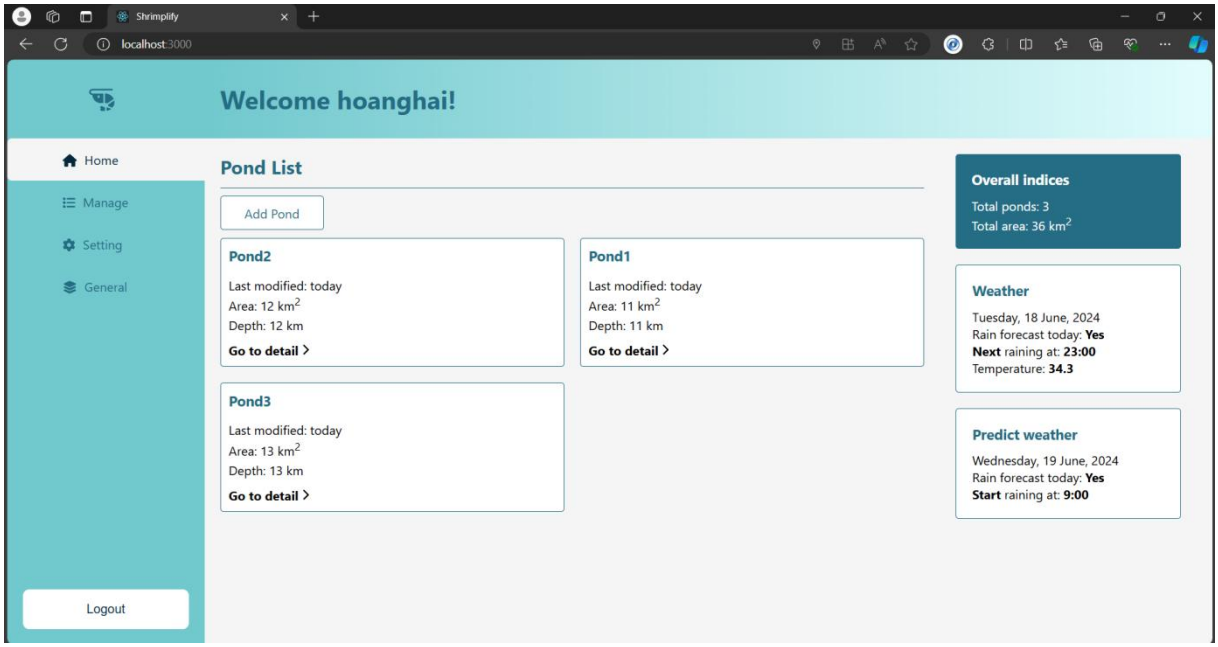


Figure 3.5. Homepage

A homepage shows some of user information such as name and list of ponds, as well as the weather prediction.

The information about the ponds that users now possess is shown on the homepage above and is unique to each account. Furthermore, there is a weather section that provides forecasts for the following day as well as the present weather. Here, users may manage their businesses by observing general stats and creating a new pond.

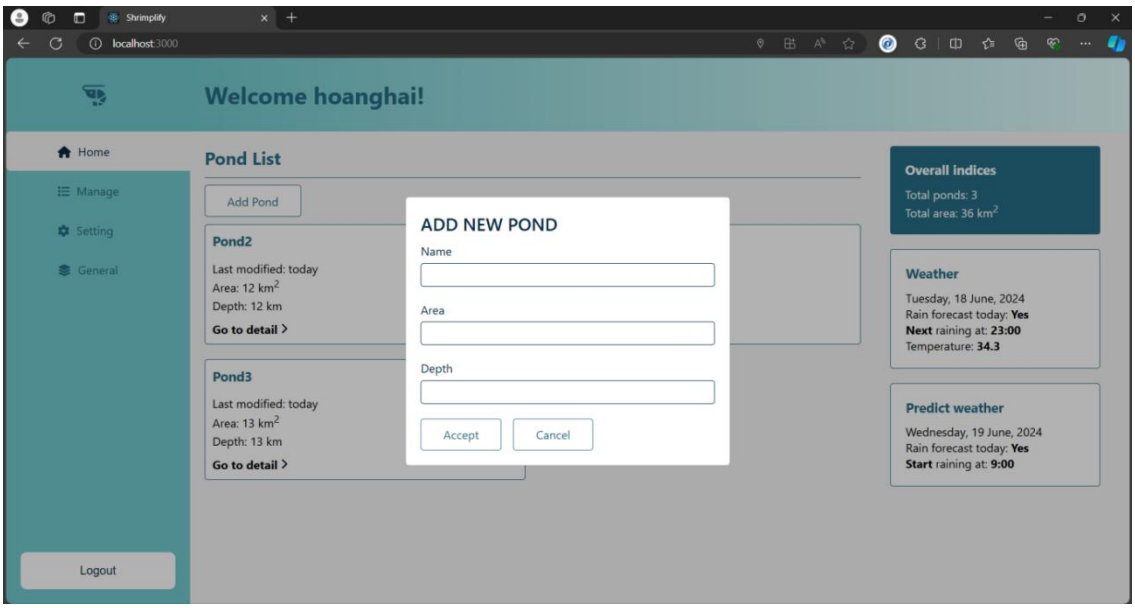


Figure 3.6. Add pond form

A form that allow user to input specific attributes to new pond

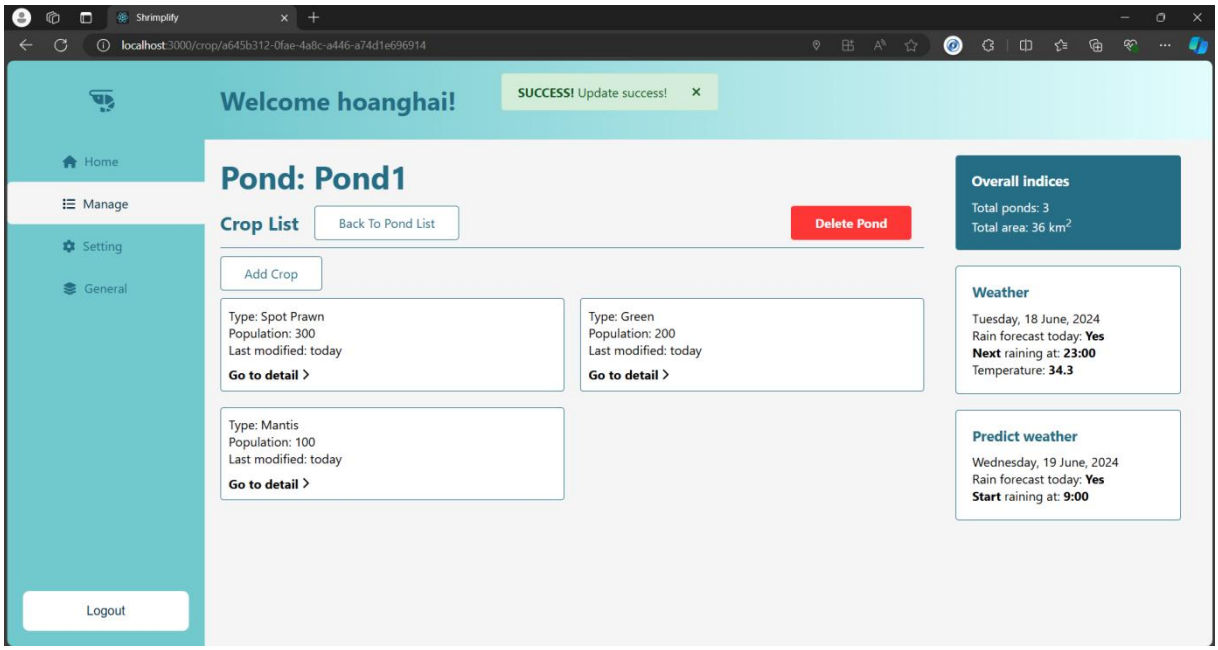


Figure 3.7. Manage screen

A screen in which shows the list of crop in one specific chosen pond.

After the user choose to examine the details of a particular pond, this page appears. A list of the species is displayed along with their quantity. User can also manage this pond population by adding more species to it. The option to delete the selected pond is also located here.

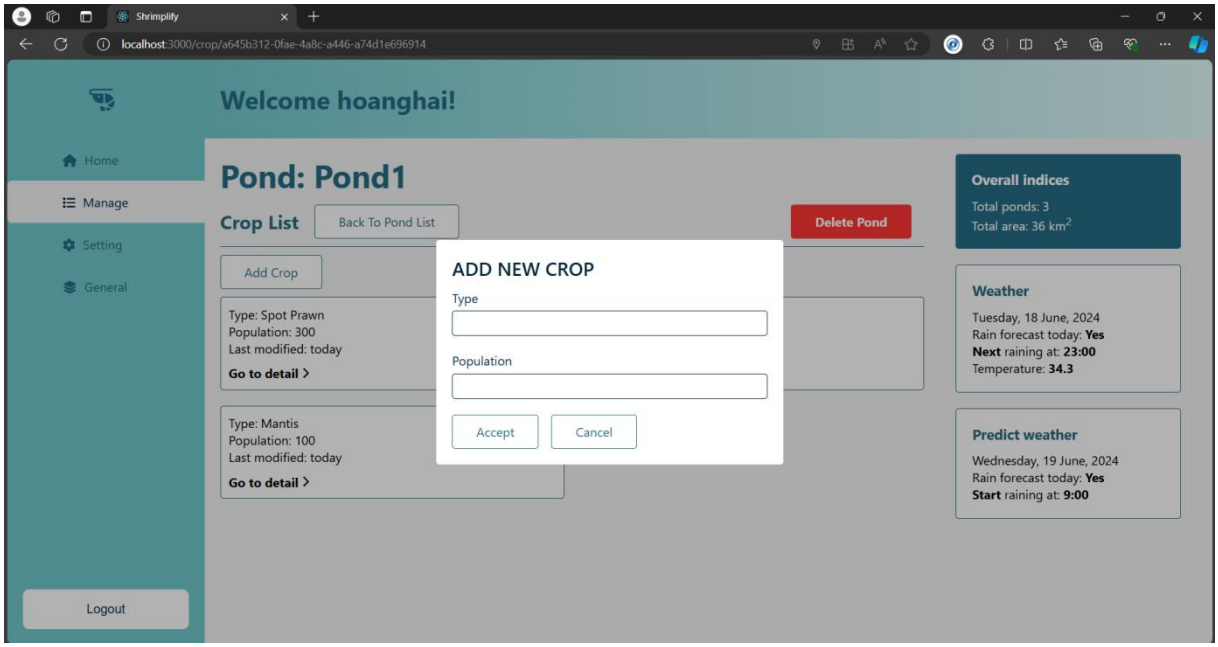


Figure 163.8. Add crop form

A form that allow user to input specific attributes to new crop.

✓ **Graphical Representation:** The system employs a graphical technique to update the selected pond statistics in real-time. This visualization allows farmers to monitor key metrics effortlessly. Each green dot on the graph represents a stat check and storage cycle; in this example, it tracks the temperature. The system automatically logs events when the temperature rises or falls below a predefined threshold. When such an occurrence is detected, the system triggers IoT devices to take appropriate actions, ensuring that the pond environment remains optimal. This real-time monitoring and automated response mechanism enhance the precision and efficiency of pond management.

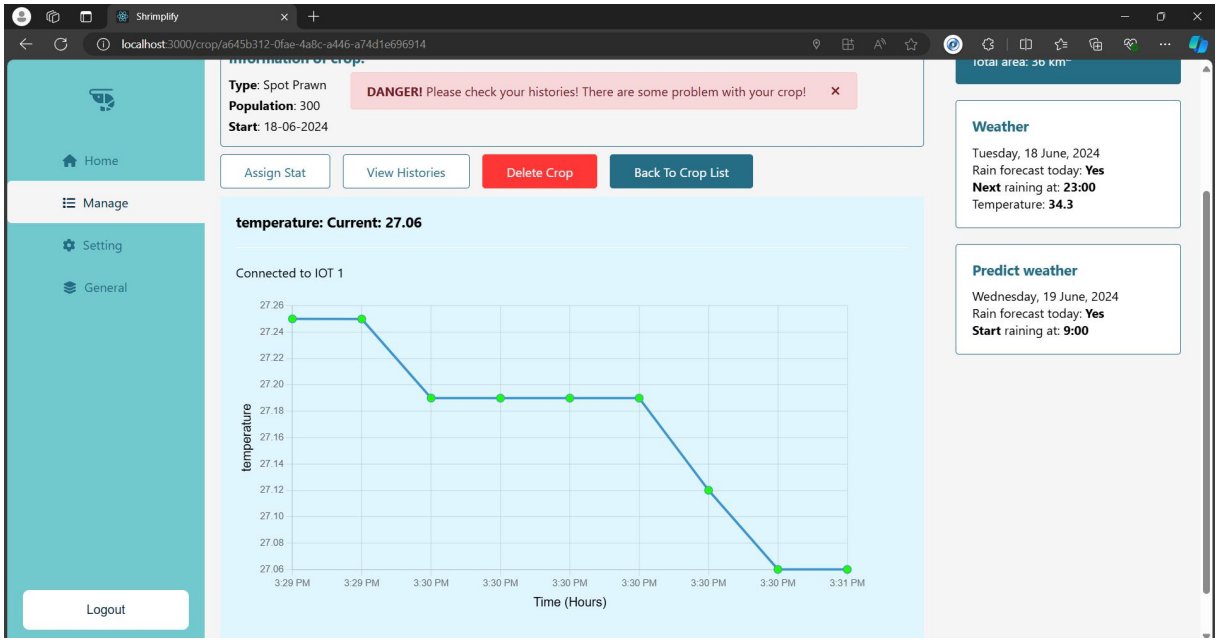


Figure 3.9. Crop stat in graph

A graph that appear when specific crop was assigned with a IoT devices.

