

**WATER QUALITY MONITORING IN AQUACULTURE TO
INCREASE FISH GROWTH PERFORMANCE BASED ON SENSOR
OUTCOMES.**

PHANG JUN SEN


**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Science
(Honours) Software Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

March 2023

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : Phang Jun Sen


ID No. : 010204-02-0767

Date : 5 October 2023

APPROVAL FOR SUBMISSION


I certify that this project report entitled **“WATER QUALITY MONITORING IN AQUACULTURE TO INCREASE FISH GROWTH PERFORMANCE BASED ON SENSOR OUTCOMES”** was prepared by **PHANG JUN SEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Ts Dr Sugumaran a/l Nallusamy

Date : 05/10/2023

Signature : 

Co-Supervisor : Dr Kwan Ban Hoe

Date : 5 Oct 2023

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2023, PHANG JUN SEN. All right reserved.

ACKNOWLEDGEMENT

I am deeply grateful to my project supervisor, Dr. Ts Sugumaran a/l Nallusamy, for his valuable insights, guidance and unwavering help throughout the project. His expertise in architectural design and system functionality was instrumental in determining the direction of the project. Additionally, I am grateful that he allowed me access to his aquaculture farm, which enabled me to deploy data acquisition modules in his tanks to collect key water parameters for research.

I sincerely thank my co-supervisor Dr. Kwan Ban Hoe for generously sponsoring the necessary equipment required to set up the sensor module.

I am also grateful to Dr. Ng Oon Ee for his guidance in configuring the sensor module. His provision of the Arduino board and valuable ideas for microcontroller programming and sensor calibration were crucial to the success of the project.

My sincere appreciation goes to Universiti Tunku Abdul Rahman for their unwavering support by providing the essential resources, infrastructure, and access to library facilities, all of which greatly facilitated my research and project development.

I would also like to express my gratitude to my friends and peers who played an integral role in assisting me with building the sensor module. Their contributions and participation in the System Usability Testing to evaluate the performance of the water quality monitoring mobile application were immensely valuable.

In conclusion, the successful completion of this project was made possible through the collective efforts and support of these individuals and institutions. I am truly grateful for their contributions, which have significantly contributed to my academic journey and personal growth.

ABSTRACT

The increasing demand for sustainable aquafarming practices has prompted the development of advanced water quality monitoring systems. This project introduces a comprehensive Water Quality Monitoring System that encompasses four key modules: the Data Acquisition Module, Communication Module, Data Processing and User Interface Module. The project's objectives encompass analyzing existing aquafarming tools, conducting water quality analysis, developing a mobile application for data visualization, and evaluating water quality to optimize fish growth and maintain ideal conditions. An evolutionary prototyping approach was used for system development and successful implementation. In the end, the objectives are achieved when the water quality monitoring system was successfully developed and deployed in an aquaculture farm for water quality monitoring. The developed data collection module can efficiently collect and transmit data to the ThingSpeak cloud server, which stores and provides REST API for data processing and retrieval. The user interface module runs efficiently on the Android emulator and cooperates with the data processing module to provide data processing, user authentication and authorization, and machine learning data prediction to support real-time water parameter monitoring. In conclusion, this FYP report discusses the system's achievements, limitations, and recommendations for future enhancements. While the system achieved its goals, certain limitations emerged during testing, leading to suggestions for improvement. This project represents a significant step toward efficient and sustainable aquafarming practices through advanced water quality monitoring.

TABLE OF CONTENTS

DECLARATION	i
APPROVAL FOR SUBMISSION	ii
ACKNOWLEDGEMENT	iv
ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF SYMBOLS / ABBREVIATIONS	xxii
LIST OF APPENDICES	xxiii
CHAPTER 1 24	
INTRODUCTION	24
1.1 General Introduction	24
1.2 Problem Statement	25
1.2.1 Aeration system failure	25
1.2.2 Algae Bloom	26
1.2.3 Unpredictable timing to change water	26
1.3 Aim and Objectives	27
1.4 Proposed Solution	28
1.5 Proposed Approach	29
1.6 Project Scope	30
1.6.1 Sensor Module	30
1.6.2 Data Acquisition Module	30
1.6.3 Communication Module	31
1.6.4 User Interface Module	31
CHAPTER 2 32	
LITERATURE REVIEW	32
2.1 Aquaculture	32
2.2 Recirculating Aquaculture System	33
2.3 Internet of Things in Aquaculture	33
2.4 Water Quality	36
2.4.1 Importance of Water Quality	36
2.4.2 Water Quality Parameters	36

2.5	Existing Aquaculture Monitoring System	40
2.5.1	Automated water quality monitoring system development via LabVIEW for aquaculture industry (Tilapia) in Malaysia	40
2.5.2	IoT-based Smart Aquaculture System with Automatic Aerating and Water Quality Monitoring	49
2.5.3	Development of an IoT-based Intensive Aquaculture Monitoring System with Automatic Water Correction	58
2.6	Comparison and Analysis	69
2.7	Software Development Methodology	73
2.7.1	Waterfall	73
2.7.2	Agile	75
2.7.3	Lean	76
2.8	Comparison of Software Development Methodology	77
2.9	Justification of Aquaculture Monitoring System	78
CHAPTER 3	80	
	METHODOLOGY AND WORK PLAN	80
3.1	Introduction	80
3.2	Software Development Methodology	81
3.2.1	Initial Set of User Requirements	82
3.2.2	Quick Design	83
3.2.3	Develop Prototype	83
3.2.4	Evaluation of prototype by the customer	84
3.2.5	Refine Requirements	84
3.2.6	Refine Prototype	84
3.2.7	Test the final product	84
3.2.8	Deliver and maintain	85
3.3	Project Plan	85
3.3.1	Work Breakdown Structure (WBS)	85
3.3.2	Gantt Chart	88

3.3.3	Development Tool	97
CHAPTER 4	100	
	PROJECT SPECIFICATION	100
4.1	Introduction	100
4.2	Proposed System Architecture	100
4.2.1	Sensor	101
4.2.2	Data Acquisition Module	101
4.2.3	Cloud Server	102
4.2.4	Web and Mobile Application	102
4.3	Requirement Specification	105
4.3.1	Functional Requirement	105
4.3.2	Non-Functional Requirement	106
4.3.3	Use Case Diagram	108
4.3.4	Use Case Description	109
CHAPTER 5	121	
	SYSTEM DESIGN	121
5.1	Introduction	121
5.2	System Architecture Design	121
5.2.1	Data Acquisition Module Architecture	122
5.2.2	Communication Module Architecture	127
5.2.3	Data Processing Module	131
5.2.4	User Interface Module	135
5.3	Conclusion	138
CHAPTER 6	139	
	SYSTEM IMPLEMENTATION	139
6.1	Introduction	139
6.2	Project Setup	139
6.2.1	Communication Module Setup	139
6.2.2	Data Acquisition Module Setup	143
6.2.3	Data Processing Module Setup	152
6.2.4	User Interface Module Setup (React Native Android Application)	158

6.3	Water Quality Monitoring Mobile Application	
	functions	163
6.3.1	Register	163
6.3.2	Login	170
6.3.3	Channels	176
6.3.4	Dashboard	186
6.3.5	Activity Record	199
6.3.6	Analysis	206
6.3.7	Notification	219
6.4	System Deployment	223
6.4.1	Data Acquisition Module Deployment	223
6.4.2	Communication Module Deployment	225
6.4.3	Data Processing Module Deployment	227
6.4.4	User Interface Module Deployment	228
6.5	Conclusion	230
CHAPTER 7	231	
	SYSTEM TESTING	231
7.1	Introduction	231
7.2	Unit Testing	232
7.2.1	Unit Testing for Register	232
7.2.2	Unit Testing for Login	236
7.2.3	Unit Testing for Channels	240
7.2.4	Unit Testing for Dashboard	246
7.2.5	Unit Testing for Activity Record	252
7.2.6	Unit Testing for Analysis	256
7.3	Performance Testing	258
7.3.1	Register an account	258
7.3.2	Login	259
7.3.3	Dashboard	260
7.3.4	Add Event	261
7.3.5	Analysis	262
7.3.6	Laravel Back-end Server API Endpoint	
	Response Time Testing	264

7.4	System Usability Test	264
7.4.1	Test Scenario for Usability Testing	265
7.4.2	Result of Usability Testing	266
CHAPTER 8	270	
	CONCLUSION & RECOMMENDATION	270
8.1	Conclusion	270
8.2	Limitation and Recommendation for future work	271
REFERENCES		273
APPENDICES		280

LIST OF TABLES

Table 2.2.4.1 Standard range for water quality parameter set by fisheries research institute (FRI) Malaysia (Azlan Othman et al., 2020)	39
Table 2.2.4.2 Water quality tolerance by species (FRESHWATER-AQUACULTURE, 2019)	39
Table 2.5.1 The accuracy fo temperature sensor of the system (Azlan Othman et al., 2020)	46
Table 2.5.2 Table of temperature and pH level with time (Azlan Othman et al., 2020)	47
Table 2.5.3 Fish Growth In Each Setup (Tolentino et al., 2021)	69
Table 2.6.1 Comparison table between LabVIEW, ISAS, IAMS.	69
Table 4.3.1 Functional requirements of water quality monitoring mobile application	105
Table 6.3.1 Register Function Success Case	163
Table 6.3.2 Register Function Failure Case	165
Table 7.4.1 Tables of Test Scenarios	265
Table 7.4.2 Template of User Satisfactory Survey	267
Table 7.4.3 SUS results	267
Table 7.4.4 Summary of Respondents' Most Liked Features of the System 267	
Table 7.4.5 Summary of Respondents' Least Liked Features of the System 268	
Table 7.4.6 Summary of Respondents' Suggestion to improve the system 268	
Table 8.2.1 Limitation and Recommendation	271

LIST OF FIGURES

Figure 1.1.1 World aquaculture production of aquatic animals and aquatic plants, 1990–2017 (Zhou, 2019)	24
Figure 1.4.1 Proposed system architecture	28
Figure 2.5.1 The block diagram of water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)	40
Figure 2.5.2 The block diagram of water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)	41
Figure 2.5.3 The layout water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)	43
Figure 2.5.4 The front panel of the system via LabVIEW software (Azlan Othman et al., 2020)	44
Figure 2.5.5 Manual set-up versus automated set-up (Azlan Othman et al., 2020)	45
Figure 2.5.6 The front panel shows when the temperature exceeds 32 °C (Azlan Othman et al., 2020)	46
Figure 2.5.7 The architecture of ISAS (Tsai et al., 2022)	49
Figure 2.5.8 The sensors and data collection platforms of the ISAS (Tsai et al., 2022)	50
Figure 2.5.9 The automatic control flow of the ISAS (Tsai et al., 2022)	52
Figure 2.5.10 The temperature sensed data example (Tsai et al., 2022)	53
Figure 2.5.11 Sensed data dashboard from user's mobile device (Tsai et al., 2022)	54
Figure 2.5.12 Sensed data dashboard from user's mobile device (Tsai et al., 2022)	54
Figure 2.5.13 Experimental group and control group of the ISAS (Tsai et al., 2022).	55
Figure 2.5.14 Survival rates of shrimps during the 1.5 months of our experiment (Tsai et al., 2022)	56
Figure 2.5.15 The architecture of ISAS (Tolentino et al., 2021)	58

Figure 2.5.16 TeamLapia Web Application Interface (Tolentino et al., 2021)	61
Figure 2.5.17 TeamLapia Web Application Structure Diagram (Tolentino et al., 2021)	62
Figure 2.5.18 Experimental group and control group of the ISAS (Tolentino et al., 2021)	64
Figure 2.5.19 pH sensor Readings with Correction Response (Tolentino et al., 2021)	65
Figure 2.5.20 Turbidity Sensor Readings with Correction Response (Tolentino et al., 2021)	65
Figure 2.5.21 Oxidation Reduction Potential Sensor Readings with Correction Response (Tolentino et al., 2021)	66
Figure 2.5.22 Temperature Sensor Readings with Correction Response (Tolentino et al., 2021)	66
Figure 2.5.23 Salinity Sensor Readings with Correction Response (Tolentino et al., 2021)	67
Figure 2.5.24 Dissolved Oxygen Sensor Readings with Correction Response (Tolentino et al., 2021)	67
Figure 2.5.25 Controlled vs. Conventional Fish Growth measured every week (Tolentino et al., 2021)	68
Figure 2.6.1 ISAS dashboard in mobile application (Tsai et al., 2022).	72
Figure 2.6.2 The dissolved oxygen sensed data example (Tsai et al., 2022). 73	
Figure 2.7.1 Waterfall approach (Visual Paradigm, n.d.)	74
Figure 2.7.2 Principle of Agile Methodology (OS-system, 2020)	76
Figure 2.7.3 Principle of Lean Methodology (OS-system, 2020)	77
Figure 3.2.1 The difference between the evolutionary prototyping and rapid throw away prototyping (02DCE @02DCE, 2020).	82
Figure 3.3.1 Gantt Chart for Project Initialization from 30/1/2023 to 27/2/2023	88
Figure 3.3.2 Gantt Chart for Project Initialization from 27/2/2023 to 27/5/2023	89

Figure 3.3.3 Gantt Chart for Project Initialization from 27/5/2023 to 24/4/2023	90
Figure 3.3.4 Gantt Chart for System Development from 20/4/2023 to 15/5/2023	91
Figure 3.3.5 Gantt Chart for System Development from 15/5/2023 to 12/6/2023	92
Figure 3.3.6 Gantt Chart for System Development from 12/6/2023 to 10/7/2023	93
Figure 3.3.7 Gantt Chart for System Development from 10/7/2023 to 7/8/2023	94
Figure 3.3.8 Gantt Chart for System testing and Deployment from 12/8/2023 to 28/8/2023	95
Figure 4.2.1 Proposed system architecture	100
Figure 4.2.2 Login View	103
Figure 4.2.3 Fish Tanks	103
Figure 4.2.4 Water Tank A Water Parameters Gauge Dashboard	103
Figure 4.2.5 Water Tank A Temperature Details	104
Figure 4.2.6 Warning Message	104
Figure 4.3.1 Use case diagram of Water Quality Monitoring Web and Mobile Application	108
Figure 5.2.1 Water Quality Monitoring System Architecture	121
Figure 5.2.2 Atlas Scientific Industrial Dissolved Oxygen Probe connected to EZO™ Dissolved Oxygen Circuit ESP8266 microcontroller ("Industrial Dissolved Oxygen Probe," n.d.).	122
Figure 5.2.3 The EZO™ Dissolved Oxygen Circuit	123
Figure 5.2.4 Industrial pH/ORP/Temp Probe + EZO™ RTD Temperature Circuit + EZO™ pH Circuit ("Industrial pH/ORP/Temp Probe," n.d.)	124
Figure 5.2.5 EZO™ RTD Temperature Circuit ("EZO™ RTD Temperature Circuit," n.d.)	125

Figure 5.2.6 EZO™ pH Circuit ("EZO™ pH Circuit," n.d.)	125
Figure 5.2.7 ESP8266 microcontroller	126
Figure 5.2.8 Arduino IDE	126
Figure 5.2.9 ThingSpeak Website	127
Figure 5.2.10 Sample Channel	128
Figure 5.2.11 API Keys of a Channel	129
Figure 5.2.12 Channel Settings	130
Figure 5.2.13 Public View of a Channel	130
Figure 5.2.14 Laravel, a PHP framework (Otwell, n.d.)	131
Figure 5.2.15 Python	133
Figure 5.2.16 WampServer (Bourdon, n.d.)	134
Figure 5.2.17 React Native, a JavaScript framework	136
Figure 5.2.18 Android Studio	137
Figure 6.2.1 Register Mathworks account	139
Figure 6.2.2 My Channels page	140
Figure 6.2.3 Create new channel	140
Figure 6.2.4 new channel page	141
Figure 6.2.5 Field 1 line graph	141
Figure 6.2.6 Add new gauge	142
Figure 6.2.7 RTD and pH Gauge Options	142
Figure 6.2.8 DO, RTD and pH gauges	142
Figure 6.2.9 Data Acquisition Module components connection	144
Figure 6.2.10 ESP8266 pins	144
Figure 6.2.11 EZO circuit pins	145
Figure 6.2.12 DO probe central wires and outer wires	145

8)	Figure 6.2.13 Male SMAs of Industrial pH/ORP/Temp Probe	146
	Figure 6.2.14 ThingSpeak channel's line graphs	152
	Figure 6.2.15 Wampserver version and its components	153
	Figure 6.2.16 Laravel Back-end project source code	154
	Figure 6.2.17 composer.json	155
	Figure 6.2.18 Create new database	156
	Figure 6.2.19 Database Migration Files	157
	Figure 6.2.20 php artisan server	157
	Figure 6.2.21 node -v and npm -v	158
	Figure 6.2.22 java -version	159
	Figure 6.2.23 SDK Manager	159
	Figure 6.2.24 build.gradle	160
	Figure 6.2.25 react-native -v	160
	Figure 6.2.26 React Native Front End Project	161
	Figure 6.2.27 dependencies in package.json	161
	Figure 6.2.28 run Metro	162
	Figure 6.2.29 npm run start	162
	Figure 6.2.30 emulator is running	162
	Figure 6.3.1 Success Register page	164
	Figure 6.3.2 Registration Successful	164
	Figure 6.3.3 Redirect to Login Page	165
	Figure 6.3.4 Fail Register page	166
	Figure 6.3.5 Registration Fail	166
	Figure 6.3.6 React Native registration form	167

Figure 6.3.7 SubmitButtonClick	168
Figure 6.3.8 route::api/register - POST	169
Figure 6.3.9 RegisteredUserController store function	169
Figure 6.3.10 Login Screen	170
Figure 6.3.11 Channel Page	171
Figure 6.3.12 Login Page with wrong credentials	171
Figure 6.3.13 Email not found	172
Figure 6.3.14 Password does not match	172
Figure 6.3.15 Login Form	173
Figure 6.3.16 LoginButtonClick	174
Figure 6.3.17 route::api/login – POST	174
Figure 6.3.18 AuthenticatedSessionController store function	175
Figure 6.3.19 Channel Page	176
Figure 6.3.20 Add New Channel	177
Figure 6.3.21 Add Channel Success	177
Figure 6.3.22 Channel ID not found	178
Figure 6.3.23 Channel Page with a channel	179
Figure 6.3.24 Channel Options	179
Figure 6.3.25 Delete Channel	180
Figure 6.3.26 Channel Deleted	180
Figure 6.3.27 submitForm	181
Figure 6.3.28 route::api/channels - POST	181
Figure 6.3.29 ChannelsController store function	182
Figure 6.3.30 deleteChannelById	184
Figure 6.3.31 route::api/channels/{channelId} – DELETE	185

Figure 6.3.32 ChannelsController deleteChannelById function	185
Figure 6.3.33 User click on a channel	186
Figure 6.3.34 Dashboard gauges page	187
Figure 6.3.35 Dashboard gauges page scroll down	187
Figure 6.3.36 Line Graphs Dashboard	188
Figure 6.3.37 Line Graphs Dashboard scroll down	188
Figure 6.3.38 Calendar page analysis section	189
Figure 6.3.39 Calendar Tab record actions section	189
Figure 6.3.40 Calendar Tab activity list section	190
Figure 6.3.41 Channel Page toggleChannelDetails function	190
Figure 6.3.42 Dashboard Page ComponentDidMount function	191
Figure 6.3.43 Dashboard Page fetchData function	191
Figure 6.3.44 fetchActionData	193
Figure 6.3.45 API routes for displaying activity analysis	193
Figure 6.3.46 calculateDurationForCurrentMonth	195
Figure 6.3.47 Change Water Analysis	196
Figure 6.3.48 Dashboard Page Gauges Webview	197
Figure 6.3.49 fetchUserGaugeSettings	197
Figure 6.3.50 route::api/waterparams POST	198
Figure 6.3.51 WaterParamsController byUserId function	198
Figure 6.3.52 Dashboard Page Line Graphs Webview	199
Figure 6.3.53 Record Actions Calendar	199
Figure 6.3.54 Add Event Dialog Box	200
Figure 6.3.55 Select Time	200
Figure 6.3.56 Select Activity	201

Figure 6.3.57 fill in description	201
Figure 6.3.58 Dashboard Page Calendar component	202
Figure 6.3.59 AddEvent dialog box	203
Figure 6.3.60 submitForm in AddEvent components	203
Figure 6.3.61 Dashboard Page handleEventSubmit function	204
Figure 6.3.62 readActions function	205
Figure 6.3.63 route::api/actions/channel/{channel_id} – GET	205
Figure 6.3.64 showByChannel function	206
Figure 6.3.65 Activity List	206
Figure 6.3.66 Dashboard page	207
Figure 6.3.67 Analysis Page	207
Figure 6.3.68 Dashboard page toggleAnalysis function	209
Figure 6.3.69 Analysis Page componentDidMount function	210
Figure 6.3.70 fetchDataAnHour	211
Figure 6.3.71 countRowsWithNullValues	211
Figure 6.3.72 calculateMinMaxMedianMean	212
Figure 6.3.73 countExtremeValues	213
Figure 6.3.74 calculateHealthIndex	214
Figure 6.3.75 arimaPredict funtion	215
Figure 6.3.76 auth::api/predict - POST	215
Figure 6.3.77 PredictionController predict funtion	215
Figure 6.3.78 arima_predict.py	216
Figure 6.3.79 TrendlineChart component	217
Figure 6.3.80 TrendlineChart component	218
Figure 6.3.81 Tab Navigator	219

Figure 6.3.82 Profile page	220
Figure 6.3.83 Notification	220
Figure 6.3.84 profile page	221
Figure 6.3.85 startBackgroundService	221
Figure 6.3.86 BackgroundTaskModule.startBackgroundService	222
Figure 6.3.87 stopBackgroundService	222
Figure 6.3.88 BackgroundTaskModule.stopBackgroundService	222
Figure 6.4.1 Data Acquisition system circuit deployed	223
Figure 6.4.2 Junction Box protects the Data Acquisition Module	224
Figure 6.4.3 DO, RTD and PH sensors deployed	224
Figure 6.4.4 ThingSpeak Channel Public View	226
Figure 6.4.5 Laravel back-end server	227
Figure 6.4.6 Metro running	228
Figure 6.4.7 Build successful	229
Figure 6.4.8 Android Emulator is running	229
Figure 7.2.1 UNIT-101	234
Figure 7.2.2 UNIT-102	235
Figure 7.2.3 UNIT-201	237
Figure 7.2.4 UNIT-202	238
Figure 7.2.5 UNIT-203	239
Figure 7.2.6 UNIT-301	242
Figure 7.2.7 UNIT-302	243
Figure 7.2.8 UNIT-303	244
Figure 7.2.9 UNIT-304	245
Figure 7.2.10 UNIT-401	248

Figure 7.2.11 UNIT-402	249
Figure 7.2.12 UNIT-403	250
Figure 7.2.13 UNIT-404	251
Figure 7.2.14 UNIT-501	254
Figure 7.2.15 UNIT-502	255
Figure 7.2.16 UNIT-601	257
Figure 7.3.1 PFT-101 Register an account (1967ms)	258
Figure 7.3.2 PFT-102 Register an account (13069KB)	259
Figure 7.3.3 PFT-201 Login Process(1826ms)	259
Figure 7.3.4 PFT-102 Register an account (13249KB)	260
Figure 7.3.5 PFT-301 Load Dashboard Process(4089ms)	260
Figure 7.3.6 PFT-302 Load Dashboard Process (13658KB)	261
Figure 7.3.7 PFT-401 Add Eevent Process(2339ms)	262
Figure 7.3.8 PFT-402 Add Event Process (13505KB)	262
Figure 7.3.9 PFT-501 Analysis Process(6851ms)	263
Figure 7.3.10 PFT-502 Analysis Process (13492KB)	264

LIST OF SYMBOLS / ABBREVIATIONS

API	-	Application Programming Interfaces
CLI	-	Command Line Interface
CSS	-	Cascading Style Sheet
HTML	-	Hypertext Markup Language
HTTP	-	Hypertext Transfer Protocol
JSON	-	JavaScript Object Notation
URL	-	Uniform Resource Locator
WAMP	-	Windows, Apache, MySQL, and PHP server
WBS	-	Work Breakdown Structure
pH	-	Potential of Hydrogen
DO	-	Dissolved Oxygen
RTD	-	Resistance Temperature Detector
RAS	-	Recirculating Aquaculture System
ppm	-	Parts Per Million
Degree Celsius	-	Degrees Celsius
LAN	-	Local Area Network
ISAS	-	IoT-based Smart Aquaculture System with Automatic Aerating and Water Quality Monitoring
IAMS	-	IoT-based Intensive Aquaculture Monitoring System with Automatic Water Correction
MQTT	-	Message Queuing Telemetry Transport
NaHCO ₃	-	Sodium Bicarbonate
QA	-	Quality Assurance
UAT	-	User Acceptance Testing
IoT	-	Internet of Things
IDE	-	Integrated Development Environment
CLI	-	Command Line Interface
SMA	-	SubMiniature version A (commonly used in RF connectors)
DC	-	Direct Current (or Data Center, depending on the context)

LIST OF APPENDICES

Appendix A: Template of User Satisfactory Survey	280
Appendix B: Usability Test Responses	281

CHAPTER 1

INTRODUCTION

1.1 General Introduction

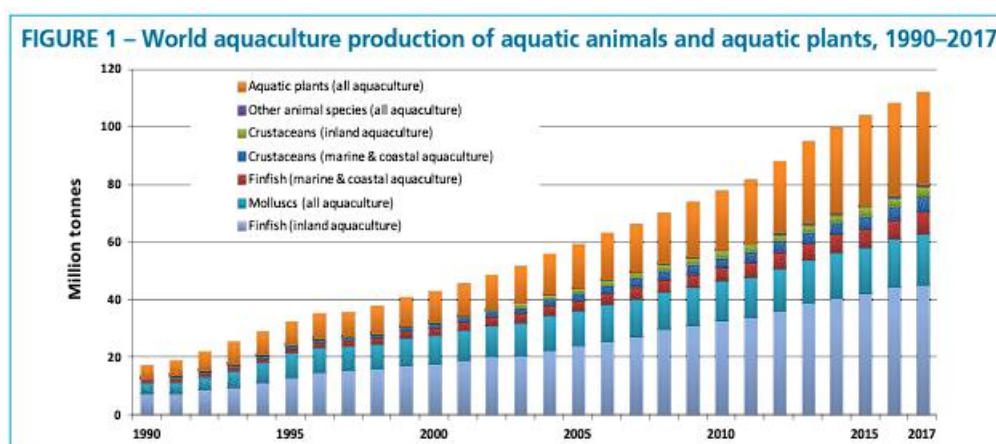


Figure 1.1.1 World aquaculture production of aquatic animals and aquatic plants, 1990–2017 (Zhou, 2019)

A large amount of the world's seafood supply is produced by the growingly significant business known as aquaculture. According to Figure 1.1, world aquaculture productivity has increased significantly from 1990 until 2017 (Zhou, 2019, p.6). It can be seen that aquaculture is becoming the primary source of aquatic products for humans. In order to meet the expectations of the aquaculture business, aquaculture equipment must also improve along with technical advancements (Zhou, 2019, p.6).

Maintaining ideal water quality conditions for fish growth and health is a crucial element in the success of aquaculture operations. Temperature, dissolved oxygen, pH, and ammonia levels are just a few of the water quality factors that can significantly affect fish growth and overall production.

Aquaculture water quality monitoring has traditionally been done manually with a variety of tools and methods. These techniques, however, are frequently labor- and time-intensive, and they might not offer up-to-the-minute

data on water quality conditions. The use of sensors to track water quality in aquaculture operations has gained popularity in recent years.

In order to optimise fish growth performance and monitor water quality in aquaculture operations, this project will focus on researching and investigating the application of IoT in this aquaculture operation. The project will concentrate on developing an aquaculture monitoring system that uses sensors to gather water quality data and a mobile application that helps users track these important water quality indicators in a fish farming environment, including temperature, dissolved oxygen, pH, and ammonia levels.

The outcomes of this project will aid in the creation of water quality monitoring systems for aquaculture operations that are more effective and efficient. The project has the potential to significantly impact the aquaculture industry and help satisfy the rising demand for sustainable seafood production by optimising water quality conditions to improve fish growth performance.

1.2 Problem Statement

Urban aquaculture faces many difficulties due to the lack of suitable methods and equipment to monitor the water quality of aquaculture. This will undoubtedly result in the fact that aquafarmers are always too late to take the appropriate measures when there is a problem with the water quality, which will cause the fish in the tank to start dying off one by one and lead to severe damage and loss, such as fish kill events.

1.2.1 Aeration system failure

The aeration system is supposed to function continuously to provide sufficient oxygen for the fish to survive in the water tank (Masser et al., 1992). However, there is a possibility that aeration system failure can occur without notice by aquafarmers. Given that many tanks and recirculation systems are densely populated, an aeration system failure can result in fish mortality events within minutes in the water tank due to a lack of dissolved oxygen (D.O) (Purina Animal Nutrition, 2023). Consequently, a water monitoring system that notifies

the user when the level of dissolved oxygen in the water tank drops below a safe level is necessary to mitigate this risk.

1.2.2 Algae Bloom

Algae often benefit aquaculture by providing farmed fish with oxygen and a natural food source (Priyadarshani, Sahu, and Rath, 2012). Previous research by Fernando et al. (2015) has demonstrated that adding algae to fish feed (aquafeed) in small amounts (10% of the diet) has a positive impact on growth performance and feed utilization efficiency. However, excessive algae growth or the occurrence of algal blooms can result in low dissolved oxygen (DO), as noted by the Minnesota Pollution Control Agency (2009). Nutrients such as phosphorus and nitrogen contribute to algae growth, and there are various ways these elements can enter our water resources (Kenekar, A., 2020).

Dissolved oxygen is consumed during the decomposition and death of algae, which can deprive aquatic species like fish of sufficient oxygen, ultimately leading to fish kills. Notably, an algal bloom in the Mumbai coastal areas in 2018 caused large-scale fish mortality (Minnesota Pollution Control Agency, 2009). Therefore, equipping aquafarmers with sensors to monitor dissolved oxygen, nitrogen, and phosphorus levels enables them to predict and respond to algae bloom events promptly.

1.2.3 Unpredictable timing to change water

Changing water is a common practice for maintaining good water quality in fish tanks, ensuring the best environment for optimal fish growth performance (WebMD Editor Contributors, 2023). However, aquafarmers often face the challenge of determining the ideal timing for water exchanges. The composition of the water in the tank is constantly changing due to climatic and seasonal variations, as well as pond usage (The Fish Site, 2015). Aquafarmers may struggle to determine when water should be changed without conducting water testing regularly, which can be cumbersome. Consequently, many aquaculture farmers opt to change the water when it appears dirty, even if it might still be usable.

Even when the water appears clear, contamination can occur due to the presence of food particles and waste. Waste can convert into chemicals such as nitrate and phosphate, which can negatively affect factors like dissolved oxygen, temperature, and other indicators (Environmental Protection Agency, 2012). To address this issue, water monitoring systems can be installed in water tanks to continuously monitor water parameters and provide water quality forecasts. According to Li et al. (2022), historical data collected from these sensors can be used in Machine-Learning (ML) models to predict water quality.

By implementing water quality prediction systems, aquaculture systems can maintain stability, reducing the incidence of fish diseases caused by deteriorating water quality. Armed with accurate forecasts, aquafarmers can make informed decisions about when to replace the water in the tank.

1.3 Aim and Objectives

The project aims to help aquaculture farmers monitor and optimize the water quality in their tanks to maintain a good fish farming environment, thereby improving fish growth performance, increasing the efficiency of the water monitoring process and producing high-quality fish products.

Objectives:

1. To analyze existing available tools related to aquafarming to develop ideas for designing usable water monitoring systems.
2. To perform analysis on the of water quality of the water tank to discover the trends and useful information to assist in decision making on the water quality monitoring process.
3. To develop a mobile application that provides a dashboard that displays all the water parameter readings of the aquaculture system anytime anywhere to allow user to check the current water condition in water tank.
4. To evaluate the water quality of the fish tank so that user are able to take immediate initiative to maintain its water quality to optimize fish growth and prevent undesirable condition.

1.4 Proposed Solution

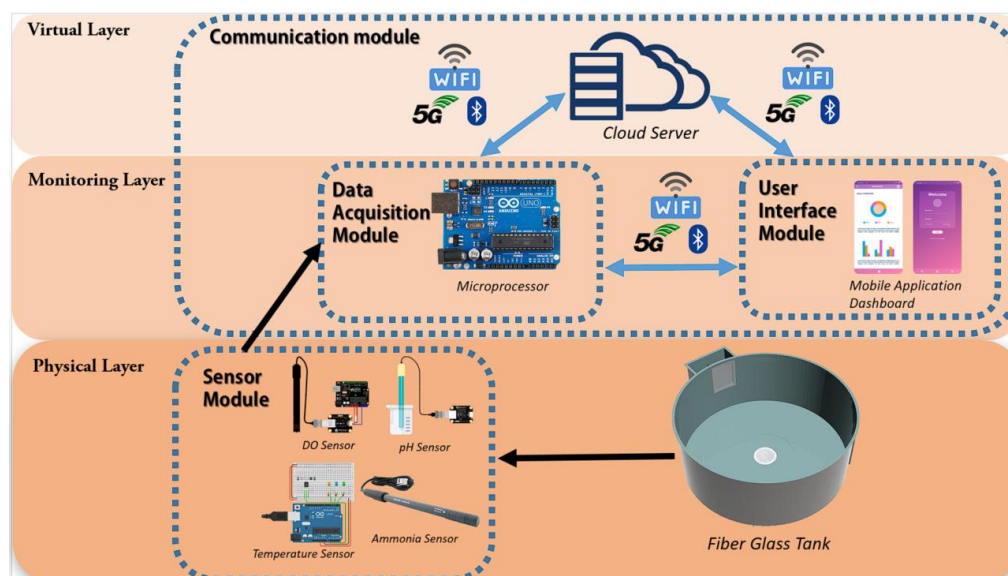


Figure 1.4.1 Proposed system architecture

The proposed solution to address the problem outlined in the problem statement involves the installation of an Internet of Things (IoT) water monitoring system in an aquafarming fish tank. This system integrates physical sensors, microprocessors, cloud servers, and a mobile-based Android application dashboard. Figure 1.2 illustrates the architecture.

In the Physical Layer, sensors are placed inside the fiberglass tank to measure water parameters such as Dissolved Oxygen (DO), pH level, temperature, and ammonia levels. These sensors collect data, which is then transmitted to the microprocessor in the Monitoring Layer.

At the Monitoring Layer, the microprocessor plays a crucial role. It converts analog signals from the sensors into digital data if the sensor lacks data logging functionality. In cases where the sensor itself can log data, the microprocessor serves as a data storage and management unit, utilizing SD cards as a backup for collected data.

The primary function of the microprocessor is to manage the Communication Module, facilitating communication between sensors and other devices. It collects data from all sensors and transmits this data to a Cloud Server

for storage and management. Cloud servers are chosen for their ability to securely store large quantities of data, allowing aquafarmers to monitor water quality from anywhere in the world with an internet connection. Long-distance water quality monitoring is achieved through the development of a Mobile Application, which connects to the Cloud Server database over the internet. This application organizes the data into a user-friendly dashboard that is easily comprehensible for aquafarmers. The framework used for developing this native Android Mobile Application is React Native.

In addition to data storage, Cloud Servers provide data processing and analysis capabilities, leveraging powerful algorithms and machine learning techniques to assist aquafarmers in monitoring their aquaculture operations. The prediction results are displayed within the Mobile Application, enabling aquafarmers to make informed decisions. Communication protocols within the system may include WiFi, cellular communication (such as 5G), and Bluetooth.

1.5 Proposed Approach

In this project, Evolutionary Prototyping has been selected as the development approach for the Water Monitoring Application. This choice is grounded in the advantages that make it particularly suitable for this project.

Firstly, Evolutionary Prototyping recommends segmenting the project into modules, enabling comprehensive testing of each module individually. This approach significantly reduces errors within each module of the project.

Additionally, this approach places a strong emphasis on meeting user requirements. It achieves this by incorporating prototypes into iterative cycles of feedback and refinement based on user demands. This ensures that the project's design aligns with user needs, as users actively participate in the system's design after reviewing each prototype. The use of prototypes ensures that the final dashboard design is user-friendly and easy for fish farmers to use when monitoring water quality.

Furthermore, Evolutionary Prototyping helps identify missing functionality in the product. As this project aims to deliver a dashboard for monitoring water quality on fish farms, the omission of any vital information could lead to critical issues like fish mortality incidents. Therefore, it is crucial to employ this approach to ensure that all necessary information is included in the dashboard, enabling users to make informed decisions when monitoring water quality (Martinez, 2021).

1.6 Project Scope

This project involves researching existing Aquaculture monitoring systems and developing the concept of integrating hardware components and software components, including sensors, microprocessors, and Cloud Servers. The ultimate goal is to create a mobile-based water monitoring application that offers a comprehensive solution for monitoring water parameters in fish tanks, assisting aquaculturists in their work.

The project consists of several key modules that need to be completed:

1. Sensor module
2. Data acquisition module
3. Communication module
4. User interface module

Each of these modules plays a crucial role in the overall functionality of the water monitoring system.

1.6.1 Sensor Module

The Sensor module is responsible for collecting and logging all measurements of water quality attributes in the fish tank, such as Dissolved Oxygen (DO), pH, temperature, ammonia, nitrite, and nitrate

1.6.2 Data Acquisition Module

The Data acquisition module includes a microprocessor that is responsible for data processing. This includes converting raw sensor readings into meaningful units and performing data smoothing.

1.6.3 Communication Module

The Communication module is essential for transmitting data from the data acquisition module to the user interface module and for sharing data over the network with the Cloud Server, which performs Machine Learning to provide valuable insights from the gathered data. This module uses protocols such as Wi-Fi, Bluetooth, or 5G for data transmission.

1.6.4 User Interface Module

The User interface module is essential for visualizing the current water quality of the fish tank. This module involves an Android-based mobile application that converts the received data from the Communication Module into a user-friendly dashboard. This dashboard displays all the necessary information for the water monitoring process. Additionally, the module alerts aquaculturists when any water quality attribute in the tank exceeds or falls below safe limits. It also provides water quality forecasts to assist users in taking corrective action and making informed decisions.

CHAPTER 2

LITERATURE REVIEW

2.1 Aquaculture

Aquaculture is the process of cultivating aquatic organisms in all types of water environments (Global Seafood Alliance, 2019). It shares a similar concept with agriculture, where humans grow crops and raise animals on land to produce resources like food, wool, and other useful products. The primary distinction lies in aquaculture, which involves breeding, rearing, and harvesting aquatic organisms such as fish, shellfish, and algae (NOAA, 2023). One of the primary benefits of aquaculture is its contribution to human consumption, particularly seafood. As the global population continues to grow, the demand for seafood is on the rise. According to the 'Fish To 2030: Prospects for Fisheries and Aquaculture: World Bank Report Number 83177-GLB,' predictive models indicate that by 2030, aquaculture is expected to produce 62 percent of food fish. Beyond 2030, aquaculture is likely to dominate the global fish supply (The World Bank, 2012). This underscores the importance of the mainstream development of aquaculture technology to remain competitive in the market.

To ensure the success of aquaculture, continuous monitoring of water quality is imperative (Lang, 2019). Poor water quality can threaten the health and growth of fish, making it essential for aquaculturists to pay close attention to water quality parameters such as temperature, suspended solids, photosynthesis, dissolved oxygen levels, carbon dioxide, nitrogen, ammonia, and pH values (Towers, 2015). According to Edinburgh Sensor (2022), there are 4 keys of water quality factors that need to be monitored, if not, it will affect the aquaculture operations. They are:

1. Physical parameters: pH, temperature, salinity, dissolved oxygen, and carbon dioxide levels.
2. Organic contaminants
3. Biochemical hazards: e.g. cyanotoxins
4. Biological contaminants: e.g. pathogens

Effective management of these parameters within safe limits is critical to prevent adverse conditions, such as stress or disease, that could impact the overall health and performance of aquaculture.

2.2 Recirculating Aquaculture System

According to Yue et al. (2022), the Recirculating Aquaculture System (RAS) is one of the modern aquaculture systems that has emerged due to disruptive technologies. It is an almost completely closed-circuit, tank-based aquaculture system designed for the controlled monitoring of farmed fish. RAS offers several advantages for fish farms, including the reduction of water changes, prevention of harmful bacterial invasions that may lead to diseases in fish, improved growth performance, and higher yields.

As Joseph et al. (2019) explain, the RAS operates by raising fish at high densities within the tank, where environmental conditions are meticulously controlled. This control is achieved through the system's ability to recycle water within the tank for filtration and cleaning purposes, reducing the need for frequent water replacements. Additionally, the RAS includes various subsystems for removing waste products from the fish farm, such as solid waste, ammonia, and carbon dioxide, or converting them into non-toxic forms. Furthermore, the purified water in the system is oxygenated to ensure it is not deficient in oxygen. This is achieved through aeration systems or the use of liquid oxygen before the water is returned to the tank.

The recirculation of water in the RAS serves multiple purposes, including reducing the rate of water replacement, maintaining optimal water quality conditions, and addressing water supply shortages.

2.3 Internet of Things in Aquaculture

The exponential growth of the Internet of Things (IoT) has significantly enhanced the operations of the aquaculture industry. While monitoring water quality is paramount in aquaculture operations, its complexity has always posed challenges for fish farmers (Dupont et al., 2018). Therefore, by integrating IoT into aquaculture, this issue can be addressed through the development of a smart,

affordable, reliable, and efficient automated water quality monitoring system, empowering aquaculture farmers to enhance their performance. According to Yue et al. (2022), implementing IoT technology in the aquaculture sector offers several advantages:

1. Using cameras and sensors in aquaculture farms allows real-time monitoring of environmental conditions.
2. Timely and continuous real-time monitoring of the effects of fish farms on the ecosystem enables improved environmental management, particularly applicable to offshore aquaculture systems.
3. Combining IoT with machine learning and data collected over time enables the creation of predictive models, enhancing decision-making and allowing for timely risk alerts.

In aquaculture, sensors are responsible for collecting data on water physical parameters, such as dissolved oxygen (DO) levels, temperature, pH values, and salinity. Fluctuations in these parameters outside their safe ranges can significantly impact fish health. To achieve IoT in aquaculture, inexpensive sensors are deployed to simultaneously monitor multiple parameters through wireless sensor networks (WSN). WSN consists of numerous automated sensors in aquafarms that measure, gather, transmit, and process water quality data in real-time. The collected data is then displayed on a computer or sent to farmers as messages for real-time updates. This real-time water quality monitoring capability simplifies data collection, reducing human errors and time delays, thereby improving the quantity and quality of collected data (Su et al., 2020).

Many studies have demonstrated that the primary reason for using WSN in aquaculture is real-time measurement of crucial physical parameters and immediate notification of relevant personnel in case of problems, ensuring prompt resolution. Espinosa-Faller and Rendón-Rodríguez (2012) describe a WSN-based water monitoring system that collects and sends data to a database for storage. In case of issues, SMS or email alerts are sent to responsible individuals. Zhang et al. (2011) propose another implementation using software for real-time water quality monitoring in fish farms, triggering SMS alerts to

users in the event of serious issues. Their software promotes scalability and reusability through separate logic, display, and data layers. Additionally, Huang et al. (2013) present a water monitoring system with a real-time interface displaying data numerically and graphically.

Integrating the Internet of Things into a recirculating aquaculture system allows sensors to send collected data to a microprocessor. When any measured parameter falls outside the desired range, the microprocessor triggers the corresponding controller in the RAS to execute the necessary resolution (Abinaya et al., 2019). According to Tsai et al. (2022), this research suggests an IoT-based Smart Aquaculture System that automatically activates aeration systems and feeders based on fuzzy processing results. If the DO level falls below the safety range, the fuzzy inference process assesses water parameters such as water temperature, pH, and hardness. If the fuzzy inference process concludes that corrective action is required, it signals the Raspberry Pi computer to activate the aerator and feeder.

In conclusion, IoT-based Smart Aquaculture Systems have revolutionized traditional aquaculture operations by enabling real-time water quality monitoring without the need for manual intervention. This allows for remote monitoring of aquafarms, automatic water quality adjustments, predictions of potential water quality issues, and informed decision-making.

2.4 Water Quality

2.4.1 Importance of Water Quality

For an aquaculture process to be successful, the maintenance of water quality is the primary task to ensure the growth performance of aquatic products. As stated in The Fish Site, (2015), water quality directly impacts the health of fish, their behavior, and growth performance. Poor water quality can result in reduced growth efficiency, erratic behavior, symptoms of disease or parasitic infestation, and, in the worst-case scenario, fish kill events. To sustain ideal water quality management, aquafarmers must have a thorough understanding of the optimal ranges for water parameters that allow fish to thrive, grow, and reproduce. They should also comprehend the relationships between different water parameters, the factors that lead to poor water quality, and the methods for maintaining good water quality.

2.4.2 Water Quality Parameters

There are several water quality parameters need to monitor, the major parameters include:

1) **Temperature:**

Temperature is a measure of the heat present in the water. Its standard measure unit is Celcius (°C) or Fahrenheit (°F).

When temperature increase, the activity level and the metabolism of aquaculture organisms also increase, and also boost the fish growth rate. If the temperature too high and exceed the physical and nutritional tolerance for too long, the fishes may get contaminated with bacteria, lose their balance when swimming, and could die from exhaustion. If the temperature too low, the feed intake and metabolism decrease and causing poor growth performance. Especially in aquafarm, temperature affects more on the aquatics organisms because higher biomass and less water volume.

2) Dissolved oxygen:

Dissolved oxygen is the amount of oxygen dissolved in water. Its standard measure unit is milligrams per liter (mg/L) or parts per million (ppm).

Dissolved Oxygen is one of the critical water parameters, aquatic organism used dissolved oxygen for respiration and accommodate metabolism. Oxygen affects the solubility and availability of many nutrients. Therefore, low level of dissolved oxygen in water can cause fish death and also increase in the poisonous metabolites.

3) pH level:

pH is a measure of the acidity or alkalinity of water. It is measured by using the scale from 0 to 14. When pH level of the water exceed pH 9, means the water is too alkaline. This situation cause the conversion of ammonium to become toxic ammonia which can kill the fish. While water is too acidic with pH lower than 5 can filter out metals from rocks and sediments. The metals affects the fish health and can be fatal. Fish kills usually occurs when the pH of water is below 4.5 or greater than 10.

4) Salinity

Salinity is the concentration of salt in water. It is measured in parts per thousand (ppt) or practical salinity units (psu). Salinity plays an important role in aquafarming. This is because it directly affects the osmoregulation of aquatic organisms. Osmoregulation is to balance the water and ions in their body fluids to maintain internal stability. Excessive salinity level cause aquatics to lose water, while insufficient salinity level cause too much water in aquatics body fluid. Any one of this situation will affects the growth rate, metabolic rate, food intake, food conversion and hormonal stimulation.

5) Turbidity

Turbidity is a measure of water clarity. It is measured in Nephelometric Turbidity Units (NTU). Turbidity can affect the light penetration in water and thus limit photosynthesis carried out for plants in water. High turbidity causes temperature and DO stratification in water.

6) Ammonia, NH_3

Is dissolved metabolic organics in water. It is measured in milligrams per liter (mg/L) or parts per million (ppm). High level of ammonia can lead to stress and death in aquatic organisms. The cause of increase in ammonia is overfeeding. The decay of rich-protein feed will release toxic ammonia gas.

7) Nitrate, NO_3

It is measured in milligrams per liter (mg/L) or parts per million (ppm). High level nitrate leads to algal blooms and then cause low Dissolved Oxygen in water and hence lead to fish death.

8) Nitrite, NO_2

This is a toxic compound produced from oxidation of ammonia in water. It is measured in milligrams per liter (mg/L) or parts per million (ppm). Based on Ciji and Akhtar (2020), excessive nitrite disrupt the oxygen uptake in the blood of aquatic organisms, cause damage to the fish gills and destroy ionic and water balance and finally lead to fish death. Besides, high level of nitrite also reduce the reproductive performance of fish.

Table 2.2.4.1 Standard range for water quality parameter set by fisheries research institute (FRI) Malaysia (Azlan Othman et al., 2020)

Parameter	Standard Ranges	Unit
Temperature	28 – 32	°C
DO	> 4	ppm
pH	6.5 – 8.5	
Salinity	24 - 32	ppt
Ammonia (NH ₃ -N)	0.1 – 0.5	ppm
Nitrate (NO ₃)	0 – 10	ppm
Nitrite (NO ₂)	< 0.3	ppm

This standard shows the tolerance range that ensure the products produced are safe for human consumption. However, different types of fish have their own specific favourable ranges for each water parameter to achieve its optimum growth performance (The Fish Site, 2015). For example, Table 2.2 also shows that different species can tolerate different range of the water quality to optimize their growth (FRESHWATER-AQUACULTURE, 2019).

Table 2.2.4.2 Water quality tolerance by species (FRESHWATER-AQUACULTURE, 2019)

Species	Temp °F	Dissolved Oxygen mg/L	pH	Alkalinity mg/L	Ammonia %	Nitrite mg/L
Baitfish	60-75	4-10	6-8	50-250	0-0.03	0-0.6
Catfish/Carp	65-80	3-10	6-8	50-250	0-0.03	0-0.6
Hybrid Striped Bass	70-85	4-10	6-8	50-250	0-0.03	0-0.6
Perch/Walleye	50-65	5-10	6-8	50-250	0-0.03	0-0.6
Salmon/Trout	45-68	5-12	6-8	50-250	0-0.03	0-0.6
Tilapia	75-94	3-10	6-8	50-250	0-0.03	0-0.6
Tropical Ornamentals	68-84	4-10	6-8	50-250	0-0.03	0-0.5

2.5 Existing Aquaculture Monitoring System

2.5.1 Automated water quality monitoring system development via LabVIEW for aquaculture industry (Tilapia) in Malaysia

According to Azlan Othman et al. (2020), it propose a automated water quality monitoring system for the tilapia industry using LabVIEW software. This system focus efficient data logging and analysis to help aquafarmers to monitor the water quality of aquaculture real-time continuously. Besides, this system has the alarm system to alert user when any fluctuation of the water parameters. This system currently does not include any automated water quality correcting system. However, it is upgradable due to its flexibility in accepting more types of sensors to the system and also can integrate with other water quality correcting system when any monitored parameter falls away from the safety range.

2.5.1.1 System Design and Architecture

Below figure 2.1 shows the block diagram of water quality monitoring system for the tilapia industry.

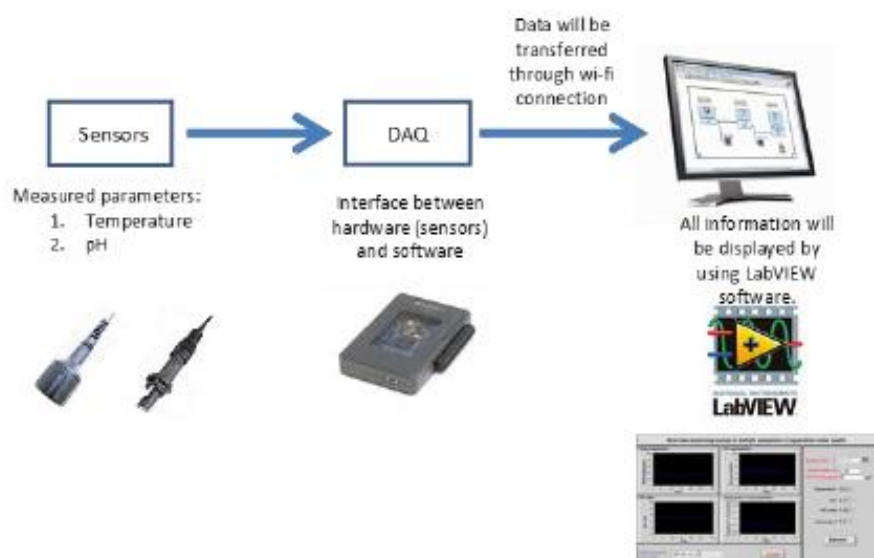


Figure 2.5.1 The block diagram of water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)

Based on figure 1, the system consist of 3 main hardware components such as sensors, microcontroller, and workstation/computer that contains LabVIEW software. There are 2 sensors used in this system, temperature and pH sensors. They are connected to NI myRIO-1900, a microcontroller that act as Data Acquisition System(DAQ). The microcontroller also connected to the workstation, a computer that has the LabVIEW software.

2.5.1.1.1 Water Paramter Sensors

There are 2 types of sensors used in this system: Temperature and pH sensors. Figure 2.2 shows the sensors connection setup. These 2 sensors should be connect to the microcontroller to send the sensor data.

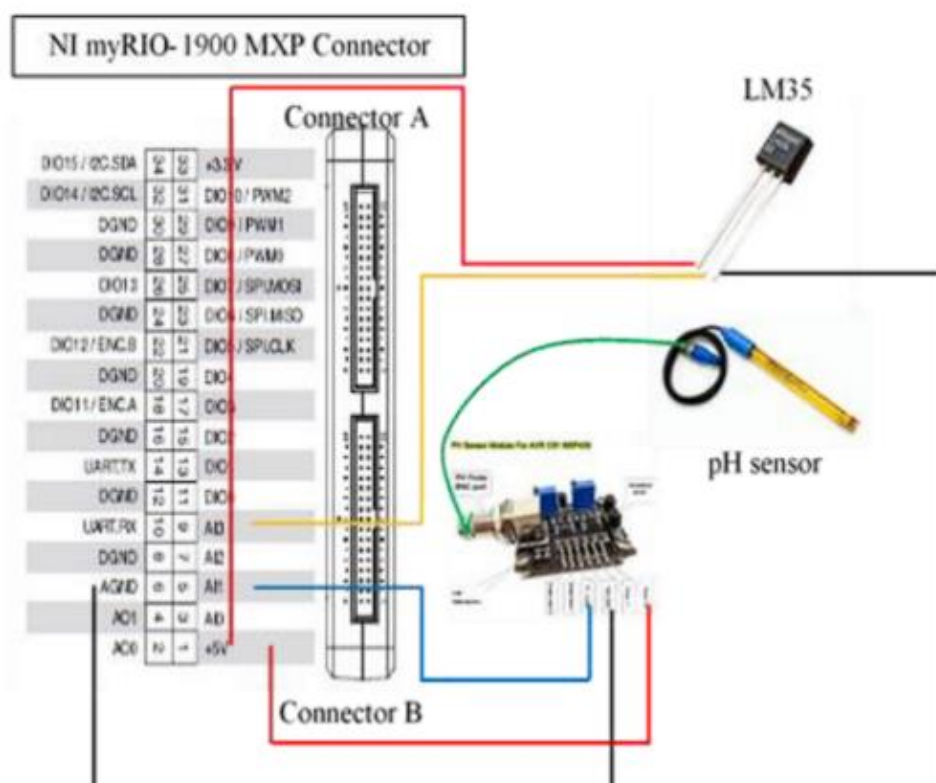


Figure 2.5.2 The block diagram of water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)

1) Temperature:

The sensor used is a modified temperature sensor, LM35 used to measure the temperature of the aquafarm. LM35 connect to the analog input pin 9 on connector B of microcontroller(NI myRIO-1900).

2) pH level:

pH sensor used is Atlas scientific pH sensor that measure pH level. The pH sensor is connected to pH probe BNC connector of the pH circuit. Then the pH output terminal from the pH circuit is connected to analog input 5 connector A of microcontroller, NI myRIO-1900. The pH circuit will acts as signal conditioning, which means pH circuit will modify the signal output from pH sensor to become accurate and reliable signal for microcontroller to read.

2.5.1.1.2 Microcontroller

The only microcontroller used in this system is NI myRIO-1900. NI myRIO-1900 is a compact embedded device that has analog input (AI), analog output (AO), digital input and output (DIO), audio, and energy output. It act as data acquisition system (DAQ) which is an interface that connected between sensors and monitoring workstation for them to communicate between each other. The data collected by the sensors must pass through DAQ, the microcontroller, NI myRIO-1900, before reaching to workstation. The NI myRIO-1900 can connected to a host/computer/workstation using USB or 802.11b.g.n wireless networking standard that allows local area network (LAN) communication. Due to its built-in wifi feature, wi-fi connection is the protocol used to send sensor data from microcontroller to monitoring workstation for LabVIEW software to perform Data Visualization and Analysis. Figure 2.3 shows the layout water quality monitoring system for the tilapia industry.

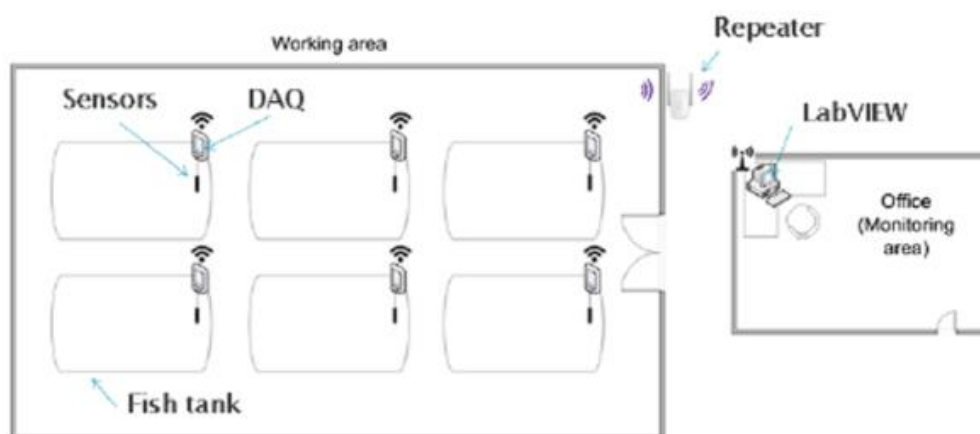


Figure 2.5.3 The layout water quality monitoring system for the tilapia industry (Azlan Othman et al., 2020)

2.5.1.1.3 Workstation (LabVIEW)

Monitoring Workstation, is a computer that contain LabVIEW Software. LabVIEW is a visual programming language use to create automated test systems that has variety of analysis features, interactive display elements, drivers for automating each instrument and data gathering hardware, links to other languages and industry-standard protocols (Anon, 2023). Therefore, labVIEW is used to program a LabVIEW software application for this system to detect the signal from the installed sensors to perform data visualization and data analysis to monitor the water quality. Figure 2.4 shows the front panel of the system via LabVIEW software.

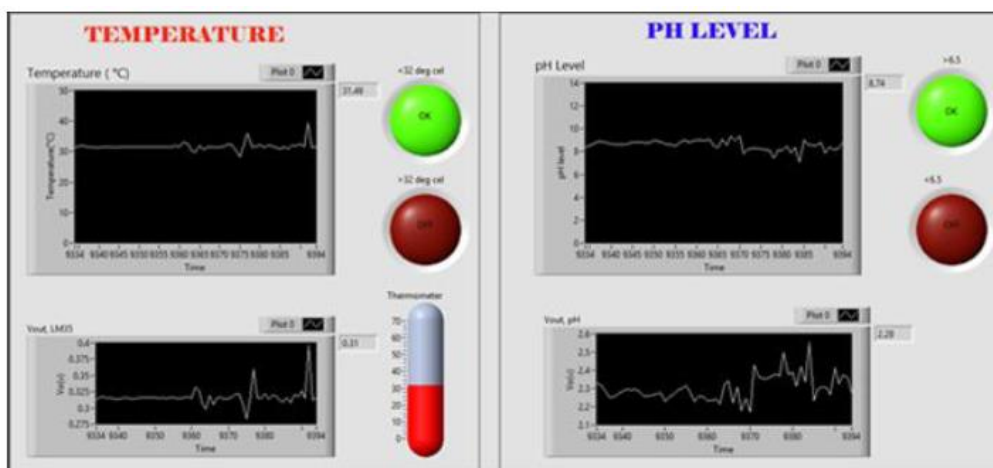


Figure 2.5.4 The front panel of the system via LabVIEW software (Azlan Othman et al., 2020)

This LabVIEW front panel consist of the aquafarm current water quality information retrieved from the sensors installed, temperature and ph level. Both information have the line graph showing the status level of the water quality over time, for example: temperature over time and ph level over time. Next, below of the interface also provide the line graph of output voltage of each sensor over a period of time. The two circle in each water quality interface, green circle represent the safety range for each water quality. Oppositely, red circle represent the dangerous range for the water quality. It tells user if the water quality is safe, the circle should be green and is out of range will be red. For temperature page, there also show a thermometer measuring current temperature of the aquafarm. The LabVIEW software page for water quality monitoring is easy to understand.

2.5.1.2 Experiment for implementation this system

Experiment Setup

The experiment objective is to find out whether using this system to measure water parameters is usable. Therefore, this experiment should:

- 1) test whether this system will alert user if any water parameter is in dangerous status,
- 2) determine the accuracy of automated system by calculating the error percentage in reading the water quality parameters in the

aquafarm. Figure 5 shows the formula to calculate the error percentage is:

$$\% \text{ error} = \left| \frac{\text{Value}_{\text{automated}} - \text{Value}_{\text{manual}}}{\text{Value}_{\text{manual}}} \right| \times 100$$

Equation 2.1 Formula to calculate error percentage of this system

3) allow aquafarmer perform analysis based on the data collected.

Two different set-ups of the system is place at the same fish tank, automated system and manual system. Figure 2.5 shows the manual set-up versus automated set-up.



Figure 2.5.5 Manual set-up versus automated set-up (Azlan Othman et al., 2020)

Right image is the Automated system, which is the proposed system using sensor and LabVIEW software, while left side is the Manual system that require aquafarmer to use portable sensor to measure the water quality.

Experiment Result

1) Objective:

Test whether this system will alert user if any water parameter is in dangerous status.

Results:

Figure 2.6 shows the front panel of temperature when the temperature exceeds 32°C.

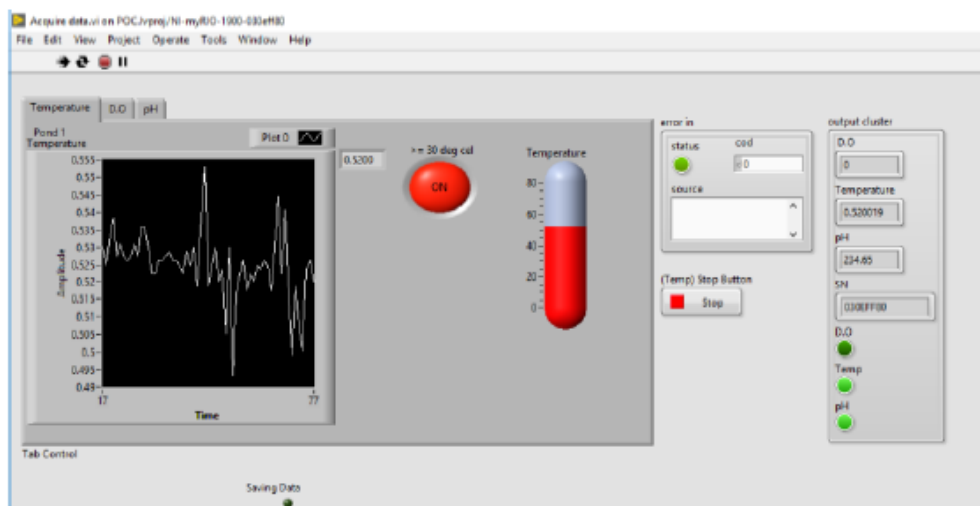


Figure 2.5.6 The front panel shows when the temperature exceeds 32 °C

(Azlan Othman et al., 2020)

By using the formula:

$$\text{Temperature } (^{\circ}\text{C}) = \text{Vout} \times 100$$

System can get the temperature value, when the temperature retrieved that time is 32°C or above. The circle LED will turn into red colour, and with the word “CN” indicates “CAUTION” to alert user that the temperature is under unbehaviorable condition. Therefore, it tells aquafarmers have to take steps to decrease the water temperature.

2) Objective:

To determine the accuracy of automated system by calculating the error percentage in reading the water quality parameters in the aquafarm.

Results:

Table 2.3 shows the accuracy of automated system comparing to the data gathered in manual set-up.

Table 2.5.1 The accuracy fo temperature sensor of the system (Azlan Othman et al., 2020)

No of data taken	Automated set-up (°C)	Manual set-up (°C)	Percentage of error (%)
1st	49.56	49.50	0.12
2nd	47.36	45.30	4.55
3rd	41.26	39.80	3.67
4th	36.50	34.20	6.73
5th	32.23	30.60	5.33
6th	29.17	27.70	5.31

This table shows that the average of the percentage error of automated system compared to the manual measurement is less than 7% for the temperature parameter. This proved that the developed automated system is able to measure these water parameters correctly as manual system with very low error percentage. Therefore, this automated monitoring and data collection system is suitable to be developed for aquaculture industries to monitor aquaculture.

3) Objective:

To allow aquafarmer perform analysis based on the data collected.

Results:

Table 2.4 shows the temperature and pH level collected in 2 days.

Table 2.5.2 Table of temperature and pH level with time (Azlan Othman et al., 2020)

Date/Time	Temperature(°C)	pH level
28/5/2019 8:00	31.86	6.47
28/5/2019 11:20	31.01	6.70
28/5/2019 14:40	31.74	6.29
28/5/2019 18:00	31.74	6.24
28/5/2019 21:20	31.74	6.34
29/5/2019 0:40	31.74	6.31
29/5/2019 4:00	31.62	6.63
29/5/2019 7:20	31.37	6.49
29/5/2019 10:40	30.52	6.39
29/5/2019 14:00	30.27	6.38
29/5/2019 17:20	29.91	6.29
29/5/2019 20:40	30.52	6.37

The relationship between temperature and pH level can be analysed out through the table. Based on the table, aquafarmers come out with the conclusion that at the highest temperature, the temperature is lower and vice versa. This is supported with the evidence that during 28 May 2019, 11.20am, when the temperature is at 31°C, the pH level become higher until 6.702. While on 28 May 2019, 6pm, when the temperature is increased to 31.738°C, the pH level decrease to

6.242. Unfortunately this condition only occur on the first day due to the water sample properties started become acidic on the next day.

2.5.2 IoT-based Smart Aquaculture System with Automatic Aerating and Water Quality Monitoring

According to Tsai et al. (2022), it shows an IoT-based smart aquaculture system (ISAS) to help aquafarmer to monitor various water quality parameters. The proposed ISAS are equipped with sensors and also automated water quality control devices to monitor water quality to build a suitable aquaculture environment in aquafarm to increase fish growth performance. This ISAS system make use of fuzzy inference process for quick automatic operation of aerators and feeders to ensure the water parameters is within the safety range. Users can make use of ISAS to monitor water quality in aquafarm easily at anywhere and anytime using mobile devices and remote computers as long as the device have the internet connection due to implementation of cloud database in this system. In addition, users also could receive warning messages when the water parameters falls below or exceed the safety range.

2.5.2.1 System Design and Architecture

Below figure 2.7 shows the architecture of the ISAS.

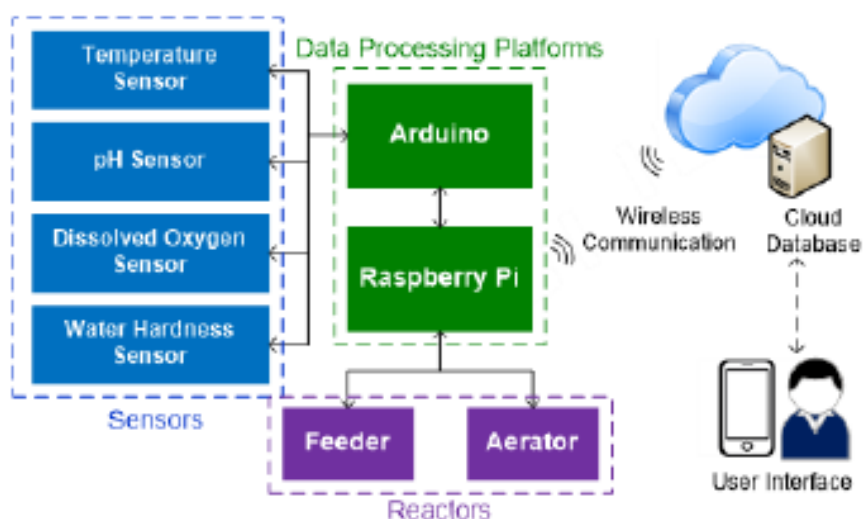


Figure 2.5.7 The architecture of ISAS (Tsai et al., 2022)

Based on figure 2.7, the system is made up of 4 main components. They are sensors, data processing platforms, reactors and cloud database that provides data to user interface. The sensors that used in ISAS is, Temperature sensor, pH sensor, Dissolved oxygen sensor and Water Hardness sensor. These sensor will be connected to data processing platform consist of Arduino microcontroller,

and Raspberry Pi computer. Then Raspberry Pi of Data Processing Platforms will be connected to Cloud database to store the sensors data, and also connected to reactors to perform automatic water parameters control in water tank. The communication between Cloud database and Raspberry Pi computer is through wireless connection. User can view water parameters data and receive alert message through user interface that display the data from the Cloud database (Tsai et al., 2022).

2.5.2.1.1 Water Paramter Sensors

There are 4 types of sensors used in this system: Temperature sensor, pH sensor, dissolved oxygen sensor and water hardness sensor. Figure 2.8 shows the 4 sensors and data collection platforms used in this system.

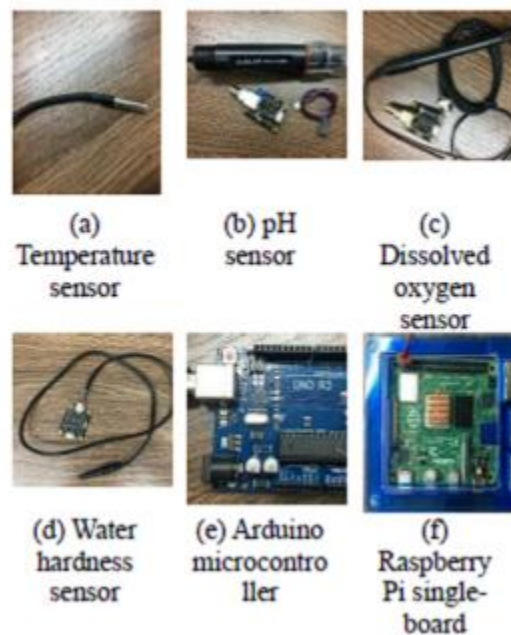


Figure 2.5.8 The sensors and data collection platforms of the ISAS (Tsai et al., 2022)

1) Temperature:

DS18B20 digital thermometer (Figure 2.8(a)) is used as the temperature sensor in ISAS. It can measure temperatures from -55°C to $+125^{\circ}\text{C}$ and provides programmable resolution of 9-12 bits. Higher bit of resolution means more accurate measurement, however need to processing power and storage.

2) pH level:

DFROBOT SEN0169 analog pH meter (Figure 2.8(b)) is used to measure pH level in this ISAS. This pH sensor has the ability of fast response and good thermal stability, which is suitable for long-period monitoring.

3) Dissolved Oxygen (DO) level:

A DFROBOT SEN0237 sensor (Figure 2.8(c)) is used to detect dissolved oxygen level. It has detection range of 0–20 mg/L.

4) Water hardness:

A DFROBOT SEN0244 analog total dissolved solid sensor (Figure 2.8(d)) is used to measure water hardness in ISAS. It has the measurement range of 0 – 1000 mg/L.

2.5.2.1.2 Microcontroller

The microcontroller used in this system is Arduino Uno microcontroller (Figure 1.8(e)). The task of microcontroller here is to combine all the data provide by each sensors together, then sends these water quality parameters data to Raspberry Pi for processing.

2.5.2.1.3 Computer (Raspberry Pi)

The Raspberry PI, is a single-board computer consist of ARM Cortex A72 processor and 8gb of RAM with multiple communication interfaces such as WiFi, Bluetooth, USB and Mini HDMI. The Raspberry Pi task in ISAS is to store the last 7 days' sensors data into its internal memory, and all sensors data should send by Raspberry Pi to cloud database using WiFi and the MQTT protocol. MQTT protocol is known as Message Queing Telemetry Transport protocol, is a lightweight messaging protocol for IoT. It is Ideal for connecting remote devices with a small code footprint and minimal network bandwidth (MQTT, 2022).

Besides, Raspberry Pi computer contain the fuzzy inference process, which perform assessment on water temperature, pH and hardness then send

fuzzy output to Raspberry Pi computer to control the water control system. Figure 2.9 shows the flow diagram of automatic control flow of the ISAS.

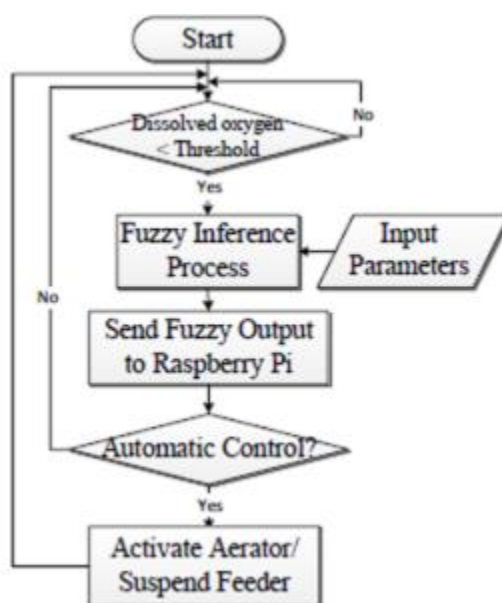


Figure 2.5.9 The automatic control flow of the ISAS (Tsai et al., 2022)

The sensors collect the parameters data and send to Raspberry Pi, then Fuzzy Inference Process perform assessment. If the assessment result shows that water quality is poor and not suitable for aquaculture, then it will send fuzzy output to Raspberry Pi to activates each water quality control devices such as aerator to maintain the dissolved oxygen level at certain range, and suspend feeder to feed the fish to avoid further contamination of the water.

This ISAS system use fuzzy-based control instead of machine learning which may produce better decision making in automatically activate the water control devices to maintain water quality. The main reason is fuzzy-based control can provide fast response that ML cannot serve, and also it is easier to apply than ML because ML cannot be done in Raspberry Pi and Arduino due to their limitation of computing power. ML based-systems require long training time and different aquatics require different design. Thus, Fuzzy inference process is used in this case.

2.5.2.1.4 Cloud platform

Cloud platform contain database that are used to store all history of sensors data. The advantages of using Cloud platform to store data is due to its larger capacity which is able to store all the history of the water quality parameters. The main reason is that the data stored in Cloud platform can be access by smartphone, computer and web application for data visualization and data analysis in user interface as long as there is internet connection. Besides, these data can also be used in ML in cloud platform to provide useful information for aquafarmers (Tsai et al., 2022).

2.5.2.1.5 Application of ISAS system

There are 2 types of application can be used to monitor water quality in aquafarm based on the ISAS system such as web application, and smartphone application. Aquafarmers can view the aquafarm's water quality condition through these application. Both types of application serves the same functionality, they retrieve data from cloud database and perform data visualization to provide user-friendly user interface to display the data in user readable format. Figure 2.10 shows the line graph of temperature sensed data through web application (Tsai et al., 2022).



Figure 2.5.10 The temperature sensed data example (Tsai et al., 2022)

The temperature line graph allows user to track the change of temperature over time. Figure 2.11 shows the line graph of dissolved oxygen sensed data through web application

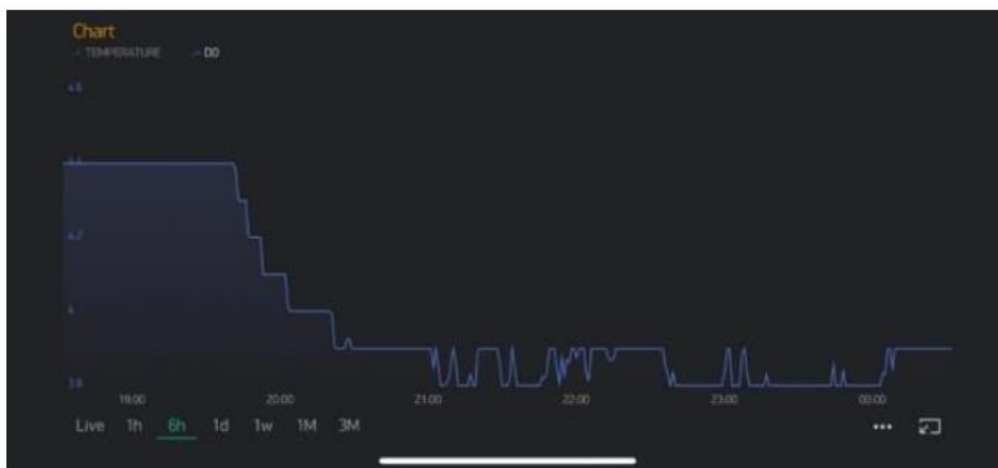


Figure 2.5.11 Sensed data dashboard from user's mobile device (Tsai et al., 2022)

The dissolved oxygen line graph also allows aquafarmers to track the change of dissolved oxygen level within a period.

Moreover, both of the application also provide dashboard for real-time numerical data of current water quality condition as shown in figure 2.12.

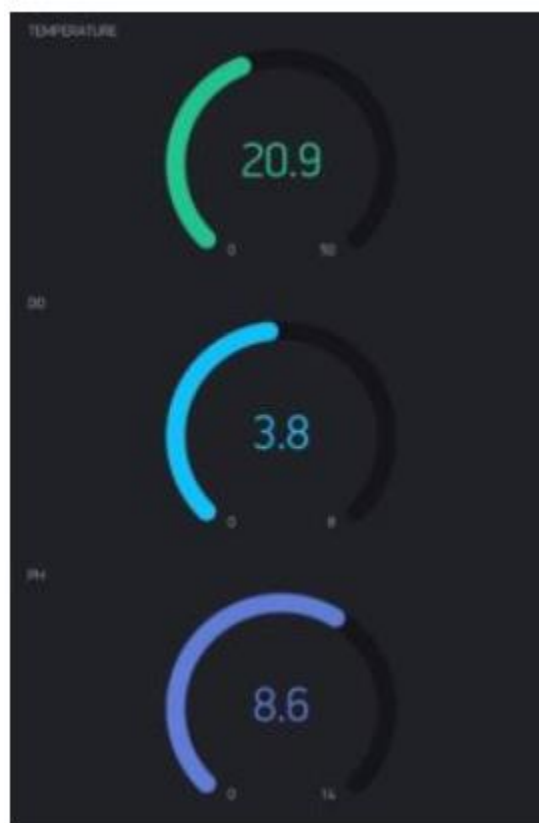


Figure 2.5.12 Sensed data dashboard from user's mobile device (Tsai et al., 2022)

The dashboard shows the current temperature is 20.9°C, DO is 3.8mg/L, and pH is 8.6. Different colours indicates different types of water parameter monitored.

2.5.2.1.6 Water Quality Control Devices

This ISAS system also include 2 devices to control the water parameters to maintain in healthy state, they are aerators, and feeder. Their responsibility is to receive signal from Raspberry Pi that perform Fuzzy Inference Process, in order to decide whether to activate aerators to increase DO level in water, and also suspend feeder to stop feeding (Tsai et al., 2022).

2.5.2.2 Experiment for implementation of ISAS

Experiment Setup

The experiment objective is to verify the feasibility of ISAS. Therefore, this experiment should:

- 1) find out does ISAS system increase the survival rates of the shrimp.

There are 2 different set-ups of aquarium were build, 1 experimental group which implement ISAS and another 1 control group without ISAS. Figure 2.13 shows the experiment setup.



Figure 2.5.13 Experimental group and control group of the ISAS (Tsai et al., 2022).

Based on Figure 2.13, two different set-ups of the system is place at different aquarium, left side is the experimental group with ISAS, and the right side is the control group. The experiment starts with 60 shrimps and some aquatic plant

into both groups. Then, control group is fed once a day daily, while experimental group were fed based on the fuzzy result decision. The experiment continues 1.5 months, the survival rates of the shrimp calculate using below formula:

$$\text{Survival rate} = (\text{Number of shrimp remaining} / \text{Number of shrimp at the beginning}) \times 100\%$$

Experiment Result

4) Objective:

To find out does ISAS system increase the survival rates of the shrimp.

Results:

Figure 2.14 shows the line graph of survival rates of shrimps of experimental and control group within 1.5 months.

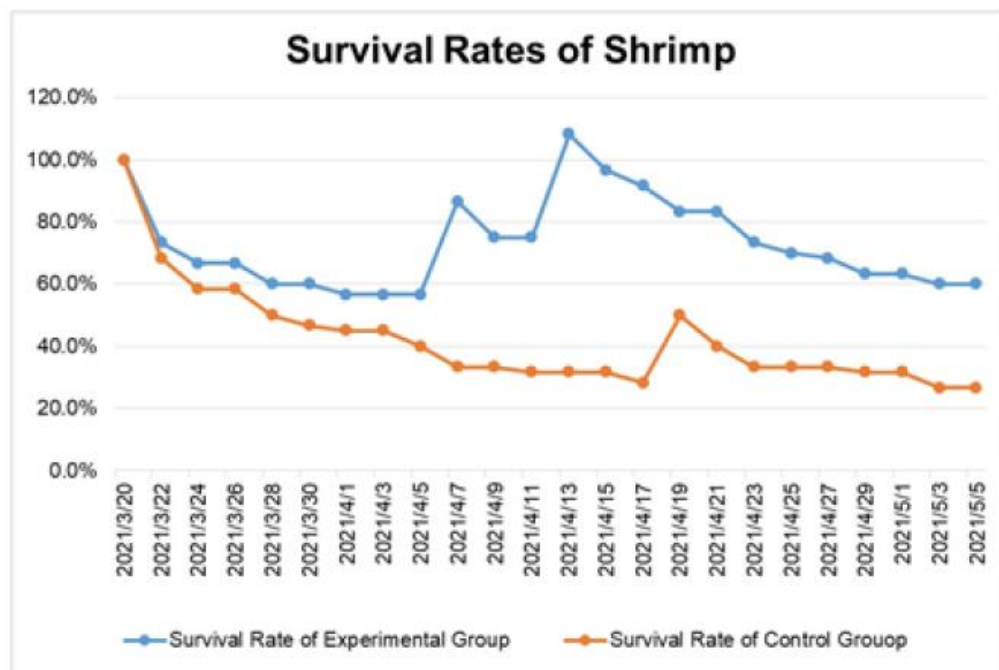


Figure 2.5.14 Survival rates of shrimps during the 1.5 months of our experiment (Tsai et al., 2022)

The line graph shows declining of both groups of shrimp during the first week. The survival rate of Experimental Group increase on 2021/4/7 and 2021/4/13 is due to new baby shrimp were born, while in Control group new baby shrimp were born on 2021/4/19. On the last day of the experiment 2021/5/5, the remaining shrimp in control group is 16, therefore its survival

rate is only 26.7%. However, the Experimental Group still have 36 number of shrimp survived, and it reach 60% of survival rate, which is 33.3% greater than Control Group. In conclusion, ISAS can increase the survival rate of aquatics. Therefore, ISAS is feasible to be used in aquaculture industry.

2.5.3 Development of an IoT-based Intensive Aquaculture Monitoring System with Automatic Water Correction

According to Tolentino et al. (2021), it propose an IoT-based Intensive Aquaculture Monitoring that monitor and automatically corrects the water quality parameters to maintain good water quality environment for aquatics to grow in aquafarm. Due to the IoT structure in this aquaculture system, an internet based application is then use to perform data visualization for user to monitor the aquaculture easily. The application should provide user a dashboard that shows precise real-time condition of each water parameters and also should provide the average length and weight of the fish to determine their growth (Tolentino et al., 2021).

2.5.3.1 System Design and Architecture

Below figure 2.15 shows the architecture of the IoT-based Intensive Aquaculture Monitoring System with Automatic Water Correction.

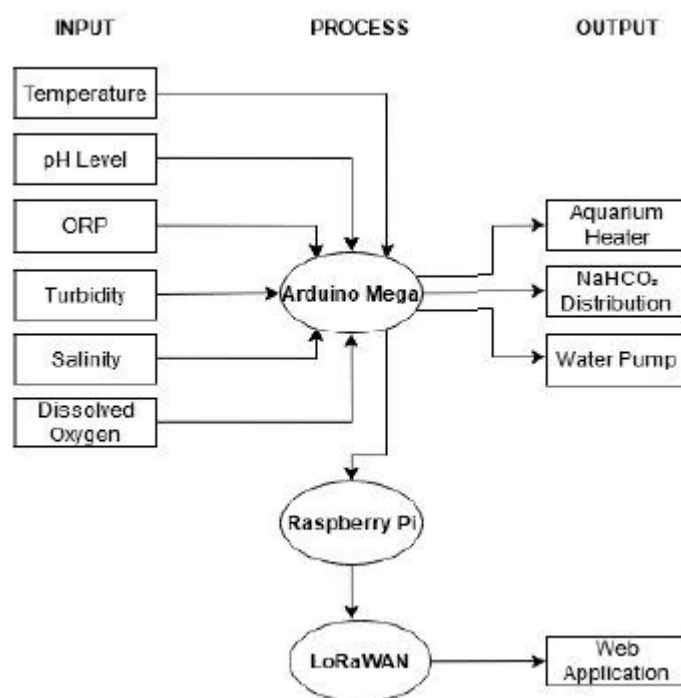


Figure 2.5.15 The architecture of ISAS (Tolentino et al., 2021)

Based on figure 2.15, the system is made up of 3 layers. They are input, process, and output layers. The input contains 6 types of sensors: temperature, pH level, oxidation-reduction potential, turbidity, salinity, and dissolved oxygen. These sensors from input are then connected to the Arduino Mega

microcontroller in Process layer. There are 2 more components in Process Layer: Raspberry Pi and LoRaWan IoT Protocol. The Raspberry Pi is a computer that use LoRaWan IoT Protocol to send data Cloud Platform and then Web Application at output layer will display these data to user. Besides, output layer also have those water quality correcting device such as Aquarium Heater, NaHCO₃ distribution, Water Pump.

2.5.3.1.1 Water Paramter Sensors

There are 6 types of sensors used in this system: temperature sensor, pH level sensor, oxidation-reduction potential sensor, turbidity sensor, salinity sensor and dissolved oxygen sensor.

1) Temperature:

The temperature sensor used is a Waterproof Temperature Sensor DS18B20 to measure temperature.

2) pH level:

A DFRobot Industrial Analog pH Sensor is used to measure pH level of water.

3) Oxidation-reduction potential:

A DFRobot ORP Analog Meter is used to measure ORP.

4) Turbidity:

To measure the turbidity of water, Gravity: Analog Turbidity Sensor is used.

5) Salinity:

To measure Salinity of water in water tank, Gravity: Analog Electric Conductivity Sensor is used in this system.

6) Dissolved oxygen:

The DFRobot Gravity: Analog Dissolved Oxygen Sensor is used to measure Dissolved Oxygen in water tank.

2.5.3.1.2 Microcontroller

The microcontroller used in this system is Arduino Mega microcontroller. The task Arduino Mega is to collect all sensors analog data for each connected water parameters sensors, integrate them and send these data to Raspberry Pi computer for data processing. Moreover, it also link with water quality correctors such as Aquarium Heater, NaHCO₃ Distribution and Water Pump. These correctors will be activated by Automatic Water Quality Correction program in Arduino Mega microcontroller to perform their tasks, when their related water parameters value does not meet the desired level. With these devices help, the water quality can be maintain within safety range (Tolentino et al., 2021).

2.5.3.1.3 Computer (Raspberry Pi)

The Raspberry PI, is a single-board computer is used in this system to act as a gateway to accepts data from Arduino Mega, and then use its processing power and networking capabilities to communicate with database through Long Range Wide Area Network (LoRaWan) IoT Protocol. Then Raspberry Pi can transmit these sensors data to Web Application to perform monitoring process also via LoRaWan IoT Protocol. A 868MHz LoRaWan is used because it can reduce production costs and supports long-distance communication. Therefore using LoRa modules for data transmission is suitable for monitoring aquaculture setups.

2.5.3.1.4 Web Application

This system use Web Application to let Aquafarmers to monitor the aquafarm. This Web Application named TeamLapia, is used to exploit all the data gathered for monitoring water quality of aquafarm. It is developed using PHP and JavaScript codes and link to database that stores all the latest and keep updating water parameters data. Therefore, this Web Application can display the recent status of each water parameters with accurate numerical values and fish growth such as number of fish, average length, and weight. Besides, this application also use Html, CSS and JavaScript to produce a user-friendly User Interface

using that aquafarmers can easily understand the current condition of the aquaculture. Then it also provides the graphs that shows the changing of the water quality parameters over time for user to make analysis. Figure 2.16 shows the TeamLapia Web Application Interface.

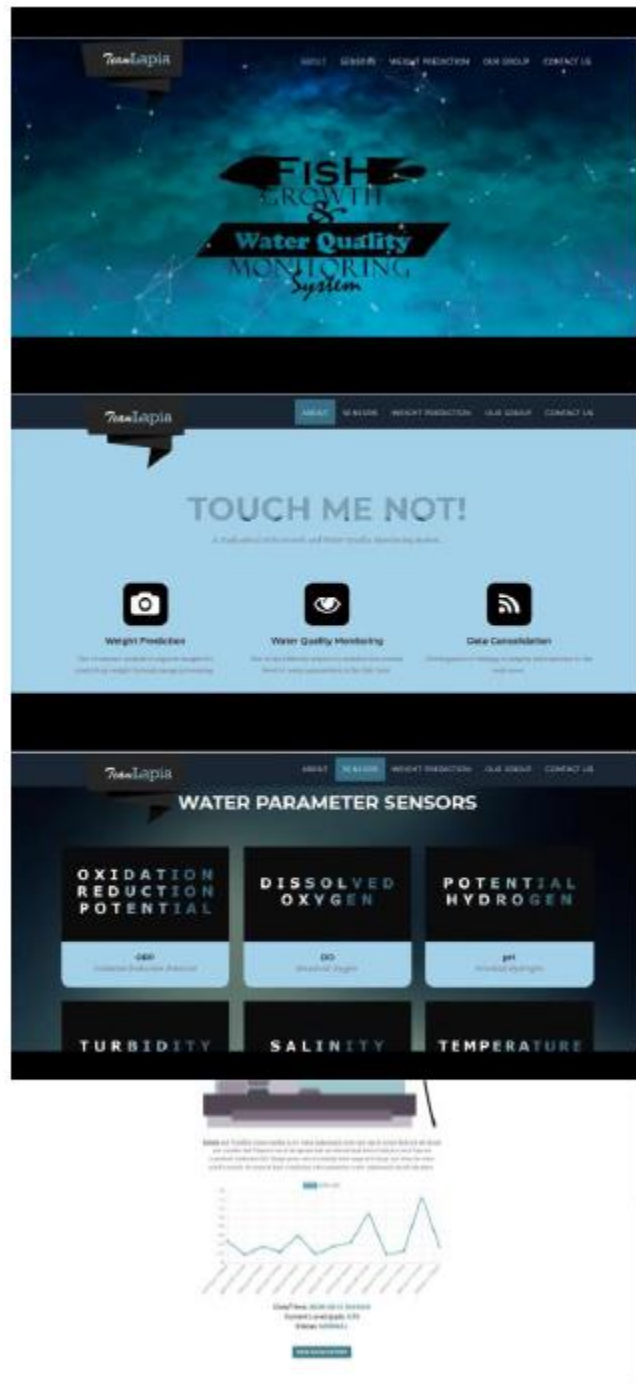


Figure 2.5.16 TeamLapia Web Application Interface (Tolentino et al., 2021)
The first page shows the header showing user that this is a “Fish Growth and Water Quality Monitoring System”. The second page shows the 3 main function of this system: Weight Prediction, Water Quality Monitoring, and Data

Consolidation. The third page is the Water Parameter Sensors page listed with all installed sensors in the aquafarm. When user click into specific water parameter they want to view, the fourth page will be displayed. This page first provide a brief description of this chosen parameter, then also provide line graph showing the values of the current water parameter on the past period of time. In addition, it also provide the current status of the parameter. Figure 2.17 also shows the TeamLapia Web Application Structure Diagram.

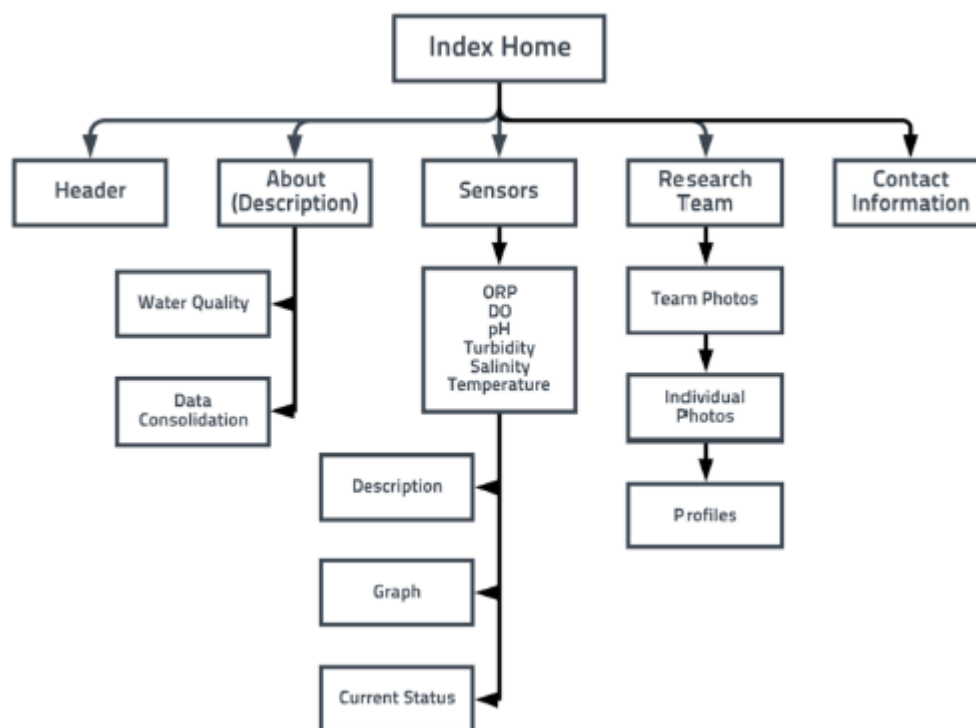


Figure 2.5.17 TeamLapia Web Application Structure Diagram (Tolentino et al., 2021)

2.5.3.1.5 Water Quality Control Devices

This system include 3 devices to control the water parameters, they are: Aquarium Heater, NaHCO₃ Distribution and Water Pump. They are activated by the Arduino Mega Micontroller when water quality does not meet the desired range.

1) Aquarium Heater

Aquarium Heater task is to keep the water temperature maintain at 28°C.

2) NaHCO₃ Distribution

NaHCO₃ is Sodium bicarbonate or baking soda, is a weak base that use to neutralize the water when the pH is dropping. Therefore, this device will be activate to release the Sodium Bicarbonate solution when pH level fall below the threshold value. When the pH level back to desired level, this device will be switch back off again.

3) Water Pump

Water Pump is activated when oxidation-reduction potential, turbidity, salinity and dissolved oxygen values are not within the ideal range optimal for fish growth. When the all the water parameters are back to desired level, this device will be switch back off again.

2.5.3.2 Experiment for implementation of IAMS

Experiment Setup

The experiment objective is to:

- 1) find out whether this system able to monitor the water parameters and automatically activate the correcting devices.
- 2) to assess the automated aquaculture system's effectiveness and reliability as well as the difference in fish growth rates between it and the traditional setup.

There are 2 different set-ups of systems were build: 1) First one is the proposed system setup that perform regular monitoring and correcting on the temperature, potential hydrogen (pH) level, oxidation-reduction potential, turbidity, salinity, and dissolved oxygen to build an environment to optimize the fish growth. 2) Second one is the conventional setup, the water will be check and change once a week. The experiment takes 12 days to conduct. The growth rate of the fish is determine by using the formula below:

$$\text{Growth rate} = (\text{Final weight} - \text{Initial weight}) / (\text{Initial weight}) \times 100\%$$



Figure 2.5.18 Experimental group and control group of the ISAS (Tolentino et al., 2021)

Based on Figure 2.18, two different set-ups of the system is place at different water tank with Nile Tilapia fishes, left side is the conventional setup, and the right side is the controlled setup. The Stereo-Vision Camera is used to measure the size of fish.

Experiment Result

1) Objective:

To find out whether this system is able to monitor the water parameters and automatically activate the correcting devices.

Results:

Figure 2.19 shows the line graph of pH level over days for the controlled aquaculture setup.

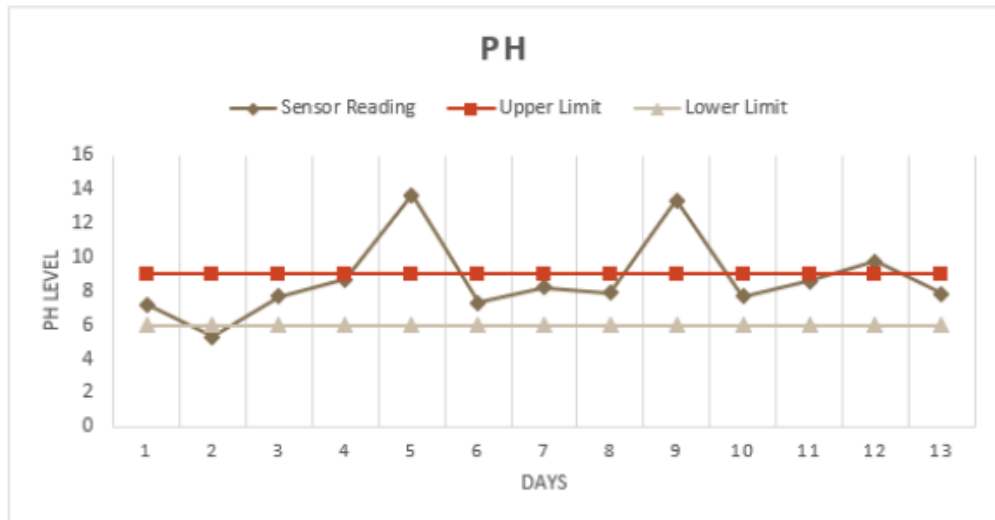


Figure 2.5.19 pH sensor Readings with Correction Response (Tolentino et al., 2021)

The desired pH level range for Nile Tilapia is between 6 to 9. It can be seen that the pH level of the water will automatically corrected back to desired range between 6-9 whenever the pH value exceed the upper limit. This is because the NaHCO_3 distribution is activated.

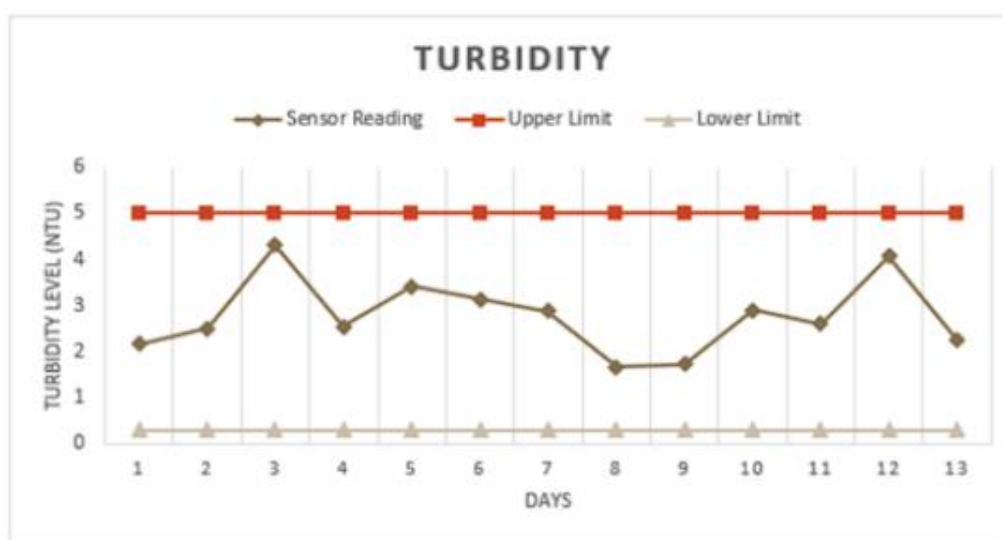


Figure 2.5.20 Turbidity Sensor Readings with Correction Response (Tolentino et al., 2021)

Figure 2.20 shows the line graph of turbidity sensor readings over days. The ideal range of Turbidity for Nile Tilapia is 0.3 to 5. The turbidity level always remain between the lower limit and upper limit.

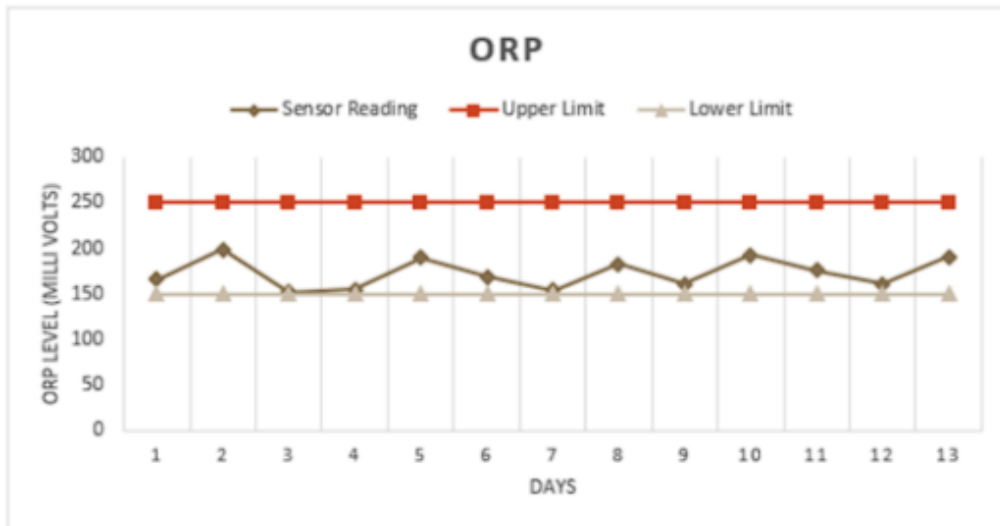


Figure 2.5.21 Oxidation Reduction Potential Sensor Readings with Correction Response (Tolentino et al., 2021)

Figure 2.21 shows the line graph of ORP level over days. The ideal range of ORP level of Nile Tilapia is 150mV to 250mV. It shows that whenever the ORP nearly fall below lower limit, it always increase back to nearly 200mV.

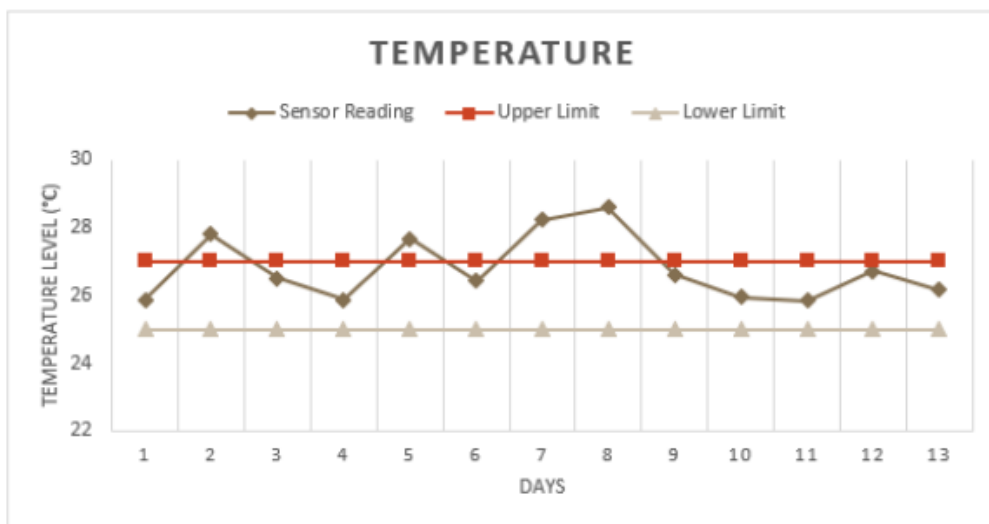


Figure 2.5.22 Temperature Sensor Readings with Correction Response (Tolentino et al., 2021)

Figure 2.22 shows the line graph of temperature readings over days. The desired range for temperature is between 25 to 27°C. The graph shows that the

temperature is always exceed the upper limit, however it is able to be corrected back using corrective device.

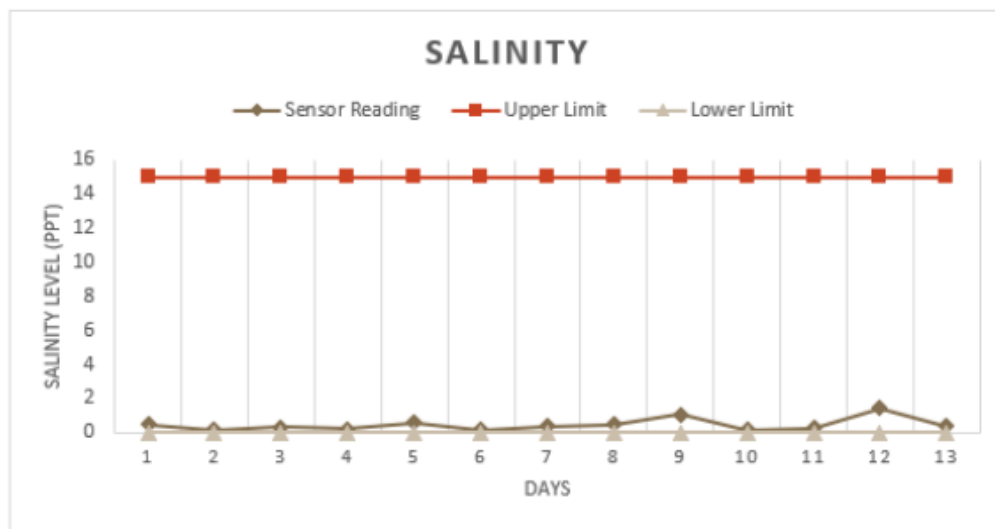


Figure 2.5.23 Salinity Sensor Readings with Correction Response (Tolentino et al., 2021)

Figure 2.23 shows the line graph of Salinity sensor readings over days. The desired range for Nile Tilapia to growth is below 5ppt. However, Nile Tilapia can tolerate with wide range of Salinity from 0 to 15ppt. The Salinity of the water tank in controlled setups shows that the Salinity level is always stay within desired range for Nile Tilapia to growth.

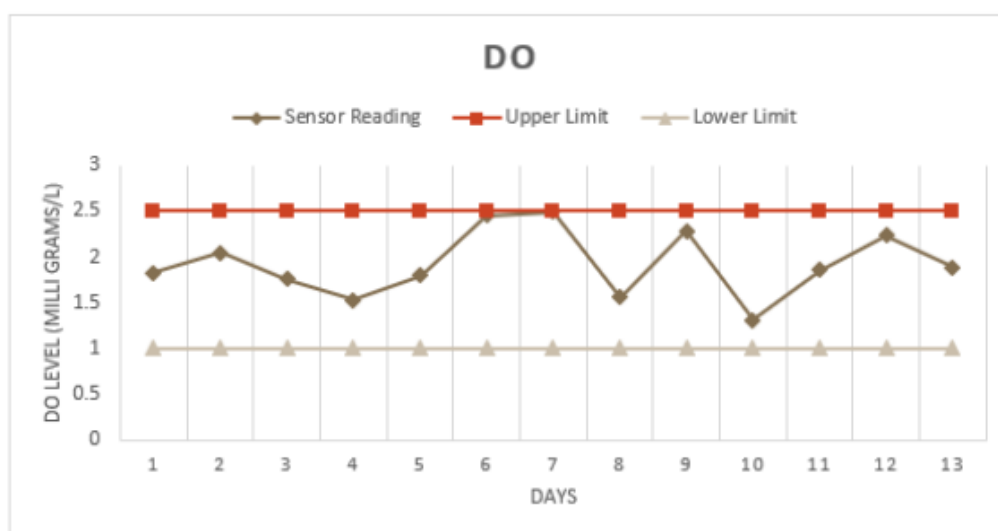


Figure 2.5.24 Dissolved Oxygen Sensor Readings with Correction Response (Tolentino et al., 2021)

Figure 25 shows the line graph of DO readings over 13 days. The desired range for Dissolved Oxygen is between 1 and 2.5mg/L. It can be seen that whenever the DO level nearly reach the lower limit, aerators will be triggered to increase the DO level. Therefore, the DO level is able to maintain within the desired range throughout this period.

In conclusion, this system is able to monitor the water parameters and also automatically activate the correcting devices. Therefore, the Nile Tilapia growth performance is assured with the help of this system to maintain the desired water quality.

2) Objective:

To determine the efficiency and reliability of the system and the difference of the growth rate of the fishes between the automated aquaculture system and the conventional setup.

Results:

Figure 2.25 shows the comparison bar graph of the the fish weights between Conventional Setup and Controlled Setup in 2 weeks.

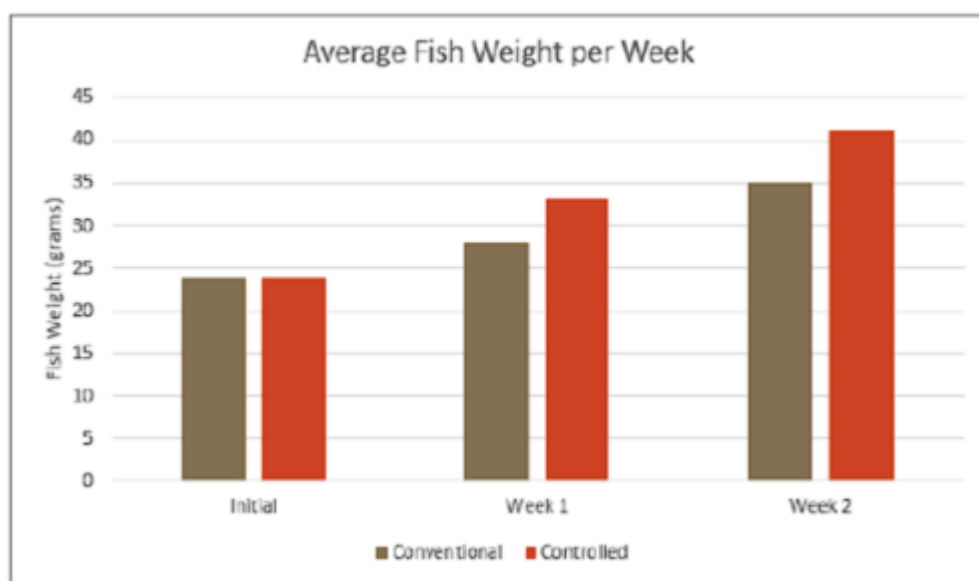


Figure 2.5.25 Controlled vs. Conventional Fish Growth measured every week
(Tolentino et al., 2021)

From this bar graph, the controlled setup have higher average fish weight than conventional setup in week 1 and week 2. This means that fishes in controlled setup water tank have greater growth performance. Table 2.5 shows the detail average fish weight of proposed aquaculture setup versus conventional setup in from day 0 to day 7 then day 12.

Table 2.5.3 Fish Growth In Each Setup (Tolentino et al., 2021)

Days Elapsed	Average Fish Weight (g)	
	Proposed Aquaculture Setup	Conventional Setup
0 (initial)	24	24
7 (week 1)	33	28
12 (week 2)	41	35

Based on Table 2.5, the average fish weight of the fishes in proposed aquaculture setup increased total 17g (from 24g to 41g) while in the conventional setup only increase 11g (from 24g to 35g). The calculated growth rate per week is 30.70% for proposed Aquaculture setup, and 20.76% for conventional setup. Therefore, this proves that the proposed aquaculture setup is efficient and reliable as it increase growth performance of fishes in conventional setup by 46.88%.

2.6 Comparison and Analysis

Table 2.6.1 Comparison table between LabVIEW, ISAS, IAMS.

Features	Systems		
	LabVIEW	ISAS	IAMS
Provide real time water quality monitoring (dashboard)	✓	✓	
Provide data visualization (graph)	✓	✓	✓
Provide water quality auto-correction system		✓	✓
Use computer software as monitoring interface	✓		
Use web application as monitoring interface		✓	✓

Use mobile application as monitoring interface		✓	
Implement cloud server as database		✓	
Implement local server as database			✓
Implement local database	✓		
Support long distance water quality monitoring		✓	✓
Provide fish growth prediction (length & weight)			✓
Upgradeable (Add more sensor and features)	✓	✓	✓
Alert user when water parameters fall out from desired range	✓	✓	

Advantages

By comparing these 3 types of aquaculture monitoring system architecture, IoT-based Smart Aquaculture System (ISAS) is the best system in terms of functionality.