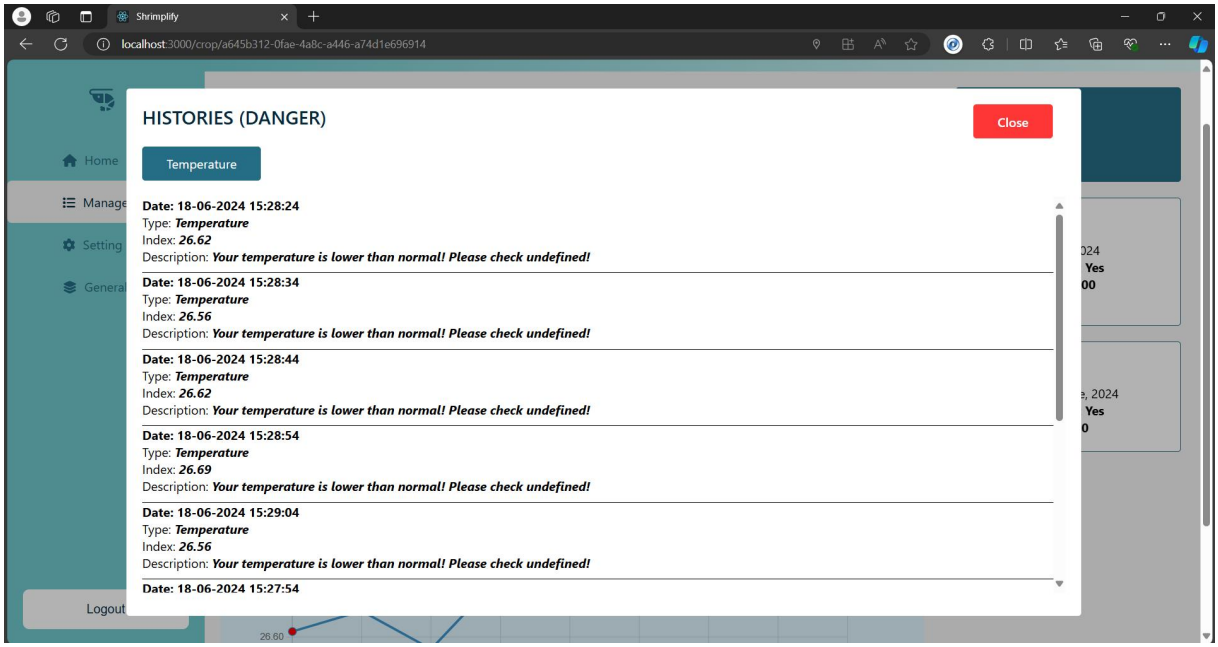


Figure 3.10. History of abnormal signs

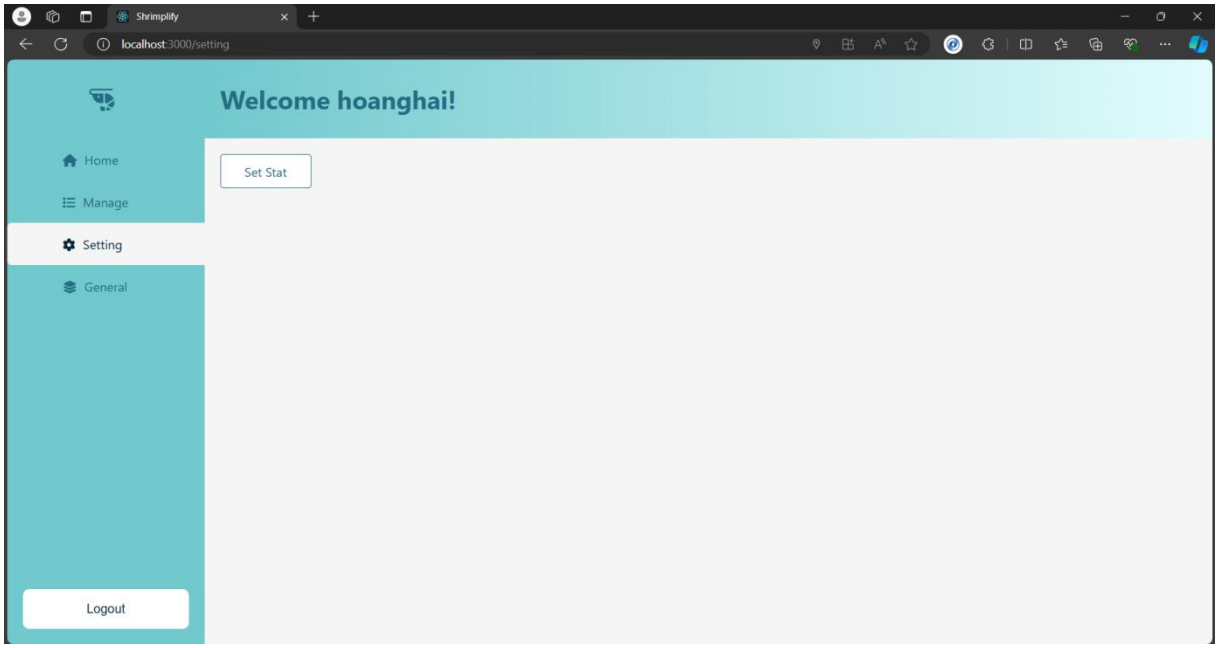


Figure 3.11. Setting screen

Setting screen is admin access only. This simply grant user the ability to set the limit range for stats without the need to dive into database system. There aren't many admin options on this website right now because the major focus of this thesis is shrimp pond statistics. This page might be the administration page, where changes and configurations can be made to the entire program.

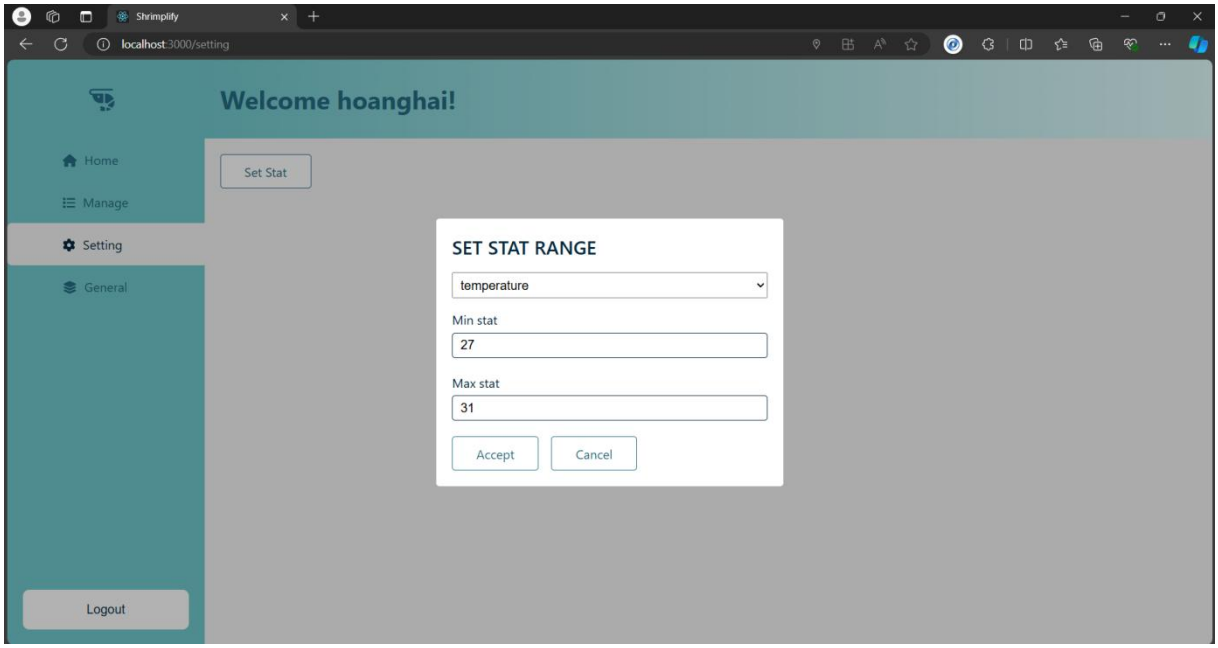


Figure 3.12. Stat setting

A form that allows admin to make change to stat range, bottom limit to top limit.

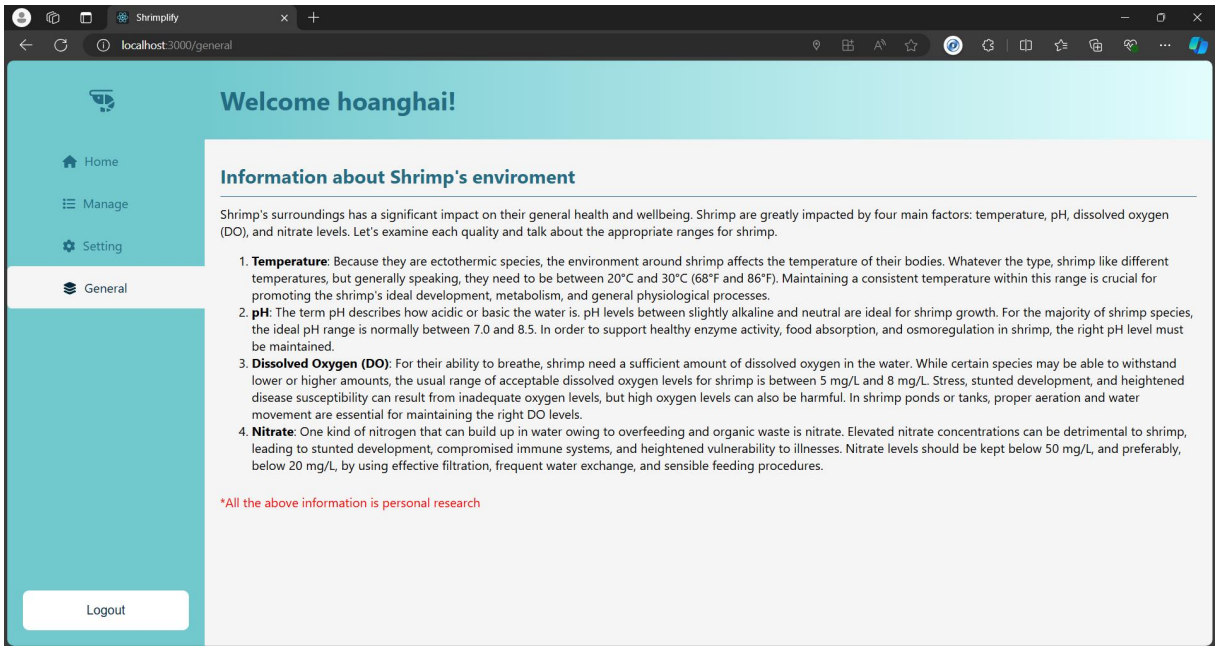


Figure 213.13. General screen

This is an information page, which include general information of some critical stats toward shrimp's environment, which is the pond quality. In the future, this page may serve as the news page with articles and research from farmers and other users, greatly enhancing community connections.

3.2.3 APIs

An application programming interface, or API, is a collection of guidelines and conventions used while creating and utilizing software applications. It outlines the protocols and information types that apps may utilize to talk to one another. In this article, these are all the applied APIs divine into categories:

Method	**API url**	**Payload**	**Response**
POST	/auth	email: string password: string	name: string access_token: string refresh_token: string is_admin: boolean
POST	/auth/register	email: string password: string	<NONE>
POST	/auth/refreshToken	refresh_token: string	access_token: string

Table 3.1. Authentication APIs

Method	**API url**	**Payload**	**Response**
GET (list)	/pond/:username	<NONE>	[{ id: string name: string userName: string area: number deep: number startDate: Date }, …]

GET (by id)	/pond/p/:id	<NONE>	id: string name: string userName: string area: number deep: number startDate: Date
POST	/pond	name: string userName: string area: number deep: number startDate: Date	<NONE>
PUT	/pond	id: string name: string userName: string area: number deep: number startDate: Date	<NONE>
DELETE	/pond/:id	<NONE>	<NONE>

Table 3.2. Pond Management APIs

Method	**API url**	**Payload**	**Response**
GET (list by pondId)	/crop/pond/:id	<NONE>	[{ id: string pondId: string type: string

			number: number startDate: Date }, …]
GET (by id)	/crop /:id	<NONE>	id: string pondId: string type: string number: number startDate: Date
POST	/crop	pondId: string type: string number: number	<NONE>
PUT	/crop	id: string pondId: string type: string number: number	<NONE>
DELETE	/crop/:id	<NONE>	<NONE>
GET (tracking crops)	/crop/tracking?username&statId	<NONE>	[{ id: string pondId: string type: string number: number startDate: Date }, …]
GET (stats	/crop/stat/:id	<NONE>	cropId: string statId: string

tracking by crop)			isActive: boolean
POST (set stats for crop)	/crop/stat/:id	lstStats: string lstActive: string	<NONE>
PUT (update stats of crop)	/crop/stat/:id	statId: string isActive: boolean	<NONE>
GET (all histories)	/crop/history/all/:id	<NONE>	[{ id: string, cropId: string, pondId: string, history_date: Date, num_stat: number, isDanger: Boolean, description: string }, …]
GET (history by id)	/crop/history /:id	<NONE>	id: string, cropId: string, pondId: string, history_date: Date, num_stat: number, isDanger: Boolean, description: string

POST	/crop/history	cropId: string, pondId: string, num_stat: number, isDanger: Boolean, description: string	<NONE>
DELETE	/crop/history /:id	<NONE>	<NONE>

Table 3.3. Crop APIs

Method	**API url**	**Payload**	**Response**
PUT	/stat	id: number name: string from_stat: number to_stat: number username: string	<NONE>
GET	/stat/:id	<NONE>	id: number name: string from_stat: number to_stat: number username: string
GET (all)	/stat		[{ id: number name: string from_stat: number to_stat: number username: string }, …]

Table 3.4. Stat APIs

Method	API url	Payload	Response

GET	https://api.open-meteo.com/v1/forecast	Query: {
		latitude: number
		longitude: number
		current_weather: boolean
		hourly: wethercode
		}
			longtitude: number,
			latitude: number,
			current_weather: {
			temperature: number,
			windspeed: number,
			weathercode: number,
			…
			},
			Weathercode: […n umber],
			…

Table 3.5. Weather APIs

3.3. Security

3.3.1. Encrypting

Encryption [13] is the act of protecting data before it is sent or stored by transforming it into an unintelligible format known as ciphertext using algorithms and keys. Encryption is used to protect data security and privacy by making it impossible for unauthorized parties to read, comprehend, or alter the data without the proper decryption key. An encryption algorithm and a key, which is a string of values used to carry out the encryption and decryption process, are usually employed in encryption. There are two main types of encryptions:

- **Symmetric Encryption** uses the same key for both encryption and decryption of data.

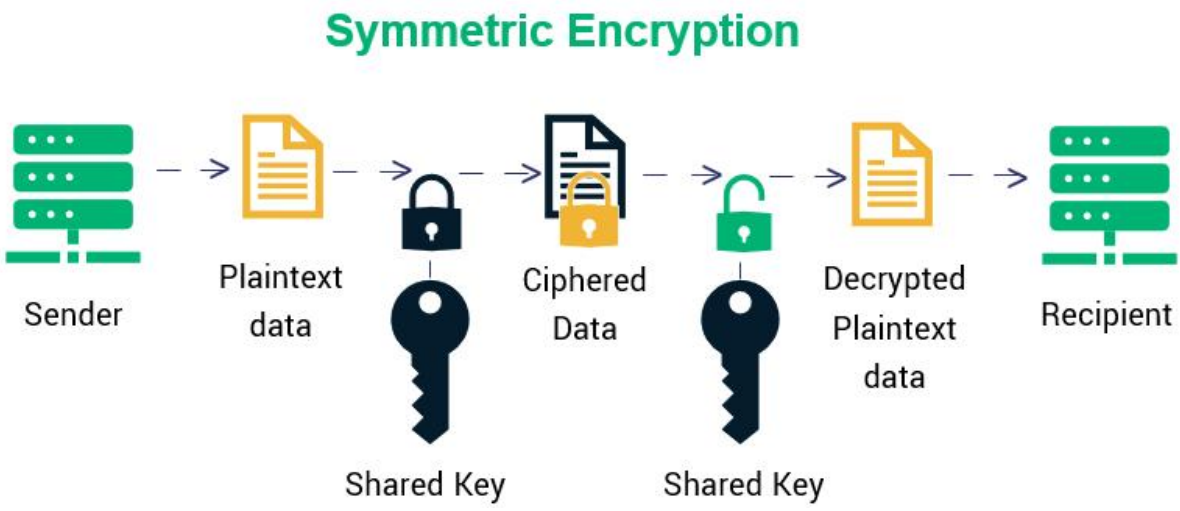


Figure 3.14. Symmetric Encryption

- **Asymmetric encryption** uses a pair of keys, consisting of a public key and a private key. The data is encrypted using the public key and can only be decrypted with the corresponding private key.

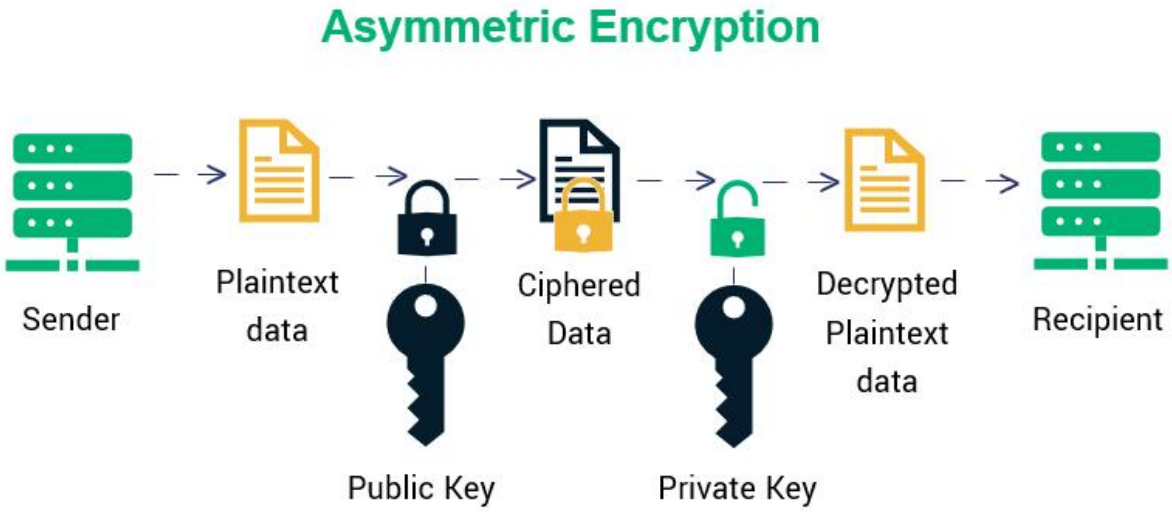


Figure 3.15. Asymmetric Encryption

3.3.2. Hashing

The process of hashing [14] involves taking a plaintext input and turning it into a fixed-length string of characters or integers known as a hash value (or hash code). A specific hash algorithm that converts the input data into a unique or nearly unique result is used for

this hashing process. The hashing function creates a string by combining many complicated parts.

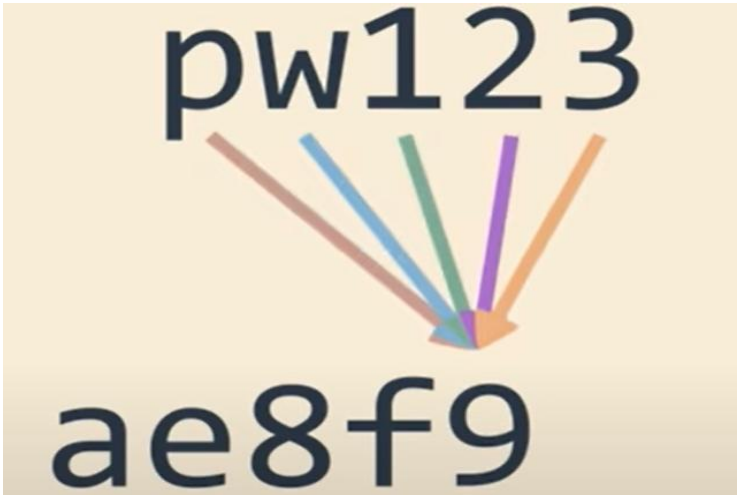


Figure 3.16. Hashing

Hash functions typically have the following properties:

- **Fixed length:** The hash value has a fixed length, regardless of the size of the input data.

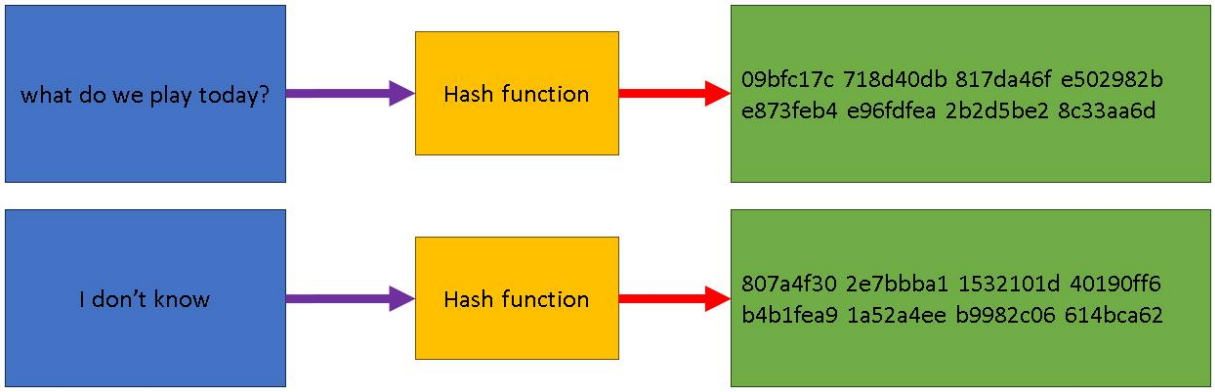


Figure 3.17. Fixed length hashing

- **Non-reversible:** The hash value cannot be used to retrieve the original input data. Consider it as a way to create a central color by combining three different hues. Nevertheless, it would be challenging to identify the precise colors that were combined to create the color in the middle if you just had that one color.

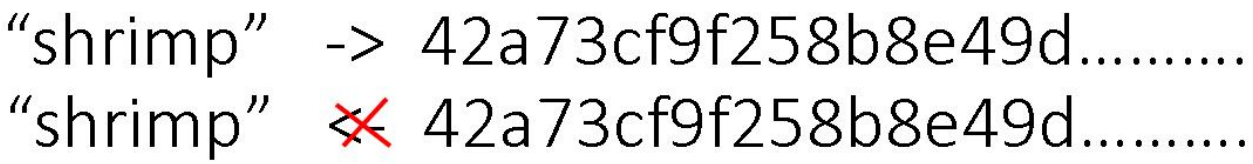


Figure 3.18. One way hashing

- **Independence:** The hash value that is produced will vary significantly in response to even slight changes in the input data. This is the outcome of the hash function's mixing.

Figure 3.19. Word-by-word hashing

3.3.3. Password security

Sensitive data, particularly user passwords, must be securely stored when developing a web application that needs user login capabilities.

3.3.3.1. Plaintext Storage

Plaintext storage is the simplest type of storage without using any related techniques, just storing the exact password that the user has registered to store in the database.

user_id	account	password
1	hello@tsa.com	123456
2	lovely@gmail.com	qwerty
3	ok@yahoo.com	Shrimp@thesis
4	hahaha@okia.com	Abc123456
5	learning@student.com	A1b2c3d4e5@xyz

Figure 3.20. Plaintext storage

Passwords stored insecurely carry a number of problems, particularly if the storage system is breached. Hackers can quickly obtain and decipher passwords that are kept in plaintext, potentially resulting in security lapses and illegal access to user accounts.

3.3.3.2. Encrypting Storage

Encrypted data storage contributes to storage security in part because, even in the unlikely event that hackers manage to obtain it, deciphering the encrypted passwords becomes exceedingly challenging for them. When users log in, the system typically stores and decrypts passwords using symmetric encryption.

user_id	account	password
1	hello@tsa.com	<encrypted>
2	lovely@gmail.com	<encrypted>
3	ok@yahoo.com	<encrypted>
4	hahaha@okia.com	<encrypted>
5	learning@student.com	<encrypted>

Figure 3.21. Encrypted storage

On the other hand, it is not hard for hackers to obtain the decryption key if they manage to infiltrate the database. Usually, a configuration file or the server contains this key. The plain text version of your passwords will become visible once they get the decryption key. Furthermore, some employees of such servers frequently have special access to these decryption keys, allowing them to decipher user passwords for illegal access.

3.3.3.3. Hasing Storage

By removing the need to save the original passwords, hashing improves the security of password storage. A hashing function is used to process a user's password upon login, generating a hashed password that is compared to the hashed password that is saved in the database. Since there is no decryption key required, this method represents a substantial security improvement.

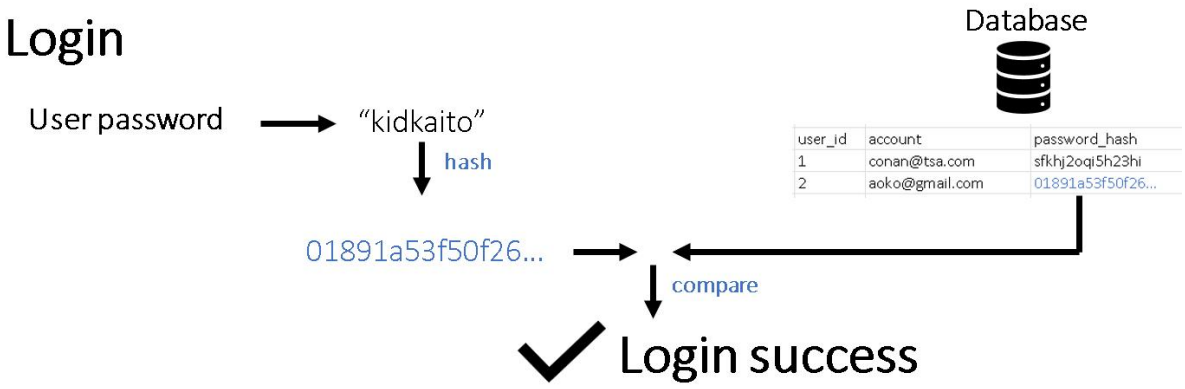


Figure 3.22. Hashing workflow

Hashing has its restrictions, though. If users set the same password, the database will store the same hash, which is not recommended.

user_id	account	password_hash
1	conan@tsa.com	8d969eef6ecad3c29a3...
2	aoko@gmail.com	8d969eef6ecad3c29a3....

Figure 3.23. Storing alike password

The hacker can still use the hash function to construct a password and compare it with the stolen database even if the hashed password cannot be changed back to its original form. Hackers can quickly figure out passwords that people create if they are overly easy. We call this kind of assault a dictionary attack.