First of all, ISAS provide the most important features which is support long distance water quality monitoring. This is because it used cloud server as database, therefore its mobile application and web application can use sensed data stored in cloud, and display them to user. Therefore, user are able to view the current water quality condition of the aquafarm at anywhere as long as having internet connection. LabVIEW does not support long distance monitoring as the data captured is stored in local database of its workstation.

Besides, ISAS also contain fuzzy inference process in its system to automatically activate water control device to correcting the water parameters to maintain within safety range. This reduce the workload of aquafarmers need to keep changing water and also can optimize the fish growth performance as the water quality is always maintain at aquatics favourable condition. Oppositely, LabVIEW does not contain water quality auto-correction system.

Next, ISAS also surpass the 2 other system with its real-time monitoring features showing the latest current condition of water parameters in a dashboard. The sensors keep updating the water quality 3 minutes as time interval, and keep the database is always the latest all the time. LabVIEW also provide real-time monitoring but its data are captured 10 minutes which its time interval is longer than ISAS, reduce time for aquafarm to take early corrective steps. But IAMS takes 1 day as the interval for reading sensors data, therefore it does not provide real-time water quality monitoring.

The last features that ISAS provide is automatically send warning message to the user to alert that the current quality of the water is poor. IAMS does not provide this features, but LabVIEW does. However, due to its sensor data collection time takes longer than ISAS. Therefore, the alert message also will be late to received by user compared to to ISAS. Thus, may cause late corrective action aquafarmers.

In conclusion, ISAS is the best water monitoring system among them, it provides many important and useful functionality that other system lack of. For example, long distance monitoring, automatically water parameters control, real-time water quality monitoring and warning messages to alert user about the poor condition of water in aquafarm.

Disadvantages

Although ISAS is the best among these system. However, it also have some weaknesses that it should be improved. Firstly, the dashboard of this system can be improved to include more information for user to have better monitoring experience in using this application. Figure 2.26 shows the ISAS dashboard.

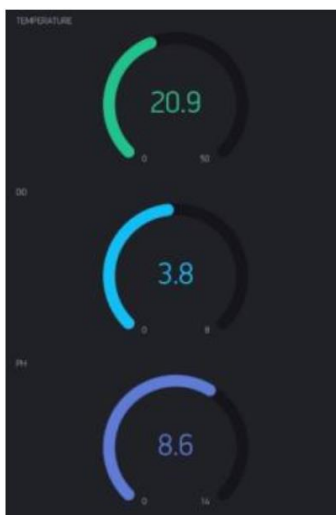


Figure 2.6.1 ISAS dashboard in mobile application (Tsai et al., 2022).

This dashboard does not show the desired range of the water parameters. Then when one of them is nearly exit the safety range, the gauge colour should turn into orange colour and red colour when exit the desired range. This can help user to easily focus on the dangerous water parameter. The design is not user-friendly as the indicator of the gauge is small, difficult to find out which gauge belongs to which water parameters. Therefore, here should increase the text size and also put some icon and standard unit suitable to represent their corresponding water parameters.

Besides, figure 2.27 also shows the line graph chart displayed using web application. However, the journal does not provide how line graph chart is displaying in its mobile application. Therefore, mobile application should also able to view the trends of the water parameters using line graph chart, and it should be easy to understand and identify the trends. However, this line graph chart also lack of providing the information of desired range of current water parameters that the fish is favourable.



Figure 2.6.2 The dissolved oxygen sensed data example (Tsai et al., 2022).

Lastly, this system provides warning messages to alert user about the poor condition of water. However, the research paper does not show the detailed implementation of the alert system. Therefore, it is assumed that the alert message is sent to user only when the app is open. Thus, this type of alert message is not complete enough, as it needs user to open app only can receive the warning message. It lacks a pop out notification feature in the smartphones for that user can receive the alert message anytime at anyplace whenever there is internet connection as the application is running in the background.

2.7 Software Development Methodology

There are three types of Software Development Methodology can be used to develop Water Quality Monitoring Web and Mobile Application, such as Waterfall, Agile and Lean.

2.7.1 Waterfall

2.7.1.1 Overview

In waterfall software development approach, one phase must be completed only can proceed to the next phase. Based on figure 2.28, there are 5 main phases in waterfall model:

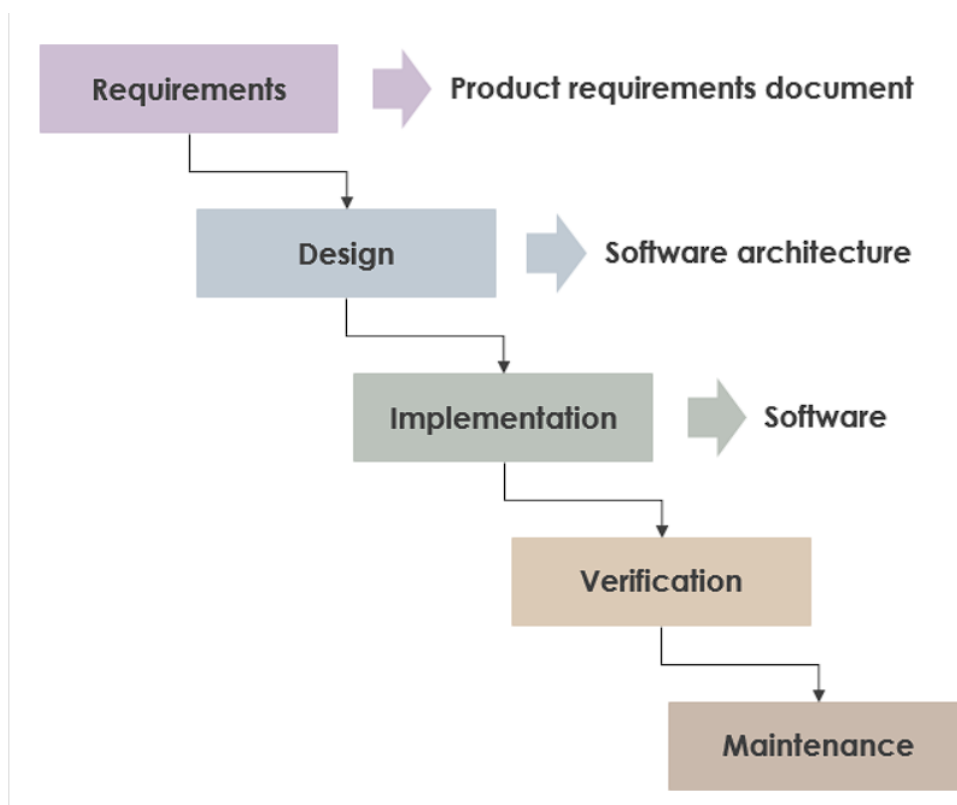


Figure 2.7.1 Waterfall approach (Visual Paradigm, n.d.)

1) Requirements:

In this phase, the system and software requirements should be documented in order to have the clear image on what the final product should look like. Therefore, it is required for all the stakeholders to involve in this phase in order to let development team to have clear concept and understanding on the requirements and expectation of the software product (OS-system, 2020). This project should be analysed in addition to the requirements to establish the budget, risks, dependencies, completion dates, and success measures. All the documentation should be completed before moving to another steps (Andersen, 2023).

2) Design:

In this phase, software developers will need to design the software architecture, business logic and concepts.

3) Implementation:

Developing software based on the documentation requirements and design.

4) Verification:

Testing whether the software is able to run properly without any error, and ensure it meets the user expectations.

5) Maintenance:

Once complete testing and verification, this product is then release to the stakeholders and regular maintenance is needed to always keep the product workable.

Pros:

- 1) Have a thorough and clear understanding on what the stakeholders expect on the products.
- 2) Easy to schedule and manage the tasks requirements.
- 3) Precise evaluation and calculation on the project cost.

Cons:

- 1) Not flexible and the cost of changes is literally high.
- 2) May cause the stakeholders to lose confident with development team due to late deliverable of the project product for user to review and give feedback.
- 3) Any miscommunication and misunderstanding with initial requirements, will greatly impact the final products.

2.7.2 Agile

2.7.2.1 Overview

Agile is a flexible software development approach. It starts with breaking down the projects into several small manageable modules. Then the developers focus on complete each module one by one following the completion target planning. Therefore, the products is deliver in an incremental ways which encourages the stakeholders to keep reviewing, and provide feedback, and make improvement based on the feedback. Figure 2.29 shows that the development stages of Agile is not a line, but in a circle due to its incremental and .

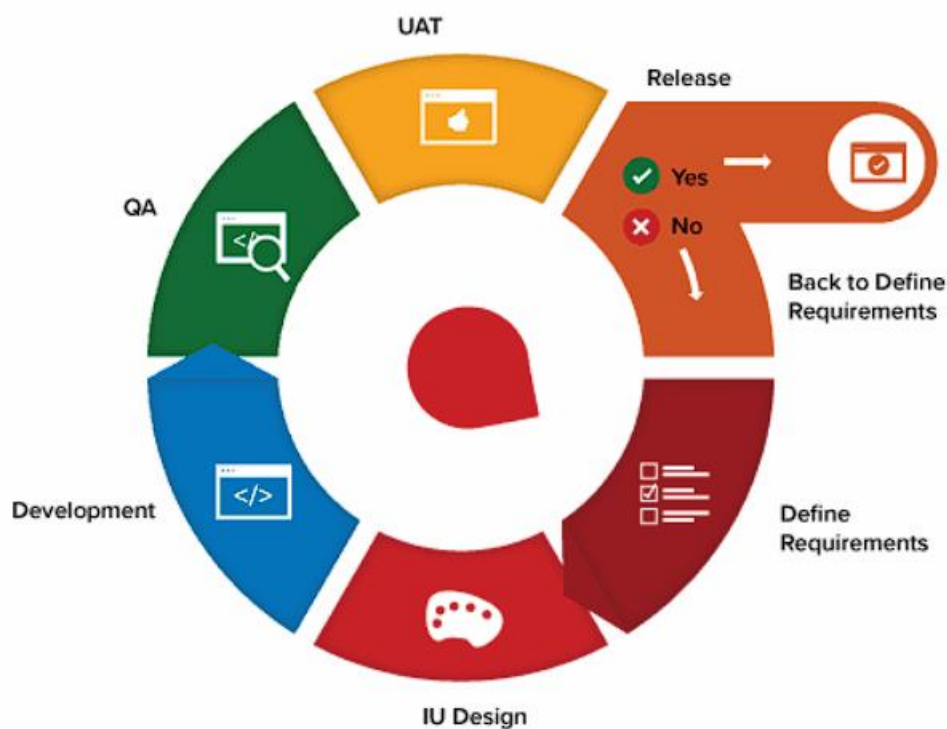


Figure 2.7.2 Principle of Agile Methodology (OS-system, 2020)

Pros:

- 1) All the bugs can be found and fixed in early stage, which in turns reduce the cost of fixing bugs.
- 2) Highly flexible and ensure that the stakehodlers has high probability to accept the final product as this approach requires them to participate actively in developing the software.
- 3) Faster software development lifecycle.

Cons:

- 1) Due to the nature of keep reviewing and changing, the final products may have different with the initial expectation of the software by stakeholders.

2.7.3 Lean

2.7.3.1 Overview

Lean startup is a data-based approach, this methodology is used to boost the project works, and also to reduce the cost and time. It starts initially by creating smaller subset of features of an application, then release the product into market.

Then the product is then keep improving using the user feedback repeatedly until the product achieve desired results. This approach make use of the fast delivery of product to quickly identify the requirements of the market towards this application and thus helps to determine to correct path to develop the application.

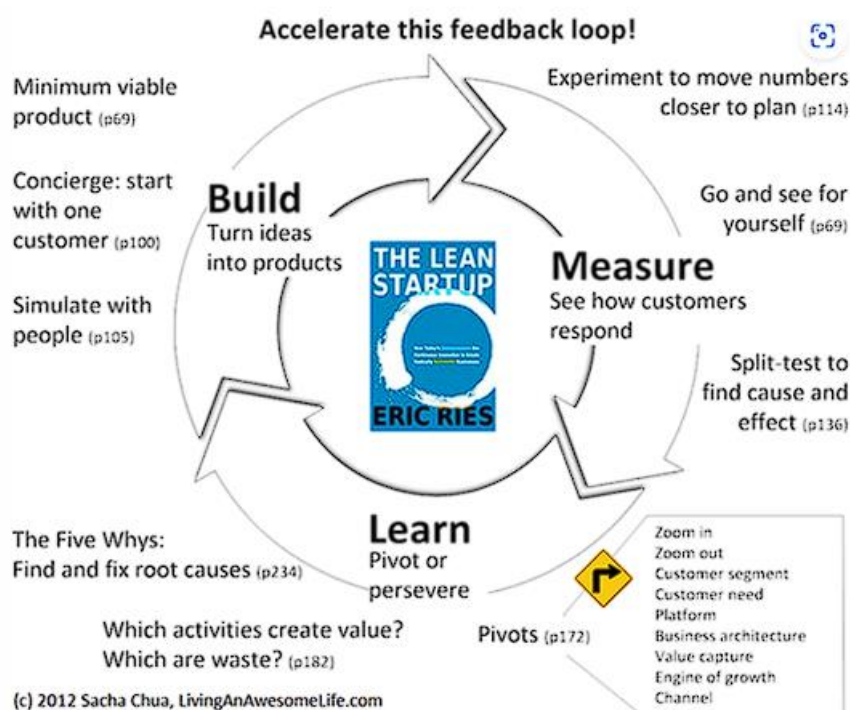


Figure 2.7.3 Principle of Lean Methodology (OS-system, 2020)

Pros:

- 1) The product can compete in the market quickly.
- 2) A smart and strategic way to develop a software product efficiently.
- 3) Flexible way of development as it can quickly adapt to the changing market.

Cons:

- 1) Unable to be used in high uncertainty project.
- 2) Not ideal for large and complex projects

2.8 Comparison of Software Development Methodology

These three Software Development Methodology are useful in different conditions. But in the scenario of developing a Water Quality Monitoring Application, Agile is the most suitable method to implement.

This is because Water Quality Monitoring Application may always require changes, because this project is focus on the Dashboard to let user to monitor the water quality easily. Therefore, stakeholders might always request changes because the better design of the dashboard is always existed. Agile methodology can easily adapt to the changes easily when there is adjustment on the project scope. Waterfall is not suitable to be used here.

Next, this application has high level of complexity and uncertainty. Therefore Agile methodology nature of breaking project into smaller manageable modules, allow the project to be developed incrementally to reduce the complexity. While at the same time continuous testing is needed to test these small modules, repeated validation and feedback can help this project to identify and address issue in early stage. Lean is not suitable to be used here because it focus on reducing waste and increasing efficiency in the development process, but there are too many unknown and changes might needed which makes this approach not effective as changes means increase cost.

Moreover, Water Quality monitoring involves multiple technologies, therefore need to have collaboration between different areas of technical expertise. Agile approach emphasize teamwork and collaboration, which frequent meeting is needed, therefore it is more suitable for different expertise to communicate and collaborate well in delivering this system.

2.9 Justification of Aquaculture Monitoring System

The use of the latest technologies, such as intelligent aquaculture monitoring systems, is necessary to replace traditional aquaculture practices. This is because it can overcome the weaknesses of traditional methods and provide a more convenient and relevant way to monitor aquaculture. For example, the weaknesses of conventional method for aquaculture are:

- 1) It takes longer time, more cost, and less consistent and accuracy in measuring water quality.

According to Huang et al. (2013), Labor cost is discovered to have the highest proportion in aquafarming cost which is 37.2% of total costs. Das and Jain, (2017) also complains that conventional ways of measuring water quality is not efficient, it needs to collect water samples manually and then send to lab to test and analyze. This ways of testing water is time intensive, cost ineffective and waste of human resource.

The implementation of an IoT water quality monitoring system in turn reduces labor costs and is cost effective in the long run. Using of sensors to measure water quality can avoid human error and inconsistency in testing water quality due to the different levels of experience and skill of personnel. In addition, water quality can be measured in real time, which is impossible to do with a human.

- 2) Aquafarmer cannot be alerted or warned when the water quality suddenly becomes bad.

Due to extra long time cost to test water quality, the frequency of water quality testing is very low, usually is done once per day or a week. Therefore, it is unable for the aquafarmer to immediately to realise that the water quality suddenly turns bad due to some unexpected events such as algae bloom, toxic chemical is presence in the water and finally leads to fish kill events within short period of time.

Due to the real time water quality testing provided by the IoT water quality monitoring system, an alert system to warn user when the water quality is poor is now available. This helps aquafarmer to

immediately notice the worsen of water quality, and are able to take immediate action to find out the reason and resolve the issue.

- 3) Uneffective productivity, profitability and sustainability of aquaculture operation.

With bad water quality monitoring capability of conventional methods, it is unable to fully optimize the growth performance of farmed fish. In addition to the lack of an alarm system and infrequent water quality testing, the potential for fish kills to occur and cause damage to the revenue is simply high. In addition, the sustainability of aquaculture operations is very poor due to ineffective growth performance and expensive labor costs as stated above.

IoT-based aquaculture systems optimize water quality, resulting in increased productivity, improved profitability due to reduced likelihood of fish kills, and increased sustainability due to elimination of inefficient labor costs.

The existence of aquaculture monitoring systems developed using IoT technologies eliminates these vulnerabilities faced by traditional methods.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

To deliver this project, it is required to discuss the methodology to use and the work planning to ensure the project can be completed within time. So in this chapter, we will discuss about the software development methodology, project plan and development tool. Software development methodology section discuss about the implementation of evolutionary prototyping methodology into this project. Next, the project planning section provides a Work Breakdown Structure (WBS) and Gantt chart as a plan to track the work progress. Finally,

the development tool section discuss about the software tools and technology to develop the system.

3.2 Software Development Methodology

The Evolutionary Prototyping is one of the variant of prototyping methodology, which is one of the most popular models used in several project development. Prototyping methodology is a process that repeatedly refine the prototype based on customer feedback until the prototype satisfy the customer needs. However, there are different ways to perform prototyping methodology. Evolutionary Prototyping that we use in this project differs from other variants.

Figure 28 shows the difference between the flow of evolutionary prototyping and rapid throw away prototyping (02DCE @02DCE, 2020).

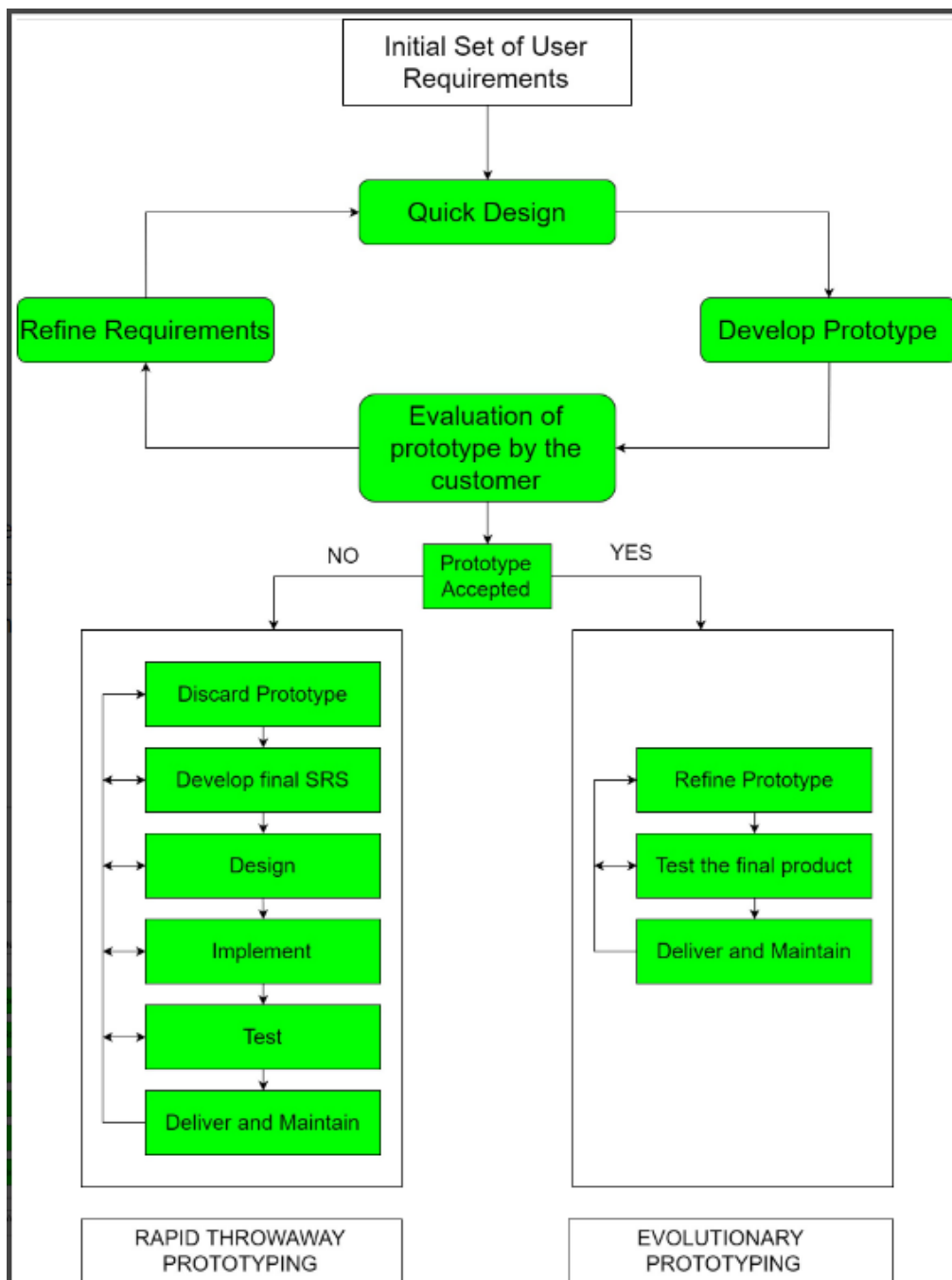


Figure 3.2.1 The difference between the evolutionary prototyping and rapid throw away prototyping (02DCE @02DCE, 2020).

3.2.1 Initial Set of User Requirements

The evolutionary prototyping software development methodology model starts with getting the initial sets of user requirements. The information for the software requirements is collected from stakeholders, especially end-users.

Then the software requirements is documented based on their information provided.

In this project, the initial requirements are collected by reviewing current existing water quality monitoring application's dashboard. In addition, further discussion and reviewing of requirement specification should be done with the aquafarm owner to confirm with the software requirements. After the initial requirement specifications is identified, WBS and Gantt Chart also should be prepared to determine the deliverables of each stage of the project cycle. The WBS helps determine what should be done, and the Gantt chart arranges time schedule to complete each WBS components to allow stakeholders to track on the project progress in order to ensure the project is able to be complete on time.

3.2.2 Quick Design

In this phase, a high-level design of prototype is produced. It should includes an overall basic architecture of the software, data structure and interfaces.

This stage used ThingSpeak and ThingsBoard to build the dashboard prototype for data visualization of this system. This is a low-fidelity prototype that shows how the interface looks like and what function and features should be included in this project.

3.2.3 Develop Prototype

Based on the quick design, develop the basic prototype for the software system. This prototype should be functional and can be tested by users to provide feedback.

This stage used ThingSpeak and ThingsBoard to build the dashboard prototype for data visualization of this system. This is a mid-fidelity prototype that does not really obtain the collected sensor data form cloud. But this prototype is clickable to allow end-user to testing whether they can easily understand and use the function and features provided in this system. The aim of this prototype

is to encourage end user to determine whether the software requirements is satisfied.

3.2.4 Evaluation of prototype by the customer

Based on the developed prototype, stakeholders and end users evaluate the prototype and provide feedback on functionality and features.

During this phase, aquaculturists assess whether previously established prototypes are usable and acceptable, and if not, provide their valuable feedback for improvement in the next round of the cycle.

3.2.5 Refine Requirements

If any requirements are not met or are missing, this leads to modification and redefinition of software requirements. Then the cycles from quick design to evaluation keep recycling until the stakeholders satisfy and with the prototype.

This phase is only entered when the prototype fails to satisfy user expectation of software requirements. After having their feedback, this stage need to revise the software requirement specifications.

3.2.6 Refine Prototype

After the user accept the prototype means that the basic functionality is already finalize. However, there is still need to refine the prototype by adding new detailed functionality and features.

In this phase the prototype is further refined after the stakeholders accept the prototype design.

3.2.7 Test the final product

After the prototype is refined, this prototype need to be tested and evaluated by stakeholders to ensure it meet the specification requirements. If not yet satisfied, returning back to the previous stage to refine the prototype.

3.2.8 Deliver and maintain

Once the prototype passes all tests and is approved by the stakeholders, this software system can be delivered to the end-users. However, this system should always be maintained and updated to meet the changing needs of the end-users and meets quality standards.

3.3 Project Plan

3.3.1 Work Breakdown Structure (WBS)

0.0 Web and mobile application development for water quality monitoring

1.0 Project Initialization

1.1 Preliminary Planning

1.1.1 Understanding project background

1.1.2 Identify problem of current conventional solution

1.1.3 Determine project objectives

1.1.4 Define project proposed solution

1.1.5 Confirm project approach

1.1.6 Define project scope

1.2 Literature Review

1.2.1 Review the paper that study Smart Aquaculture System using IoT technology

1.2.2 Review current existing Smart Aquaculture System

1.3 Methodology and work plan

1.3.1 Select software development methodology

1.3.2 Identify software development tool

1.3.3 Develop Work Breakdown structure (WBS)

1.3.4 Develop Gantt Chart

1.4 Requirement identification

1.4.1 Requirement Specification

1.4.1.1 Collect functional requirement

1.4.1.2 Collect system requirement

1.4.1.3 Collect non-functional requirement

1.4.2 UML modelling

- 1.4.2.1 Create use case diagram
 - 1.4.2.2 Create use case description
- 2.0 System Development
 - 2.1 Design interfaces
 - 2.1.1 Develop low-fidelity web application prototype
 - 2.2 System Design
 - 2.2.1 Set up Thingsboard cloud-based environment
 - 2.2.2 Database' table design
 - 2.2.3 Create data dictionary
 - 2.2.4 Create data flow diagram
 - 2.2.5 Create activity diagram
 - 2.2.6 Develop mid-fidelity web application prototype
 - 2.3 System Development
 - 2.3.1 Configure the sensor and devices for water quality data collection to Thingsboard cloud platform.
 - 2.3.1.1 Connect and configure microcontroller to read sensor data
 - 2.3.1.2 Create Thingsboard account
 - 2.3.1.3 Create API for microncontroller to send data to Thingsboard Cloud Server
 - 2.3.2 Create dashboard using Thingsboard
 - 2.3.2.1 Create water parameter analog and digital gauges
 - 2.3.2.2 Create line graphs over time for each water parameter
 - 2.3.2.3 Provide easy analysis based on the data
 - 2.3.3 Develop alert system
- 3.0 System Testing
 - 3.1 Develop test plan
 - 3.2 Develop test cases
 - 3.3 Create user acceptance test

- 3.4 Apply system usability scale
- 4.0 Deployment
 - 4.1 System Deployment

3.3.2 Gantt Chart

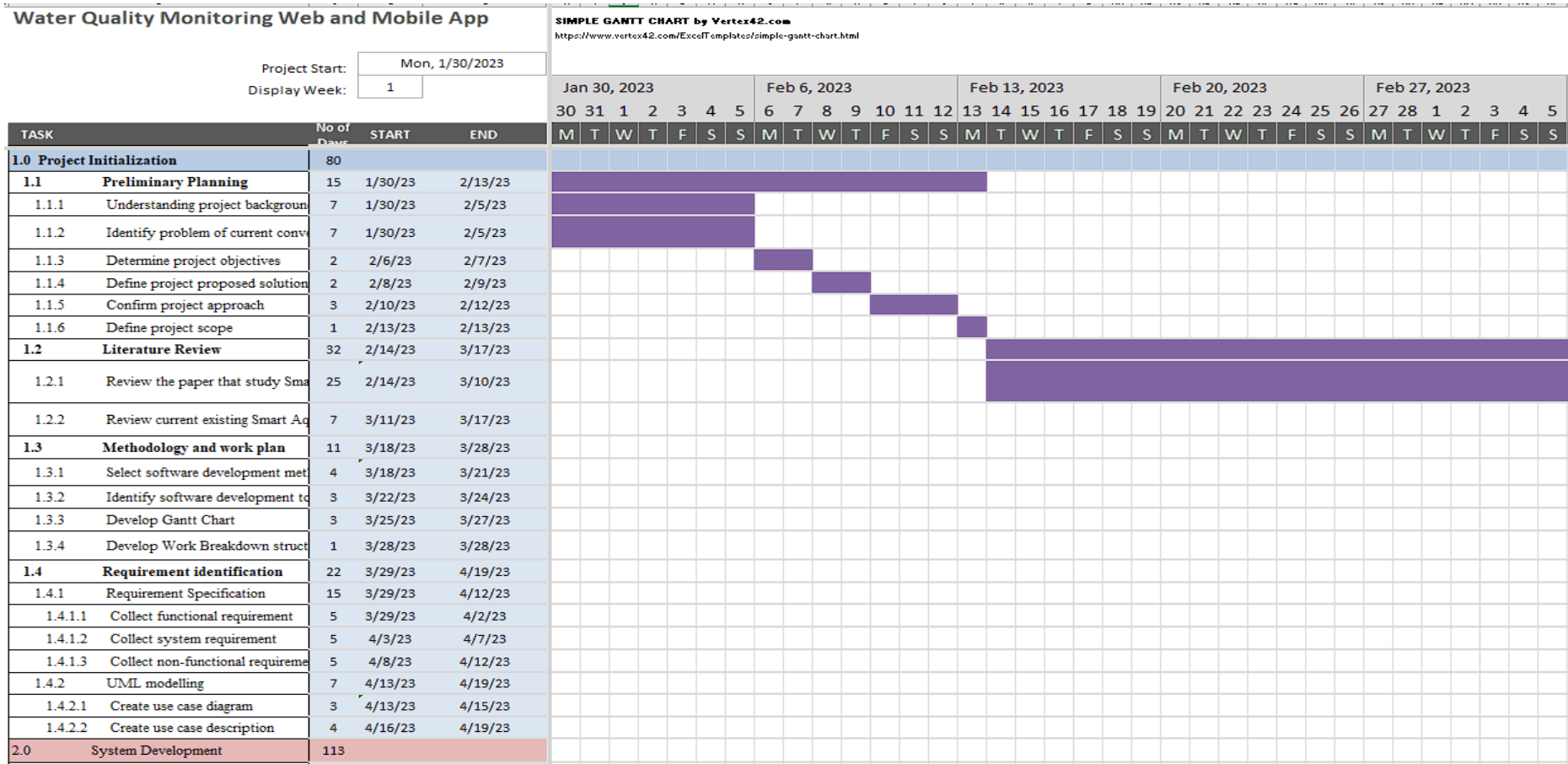


Figure 3.3.1 Gantt Chart for Project Initialization from 30/1/2023 to 27/2/2023

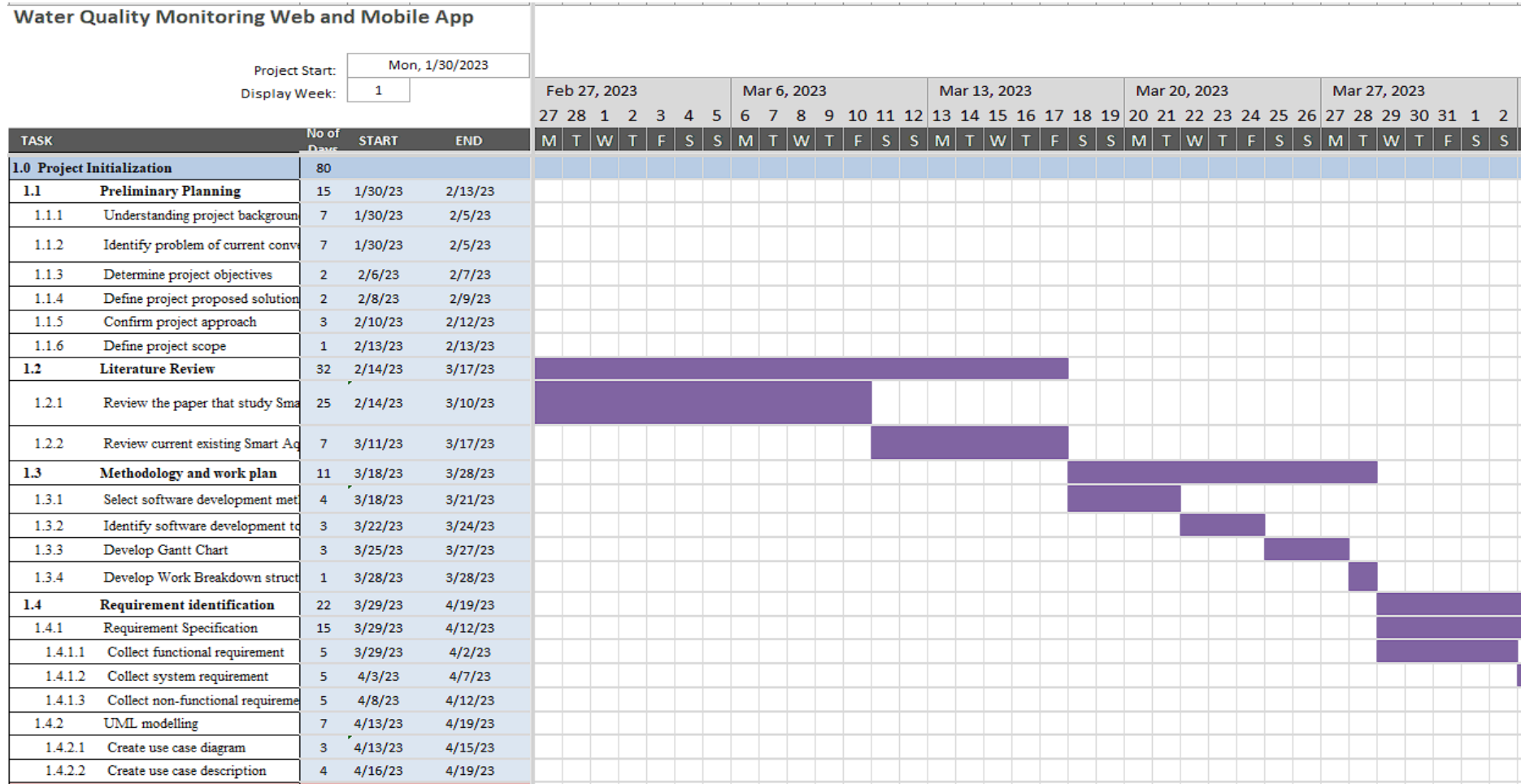


Figure 3.3.2 Gantt Chart for Project Initialization from 27/2/2023 to 27/5/2023

Water Quality Monitoring Web and Mobile App

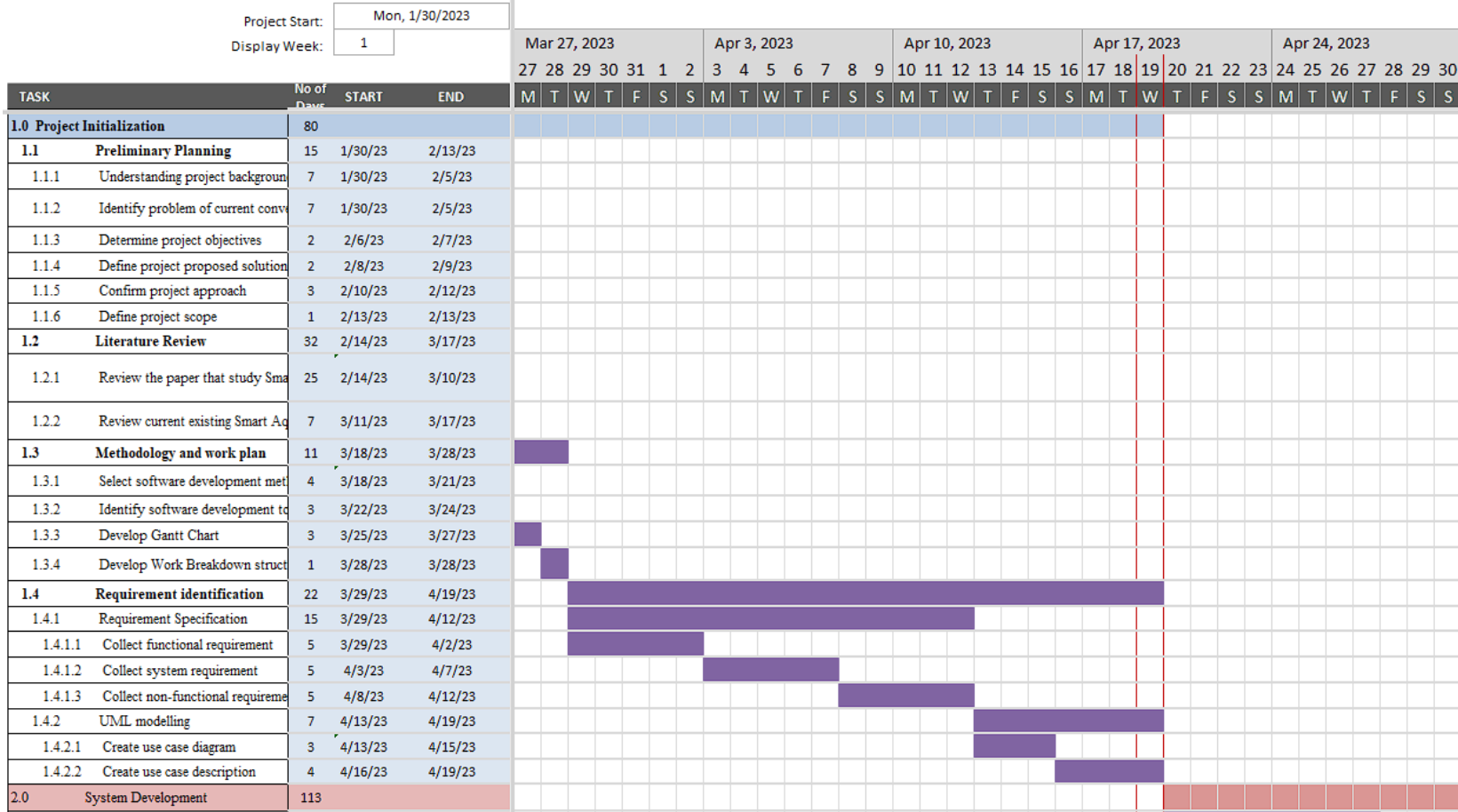


Figure 3.3.3 Gantt Chart for Project Initialization from 27/5/2023 to 24/4/2023

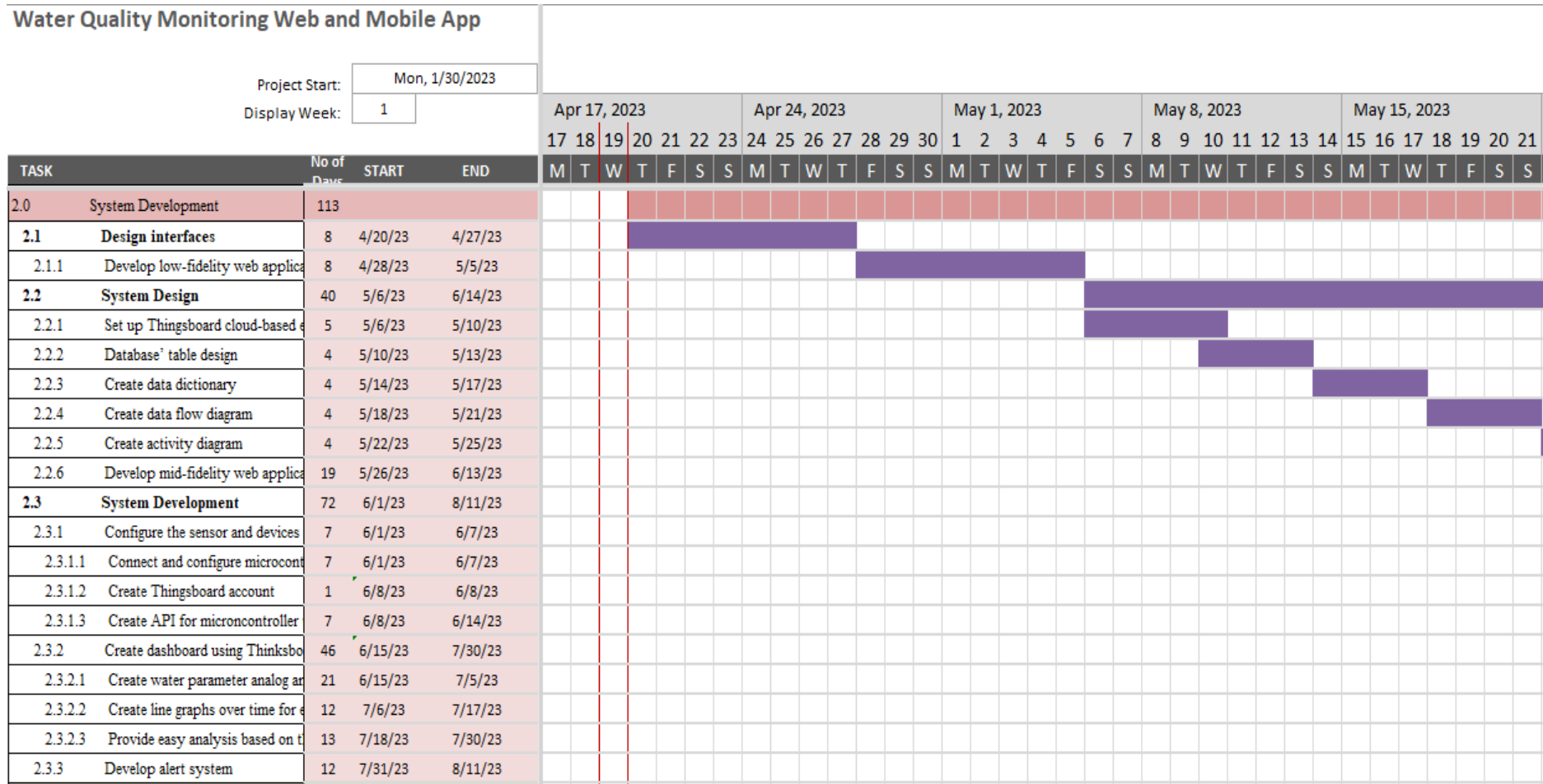


Figure 3.3.4 Gantt Chart for System Development from 20/4/2023 to 15/5/2023

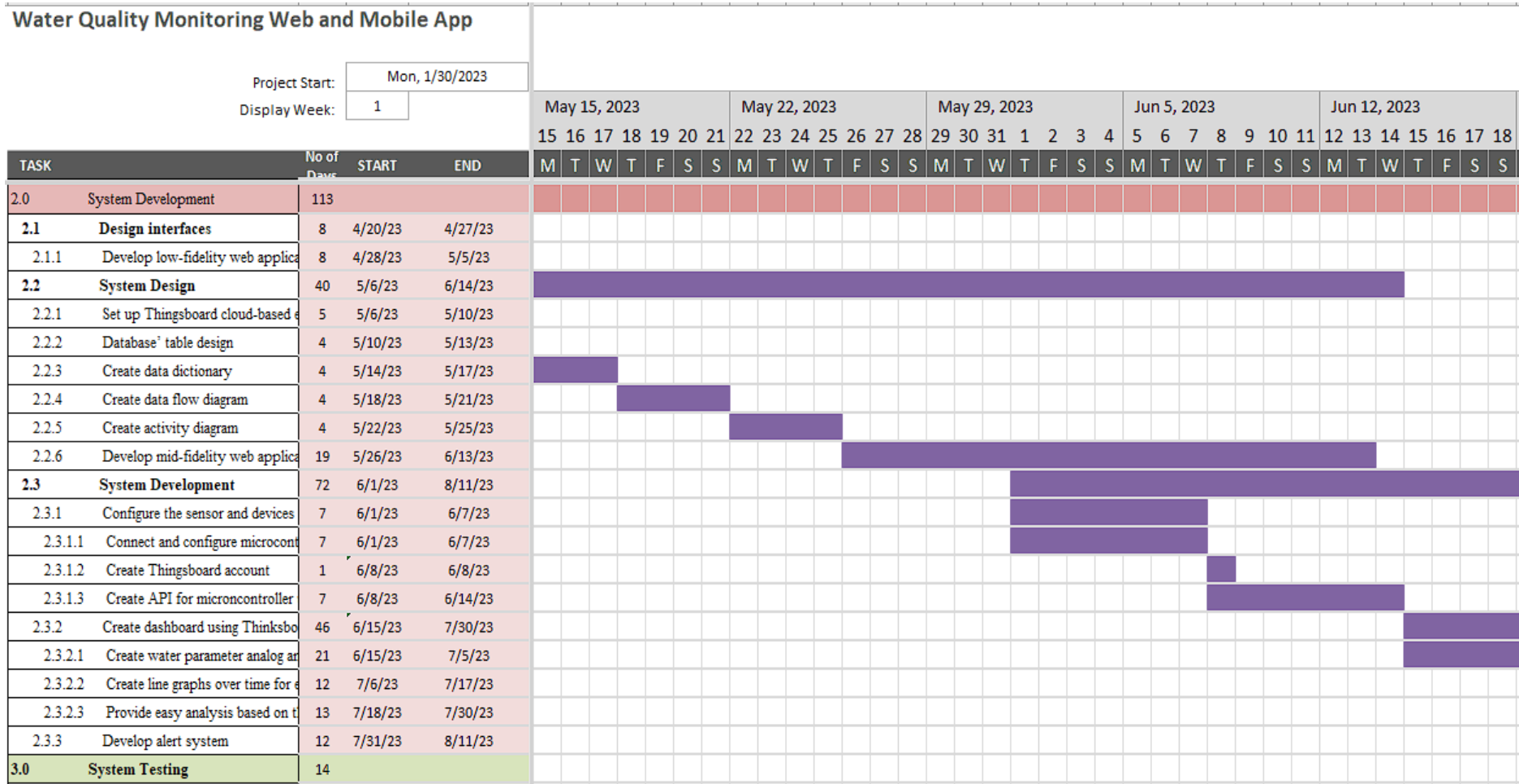


Figure 3.3.5 Gantt Chart for System Development from 15/5/2023 to 12/6/2023

Water Quality Monitoring Web and Mobile App

Project Start:

Display Week:

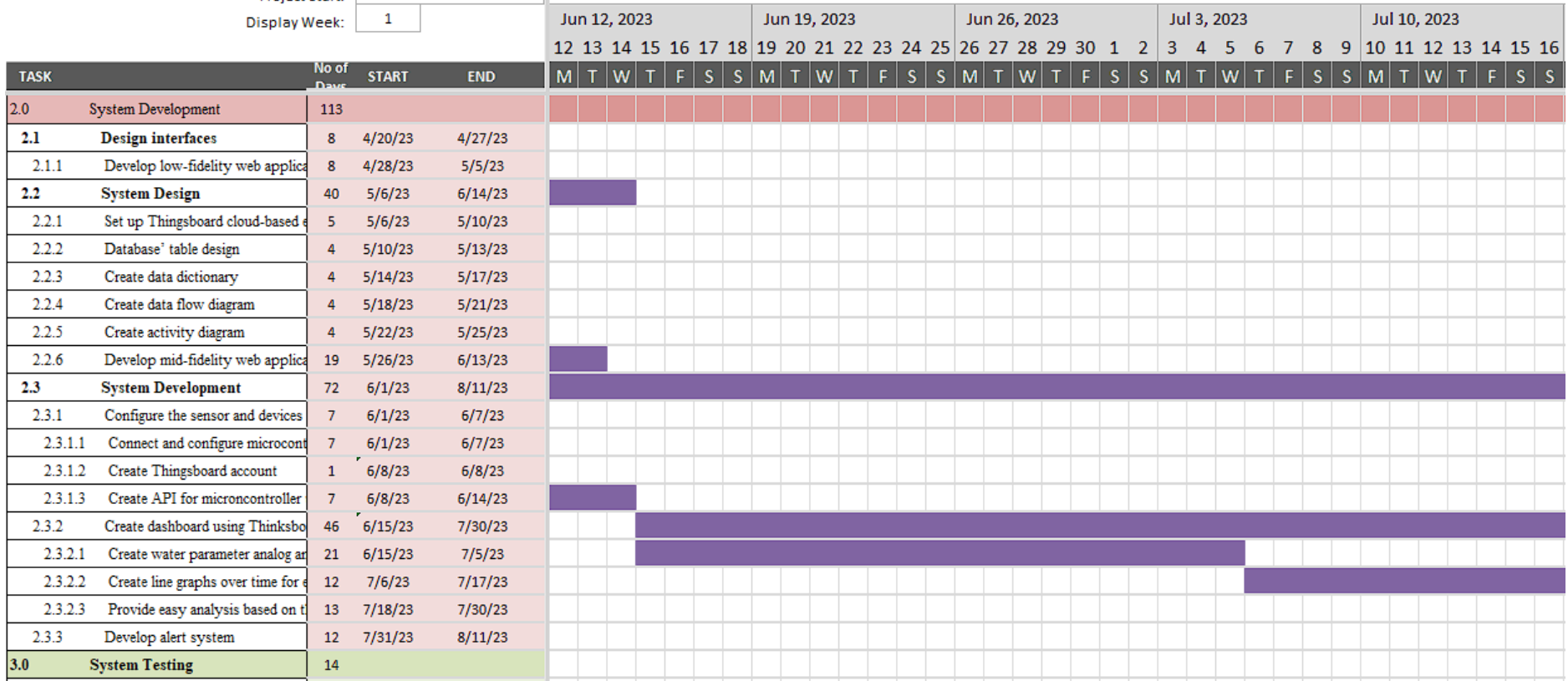


Figure 3.3.6 Gantt Chart for System Development from 12/6/2023 to 10/7/2023

Water Quality Monitoring Web and Mobile App

Project Start:
 Display Week:

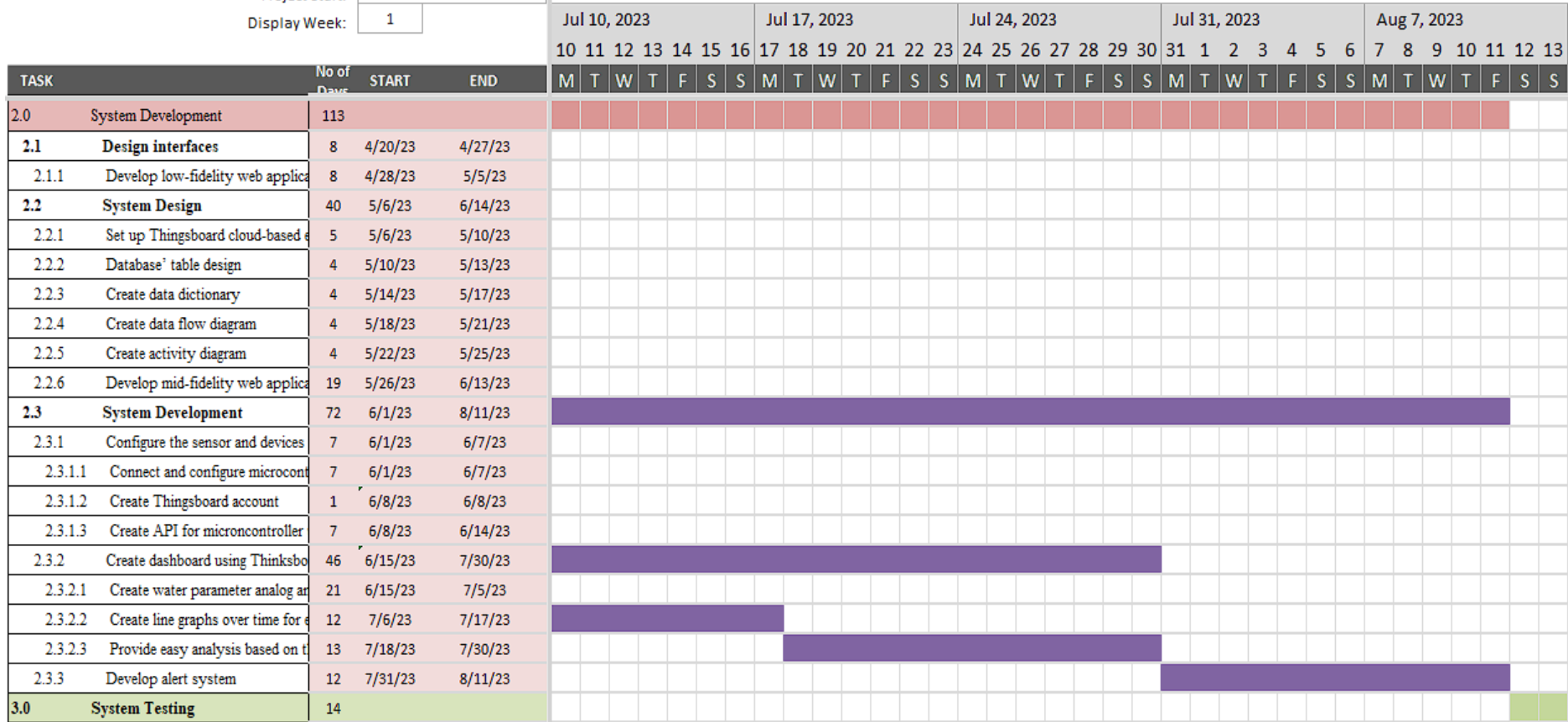


Figure 3.3.7 Gantt Chart for System Development from 10/7/2023 to 7/8/2023

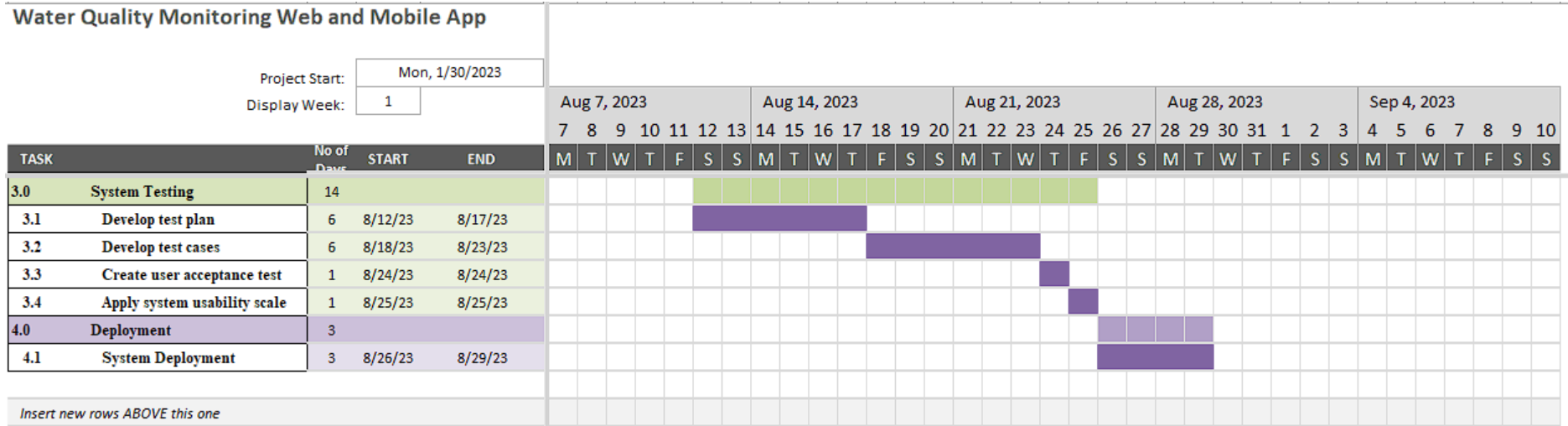


Figure 3.3.8 Gantt Chart for System testing and Deployment from 12/8/2023 to 28/8/2023

3.3.3 Development Tool

The development tool used in the project consists of Sensors, Sensors, Arduino microcontroller, Arduino IDE, Thingsboard, Node.js, React Native, React Native CLI, Visual Studio Code.

Sensors

There are three types of water quality sensors will be used: Dissolved Oxygen, pH, and Temperature sensors. These sensors should connect to Arduino Microcontroller and read the water quality data from the fish tank and then transmit the data to microcontroller.