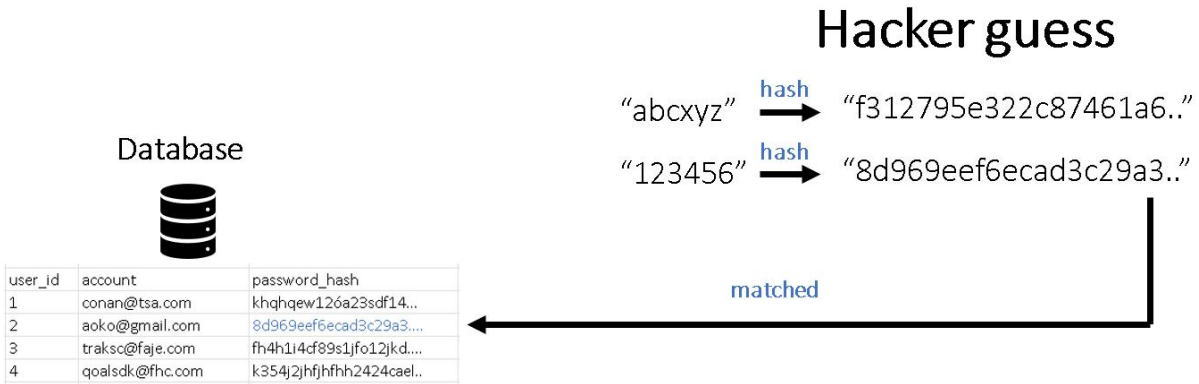


Figure 3.24. Dictionary attack

Hackers can also pre-create a sizable database of hashed passwords using passwords that are frequently used, and then compare that database with the stolen information. Rainbow tables is the name given to these database.

3.3.3.4. Hashing and Salting Storage

To improve security during the password hashing process, salting is a technique utilized. Salting is the act of adding a random string, or "salt," to the input data before running the hashing algorithm when utilizing a hash function to save passwords. Salt is a fixed-length random string that is often made up of random letters or bytes. Along with the password's hash value, each password that is kept will have a unique salt. The salt will be used to combine the password supplied when the user signs in, followed by the hashing procedure. Depending on the salt, each password has a distinct hash value as a consequence.

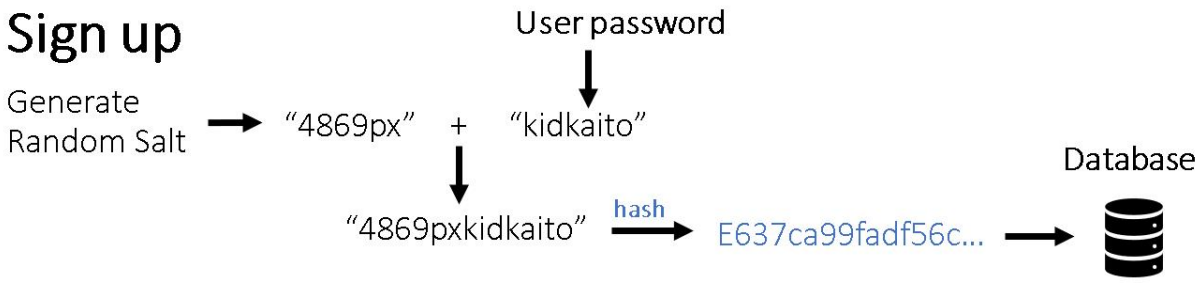


Figure 3.25. Salting + Hashing workflow

user_id	account	password_hash	salt
1	conan@tsa.com	3f53a8af1919f96..	okie2
2	aoko@gmail.com	e637ca99fadf56c...	4869px

Figure 3.26. Salting + Hashing storage

Because it has no salt, this can reduce assaults using rainbow tables. It is still unable to withstand dictionary attacks, though. Hackers can run predictions on sophisticated hardware, capable of hashing over 10 billion hashes in a second.

3.3.4. JWT (JSON Web Token)

An open standard called JWT (JSON Web Token) [15] offers a defined technique for safely and dependably sending data between parties in the form of JSON objects. Authentication and authorization in online services and apps are among its main use cases. The three primary components of a JWT are the Header, Payload, and Signature. Information about a JWT's type and the cryptographic algorithm—such as RSA or HMAC—used is contained in the header. The actual data to be delivered, such as user details, access rights, and any other optional data, is stored in the payload. Encoding is commonly used in JWTs to convert data in both the header and payload portions. If users have access to the JWT, they may easily decode the encrypted data to obtain the original information. It is thus advised to refrain from putting sensitive information in these parts. The data is not meant to be encrypted or secure by the JWT encoding method, such as Base64Url encoding. Its primary goal is to guarantee the token's efficient and reliable transfer. It makes it possible to express binary data in a way that makes it transferable between various systems with ease. Last but not least, the Signature is a string created by utilizing a secret key specific to the token to sign the Header and Payload. Verifying the integrity and validity of the token is the goal of the signature.

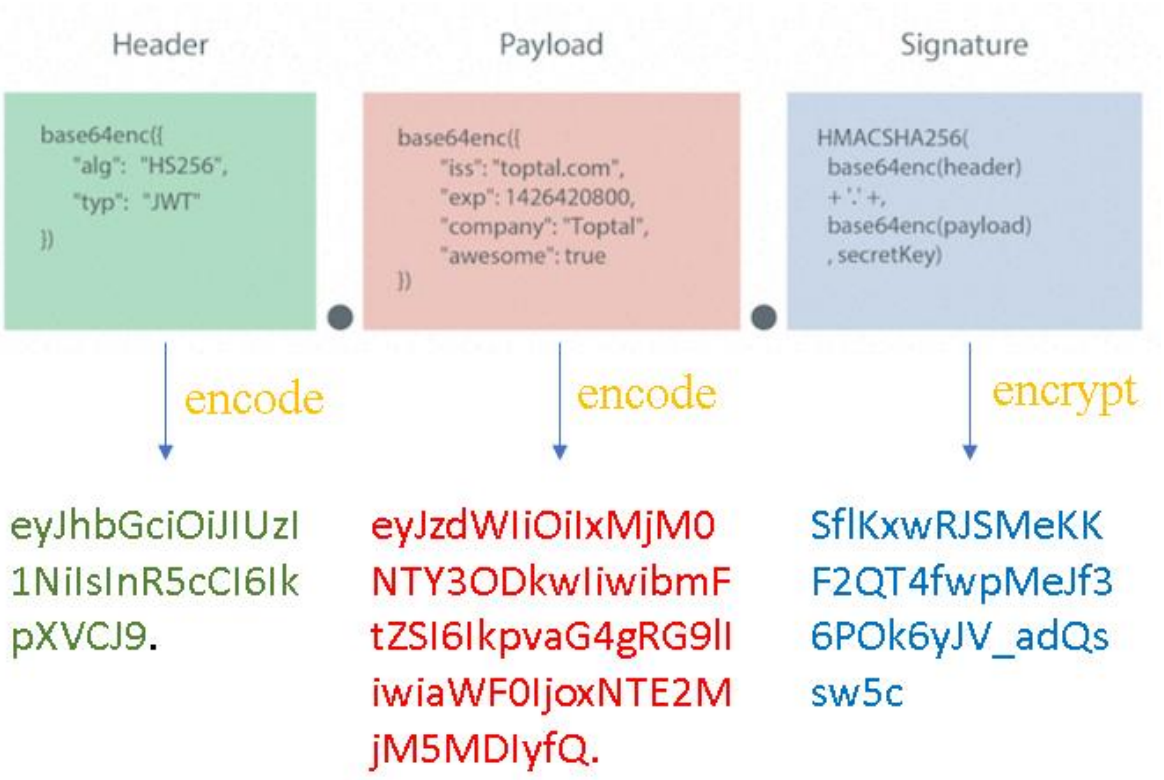


Figure 3.27. Structure of JWT

The following stages are usually included in the JWT process. With their login information (password, email address, username, and so on), users log in. After confirming the login details, the server creates a JWT (JSON Web Token), which is then returned to the user. In further queries to the server, the user adds the JWT to the Authorization header. When the server receives a request, it checks the signature and decodes the payload to confirm the JWT's integrity and validity. The payload contains information that the server uses to authenticate and authorize the user, allowing them to access the resources or services they have requested.

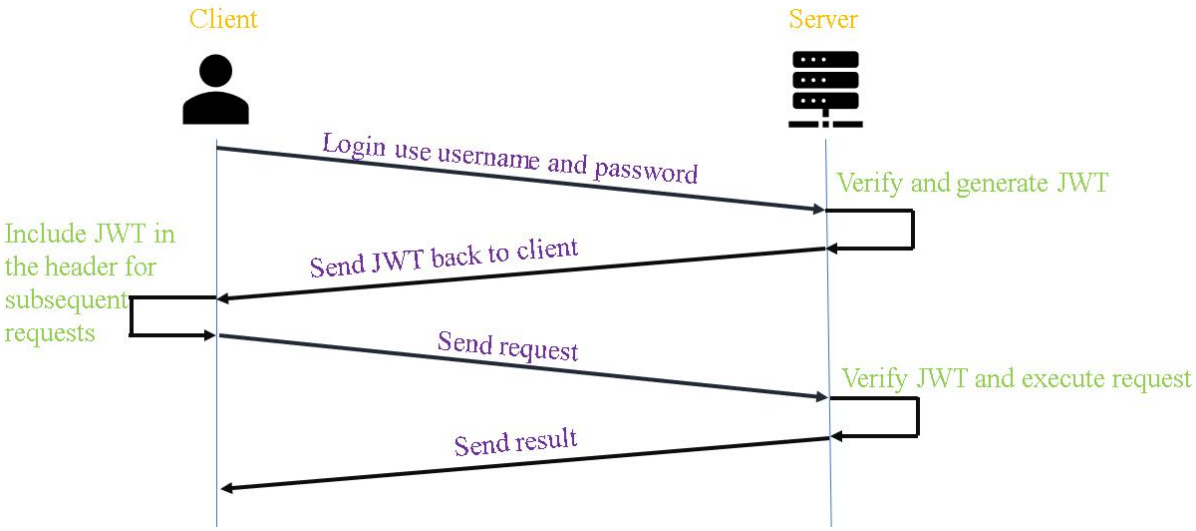


Figure 3.28. JWT workflow

Sending requests with JWT tokens back and forth over the internet on a regular basis, however, increases the likelihood of their being stolen. As a result, in order to guarantee user safety, this token must be changed often. However, there is also the issue of users having to log in again in order to receive a new JWT token, which will be inconvenient during the process. This regular re-authentication can disrupt the user experience and lead to frustration, particularly if users are frequently required to re-enter their credentials. Additionally, the increased frequency of login requests may create additional load on the authentication server, potentially leading to performance issues. Balancing the need for security with the need for a smooth and uninterrupted user experience is a critical challenge in designing systems that rely on JWT tokens for authentication. Effective solutions must carefully consider both security risks and user convenience to ensure that neither is unduly compromised.

3.3.5. Access and refresh token

We will now utilize two tokens in place of one to have complete power to send requests to the server. While both access and refresh tokens are JWT tokens, access tokens have a far shorter lifespan than refresh tokens. Every time an access token expires or loses its validity, a refresh token will be delivered in order to renew the access token. To issue new tokens, the user will have to log in again if the refresh token has expired. Unlike access tokens, which must be given with every request, refresh tokens are only transmitted when the access token expires, limiting the chance of theft. It will be less likely for stolen access tokens to be used maliciously if their lifetime is reduced to the point where they must be continuously renewed [16].

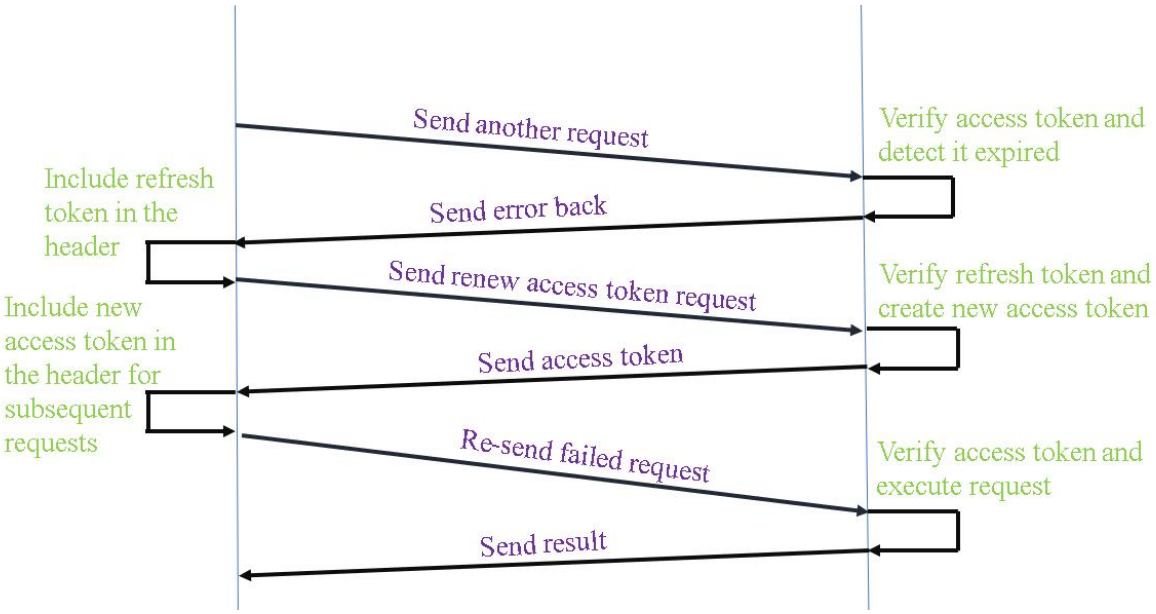


Figure 3.29. Access and refresh token workflow

CHAPTER 4

IMPLEMENT AND RESULTS

4.1. Implement

4.1.1. Hardware

4.1.1.1. Arduino Uno

This device will be the center to every other device to connect, making a system.