Arduino Microcontroller

Arduino microcontroller should be able to receive the water parameters data from three sensors. The microcontroller should be able to connect to WiFi so that we can programmed the microcontroller to send the data to the Thingsboard Cloud Server through Http Protocol and MQTT protocol. This microcontroller can be programmed using Arduino Integrated Development Environment using another computer and later transfer the code into the microcontroller board using USB.

Arduino IDE

Arduino IDE is a code editor, compiler and uploader to program microcontroller. The programming language used here is variant of C++ called Arduino programming language. Arduino IDE has a Serial Monitor tool used to displays the data sent and received by the microcontroller, which is useful for us to show what is doing inside microcontroller, such as print error message, sensor readings, other processing data, or data send to other devices. The code to send sensor data to Thingsboard cloud is done here. The code program that use Thingsboard REST API to communicate with the Thingsboard cloud server is programmed using Arduino IDE and then upload to the microcontroller board.

Thingsboard

Thingsboard is an open source IoT platform that this project used as a Cloud Server database to store the data send by the microcontroller using REST API. Besides, Thingsboard provide many API and tools to support this project to build the IoT water quality monitoring application that include the dashboard and also alert system. The dashbaord shows all collected water quality parameter using meaningful diagram and to perform some simple analysis. Alert system is used to warn user when there is problem with the water quality.

Node.js

Node.js can be used to write server side application and also client side application. It is a javascript platform that allow this project to run javascript code outside browser, which means we are building a native mobile application. This Node.js provide Node Package Manager for JavaScript programming language which allow this project to install, manage the package. These packages include libraries, frameworks, tools and other dependencies. This is necessary tools to build the React Native Mobile Application.

Android Studio

Android Studio act as emulator, a virtual Android device to run the React Native project code during development, testing and debugging.

React Native

React Native is a framework of using JavaScript and React to build the Mobile Application. React is a library that helps to create user interface easily, while JavaScript is the programming language used together with React to write React Native Mobile Application. It include a tool called Metro, a JavaScript bundler used to compile and package the JavaScript code of React Native app. Then run this compiled program in Emulator for development and provide hot reloading function for faster development cycles. This can save a ot of time during development and testing.

Visual Studio Code

Visual Studio Code is an Integrated Development Environment (IDE) that provides code editor and compiler that can be used in developing React Native mobile application for this project. It provides many other functionality to improve the coding performance such as syntax highlighting, debugging, version control, and many extensions.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

This chapter discuss the requirements gathering process for developing a water quality monitoring system. This process is necessary so understand the stakeholders need and expectation for this system in order to ensure the system is developed correctly that follows their expectation without any confusion and misunderstanding.

4.2 Proposed System Architecture

The suitable system architecture of the aquaculture system should looks like figure 29, the proposed system architecture.

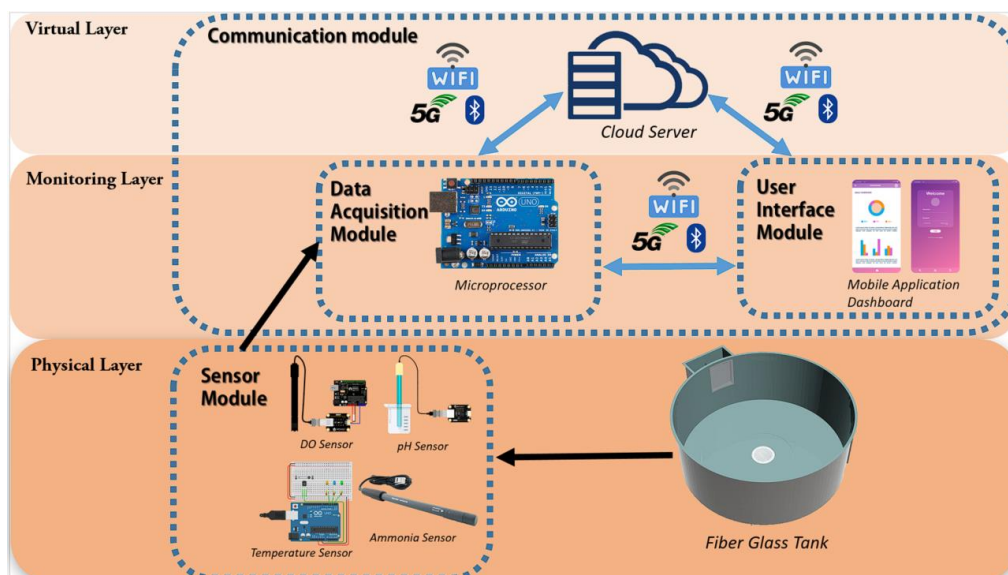


Figure 4.2.1 Proposed system architecture

This system should have at least 1 type of sensors: DO sensor, a Data Acquisition Module, a Cloud Server to setup a Water Monitoring System that are able to update the sensors data into the Cloud Server database.

4.2.1 Sensor

The suitable DO sensor can be used here should have similar functionalities with DFROBOT SEN0237 sensor in ISAS. This sensor should be able to read the dissolved oxygen (DO) level and must have the analog output with Analog-to-Digital Converter (ADC) function (Anon, n.d.). With this function, the sensor data in original sensed analog signal data is able to convert its format to digital values that is readable by microcontroller. Therefore, it must be compatible with the microcontroller used. Besides, it should have detection range between 0 – 20 mg/L which could cover full range of possible DO level in water.

4.2.2 Data Acquisition Module

There are 2 types of setup in Data Acquisition Module: 1) single Microcontroller with WiFi connectivity or 2) a Microprocessor and a Computer.

1) Using single Microcontroller with WiFi connectivity

The microcontroller used in this system should have the functionality similar with ESP32 microcontroller. This microcontroller should be compatible with the sensor chosen above in order to read the digital values of water parameter data. The most important feature that ESP32 provide is its Wi-Fi and Bluetooth connectivity function (Anon, n.d.). This is because we need to store the collected sensor data to Cloud Server, therefore microcontroller with built-in Wi-Fi are able to be programmed to achieve the task above. This cost of this setup is more cheaper than second setup as it does not need to have a computer to communicate with Cloud Server using any suitable Protocol such as MQTT and LoRaWan IoT Protocol.

2) Using a Microprocessor and a Computer

This setup contains a microprocessor to receive the digital values send by sensors, and combine these sensors data then store the data into local database of a computer. Then the computer responsibility is to keep updating the new received data into the Cloud Database. The microcontroller here should have similar ability as Arduino Uno, while the computer chosen here can refer to Raspberry Pi, a single-board computer consist of ARM Cortex

A72 processor and 8gb of RAM with multiple communication interfaces such as WiFi, Bluetooth, USB and Mini HDMI. Thus, with the WiFi connection features, this computer should be programmed to store the sensors data into the Cloud Server database using any suitable Protocol such as MQTT and LoRaWan IoT Protocol.

4.2.3 Cloud Server

The Cloud Server can be used here are Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform and others. The Cloud Server should be able used to store the water parameters data. Then these data will be access by the Mobile Application to perform data visualization.

4.2.4 Web and Mobile Application

The web application prototype is developed using Thingsboard. The mobile application is developed using React Native and Android Studio IDE as the development platform. Then this web and mobile application should use API or SDK to establish connection with the Cloud Server database. Then, use the retrieved water parameter data to perform Data Visualisation. For example, shows the latest current water parameter status in a Dashboard. The design of the dashboard should be informative, easy to understand, and also user-friendly. Same as the line graph of the water parameter over time, that let user to view the trends of the change of the water parameter in order to perform analysis. Next, the mobile application should run at the background and keep updating the water parameter readings. When the condition of water quality of fish farm is poor, the system should alert user as a pop out notification to notify user that the water quality need attention.

Figures below shows the sample views of the Web Application.

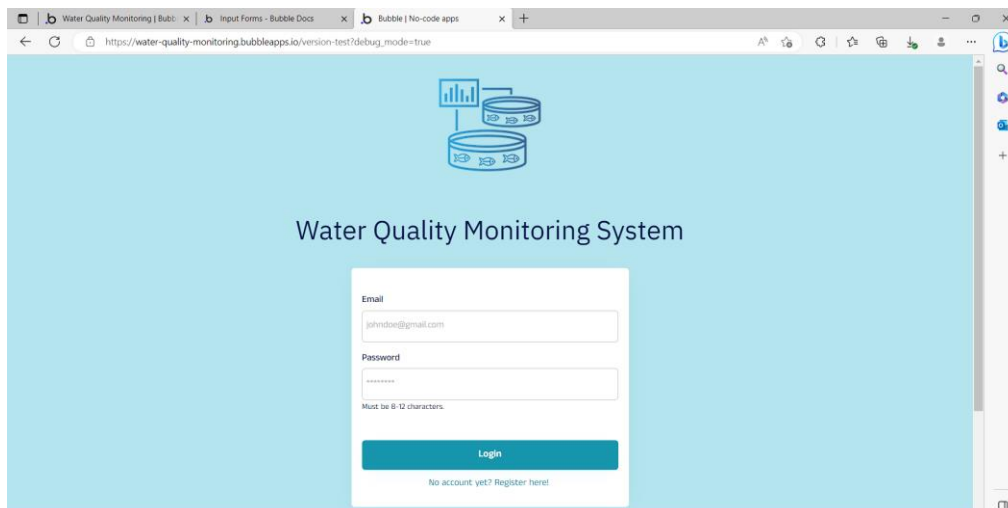


Figure 4.2.2 Login View

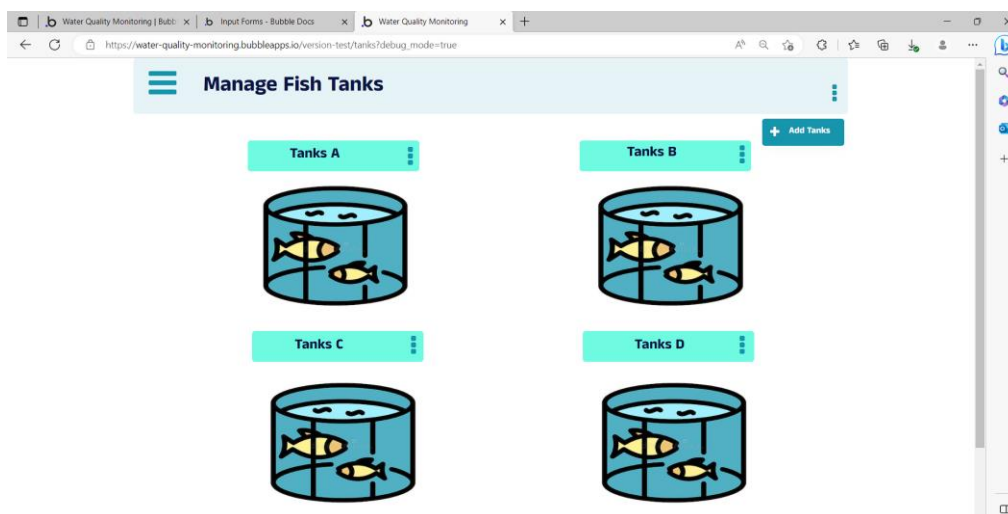


Figure 4.2.3 Fish Tanks

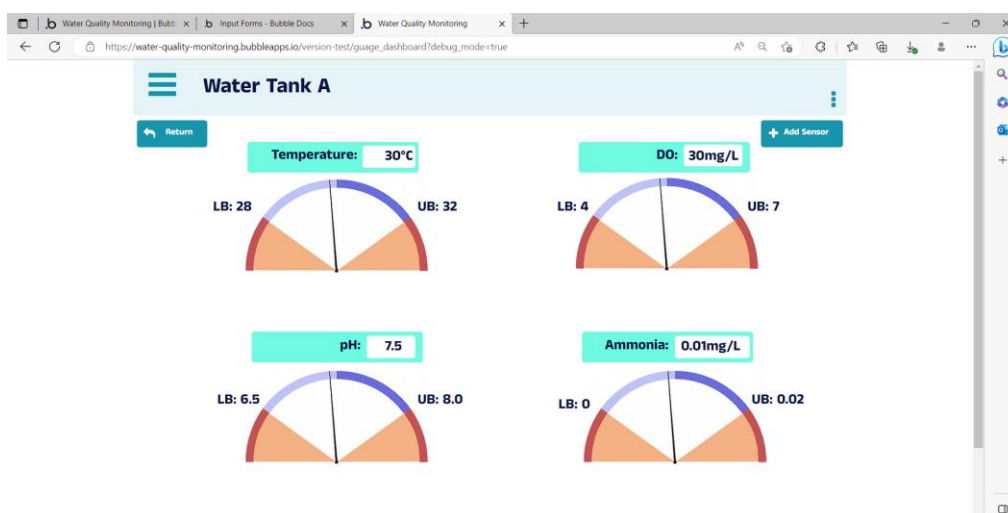


Figure 4.2.4 Water Tank A Water Parameters Gauge Dashboard

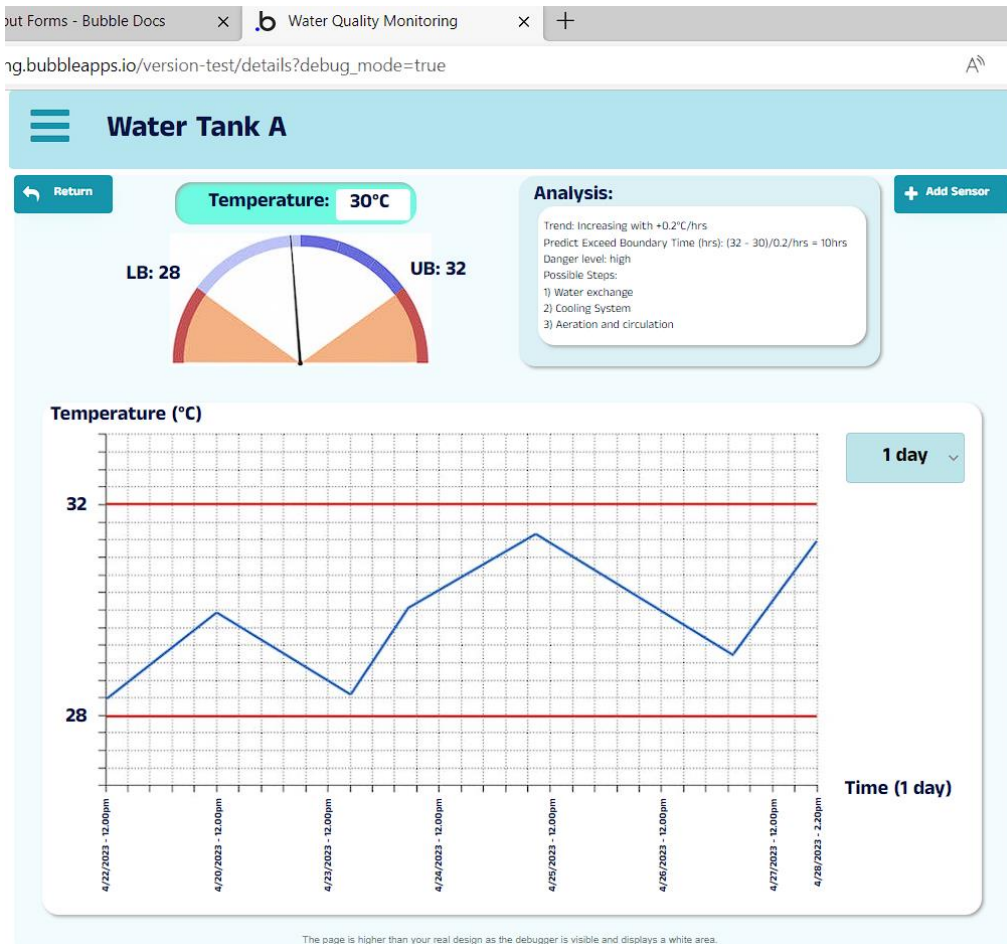


Figure 4.2.5 Water Tank A Temperature Details

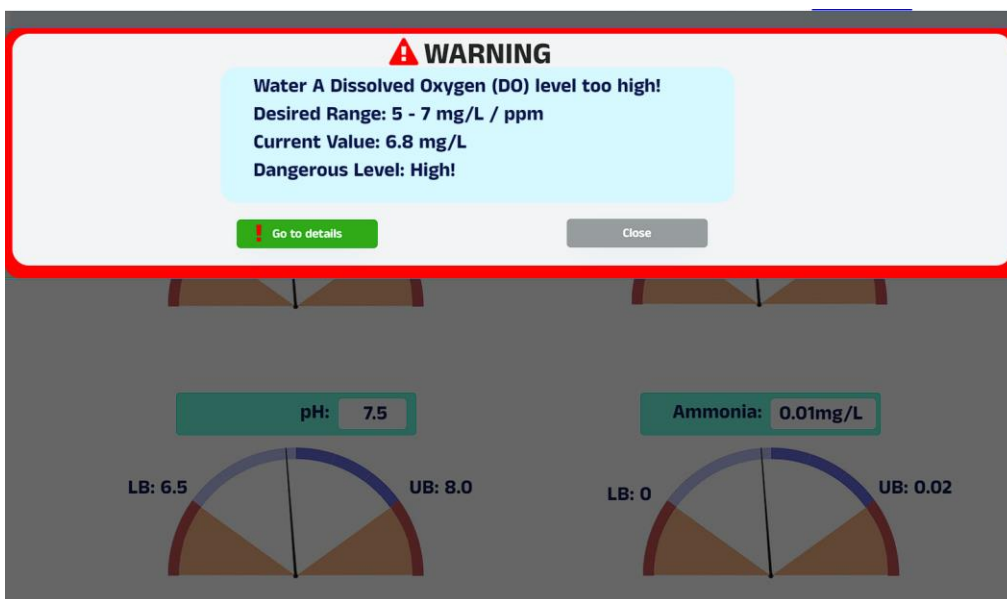


Figure 4.2.6 Warning Message

4.3 Requirement Specification

Therefore, Software Requirements Specification is include in this chapter, it contain functional requirement and non-functional requirement to define the software function and nature. use case diagram and use case description clearly identified how the end-users interact with the systems. Assumption and constraint of are defined in requirement specification.

4.3.1 Functional Requirement

Below table 4.3.1 shows the Functional Requirements of Water Quality Monitoring Mobile Application.

Table 4.3.1 Functional requirements of water quality monitoring mobile application

Function	Functional Requirements
Register	This system shall allow user to register a user account.
Login	This system shall allow user to login to the system to user the functionality of this system,
Dashboard	The system shall display a gauge dashboard page that allow user to check on the the latest condition for for the DO, pH, and temperature level for each water tanks.
	The system shall allow user to check the line graphs of each water parameter over time, in 12 hours, 1 day, 3 days, or 1 week.
	The system shall be able to let user to set the water quality parameter safety range and their dangerous limit, each tanks can have different range.
Manage	The system shall allow user to add or remove water tank for monitoring.
	The system shall allow user to add or remove sensor reading for each water tanks.
	The system shall be able to allow user to add fish species to each water tank, then can set their desired water quality parameter range for the added species to be implement in the dashboard.

Alert	The system shall be able to pop out notification to alert user when the water parameter is nearly exceeding or already exceed the safety range, only when user open the mobile application.
	The system shall be able to send SMS message to alert user clearly which water tanks need attention when its water parameter is nearly exceeding or already exceed the safety range.
	The system shall be able to change color of the gauge and line graph to red color to alert user when the water parameter is nearly exceeding or already exceed the safety range.
Analysis	The system shall be able to help user to perform simple analysis on based on the line graph, such as prediction of when will the water quality will exceed the safety range.

4.3.2 Non-Functional Requirement

- Performance
 - The system shall be able to response within 2 seconds when user are using the functionality of application, such as gauge dashboard, line graph, simple analysis and prediction.
 - The system shall be able to retrieve and handle these large value of data from cloud server.
 - The system shall automatically update the data for dashboard when the cloud database is updated with latest water parameters data.
- Reliability
 - The application shall be able to receive the lastest data from cloud server without any error.
 - The application shall be able to alert user when the the dangerous limit is reached without any mistake or miss out the notification.
- Accuracy
 - The application shall be able to draw the line graph correctly following the specified time frame.

- The application shall be able to make prediction reasonable and correctly using the following the algorithm formula used.
- Usability
 - The system shall allow user to understand the application easily and can be expert within 3 minutes.
- Scalability
 - The system shall allow adding more water parameters in dashboard when new sensors is adding to this system in the future.
 - The system shall be able to add more or remove water tanks to perform water quality monitoring, as the number of water tanks to monitor can be increasing or decreasing.
- Maintainability
 - The system shall be design for easy to maintain and update without causing any functionality disorder.
- Interoperability
 - The system shall be able to sync the changes between mobile and web application, when one of them made changes, another platform also should get changes.

4.3.3 Use Case Diagram

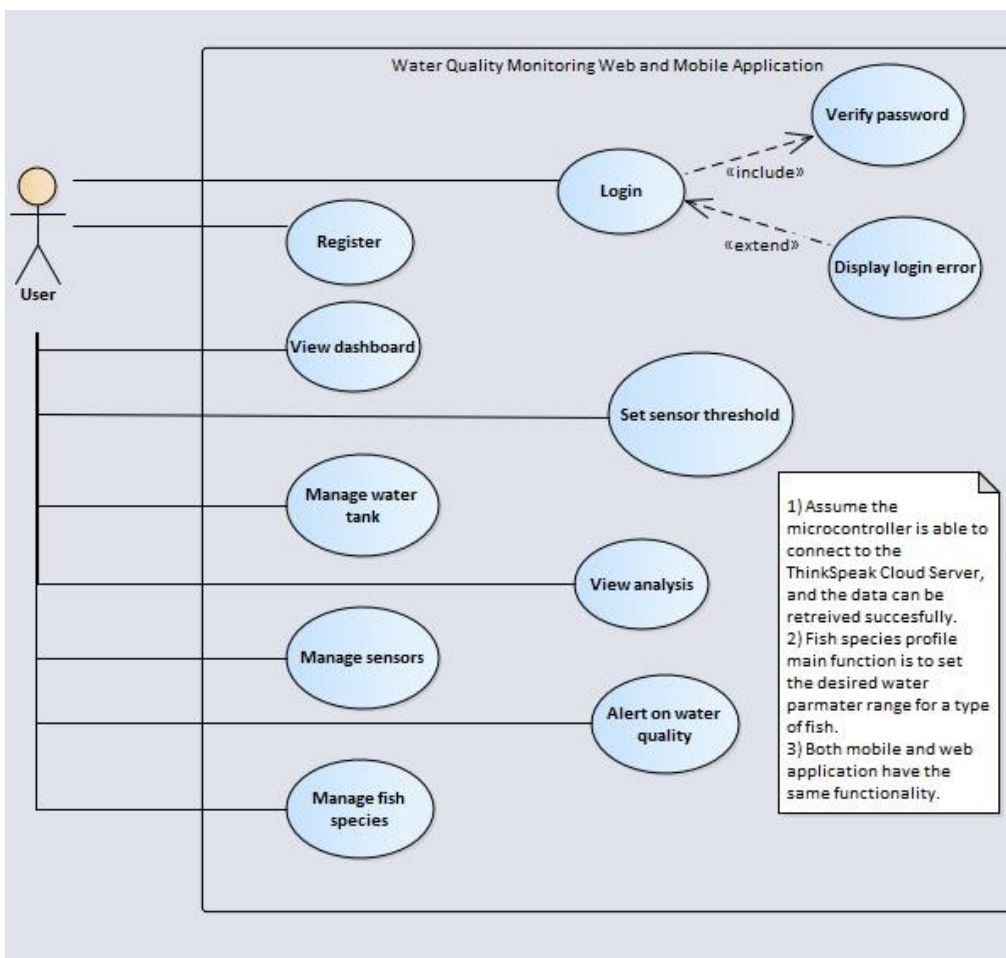


Figure 4.3.1 Use case diagram of Water Quality Monitoring Web and Mobile Application

4.3.4 Use Case Description

Use Case Name: Register	ID: 1	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
<p>Stakeholders and Interests: User – Users who does not have account but wish to use this web-based or mobile-based water quality monitoring system.</p>		
<p>Brief Description: This use case describes how the user can create an account to the web-based or mobile-based water quality monitoring system by filling up the email address, phone number, password and confirm password input fields to register an account.</p>		
<p>Trigger: A user wants to access the application but do not have account.</p>		
<p>Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A</p>		
<p>Normal Flow of Events:</p> <ol style="list-style-type: none"> 1. The user reach the login page. 2. The user click on the below link named Register an account. 3. The user then fill in the registration form. 4. The user click register an account button. 		
<p>Sub-flows:</p> <ol style="list-style-type: none"> 2.1 User will be redirect to registration page. 3.1 User fill in the email address, phone number, password and confirm password input field. 		
<p>Alternate/Exceptional Flows:</p> <ol style="list-style-type: none"> 4.1 If the email is registered, phone number format, email format or confirm password is wrong, the system will show user warning message and reject the registration, and ask user to provide correct information. 4.2 If the email is not yet registred, phone number format, email format and confirm password is correct, then system will register an account for user using these user input. 		

Use Case Name: Login	ID: 2	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who want to login to the system to use the water quality monitoring system features.		
Brief Description: This use case describes how the user can login to the system using the registered account credentials, using as email and password.		
Trigger: A user wants to access the application using his registered account credentials.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The user reach the login page.. 2. The user then fill in the login form. 3. The user click login button. 		
Sub-flows: 2.1 User fill in the email address, and password input fields.		
Alternate/Exceptional Flows: <ol style="list-style-type: none"> 3.1 If the email address format is incorrect or email and the password does not match the database, then prompt error and reject the login request. 3.2 If the email address format is correct and email and password are match with the database, then redirect user to home page. 		

Use Case Name: View Dashboard	ID: 3	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to view the condition and trends of each water parameters in a fish tank.		
Brief Description: This use case describes how the user can view the dashboard such as gauge dashboard and line graphs for the water quality parameters of a chosen fish tank.		
Trigger: A user wants to view the water quality dashboard of a water tank.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The user is now at the default water tank dashboard. 2. Then user click on the change water tank button to change to the correct water tank that the user want to view. 3. Then the user are redirect to the selected water tank dashboard which includes all the water parameter gauges with their readings. 4. Then user can click on the specific water parameter's gauge to redirect into the line graph dashboard page. 		
Sub-flows: <ol style="list-style-type: none"> 2.1 After user clicked the change water tank button, the system list all the water tanks that can be monitored. 2.2 Then user click on the water tank that user wish to monitor. 		
Alternate/Exceptional Flows:		

Use Case Name: Set Sensor Threshold	ID: 4	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to set the upper limit and lower limit of a specific water parameter gauge.		
Brief Description: This use case describes how the user can set the maximum and minimum threshold of the selected water parameter as the desired range so that to activate the alert system when the water parameter exceed the get near or exceed the threshold.		
Trigger: A user wants to set the maximum and minimum threshold of a specific water parameter of a water tank to activate alert system.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The user is now at the chosen water tank dashboard, the one that user wish to make changes. 2. Then user click on the water parameter's gauge that he wish to amend. 3. Then the user are redirected to that selected water parameter details page which contains the line graph. 4. Then user can view the current minimum and maximum threshold of the desired range to activate alert system. 5. Then the user click on the change desired water parameter range button to change the threshold to activate the alert system. 6. System shows a view provide the input field for user to key in the new value for the minimum and maximum threshold. 7. User fill in the new value for upper or lower limit or both. 8. User then click save, if fill in value is valid then successfully edited, if not then reject edit request. 		
Sub-flows:		
Alternate/Exceptional Flows:		

Use Case Name: Manage Water Tank	ID: 5	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to manage all the water tank that user wish to monitor.		
Brief Description: This use case describes how the user can manage all the water tanks.		
Trigger: A user wants to view/add/delete/edit water tanks that should already set up the sensors and devices to connect to this application.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events:		
View		
<ol style="list-style-type: none"> 1. The user now is at the Manage Water Tanks page, which can be access by clicking the change water tanks button at the default water tank dashboard page. 2. System show and list all the existing available water tanks in this page. 		
Add		
<ol style="list-style-type: none"> 3. If user want to add new tank, can click the add new tank button. 4. Then user fill in the necessary information to create a new device in ThinkSpeak cloud server, then get the access token. 5. The new water tank is now added, and the access token is used by microcontroller to setup REST API to send data to the application. 		
3. User click on the water tank, then a view with several options shows up.		
Edit		
<ol style="list-style-type: none"> 4. If user want to edit exising tank, user click on the edit water tank button in the view options. 5. Then user can edit the water tank information in the edit form. 6. Then click save button to save the edit. 		
Delete		
<ol style="list-style-type: none"> 4. If user want to delete the existing tank, user can click on the delete tank button from the view options. 5. Then system will ask user to confirm the delete operation. 6. User click confirm, the water tank is now deleted. 		
Sub-flows:		
Alternate/Exceptional Flows:		

Use Case Name: Manage Sensors	ID: 6	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to manage all the water parameter sensors in a water tank that user wish to monitor.		
Brief Description: This use case describes how the user can manage all the water parameter sensors in a water tank.		
Trigger: A user wants to view/add/edit/delete water sensors.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: View <ol style="list-style-type: none"> 1. User is now starting from the the water parameters gauge dashboard of a selected water tanks. 2. This system list shows all the existing water parameters gauge that are included in this water tank. Add <ol style="list-style-type: none"> 3. If user want to add new sensors, click add sensors. 4. Then user fill in the necessary information to create a new sensors of new water parameter. 5. Them system start to search whether the new water parameters data is retrieved inside the ThinkSpeak cloud server. <ol style="list-style-type: none"> 3. User click on the water parameter gauge, then user are redirected to the water parameter details page. 4. User click on the triple dot icon on right top corner, then a small view pops out showing the options. Edit <ol style="list-style-type: none"> 5. If user want to edit existing sensor, user click the edit button 6. Then the edit form show up, user fill in the edit form. 7. Then click save button to save the edit. Delete <ol style="list-style-type: none"> 5. If user want to delete the existing sensor, user can click on the delete sensor button from the view options. 6. Then system will ask user to confirm the delete operation. 7. User click confirm, the water sensor is now deleted. 		
Sub-flows:		
Alternate/Exceptional Flows: Add		

5.1 If yes, then the new water parameter gauge is added to the gauge dashboard, and this water parameter gauge is clickable to move into the details page.

5.2 If no, the water parameter gauge is not created and prompt a message saying that the water parameter sensor is not connected properly.

Use Case Name: Manage Fish Species	ID: 7	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
<p>Stakeholders and Interests: User – Users who wants to manage profile for fish species which can be used to set the water quality threshold according to the best optimum water parameter range of a fish inserted by user.</p>		
<p>Brief Description: This use case describes how the user can manage the fish species profile.</p>		
<p>Trigger: A user wants to view/search/add/edit/delete fish species profile.</p>		
<p>Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A</p>		
<p>Normal Flow of Events:</p> <p>View</p> <ol style="list-style-type: none"> 1. User is now starting from the the water tanks dashboard. 2. This system list shows all the existing water tanks. 3. User click on the triple dot button at the top right corner, a small option list pops out. 4. User click on manage fish species. 5. Users are redirected to Manage Fish species page. 6. System list all the current existing fish species. <p>Search</p> <ol style="list-style-type: none"> 7. There is a search bar for user to search for the specific fish using fish names. <p>Add</p> <ol style="list-style-type: none"> 7. If user want to add new Fish Species, click add new fish button. 8. User are redirected to a add new fish species page. 9. Then user fill in all necessary information such as fish name, the minimum and maximum water parameter limit. 10. Then press save to add the new fish into the fish species list. <ol style="list-style-type: none"> 7. User click on the fish name, then user will be redirect to the fish species details page. 8. User can click on the triple dot button to see more options view. <p>Edit</p> <ol style="list-style-type: none"> 9. If user want to edit fish species, user click the edit button from the view. 10. Then, all the water pamameters minimum and maximum threshold is editable now. 11. Then click save button to save the edit. <p>Delete</p>		

9. If user want to delete the existing fish species, user can click on the delete button from the options view.
10. Then system will ask user to confirm the delete operation.
11. User click confirm, the fish species is now deleted.

Sub-flows:

Alternate/Exceptional Flows:

Use Case Name: View Analysis	ID: 8	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to view some analysis and prediction on a specific water quality of a water tank.		
Brief Description: This use case describes how the user can view analysis and prediction of the trend of a water quality of a water tank.		
Trigger: A user wants to view analysis and prediction of the trend of a water quality of a water tank.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: View <ol style="list-style-type: none"> 1. User is now at the gauges dashboard. 2. User click on the water parameter gauges that he wish to view prediction and analysis. 3. User redirect to that water parameter details page. 4. User are able to see the analysis and prediction below the line graph. 		
Sub-flows:		
Alternate/Exceptional Flows:		

Use Case Name: Alert on Water Quality	ID: 9	Importance Level: High
Primary Actor: User	Use Case Type: Detailed, Essential	
Stakeholders and Interests: User – Users who wants to get alert when the water quality of a water tank become nearly reach unfavourable condition.		
Brief Description: This use case describes how the user can get alert using web application and mobile application when the water quality of a water tank become nearly reach unfavourable condition.		
Trigger: A user wants to get alert when the water quality of a water tank become nearly reach unfavourable condition so that user can make immediate action to prevent the water condition become worse to save the fish.		
Relationships: Association: User Include: N/A Extend: N/A Generalisation: N/A		
Normal Flow of Events: Web <ol style="list-style-type: none"> 1. User is now at any page of the application. 2. When any water parameter of a tank goes beyond the maximum or minimum limit of a water parameter, application pop out a warning windows telling user which water parameter is now in dangerous condition, the corresponding water tank information, and link lead to the water parameter gauges page of that water tank. 3. An SMS with similar message content is sent to user phone number. 4. The water parameter gauges that is dangerous now should be shown in red colour. 5. User click on that gauges, system redirect user to details page and show a list of possible action can be taken. Mobile <ol style="list-style-type: none"> 1. User mobile device is in sleep mode or in use condition, this application is running at the background. 2. When any water parameter of a tank goes beyond the maximum or minimum limit of a water parameter, a smartphone notification receives a warning message telling user which water parameter is now in dangerous condition, the corresponding water tank information. 3. User click the notification. 4. System open the application and redirect user to that water tank's gauges dashboard. 5. The water parameter gauges that is dangerous now should be shown in red colour. 6. User click on that gauges, system redirect user to details page and show a list of possible action can be taken. 		
Sub-flows:		

Alternate/Exceptional Flows:

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

This chapter discuss about the Water Quality Monitoring System Architecture. It is integrated with 4 systems, they are: Data Acquisition System, ThingSpeak Cloud Service, Laravel Back End, React Native Front End Mobile Application. Data Acquisition System helps to collects water parameters data from the aquafarm and sends it to the ThingSpeak Cloud Server. Thing Speak Cloud Service receives the data sent and store them into database and provide data visualization which can be accessed through browser. Laravel Framework Back End Server provides API to communicate with React Native Front End Mobile Application in order to provide it with data processing, data analysis, machine learning prediction , user authentication and authorization and other related operation functions. React Native Front End is the mobile application where user can interact with it to monitor the water quality, user can view the dashboard, record activity, open notification, user registration and login. With the integration of all these systems, a designated Water Quality Monitoring Systems could be successfully developed.

5.2 System Architecture Design

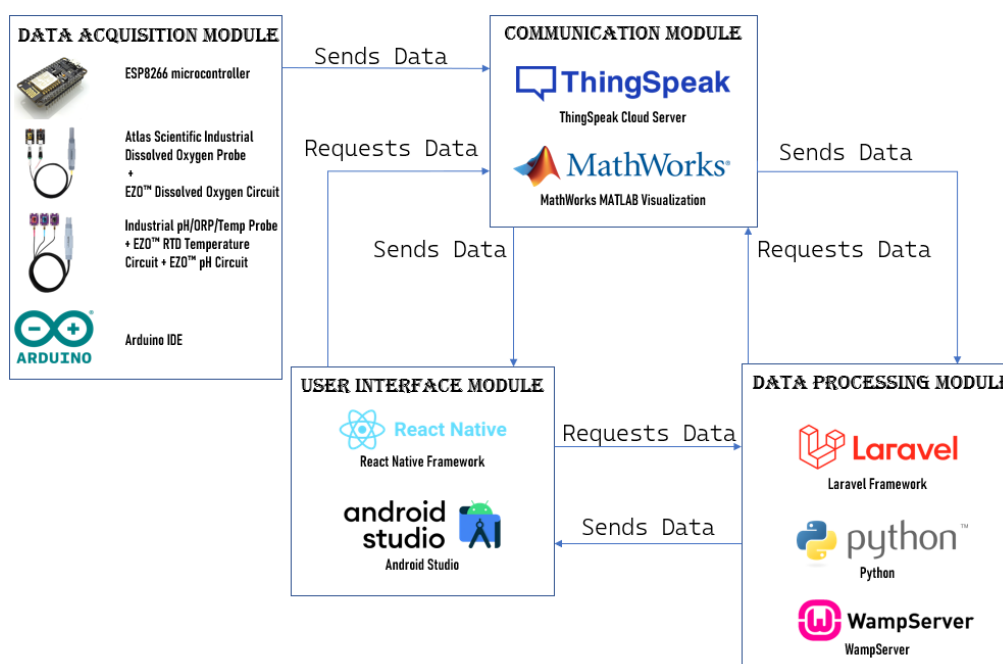


Figure 5.2.1 Water Quality Monitoring System Architecture

Figure 5.1 shows the complete System Architecture Design, this section discuss the components use to build the overall system architecture, each component's usage will be discuss in their corresponding sub-system architecture later. The Data Acquisition Module consist of ESP8266 microcontroller, Atlas Scientific Industrial Dissolved Oxygen Probe, EZO Dissolved Oxygen Probe, Industrial pH/ORP/Temp Probe, EZO RTD Temperature Circuit, EZO pH Circuit and Arduino IDE. Next, Communication Module consist of ThingSpeak Cloud Server service and MathWorks MATLAB Visualization. Data Processing Module consist of Laravel framework, Python, and Wamp Server. Lastly, User Interface Module consist of React Native Framework and Android Studio.

5.2.1 Data Acquisition Module Architecture

This section discuss the usage of each component in Data Acquisition Module, and how they works together to perform their task.

5.2.1.1 Atlas Scientific Industrial Dissolved Oxygen Probe + EZO™ Dissolved Oxygen Circuit ESP8266 microcontroller

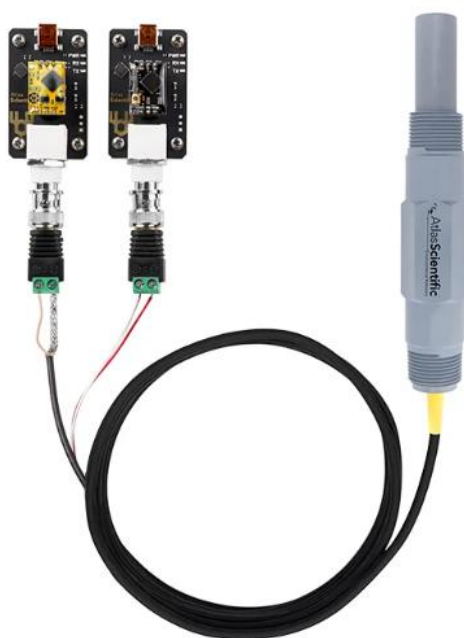


Figure 5.2.2 Atlas Scientific Industrial Dissolved Oxygen Probe connected to EZO™ Dissolved Oxygen Circuit ESP8266 microcontroller ("**Industrial Dissolved Oxygen Probe,**" n.d.).

Atlas Scientific Industrial Dissolved Oxygen Probe is the sensor used to collect Dissolved Oxygen in the liquid samples. It is chosen because it allows long term readings. This features is supported by its massive amount of electrolyte ensuring it to be working fine for a long period until the next calibration ("Industrial Dissolved Oxygen Probe," n.d.). This sensor collects the data and sends it to the EZO™ Dissolved Oxygen Circuit for accurate measuring.



Figure 5.2.3 The EZO™ Dissolved Oxygen Circuit

The purpose of this circuit is to preprocess the Dissolved Oxygen data value collected from the sensor to become highly accurate in Mg/L. Without this circuit, it is very difficult to calculate Dissolved Oxygen level accurately because it requires many mathematical calculations and needs around hundred times of chemical titration tests to validate the readings. This circuit using its functionality of temperature, salinity and pressure compensation to finalize the correct DO values ("EZO™ Dissolved Oxygen Circuit," n.d.).

5.2.1.2 Industrial pH/ORP/Temp Probe + EZO™ RTD Temperature Circuit + EZO™ pH Circuit



Figure 5.2.4 Industrial pH/ORP/Temp Probe + EZO™ RTD Temperature Circuit + EZO™ pH Circuit ("Industrial pH/ORP/Temp Probe," n.d.)

Atlas Scientific Industrial pH/ORP/Temp Probe is the sensor used to collect pH value and Temperature in the liquid samples. It is chosen because it is highly durable, and can last with a long period until the next calibration. Besides, this probes offers equivalent quality of sensing capabilities as lab-grade probes for pH and ORP. This means that it can provide accurate measurement result of pH. Moreover, this sensor also have built-in Temperature Sensor (PT-1000) which allows precise temperature measurements as temperature also helps to increase the accuracy of PH readings. This sensor collects the Temperature and pH data and sends them to the EZO™ RTD Temperature Circuit and EZO™ pH Circuit respectively ("Industrial pH/ORP/Temp Probe," n.d.).

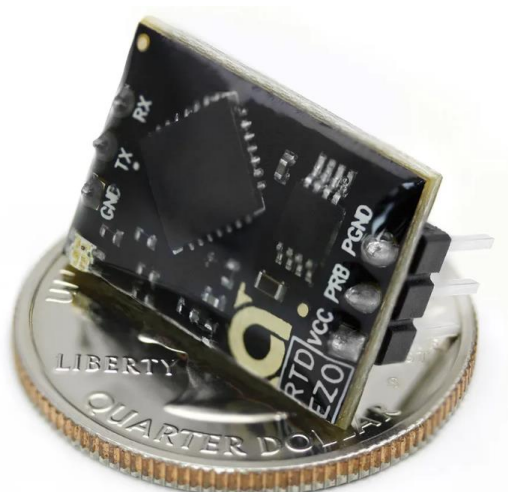


Figure 5.2.5 EZO™ RTD Temperature Circuit ("EZO™ RTD Temperature Circuit," n.d.)

The purpose of this circuit is to preprocess the RTD Temperature data value collected from the sensor and provide the most accurate temperature reading in °C.

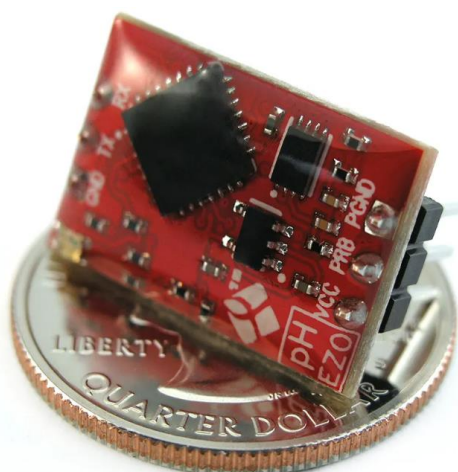


Figure 5.2.6 EZO™ pH Circuit ("EZO™ pH Circuit," n.d.)

The purpose of this circuit is to preprocess the pH value data collected from the sensor and ensure its accuracy to be on par with expensive bench-top pH meters.

5.2.1.3 ESP8266 microcontroller

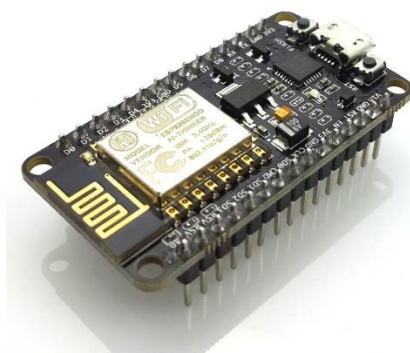


Figure 5.2.7 ESP8266 microcontroller

ESP8266 microcontroller is chosen to be used here because it contains built-in Wi-Fi capabilities, which allows internet connection. Then it can be programmed to send data to ThingSpeak Server using REST API provided by ThingSpeak. The programming language we used here is Arduino, and the tools used to program it is Arduino IDE. Besides, this microcontroller also can be programmed to keep requesting data from three EZO Circuit, DO, RTD, and pH with an interval of 15 seconds, once all the data from the sensors is collected successfully, then we will use the ThingSpeak Library function to sends the data to the Cloud Server for data storing.

5.2.1.4 Arduino IDE



Figure 5.2.8 Arduino IDE

Arduino IDE is an Integrated Development Environment that will be used to program the microcontroller in this project. This choice is driven by its open-source nature, and it has a lot of libraries including ThingSpeak.h, ESP8266Wifi.h, SoftwareSerial.h. ThingSpeak.h library provides convenient function to sends data to ThingSpeak server. ESP8266Wifi.h library allows microcontroller to establish internet connection via ESP8266 microcontroller's Wi-Fi capability. SoftwareSerial.h library enables communication between microcontroller and EZO circuits using baud rate of 9600.

5.2.2 Communication Module Architecture

5.2.2.1 Cloud Integration with ThingSpeak

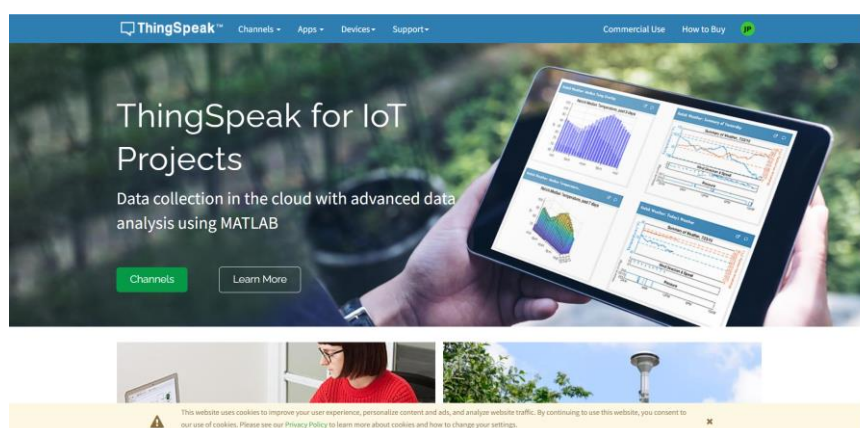


Figure 5.2.9 ThingSpeak Website

Figure 5.9 shows the website interface for utilizing the ThingSpeak Cloud service. ThingSpeak has been selected in this project due to its suitability for IoT projects for data collection in cloud, and it also provides several features such as Data Visualization and Data Analysis. In order to start the service, we need to create a new Channel and get the Channel ID, the Channel access must be set to public and the Channel ID is essential for the microcontroller to send data to the correct channel. Besides, in the user interface module and data processing module, this Channel ID is a necessary component, serving as a key required for making REST API requests.

5.2.2.1.1 Channel

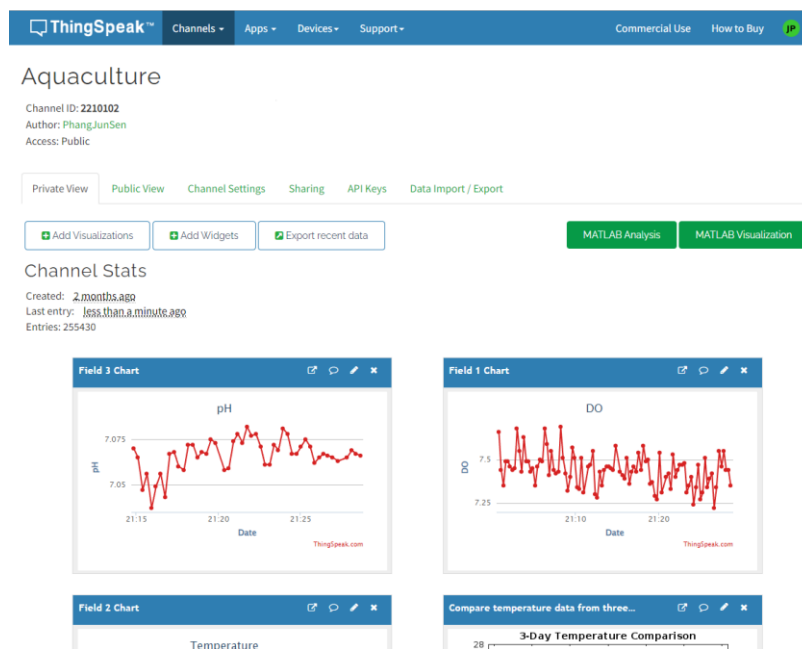


Figure 5.2.10 Sample Channel

Figure 5.10 is a sample view of ThingSpeak Channel, it has Channel ID, and user can add Visualization, Add Widgets, and integrate it with MATLAB Visualization and MATLAB Analysis to perform more advanced visualization and analysis. The line graph shown is the default line graph to show the current database columns storage condition. Each line graph stands for the data collected in each column over time.

5.2.2.1.2 API Keys

The screenshot displays the 'API Keys' management interface for a ThingSpeak channel. The channel name is 'Aquaculture' with ID '2210102'. The interface is divided into several sections:

- Write API Key:** Shows a key 'UJT4ZI3HV5BCLQQ7' and a 'Generate New Write API Key' button.
- Read API Keys:** Shows a key 'TRYX3FPTI2HU1ZG1', a 'Note' field, and buttons for 'Save Note', 'Delete API Key', and 'Add New Read API Key'.
- Help:** Explains that API keys enable writing or reading data from a channel. It lists:
 - Write API Key:** Used for writing data to a channel.
 - Read API Keys:** Used for viewing private channel feeds and charts.
 - Note:** Used for tracking channel access.
- API Requests:** Lists three example GET requests:
 - Write a Channel Feed:** `GET https://api.thingspeak.com/update?api_key=UJT4ZI3HV5BCLQQ7&field=`
 - Read a Channel Feed:** `GET https://api.thingspeak.com/channels/2210102/feeds.json?results=2`
 - Read a Channel Field:** `GET https://api.thingspeak.com/channels/2210102/fields/1.json?result=`

Figure 5.2.11 API Keys of a Channel

Figure 5.11 shows the API Keys and the API Requests to read and write data of this channel. The Write API Key is essential for microcontroller to send data to this channel. The API Requests is essential for React Native Front End from User Interface Module and Laravel Back End from Data Processing Module to retrieve the collected Water Quality data.

5.2.2.1.3 Channel Settings

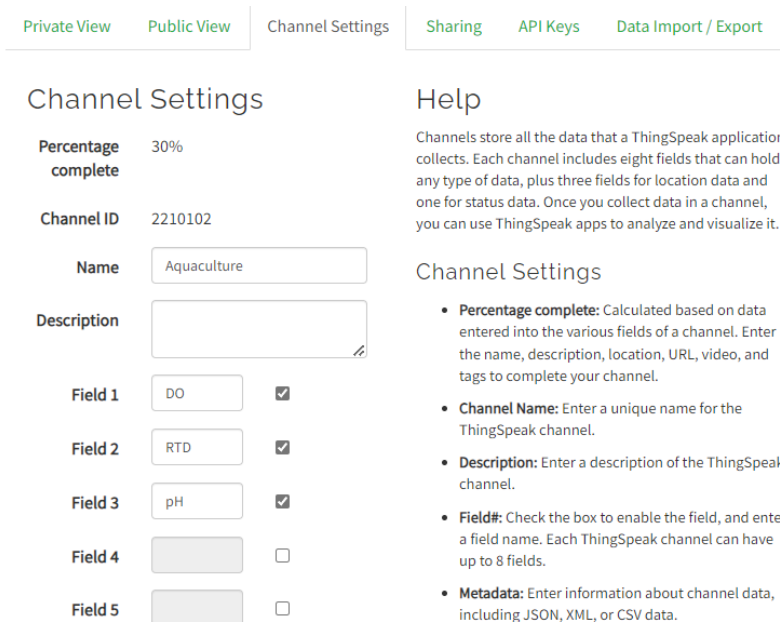


Figure 5.2.12 Channel Settings

Figure 5.12 is the settings of a ThingSpeak Channel, the fields is the column of a database. Therefore, we need to define the fields with the water parameters we have to monitor in this project. The microcontroller should send the data to the correct fields later.

5.2.2.1.4 Public view

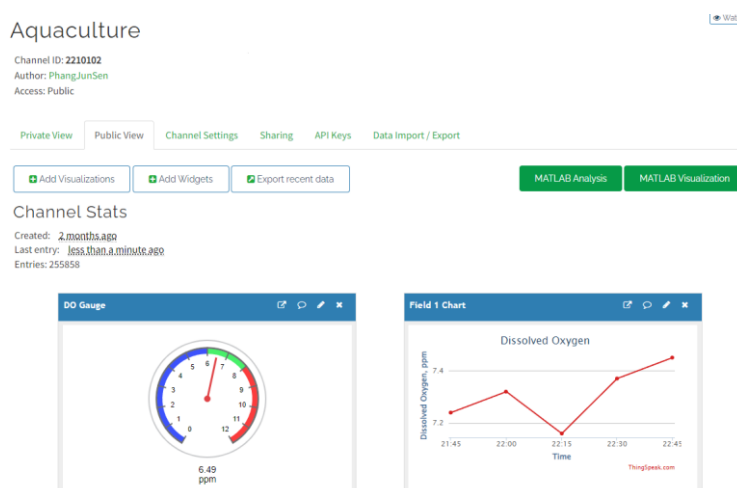


Figure 5.2.13 Public View of a Channel

Figure 5.12 shows the public view of a channel, which is required for setting up gauges and line graphs of each water parameter. This setup is crucial because the application will directly use the channel ID and their corresponding chart links to retrieve the gauges and line graphs in the public view and display them

in the dashboard. If no gauges and line graphs are prepared in public view, the dashboard will not display these elements.

5.2.2.2 MathWorks MATLAB

5.2.3 Data Processing Module

5.2.3.1 Laravel Framework Back End Architecture

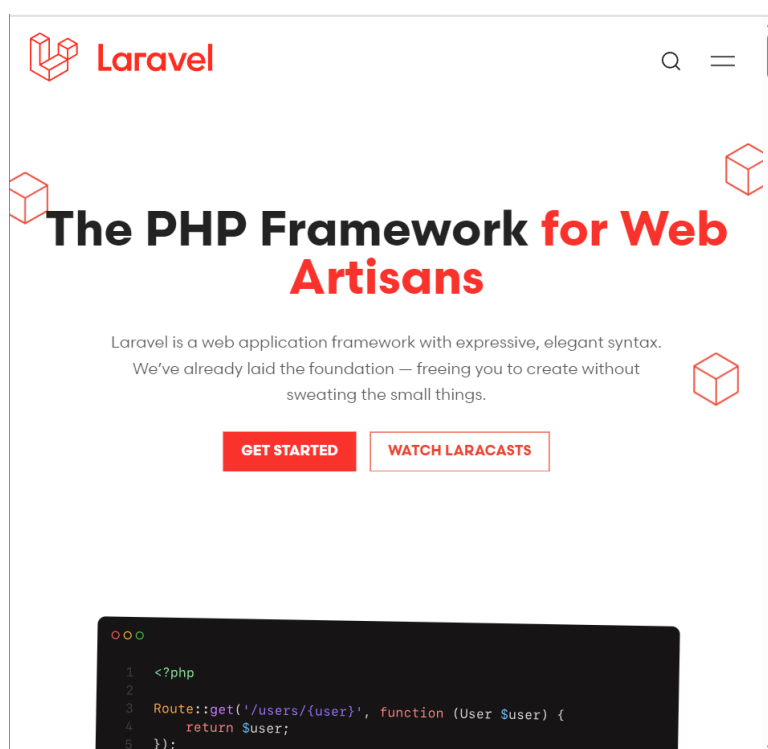


Figure 5.2.14 Laravel, a PHP framework (Otwell, n.d.)

Laravel is a PHP framework used to build web applications, it is known for its expressive and elegant syntax (Otwell, n.d.). In addition, Laravel can be used to build a reliable backend API for a mobile application. This statement is supported with the fact that it makes the development tasks easier and more efficient with the robust set of tools it provides and well-structured architecture. Laravel uses the Model-View-Controller (MVC) architectural pattern that separates the application's logic into 3 distinct components:

1. Model:

Represent the application's data and business logic. It interacts with the database, defines the data structure and the relationships between different data entities.

2. View:

Represents the user interface and the presentation layer of the application. Views are responsible for rendering data and displaying it to the user.

3. Controller:

Represent an intermediary between the Model and View. Controllers handle incoming HTTP requests, process the data, and determine which view to display.

Laravel Framework's architecture is complete and encompasses both the development of front-end and back-end together, as shown by the MVC architecture explained earlier. However, it does not support the development of mobile application, and thus we only utilize the Model and Controller in Laravel framework to build our back end application. Then, we use React Native to build mobile application and act as front end to replace the View.

A Laravel back-end application include these key features:

1) Artisan:

a robust command-line tool called Artisan for automating common development tasks, such as code generation, database migration and running tests.

2) Eloquent ORM:

Eloquent is Laravel's built-in Object-Relational Mapping (ORM) system, more simplified way to work with databases using simple and expressive syntax.

3) Routing:

Laravel offers a clean and elegant way to define web routes, allows managing HTTP requests and create RESTful APIs.

4) Authentication and Authorization:

Laravel provides an easy authentication system setup, allows developer to implement user registration, login, and password reset functionalities efficiently. It also offers a robust authorization mechanism for controlling access to specific parts of the application.

5) Database migrations:

Laravel's migration system allows for version control of database schemas, simplifying database structure changes and data management.

6) Testing Support:

Laravel is equipped with support for testing using PHPUnit, allowing developers to write unit and integration tests for their applications.

Above are the features that is used to build the Laravel back-end application.

5.2.3.2 Python

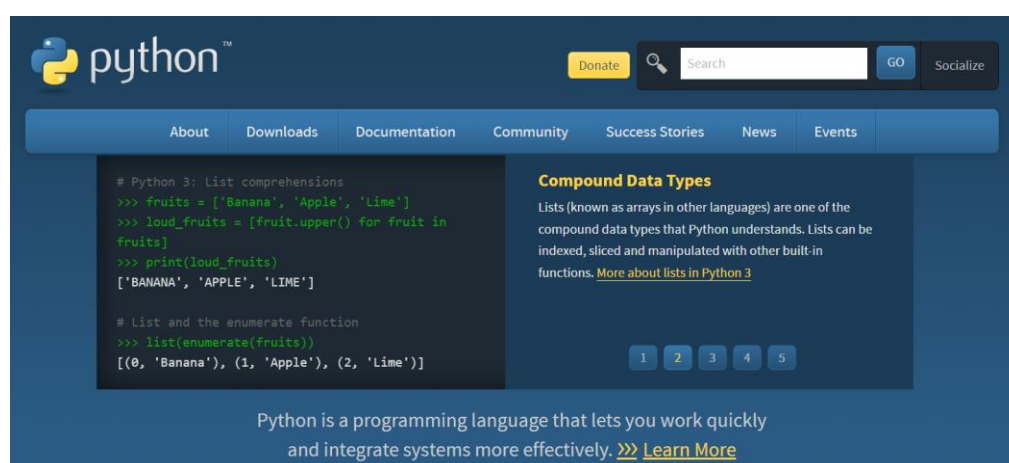


Figure 5.2.15 Python

Python is used in this project because it is a powerful tool to perform machine learning tasks in Laravel back-end server. It provides these key features:

1) Rich Ecosystem of Libraries:

Python has a large ecosystem of libraries and frameworks designed for machine learning.

2) Cross-Platform Compatibility:

Python is available on various operating systems, making it easy to develop and deploy machine learning solutions across different platforms.

3) Scalability:

Python can be used for small-scale and large-scale machine learning projects.

4) Machine Learning Frameworks:

Python allows developers to work with various machine learning frameworks and tools, ensuring the flexibility in choosing the correct tool for the task.

5.2.3.3 WampServer

The image shows the homepage of WampServer. At the top, there is a dark navigation bar with the WampServer logo (a 'W' in a circle) and the text 'WampServer Apache, PHP, MySQL sous Windows'. To the right of the logo are several navigation buttons: 'START', 'DOWNLOAD', 'TRAINING', 'FORUM', and 'CLOUD HOSTING'. In the top right corner, there are language options: 'FRANÇAIS' and 'РУССКИЙ'. Below the navigation bar, the main content area features a large, bold title 'WAMPSEVER,' with a red stamp that says 'CONTRIBUTION ALTER WAY'. Underneath the title is the subtitle 'a Windows web development environment.' and a paragraph of text: 'WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your databases.' To the right of the text is a cartoon illustration of a scientist with grey hair, glasses, and a white lab coat, holding a flask with purple liquid and a test tube with blue liquid. Below this section is a yellow banner with the text 'START WITH WAMPSEVER' and a small image of three beakers (green, blue, and purple). At the bottom of the banner, there is a short paragraph: 'WampServer installs automatically all you need to start developing web applications and is very intuitive to use. You will be able to tune your server without even touching the setting files.'

Figure 5.2.16 WampServer (Bourdon, n.d.)

WampServer is a Windows web development environment that this project can utilize it to create a web applications (Bourdon, n.d.). Its key features are:

1) Apache Web Server:

Apache is a popular open-source web server that allows developer to host and server web application locally.

2) MySQL Database:

MySQL is a widely used open-source relational database management system, it allows developers to create, manage, and interact with databases for your web applications.

3) PHP:

PHP is a server-side scripting language commonly used for web development. It allows developers to writ dynamic web applications and scripts that run on the server.

4) phpMyAdmin:

phpMyAdmin is a web-based database administration tool for managing MySQL databases. It provides user-friendly interface for tasks like database creation, table management, and data manipulation.

5) Development and Testing:

It allows developers to build and test web applications locally before deploying them to production servers.

5.2.4 User Interface Module

5.2.4.1 React Native Front End Architecture



Figure 5.2.17 React Native, a JavaScript framework

React native is a JavaScript framework that use to build mobile applications. It is open source and is well known of its code reusability, supported with large ecosystem of libraries and have fast development cycle. React Native front-end application consist of this these key features and components in its architecture:

1. Components:

The archirecture of React Native is component-based. Developers can create their own UI components, such as buttons, text input fields, and navigation elements. These created components are reusable throughout the app.

2. JSX

JSX is a syntax extension for JavaScript encourage developers to create UI components in a descriptive and clear manner.

3. React Navigation

A library that supports navigation of the React Native application. It contains stack navigation, tab navigation, and drawer navigation. This is necessary for creating the app's navigation structure.

4. State Management

In order to manage the state and data flow in React Native applications, Mobx or Redux can be used to manage the app's state and data flow. This is required to ensure the UI is in sync with the application's data.

5. API Integration

As it is front-end application, it is needed to communicate with back-end services and APIs. Therefore, react native apps can use the built-in 'fetch' function or popular libraries such as Axios to make HTTP requests and handle responses.

6. Styling

React Native applications using CSS to style components for the design of UI that needed to be display in the application.

7. Third-Party Libraries

React Native has a large ecosystem of third-party libraries and packages and make the development easier and faster.

8. Native Modules

For tasks that are not supported by native functionality, React native allows developers to custom native modules in Java and bridge them to JavaScript. For example, background service functionality.

5.2.4.2 Android Studio

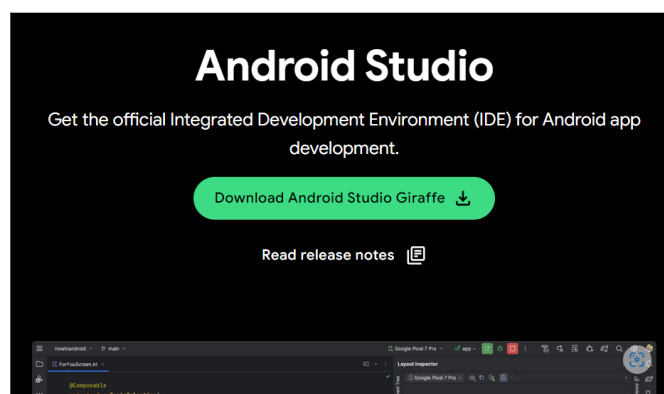


Figure 5.2.18 Android Studio

Android Studio is the official integrated development environment (IDE) for Android App development. Its main usage is to build, test and debug android applications. The key features of Android Studio that will be used in this project are:

1. Emulator:

Android studio has built-in android emulator that allows the developer to test their apps in computer using the virtual android devices. It provides various Android versions and device configurations, which allows developers to test their app on different screen sizes and resolutions.

2. Debugger:

A powerful debugger features that helps developers to identify and fix bugs in their code, such as real-time debugging and inspection of variables, breakpoints and logs.

3. App Signing:

Android Studio provides tools for signing and packaging Android apps for register it on the Google Play Store for other user to download the application.

5.3 Conclusion

The Water Quality Monitoring System architecture comprises of four essential modules: the Data Acquisition Module, Communication Module, User Interface Module, and Data Processing Module. Each of these modules is designed with robust tools and components to ensure their successful operation. The integration of these well-designed modules culminates in the creation of a functional and comprehensive system.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

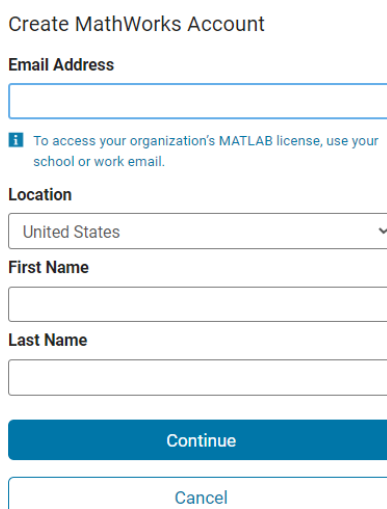
This chapter will discuss the system implementation of Water Quality Monitoring System. System implementation is an important phase of the software development lifecycle, it brings the planned designs and concepts into reality and make it contribute to its own objectives. Therefore, this chapter will discuss from project setup to system deployment. Besides, this chapter also discuss how the front-end User Interface module works with back-end Data Processing Modules to fulfill the use cases and requirements specification discussed in chapter 4.

6.2 Project Setup

This section discuss how to setup the Data Acquisition Module, Communication Module Setup, Data Processing Module, and the User Interface Module

6.2.1 Communication Module Setup

In order to setup ThingSpeak Cloud Server to store the sensors data, first we need to create a MathWorks account.



The image shows a web form titled "Create MathWorks Account". It contains the following fields and elements:

- Email Address:** A text input field.
- Information:** A blue icon with an 'i' followed by the text: "To access your organization's MATLAB license, use your school or work email."
- Location:** A dropdown menu currently showing "United States".
- First Name:** A text input field.
- Last Name:** A text input field.
- Buttons:** A blue "Continue" button and a white "Cancel" button with a blue border.

Figure 6.2.1 Register Mathworks account

After successfully register an account, we need to login to your account, and ThingSpeak will automatically redirect you to My Channels page.

ThingSpeak™

My Channels

New Channel

Search by tag

Name ↕	Updated ↕
🔒 Aquaculture	2023-08-27 23:50

Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click **New Channel** to create a new ThingSpeak channel.

Click on the column headers of the table to sort by the entries in that column or click on a tag to show channels with that tag.

Learn to [create channels](#), explore and transform data

Figure 6.2.2 My Channels page

Then, we need to press the new channel button to create a new channel.

New Channel

Name

Description

Field 1

Field 2

Field 3

Field 4

Field 5

Field 6

Field 7

Field 8

Metadata

Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.

Figure 6.2.3 Create new channel

Fill in the necessary information, such as Name is Aquaculture, Field 1 is DO, Field 2 is RTD and Field 3 is pH. After fill in all these information, you can hit the save channel button at the pages below. After that you will get a new channel looks like this:

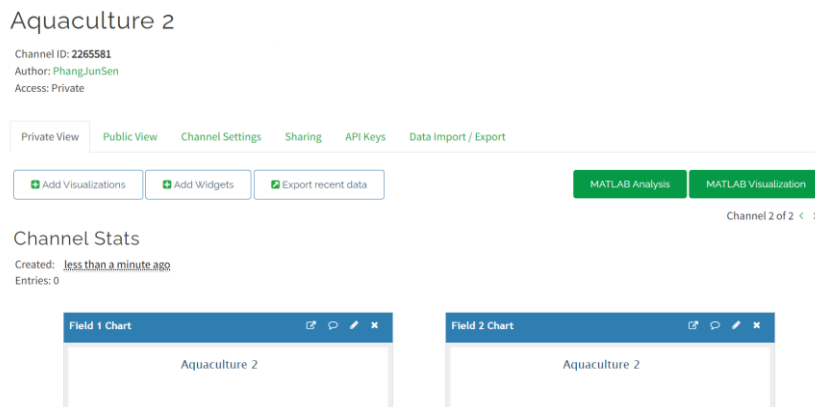


Figure 6.2.4 new channel page

It will automatically provides the line graphs based how many fields you defined in the previous steps. In our case, we have 3 fields, DO, RTD and pH, so there will be 3 default line graphs prepared.

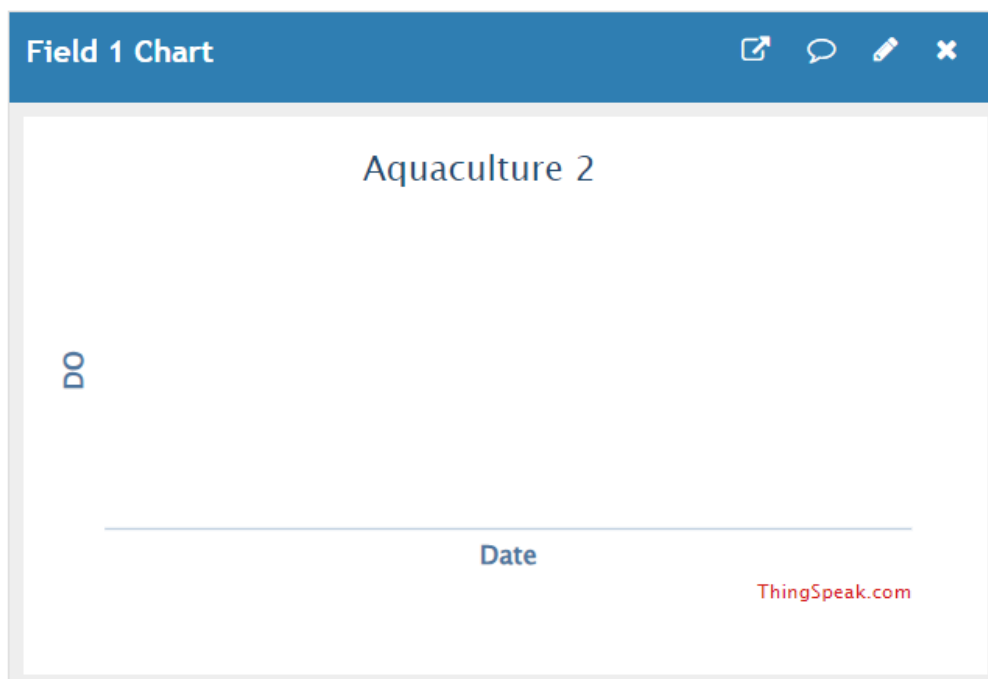


Figure 6.2.5 Field 1 line graph

There is no line graphs shown yet because there is no data received by the channels. Line graphs is for showing the trends of the water parameters, a gauge chart is to show the current water parameter value, and its safety range. In order to add a gauge chart, we need to press the “Add Widgets” button from figure 6.4 > choose Gauge > press “Next” > Fill in the gauge settings > save.



Figure 6.2.6 Add new gauge

Figure 6.6 shows the process to add the new gauge, we need to repeat it for RTD and pH gauge. Their settings are:

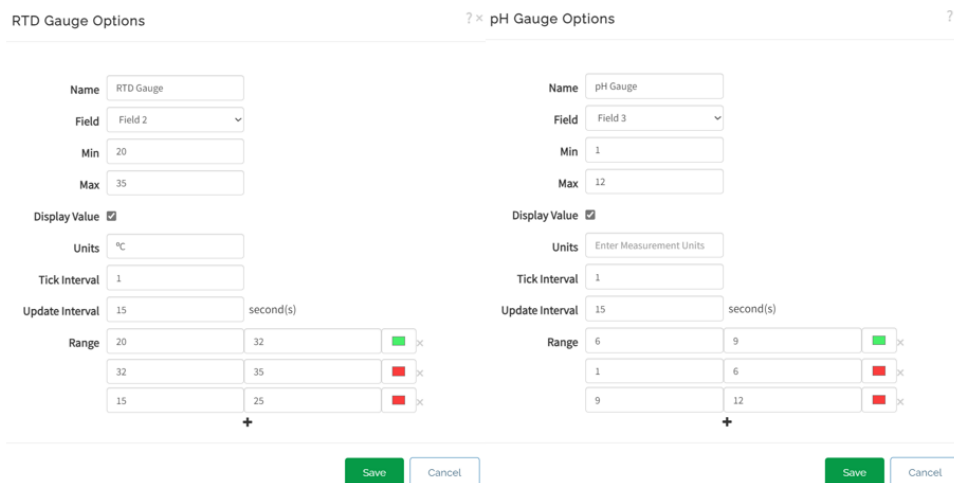


Figure 6.2.7 RTD and pH Gauge Options

After successfully added the gauges, the result will look like this:



Figure 6.2.8 DO, RTD and pH gauges

Finally, the Communication Module is all set now, but it still needs to integrate with Data Acquisition Module in order to get the data to store in the cloud database.

6.2.2 Data Acquisition Module Setup

In order to setup this module, we first need to prepare all the components as stated in Chapter 5:

- 1) Microcontroller:
 - ESP8266 microcontroller
- 2) Dissolved Oxygen Sensors:
 - Atlas Scientific Industrial Dissolved Oxygen Probe
 - EZO™ Dissolved Oxygen Circuit ESP8266 microcontroller
- 3) pH & Temperature Sensors:
 - Industrial pH/ORP/Temp Probe
 - EZO™ RTD Temperature Circuit
 - EZO™ pH Circuit

Firstly, we need to connect these components to become a complete electronic circuit using connector such as:

- Jumper Wires
- SMA Female (30cm) cable
- DC Plug 2.1mm C/W 2 way Green Terminal Block
- DC Jack C/W 2 way Green Terminal Block



Figure 6.2.9 Data Acquisition Module components connection

Figure above shows the correct connection of the Data Acquisition Module using all the components prepared.

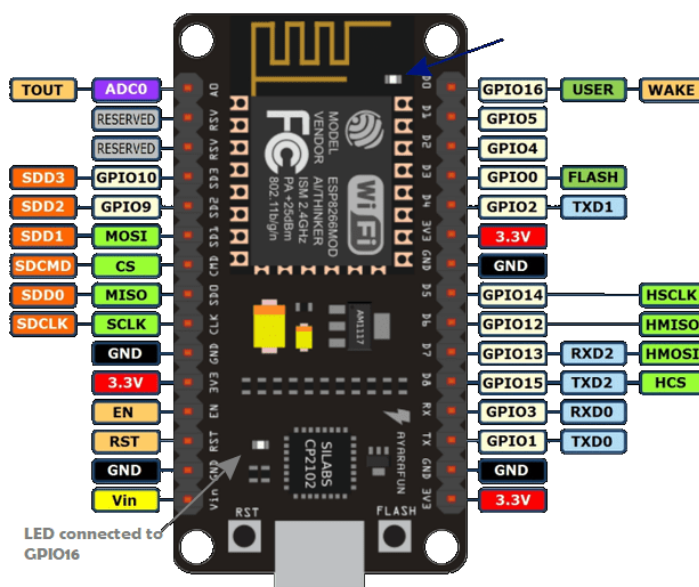


Figure 6.2.10 ESP8266 pins



Figure 6.2.11 EZO circuit pins



Figure 6.2.12 DO probe central wires and outer wires

Above shows the wires of the sensors and pins of EZO circuit and microcontroller. First, jumper wires are used to connect between EZO circuits, microcontroller and sensors:

- 1) TX from EZO connects to RX from microcontroller
- 2) RX from EZO connects to TX from microcontroller
- 3) GND from EZO connects to GND from microcontroller
- 4) VCC from EZO connects to 3.3V from microcontroller
- 5) Central wire from sensors to PRB from microcontroller
- 6) Outer wire from sensors to PGND from microcontroller



7)

8) Figure 6.2.13 Male SMAs of Industrial pH/ORP/Temp Probe

Due to Industrial pH/ORP/Temp Probe only provides Male SMA wires, we need to use a Female SMA cable to convert it to raw wires which contains central and outer wires.

Next, we need to program the ESP8266 to perform the task of sending the collected data at intervals of 15s once. Therefore, we need to use data cable to connects ESP8266 with a computer that already installed Arduino IDE. Then below is the Arduino code, with comments to explain how the code works to complete the task:

```
#include <ThingSpeak.h> //to interact with thingspeak server
#include <ESP8266WiFi.h> //to connects to wifi
#include <SoftwareSerial.h> //to communicate with sensors
#define rx_D0 D1 //This is rx pin for D0
#define tx_D0 D2 //This is tx pin for D0
#define rx_RTD D3 //This is rx pin for RTD
#define tx_RTD D5 //This is tx pin for RTD
#define rx_pH D6 //This is rx pin for pH
#define tx_pH D7 //This is tx pin for pH

const int ARRAY_SIZE = 5; // Define the size of the array
SoftwareSerial myserial_D0(rx_D0, tx_D0); //define how the soft
serial port is going to work with D0 sensors using D1,D2
SoftwareSerial myserial_RTD(rx_RTD, tx_RTD); //define how the soft
serial port is going to work with RTD sensors using D3,D5
```

```

SoftwareSerial myserial_pH(rx_pH, tx_pH); //define how the soft
serial port is going to work with pH sensors using D6,D7

String inputstring = ""; //a string to hold incoming data from the PC
String sensorstring_DO = ""; //a string to hold the data from the DO
sensors
String sensorstring_RTD = ""; //a string to hold the data from the
RTD sensors
String sensorstring_pH = ""; //a string to hold the data from the pH
sensors
boolean input_string_complete = false; //have we received all the
data from the PC
boolean sensor_string_complete_DO = false; //have we received all the
data from DO sensors
boolean sensor_string_complete_RTD = false; //have we received all
the data from RTD sensors
boolean sensor_string_complete_pH = false //have we received all the
data from the pH sensors

boolean run_DO = true; // controls to read DO data
boolean run_RTD = false; // controls to read RTD data
boolean run_pH = false; // controls to read pH data

float DO; //used to hold a floating point number that is the DO
float RTD; //used to hold a floating point number that is the RTD
float pH; //used to hold a floating point number that is the pH

WiFiClient client; // create a WiFiClient object

// ThingSpeak information
char thingSpeakAddress[] = "api.thingspeak.com"; // ThingSpeak
address
unsigned long myChannelNumber = 2210102; // Channel ID
char* readAPIKey = "TRYX3FPTI2HU1ZG1"; // Channel's read API key
char* myWriteAPIKey = "UJT4ZI3HV5BCLQQ7"; // Channel's write API key

// write Water Parameters into ThingSpeak database columns
unsigned int dataField_DO = 1; // write DO data in field1
unsigned int dataField_RTD = 2; // write temperature data in field2
unsigned int dataField_pH = 3; // write pH data in field3

//set up the hardware
void setup() {
  Serial.begin(9600); //set baud rate for the hardware serial port_0
to 9600
  myserial_DO.begin(9600); //set baud rate for the software serial
port to 9600

```

```

    myserial_RTD.begin(9600);//set baud rate for the software serial
port to 9600
    myserial_pH.begin(9600);//set baud rate for the software serial
port to 9600
    inputstring.reserve(10);//set aside some bytes for receiving data
from the PC
    sensorstring_DO.reserve(30);//set aside some bytes for receiving
data from DO sensor
    sensorstring_RTD.reserve(30);//set aside some bytes for receiving
data from RTD sensor
    sensorstring_pH.reserve(30);//set aside some bytes for receiving
data from pH sensor
    ThingSpeak.begin(client); // Initialize ThingSpeak

//setup wifi connection
//+++++
WiFi.begin("WIFI_NAME", "WIFI_PASSWORD");
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}
Serial.println();

Serial.print("Connected, IP address: ");
Serial.println(WiFi.localIP());
//-----
}

// Receive commands from PC to sensors.
void serialEvent() {//if the hardware serial port_0 receives a char
    inputstring = Serial.readStringUntil(13);//read the string until we
see a <CR>
    input_string_complete = true;//set the flag used to tell if we have
received a completed string from the PC
}

// The task is running here
void loop() {//here we go...

    if (input_string_complete == true) {//if a string from the PC has
been received in its entirety
        myserial_DO.print(inputstring);//send that string to the Atlas
Scientific product
        myserial_DO.print('\n');//add a <CR> to the end of the string
        inputstring = "";//clear the string
    }
}

```

```

    input_string_complete = false; //reset the flag used to tell if we
    have received a completed string from the PC
  }

  //read sensor data
  //starts to read DO sensor's value and the run_RTD and run_pH is
  set to false at the beginning to prevent the mix of reading
  if (run_DO && myserial_DO.available() > 0) { //if run_DO == true and
  myserial_DO.available() > 0 means this connection receives a
  character
    Serial.println("DO available: " + sensorstring_DO);
    char inchar_DO = (char)myserial_DO.read(); //get the char we just
    received
    sensorstring_DO += inchar_DO; //add the char to the var called
    sensorstring
    if (inchar_DO == '\r') { //if the incoming character is a <CR>
    means the reading is complete
      Serial.println("DO read done: " + sensorstring_DO);
      sensor_string_complete_DO = true; //set to true means the DO
      reading is read succesfully
      run_DO = false; // since the DO is read complete, then we
      should prevent it from reading anymore until the other water
      parameter is reading complete
      run_RTD = true; // set this to true, means that we should
      proceed to read RTD value now.
    }
  }

  //read sensor data
  if (run_RTD && myserial_RTD.available() > 0) { //if we see that the
  RTD sensors has sent a character and run_RTD is set to true
    Serial.println("RTD available: " + sensorstring_RTD);
    char inchar_RTD = (char)myserial_RTD.read(); //get the char we
    just received
    sensorstring_RTD += inchar_RTD; //add the char to the var called
    sensorstring
    if (inchar_RTD == '\r') { //if the incoming character is a <CR>
      Serial.println("RTD read done: " + sensorstring_RTD);
      sensor_string_complete_RTD = true; //set to true means the RTD
      reading is read successfully
      run_RTD = false; //set to false to stop read RTD value until it
      is set to true
      run_pH = true; //set to true means we can start to read pH
      value now
    }
  }
}

```

```

    if (run_pH && myserial_pH.available() > 0) { //if we see that the pH
sensor has sent a character, and run_pH is set to true
        Serial.println("pH available: " + sensorstring_pH);
        char inchar_pH = (char)myserial_pH.read();//get the char we just
received
        sensorstring_pH += inchar_pH;//add the char to the var called
sensorstring
        if (inchar_pH == '\r') { //if the incoming character is a <CR>
            Serial.println("pH read done: " + sensorstring_pH);
            sensor_string_complete_pH = true; //set to true means the pH
reading is read succesfully
            run_pH = false; //set to false stop reading pH value until it
is set to true again
        }
    }

    if (sensor_string_complete_DO == true && sensor_string_complete_RTD
== true && sensor_string_complete_pH == true) { //if
all water parameters reading is collected then returns true
        sensor_string_complete_DO = false; //set this back to start next
round of data collecting
        sensor_string_complete_RTD = false; //set this back to start next
round of data collecting
        sensor_string_complete_pH = false; //set this back to start next
round of data collecting
        DO = 0; //set this float to 0 to reset the reading
        RTD = 0; //set this float to 0 to reset the reading
        pH = 0; //set this float to 0 to reset the reading

        if (isdigit(sensorstring_DO[0])) { //if the first character in
the string is a digit
            DO = sensorstring_DO.toFloat(); //convert the string to a
floating point number so it can be evaluated by the Arduino
            sensorstring_DO = ""; //set this string to "" to reset the
reading
            Serial.println("DO convert done: sensorstring_DO = " +
sensorstring_DO);
            sensor_string_complete_DO = false; //set this back to start
next round of data collecting
            ThingSpeak.setField(1, DO); // use this to set the field1 to
store DO value
        }
        if (isdigit(sensorstring_RTD[0])) { //if the first character in
the string is a digit
            RTD = sensorstring_RTD.toFloat(); //convert the string to a
floating point number so it can be evaluated by the Arduino
            sensorstring_RTD = "";

```

```

        Serial.println("RTD convert done: sensorstring_RTD = " +
sensorstring_RTD);
        sensor_string_complete_RTD = false; //set this back to start
next round of data collecting
        ThingSpeak.setField(2, RTD); // use this to set the field2 to
store RTD value
    }
    if (isdigit(sensorstring_pH[0])) { //if the first character in
the string is a digit
        pH = sensorstring_pH.toFloat(); //convert the string to a
floating point number so it can be evaluated by the Arduino
        sensorstring_pH = "";
        Serial.println("pH convert done: sensorstring_pH = " +
sensorstring_pH);
        sensor_string_complete_pH = false; //set this back to false and
start the next round of data collecting
        ThingSpeak.setField(3, pH); // use this to set the field3 to
store RTD value
    }
}

Serial.println("DO: " + String(DO) + " | RTD: " + String(RTD) + "
| pH: " + String(pH)); //send that string to the PC's serial monitor
sensorstring_DO = "";
sensorstring_pH = "";
sensorstring_RTD = "";

// sends the fields data we defined before as a row to that
ThingSpeak channel using channel ID, and write API key
int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

if(x == 200){
    Serial.println("Channel update successful.");
}
else{
    Serial.println("Problem updating channel. HTTP error code " +
String(x));
}
// wait 15 seconds and start the next round of data collection
delay(15000);
// DO = 0;
// RTD = 0;
// pH = 0;
sensorstring_DO = "";
sensorstring_pH = "";

```



```

sensorstring_RTD = "";

run_DO = true; // set this to true to start the next round of
data collection of DO value as the opening
}
}

```

Upload this code to the ESP8266 to start the data acquisition module, then we can detach the data cable from the computer. Next, we need to connect the microcontroller to power source, and place the sensors into the water tank. Finally, we can start to observe the data collected by this module using the line graphs provided by the ThingSpeak channel.

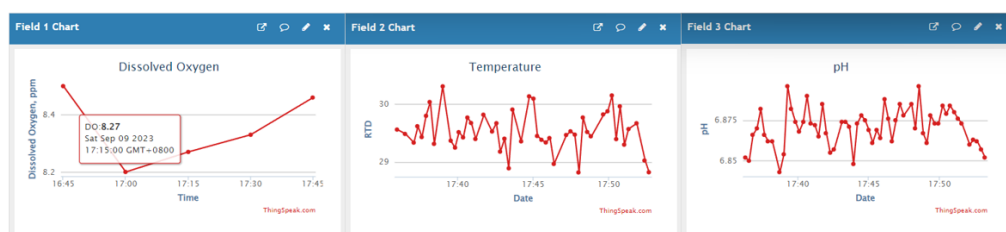


Figure 6.2.14 ThingSpeak channel's line graphs

If the line graphs showing the data is collected, means the data acquisition module is setup correctly. Notes that this steps can only be done when Communication Module is already done setting up.

6.2.3 Data Processing Module Setup

The Data Processing Module of water quality monitoring system is a Laravel back-end server. Due to the limitation of time, we may not be able to deploy the back-end server to a cloud-based environment or configure it as a web service accessible via a domain. Therefore, the back-end server will be deployed locally. This section will explain how to setting up, building, and running the existing Laravel back-end project to complete the Data Processing Module of water quality monitoring system.

6.2.3.1 Prerequisites

Before launching the Laravel back-end server, ensure the the device to run the server have the following prerequisites met:

1) WAMP Server:

Make sure the WAMP Server is installed and running on your local machine. This includes Apache, MySQL, and PHP components, their version are:

- a. WAMP Server – 3.3.0 – 64bit
- b. Apache – 2.4.54.2
- c. PHP – 8.2.0
- d. MySQL – 9.0.31

Figure below shows the WAMP Server and its components version that use to run the back-end server.

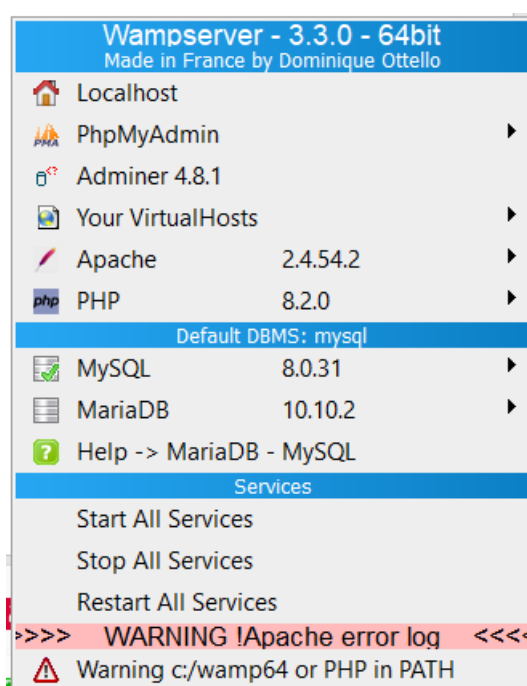


Figure 6.2.15 Wampserver version and its components

2) Composer:

Composer, the PHP dependency manager, should be installed on the local machine.

6.2.3.2 Project Setup

Step 1: Navigate to Project Directory

In order to run the project, first we need to use Visual Studio Code to open the project terminal. Figure below shows the Laravel back-end project source code open with Visual Studio Code.

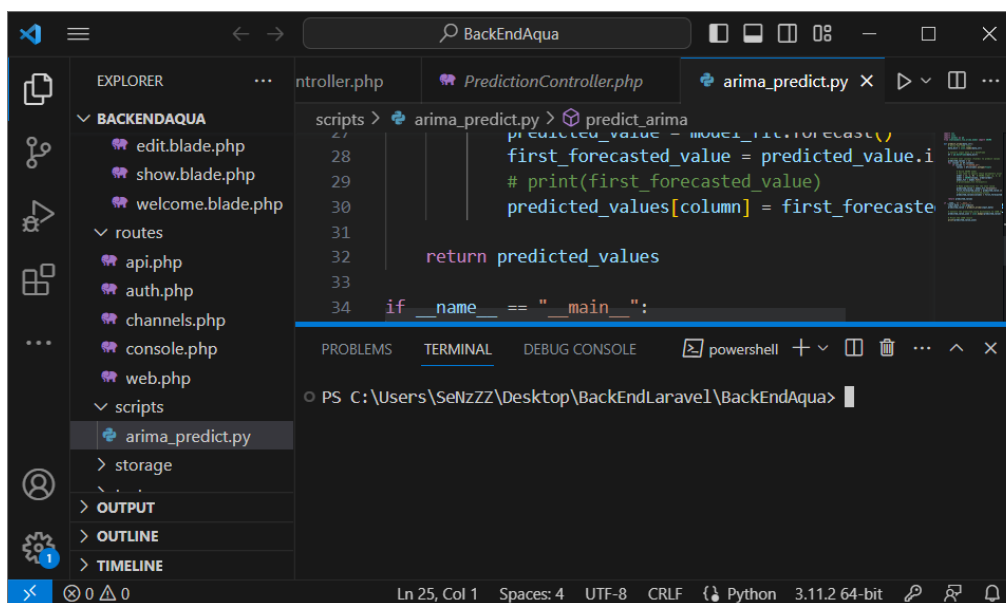


Figure 6.2.16 Laravel Back-end project source code

From the figure above, the terminal is opened and is navigated to the project directory.

Step 2: Install Dependencies:

Run the “composer install” command in the terminal to install project dependencies specified in the ‘composer.json’ file.

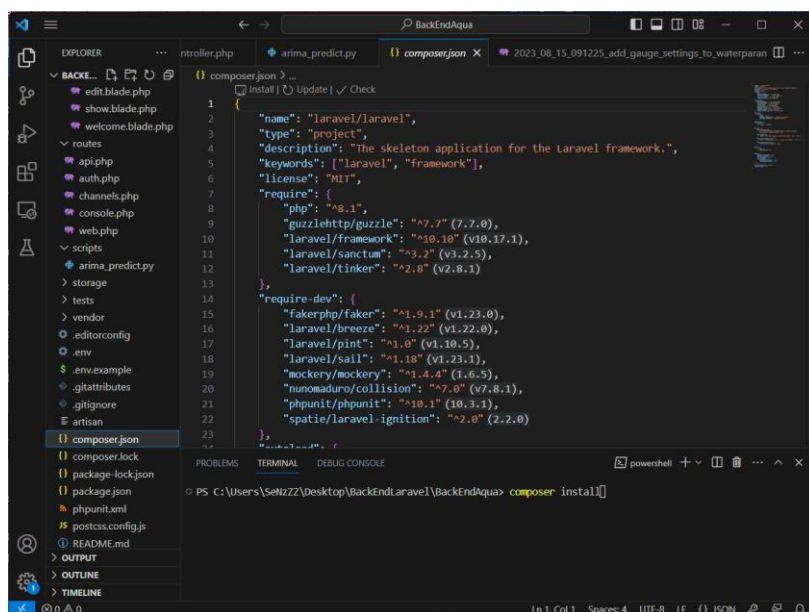


Figure 6.2.17 composer.json

6.2.3.3 Environment Configuration

Step 1: Environment File:

Ensure the .env file is placed in the project root. If not please copy and paste using the .env.example.

Step 2: Generate Application key:

Then, run the “php artisan key:generate” to generate a unique application key for the project.

6.2.3.4 Database Setup

Step 1: Database Creation

Open phpMyAdmin from the WampServer, and then create a new database named ‘aquafarm’.

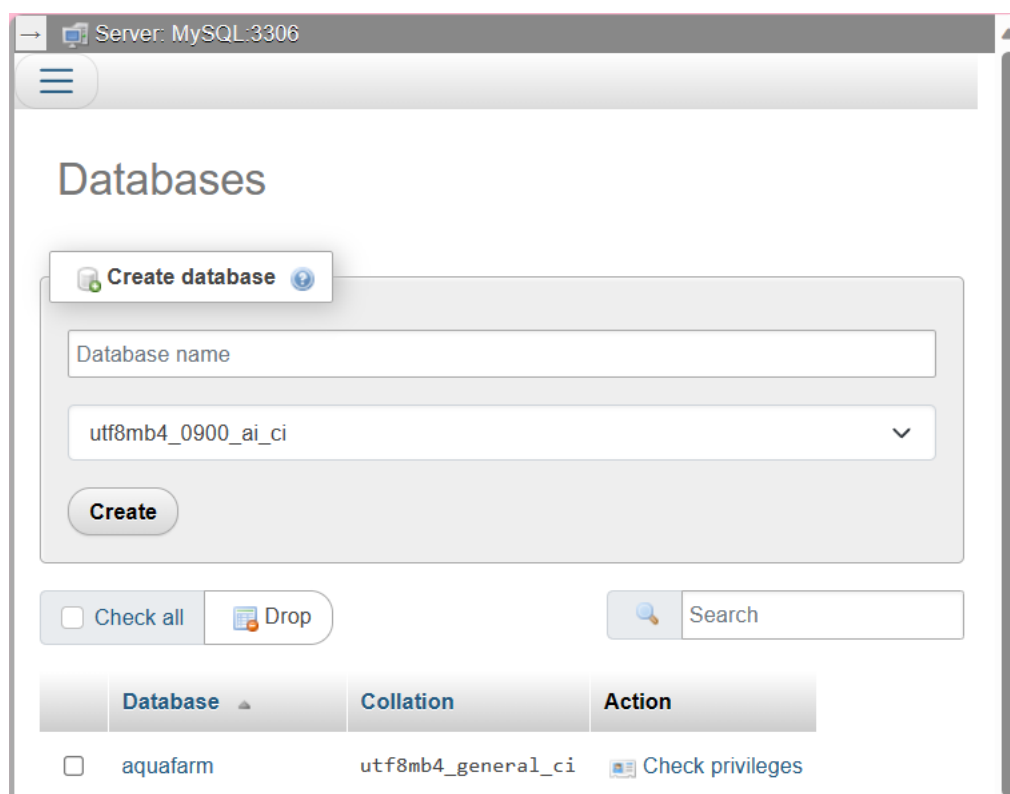


Figure 6.2.18 Create new database

Step 2: Database Migration

Then go back to the terminal and run “php artisan migrate” to apply the database migrations and create the necessary tables. Database migration files are located at [PROJECT DIRECTORY]\database\Migrations.

```

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('users', function (Blueprint $table) {
15             $table->id();
16             $table->string('name');
17             $table->string('email')->unique();
18             $table->timestamp('email_verified_at')->nullable();
19             $table->string('password');
20             $table->rememberToken();
21             $table->timestamps();
22         });
23     }
24
25     /**
26      * Reverse the migrations.
27      */

```

Figure 6.2.19 Database Migration Files

6.2.3.5 Starting the Laravel back-end server

Step 1: Start Laravel Server

Run this command “php artisan serve” to launch the server. This server will be host locally at ‘http://127.0.0.1:8000’.

```

PS C:\Users\SeNzZZ\Desktop\BackEndLaravel\BackEndAqua> php artisan serve

  INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server

```

Figure 6.2.20 php artisan server

After the terminal shows the Laravel Back-end server is hosted locally, it confirms that the successful setup of the Data Processing Module.

6.2.3.6 Conclusion

In conclusion, the local machine the run the Laravel back-end server should fulfill the prerequisites by having WAMP Server and Composer installed. Then need to open the project, and perform configuration of environment and database. Finally, serve the application to host it locally and the Data Acquisition Module is now prepared to perform its task.

6.2.4 User Interface Module Setup (React Native Android Application)

This section explain how to setup the User Interface Module of Water quality monitoring system. User interface Module is a mobile android application builded using React Native Framework. However, due to the limitation of time, we could not make it deployable on a real android mobile app. Instead, we will deploy it on the android emulator on a local machine.

6.2.4.1 Prerequisites

Before proceed to React Native Android Application setup, these are the prerequisites that the device running this project should be met:

1) Node.js and npm:

Ensure that Node.js and npm are installed on the device, because these 2 are the fundamental for managing JavaScript packages and running React Native commands. Run “node -v” to check whether Node.js is installed, and “npm -v” to verify npm.

```
PS C:\Users\SeNzZZ\Desktop\Aquaculture2\Aquaculture>node -v
v18.17.0
PS C:\Users\SeNzZZ\Desktop\Aquaculture2\Aquaculture>npm -v
9.8.1
```

Figure 6.2.21 node -v and npm -v

Figure above shows the Node.js and npm is installed in this device.

2) Java Development Kit (JDK):

Ensure that the Java Development Kit (JDK) version 8 or later is installed. The JDK is essential for Android app development. Run “java -version” to verify java.

```
PS C:\Users\SeNzZZ\Desktop\Aquaculture2\Aquaculture>java -version
java version "20.0.1" 2023-04-18
Java(TM) SE Runtime Environment (build 20.0.1+9-29)
Java HotSpot(TM) 64-Bit Server VM (build 20.0.1+9-29, mixed mode, sharing)
```

Figure 6.2.22 java -version

3) Android Studio and Android SDK Components:

Ensure Android Studio and Android SDK Components is installed including specific API Levels required for the project for Android App development and the Android Emulator.

Steps:

- 1) Launch Android Studio.
- 2) Open SDK Manager and build.gradle in react native project.

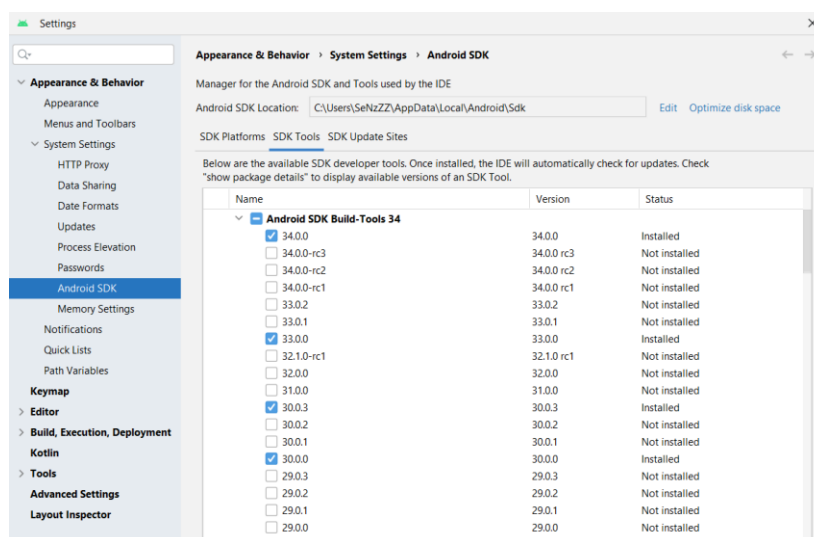


Figure 6.2.23 SDK Manager

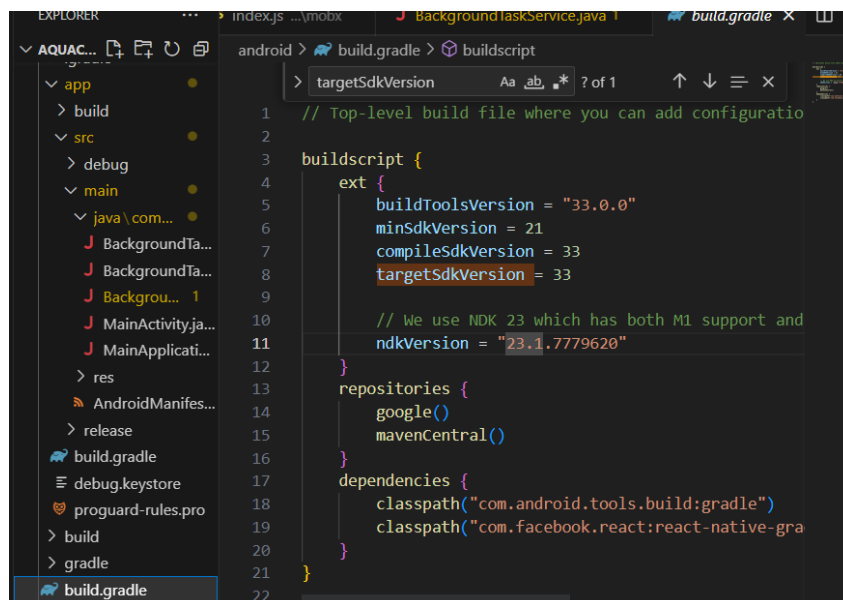


Figure 6.2.24 build.gradle

- 3) Verify the buildToolsVersion, compileSdkVersion and targetSdkversion is same.

4) React Native CLI:

Ensure React Native Command Line Interface (CLI) is installed globally via npm to facilitate project management.

```

PS C:\Users\SeNzZZ\Desktop\Aquaculture2\Aquaculture> react-native
react-native-cli: 2.0.1
react-native: 0.72.4

```

Figure 6.2.25 react-native -v

If all the prerequisites is passed, then we can now proceed to Project Initialization.

6.2.4.2 Open Existing React Native Project

Open the existing React Native Project.

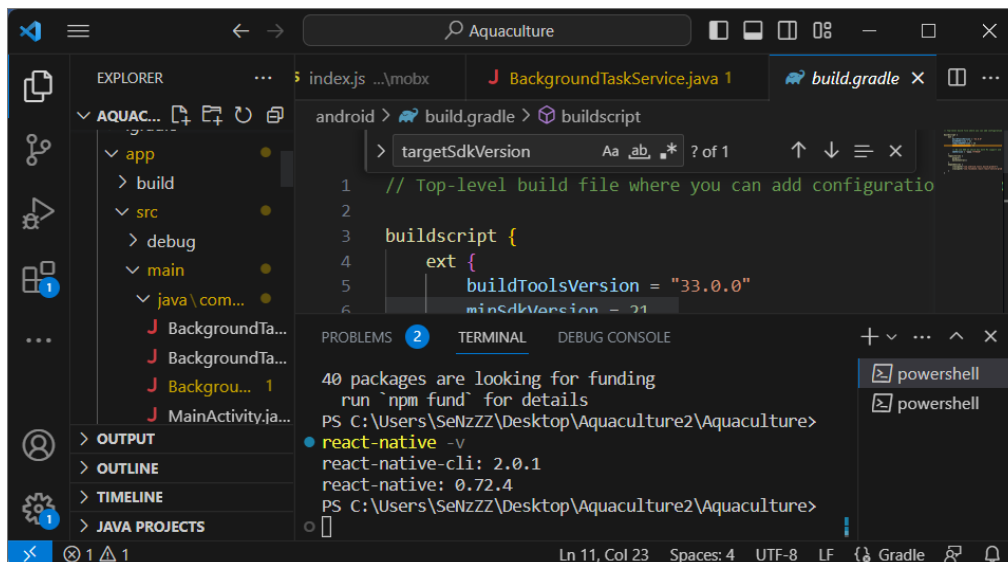


Figure 6.2.26 React Native Front End Project

6.2.4.3 Dependency Installation

Run “npm install” to install the dependencies specified in package.json.

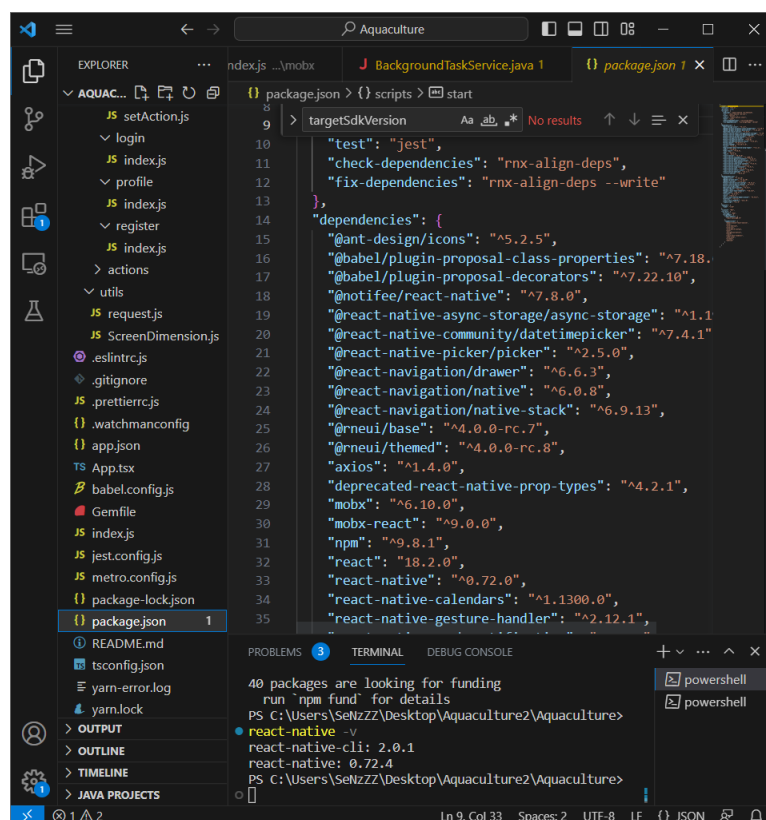


Figure 6.2.27 dependencies in package.json

6.2.4.4 Running on an Android Emulator

After that, run 2 terminals, first terminal is for “npm run start” to open the Metro.
Another terminal is for “npm run android”.

```

PS C:\Users\SeNzZZ\Desktop\Aquaculture2\Aquaculture>n
pm run start

> Aquaculture@0.0.1 start
> react-native start

Welcome to Metro v0.76.7
Fast - Scalable - Integrated

r - reload the app
d - open developer menu
i - run on iOS
a - run on Android

```

Figure 6.2.28 run Metro

```

See https://docs.gradle.org/8.0.1/userguide/command\_line\_interface.html#sec:command\_line\_warnings

BUILD SUCCESSFUL in 35s
263 actionable tasks: 13 executed, 250 up-to-date
info Connecting to the development server...
8081
info Starting the app on "emulator-5554"...
Starting: Intent { act=android.intent.action.MAIN cat
=[android.intent.category.LAUNCHER] cmp=com.aquacultu
re/.MainActivity }

```

Figure 6.2.29 npm run start

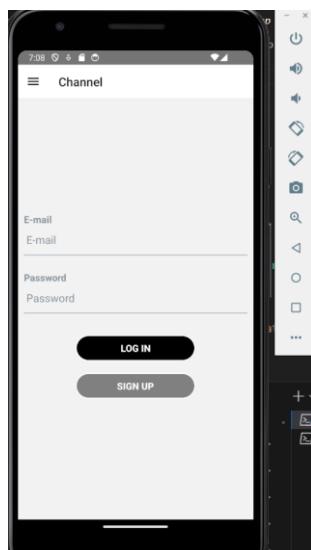


Figure 6.2.30 emulator is running

After the emulator is running, it means the User Interface Module is now completely set up.

6.3 Water Quality Monitoring Mobile Application functions

After completing the setup process of the Water Quality Monitoring System, we are now ready to utilize the Water Quality Monitoring Mobile application for monitoring water quality. This section comprehensively addresses the implementation of the features outlined in the use case description, as listed in Chapter 4. Additionally, each use case will be supported with a detailed breakdown of the code responsible for the functionality, encompassing both the front-end and the back-end.

6.3.1 Register

This section explain how user perform register function.

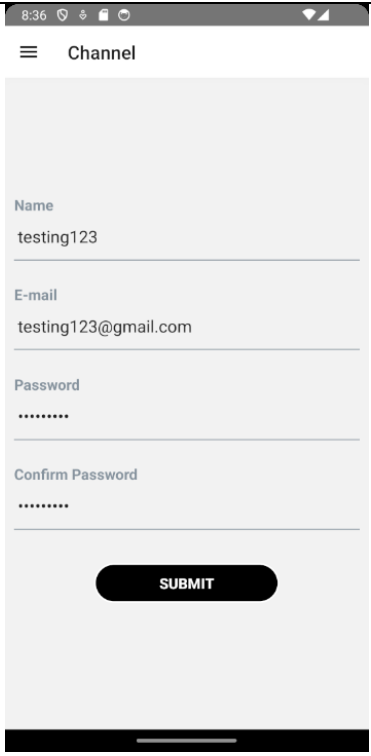
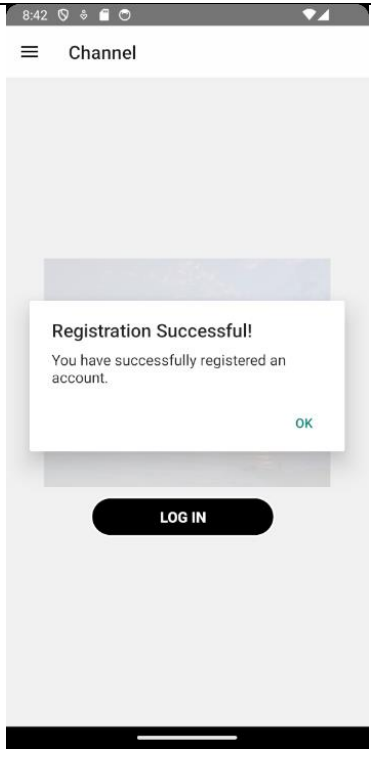
6.3.1.1 Feature demonstration

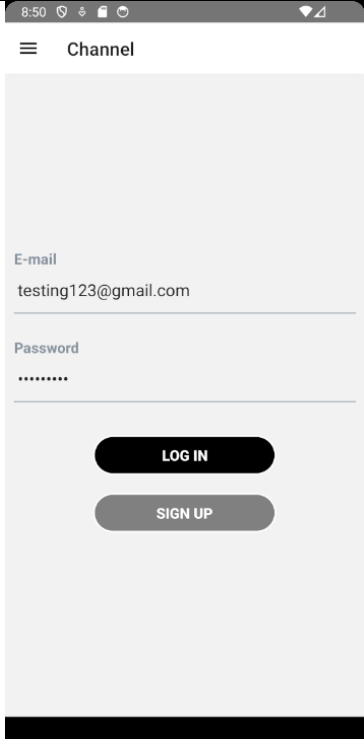
Below shows the process of using Register function of Water Quality Montoring mobile application.