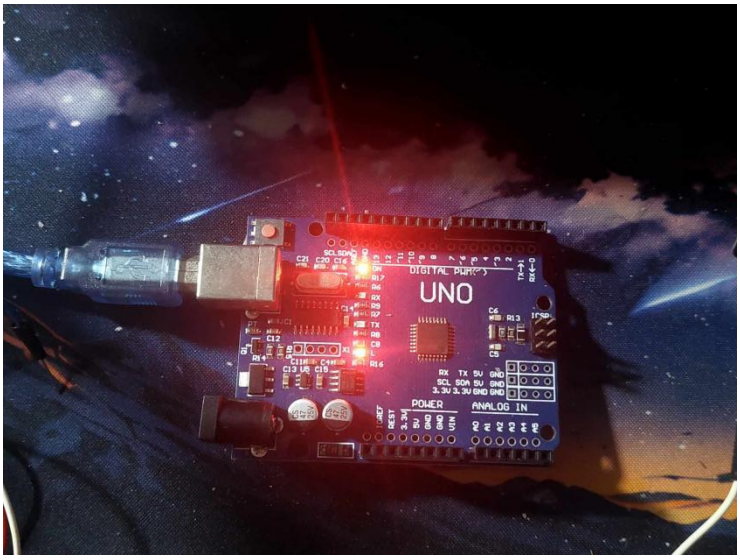


Figure 4.1. Arduino Uno

4.1.1.2. DS18B20

MAXIM produces a cutting-edge digital temperature sensor called the DS18B20. It has an amazing 12 bits of resolution. Although this temperature sensor may operate at temperatures as high as 125 degrees Celsius, it is recommended that it be kept below 100 degrees because the cord is coated with PVC.

Figure 4.2. DS18B20

To read and access the temperature of DS18B20, there are two libraries needed.

ds18b20.ino

```
1   #include <DallasTemperature.h>
2   #include <OneWire.h>
3
```

Figure 4.3. DS18B20 libraries

After including necessary libraries, the remaining code is rather straightforward. There is a function to request DS18B20 to commit data, then we set some needed condition to received it properly with the delay time of 5000 millisecond (which equal to 5 seconds)

ds18b20 | Arduino IDE 2.3.2

File Edit Sketch Tools Help

Arduino Uno

ds18b20.ino

```
12
13  void setup() {
14    // put your setup code here, to run once:
15    Serial.begin(9600);
16    pinMode(RELAY, OUTPUT);
17  }
18
19  void loop() {
20    // put your main code here, to run repeatedly:
21    getTemperature();
22    Serial.println(tempC);
23
24    if (DS18B20.getTempCByIndex(0) < upper_limit){
25      digitalWrite(RELAY, HIGH);
26    } else{
27      digitalWrite(RELAY, LOW);
28    }
29    delay(5000);
30
31  }
32
33  void getTemperature() {
34    do {
35      DS18B20.requestTemperatures();
36      tempC = DS18B20.getTempCByIndex(0);
37      if (tempC == (-127)) {
38        delay(100);
39      }
40    } while (tempC == (-127.0));
41  }
42
```

Figure 4.4. DS18B20 code

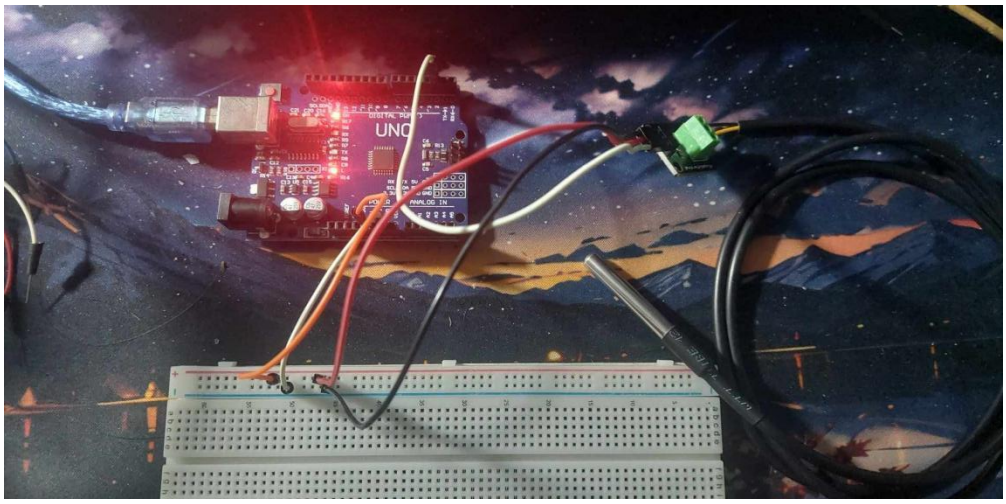


Figure 4.5 Arduino Uno + DS18B20

4.1.1.3. ESP8266 (ESP8266 NodeMCU Lua CP2102)

Made by Espressif Systems, the ESP8266 is a low-cost Wi-Fi microprocessor that has complete TCP/IP stack and microcontroller capabilities. Because to its adaptability, simplicity of use, and Wi-Fi network connectivity, it is well-liked by makers and developers.

Figure 4.6. ESP8266

The following libraries are required for successful operation:

- ✓ ESP8226WiFi, ESP8266WiFiMulti: libraries help us to connect to Wi-Fi.
- ✓ ESP8266HTTPClient: this library facilitates the management of API calls by utilizing URLs on the server-side of the back-end system.
- ✓ SocketIOclient_Generic: This library carries out communication tasks between Arduino and the back-end server.
- ✓ ArduinoJson: create a json object for call API.

esp8622.ino

```
1  #include <ESP8266WiFi.h>
2  #include <ESP8266WiFiMulti.h>
3  #include <ESP8266HTTPClient.h>
4
5  #include <SocketIOclient_Generic.h>
6  #include <ArduinoJson.h>
```

Figure 4.7. ESP8266 libraries

Establish the required parameters:

```
const char* init_ssid = "Dinh Tu";
const char* init_password = "07071977";

IPAddress serverIP(192, 168, 2, 15);
// String serverIP = "192.168.2.15";
uint16_t serverPort = 7001;

const char* GET = "GET";
const char* POST = "POST";
const char* PUT = "GET";
const char* DELETE = "DELETE";

//SOCKET: EVENT NAME
const char* CHANGE_WIFI = "CHANGE_WIFI";
const char* GET_CROP = "GET_CROP";
const char* START_SOCKET = "START_SOCKET_FROM_BACKEND";
```

Figure 4.8. Parameters

- serverIP, serverPort: Open a socket connection.
- Init_ssid, init_password: initial, configurable value for a wifi connection to the ESP8622.
- GET, POST, PUT, DELETE: declare all HTTP methods to call API.
- CHANGE_WIFI, GET_CROP, and START_SOCKET: identifies the socket event that the server will receive or transmit.

Then some variables:

```
const String baseURL = "http://192.168.2.15:7001/";
const String apiSendData = "data?statId=1";
const String checkIoTConnected = "data/check";
const String createIoTDevice = "data/create";
const String getCropByIot = "data/crop";

const int delayTime = 10000;
unsigned long messageTimestamp = 0;

bool isRunningSocket = false;

int iot_id = 1;
String crop_id = "";
```

Figure 4.9. Variables

Define functions for connecting to WiFi. Function addAP() will add an access point (AP) and connect WiFi using the supplied password and SSID. Before doing the next actions, it will keep trying the connection until it receives the code WL_CONNECTED and is successful.

```
void connectToWifi(const char* ssid, const char* password) {
  if (WiFi.getMode() & WIFI_AP) {
    WiFi.softAPdisconnect(true);
  }

  WiFiMulti.addAP(ssid, password);

  //WiFi.disconnect();
  while (WiFiMulti.run() != WL_CONNECTED) {
    Serial.println("Connecting to WiFi...");
    delay(100);
  }

  Serial.println("WiFi connected");
  Serial.println("IP address: " + WiFi.localIP().toString());
  Serial.println("Server started");
}
```

Figure 4.10. ConnectoWifi Function

Create a function to access the API, declare wifiClient, and use http.begin() to call the server's API using the required HTTP method and URL. Return the response's value at the end:

```
String callApi(String url, const char* method, String data) {
  String response = "";
  WiFiClient wifiClient;
  HTTPClient http;
  http.begin(wifiClient, (url).c_str());
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.sendRequest(method, data);
  if (httpResponseCode > 0) {
    response = http.getString();
    Serial.println("API Response: " + response);
  } else {
    Serial.print("Error in sending data to API check IoT connected. Error code: ");
    Serial.println(httpResponseCode);
  }
  return response;
}
```

Figure 4.11. callApi

Define a function to send data to a socket server. This feature requires three values to be passed to the socket server: the object's name, keys, and values. It then uses methods to filter the data and use serializeJson() to transform it into a JSON array.

```
void sendDataSocket(const char* eventName, const std::vector<String>& keys, const std::vector<String>& values) {
  if (eventName && keys.size() == values.size()) {
    DynamicJsonDocument doc(1024);
    JsonArray array = doc.to<JsonArray>();

    array.add(eventName);
    for (int i = 0; i < keys.size(); i++) {
      JsonObject param = array.createNestedObject();
      param[keys[i]] = values[i];
      delay(10);
    }

    String output;
    serializeJson(doc, output);
    // Send event
    socketIO.sendEVENT(output);
  }
}
```

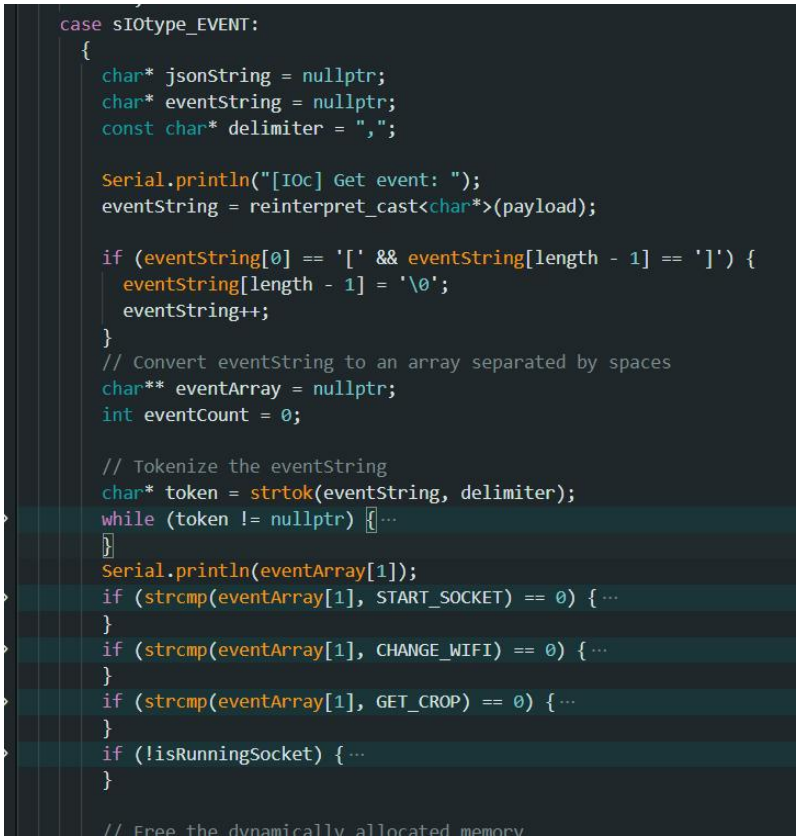
Figure 4.12. sendDataSocket

Define middleware functions for a specific socket event:

```
void socketIOEvent(const socketIOmessageType_t& type, uint8_t* payload, const size_t& length) {
  switch (type) {
    case sIOtype_DISCONNECT: ···
    case sIOtype_CONNECT:
      Serial.print("[IOc] Connected to url: ");
      Serial.println(length);
      Serial.println((char*)payload);
      // join default namespace (no auto join in Socket.IO V3)
      socketIO.send(sIOtype_CONNECT, "/");
      break;
    case sIOtype_EVENT: ···
    case sIOtype_ACK: ···
    case sIOtype_ERROR: ···
    case sIOtype_BINARY_EVENT: ···
    case sIOtype_BINARY_ACK: ···
    case sIOtype_PING:
      Serial.println("[IOc] Get PING");
      break;
    case sIOtype_PONG:
      Serial.println("[IOc] Get PONG");
      break;
    default:
      break;
  }
}
```

Figure 4.13. socketIOEvent

This project will link the Arduino to the back-end server (NodeJs) via the socket mechanism, which allows the library to exchange real-time data. Create a method to extract the event name and data from the string in the case of sIOtype_EVENT by removing any extraneous marks from the element.



```
case sIOtype_EVENT:
  {
    char* jsonString = nullptr;
    char* eventString = nullptr;
    const char* delimiter = ",";

    Serial.println("[IOc] Get event: ");
    eventString = reinterpret_cast<char*>(payload);

    if (eventString[0] == '[' && eventString[length - 1] == ']') {
      eventString[length - 1] = '\0';
      eventString++;
    }
    // Convert eventString to an array separated by spaces
    char** eventArray = nullptr;
    int eventCount = 0;

    // Tokenize the eventString
    char* token = strtok(eventString, delimiter);
    while (token != nullptr) {···
    }
    Serial.println(eventArray[1]);
    if (strcmp(eventArray[1], START_SOCKET) == 0) {···
    }
    if (strcmp(eventArray[1], CHANGE_WIFI) == 0) {···
    }
    if (strcmp(eventArray[1], GET_CROP) == 0) {···
    }
    if (!isRunningSocket) {···
    }

    // Free the dynamically allocated memory
```

Figure 4.14. sIOtype_EVENT

Next, compare every event name, perform each event's tasks, and, at the end, use free() to release the memory of the eventArray that was formed by realloc.

```
Serial.println(eventArray[1]);
if (strcmp(eventArray[1], START_SOCKET) == 0) {
  if (!isRunningSocket) {
    isRunningSocket = true;
  }
}
if (strcmp(eventArray[1], CHANGE_WIFI) == 0) {
  const char* ssid = eventArray[2];
  const char* password = eventArray[3];
  delay(200);
  connectToWifi(ssid, password);
}
if (strcmp(eventArray[1], GET_CROP) == 0) {
  if (crop_id != "") {
    crop_id = eventArray[2];
  }
}
if (!isRunningSocket) {
  isRunningSocket = true;
}

// Free the dynamically allocated memory
free(eventArray);
Serial.println(isRunningSocket);

break;
}
```

Figure 4.15. free(eventArray)

4.1.1.4. DS18B20 and ESP8266 connection

The data printed by DS18B20 via Serial will be read by Esp8266.While the TX and RX are connected, use readStringUntil().