Success Case:

Table 6.3.1 Register Function Success Case

No.	Mobile Screen	Process
-----	---------------	---------

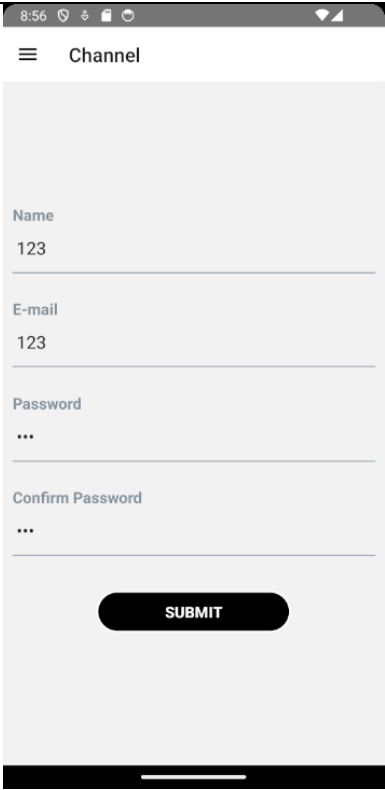
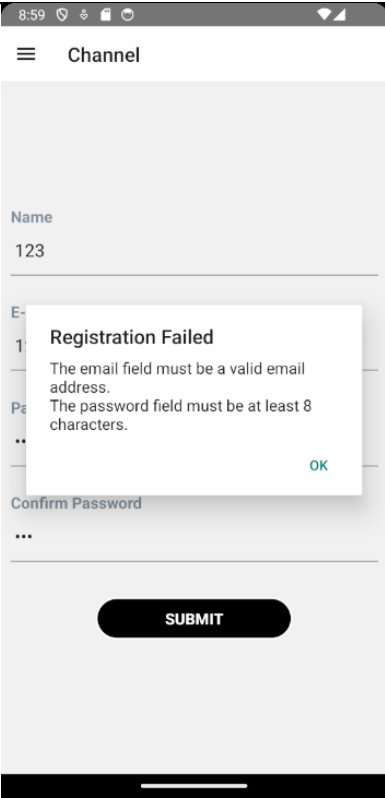
1	 <p>Figure 6.3.1 Success Register page</p>	<ol style="list-style-type: none">1) User go to the 'Register Page'.2) User can see the account registration page with the registration form and the submit button.3) User then fill in the Name, E-mail, Password and Confirm Password correctly.4) User click the 'SUBMIT' Button
2	 <p>Figure 6.3.2 Registration Successful</p>	<ol style="list-style-type: none">1) Then, user can see a Registration Successful Message pops out.2) User click on the 'OK' button

3	 <p>Figure 6.3.3 Redirect to Login Page</p>	<ol style="list-style-type: none"> 1) Then user will be redirect to “Login” page. 2) The login credentials will be fill in automatically.
---	--	---

Failure Case:

Table 6.3.2 Register Function Failure Case

No.	Mobile Screen	Process
-----	---------------	---------

1	 <p>Figure 6.3.4 Fail Register page</p>	<ol style="list-style-type: none"> 1) User go to the 'Register Page'. 2) User can see the account registration page with the registration form and the submit button. 3) User then fill in the INVALID E-mail and Password correctly. 4) User click the 'SUBMIT' Button
2	 <p>Figure 6.3.5 Registration Fail</p>	<ol style="list-style-type: none"> 1) Then, user can see a Registration Failed Message pops out. 2) User click on the 'OK' button 3) Then user need to fill in the registration form again.

6.3.1.2 Code explanation

Explanation starts from the view from the front-end mobile application, and then how it triggers the Laravel back-end function to perform registration.

React Native (Front-End) – Aquaculture\src\pages\account\register\index.js

```

{!this.state.complete &&
  <View style={{ width: screenWidth, alignItems: 'center', justifyContent:
'center' }}>

    <Input
      label="Name"
      placeholder='Name'
      onChangeText={this.NameChangeText}
    />
    <Input
      label="E-mail"
      placeholder='E-mail'
      onChangeText={this.EmailChangeText}
    />
    <Input
      label="Password"
      placeholder='Password'
      onChangeText={this.PasswordChangeText}
      secureTextEntry={true}
      textContentType="password"
    />
    <Input
      label="Confirm Password"
      placeholder='Confirm Password'
      onChangeText={this.ConfirmPasswordChangeText}
      secureTextEntry={true}
    />
    <Button
      title="SUBMIT"
      buttonStyle={{
        backgroundColor: 'black',
        borderWidth: 2,
        borderColor: 'white',
        borderRadius: 30,
      }}
      containerStyle={{
        width: 200,
        marginHorizontal: 50,
        marginVertical: 10,
      }}
      titleStyle={{ fontWeight: 'bold' }}
      onPress={this.SubmitButtonClick}
    />
  </View>
}

```

Figure 6.3.6 React Native registration form

Code snippet above shows the registration form, this form shows everytime when user navigate to this page, when user click the “SUBMIT” button, it triggers the “this.SubmitButtonClick” function.

```

SubmitButtonClick = async () => {
  const userData = {
    name: this.state.name,
    email: this.state.email,
    password: this.state.password,
    password_confirmation: this.state.confirmpassword,
  };
}

```



```

    try {
      const response = await axios.post('http://10.0.2.2:8000/api/register',
      userData);
      console.log(response.data);

      const complete = response.data.complete;
      const message = response.data.message;
      const user = response.data.user;

      this.setState({ complete, message, user });

      console.log(this.state.complete);
      console.log(this.state.message);
      console.log(this.state.user);
      Alert.alert(
        'Registration Successful!',
        'You have successfully registered an account.',
        [
          {
            text: 'OK', onPress: () => this.props.navigation.navigate('Login',
            { login_credentials: this.state })
          },
        ],
        { cancelable: false }
      );
    } catch (error) {
      let errorMessage = 'An error occurred during registration';

      if (error.response) {
        const responseData = error.response.data;
        if (responseData && responseData.errors) {
          errorMessage = Object.values(responseData.errors).join('\n');
        }
      } else if (error.message) {
        errorMessage = error.message;
      }

      Alert.alert(
        'Registration Failed',
        errorMessage,
        [
          { text: 'OK', onPress: () => console.log('OK Pressed') },
        ],
        { cancelable: true }
      );
    }
  }
}

```

Figure 6.3.7 SubmitButtonClick

Then SubmitButtonClick function first store the user registration information into a object called userData. Then it try to call axios post method to call the back-end server API with the routes ('http://10.0.2.2:8000/api/register') and sends the userData object as the request payload. Then this post request will be handle by the data processing module, the Laravel back-end server as shown below.

```

Route::middleware('api')->group(function () {
    ...
    Route::post('api/register', [RegisteredUserController::class, 'store']);
    ...
});

```

Figure 6.3.8 route::api/register – POST

This is the route that handles the request, so this route corresponds to the controller function from RegisteredUserController, named ‘store’ as shown in figure below.

```
public function store(Request $request): JsonResponse
{
    $validator = Validator::make($request->all(), [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'confirmed', Rules\Password::defaults()],
    ]);

    // Check if validation fails
    if ($validator->fails()) {
        return response()->json([
            'errors' => $validator->errors(),
        ], 422); // 422 Unprocessable Entity status code
    }

    // Attempt to create the user
    try {
        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        event(new Registered($user));

        // Return a JSON response indicating successful registration
        return response()->json([
            'message' => 'Registration successful',
            'user' => $user,
            'complete' => true,
        ]);
    } catch (\Exception $e) {
        // Return an error response if an exception occurs during user creation
        return response()->json([
            'message' => 'Registration failed',
            'error' => $e->getMessage(), // You can customize the error message
            here
        ], 500); // 500 Internal Server Error status code
    }
}
```

Figure 6.3.9 RegisteredUserController store function

This store function will first starts by validate the incoming HTTP request data by checking whether the name, email, password and confirm password is valid and correct. Then if the validation fail, it will immediately return a JSON response showing there is a validation errors with a status code of 422, indicating that the request data is invalid. If the validation is not fail, then it will try to create a new user record in database using ‘User::create’ method, and it also hash the ‘password’ for security purpose using “Hash:make” method. After the user is successfully created in database, it will return a JSON response

indicating successful registration. If there is any exception occurs during the user creating process, it will also return a JSON response indicating the registration has failed with the HTTP status code 500.

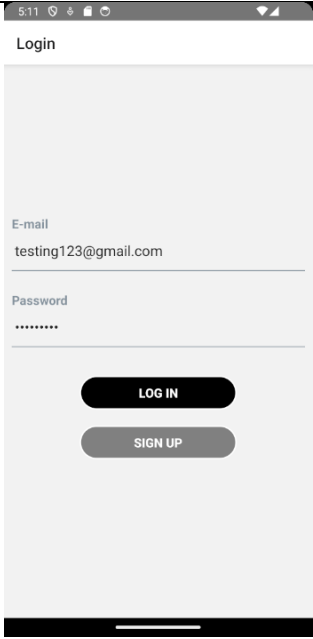
Then if the JSON response is returned without any exception, the SubmitButtonClick function will continue to set the state using the returned JSON response from the “store” function. Then the Registration successful message will be shown. Then if user click “OK” button, it will navigate user to login page, with the “login_credentials” as parameter to fill in the login credentials at the login page automatically. If the JSON response is an exception, the exception error will be catch and then a “Registration Failed” alert box will be shown with the error message returned. When user click OK, then closed the “Registration Failed” alert box.


6.3.2 Login

6.3.2.1 Feature demonstration

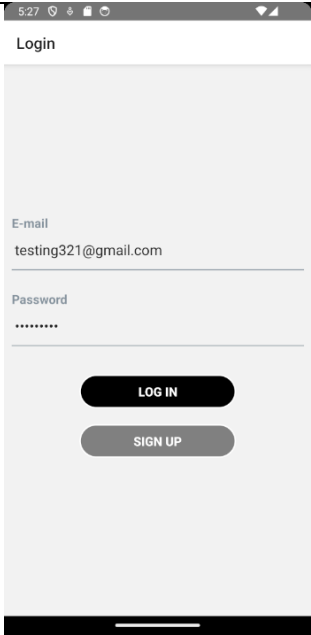
Below shows the process of using Login function of Water Quality Monitoring mobile application.

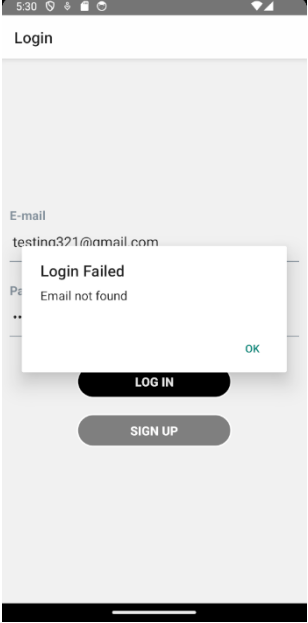
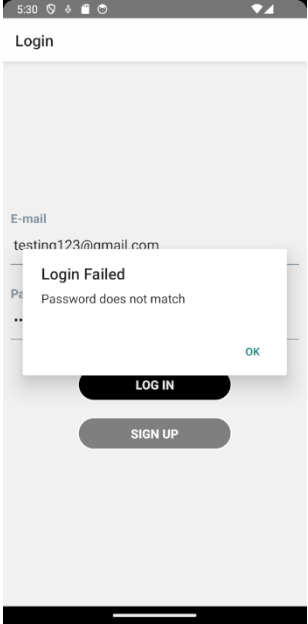
Success Case:

No.	Mobile Screen	Process
1	 <p data-bbox="448 1973 807 2007">Figure 6.3.10 Login Screen</p>	<ol style="list-style-type: none"> <li data-bbox="906 1323 1294 1464">1) User fill in the correct login credentials at the login page. <li data-bbox="906 1487 1294 1570">2) User click on the “LOG IN” button.

2	 <p data-bbox="448 943 810 974">Figure 6.3.11 Channel Page</p>	1) User will be redirect to Channel Page.
---	---	---

Failure Case:

No.	Mobile Screen	Process
1	 <p data-bbox="376 1760 863 1845">Figure 6.3.12 Login Page with wrong credentials</p>	1) Fill in the incorrect login credentials 2) Click the “LOG IN” button

2	 <p>Figure 6.3.13 Email not found</p>  <p>Figure 6.3.14 Password does not match</p>	<ol style="list-style-type: none"> 1) The “Login Failed” alert box pops out showing the error message such as “email not found”, and “password does not match”. 2) User click “OK” button to retry login again.
---	--	---

6.3.2.2 Code explanation

Explanation starts from the view from the front-end mobile application, and then how it triggers the Laravel back-end function to perform login. So the login page is located at `Aquaculture\src\pages\account\login\index.js`.

```
<View style={{ flex: 1, justifyContent: 'center' }}>
  <Input
```

```

    label="E-mail"
    placeholder='E-mail'
    onChangeText={this.EmailChangeText}
    value={this.state.email}
  />
  <Input
    label="Password"
    placeholder='Password'
    onChangeText={this.PasswordChangeText}
    secureTextEntry={true}
    textContentType="password"
    value={this.state.password}
  />
  <View style={{
    alignItems: 'center'
  }}>
    <Button
      title="LOG IN"
      buttonStyle={{
        backgroundColor: 'black',
        borderWidth: 2,
        borderColor: 'white',
        borderRadius: 30,
      }}
      containerStyle={{
        width: 200,
        marginHorizontal: 50,
        marginVertical: 10,
      }}
      titleStyle={{ fontWeight: 'bold' }}
      onPress={this.LoginButtonClick}
    />
    <Button
      title="SIGN UP"
      buttonStyle={{
        backgroundColor: 'grey',
        borderWidth: 2,
        borderColor: 'white',
        borderRadius: 30,
      }}
      containerStyle={{
        width: 200,
        marginHorizontal: 50,
        marginVertical: 10,
      }}
      titleStyle={{ fontWeight: 'bold' }}
      onPress={this.RegisterButtonPress}
    />
    { /* <TouchableOpacity
      onPress={this.keepLogin()}
    /> */ }
  </View>
</View>

```

Figure 6.3.15 Login Form

From this form, after user fill in the login credentials, user click the “LOG IN” button to trigger the “LoginButtonClick”.

```

LoginButtonClick = async () => {
  const userData = {
    email: this.state.email,
    password: this.state.password,
  };
  try {

```

```

    const response = await axios.post('http://10.0.2.2:8000/api/login',
    userData);
    console.log(response.data);

    const token = response.data.token;
    const user = response.data.user;

    // Set the message from the server to state
    this.setState({ user }, () => {
        console.log("LoginButtonClick",this.state.email);
        console.log("LoginButtonClick",token);
        console.log("LoginButtonClick",user.id);
        // Set user info in the MobX store
        this.props.rootStore.setUserInfo(user.email, token, user.id);
        console.log("login.rootStore", this.props.rootStore)
        // Update the isLogin state to trigger navigation
        this.setState({ isLogin: true }, () => {
            // Navigate to Dashboard once isLogin is set to true
            this.props.navigation.push('DrawerNavigator', { screen: 'ChannelStack',
params: { screen: 'Channel' } });
        });
    });
} catch (error) {
    console.log(error.response); // Log the entire error response for debugging

    let errorMessage = 'An error occurred during login';

    if (error.response) {
        const responseData = error.response.data;
        if (responseData && responseData.message) {
            errorMessage = responseData.message; // Use the server-provided error
message
        }
    }

    Alert.alert(
        'Login Failed',
        errorMessage,
        [
            { text: 'OK', onPress: () => console.log('OK Pressed') },
        ],
        { cancelable: true }
    );
}
};

```

Figure 6.3.16 LoginButtonClick

When LoginButtonClick is trigger, this code uses the login credentials to make a POST request to a login API endpoint using ('http://10.0.2.2:8000/api/login') using Axios Library. Then the Laravel back-end server receive the POST request. Below shows the route for this API endpoint.

```

Route::middleware('api')->group(function () {
    ...
    Route::post('api/login',
[AuthenticatedSessionController::class, 'store']);
    ...
});

```

Figure 6.3.17 route::api/login – POST

When this route is called, it will trigger the “store” function in `AuthenticatedSessionController`. Figure below shows the store function.

```
public function store(Request $request): JsonResponse
{
    // Attempt to authenticate the user
    if (Auth::attempt($request->only('email', 'password'))) {
        // Get the authenticated user
        $user = Auth::user();

        // Generate a token for the user
        $token = $user->createToken('authToken')->plainTextToken;

        // Return the token as a JSON response
        return response()->json(['token' => $token, 'user' => $user]);
    }

    // Check if the email exists in the database
    $user = User::where('email', $request->input('email'))->first();

    if (!$user) {
        // Email not found
        return response()->json(['message' => 'Email not found'], 401);
    }

    // Password does not match
    return response()->json(['message' => 'Password does not match'], 401);
}
```

Figure 6.3.18 `AuthenticatedSessionController` store function

In the store function, first it try to authenticate the user using the Laravel “`Auth::attempt`” method. If the email and the password match the credentials of a user in the database, then it will generate an authentication token and returns it along with the user information in a JSON response. If the authentication fails, then it will starts to check whether the email provided exists in the database, if it is not, then return a JSON response with a message saying “Email not found”. Lastly, if the email is found in the database, then it means that the password is incorrect, and then it will return a message of “Password does not match” in a JSON response.

Now, the JSON response is sent back to the `LoginButtonClick` function. If the API responds with a success status, it extracts the authentication token and user information from the response data. Then it sets the user information in the component’s state using “`this.setState`”.

Next, it use `MobX` to set the user’s information (email, token and `userID`) in a store for later use throughout the application. Then it will navigate

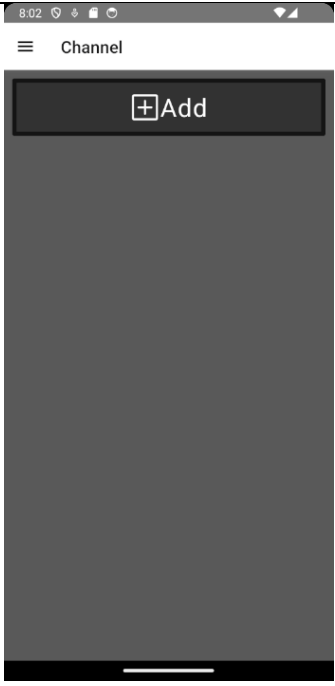
user to a specific screen called “ChannelStack” which uses “Channel” page as the initial page. In case of there is an error caughts from the login request, it will show an alert box telling the user “Login Failed” attached with the error message. User click the “OK” button to retry login.

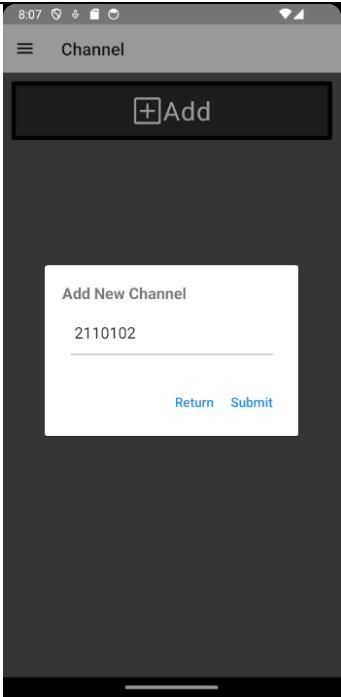

6.3.3 Channels

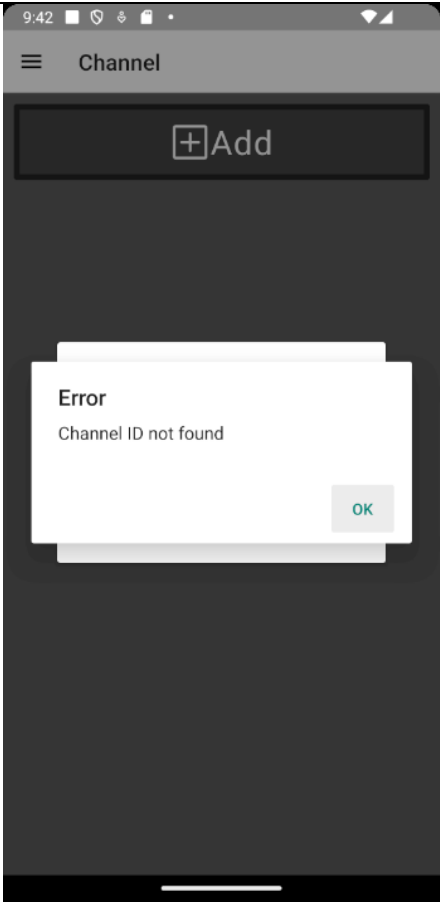
6.3.3.1 Feature demonstration

Below shows the process of using Channel function of Water Quality Montoring mobile application.

Add Channel Success Case:


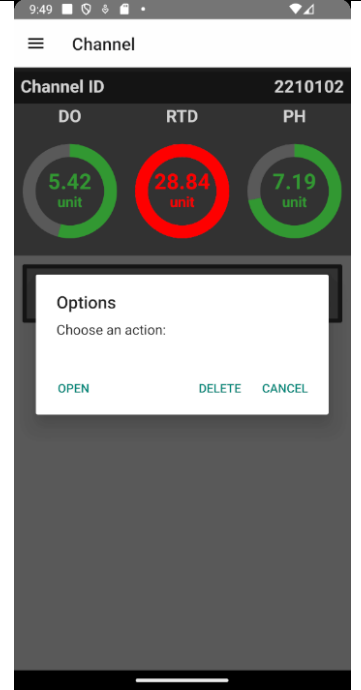
No.	Mobile Screen	Process
1	 <p data-bbox="448 1518 815 1552">Figure 6.3.19 Channel Page</p>	<ol style="list-style-type: none"> <li data-bbox="911 824 1294 920">1. User go to “Channel” page <li data-bbox="911 936 1294 1133">2. User wants to add a channel (Water Tank) to monitor, user click on the “Add” button.

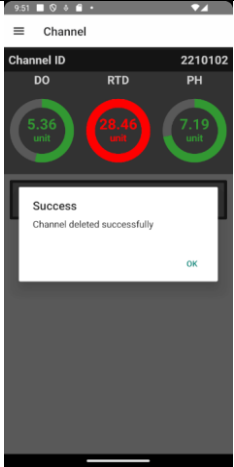

<p>2</p>	 <p>Figure 6.3.20 Add New Channel</p>	<ol style="list-style-type: none"> 1. User see a Add New Channel Dialog box pops out. 2. User key in the Channel ID. Channel ID can get from the ThingSpeak Channel page. 3. User click “Submit” button to add a channel.
<p>3</p>	 <p>Figure 6.3.21 Add Channel Success</p>	<ol style="list-style-type: none"> 1. User can see a new channel is added now.
<p style="text-align: center;">Error Handling</p>		

3	 <p>Figure 6.3.22 Channel ID not found</p>	1) If user key in invalid Channel ID, an alert message pops out telling user the Channel ID is not found.
---	--	---

Delete Channel Case:

No.	Mobile Screen	Process
-----	---------------	---------

1	 <p>Figure 6.3.23 Channel Page with a channel</p>	<ol style="list-style-type: none">1. User successfully added a new channel.2. Now user want to delete this channel.3. User long press the channel.
2	 <p>Figure 6.3.24 Channel Options</p>	<ol style="list-style-type: none">1. User can click “Delete” button to delete this channel.

3	 <p>Figure 6.3.25 Delete Channel</p>  <p>Figure 6.3.26 Channel Deleted</p>	<ol style="list-style-type: none"> 1. After the channel is deleted from database, a success message pops out telling user that channel is deleted successfully. 2. User press “OK” button and the Channel page will be refreshed and now the channel is deleted from the Channale page.
---	---	---

6.3.3.2 Code explanation

1) Add Channel

From the Channel Page, when user submit the “Add New Channel” form, then the submitForm function will be triggered. Below shows the code snippet of the submitForm function.

```
submitForm = async () => {
  // Perform Axios POST request here
  try {
    const response = await axios.post('http://10.0.2.2:8000/api/channels', {
      user_id: this.props.rootStore.userId, // Replace with your user ID
      channel_id: this.state.channelID,
    }, {
      headers: {
        'Authorization': 'Bearer ' + this.props.rootStore.token, // Replace with
        your actual token
      },
    });

    if (response.status === 201) {
      console.log('POST request success:', response.data);
      // You can also close the dialog or perform other actions after success
    }
  }
}
```

```

    this.toggleAddDialog();
    this.props.onPostSuccess(); // Call the callback from props
  } else {
    // Show an alert for non-successful response
    const errorMessage = response.data.error || 'Failed to create a channel.
Please try again later.';
    Alert.alert(
      'Error',
      errorMessage,
      [
        { text: 'OK', onPress: () => console.log('OK Pressed') },
      ],
      { cancelable: true }
    );
  }
} catch (error) {
  // Show an alert for network errors or other issues
  let errorMessage = error.response.data.error;
  Alert.alert(
    'Error',
    errorMessage,
    [
      { text: 'OK', onPress: () => console.log('OK Pressed') },
    ],
    { cancelable: true }
  );
}
};

```

Figure 6.3.27 submitForm

This submitForm function will request make a POST request to the URL ('http://10.0.2.2:8000/api/channels') with user_id and channel_id as request body data. Then this POST request will later call this route from back-end server.

```

Route::middleware('auth:sanctum')->group(function () {
  Route::post('/channels', [ChannelsController::class, 'store']);
});

```

Figure 6.3.28 route::api/channels – POST

Note that the route above is protected by the Sanctum authentication middleware, which means that only authenticated users with valid Sanctum token will be able to access this routes. This route will call the “store” function in the ChannelsController. Below is the code snippet for the “store” function.

```

public function store(Request $request)
{
  $user = Auth::user();
  // Validate the incoming request
  $request->validate([
    'channel_id' => 'required|numeric',
    // Add other validation rules for your fields
  ]);
  $channelId = $request->input('channel_id');
  // Call ThingSpeak API to get data for the provided channel_id
}

```

```

    $apiUrl =
    "https://api.thingspeak.com/channels/{channelId}/feeds.json?results=1&timezone=Asia%2FKuala_Lumpur";
    try {
        // Attempt to fetch data from the API using file_get_contents
        $apiResponse = file_get_contents($apiUrl);

        if ($apiResponse === false) {
            // Handle file_get_contents error
            return response()->json([
                'error' => 'Unable to fetch data from ThingSpeak API',
            ], 400);
        }
        // Parse the JSON response
        $apiData = json_decode($apiResponse, true);
        // Check if the channel name matches "Aquaculture"
        if (isset($apiData['channel']) && $apiData['channel']['name'] ===
'Aquaculture') {
            // Create a new channel record in the database
            $channel = $user->channels()->create([
                'channel_id' => $apiData['channel']['id'],
                // Assign other fields
            ]);
            // Additional logic to create waterparams entries using the API
result
            foreach ($apiData['channel'] as $key => $value) {
                if (strpos($key, 'field') === 0 && $value) {
                    $fieldNumber = substr($key, 5); // Extract field number
                    $chartTitle = ucfirst($value); // Convert to title case

                    $channel->waterparams()->create([
                        'water_parameter' => $chartTitle,
                        'chart_id' => $key,
                        'chart_title' => $chartTitle,
                        'field_id' => $key,
                        'min_level' => 0,
                        'max_level' => 10,
                        'min_safe' => 4,
                        'max_safe' => 8,
                        'normal_color' => '#339933',
                        'warning_color' => 'red',
                        'unit' => 'unit',
                        'line_graph_webview_link' => '',
                        'gauge_webview_link' => '',
                    ]);
                }
            }
            $user->load('channels.waterparams');
            return response()->json([
                'message' => 'Channel created successfully',
                'user' => $user,
            ], 201);
        } else {
            // Channel name doesn't match
            return response()->json([
                'error' => 'Channel ID not found',
            ], 400);
        }
    } catch (\Exception $e) {
        // Handle exceptions here (e.g., network errors)
        return response()->json([
            'error' => 'Channel ID not found',
        ], 400);
    }
}

```

Figure 6.3.29 ChannelsController store function

This store function first get the authenticated user from the Laravel authentication system. Next, it validates the channel_id by checking whether it

has value and is numeric. Then it will starts to fetch data from the ThingSpeak API using “file_get_contents” function with a url (‘https://api.thingspeak.com/channels/{\$channelId}/feeds.json?results=1&time zone=Asia%2FKuala_Lumpur’). If there is an issue with the fetching data such as network error or bad request, then it will trigger the catch block and return the “Channel ID not found” as error message. If the data successfully fetched from the API, it proceeds to parse the JSON response and store them into the ‘\$apiData’ variable. Then it checks if the retrieved channel name within “\$apiData” matches the expected name “Aquaculture”. If does not match, it returns a JSON response indicating that the channel ID was not found. If the channel name matches, it continues to create a new channel record in the application’s database using the authenticated user’s relationship with channels. Then it iterates through the channel fields in the API response and creates a ‘waterparams’ entries in the database based on the field information. After the database record is successfully created, it then returns a JSON response indicating that the channel was creating successfully, along with the user’s updated information.

Then the JSON response will then finally returns back to the submitForm function. If the status code response is ‘201’, it means that the creation of channel is successful. If the status code is not ‘201’, it means there is an issue with request, and the error will be displayed on the alert box. In the catch block also use to handle any network errors that may occurs during the request. So if any issues happens and being catch in the catch block, an alert box will be shown along with the error message. The alert box have an “OK” button to retry the add channel function.

2) Delete Channel

In the Channel page, when user long press a channel, and click on the delete options. This action will trigger the deleteChannelById function, the details of the function is provided in the figure below.

```
deleteChannelById = async (channelId) => {
  console.log("deleteChannelById", channelId)
```



```

try {
  const response = await axios.delete(
    `http://10.0.2.2:8000/api/channels/${channelId}`,
    {
      headers: {
        Authorization: 'Bearer ' + this.props.rootStore.token,
      },
    },
  );

  if (response.status === 200) {
    // Channel deletion was successful
    Alert.alert(
      'Success',
      'Channel deleted successfully',
      [
        {
          text: 'OK',
          onPress: () => this.props.navigation.replace('Channel'),
        },
      ],
    );
  } else {
    // Handle non-successful response (e.g., display an error message)
    Alert.alert(
      'Error',
      response.data.error || 'Error deleting channel',
      [
        {
          text: 'OK',
        },
      ],
    );
  }
} catch (error) {
  console.error(error);
  // Handle network errors or other issues
  Alert.alert(
    'Error',
    'An error occurred while deleting the channel. Please try again later.',
    [
      {
        text: 'OK',
      },
    ],
  );
}
};

```

Figure 6.3.30 deleteChannelById

This function first makes an asynchronous Axios DELETE request to an API endpoint using this url ('http://10.0.2.2:8000/api/channels/\${channelId}'). The request includes an Authorization header with a bearer token, which is for user authentication and authorization. If the response status code is 200, means that the channel deletion is successful. Then it will displays an alert box indicating that the channel is deleted successfully. An "OK" button is provided, when it is pressed it will refresh the Channel page to re-render the page. Moreover, if the response status code is not 200, it means that there is issue with the API request. Then an alert

box will be shown indicating the error and provided with the error details. Besides, there is also a catch block also helps to catch and shown error in alert box if there is error exception thrown in the try block.

Then lets move to the route for this API endpoint call ('http://10.0.2.2:8000/api/channels/{channelId}'). It will calls this route as shown in the figure below.

```
Route::middleware('auth:sanctum')->group(function () {
    Route::delete('/channels/{channelId}', [ChannelsController::class,
    'deleteChannelById']);
});
```

Figure 6.3.31 route::api/channels/{channelId} – DELETE

This route will then triggers the deleteChannelById function in the ChannelsController class, the function details is provided below.

```
public function deleteChannelById($channelId)
{
    // Retrieve the authenticated user
    $user = Auth::user();
    // Find the channel by its ID
    $channel = $user->channels->find($channelId);
    if (!$channel) {
        // Channel not found, return an error response
        return response()->json([
            'error' => 'Channel not found',
        ], 404); // You can use a different HTTP status code if needed
    }
    // Delete the channel and its related waterparams
    $channel->delete();
    // Return a success message
    return response()->json([
        'message' => 'Channel deleted successfully',
    ]);
}
```

Figure 6.3.32 ChannelsController deleteChannelById function

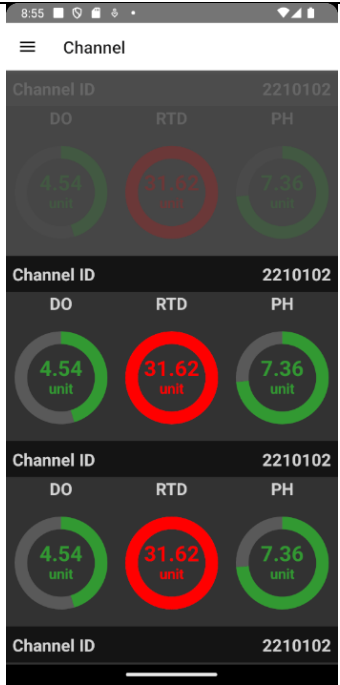
It first retrieves the authenticated user and loads the user's information into the '\$user' variable. Then it attempts to find the channel to be deleted using the '\$channelId'. If that channel is not found, then it returns a JSON response with a 404 status code and an error message indicating that the channel is not found, and thus channel deletion fails. If the channel is found, it proceeds to delete it. After successful deletion, it returns a JSON response with a status code of 200 and a success message indicates the channel is deleted successfully.

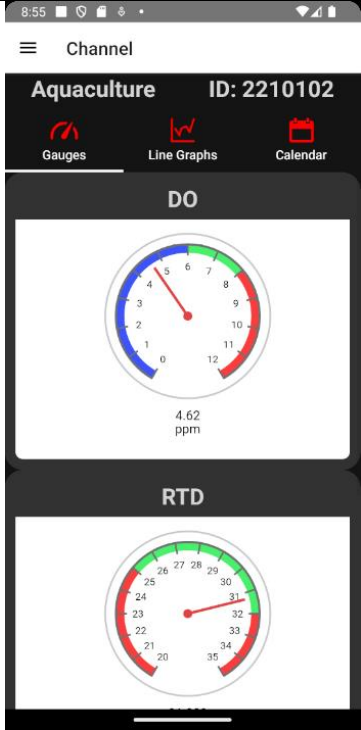
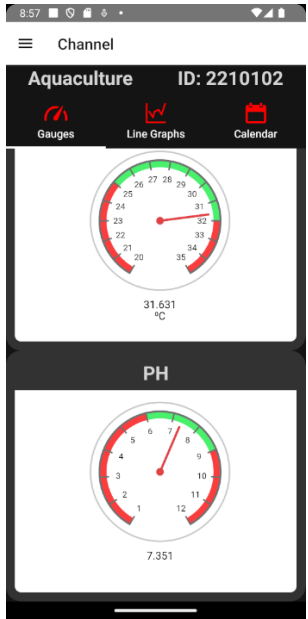
6.3.4 Dashboard

6.3.4.1 Feature demonstration

Below shows the process of using Register function of Water Quality Monitoring mobile application.

Display Dashboard Case:

No.	Mobile Screen	Process
1	 <p>Figure 6.3.33 User click on a channel</p>	<ol style="list-style-type: none"> 1. User click on a channel, and it will redirect to the dashboard page of user clicked channel.

2	 <p>The screenshot shows a mobile application interface for an aquaculture channel. At the top, it displays 'Channel' with a menu icon, 'Aquaculture ID: 2210102', and three navigation tabs: 'Gauges', 'Line Graphs', and 'Calendar'. The 'Gauges' tab is active. Below the tabs, there are two gauge displays. The first is labeled 'DO' (Dissolved Oxygen) and shows a needle pointing to 4.62 ppm on a scale from 0 to 12. The second is labeled 'RTD' (Real Time Display) and shows a needle pointing to 31.631 °C on a scale from 20 to 35.</p> <p>Figure 6.3.34 Dashboard gauges page</p>  <p>This screenshot shows the same application interface as Figure 6.3.34, but after scrolling down. The 'Line Graphs' tab is now active. The 'RTD' gauge is at the top, showing 31.631 °C. Below it is the 'PH' gauge, showing a needle pointing to 7.351 on a scale from 1 to 12.</p> <p>Figure 6.3.35 Dashboard gauges page scroll down</p>	<ol style="list-style-type: none"> 1. User can see the dashboard gauges page. 2. User can scroll up and down to view other gauge. 3. User click the “Line Graphs” tab or slide the screen from right to left to show the line graph dashboard.
---	---	---

3

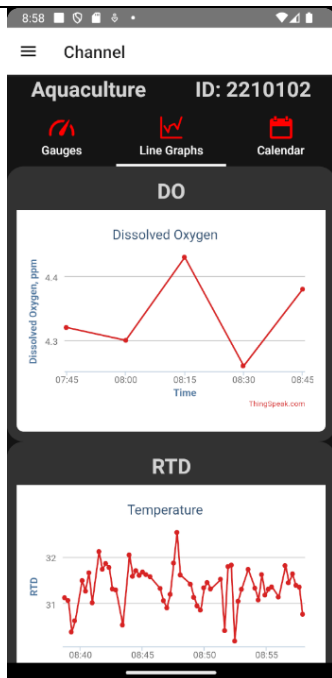


Figure 6.3.36 Line Graphs Dashboard

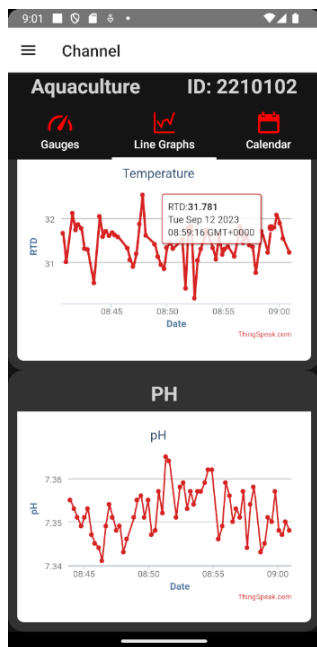


Figure 6.3.37 Line Graphs Dashboard
scroll down

1. User can view the line graphs dashboard and it is also scrollable.
2. Then user can click on the Calendar tab to switch the the calendar dashboard.

4

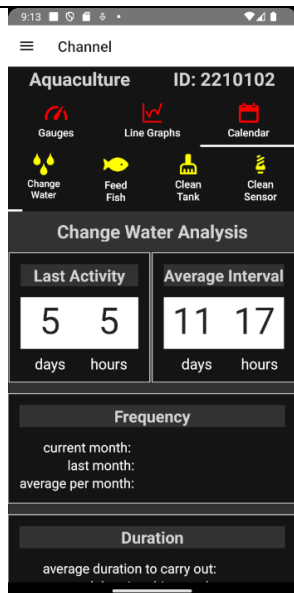


Figure 6.3.38 Calendar page analysis section

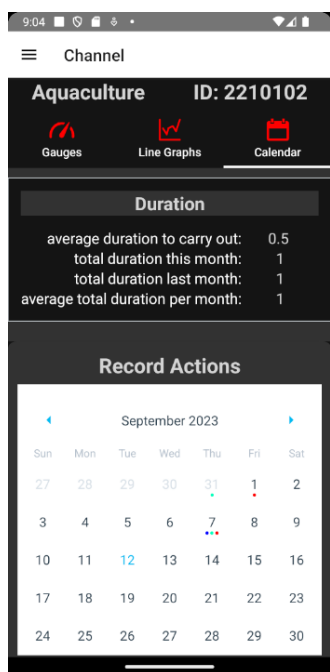
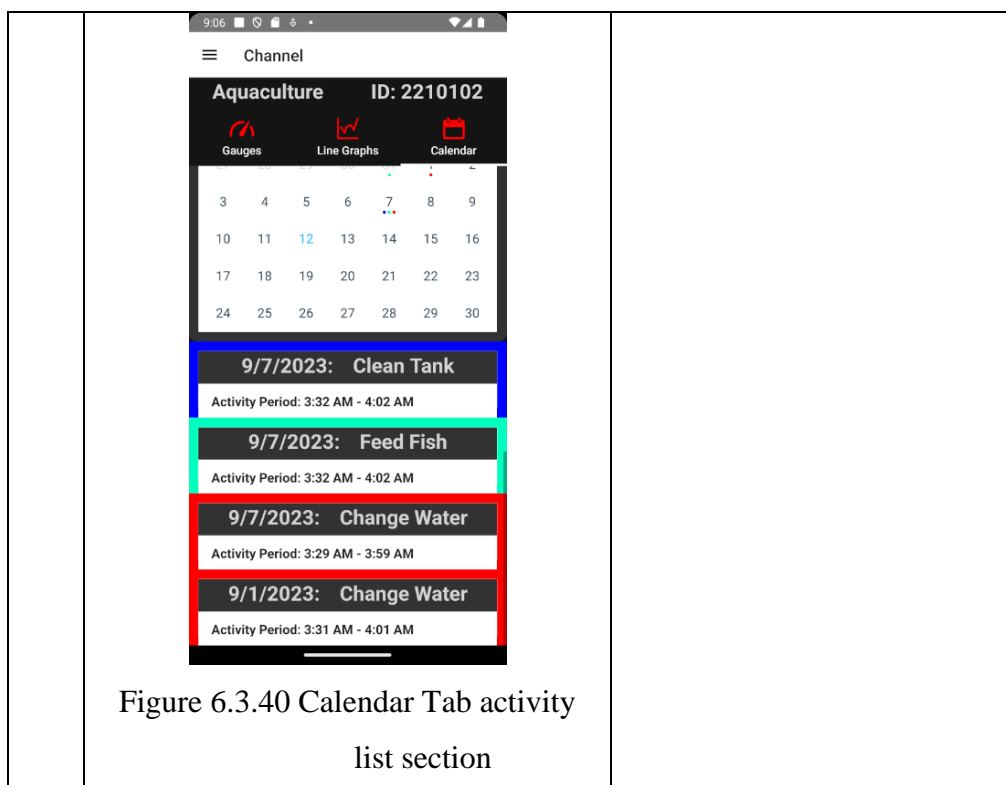


Figure 6.3.39 Calendar Tab record actions section

1. User can see the Change Water Analysis Page.
2. User can scroll down to visit “Record Activity” section, and scroll further to reach “Activity List” section.
3. User can also click on the “Activity Analysis” tab to switch between “Change Water”



6.3.4.2 Code explanation

In the Channel page, when user click on a channel it will trigger this function called `toggleChannelDetails` with `index` and `channelId` as parameters. The ‘`index`’ stands for the index for this channel in the channel list. The ‘`channelId`’ is the ID of this channel is the database record.

```
toggleChannelDetails = (index, channelId) => {
  console.log("toggleChannelDetails", this.state, index, channelId);
  this.props.navigation.push("Dashboard", { state: this.state, index,
  channelId });
};
```

Figure 6.3.41 Channel Page `toggleChannelDetails` function

So when `toggleChannelDetails` is triggered, it will navigate to a screen named “Dashboard” and also passes data to that screen via the route’s parameters. The data includes the current state of the component, index of current channel and current channel’s ID. After navigated to the dashboard page, figure below shows the `componentDidMount` function triggered when user navigated to dashboard page.

```
componentDidMount() {
```

```

console.log("haha", this.props?.route?.params?.state);
// doneSetState
const receivedState = this.props.route.params?.state;
const channelDataIndex = this.props.route.params?.index;
const channelId = this.props.route.params?.channelId;

this.setState({ receivedState, channelDataIndex, channelId }, () => {
  console.log("receivedState", this.state.receivedState);
  console.log("receivedState.channelDataIndex", this.state.channelDataIndex);
  console.log("receivedState.channelId", this.state.channelId);
  this.setState({doneSetState: !this.state.doneSetState});
  this.readActions(); // Call the function after setting state
});

this.fetchData();

// Set up an interval to fetch data every hour (3600000 milliseconds)
this.dataInterval = setInterval(this.fetchData, 3600000);
}

```

Figure 6.3.42 Dashboard Page ComponentDidMount function

So this function will first sets the component's state using the parameter from the route's parameters. Then it starts to run the fetchData function and will keep calling this function every hour. The fetchData function will get all the necessary data for the activity analysis dashboard.

1) Activity Analysis Dashboard

```

fetchData = () => {
  const channelId = this.props.route.params?.state.channelData[0].channel.id;

  // Fetch data for different actions
  this.fetchActionData(channelId, 'Feed Fish', 'lastFeedFish');
  this.fetchActionData(channelId, 'Change Water', 'lastChangeWater');
  this.fetchActionData(channelId, 'Clean Tank', 'lastCleanTank');
  this.fetchActionData(channelId, 'Clean Sensor', 'lastCleanSensor');

  this.fetchActionAverageData(channelId, 'Feed Fish', 'averageFeedFish');
  this.fetchActionAverageData(channelId, 'Change Water', 'averageChangeWater');
  this.fetchActionAverageData(channelId, 'Clean Tank', 'averageCleanTank');
  this.fetchActionAverageData(channelId, 'Clean Sensor', 'averageCleanSensor');

  this.fetchMonthFrequencyData(channelId, 'Feed Fish', 'frequencyFeedFish');
  this.fetchMonthFrequencyData(channelId, 'Change Water',
'frequencyChangeWater');
  this.fetchMonthFrequencyData(channelId, 'Clean Tank', 'frequencyCleanTank');
  this.fetchMonthFrequencyData(channelId, 'Clean Sensor',
'frequencyCleanSensor');

  this.fetchDurationData(channelId, 'Feed Fish', 'durationFeedFish');
  this.fetchDurationData(channelId, 'Change Water', 'durationChangeWater');
  this.fetchDurationData(channelId, 'Clean Tank', 'durationCleanTank');
  this.fetchDurationData(channelId, 'Clean Sensor', 'durationCleanSensor');
};

```

Figure 6.3.43 Dashboard Page fetchData function

The fetchData function first get the ThingSpeak Channel ID from the route parameter. Then, use this channelId to make multiple calls to 'fetchActionData',

'fetchActionAverageData', 'fetchMonthFrequencyData', 'fetchDurationData'.

Each of these functions will be called 4 times, each with 4 sets of parameter:

- 1) 'Feed Fish', 'lastFeedFish'
- 2) 'Change Water', 'lastChangeWater'
- 3) 'Clean Tank', 'lastCleanTank'
- 4) 'Clean Sensor', 'lastCleanSensor'

```

fetchActionData = (channelId, title, stateKey) => {
  axios
    .post('http://10.0.2.2:8000/api/actions/lastinterval/' + channelId, {
      title: title,
    }, {
      headers: {
        'Authorization': 'Bearer ' + this.props.rootStore.token,
        'Content-Type': 'application/json',
      },
    })
    .then(response => {
      this.setState({ [stateKey]: response.data }, () => {
        console.log('fetchActionData', response.data);
      });
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
};

fetchDurationData = (channelId, title, stateKey) => {
  axios
    .post('http://10.0.2.2:8000/api/actions/duration/' + channelId, {
      title: title,
    }, {
      headers: {
        'Authorization': 'Bearer ' + this.props.rootStore.token,
        'Content-Type': 'application/json',
      },
    })
    .then(response => {
      this.setState({ [stateKey]: response.data }, () => {
        console.log('fetchDurationData', response.data);
      });
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
};

fetchMonthFrequencyData = (channelId, title, stateKey) => {
  axios
    .post('http://10.0.2.2:8000/api/actions/frequency/' + channelId, {
      title: title,
    }, {
      headers: {
        'Authorization': 'Bearer ' + this.props.rootStore.token,
        'Content-Type': 'application/json',
      },
    })
    .then(response => {
      this.setState({ [stateKey]: response.data }, () => {
        console.log('fetchMonthFrequencyData', response.data);
      });
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
};

fetchActionAverageData = (channelId, title, stateKey) => {

```

```

axios
  .post('http://10.0.2.2:8000/api/actions/averageinterval/' + channelId, {
    title: title,
  }, {
    headers: {
      'Authorization': 'Bearer ' + this.props.rootStore.token,
      'Content-Type': 'application/json',
    },
  })
  .then(response => {
    this.setState({ [stateKey]: response.data }, () => {
      console.log('fetchActionAverageData', response.data);
    });
  })
  .catch(error => {
    console.error('Error fetching data:', error);
  });
};

```

Figure 6.3.44 fetchActionData

Inside this function, it makes an HTTP POST request to ('http://10.0.2.2:8000/api/actions/lastinterval/+ channelId') with a JSON payload in the request body, which includes a 'title' property with the value of 'title' parameter. The authentication token is also included for authentication purpose. Then it handles the response from the HTTP request, and set the component's state where 'stateKey' as the key and response data is set as the value. Error Handling in this function.

For 'fetchActionAverageData', 'fetchMonthFrequencyData', and 'fetchDurationData'. These function are similar, their difference is only the HTTP POST request url that they are calling. For 'fetchActionAverageData', the url is ('http://10.0.2.2:8000/api/actions/averageinterval/ + channelId'). Next, 'fetchMonthFrequencyData' is using url of ('http://10.0.2.2:8000/api/actions/frequency/ + channelId'). Lastly, for 'fetchDurationData', its url is ('http://10.0.2.2:8000/api/actions/duration/ + channelId'). Figure below shown the samples of the route that these function are calling.

```

Route::post('actions/lastinterval/{channel_id}', [ActionController::class,
'getLastActivityIntervals']);
Route::post('actions/duration/{channel_id}', [ActionController::class,
'calculateDurationForCurrentMonth']);
Route::post('actions/frequency/{channel_id}', [ActionController::class,
'getCurrentAndLastMonthFrequency']);
Route::post('actions/averageinterval/{channel_id}', [ActionController::class,
'getAverageIntervals']);

```

Figure 6.3.45 API routes for displaying activity analysis

The function ‘calculateDurationForCurrentMonth’ from the ActionController responsible for calculating the average duration to carry out this activity, total duration used for this activity within this month and last month, and average total duration per month. The details of the code will be shown at figure below.

```
public function calculateDurationForCurrentMonth(Request $request, $channel_id)
{
    $title = $request->input('title');
    // Retrieve authenticated user's channel IDs
    $userChannelIds = Auth::user()->channels->pluck('channel_id');

    // Check if the provided channel_id is within the user's channels
    if ($userChannelIds->contains($channel_id)) {
        $now = Carbon::now();
        $startOfMonth = $now->copy()->startOfMonth();
        $endOfMonth = $now->copy()->endOfMonth();

        // Calculate total duration of an action in the current month
        $currentMonthActions = Action::where('channel_id', $channel_id)
            ->where('title', $title)
            ->whereBetween('start_time', [$startOfMonth, $endOfMonth])
            ->get();

        $totalDurationCurrentMonth = 0;

        foreach ($currentMonthActions as $action) {
            $startTime = Carbon::parse($action->start_time);
            $endTime = Carbon::parse($action->end_time);

            $duration = $endTime->diffInSeconds($startTime);
            $totalDurationCurrentMonth += $duration;
        }

        // Calculate total duration of an action in months excluding the
current month
        $totalDurationOtherMonths = 0;
        $months = collect([]);
        $currentMonth = Carbon::now()->month;

        // Get actions for each month (except the current month)
        for ($i = 1; $i <= 12; $i++) {
            if ($i !== $currentMonth) {
                $startOfMonth = Carbon::create(null, $i, 1)->startOfMonth();
                $endOfMonth = Carbon::create(null, $i, 1)->endOfMonth();

                $actions = Action::where('channel_id', $channel_id)
                    ->where('title', $title)
                    ->whereBetween('start_time', [$startOfMonth, $endOfMonth])
                    ->get();

                $totalDuration = 0;

                foreach ($actions as $action) {
                    $startTime = Carbon::parse($action->start_time);
                    $endTime = Carbon::parse($action->end_time);

                    $duration = $endTime->diffInSeconds($startTime);
                    $totalDuration += $duration;
                }

                $totalDurationOtherMonths += $totalDuration;
            }
        }
    }
}
```

```

// Calculate average total duration of an action in months excluding
the current month
$averageTotalDurationOtherMonths = $totalDurationOtherMonths /
($months->count() ?: 1);

// Calculate average duration to carry out the action
$averageDurationToCarryOut = $totalDurationCurrentMonth /
($currentMonthActions->count() ?: 1);

// Calculate total duration of an action in the last month
$lastMonth = Carbon::now()->subMonth();
$lastMonthStart = $lastMonth->copy()->startOfMonth();
$lastMonthEnd = $lastMonth->copy()->endOfMonth();

$lastMonthActions = Action::where('channel_id', $channel_id)
->where('title', $title)
->whereBetween('start_time', [$lastMonthStart, $lastMonthEnd])
->get();

$totalDurationLastMonth = 0;

foreach ($lastMonthActions as $action) {
    $startTime = Carbon::parse($action->start_time);
    $endTime = Carbon::parse($action->end_time);

    $duration = $endTime->diffInSeconds($startTime);
    $totalDurationLastMonth += $duration;
}

$result = [
    'total_duration_current_month' => round($totalDurationCurrentMonth
/ 3600, 1),
    'average_total_duration_other_months' =>
round($averageTotalDurationOtherMonths / 3600, 1),
    'average_duration_to_carry_out' => round($averageDurationToCarryOut
/ 3600, 1),
    'total_duration_last_month' => round($totalDurationLastMonth /
3600, 1),
];

return response()->json($result);
} else {
    return response()->json(['error' => 'Unauthorized'], 401);
}
}

```

Figure 6.3.46 calculateDurationForCurrentMonth

For the other routes, their codes are similar with some difference in their computation for their output. For example, ‘getAverageIntervals’ will calculate the average interval of all records of single activity within a month that it records in database. The activity could be “Change Water”, “Feed Fish”, “Clean Tank” and “Clean Sensor”. The JSON response output will be in days and hour. Next, ‘getLastActivityIntervals’ helps to calculate the time of how long does the last activity is being taken until now. Moreover, ‘calculateDurationForCurrentMonth’ calculate the total duration of this activity has been taken within this month. Lastly, ‘getCurrentAndLastMonthFrequency’ helps to calculate the total frequency of this activity taken in this month, last month and also the average frequency between every month.

As a prove that these function can helps to provides these data for activity analysis, the figure below will shows the activity analysis dashboard page which display the returned data of the ‘getLastActivityIntervals’, ‘calculateDurationForCurrentMonth’, ‘getCurrentAndLastMonthFrequency’, ‘getAverageIntervals’.

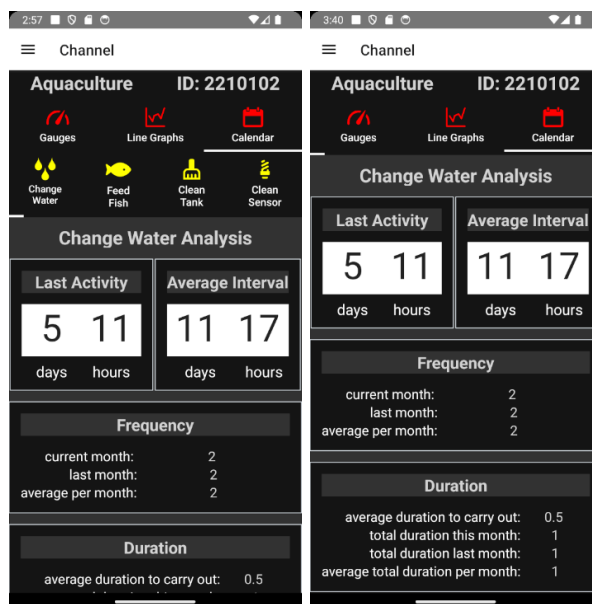


Figure 6.3.47 Change Water Analysis

From the figure above, the output from ‘getLastActivityIntervals’ is shown in the “Last Activity” cardbox, which indicates that the last change water activity is taken 5 days and 11 hours ago. Next, the output from ‘calculateDurationForCurrentMonth’ is shown in the Average Interval “cardbox”, which indicates that the average interval of all records of this activity is 11 days and 17 hours. In addition, the output from ‘getCurrentAndLastMonthFrequency’ will be shown in “Frequency” cardbox, with the value of current month is 2, last month is 2, average month is 2. Lastly, the output from ‘calculateDurationForCurrentMonth’ is the “Duration” cardbox, with the value of average duration to carry out is 0.5 hrs, total duration this month is 1 hrs, total duration last month is 1 hrs, and average total duration per month is 1 hrs.

2) Gauges Dashboard and Line Graphs Dashboard

Code snippet below is the WebView component which use the url of gauges provided by ThingSpeak to display the gauges.

```

<WebView
    scalesPageToFit={false}
    bounces={false}
    key={gaugeSettings.field_id} // Use a unique key
    showsHorizontalScrollIndicator={false}
    showsVerticalScrollIndicator={false}
    overScrollMode="never"
    scrollEnabled={false}
    style={{
        margin: 0,
        padding: 0,
        top: -23,
        left: -20,
        opacity: 0.99,
        height: 250,
        flex: 1,
    }} // Adjust the height as needed
    nestedScrollEnabled={false}
    source={{
        uri: gaugeSettings.gauge_webview_link, // Use
gaugeSetting here
    }}
/>

```

Figure 6.3.48 Dashboard Page Gauges Webview

From the figure above, the ThingSpeak gauge link is stored in the 'gaugeSettings.gauge_webview_link'. The url of gauges can get from the function below which has been triggered in the Channel page, and is being pass to the Dashboard page through the route parameters, and then being set to the 'gaugeSettings'. That function is 'fetchUserGaugeSettings' placed in `Aquaculture\src\pages\account\channel\index.js`.

```

fetchUserGaugeSettings = async callback => {
  try {
    const response = await axios.post(
      'http://10.0.2.2:8000/api/waterparams',
      {},
      {
        headers: {
          Authorization: 'Bearer ' + this.props.rootStore.token,
        },
      },
    );
    console.log("gaugeSettings.only", response.data);
    this.setState({ gaugeSettings: response.data }, callback);
  } catch (error) {
    console.error('Error fetching user gauge settings:', error);
  }
};

```

Figure 6.3.49 fetchUserGaugeSettings

The POST request will trigger this route:

```
Route::middleware('auth:sanctum')->group(function () {
    Route::post('/waterparams', [WaterParamsController::class, 'byUserId']);
});
```

Figure 6.3.50 route::api/waterparams POST

Then, this route will trigger the “byUserId” function in WaterParamsController class.

```
public function byUserId(Request $request){
    $channelIds = Auth::user()->channels->pluck('channel_id');
    $params = WaterParam::whereIn('channel_id', $channelIds)->get();

    return json_encode($params);
}
```

Figure 6.3.51 WaterParamsController byUserId function

This function will return the water parameters as a JSON-encoded response. The water parameters data is the gauge settings data that will need to store in the component’s state, and then pass as route parameters to the Dashboard page, then the WebView component uses the gaugeSettings in the route parameters to display the gauges.

Now, we have the ‘gaugeSettings’ data, therefore, we also can retrieve the ThingSpeak line graphs url to display the line graphs using the WebView component. The Webview components of the line graphs is shown below:

```
<WebView
    scalesPageToFit={false}
    bounces={false}
    key={gaugeSettings.field_id} // Use a unique key
    showsHorizontalScrollIndicator={false}
    showsVerticalScrollIndicator={false}
    overScrollMode="never"
    scrollEnabled={false}
    style={{
        margin: 0,
        padding: 0,
        top: -10,
        left: 0,
        opacity: 0.99,
        height: 250,
        flex: 1,
    }} // Adjust the height as needed
    nestedScrollEnabled={false}
    source={{
        uri:
            gaugeSettings.line_graph_webview_link +
            '&height=auto&width=auto', // Use gaugeSetting here
    }}
/>
```

Figure 6.3.52 Dashboard Page Line Graphs Webview

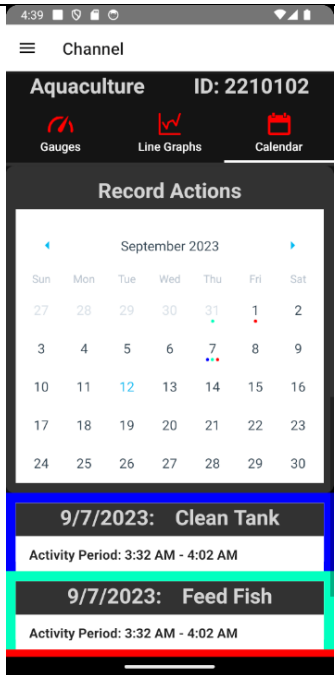
The 'gaugeSettings.line_graph_webview_link' stored the link of ThingSpeak line graph.

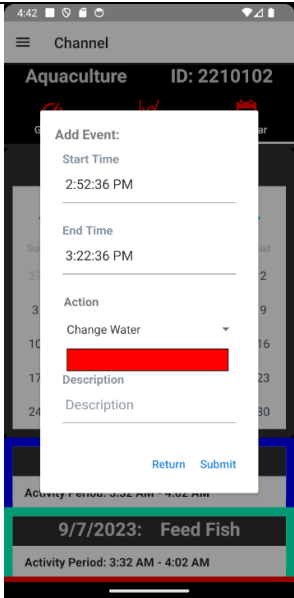
6.3.5 Activity Record

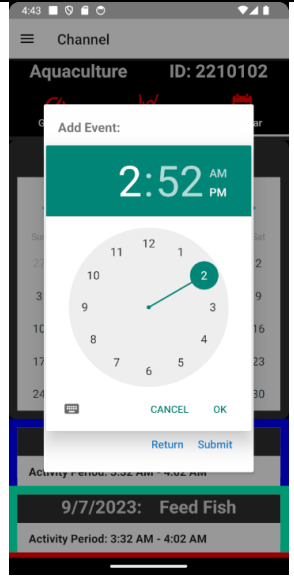
6.3.5.1 Feature demonstration

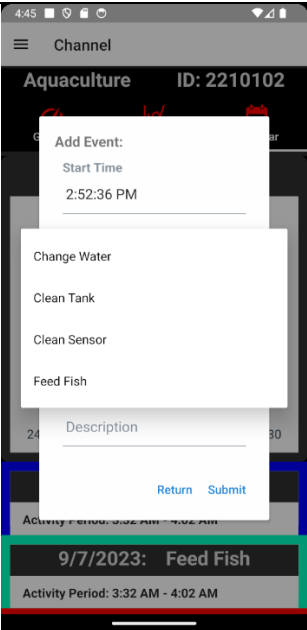
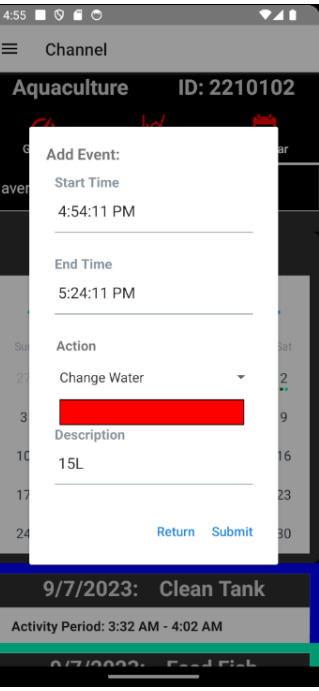
Below shows the process of using Activity Record function of Water Quality Monitoring mobile application.

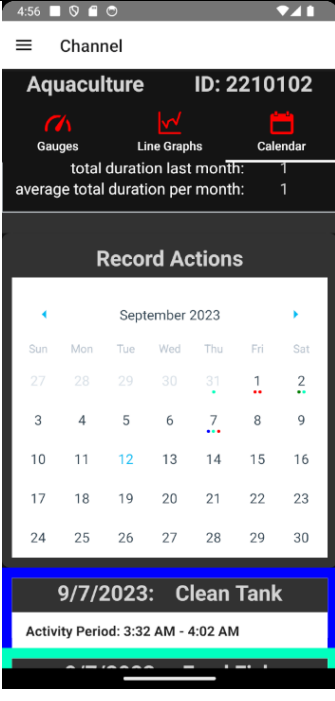

Record Activity:

No.	Mobile Screen	Process
1	 <p style="text-align: center;">Figure 6.3.53 Record Actions Calendar</p>	<ol style="list-style-type: none"> 1) User navigate to the 'Record Actions' section of the Acticity Analysis Dashboard. 2) User can see an interactive calendar. 3) User click on a date (eg. 1 September 2023) he wants to register an activity.

<p>2</p>	 <p>Figure 6.3.54 Add Event Dialog Box</p>	<p>1) Then user can see a Add Event Dialog box pops out.</p>
----------	---	--

<p>3</p>	 <p>Figure 6.3.55 Select Time</p>	<p>1) User can select start time and end time.</p>
----------	---	--

<p>4</p>	 <p>Figure 6.3.56 Select Activity</p>	<p>1) User can select the activity they want to register.</p>
<p>5</p>	 <p>Figure 6.3.57 fill in description</p>	<p>1) User fill in the description. 2) User hit the “Submit” button.</p>

6		1) Red dots added to the date (1 September 2023) indicates that the activity is recorded successfully.
7		1) User scroll down to see the activity list.

6.3.5.2 Code explanation

The record actions function can be build starts from the Calendar components in the dashboard page as shown in the figure below.

```

<Calendar
  markingType={'multi-dot'}
  onDayPress={this.handleDayPress}
  markedDates={this.state.markedDates}
  onMonthChange={this.handleMonthChange} // Add this line
/>

```

Figure 6.3.58 Dashboard Page Calendar component

When user click a date, it will triggers “handleDayPress” function. This handleDayPress finally will make the “AddEvent” component visible.

```

<AddEvent
  isAddEventVisible={this.state.isAddEventVisible}
  onToggleDialog={() =>
this.setState({ isAddEventVisible: !this.state.isAddEventVisible })}
  channel_id={this.state?.receivedState?.channelData[this.state.channelDa
taIndex]?.channel.id}
  onSubmit={this.handleEventSubmit} // Pass the function to the AddEvent
component
/>

```

Figure 6.3.59 AddEvent dialog box

Then after user finish filling up the form, and then click th “Submit” button in the AddEvent component, it will triggers “submitForm” function in the AddEvent component. The submitForm function looks like:

```

submitForm = () => {
  console.log(this.state);
  const requestData = {
    channel_id: this.props.channel_id,
    title: this.state.action.label,
    start_time: this.state.startTime,
    end_time: this.state.endTime,
    description: this.state.description,
    color: this.state.action.color,
  };

  if (this.state.description) {
    requestData.description = this.state.description;
  }

  // Pass the requestData to the onSubmit prop function
  this.props.onSubmit(requestData);
};

```

Figure 6.3.60 submitForm in AddEvent components

This code will then trigger the “onSubmit” function in the props and finally call the “handleEventSubmit” in the dashboard page, the code snippet of this function is shown at below.

```

handleEventSubmit = async (requestData) => {
  this.state.selectedDate.setHours(requestData.start_time.getHours());
  this.state.selectedDate.setMinutes(requestData.start_time.getMinutes());
  const start_time = this.state.selectedDate.toISOString();
  console.log("start_time", this.state.selectedDate);

  this.state.selectedDate.setHours(requestData.end_time.getHours());
  this.state.selectedDate.setMinutes(requestData.end_time.getMinutes());
  const end_time = this.state.selectedDate.toISOString();
  console.log("end_time", this.state.selectedDate);
  console.log("requestData", requestData);

  const requestData1 = {

```

```

channel_id: requestData.channel_id,
title: requestData.title,
start_time: start_time,
end_time: end_time,
color: requestData.color,
description: requestData.description,
});
console.log("requestData1", requestData1);

try {
  const response = await axios.post(
    'http://10.0.2.2:8000/api/actions',
    requestData1,
    {
      headers: {
        'Authorization': 'Bearer ' + this.props.rootStore.token,
      },
    },
  );

  console.log('POST request success:', response.data);
  this.toggleEventDialogVisibility();
  this.readActions(); // Call the function after setting state
  console.log()
  // this.props.navigation.navigate('DrawerNavigator', { screen:
  'ChannelStack', params: { screen: 'Dashboard' } }, { state:
  this.props.route.params?.state });

  // this.props.navigation.push("Dashboard", { state:
  this.props.route.params.state });

  // You can also perform actions after success
} catch (error) {
  console.error('Error in POST request:', error);
  Alert.alert(
    'Oops! Something Wrong, Please try again!',
    error,
    [
      { text: 'OK', onPress: () => console.log('OK Pressed') },
    ],
    { cancelable: false }
  );
}
};

```

Figure 6.3.61 Dashboard Page handleEventSubmit function

The handleEventSubmit function responsible for make a POST request to the API Endpoint using this url ('http://10.0.2.2:8000/api/actions') using 'requestData1' object as the request body, including an authorization header. If the POST request is successful, it then run the "readActions()" function to re-read the activities from the back-end server and close the event dialog. Below is the code snippet for "readActions" function.

```

readActions = async () => {
  try {
    const response = await axios.get(
      `http://10.0.2.2:8000/api/actions/channel/` +
      this.state.receivedState.channelData[0].channel_id,
      {
        headers: {
          'Authorization': 'Bearer ' + this.props.rootStore.token,

```

```

    },
  }
);

// console.log('GET request success:', response.data);
this.setState({ actions: response.data.sort((a, b) => new Date(b.start_time)
- new Date(a.start_time)) }, () => {
  this.setState({ markedDates:
this.generateMarkedDates(this.state.actions) }, () => {
  console.log("actions", this.state.actions);
  console.log("markedDates", this.state.markedDates);
  this.filterActionsByMonthAndYear();
  });
});

// You can also perform actions after success
} catch (error) {
  console.error('Error in GET request:', error);
}
}
}

```

Figure 6.3.62 readActions function

This function is responsible for making a GET request to ('http://10.0.2.2:8000/api/actions/channel/') by appending 'channel.id' into this base URL. If the GET request is successful, it then updates the component's state with the retrieved actions and marked dates. It also calls function generateMarkedDates to create an object representing marked dates on a calendar. It also calls filterActionsByMonthAndYear to filter the actions by month and year. All these function will contribute to the correct marking and display of the Calendar components.

The GET request to ('http://10.0.2.2:8000/api/actions/channel/') will trigger this route in the back-end server as shown in the figure below.

```

Route::middleware('auth:sanctum')->group(function () {
  Route::get('/actions/channel/{channel_id}', [ActionController::class,
'showByChannel']);
});

```

Figure 6.3.63 route::api/actions/channel/{channel_id} – GET

The route above will trigger the “showByChanel” function in the ActionController class. This function codes looks like:

```

public function showByChannel($channel_id)
{
  // Retrieve all actions with the specified channel_id
  $actions = Action::where('channel_id', $channel_id)->get();

  return response()->json($actions);
}

```

```
}

```

Figure 6.3.64 showByChannel function

This function is responsible for fetching and returning a list of actions associated with a specific channel based on the provided '\$channel_id'.

Then due to the component's state is updated, the marks in the calendar will be updated as well as the activity list. Below is the code to render the activity list:

```
<ListItem key={action.id} bottomDivider>
  <ListItem.Content>
    <ListItem.Subtitle style={styles.actiontext}>
      <Text>Activity Period: {startTimeString} -
{endTimeString}</Text>
    </ListItem.Subtitle>
    <ListItem.Subtitle style={styles.actiontext}>
      <Text>Description: {action.description}</Text>
    </ListItem.Subtitle>
  </ListItem.Content>
</ListItem>
```

Figure 6.3.65 Activity List

In conclusion, these are the explanation of how the activity record function works.

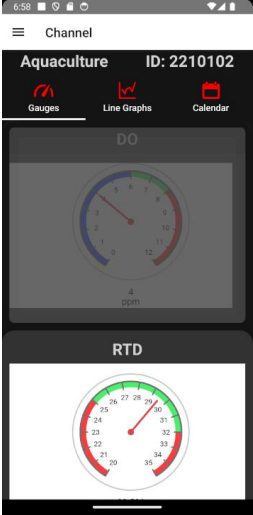
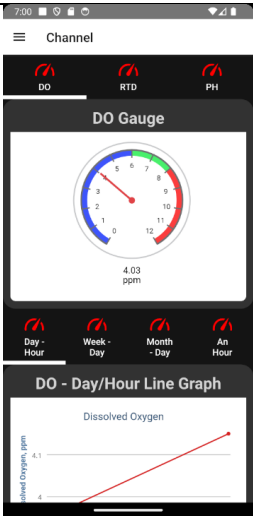
6.3.6 Analysis

6.3.6.1 Feature demonstration

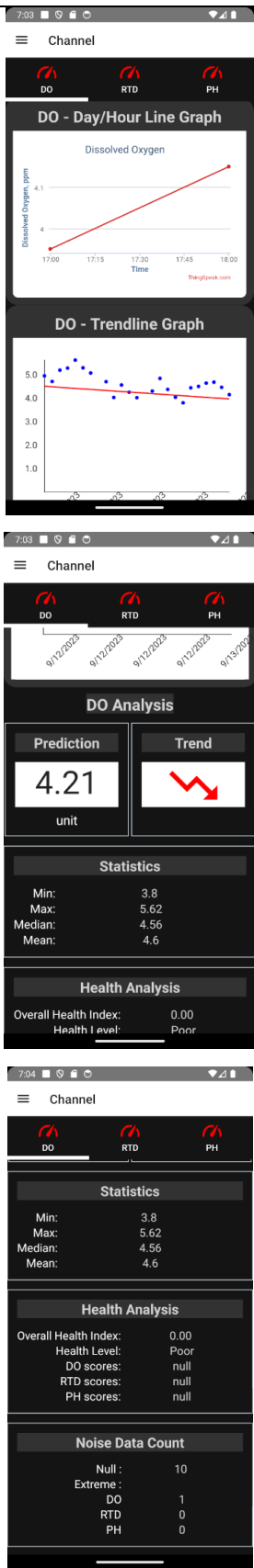
Below shows the process of using Analysis function of Water Quality Monitoring mobile application.

Show Analysis:

No.	Mobile Screen	Process

1	 <p>The screenshot shows a mobile application interface for 'Channel' (ID: 2210102). At the top, there are three tabs: 'Gauges', 'Line Graphs', and 'Calendar'. The 'Gauges' tab is active, displaying two circular gauges. The top gauge is labeled 'DO' (Dissolved Oxygen) and shows a reading of 4 ppm. The bottom gauge is labeled 'RTD' (Refractive Temperature Difference) and shows a reading of approximately 28. The background is dark, and the gauges have a light-colored face with a needle and numerical scale.</p> <p>Figure 6.3.66 Dashboard page</p>	<p>1) At the dashboard page, click the DO gauge or DO line graphs to go to Analysis page.</p>
2	 <p>The screenshot shows the 'Analysis' page for 'Channel'. At the top, there are three tabs: 'DO', 'RTD', and 'PHI'. The 'DO' tab is active, displaying a 'DO Gauge' with a reading of 4.03 ppm. Below the gauge are four tabs for time intervals: 'Day - Hour', 'Week - Day', 'Month - Day', and 'An Hour'. The 'Day - Hour' tab is selected, showing a 'DO - Day/Hour Line Graph' for 'Dissolved Oxygen'. The graph shows a red line representing the data over time, with a y-axis labeled 'Dissolved Oxygen, ppm' and a value of 4.1.</p> <p>Figure 6.3.67 Analysis Page</p>	<p>1) At the Analysis page, user can see the gauge and the line graphs.</p> <p>2) Go to line graphs section, user can click or slide to switch the tab between “Day-Hour”, “Week-Day”, “Month-Day”, and “An Hour”.</p>

3



- 1) User can scroll down to view the trendline graph to understand the trend.
- 2) User can see an water parameter analysis dashboard that shows the prediction, the trend, statistics, health analysis and noise data count.

6.3.6.2 Code explanation

When user click on the DO gauge at the dashboard page, it triggers a function named “toggleAnalysis”.

```
toggleAnalysis = (gauge, setting, channels, index) => {
  console.log('toggleAnalysis', gauge, setting, channels.user.channels[index] );
  this.props.navigation.push("Analysis", { gauge: gauge, gaugeSettings: setting,
channel: channels.user.channels[index] });
}
```

Figure 6.3.68 Dashboard page toggleAnalysis function

This toggle analysis function will navigate the screen to Analysis page, along with route parameters such as gauge, gaugeSettings and channel. In the analysis page, the gauges and the line graphs use the same Webview component introduce in the dashboard page. Therefore, this section will not shown the webview component again because the code snippet is similar, the only difference is the component’s state that store the gauge and line graphs url.

1) Water Parameter Analysis

Below shows the code for the componentDidMount of Analysis page.

```
componentDidMount() {
  const { route } = this.props;
  console.log("Analysis.componentDidMount", this.props);
  const receivedState = route.params?.gauge;
  const gaugeSettings = route.params?.gaugeSettings;
  const channelId = route.params?.channel.channel_id;
  if (receivedState) {
    this.setState({ receivedState, gaugeSettings, channelId }, () => {
      // this.fetchDataFromThingSpeak(channelId); // Fetch data immediately on
mount
    });
    let index = this.findIndex(this.state.gaugeSettings,
this.state.receivedState);
    this.fetchDataAnHour(channelId);
    this.fetchDataDayHour(channelId);
    this.fetchDataMonthDay(channelId);
    this.fetchDataWeekDay(channelId);

    this.fetchAnHourInterval = setInterval(() => {
      this.fetchDataAnHour(channelId);
    }, 15 * 60 * 1000); // 1 hour in milliseconds
    this.fetchDayHourInterval = setInterval(() => {
      this.fetchDataDayHour(channelId);
    }, 60 * 60 * 1000); // 1 hour in milliseconds
    this.fetchMonthDayInterval = setInterval(() => {
      this.fetchDataMonthDay(channelId);
    }, 24 * 60 * 60 * 1000); // 1 hour in milliseconds
    this.fetchWeekDayInterval = setInterval(() => {
      this.fetchDataWeekDay(channelId);
    }, 24 * 60 * 60 * 1000); // 1 hour in milliseconds
    this.setState({ index }, () => {
      console.log('analysis state', this.state);
    });
  });
}
```

Figure 6.3.69 Analysis Page componentDidMount function

This componentDidMount function first set the component's state with the route parameters. Then, it runs fetchDataAnHour, fetchDataDayHour, fetchDataDayMonth, fetchDataWeekDay function using channel ID as parameter.

```

fetchDataAnHour = async (channelId) => {
  try {
    const response = await axios.get(
      'https://api.thingspeak.com/channels/' +
      channelId +
      '/feeds.json?round=2&results=240&median=10&timezone=Asia%2FKuala_Lumpur'
    );
    const fetchDataAnHour = response.data.feeds.map((feed) => {
      // Create a copy of the feed object and omit the 'entry_id' property
      const { entry_id, ...feedWithoutEntryId } = feed;
      return feedWithoutEntryId;
    });

    this.setState({ fetchDataAnHour }, () => {
      console.log("fetchDataMonthDay",
        JSON.stringify(this.state.fetchDataAnHour));
      this.setState({ arimaPredictAnHour: null }, async () => {
        try {
          console.log("this.state.fetchDataAnHour");
          const data = this.state.fetchDataAnHour;
          const arimaPredictData = await this.arimaPredict(data);
          this.setState({ arimaPredictAnHour: arimaPredictData }, () => {
            const arimaPredictAnHour = this.state.arimaPredictAnHour;
            console.log("haha.arimaPredict", arimaPredictAnHour);
            console.log("this.state.fetchDataAnHour.data", data);
            const nullDataCount = this.countRowsWithNullValues(data);
            console.log("haha.noiseDataCount", nullDataCount);
            const result = this.calculateMinMaxMedianMean(data);
            console.log('haha.calculateMinMaxMedianMean', result);
            const fieldRanges = {
              field1: { min: 4, max: 12 },
              field2: { min: 20, max: 35 },
              field3: { min: 4, max: 9 }
            };
            const count = this.countExtremeValues(fieldRanges, data);
            console.log('haha.countExtremeValues', count);
            console.log('haha.fetchDataDayHour', data);

            const weights = {
              field1: 0.4,
              field2: 0.3,
              field3: 0.3,
            };

            const thresholds = {
              field1: { min: 6.0, max: 9.0 },
              field2: { min: 20.0, max: 30.0 },
              field3: { min: 6.0, max: 8.0 },
            };

            const egfpThresholds = {
              excellent: 0.8,
              good: 0.6,
              fair: 0.4,
            };

            const healthIndexResult = this.calculateHealthIndex(data, weights,
              thresholds, egfpThresholds);

```

```

        console.log("haha.healthIndexResult", healthIndexResult);

        const analysisAnHour = {
            arimaPredictData: [arimaPredictData.prediction.field1,
arimaPredictData.prediction.field2, arimaPredictData.prediction.field3],
            nullDataCount,
            minMaxMedianMean: result,
            countExtremeValues: count,
            healthIndexResult,
        };
        this.setState({ analysisAnHour }, () => {
            console.log("analysis.analysisAnHour", this.state.analysisAnHour)
        });
        console.log("haha.analysisDayHour", JSON.stringify(analysisAnHour))
    });
    } catch (error) {
        console.error('Error in arimaPredict:', error);
    }
    });
});
} catch (error) {
    console.error('Error fetching data from ThingSpeak:', error);
}
};

```

Figure 6.3.70 fetchDataAnHour

In `fetchDataAnHour`, it first makes HTTP GET request to the ThingSpeak API with this base url (`'https://api.thingspeak.com/channels/'`), and then append this url with `channelId`, stands for ThingSpeak Channel ID, and then a `'/feeds.json?round=2&results=240&median=10&timezone=Asia%2FKuala_Lumpur'` is all the URL parameters that makes it returns the data collected from the last hour. Once the data is successfully fetched, it stored the processed data in the component's state. Then it will calls the `'arimaPredict'` function to perform prediction based on the fetched data. The result of this analysis is stored in the component's state later. Next, it will performs several calculation and operation such as:

1) `countRowsWithNullValues`:

```

countRowsWithNullValues = (data) => {
    console.log("countRowsWithNullValues", data);
    let count = 0;
    for (let i = 0; i < data.length; i++) {
        const row = data[i];
        for (const key in row) {
            if (row[key] == null) {
                count++;
                break; // Move to the next row once a null value is found in the
current row
            }
        }
    }
    return count;
}

```

Figure 6.3.71 countRowsWithNullValues

This function counts rows in the data which contains null values.

2) calculateMinMaxMedianMean:

```

calculateMinMaxMedianMean = (data) => {
  // Function to find the maximum value in an array of numbers
  function findMax(numbers) {
    return Math.max(...numbers);
  }

  // Function to find the minimum value in an array of numbers
  function findMin(numbers) {
    return Math.min(...numbers);
  }

  // Function to find the median value in an array of numbers
  function findMedian(numbers) {
    const sorted = numbers.filter(value => value !== null).sort((a, b) =>
a - b);
    const length = sorted.length;
    const middle = Math.floor(length / 2);
    if (length % 2 === 0) {
      return (sorted[middle - 1] + sorted[middle]) / 2;
    } else {
      return sorted[middle];
    }
  }

  // Function to find the mean value in an array of numbers
  function findMean(numbers) {
    const filtered = numbers.filter(value => value !== null);
    const sum = filtered.reduce((acc, val) => acc + val, 0);
    return sum / filtered.length;
  }

  // Detect the field names dynamically from the first data row
  const fieldNames = Object.keys(data[0]).filter(key => key !==
"created_at");

  // Initialize an object to hold field values
  const fieldValues = {};
  fieldNames.forEach(key => {
    fieldValues[key] = [];
  });

  // Loop through the data and collect values for each field
  data.forEach(row => {
    fieldNames.forEach(key => {
      if (row[key] !== null) {
        fieldValues[key].push(parseFloat(row[key])); // Convert to number
      }
    });
  });

  // Calculate and construct the output object
  const output = {};
  fieldNames.forEach(key => {
    const max = parseFloat(findMax(fieldValues[key])).toFixed(2);
    const min = parseFloat(findMin(fieldValues[key])).toFixed(2);
    const median = parseFloat(findMedian(fieldValues[key])).toFixed(2);
    const mean = parseFloat(findMean(fieldValues[key])).toFixed(2);
    output[key] = { max, min, median, mean };
  });

  return output;
}

```

Figure 6.3.72 calculateMinMaxMedianMean

This function calculates the minimum, maximum, median, and mean values of the fetched data.

3) countExtremeValues:

```
countExtremeValues = (fieldRanges, data) => {
  const fieldCounts = {};

  for (const item of data) {
    for (const field in item) {
      if (field !== "created_at" && item[field] !== null) {
        const fieldValue = parseFloat(item[field]);
        const fieldRange = fieldRanges[field];

        if (fieldRange && (fieldValue < fieldRange.min || fieldValue >
fieldRange.max)) {
          if (!fieldCounts[field]) {
            fieldCounts[field] = 1;
          } else {
            fieldCounts[field]++;
          }
        }
      }
    }
  }

  // Add fields with count 0
  for (const field in fieldRanges) {
    if (!fieldCounts[field]) {
      fieldCounts[field] = 0;
    }
  }

  return fieldCounts;
};
```

Figure 6.3.73 countExtremeValues

This function counts rows of extreme values in the fetched data.

4) calculateHealthIndex:

```
calculateHealthIndex(data, weights, thresholds, egfpThresholds) {
  console.log('calculateHealthIndex.data', data);
  console.log('calculateHealthIndex.weights', weights);
  console.log('calculateHealthIndex.thresholds', thresholds);
  console.log('calculateHealthIndex.daegfpThresholdsta', egfpThresholds);

  function normalizeScore(values, threshold) {
    const normalizedValues = values.map(value => {
      if (value < threshold.min) {
        return 0;
      } else if (value > threshold.max) {
        return 1;
      } else {
        return (value - threshold.min) / (threshold.max - threshold.min);
      }
    });
    return normalizedValues;
  }

  function calculateComponentScore(normalizedScore, egfpThresholds) {
    if (normalizedScore >= egfpThresholds.excellent) {
      return 1;
    } else if (normalizedScore >= egfpThresholds.good) {
      return 0.75;
    } else if (normalizedScore >= egfpThresholds.fair) {
```

```

    return 0.5;
  } else {
    return 0.25;
  }
}

function determineHealthLevel(overallHealthIndex) {
  if (overallHealthIndex >= 0.75) {
    return "Excellent";
  } else if (overallHealthIndex >= 0.5) {
    return "Good";
  } else if (overallHealthIndex >= 0.25) {
    return "Fair";
  } else {
    return "Poor";
  }
}

const componentScores = {};
let overallHealthIndex = 0;

for (const field of ['field1', 'field2', 'field3']) {
  if (data.some(entry => entry[field] === null)) {
    // Skip if any entry has null value for the field
    componentScores[field] = null;
    continue;
  }

  const fieldValues = data.map(entry => entry[field]);
  const normalizedScore = normalizeScore(fieldValues,
thresholds[field]);
  const componentScore = calculateComponentScore(normalizedScore,
egfpThresholds);
  componentScores[field] = componentScore;

  overallHealthIndex += componentScore * weights[field];
}

const healthLevel = determineHealthLevel(overallHealthIndex);
console.log("calculateHealthIndex.overallHealthIndex",
overallHealthIndex);
console.log("calculateHealthIndex.healthLevel", healthLevel);
console.log("calculateHealthIndex.componentScores", componentScores);

return {
  overallHealthIndex,
  healthLevel,
  componentScores,
};
}

```

Figure 6.3.74 calculateHealthIndex

This function calculates the health index and returns overallHealthIndex, healthLevel, and componentScores.

The code snippet of 'arimaPredict' function is shown below.

```

arimaPredict = async (data) => {
  try {
    const response = await axios.post(
      'http://10.0.2.2:8000/api/predict',
      {
        data: data,
      },
      {
        headers: {
          Authorization: 'Bearer ' + this.props.rootStore.token,
        },
      },
    );
  }
}

```

```

    );
    console.log('arimaPredict response:', response.data); // Log the response
  data
  return response.data;
} catch (error) {
  console.error('Error fetching user channels:', error);
  throw error; // Rethrow the error to handle it in the calling code
}
};

```

Figure 6.3.75 arimaPredict function

This arimaPredict function first send a POST request to the URL ('http://10.0.2.2:8000/api/predict'), includes a property named 'data' in the request body along with Authorization header. This URL is an API endpoint of back-end server which is defined in the route as shown in the figure below.

```

Route::middleware('auth:sanctum')->group(function () {
  Route::post('/predict', [PredictionController::class, 'predict']);
});

```

Figure 6.3.76 auth::api/predict – POST

This route will trigger the predict function in the PredictionController class. Below is the code snippet of the predict function.

```

public function predict(Request $request)
{
    $data = $request->input('data');
    // // Prepare the input data as JSON
    $inputJson = json_encode($data);
    // return response()->json($inputJson);

    // // Escape the double quotes inside the JSON string
    $escapedInputJson = str_replace('"', '\"', $inputJson);
    // return response()->json($escapedInputJson);
    // // Execute Python script and capture output
    $pythonScriptPath = base_path('scripts/arima_predict.py');
    $command = "python $pythonScriptPath \"$escapedInputJson\"";
    $predictedValues = shell_exec($command);
    $predictedValues = json_decode($predictedValues, true);

    return response()->json(['prediction' => $predictedValues]);
}

```

Figure 6.3.77 PredictionController predict function

This function is responsible of using the data to run the Python script called 'arima_predict.py' that performs the ARIMA prediction. It uses 'shell_exec' to execute the constructed command in the shell. The shell_exec function runs the specified command in the system's shell and captures the standard output as a string. The standard output from the Python script is

assumed to be a JSON-encoded string containing the predicted values. It decodes this JSON string into a PHP array using `json_decode`. Finally, it returns a JSON response containing the predicted values in the 'prediction' field. The predicted values are sent back to the client making the API request.

Below shows the code in `arima_predict.py`.

```
import sys
import json
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

def predict_arima(data_str):
    # Parse the JSON input
    data_dict = json.loads(data_str)

    # Convert input data to a DataFrame
    df = pd.DataFrame(data_dict)

    # Iterate over columns (fields) to predict values
    predicted_values = {}
    for column in df.columns:
        if column != "created_at":
            values = df[column].astype(float)

            # Build ARIMA model
            # You need to adjust these parameters according to your data and model
            choice
            order = (1, 1, 0) # ARIMA order (p, d, q)
            model = ARIMA(values, order=order)
            model_fit = model.fit()
            # print(model_fit.forecast())

            # Make prediction for the next value
            predicted_value = model_fit.forecast()
            first_forecasted_value = predicted_value.iloc[0]
            # print(first_forecasted_value)
            predicted_values[column] = first_forecasted_value

    return predicted_values

if __name__ == "__main__":
    input_data = sys.argv[1]
    predicted_value = predict_arima(input_data)

    # Convert the predicted_value dictionary to JSON format
    predicted_value_json = json.dumps(predicted_value)

    # Print the JSON result
    print(predicted_value_json)
```

Figure 6.3.78 `arima_predict.py`

This python code begins by importing necessary Python modules, and then calls the 'predict_arima' function. This function then performs the ARIMA time series forecasting. It first pre-process the data and convert them into pandas DataFrame for easier manipulation. Due to the data consist of many water parameters, each water parameter corresponds to 1 column of the dataframe. Therefore, it will iterates for each water parameters and do these steps:

- 1) Converts the column's values to floats ('values').
- 2) Specifies an ARIMA order (p, d, q) with (1,1,0).
- 3) Initializes an ARIMA model with the specified order using `ARIMA(values, order=order)`.
- 4) Fits the ARIMA model to the data using `model.fit()`.
- 5) After fitting, it will make a prediction for the next value in the time series using `'model_fit.forecast()'`.
- 6) The predicted value is then store into the `'predicted_values'` dictionary, with the column anme as the key.

After that, the predicted values for each water parameter will be convert to a JSON string and then return the prediction result back to the predict function in PredictionController.

All the results of these operations are then combined into an `'analysisAnHour'` object, and then it is store into the component's state. Now all the required data is being fetched and set to the component's state. Next, all of these analysis and prediction data will be used to render become a water parameter analysis dashboard.

1) Trendline Graphs

```
<TrendlineChart data={this.state.fetchDataDayHour} index={index} />
```

Figure 6.3.79 TrendlineChart component

In the Analysis page, TrendlineChart component is used to draw the trendline graph. It has 2 props, data that stores `this.state.fetchDataDayHour` and an index. The `fetchDataDayHour` is a component's state and it is being set at the `componentDidMount` function. The TrendlineChart component uses VictoryChart component from the victory-native library to draw a trendline graph. The code below shows the code snippet for the VictoryChart component.

```
import React from 'react';
import { View } from 'react-native';
import { VictoryAxis, VictoryChart, VictoryLine, VictoryScatter } from 'victory-native';

const TrendlineChart = ({ data, index }) => {
```

```

// Extract data for the specified field
const selectedRow = data[0];

// Extract the keys "field1", "field2", and "field3" into an array
const fieldKeys = Object.keys(selectedRow).filter(key =>
key.startsWith("field"));

console.log("fieldToDraw.Field Keys Array:", fieldKeys);

console.log("fieldToDraw.data", JSON.stringify(data));
console.log("fieldToDraw.fieldToDraw", JSON.stringify(index));
const fieldID = "field" + (index + 1).toString();
const fieldData = data.map(item => {
  const timestamp = new Date(item.created_at).getTime();
  const fieldValue = item[fieldKeys[index]];

  if (fieldValue !== null && fieldValue !== undefined) {
    return [timestamp, parseFloat(fieldValue)];
  } else {
    return [timestamp, null];
  }
});
// Calculate the slope and y-intercept of the trendline
const n = fieldData.length;
const sumX = fieldData.reduce((acc, [x]) => acc + x, 0);
const sumY = fieldData.reduce((acc, [, y]) => acc + y, 0);
const sumXY = fieldData.reduce((acc, [x, y]) => acc + x * y, 0);
const sumXX = fieldData.reduce((acc, [x]) => acc + x * x, 0);
const slope = (n * sumXY - sumX * sumY) / (n * sumXX - sumX * sumX);
const yIntercept = (sumY - slope * sumX) / n;

// Calculate the trendline points
const trendlineData = fieldData.map(([x]) => [x, slope * x + yIntercept]);

return (
  <View style={{ backgroundColor: "#ffffff" }}>
    <VictoryChart padding={{ top: 20, bottom: 70, left: 50, right: 50 }} >
      <VictoryAxis
        tickFormat={x => new Date(x).toLocaleDateString()}
        style={{
          tickLabels: { angle: -45, dy: 13 } // Rotate tick labels by -45 degrees
        }}
      />
      <VictoryAxis
        dependentAxis
        tickFormat={y => y.toFixed(1)} // Format y-axis labels to two decimal
places
      />
      <VictoryScatter
        data={fieldData}
        x={0}
        y={1}
        style={{ data: { fill: 'blue' } }}
      />
      <VictoryLine
        data={trendlineData}
        x={0}
        y={1}
        style={{ data: { stroke: 'red' } }}
      />
    </VictoryChart>
  </View>
);
};

export default TrendlineChart;

```

Figure 6.3.80 TrendlineChart component

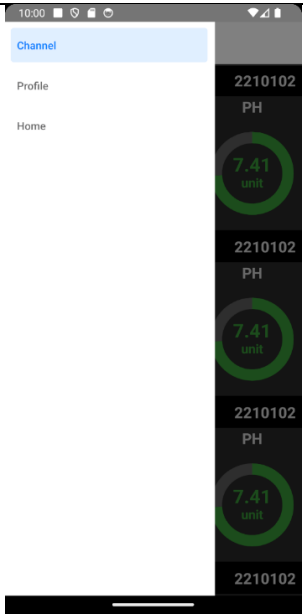
Based on the code above, it first receives two props: 'data' and 'index'. Remember that at the previous section, TrendlineChart is being called with the this.state.FetchDataDayHour, and index. It then extracts the selected field data from the 'data' array based on the provided 'index', and then calculates the slope and y-intercept of the trendline for the selected field using linear regression. These values are used to create a linear trendline for the data. Then it uses the VictoryChart component from the Victory Native Library to create a trendline graph.


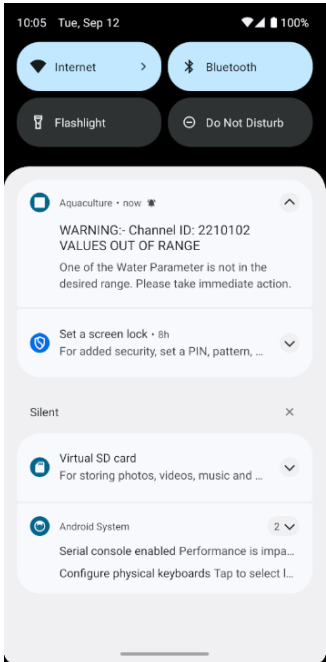
6.3.7 Notification

6.3.7.1 Feature demonstration

Below shows the process of using the Notification function of the Water Quality Monitoring mobile application.

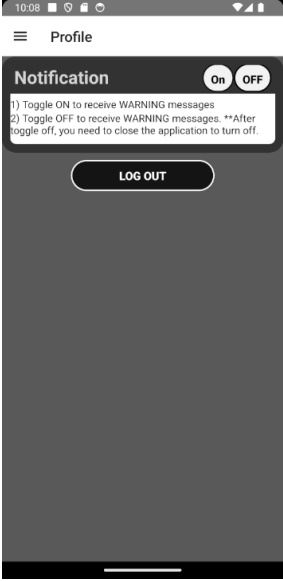
Open notification:

No.	Mobile Screen	Process
1	 <p style="text-align: center;">Figure 6.3.81 Tab Navigator</p>	<ol style="list-style-type: none"> 1) User click on stack icon at the top left to open the side bar. 2) User then click the Profile tab. 3) User then navigate to the Profile page.

<p>1</p>	 <p style="text-align: center;">Figure 6.3.82 Profile page</p>  <p style="text-align: center;">Figure 6.3.83 Notification</p>	<ol style="list-style-type: none"> 1) User click on the “ON” button, to start the background notification service. 2) User then can hear a sound indicates there is notification pops out. 3) User can see the notification of “WARNING” message. 4) User click the notification to start the application.
----------	--	--

Close notification:

No.	Mobile Screen	Process
-----	---------------	---------

1	 <p data-bbox="459 817 794 851">Figure 6.3.84 profile page</p>	1) User click on the “OFF” button and then close the application to apply the stop background service.
---	---	--

6.3.7.2 Code explanation

1) Open notification:

When user click on the “ON” button, it triggers a function called “startBackgroundService”.

```
startBackgroundService = () => {
  const jsonData = this.state.channels.user;

  // Call the extractChannelIds function to get the array of channel_ids
  const thingspeakChannelIds = this.extractChannelIds(jsonData);
  console.log("thingspeakChannelIds", thingspeakChannelIds, jsonData);

  // Start the background service with the extracted channel_ids
  BackgroundTaskModule.startBackgroundService(thingspeakChannelIds);
};
```

Figure 6.3.85 startBackgroundService

When this function is triggered it triggers ‘BackgroundTaskModule.startBackgroundService’ with thingspeakChannelIds as parameters.

```
@ReactMethod
public void startBackgroundService(ReadableArray channelIds) {
  Log.d("BackgroundTaskModule", "Received channelId: " + channelIds);
  ReactApplicationContext context = getReactApplicationContext();
  Intent intent = new Intent(context, BackgroundTaskService.class);
  intent.setAction("com.aquaculture.START_BACKGROUND_SERVICE");

  ArrayList<String> channelIdsList = new ArrayList<>();
```

```

for (int i = 0; i < channelIds.size(); i++) {
    channelIdsList.add(channelIds.getString(i));
}
intent.putStringArrayListExtra("channelIds", channelIdsList);

context.startService(intent);
}

```

Figure 6.3.86 BackgroundTaskModule.startBackgroundService

This function retrieves the `ReactApplicationContext`, which is needed to create and start a service. Then, it creates an “Intent” to start the “BackgroundTaskService” class. Next, it sets the action of the intent to “com.aquaculture.START_BACKGROUND_SERVICE.” and converts the `channelIds` passed from JavaScript (a `ReadableArray`) into a standard Java `ArrayList` of strings (`channelIdsList`). It attaches this list of `channelIds` as an extra to the intent with the key “channelIds.” This should be able to open notification service for many channels based on the `channelIds`.

2) Close notification:

When user click on the “OFF” button in the Profile page, the function named “stopBackgroundService” is then triggered.

```

stopBackgroundService = async () => {
    BackgroundTaskModule.stopBackgroundService();
};

```

Figure 6.3.87 stopBackgroundService

When this function is triggered it triggers ‘BackgroundTaskModule.stopBackgroundService’.

```

@ReactMethod
public void stopBackgroundService() {
    // Retrieve your app's context
    ReactApplicationContext context = getReactApplicationContext();

    // Create an intent for the BackgroundTaskService class
    Intent intent = new Intent(context, BackgroundTaskService.class);
    Log.d("BackgroundTaskModule", "stopBackgroundService");

    // Stop the service using the intent
    context.stopService(intent);
}

```

Figure 6.3.88 BackgroundTaskModule.stopBackgroundService

This function retrieves the `ReactApplicationContext`, which is necessary for interacting with Android components and services within a React Native module. Then, it creates an `Intent` object for the `BackgroundTaskService` class. This intent will be used to identify the service to be stopped. Next, it calls `context.stopService(intent)` to stop the background service specified in the intent. This will effectively terminate the service, stopping any ongoing background tasks associated with it.

Therefore, the start and stop notification function is explained and can be run successfully.

6.4 System Deployment

6.4.1 Data Acquisition Module Deployment

In order to deploy this system, the Data Acquisition System is placed in the Water Tank of Aquafarm. Below figures shows the Data Acquisition Module System is successfully deployed in the Water Tank.

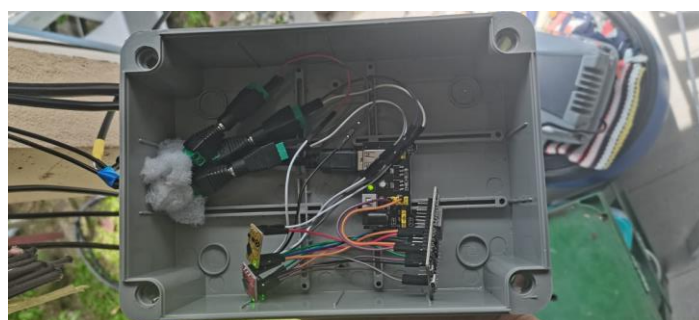


Figure 6.4.1 Data Acquisition system circuit deployed

Figure above shows the complete circuit of the Data Acquisition Module. This circuit is placed in the Junction Box to protect it from raining water.



Figure 6.4.2 Junction Box protects the Data Acquisition Module

Figure above shows the junction box is fully covered the Data Acquisition Module for protection. Moreover, it is placed at a safety position which can avoid touching with water. Besides, this position is near to power source.



Figure 6.4.3 DO, RTD and PH sensors deployed

Figure above shows the DO sensors and pH sensors that are placed into the water tank. The sensors is now become dirty because it has been deployed for a long period to collect the water parameters data.

6.4.2 Communication Module Deployment

In order to deploy the Communication System, we need to register the ThingSpeak account, and set up the Channel Page correctly. Figure below shows the ThingSpeak Channel public view page that has been setup correctly.

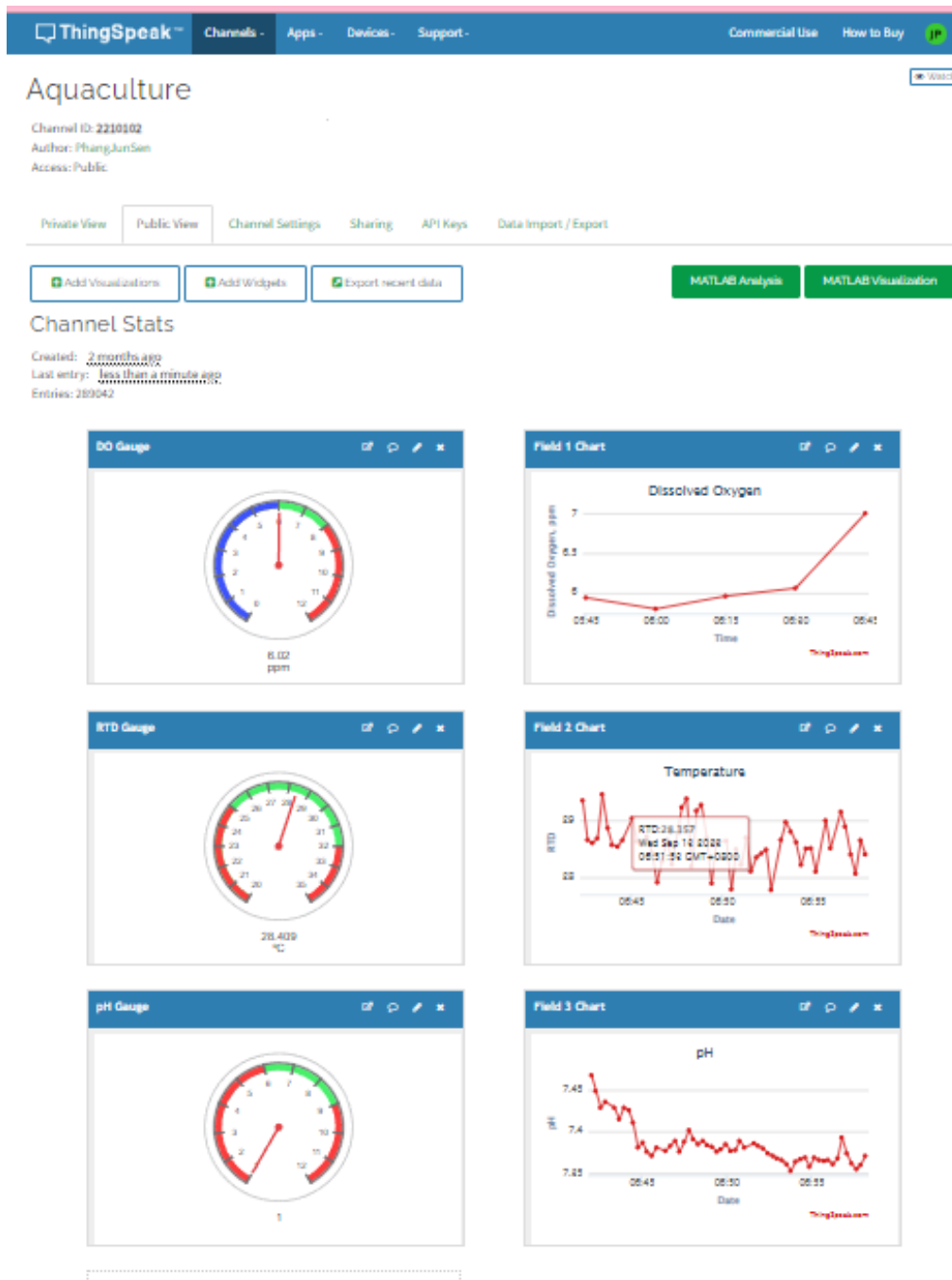


Figure 6.4.4 ThingSpeak Channel Public View

This public view page consists of 3 gauges, and 3 line graphs for each water parameter: DO, pH and RTD. With these gauges and line graphs, it means that the deployment of communication module is completed.

6.4.3 Data Processing Module Deployment

To setup and deploy this module, the Laravel back-end server application should be running in a computer which pass the minimum hardware requirements. Figure below shows the Laravel back-end server is running.

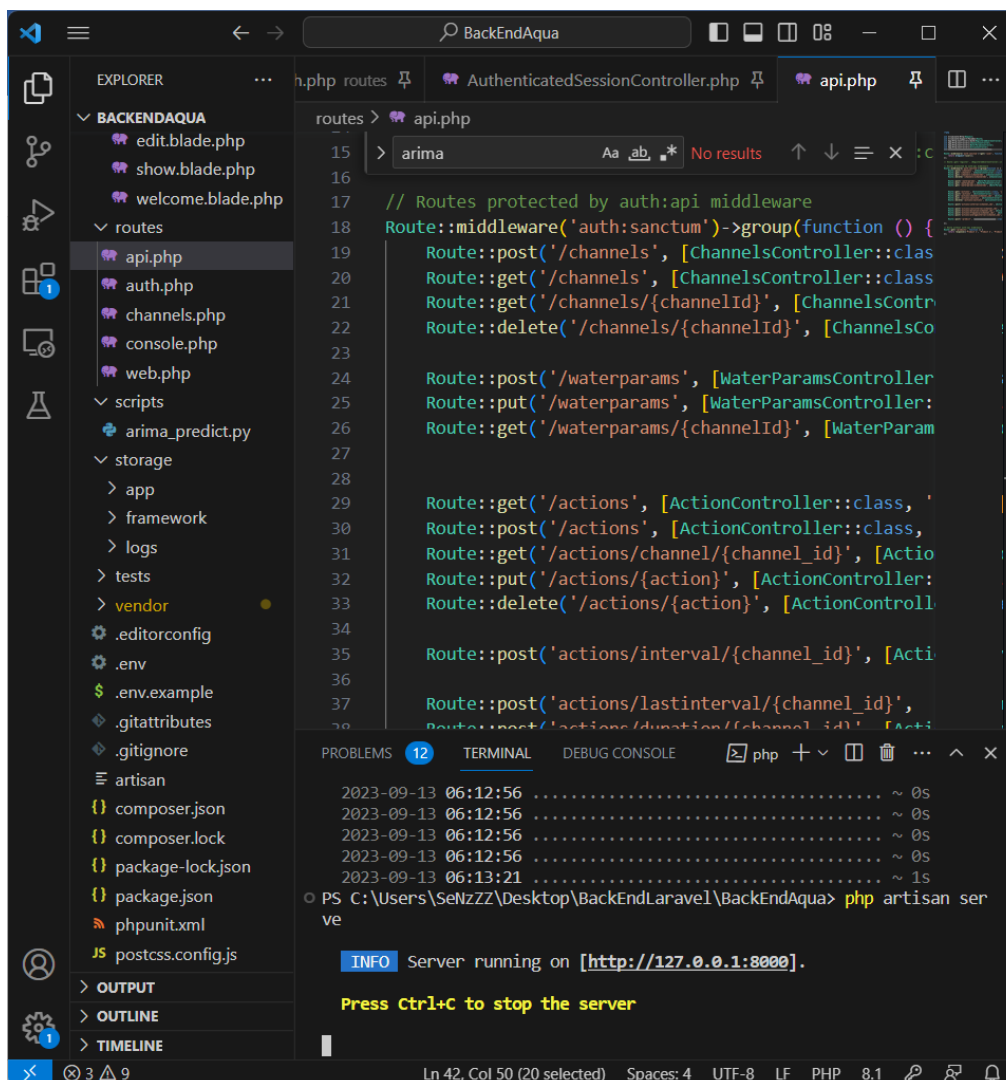


Figure 6.4.5 Laravel back-end server

The terminal below is showing the back-end server is being hosted on [http://127.0.0.1:8000]. This means that it is now hosted locally, and is ready to provide its service. Therefore, the data processing module was also deployed successfully.

6.4.4 User Interface Module Deployment

To deploy the user interface Module, we need to ensure the react-native mobile application is setup correctly. Figures below shows how the User Interface Module is deployed.

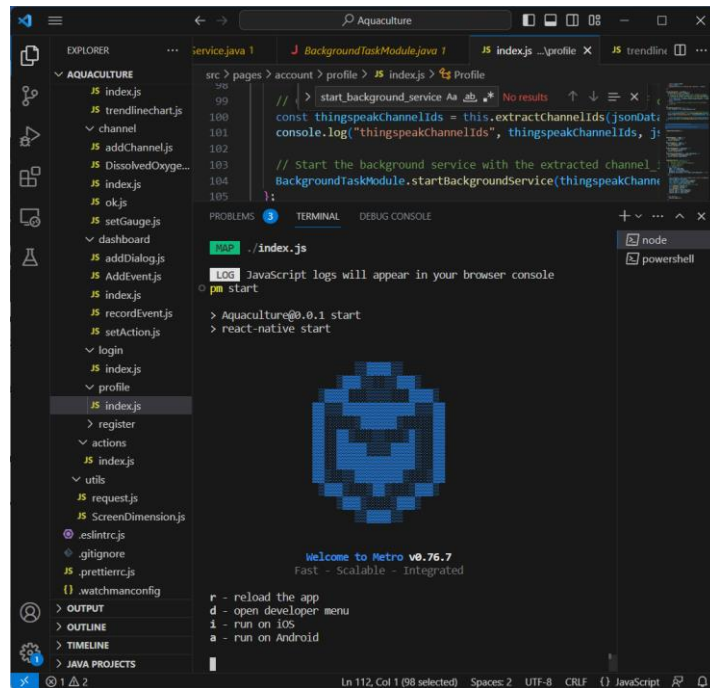


Figure 6.4.6 Metro running

Figure above shows the metro is running. Next, the application is also installed successfully to the android emulator.

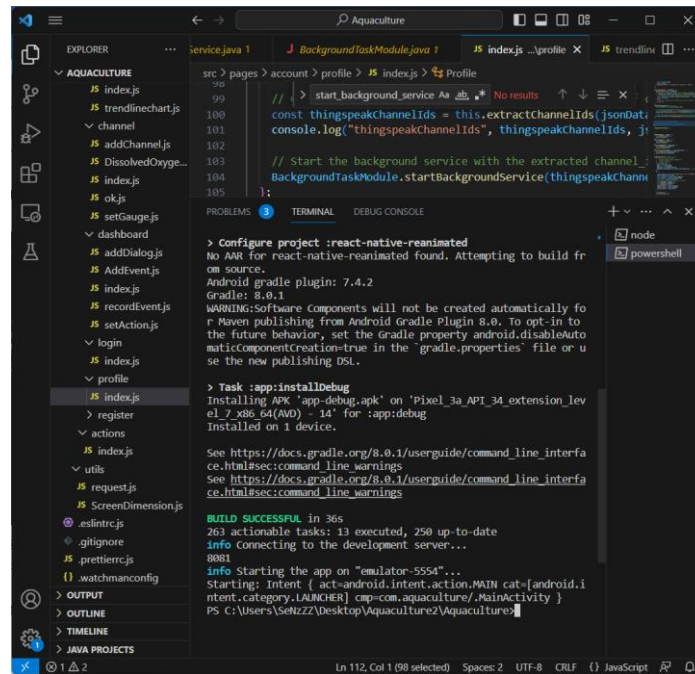


Figure 6.4.7 Build successful

Figure above shows the react-native mobile application is build successful. Then the emulator is then able to run the react-native mobile application is meant that installed and successfully.