In order to do this, you will need to connect a wire to the Arduino Uno's TX port in order to transmit data, and to the ESP8266's RX port in order to receive it.

This is the combination:

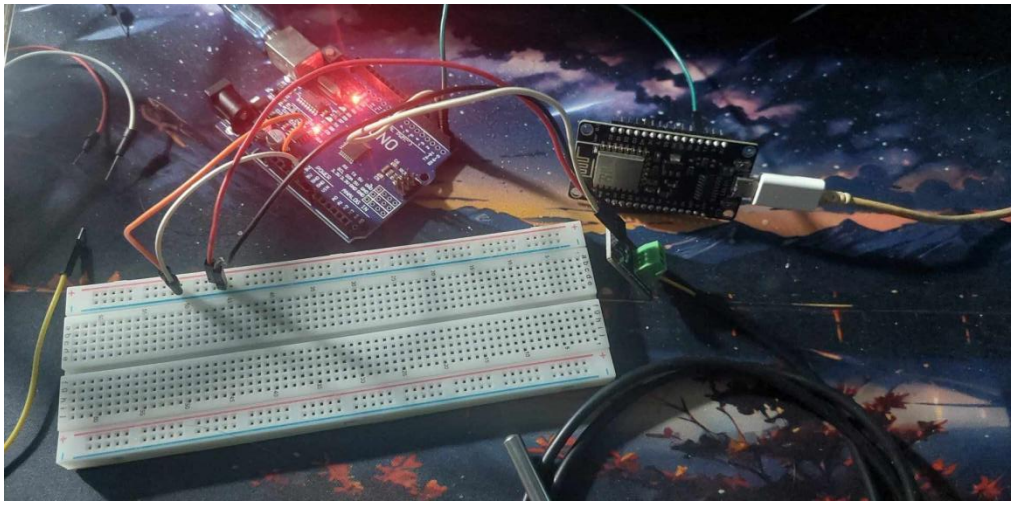


Figure 4.16. DS18B20 and ESP8266

4.1.1.5. JQC-3FF-S-Z, oxygen bubbler and its connection

JQC-3FF-S-Z functions as a lever in this system to switch the bubbler on and off under certain conditions. Just attach JQC-3FF-S-Z to an oxygen bubbler and connect the combination to the Arduino Uno, which is the primary system in this instance.

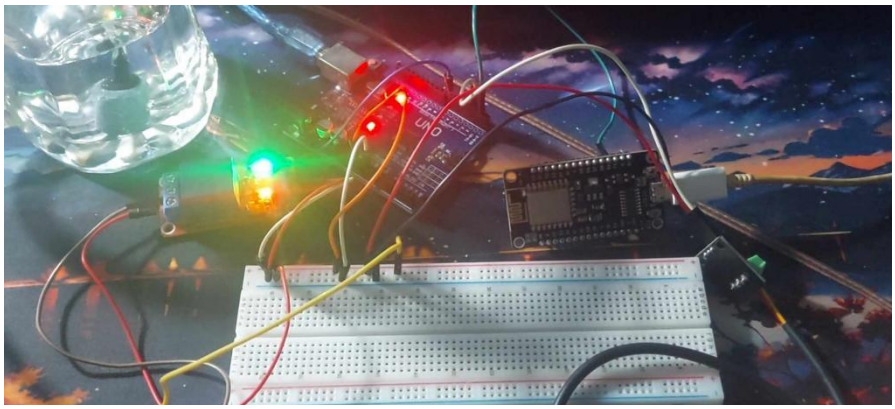


Figure 4.17. System

4.1.2. Software

4.1.2.1. Front-end

Libraries used in this project:

```
"dependencies": {
  "@fortawesome/fontawesome-svg-core": "^6.2.1",
  "@fortawesome/free-regular-svg-icons": "^6.2.1",
  "@fortawesome/free-solid-svg-icons": "^6.2.1",
  "@fortawesome/react-fontawesome": "^0.2.0",
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "@types/jest": "^27.5.2",
  "@types/node": "^16.18.4",
  "@types/react": "^18.0.26",
  "@types/react-dom": "^18.0.9",
  "axios": "^1.2.0",
  "babel-plugin-macros": "^3.1.0",
  "bootstrap": "^5.2.3",
  "chart.js": "^4.0.1",
  "jwt-decode": "^3.1.2",
  "moment": "^2.29.4",
  "react": "^18.2.0",
  "react-bootstrap": "^2.6.0",
  "react-chartjs-2": "^5.0.1",
  "react-dom": "^18.2.0",
  "react-router-dom": "^6.4.4",
  "react-scripts": "5.0.1",
  "sass": "^1.56.1",
  "socket.io-client": "^4.7.2",
  "typescript": "^4.9.3",
  "web-vitals": "^2.1.4"
```

Figure 4.18. Dependencies

Distinguished libraries consist of:

- Axios: make an API call.
- Socket.io-client: exchange real-time data by corresponding with the socket server.
- Typescript: an enhanced JavaScript programming language with robust typing that provides superior tools at all sizes.
- Chart.js: make a line graph that tracks the data.

This project's structure is designed to be easily maintained, with features like context, utils, and APIs readily visible and accessible. Declare components, models, and enums explicitly:

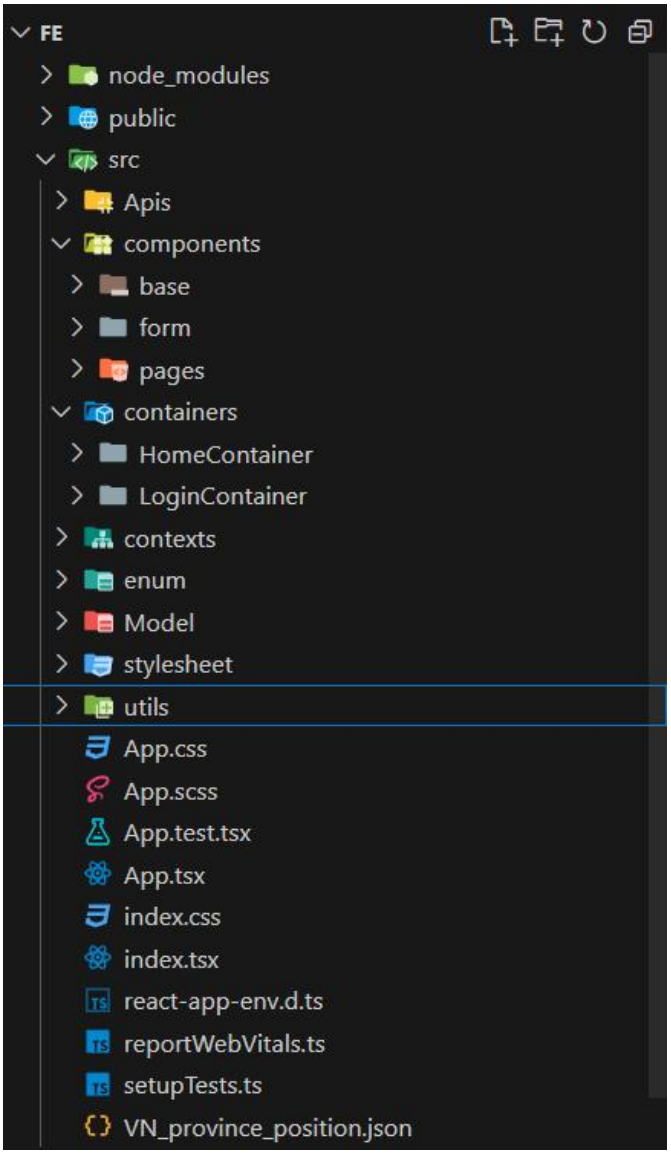


Figure 4.19. FE structure

The weather forecasting and scenario-setting aspects of this project are crucial in informing users about potential outcomes and how to avoid shrimp ponds.

Use the public weather URL (https://api.open-meteo.com/v1/forecast) to get weather forecasting information. The weather for today and the following day will be computed and returned. After supplying the longitude and latitude, a JSON object is returned; choose the properties that are required and appropriate for the prediction method:

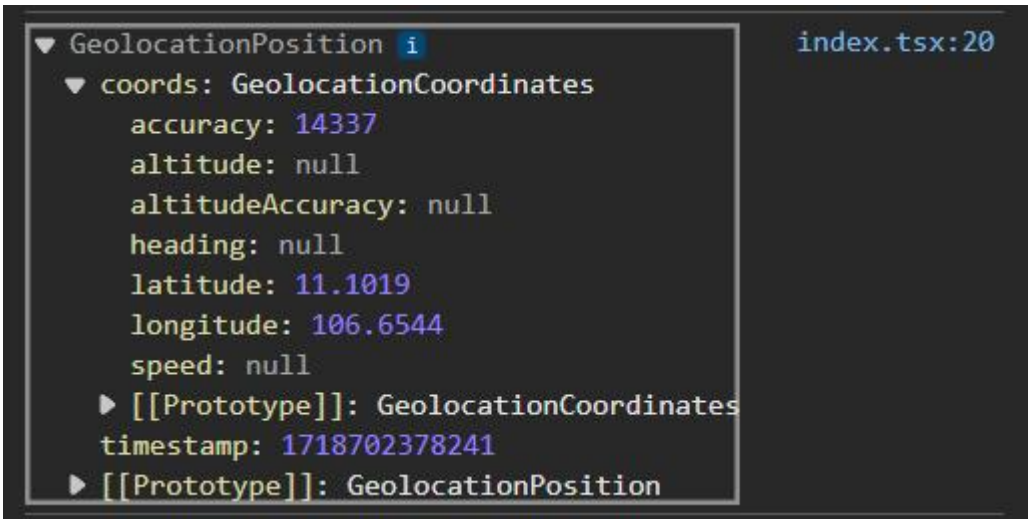


Figure 4.20. Latitude and longitude values

Latitude and longitude are coordinates used to determine any location on Earth:

- *Latitude: Measures distance north or south of the Equator (0°), ranging from 0° to 90°.*
- *Longitude: Measures distance east or west of the Prime Meridian (0°), ranging from 0° to 180°.*

These coordinates pinpoint locations precisely.

```
export const handlePredictWeather = (
  times: string[],
  weatherCode: number[]
) => {
  let isRainyToday = false;
  let isRainyTomorrow = false;

  let startRainyToday: Date = new Date();
  let startRainyTomorrow: Date = new Date();

  const today = new Date();
  today.setDate(today.getDate() + 1);

  const currentDate = new Date().getDate();
  const nextDate = today.getDate();

  times.forEach((time, index) => {
    if (new Date(time).getDate() === currentDate && weatherCode[index] > 60) {
      if (!isRainyToday && new Date(time).getHours() > new Date().getHours()) {
        isRainyToday = true;
        startRainyToday = new Date(time);
      }
    } else if (
      new Date(time).getDate() === nextDate &&
      weatherCode[index] > 60
    ) {
      if (!isRainyTomorrow) {
        isRainyTomorrow = true;
        startRainyTomorrow = new Date(time);
      }
    }
  });

  return {
    today: {
      time: new Date(startRainyToday.setHours(startRainyToday.getHours() + 7)),
      isRainy: isRainyToday,
    },
    tomorrow: {
      time: new Date(
        startRainyTomorrow.setHours(startRainyTomorrow.getHours() + 7)
      ),
      isRainy: isRainyTomorrow,
    },
  };
};
```

Figure 4.21. Weather predict function

4.1.2.2. Back-end

Libraries used in this project:

```
"dependencies": {
  "bcrypt": "^5.1.0",
  "cors": "^2.8.5",
  "dotenv": "^16.0.3",
  "express": "^4.18.2",
  "jsonwebtoken": "^9.0.0",
  "moment": "^2.29.4",
  "mysql2": "^2.3.3",
  "nodemon": "^2.0.20",
  "python-shell": "^5.0.0",
  "serialport": "^12.0.0",
  "socket.io": "^4.7.2",
  "uuid": "^9.0.0"
}
```

Figure 4.22. BE libraries

Distinguished libraries consist of:

- Python-shell: transmit and retrieve data from the Python language is a notable library.
- Socket.io: exchange real-time data by corresponding with a socket client.
- Mysql2: establish a database connection (MySQL)
- Express: offers a powerful feature set for both online and mobile apps. It is a simple and adaptable Node.js web application framework.

Database connection:

```
const mysql = require("mysql2");
const dotenv = require("dotenv");

dotenv.config();

const connection = mysql.createConnection({
    host: process.env.DB_HOST,
    port: 3306,
    user: process.env.DB_USER,
    database: process.env.DB_NAME,
    password: process.env.DB_PASSWORD,
    multipleStatements: true,
});

module.exports = connection;
```

Figure 4.23. Database connection

Socket connection:

```
const socketFunction = (io) => {
    io.on("connection", (client) => {
        console.log("Socket on!");

        client.on("init_tracking", (list_cropIds) => {
            list_cropIds.forEach((crop_id) => client.join(crop_id));
        });

        client.on("create_tracking", (crop_id) => {
            client.join(crop_id);
        });

        client.on("get_port_list", () => {
            SerialPort.list().then(function (ports) {
                client.emit("port_list", ports);
            });
        });
```

Figure 614.24. Socket connection

In order to communicate between back-end and database, routers is created with HTTP method and use the SQL query to access required data.

```
//Sign up
router.post("/register", async (req, res) => {
    try {
        console.log(req.body);
        let mail = req.body.email;
        let name = mail.split("@")[0];
        let password = req.body.password;

        const saltRounds = 10;
        const salt = bcrypt.genSaltSync(saltRounds);
        const hash = bcrypt.hashSync(password, salt);

        let sql = "INSERT INTO USER (name, password, isAdmin) VALUES (?,?,?);";

        sql &&
            name &&
            password &&
            connection.query(sql, [name, hash, false], (err, results) => {
                if (err) {
                    res.status(400).json("Information may incorrect!");
                } else {
                    res.status(200).json(results);
```

Figure 4.25. Example of router

4.1.2.3. Database

A schema is initially establish, after that several tables is created by some basic queries.

```
create table USER (
    name varchar(255) UNIQUE NOT NULL PRIMARY KEY,
    password varchar(255) NOT NULL,
    refreshToken varchar(255),
    accessToken varchar(255),
    isAdmin boolean NOT NULL
);
```

Figure 4.26. Create user table (example)

4.2. Results

4.2.1. Hardware

After connect the system successfully and load all the code, the temperature will be constantly update by DS18B20, in every 5 seconds.

Figure 4.27. Temperature value

Beside, the ESP8266 will continuously try to establish a WiFi connection until it is successful. It will print a confirmation message as soon as it is connected. When finished, it call an API to received the value “crop_id” of the crop that is already assigned by an IoT value in the database . It will then create a connection using SocketIO. Lastly, it will call the API once every ten seconds. This API send the temperature value from DS18B20 to back-end.

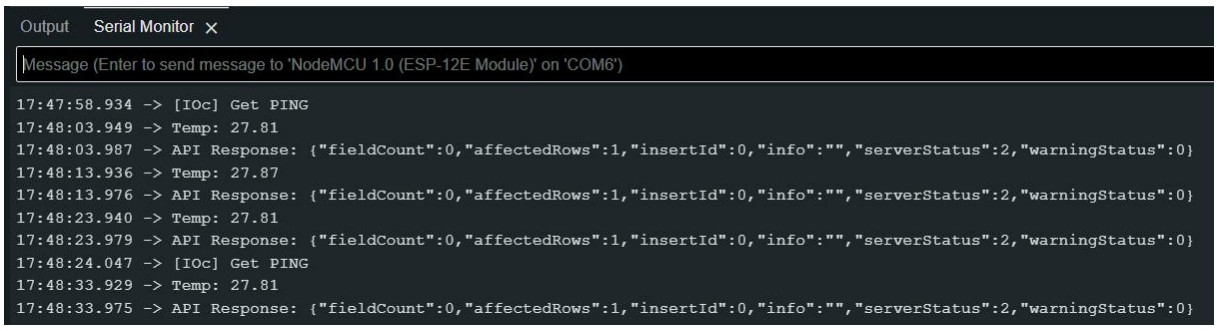


Figure 4.28. ESP8266 printed

4.2.2. Back-end

Type npm start in BE to start:

```
PS D:\Study\Thesis\Code\shrimp-app\be> npm start

> be@1.0.0 start
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
App listening at port:  7001
DB connected!
Socket on!
Socket on!
{ isConnect: 'true' }
```

Figure 4.29. BE connected

NodeJS server side will receive hardware events through SocketIO.

```
Socket on!
{ isConnect: 'true' }
Temperature received: 28.56
Temperature received: 28.56
Temperature received: 29.31
Temperature received: 30.12
Temperature received: 30.62
Temperature received: 31.00
Temperature received: 31.25
```

Figure 4.30. BE printed

4.2.3. Front-end

Type npm start in FE to start:

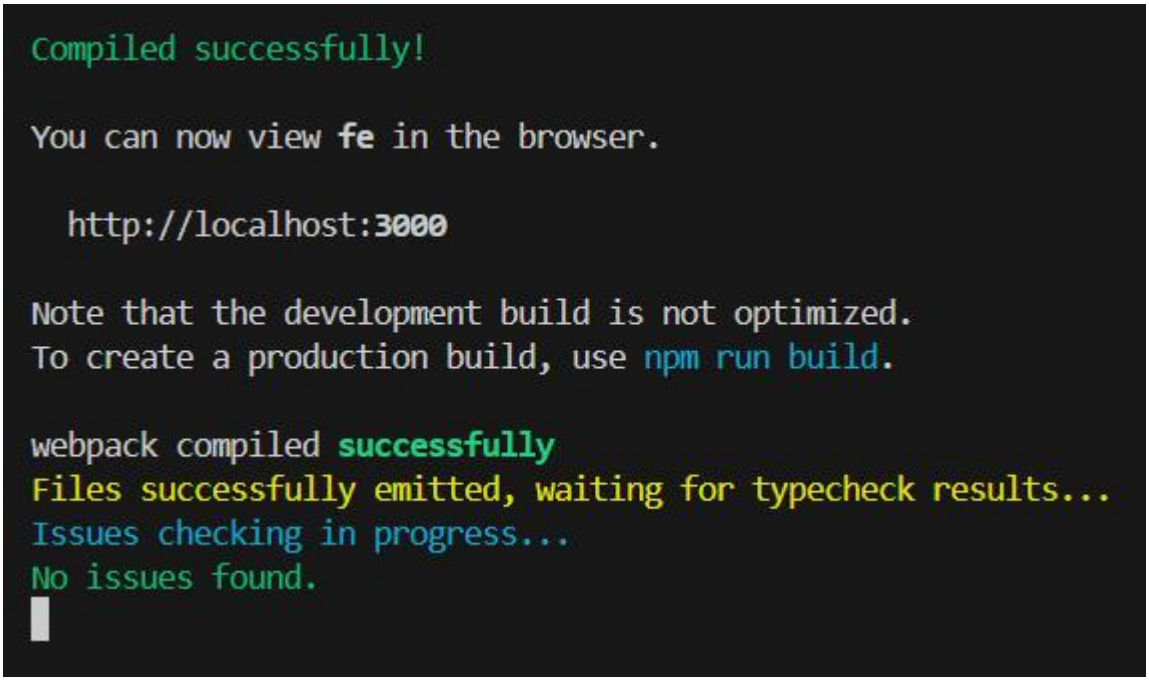


Figure 4.31. FE connected

4.2.4. Database

Figure 694.31. FE connected

There are 8 tables to separate and store data for Shrimplify, but some of them hold a critical position to the project:

- User: include user's name, password (has already been encrypted), refresh and access token provide by JWT for authentication.

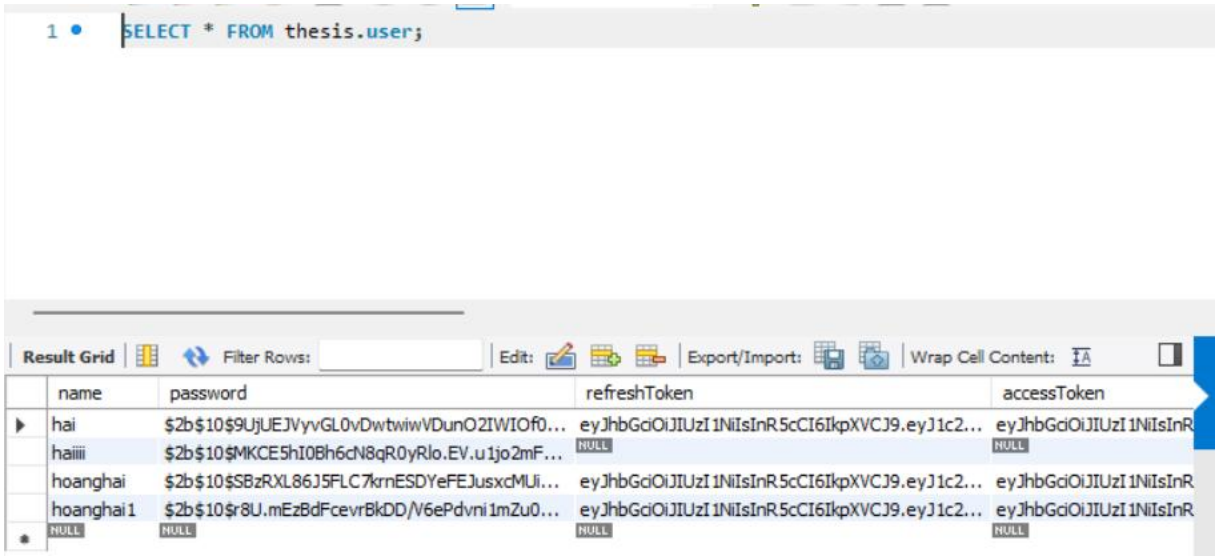


Figure 4.33. User table

- The main key is the id column, which contains the user's name and some information pertinent to the shrimp pond. The format of the crop table and the pond is the same.

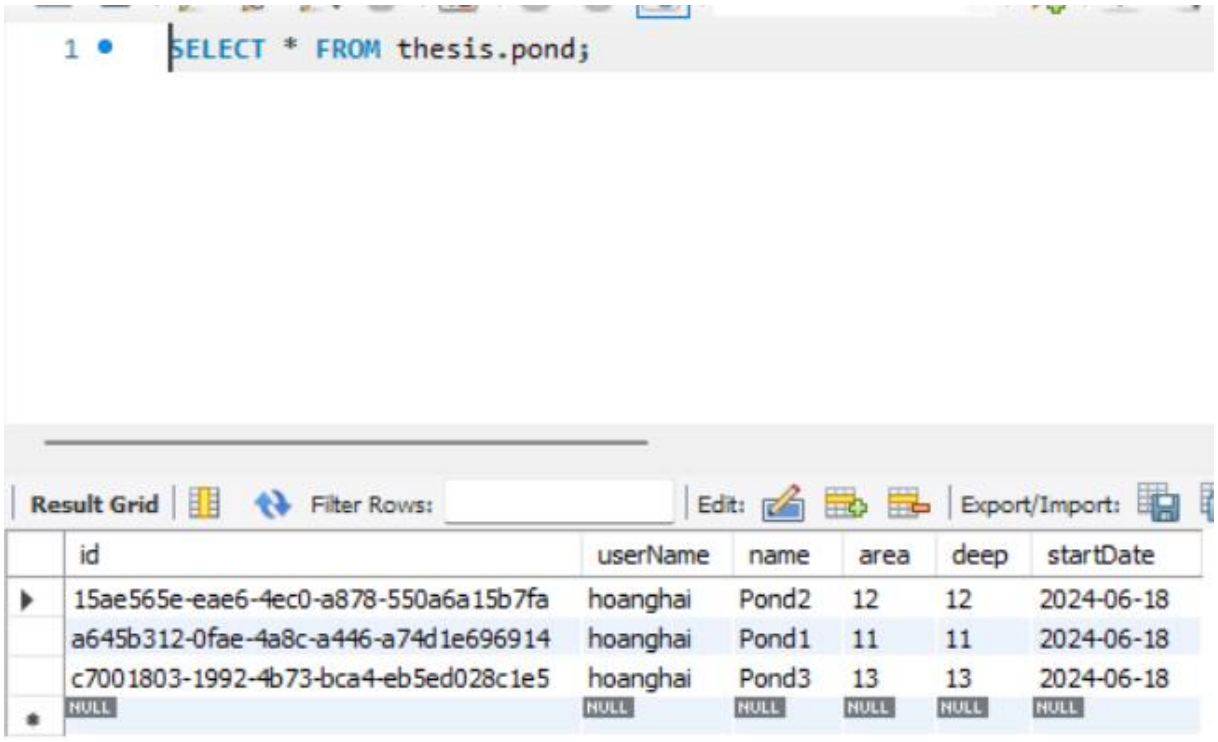


Figure 4.34. Pond table

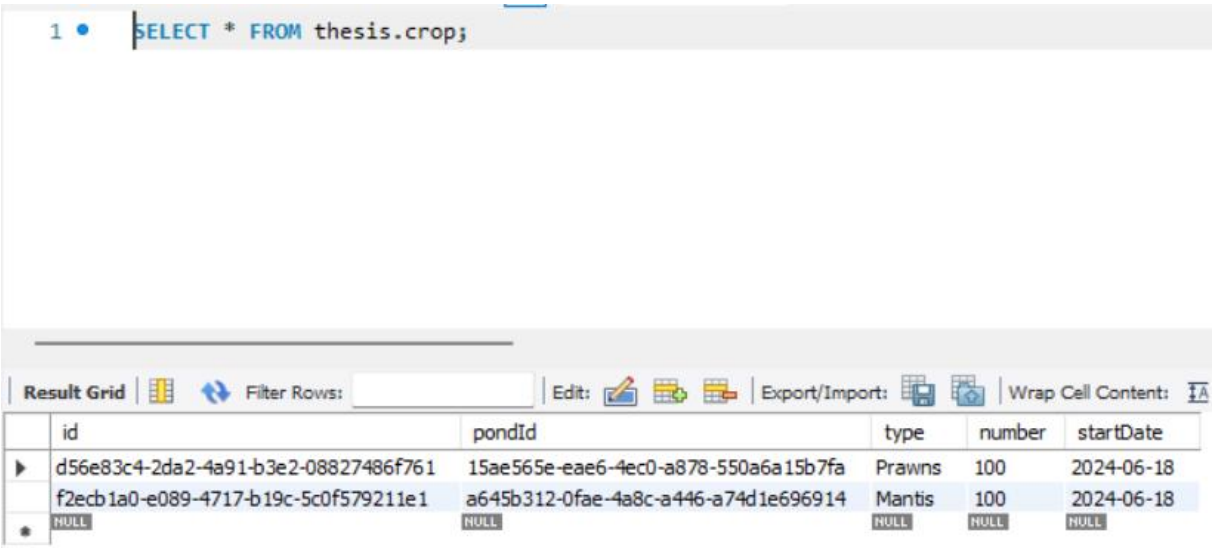


Figure 4.35. Crop table

- The crop_stat table will show us which IoT device is being used by each crop tracking element and whether it is active or not.

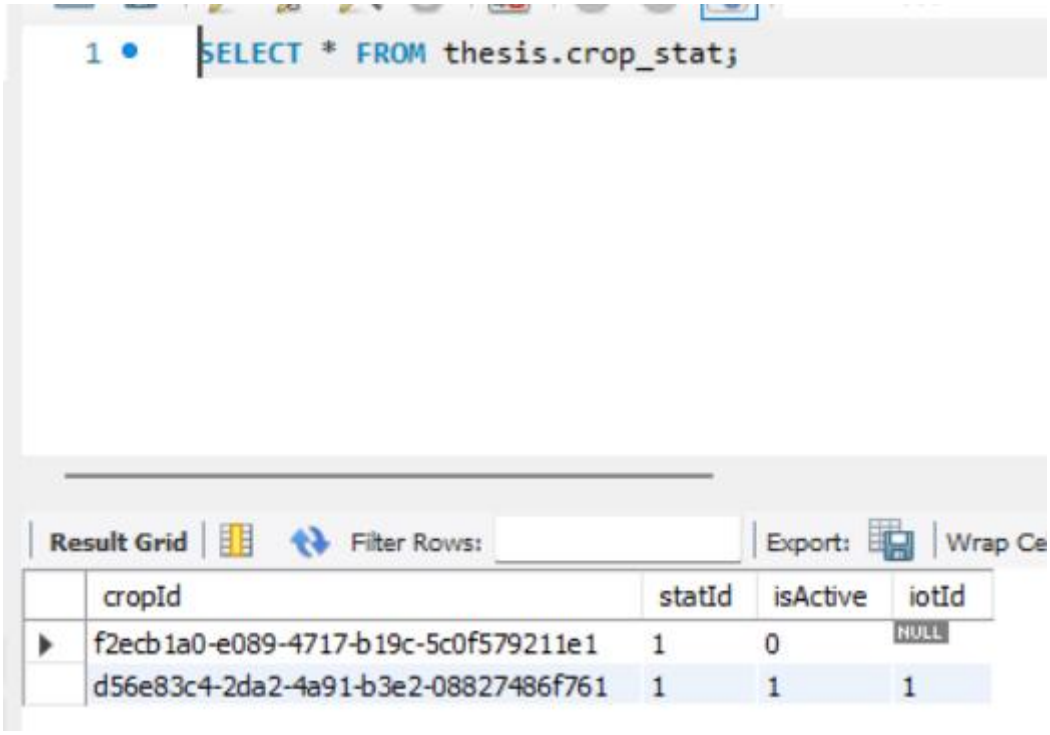


Figure 4.36. Crop_stat table

- All of the characteristics of the shrimp that users may select to connect to IoT devices will be included in the stat table. The administrator position has the ability to change the table and will choose the best range for each characteristic.

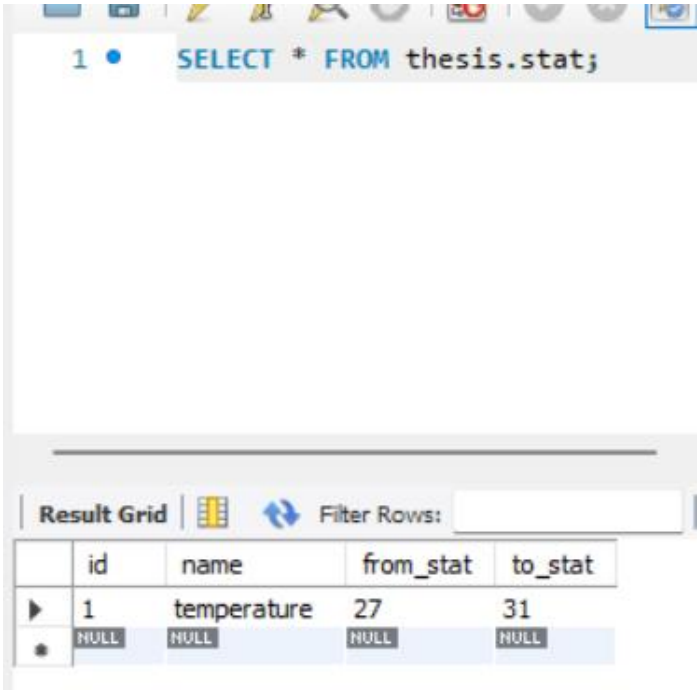


Figure 4.37. Stat table

- Dataset table for further improvement

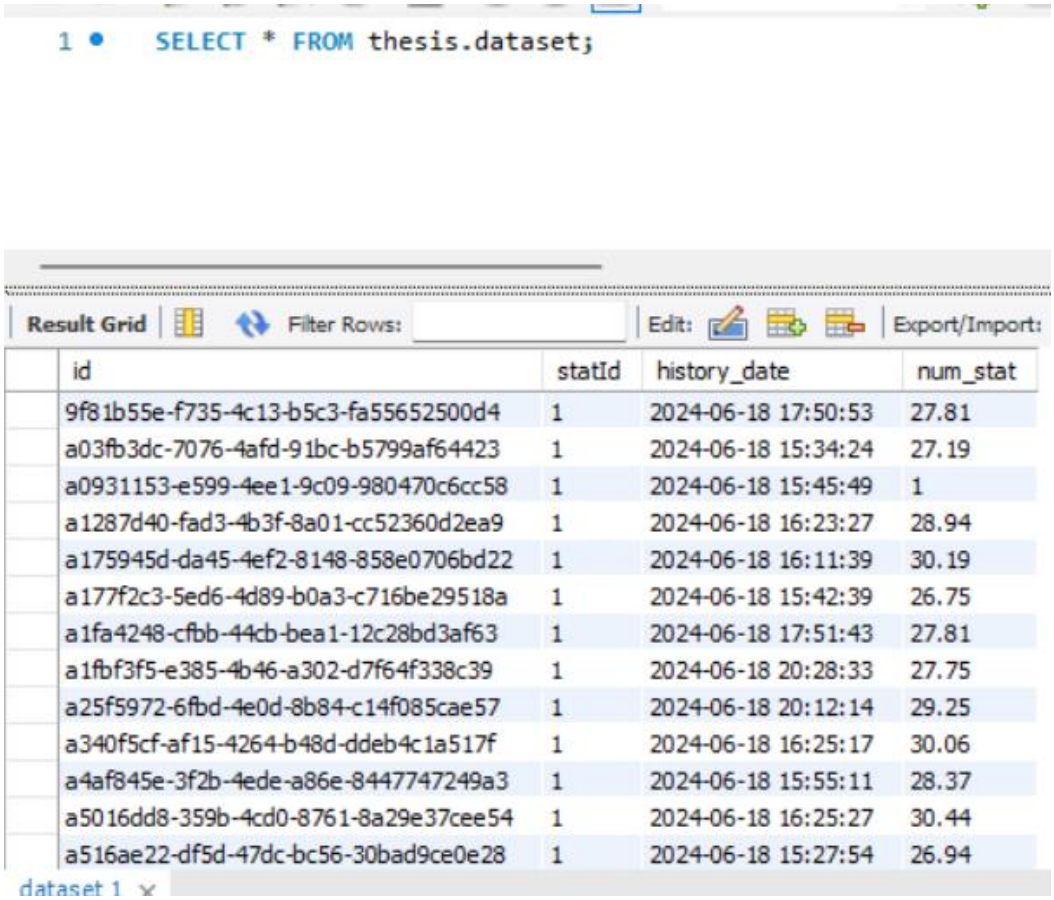


Figure 4.38. Dataset table

CHAPTER 5

DISCUSSION AND EVALUATION

5.1. Discussion

5.1.1. Key Findings

This study reveals significant advancements in pond management through Shrimplify. Real-time data tracking enables farmers to monitor critical factors like temperature, dissolved oxygen, pH, and ammonia levels. The app logs and visualizes data, helping to identify trends and potential issues, thereby reducing shrimp mortality and improving pond health. Automation of devices like oxygen bubblers ensures consistent conditions with minimal human intervention. Remote access allows farmers to manage ponds from any location, which is particularly useful for those with multiple ponds. The user-friendly interface accommodates farmers of all technological levels, and customization alerts enhance the app's utility and effectiveness.

5.1.2. Challenges

Several technical issues must be addressed to ensure the effective functioning of Shrimplify. One of the primary concerns is sensor reliability, as it is imperative to provide precise and dependable sensor data to maintain optimal pond conditions. Additionally, stable internet connectivity is crucial for remote administration and accessing real-time data, which can be a challenge in some rural areas.

User adoption also poses significant challenges. Adequate assistance and training are essential to enable efficient use of the app, particularly for farmers who may not be technologically savvy. Moreover, the initial costs associated with purchasing and setting up sensors can be a substantial obstacle for some farmers, potentially hindering widespread adoption.

5.2. Evaluation

This project is not yet finished, but it has successfully developed most of its core capabilities through a combination of practice, study, and the application of knowledge and expertise.

5.2.1. Concept

The concept and development of Shrimplify represent an innovative approach to shrimp farming. The app introduces cutting-edge features such as automated control and real-time monitoring, aiming to enhance sustainability and production in the shrimp farming industry. There is a clear market demand for improved aquaculture pond management solutions, which Shrimplify seeks to address.

5.2.2. Development

The development process involved multiple iterations of prototyping and testing to refine the app's functionality and user experience. These steps included rigorous testing phases to ensure the reliability and efficiency of the app's features. Feedback from stakeholders, including farmers and aquaculture specialists, was integral to the development process. Their suggestions were incorporated to improve the usability and effectiveness of the app, ensuring it meets the practical needs of its users.

5.2.3. Future Prospects

As Shrimplify scales, strategies include expanding user capacity and integrating advanced features such as predictive analytics and connections with larger farm management systems. Addressing technical challenges, such as connectivity issues and sensor reliability, will involve robust support and high-quality hardware. To enhance user adoption, providing thorough training and exploring cost-reduction options for initial setup will be crucial in making the app accessible to a broader range of farmers.

5.2.4. Conclusion

Emphasize how cutting-edge technology and user-centered design could revolutionize shrimp farming techniques with Shrimplify. Stress the significance of more testing, improvement, and eventual publishing in order to confirm its effectiveness and increase its influence within the aquaculture sector.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

This shrimp farming application, Shrimplify, gives small-scale shrimp farmers a crucial tool for efficiently running their businesses. Shrimplify assists farmers in making the most use of their resources and raising production by providing tools like feed management, health tracking for shrimp, and real-time water quality monitoring. Even those with no technical experience may readily use the tool to improve their agricultural operations because to its user-friendly layout. This research shows how technology may help with successful and sustainable shrimp farming.

6.2. Future Work

To enhance Shrimplify, several key feature expansions are planned to improve usability, functionality, and community support.

- Predictive Analytics: Implement integrated machine learning to forecast pond conditions, helping farmers anticipate and manage potential issues before they arise, thereby optimizing shrimp health and farm productivity.
- System Integration: Enhance communication across different farm management platforms to ensure seamless data flow and interoperability, allowing farmers to have a more holistic view of their operations.
- Mobile Compatibility: Develop a mobile-friendly version of Shrimplify to enable farmers to access and manage their operations from their smartphones, providing flexibility and real-time monitoring.
- Broader Applications: Adapt Shrimplify for use with other aquaculture species, expanding its utility and market reach. Additionally, support for more types of sensors will be added to offer thorough data on ambient conditions and shrimp health.

- Community Support: Establish a forum for farmers to share experiences, seek advice, and support each other. Create a user feedback loop to gather input and update the program regularly based on real-world use and current user demands.
- Localized Support: Adapt Shrimplify to various geographical areas by incorporating language options and customizing suggestions to suit regional farming conditions and practices, ensuring relevance and effectiveness for farmers worldwide.

REFERENCES

1. Farmext - Nuôi trồng dễ dàng
   https://farmext.com/about/vi/
2. End to End Shrimp Industry Solution | JALA
   https://jala.tech/
3. CIBA Shrimp App
   https://play.google.com/store/apps/details?id=com.vanami.shrimpapp&hl=en/
4. Farmed Shrimp | Industries | WWF (worldwildlife.org)
   https://www.worldwildlife.org/industries/farmed-shrimp
5. "QUẢN LÝ NHIỆT ĐỘ AO NUÔI TÔM HIỆU QUẢ" (tomota.vn)
   https://tomota.vn/tin-tuc/quan-ly-nhiet-do-ao-nuoi-tom-hieu-qua-50
6. Control Water Quality in Shrimp Farming:
   https://www.maheshaqua.com/technical-info/water-quality-management/
7. Kiểm soát và đo pH trong ao tôm – Tạp chí Thủy sản Việt Nam (thuysanvietnam.com.vn)
   https://thuysanvietnam.com.vn/kiem-soat-va-do-ph-trong-ao-tom/
8. [Thông tin] kỹ thuật NodeMCU ESP8266 | Mecsu.vn
   https://mecsu.vn/ho-tro-ky-thuat/thong-tin-ky-thuat-nodemcu-esp8266.R1q/
9. Cảm biến nhiệt độ DS18B20 | Mecsu.vn
   https://mecsu.vn/ho-tro-ky-thuat/cam-bien-nhiet-do-ds18b20.r1K
10. Arduino UNO R3 là gì? | Cộng đồng Arduino Việt Nam
    http://arduino.vn/bai-viet/42-arduino-uno-r3-la-gi
11. FAO (Food and Agriculture Organization of the United Nations). "Shrimp Farming." Retrieved from FAO.
12. Browdy, C. L., & Moss, S. M. (2005). "Shrimp Culture in the Twenty-First Century." Oceanography, 18(2), 35-39.
13. SSH là gì? Tổng hợp A-Z về SSH cho người bắt đầu:

https://tenten.vn/tin-tuc/ssh-la-gi-tong-hop-a-z-ve-ssh-cho-nguoi-bat-dau/

14. What is hashing and how does it work?

https://www.techtarget.com/searchdatamanagement/definition/hashing

15. JSON Web Token (JWT) là gì?

https://topdev.vn/blog/jwt-la-gi/

16. What Are Refresh Tokens and How to Use Them Securely

https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/