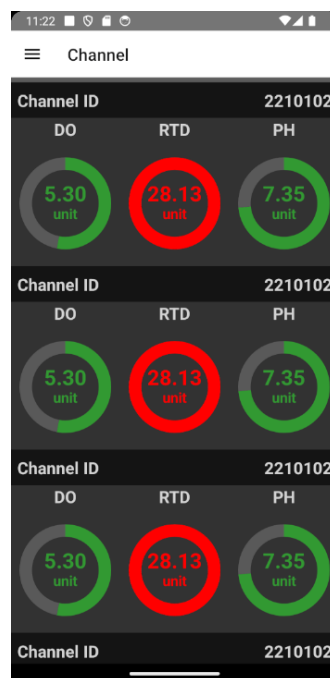


Figure 6.4.8 Android Emulator is running

6.5 Conclusion

Water Quality Monitoring System is successfully implemented. Data Acquisition Module is successfully deployed in the aquafarm, and it is able to perform its task to collect the water parameters and send the data to the ThingSpeak cloud server 15 once. Communication Module, the ThingSpeak cloud server is also successfully deployed and implemented. It can receive the data send by Data Acquisition Module and then store these data to the cloud database. Besides, it also provides REST API for the Data Processing Module and the User Interface Module to retrieve the collected data. Moreover, the Data Processing module is also deployed properly by hosting a Laravel back-end server locally. It helps to provide API endpoints to help User Interface module to perform tasks such as data prediction, analysis and perform data storing and processing .Lastly, the User Interface module is also able to working normally after deployment. It is running in an Android emulator and is able to perform its task by helping the Aquafarmer to monitor the water parameters in the water tank. Therefore, the system deployment is complete successfully.

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

This chapter discuss the system testing. System testing is needed to be carried out to ensure the functional and non-functional requirements of the water quality monitoring system project is met. System testing covers unit testing, performance testing and system usability testing.

7.2 Unit Testing

Unit Testing here is using Postman as the tools to check whether the API Endpoint provided by the back-end server can respond to the request correctly.

7.2.1 Unit Testing for Register

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-101	Register with Invalid Information	<ol style="list-style-type: none"> 1. Key in the name. 2. Key in the email where is existed in database. 3. Key in password with length less than 8. 4. Key in confirm password different from password. 	<pre>{ "name": "testing", "email": "testing321@gmail.com", "password": "ali1234", "password_confirmation": "ali12345" }</pre>	Returns error showing the email has already been taken, the password field confirmation does not match, and the password field must be at least 8 characters.	Pass

		5. Send request to corresponding url.			
UNIT-102	Register with Valid Information	<ol style="list-style-type: none"> 1. Key in the name correctly. 2. Key in the email that is not yet registered. 3. Key in password with length less than 8. 4. Key in confirm password different from password. 5. Send request to corresponding url. 	<pre>{ "name": "testing", "email": "testing001@gmail.com", "password": "testing001", "password_confirmation": "testing001" }</pre>	Returns message showing registration is success and then return the user object, and a complete variables with true values.	Pass

POST http://127.0.0.1:8000/api/register

Params Auth Headers (9) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   "name": "Ali",
3   "email": "ali123@gmail.com",
4   "password": "ali1234",
5   "password_confirmation": "ali12345"
6 }
```

Body 422 Unprocessable Content 107 ms 956 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "errors": {
3     "email": [
4       "The email has already been taken."
5     ],
6     "password": [
7       "The password field confirmation does not match.",
8       "The password field must be at least 8 characters."
9     ]
10  }
11 }
```

Figure 7.2.1 UNIT-101

POST http://127.0.0.1:8000/api/register

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
1 {
2   "name": "testing",
3   "email": "testing001@gmail.com",
4   "password": "testing001",
5   "password_confirmation": "testing001"
6 }
```

body 200 OK 493 ms 967 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Registration successful",
3   "user": {
4     "name": "testing",
5     "email": "testing001@gmail.com",
6     "updated_at": "2023-09-13T14:27:43.000000Z",
7     "created_at": "2023-09-13T14:27:43.000000Z",
8     "id": 6
9   },
10  "complete": true
11 }
```

Figure 7.2.2 UNIT-102

7.2.2 Unit Testing for Login

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-201	Login with not registered email	<ol style="list-style-type: none"> 1. Key in not registered email. 2. Key in any password. 3. Send request to corresponding url. 	<pre>{ "email": "testing002@gmail.com", "password": "testing001", }</pre>	Returns error message showing the email not found.	Pass
UNIT-202	Login with registered email, but incorrect password	<ol style="list-style-type: none"> 1. Key in registered email. 2. Key in incorrect password. 3. Send request to corresponding url. 	<pre>{ "email": "testing001@gmail.com", "password": "testing002", }</pre>	Returns error message showing the password does not match	Pass
UNIT-203	Login with correct credentials.	<ol style="list-style-type: none"> 1. Key in registered email. 2. Key in correct password. 	{	Returns a token and a user object.	Pass

		3. Send request to corresponding url.	"email": "testing001@gmail.com", "password": "testing001", }		
--	--	---------------------------------------	---	--	--

POST http://127.0.0.1:8000/api/login

Params Auth Headers (9) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```

2  ...."email": "testing002@gmail.com",
3  ...."password": "testng001"
4  }
5  // -{
6  // ..... "name": "Ali",
7  // ..... "email": "ali123@gmail.com",
8  // ..... "password": "ali12345",
9  // ..... "password_confirmation": "ali12345"
10 // -{

```

Body 401 Unauthorized 313 ms 803 B Save as Example

Pretty Raw Preview Visualize JSON

```

1  {
2  "message": "Email not found"
3  }

```

Figure 7.2.3 UNIT-201

POST http://127.0.0.1:8000/api/login Send

Params Auth Headers (9) **Body** Pre-req. Tests Settings Cookies

raw JSON Beautify

```
1 {
2   "email": "testing001@gmail.com",
3   "password": "testing002"
4 }
5 // {
6 //   "name": "Ali",
7 //   "email": "ali123@gmail.com",
8 //   "password": "ali12345",
9 //   "password_confirmation": "ali12345"
```

Body 401 Unauthorized 319 ms 811 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Password does not match"
3 }
```

Figure 7.2.4 UNIT-202

The screenshot displays a REST client interface with the following details:

- Request:** Method: POST, URL: http://127.0.0.1:8000/api/login. The request body is in JSON format with the following content:

```
1 {
2   "email": "testing001@gmail.com",
3   "password": "testing001"
4 }
5 // {
6 //   "name": "Ali",
7 //   "email": "ali123@gmail.com",
8 //   "password": "ali12345",
9 //   "password_confirmation": "ali12345"
```
- Response:** Status: 200 OK, Time: 203 ms, Size: 994 B. The response body is in JSON format, displayed in the "Pretty" view:

```
1 {
2   "token": "70|9Wo27ZzuQMnzbPt4AR6M9D2l9816k7PkvHYQ1Z01",
3   "user": {
4     "id": 6,
5     "name": "testing",
6     "email": "testing001@gmail.com",
7     "email_verified_at": null,
8     "created_at": "2023-09-13T14:27:43.000000Z",
9     "updated_at": "2023-09-13T14:27:43.000000Z"
10  }
11 }
```

Figure 7.2.5 UNIT-203

7.2.3 Unit Testing for Channels

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-301	Add new channel with incorrect ThingSpeak Channel ID	<ol style="list-style-type: none"> 1. Key in incorrect Channel ID. 2. Send request to corresponding url with Authorization Header token. 	<pre>{ "channel_id": "2110102" }</pre>	Returns error message showing the Channel ID not found	Pass
UNIT-302	Add new channel with correct ThingSpeak Channel ID	<ol style="list-style-type: none"> 3. Key in correct Channel ID. 4. Send request to corresponding url with Authorization Header token. 	<pre>{ "channel_id": "2210102" }</pre>	Returns a success message and a user objects that contains all their related channels object, and waterparams objects.	Pass

UNIT-303	Retrieve the latest water parameters data	5. Send request to corresponding url with Authorization Header token.	{}	Returns a channel object and feeds object which contains the latest water parameters data	Pass
UNIT-304	Delete existing channel	6. Send request to corresponding url with Authorization Header token.	{}	Returns a message showing this channel is deleted successfully	Pass

The screenshot displays a REST client interface for a POST request to `http://127.0.0.1:8000/api/channels`. The request body is a JSON object: `{ "channel_id": "2110102" }`. The response is a 400 Bad Request with a response time of 1225 ms and a body size of 348 B. The response body is a JSON object: `{ "error": "Channel ID not found" }`.

POST `http://127.0.0.1:8000/api/channels` Send

Params Auth Headers (11) **Body** Pre-req. Tests Settings Cookies

raw JSON Beautify

```
1 {
2   "channel_id": "2110102"
3 }
```

Body 400 Bad Request 1225 ms 348 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "error": "Channel ID not found"
3 }
```

Figure 7.2.6 UNIT-301

The screenshot displays a REST client interface for a POST request to `http://127.0.0.1:8000/api/channels`. The request body is a JSON object with a `channel_id` of `"2210102"`. The response is a JSON object indicating a successful channel creation, including user details and a list of channels.

Request:

```
POST http://127.0.0.1:8000/api/channels
{
  "channel_id": "2210102"
}
```

Response:

```
{
  "message": "Channel created successfully",
  "user": {
    "id": 6,
    "name": "testing",
    "email": "testing001@gmail.com",
    "email_verified_at": null,
    "created_at": "2023-09-13T14:27:43.000000Z",
    "updated_at": "2023-09-13T14:27:43.000000Z",
    "channels": [
      {
        "id": 12,
        "user_id": "6",
        "channel_id": "2210102",
      }
    ]
  }
}
```

Figure 7.2.7 UNIT-302

GET ▼ https://api.thingspeak.com/channels/2210102/feeds.json?results=1&timezone=Asia%2FKuala_Lumpur Send ▼

Params ● Authorization ▼ **Headers (6)** Body Pre-request Script Tests Settings ▼ Cookies

Key	Value	Description	...	Bulk Edit	Presets
Key	Value	Description			

Body ▼ Cookies Headers (13) Test Results 200 OK 991 ms 881 B Save as Example ...

Pretty Raw Preview Visualize **JSON** ▼ ☰

```

1  {
2    "channel": {
3      "id": 2210102,
4      "name": "Aquaculture",
5      "latitude": "0.0",
6      "longitude": "0.0",
7      "field1": "00",
8      "field2": "RTD",
9      "field3": "pH",
10     "created_at": "2023-07-03T10:30:58+08:00",
11     "updated_at": "2023-08-27T23:50:43+08:00",
12     "last_entry_id": 292585
13   },
14   "feeds": [
15     {
16       "created_at": "2023-09-13T23:11:03+08:00",
17       "entry_id": 292585,
18       "field1": "6.48000",
19       "field2": null,
20       "field3": null
21     }
22   ]
23 }

```

Figure 7.2.8 UNIT-303

The screenshot displays a REST client interface. At the top, a request is configured with the method **DELETE** and the URL `http://127.0.0.1:8000/api/channels/12`. A **Send** button is visible to the right. Below the request bar, the **Headers** tab is selected, showing a table with 10 headers, one of which is checked. The checked header is **Authorization** with the value `Bearer 70|9Wo27ZzuQ...`. Below the headers, the **Body** tab is selected, showing a successful response with status `200 OK`, a response time of `112 ms`, and a body size of `349 B`. The response body is displayed in **JSON** format, showing a single object with the message `"message": "Channel deleted successfully"`.

DELETE ▼ | `http://127.0.0.1:8000/api/channels/12` **Send** ▼

Params Auth Headers (10) Body ● Pre-req. Tests Settings ⋮

Headers 👁 9 hidden

	Key	Value	⋮ Bulk Edit Presets ▼
<input checked="" type="checkbox"/>	Authorization	Bearer 70 9Wo27ZzuQ...	
	Key	Value	Description

Body ▼ 🌐 200 OK 112 ms 349 B 💾 Save as Example ⋮

Pretty Raw Preview Visualize **JSON** ▼ 🔄 📄 🔍

```
1 ↳  
2   "message": "Channel deleted successfully"  
3 ↳
```

Figure 7.2.9 UNIT-304

7.2.4 Unit Testing for Dashboard

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-401	Retrieve the time period from now until the last activity.	7. Send request to corresponding url with Authorization Header token.	{ "title": "Change Water" }	Returns the time period from now until the last change water activity in days and hours.	Pass
UNIT-402	Retrieve the duration data of an activity within this month.	8. Send request to corresponding url with Authorization Header token.	{ "title": "Change Water" }	Returns the total duration of the current month, average total duration of each months, average duration to carry out change water activity, and total duration carried out for this activity last month.	Pass

UNIT-403	Retrieve the frequency data of an activity.	9. Send request to corresponding url with Authorization Header token.	{ "title": "Change Water" }	Returns the frequency of this change water activity in current month, last month, and the average from every months.	Pass
UNIT-404	Retrieve the average intervals data between all records of an activity.	10. Send request to corresponding url with Authorization Header token.	{ "title": "Change Water" }	Returns the average time intervals between all records of the change water activity.	Pass

POST ▼ | http://127.0.0.1:8000/api/actions/lastinterval/2210102 Send ▼

Params Auth Headers (10) Body ● Pre-req. Tests Settings Cookies

Headers 👁 9 hidden

	Key	Value	D...	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/>	Authorization	Bearer 71 a5XIMoDrrSIMjLXLu...				
	Key	Value	Description			

Body ▼ 🌐 200 OK 96 ms 327 B 📄 Save as Example ⋮

Pretty Raw Preview Visualize JSON ▼ ☰ 📄 🔍

```
1 {
2   "days": 0,
3   "hours": 1
4 }
```

Figure 7.2.10 UNIT-401

The screenshot displays a REST client interface. At the top, a dropdown menu is set to **POST** and the URL is `http://127.0.0.1:8000/api/actions/duration/2210102`. A blue **Send** button is to the right. Below the URL bar, tabs for **Params**, **Auth**, **Headers (10)**, **Body** (selected), **Pre-req.**, **Tests**, and **Settings** are visible. On the right, there are **Cookies** and **Beautifuly** options. The **Body** tab is active, showing a **raw** view of the request body in **JSON** format. The request body is a JSON object with a `"title"` property set to `"Change Water"`. Below the request, the **Body** section shows the response status: **200 OK**, **121 ms**, and **449 B**. There are icons for **Save as Example** and a menu. The response body is shown in a **Pretty** JSON view, containing the following data:

```
1 {
2   "total_duration_current_month": 2.5,
3   "average_total_duration_other_months": 1,
4   "average_duration_to_carry_out": 0.5,
5   "total_duration_last_month": 1
6 }
```

Figure 7.2.11 UNIT-402

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL is 'http://127.0.0.1:8000/api/actions/frequency/2210102'. A blue 'Send' button is to the right. Below this, a navigation bar includes 'Params', 'Auth', 'Headers (10)', 'Body' (which is selected and highlighted with a red underline), 'Pre-req.', 'Tests', 'Settings', and 'Cookies'.

The 'Query Params' section is visible as a table with the following structure:

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

The 'Body' section shows a response status of '200 OK', a response time of '87 ms', and a response size of '391 B'. There are icons for 'Save as Example' and a search icon. The response is displayed in 'Pretty' JSON format:

```
1 {  
2   "current_month_frequency": 5,  
3   "last_month_frequency": 2,  
4   "average_total_frequency": 3.5  
5 }
```

Figure 7.2.12 UNIT-403

The screenshot displays a REST client interface. At the top, a dropdown menu is set to **POST** and the URL is `http://127.0.0.1:8000/api/actions/averageinterval/2210102`. A blue **Send** button is to the right. Below the URL bar, tabs for **Params**, **Auth**, **Headers (10)**, **Body** (selected), **Pre-req.**, **Tests**, and **Settings** are visible. On the right side, there are **Cookies** and **Beautify** options. The **Body** tab is active, showing a **raw** view of the request body in **JSON** format. The request body is a JSON object with a `"title"` property set to `"Change Water"`. Below the request body, the response status is **200 OK** with a response time of **78 ms** and a size of **362 B**. There are also icons for **Save as Example** and a menu icon. The response body is shown in a **Pretty** view, displaying a JSON object with `"average_duration_days": 6` and `"average_duration_hours": 22`.

```
POST http://127.0.0.1:8000/api/actions/averageinterval/2210102
```

```
{
  "title": "Change Water"
}
```

200 OK 78 ms 362 B Save as Example

```
{
  "average_duration_days": 6,
  "average_duration_hours": 22
}
```

Figure 7.2.13 UNIT-404

7.2.5 Unit Testing for Activity Record

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-501	Add new event	11. User fill in the add event form. 12. User hit the submit button 13. Send request to the corresponding URL with Authorization Header token.	<pre>{ "channel_id": "2210102", "title": "Change Water", "description": "", "start_time": "2023-09-13T14:23:56.030Z", "end_time": "2023-09-13T14:53:56.030Z", "color": "red" }</pre>	Returns the created activity record objects	Pass

UNIT-502	Retrieve all recorded activities	14. Send request to the corresponding URL with Authorization Header token.	{}	Returns all the activities recorded in the database in a list	Pass
----------	----------------------------------	--	----	---	------

The screenshot displays a REST client interface. At the top, a dropdown menu is set to 'POST' and the URL is 'http://127.0.0.1:8000/api/actions'. A 'Send' button is visible to the right. Below the URL bar, there are tabs for 'Params', 'Auth', 'Headers (10)', 'Body', 'Pre-req.', 'Tests', and 'Settings'. The 'Body' tab is selected, and the content is shown in 'JSON' format. The request body is a JSON object with the following fields: 'channel_id', 'title', 'description', 'start_time', 'end_time', and 'color'. Below the request, the response is shown in 'Pretty' format, which is a JSON object with fields: 'channel_id', 'title', 'description', 'start_time', 'end_time', 'color', 'updated_at', 'created_at', and 'id'. The response status is '201 Created' with a response time of '105 ms' and a size of '564 B'. There are also icons for 'Save as Example' and a search icon.

```
POST http://127.0.0.1:8000/api/actions

{
  "channel_id": "2210102",
  "title": "Change Water",
  "description": "",
  "start_time": "2023-09-13T14:23:56.030Z",
  "end_time": "2023-09-13T14:53:56.030Z",
  "color": "red"
}
```

Body 201 Created 105 ms 564 B Save as Example

```
Pretty Raw Preview Visualize JSON

{
  "channel_id": "2210102",
  "title": "Change Water",
  "description": null,
  "start_time": "2023-09-13T14:23:56.030Z",
  "end_time": "2023-09-13T14:53:56.030Z",
  "color": "red",
  "updated_at": "2023-09-13T15:23:58.000000Z",
  "created_at": "2023-09-13T15:23:58.000000Z",
  "id": 13
}
```

Figure 7.2.14 UNIT-501

The screenshot displays a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/api/actions/channel/22101 ...
- Status:** 200 OK, 74 ms, 3.51 KB
- Headers (10):** One header is visible: Authorization: Bearer 71|a5XIMoDrrS...
- Body:** A JSON array of two objects. The first object has the following properties:
 - id: 1
 - channel_id: "2210102"
 - title: "Change Water"
 - description: null
 - start_time: "2023-08-02T23:05:00.000Z"
 - end_time: "2023-08-02T23:35:00.000Z"
 - color: "red"
 - created_at: "2023-08-30T23:08:24.000000Z"
 - updated_at: "2023-08-30T23:08:24.000000Z"
- The second object in the array has the following properties:
 - id: 2
 - channel_id: "2210102"
 - title: "Change Water"
 - description: null
 - start_time: "2023-08-08T23:08:00.000Z"
 - end_time: "2023-08-08T23:38:00.000Z"
 - color: "red"
 - created_at: "2023-08-30T23:08:47.000000Z"

Figure 7.2.15 UNIT-502

7.2.6 Unit Testing for Analysis

Test Module	Extraction Module		Test Title	File Upload from the Web Application	
Test Case ID	Test Case Description	Execution Steps	Test Data	Expected Result	Status
UNIT-601	Predict the next data for all the parameters using ARIMA time series machine learning model.	15. Send request to the corresponding URL with Authorization Header token.	{ "data":[{ "created_at": "2023-08 29T19:40:00+08:00", "field1":"9.40", "field2":"28.66", "field3":"14.00"}],{... },...] }	Returns the prediction of the next water parameters data.	Pass

The screenshot displays a REST client interface for a POST request to `http://127.0.0.1:8000/api/predict`. The request body is a JSON object with a `data` array containing three objects, each with `created_at`, `field1`, `field2`, and `field3` fields. The response body is a JSON object with `prediction`, `field1`, `field2`, and `field3` fields. The status is 200 OK, with a response time of 2.69 s and a size of 406 B.

Request:

```
1 {
2   ... "data": [
3     { "created_at": "2023-08-29T19:40:00+08:00", "field1": "9.40",
4       "field2": "28.66", "field3": "14.00" },
5     { "created_at": "2023-08-29T19:50:00+08:00", "field1": "10.17", "field2": "28.60", "field3": "8.44" },
6     { "created_at": "2023-08-29T20:00:00+08:00", "field1": "10.52", "field2": "28.48", "field3": "7.62" },
7     { "created_at": "2023-08-29T20:10:00+08:00",
```

Response:

```
1 {
2   "prediction": 5
3   "field1": 10.300078520991455,
4   "field2": 27.623185668327704,
5   "field3": 7.223580203231105
6 }
7 }
```

Metadata: 200 OK, 2.69 s, 406 B, Save as Example

Figure 7.2.16 UNIT-601

7.3 Performance Testing

7.3.1 Register an account

1) Reponse Time Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-101	Measure the response time for account registration	<ol style="list-style-type: none"> 1. Fill up the registration form correctly. 2. Submit the registration form. 3. Record the total response time for the application to register an account. 	The total execution time should be lower than 3 seconds to complete the account registration process.	Pass 1967ms

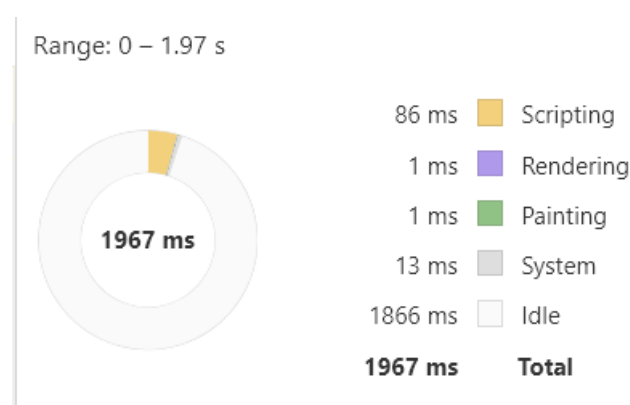


Figure 7.3.1 PFT-101 Register an account (1967ms)

2) Memory Usage Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-102	Measure the memory usage for account registration	<ol style="list-style-type: none"> 4. Fill up the registration form correctly. 5. Submit the registration form. 6. Record the total memory usage during account registration process. 	The total memory usage should be lower than 20MB to complete the account registration process.	Pass 12.76MB

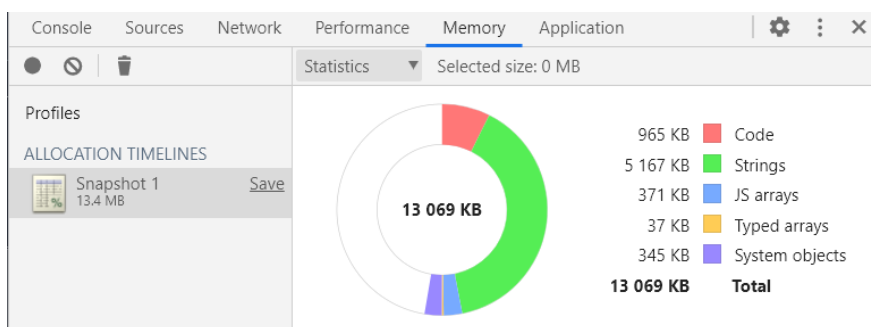


Figure 7.3.2 PFT-102 Register an account (13069KB)

7.3.2 Login

3) Reponse Time Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-201	Measure the response time for login process	7. Fill up the login form correctly. 8. Submit the login form. 9. Record the total response time for the application to login.	The total execution time should be lower than 3 seconds to complete the account login process.	Pass 1826ms

Range: 0 – 1.83 s

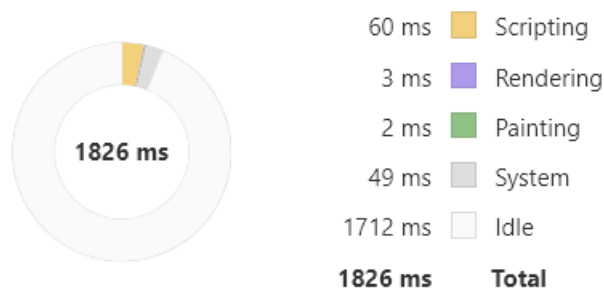


Figure 7.3.3 PFT-201 Login Process(1826ms)

4) Memory Usage Testing

Test Case ID	Test Description	Steps	Expected Result	Status
--------------	------------------	-------	-----------------	--------

PFT-202	Measure the memory usage for login process.	10. Fill up the login form correctly. 11. Submit the login form. 12. Record the total memory usage during account login process.	The total memory usage should be lower than 20MB to complete the account login process.	Pass 12.94MB
---------	---	--	---	-----------------

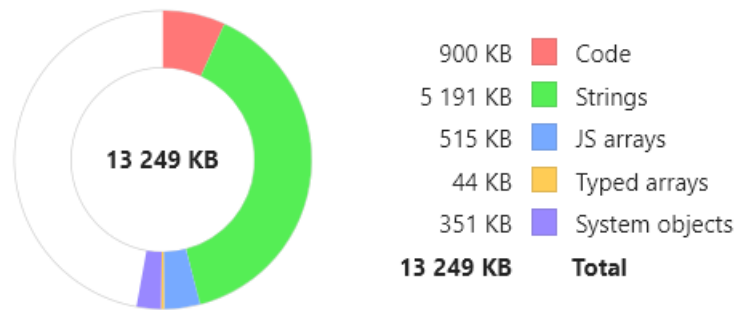


Figure 7.3.4 PFT-102 Register an account (13249KB)

7.3.3 Dashboard

5) Reponse Time Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-301	Measure the response time for Dashboard to be render.	13. User navigate to dashboard page 14. Record the total response time to render the dashboard.	The total execution time should be lower than 6 seconds to render the dashboard.	Pass 4089ms

Range: 0 – 4.09 s

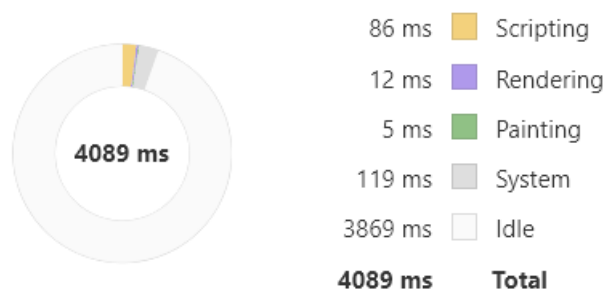


Figure 7.3.5 PFT-301 Load Dashboard Process(4089ms)

6) Memory Usage Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-302	Measure the memory usage for the dashboard to render.	15. User navigate to dashboard page 16. Record the total memory usage to render the dashboard.	The total memory usage should be lower than 20MB to complete the dashboard rendering.	Pass 13.35MB

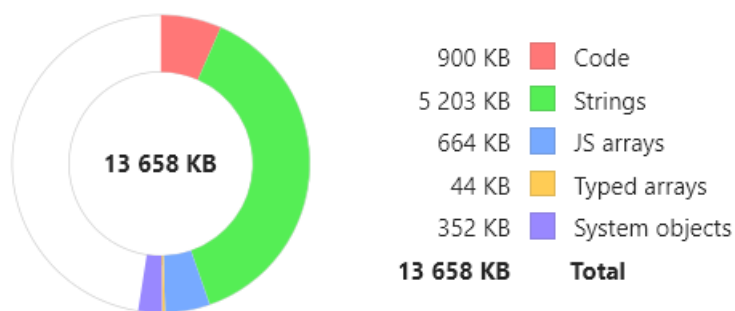


Figure 7.3.6 PFT-302 Load Dashboard Process (13658KB)

7.3.4 Add Event

7) Reponse Time Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-401	Measure the response time for the process of adding a new event and then render the dashboard.	17. User fill in the add event form. 18. User click the submit button. 19. Record the total response time to add the event and render the dashboard.	The total execution time should be lower than 3 seconds to add event and then re-render the dashboard page.	Pass 2339ms

Range: 0 – 2.34 s

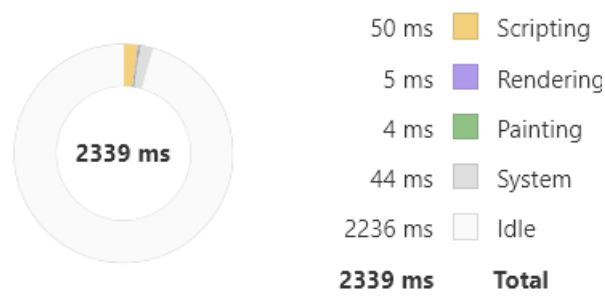


Figure 7.3.7 PFT-401 Add Eevent Process(2339ms)

8) Memory Usage Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-402	Measure the memory allocation for the process of adding a new event and then render the dashboard.	20. User fill in the add event form. 21. User click the submit button. 22. Record the total memory usage used to add the event and render the dashboard.	The total memory usage should be lower than 20MB to add event and then re-render the dashboard page.	Pass 13.19MB

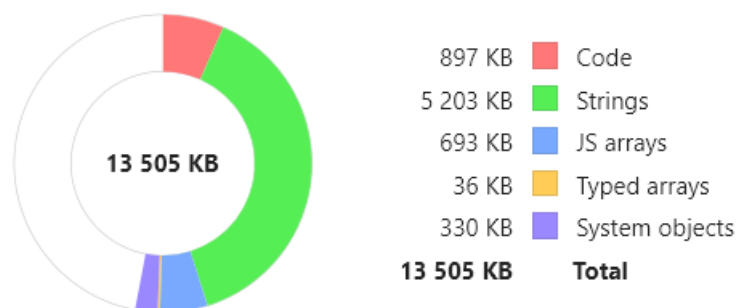


Figure 7.3.8 PFT-402 Add Event Process (13505KB)

7.3.5 Analysis

9) Reponse Time Testing

Test Case ID	Test Description	Steps	Expected Result	Status

PFT-501	Measure the response time for the process of get the analysis data and render the analysis dashboard.	23. User navigate to the analysis page. 24. Record the total response time to get the analysis data and render the analysis dashboard.	The total execution time should be lower than 8 seconds to get the analysis data and render the analysis dashboard.	Pass 6851ms
---------	---	---	---	----------------

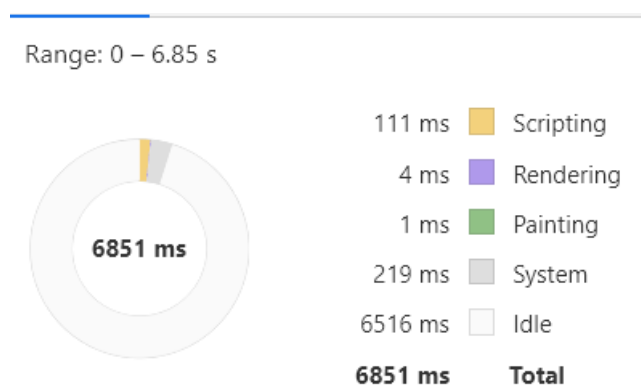


Figure 7.3.9 PFT-501 Analysis Process(6851ms)

10) Memory Usage Testing

Test Case ID	Test Description	Steps	Expected Result	Status
PFT-502	Measure the memory usage for the process of get the analysis data and render the analysis dashboard.	25. User navigate to the analysis page. 26. Record the total memory usage to get the analysis data and render the analysis dashboard.	The total memory usage should be lower than 20MB to get the analysis data and render the analysis dashboard.	Pass 13.19MB

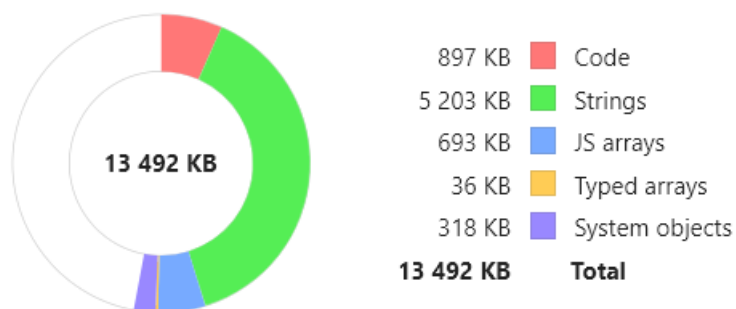


Figure 7.3.10 PFT-502 Analysis Process (13492KB)

7.3.6 Laravel Back-end Server API Endpoint Response Time Testing

API Endpoint	Test Case ID	Response time
POST - api/register	UNIT-102	493ms
POST - api/login	UNIT-203	203ms
POST - api/channels	UNIT-302	1181ms
DELETE – api/channels/{id}	UNIT-304	349ms
POST - api/actions	UNIT-501	105ms
GET - api/actions/channel/{channel_id}	UNIT-502	74ms
POST - api/predict	UNIT-601	2.69s
POST - api/actions/lastinterval/{channel_id}	UNIT-401	96ms
POST - api/actions/duration/{channel_id}	UNIT-402	121ms
POST - api/actions/frequency/{channel_id}	UNIT-403	87ms
POST - api/actions/averageinterval/{channel_id}	UNIT-404	78ms

7.4 System Usability Test

The System Usability Scale (SUS) is used to perform system usability test in this project. After a brief demonstration of the functionality of this Water Quality Monitoring mobile application, respondents need to explore and use all the functionality of the application by completing the test scenario defined at below. After that, respondents need to fill in the User Satisfactory Survey. It has 2 sections. In section A, respondents need to answer 10 questions of the questionnaire. Each question has 5 answers, and their marks range from 1 – 5, the answer categories listed as below:

- 1) Strongly Disagree – 1
- 2) Disagree – 2
- 3) Neutral – 3
- 4) Agree – 4
- 5) Strongly Agree – 5

Next, Section B requires user to answer 3 open-ended questions. Finally, the result of the usability testing will be evaluate and calculated based on the survey results. The sample user satisfactory survey form template used in this SUT can be found in Appendix A.

7.4.1 Test Scenario for Usability Testing

Table 7.4.1 Tables of Test Scenarios

Test Scenario to act as aquafarmer
Scenario 1 – Register a new account
Imagine you are a aquafarmer that is new to this system to monitor the water quality in a water tank. You want to make use of this mobile application to see the water parameters data collected by the sensors. What would you do so that you can have the access to use this system?
Scenario 2 – Login to his account
Imagine you are a aquafarmer, you already have an account to use this mobile application. What should you do so that you can use the mobile application features?
Scenario 3 – Add a new channel for monitoring
Imagine you are a aquafarmer, you want to use this mobile application to monitor the water quality of a water tank that is already implements the sensors module to collects data, and have a channel to monitor it in ThingSpeak cloud service. What should you do so that you can monitor water parameters of that particular water tank?
Scenario 4 – Update channel settings
Imagine you are a aquafarmer, you want to monitor water parameters of the channel correctly, so you need to do some settings to the channel. Now you want to change the maximum and minimum range of the gauge, and add gauge and line graphs to the dashboard, how can you do that?
Scenario 4 – Delete a channel

Imagine you are a aquafarmer, you don't want to monitor the water quality of this water tank anymore. What should you do to not monitor this water tank with your account.
Scenario 5 – View the dashboard of a channel
Imagine you are a aquafarmer, you want to view the water confition of your aquafarm. What should you do to have an overall view of water quality of all the water tanks.
Scenario 6 – View activity history and analysis
Imagine you are a aquafarmer, you want to gain more insights to decide the time you need to change the water in this water tank. What should you do so that you can exploit this water quality monitoring application to help you on deciding when to change the water?
Scenario 7 – Record new activity
Imagine you are a aquafarmer, you just successfully change the water in this water tank, and you want to record this action you have taken. What should you do to use this application to record your taken action towards this water tank?
Scenario 8 – View analysis for the water quality
Imagine you are a aquafarmer, you want to analyze the dissolved oxygen level in this water tank. What should you do with this application to allow you to immediately see the analysis of the dissolved oxygen in this water tank?
Scenario 9 – Toggle notification on and off
Imagine you are a aquafarmer, you want to get notify when there is problem with the water tanks. What should you do to use this application to help you?
Scenario 10 – Logout
Imagine you are a aquafarmer, you want to login to other account so you need to logout the current account. What should you do so that you are able to logout this application?

7.4.2 Result of Usability Testing

3 respondents were participating in this usability testing. In Section A, they need to explore the application based on the 10 test scenarios specified in section 7.4.2. Then they need to fill up the satisfactory survey form shown in section 7.4.1. Their recorded responses can be found at Appendix B.

In order to calculate the SUS score, following are the formula to calculate SUS score:

- $X = \text{Sum of the points for all odd-numbered questions} - 5$
- $Y = 25 - \text{Sum of the points for all even-numbered questions}$
- $\text{SUS Score} = (X + Y) \times 2.5$

Average SUS score can be calculated by:

- $\text{Average SUS Score} = \text{Total SUS Score} / \text{Number of participants}$

After getting the average of SUS score, we can use the tables below to perform interpretation to understand usability performance of this system (Will T, n.d.).

Table 7.4.2 Template of User Satisfactory Survey

SUS score	Grade	Adjective Rating
> 80.3	A	Excellent
68 – 80.3	B	Good
68	C	Okay
51-68	D	Poor
<51	F	Awful

After complete all the survey and calculation of the usability score, the results is shown at the table below:

Table 7.4.3 SUS results

Participants Name	Usability Score for each question										Total		Marks
	1	2	3	4	5	6	7	8	9	10	Odd	Even	
Seow Ding Han	5	2	4	2	4	2	5	2	4	2	22	15	80.0
Tang Chu Lin	4	2	4	4	5	4	5	1	4	1	22	17	75.0
Poey Wei Jun	4	1	3	4	4	2	5	1	4	2	20	15	67.5
Average SUS Score												74.17	

In Section B, respondents need to answer these open-ended questions:

- 1) What do you like best about the system?
- 2) What do you like least about the system?
- 3) Do you have any suggestions for improving the current system?

From their response in Appendix B, the summary of answers for each questions will be arrange in the tables below.

Table 7.4.4 Summary of Respondents' Most Liked Features of the System

Summary of Respondents' Most Liked Features of the System
The features provided by the system can be learn and use easily.
The dashboard design able to provide user a great monitoring experience.

The calendar components used to implement the record activity features is useful.

Table 7.4.5 Summary of Respondents' Least Liked Features of the System

Summary of Respondents' Least Liked Features of the System
The activity history features is missing functionality for edit and delete activities.
The notification system is not complete, such as user are unable to open notification selectively among all the channels.
The notification system is not user-friendly, such as there is no toggle on/off switch to use notification system.

Table 7.4.6 Summary of Respondents' Suggestion to improve the system

Summary of Respondents' Suggestion to improve the system
The activity history features should include delete and edit function.
The notification on/off button should be replaced with switch, and can selectively choose with channel to on/off notification.

In conclusion, the average SUS score achieved by this system is 74.17 which is placed under the Grade B. This means that the overall user experience of using this system is Good, but not too impressive. Therefore, there is more improvements can be done to this system to achieve impressive user experience in the future. Besides, we also received open-ended feedbacks about the best function, worst function, and suggestions from respondents throughout this System Usability Testing. Overall, the best thing about this system is it can learn to use easily, dashboard is simple and informative and the record activity process in this system is well-designed. The worst thing about this system is it missing some required functionality such as delete and edit the activity history, and notification system is not designed well. Therefore, the suggestion from the respondents are add more function to the activity history and improve the design of notification system.

CHAPTER 8

CONCLUSION & RECOMMENDATION

8.1 Conclusion

This chapter discuss the achievement, limitation, and recommendation to improve the system. This system has successfully achieve all the goals listed in Chapter 1, they are:

1. To analyze existing available tools related to aquafarming to develop ideas for designing usable water monitoring systems.
2. To perform analysis on the of water quality of the water tank to discover the trends and useful information to assist in decision making on the water quality monitoring process.
3. To develop a mobile application that provides a dashboard that displays all the water parameter readings of the aquaculture system anytime anywhere to allow user to check the current water condition in water tank.
4. To evaluate the water quality of the fish tank so that user are able to take immediate initiative to maintain its water quality to optimize fish growth and prevent undesirable condition.

Objectives 1 is completed by successfully design and develop a complete water monitoring systems using various hardware tools and software tools. Besides, Objectives 2 is achieved by the water quality monitoring system provides various analysis features, such as statistical analysis on water quality, generating trendline graphs to understand the trends of the water quality, use machine learning model to perform prediction on the next reading. Not only water parameters, there is also analysis on the activity history records to help user find out the pattern of water quality monitoring process. Next, Objectives 3 is completed by the water quality monitoring mobile application provides simple and informative dashboard to monitor water quality. The last objectives is also achieved by having a notification features provided by the mobile application, it creates a background service which can still notify user about the water quality if the application is closed.

8.2 Limitation and Recommendation for future work

Throughout the system testing phases, there are many limitations were discovered. Based on the limitation, some recommendation has been made by me and the testers involves is SUT to improve the system for future development. Below shows the tables of limitations and recommendations for this system.

Table 8.2.1 Limitation and Recommendation

Limitation	Recommendation
The mobile application and the back-end server is currently deployed and host locally. So user cannot use the mobile application in a real android device and cannot access the back-end server service through internet.	The back-end server should be hosted in a web server or cloud server, to make it have a domain to access the server. The mobile application project should be export to become a executable mobile application, so that user can install this applicatin in their android smartphone to use this application.
This system is currently able to monitor fixed 3 water paramers, dissolved oxygen, pH values, and temperatures. It is unable to work properly if the water tank have lesser water parameters or more paramters to monitor. For example, the water tank only have 2 sensors, however the mobile application is designed with 3 sensors fixed. Therefore this will cause some problems in the mobile application to works properly.	The system should be designed to be flexible to contain the water parameters of a water tank for monitoring in the mobile application. For example, if the water tank only have 2 sensors, DO and PH, the mobile application should works with only 2 water parameters not including the temperature.
The notification system is incomplete such as it cannot selectively choose a single channel or a groups of channel	The system should be designed to allow users can selectively choose which channels should toggle on or off the notifications service.

to open notification from a list of channels.	
When user record the activities with wrong information, the recorded activities is unable to be edit or deleted, which cause logging incorrect activiy and affect the accuracy and correctness of analysis.	The mobile application should include the edit and delete activity record function.

REFERENCES

Azlan Othman, N., Salwa Damanhuri, N., Syafiq Mazalan, M. A., Addayani Shamsuddin, S., Hussaini Abbas, M., & Chiew Meng, B. C., 2020. Automated water quality monitoring system development via LabVIEW for aquaculture industry (Tilapia) in Malaysia. *Indonesian Journal of Electrical Engineering and Computer Science*, 20(2), pp.805.
<https://doi.org/10.11591/ijeecs.v20.i2.pp805-812>

Zhou, X., 2019. Brief overview of world aquaculture production An update with latest available 2017 global production data. *FAO Aquaculture Newsletter*, pp6–8.

Masser, M. P., Rakocy, J., & Losordo, T. M., 1992. Recirculating Aquaculture Tank Production Systems Management of Recirculating Systems. *Southern Regional Aquaculture Center*, 452.

Purina Animal Nutrition, 2023. *Sudden Loss of Fish*. Management : Pond Management [Online]. Available at <https://www.purinamills.com/fish-and-aquatics-feed/education/detail/sudden-loss-of-fish#:~:text=Some%20fish%20culture%20systems%20rely,minutes%20of%20a%20systems%20failure> [Accessed on 18 March 2023].

Kenekar, A., 2020. *Impact Of Algal Blooms On Aquaculture And Effective Solution*. Organica Biotech [Online]. Available at <https://organicabiotech.com/impact-of-algal-blooms-on-aquaculture-and-effective-solution/#:~:text=Effects%20of%20Harmful%20Algal%20Bloom&text=The%20non%2Dtoxic%20producing%20species,eventually%20leads%20to%20fish%20kills> [Accessed on 18 March 2023].

Priyadarshani, I., Sahu, D. and Rath, B., 2012. Algae in aquaculture. *Int. J. Health Sci. Res*, 2, pp.108-114.

Norambuena, F., Hermon, K., Skrzypczyk, V., Emery, J. A., Sharon, Y., Beard, A., & Turchini, G. M., 2015. Algae in Fish Feed: Performances and Fatty Acid Metabolism in Juvenile Atlantic Salmon. *PLOS ONE*, 10(4), e0124042. <https://doi.org/10.1371/journal.pone.0124042>

Low Dissolved Oxygen in Water Causes, Impact on Aquatic Life – An Overview. (2009, February). *Minnesota Pollution Control Agency*, pp.1–2.

WebMD Editorial Contributors., 2023. *How to Change Fish Tank Water*. WebMD. <https://thefishsite.com/articles/how-to-achieve-good-water-quality-management-in-aquaculture>

How to achieve good water quality management in aquaculture., 2015. The Fish Site. <https://pets.webmd.com/how-to-change-fish-tank-water#:~:text=Changing%20your%20fish's%20water%20regularly,such%20as%20nitrate%20and%20phosphate.>

5.7 *Nitrates.*, 2012. Environmental Protection Agency. <https://archive.epa.gov/water/archive/web/html/vms57.html#:~:text=Together%20with%20phosphorus%2C%20nitrates%20in,%2C%20temperature%2C%20and%20other%20indicators.>

Li, T., Lu, J., Wu, J., Zhang, Z., & Chen, L., 2022. Predicting Aquaculture Water Quality Using Machine Learning Approaches. *Water*, 14(18), 2836. <https://doi.org/10.3390/w14182836>

Martinez, P., 2021. *What is Evolutionary Prototype?* Mockkitt Wondershare. <https://mockitt.wondershare.com/prototyping/evolutionary-prototyping.html>

F. J. Espinosa-Faller and G. E. Rendón-Rodríguez, “A ZigBee wireless sensor network for monitoring an aquaculture recirculating system,” *Journal of Applied Research and Technology*, vol. 10, no. 3, pp. 380–387, 2012.

M. Zhang, D. Li, L. Wang, D. Ma, and Q. Ding, “Design and development of water quality monitoring system based on wireless sensor network in aquaculture,” in *Computer and Computing Technologies in Agriculture IV*, D. Li, Y. Liu, and Y. Chen, Eds., pp. 629–641, Springer, 2011.

02DCE @02DCE, 2020, *Software Engineering Prototyping Model* [Online].

Abinaya, T., Ishwarya, J. and Maheswari, M., 2019. A Novel Methodology for Monitoring and Controlling of Water Quality in Aquaculture using Internet of Things (IoT). *2019 International Conference on Computer Communication and Informatics, ICCCI 2019*. 2019

Anon, *EZOTM RTD Temperature Circuit* [Online]. Available at: <https://atlas-scientific.com/embedded-solutions/ezo-rtd-temperature-circuit/> [Accessed: 14 September 2023a].

Anon, *ESP32* [Online]. Available at: <https://www.espressif.com/en/products/socs/esp32> [Accessed: 20 April 2023b].

Anon, *EZO™ Dissolved Oxygen Circuit* [Online]. Available at: <https://atlas-scientific.com/embedded-solutions/ezo-dissolved-oxygen-circuit/> [Accessed: 14 September 2023c].

Anon, *EZO™ pH Circuit* [Online]. Available at: <https://atlas-scientific.com/embedded-solutions/ezo-ph-circuit/#> [Accessed: 14 September 2023d].

Anon, 2012. *FISH TO 2030 Prospects for Fisheries and Aquaculture*, Washington.

Anon, 2015, *How to achieve good water quality management in aquaculture* [Online]. Available at: <https://thefishsite.com/articles/how-to-achieve-good-water-quality-management-in-aquaculture> [Accessed: 20 April 2023].

Anon, *Industrial Dissolved Oxygen Probe* [Online]. Available at: <https://atlas-scientific.com/probes/industrial-dissolved-oxygen-probe/> [Accessed: 14 September 2023e].

Anon, Industrial pH/ORP/Temp Probe. *Atlas Scientific*. Available at: <https://atlas-scientific.com/probes/industrial-ph-orp-temp-probe/> [Accessed: 14 September 2023f].

Anon, 2022a, *MQTT: The Standard for IoT Messaging* [Online]. Available at: <https://mqtt.org/> [Accessed: 20 April 2023].

Anon, *Scrum vs Waterfall vs Agile vs Lean vs Kanban* [Online]. Available at: <https://www.visual-paradigm.com/scrum/scrum-vs-waterfall-vs-agile-vs-lean-vs-kanban/> [Accessed: 21 April 2023g].

Anon, *SKU:SEN0237* [Online]. Available at: https://wiki.dfrobot.com/Gravity__Analog_Dissolved_Oxygen_Sensor_SKU_SEN0237#target_6 [Accessed: 20 April 2023h].

Anon, 2022b, *The Importance of Water Quality in Fish Farming* [Online]. Available at: <https://edinburghsensors.com/news-and-events/water-quality-in-fish-farming/> [Accessed: 11 April 2023].

Anon, 2019, *What Is Aquaculture and Why Do We Need It?* [Online]. Available at: <https://www.globalseafood.org/blog/what-is-aquaculture-why-do-we-need-it/> [Accessed: 11 April 2023].

Anon, 2023, *What Is LabVIEW?* [Online]. Available at: <https://www.ni.com/en-my/shop/labview.html> [Accessed: 20 April 2023].

Azlan Othman, N. et al., 2020. Automated water quality monitoring system development via LabVIEW for aquaculture industry (Tilapia) in Malaysia. *Indonesian Journal of Electrical Engineering and Computer Science*, 20(2), p.805.

Ciji, A. and Akhtar, M.S., 2020. Nitrite implications and its management strategies in aquaculture: a review. *Reviews in Aquaculture*, 12(2).

Das, B. and Jain, P.C., 2017. Real-time water quality monitoring system using Internet of Things. *2017 International Conference on Computer, Communications and Electronics, COMPTHELIX 2017*. 2017

Dupont, C., Cousin, P. and Dupont, S., 2018. IoT for aquaculture 4.0 smart and easy-to-deploy real-time water monitoring with IoT. *2018 Global Internet of Things Summit, GIoTS 2018*. 2018

Edward Lang, 2019, *The Importance of Water Quality Monitoring in Aquaculture* [Online]. Available at: <https://www.bellenviro.co.uk/blog/the-importance-of-water-quality-monitoring-in-aquaculture/> [Accessed: 11 April 2023].

Espinosa-Faller, F.J. and Rendón-Rodríguez, G.E., 2012. A zigbee wireless sensor network for monitoring an aquaculture recirculating system. *Journal of Applied Research and Technology*, 10(3).

FRESHWATER-AQUACULTURE, 2019, *Water Quality in Aquaculture* [Online]. Available at: <https://freshwater-aquaculture.extension.org/water-quality-in-aquaculture/> [Accessed: 20 April 2023].

Huang, J. et al., 2013. Development and test of aquacultural water quality monitoring system based on wireless sensor network. *Nongye Gongcheng Xuebao/Transactions of the Chinese Society of Agricultural Engineering*, 29(4).

Huang, J.F., Lee, J.M. and Sun, P.C., 2013. Prolonged culture period on production cost and factor input: A case from the Pacific Oyster, *Crassostrea gigas*, farming industry in Yunlin County, Taiwan. *Journal of the World Aquaculture Society*, 44(6).

Joseph, I. and Augustine, A., 2019. Marine biotechnology for food. *Genomics and Biotechnological Advances in Veterinary, Poultry, and Fisheries*, pp.271–283.

Lucy Towers, 2015, *Water quality: a priority for successful aquaculture* [Online]. Available at: <https://thefishsite.com/articles/water-quality-a-priority-for-successful-aquaculture> [Accessed: 11 April 2023].

Nicolai Berg Andersen, 2023, *What Is the Waterfall Methodology?* [Online].

NOAA, 2023, *What is aquaculture?* [Online]. Available at: <https://oceanservice.noaa.gov/facts/aquaculture.html> [Accessed: 11 April 2023].

OS-system, 2020, *Top 4 Software Development Methodologies: Comparison, Differences, Pros and Cons* [Online]. Available at: <https://os-system.com/contact/https://os-system.com/blog/top-software-development->

methodologies-comparison-differences-pros-and-cons/ [Accessed: 21 April 2023].

Romain Bourdon, *WAMPSEVER, a Windows web development environment*. [Online]. Available at: <https://www.wampserver.com/en/> [Accessed: 14 September 2023].

Su, X., Sutarlie, L. and Loh, X.J., 2020. Sensors, Biosensors, and Analytical Technologies for Aquaculture Water Quality. *Research*, 2020.

Taylor Otwell, *The PHP Framework for Web Artisans* [Online]. Available at: <https://laravel.com/> [Accessed: 14 September 2023].

Tolentino, L.K.S. et al., 2021. Development of an IoT-based Intensive Aquaculture Monitoring System with Automatic Water Correction. *International Journal of Computing and Digital Systems*, 10(1).

Tsai, K.L. et al., 2022. IoT based Smart Aquaculture System with Automatic Aerating and Water Quality Monitoring. *Journal of Internet Technology*, 23(1).
Will T, *Measuring and Interpreting System Usability Scale (SUS)* [Online]. Available at: <https://uiuxtrend.com/measuring-system-usability-scale-sus/> [Accessed: 14 September 2023].

Xiaowei Zhou, 2019. Brief overview of world aquaculture production An update with latest available 2017 global production data. *FAO Aquaculture Newsletter*, pp.6–8.

Yue, K. and Shen, Y., 2022. An overview of disruptive technologies for aquaculture. *Aquaculture and Fisheries*, 7(2), pp.111–120.

Zhang, M. et al., 2011. Design and development of water quality monitoring system based on wireless sensor network in aquaculture. *IFIP Advances in Information and Communication Technology*. 2011

APPENDICES

Appendix A: Template of User Satisfactory Survey

Participant No: 3					
Name: Poey Wei Jun					
Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	1	2	3	4	5
1. I think that I would like to use this system frequently.					
2. I found the system unnecessarily complex.					
3. I thought the system was easy to use.					
4. I think that I would need the support of a technical person to be able to use this system.					
5. I found the various functions in this system were well integrated.					
6. I thought there was too much inconsistency in this system.					
7. I would imagine that most people would learn to use this system very quickly.					

8. I found the system very cumbersome to use.					
9. I felt very confident using the system.					
10. I needed to learn a lot of things before I could get going with this system.					

1. What do you like best about the system?

2. What do you like least about the system?

3. Do you have any suggestions for improving the current system?

Appendix B: Usability Test Responses

Participant No: 1					
Name: Seow Ding Han					
Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	1	2	3	4	5
1. I think that I would like to use this system frequently.					✓
2. I found the system unnecessarily complex.		✓			
3. I thought the system was easy to use.				✓	
4. I think that I would need the support of a		✓			

technical person to be able to use this system.					
5. I found the various functions in this system were well integrated.				✓	
6. I thought there was too much inconsistency in this system.		✓			
7. I would imagine that most people would learn to use this system very quickly.					✓
8. I found the system very cumbersome to use.		✓			
9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

The system is easy to use after get familiar with the system.

2. What do you like least about the system?

Some function is incomplete, such as the recorded action is unable to be edit and delete.

3. Do you have any suggestions for improving the current system?

The recorded action should be able to be edit and delete.

Participant No: 2					
Name: Tang Chu Lin					
Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	1	2	3	4	5
1. I think that I would like to use this system frequently.				✓	
2. I found the system unnecessarily complex.		✓			
3. I thought the system was easy to use.				✓	
4. I think that I would need the support of a technical person to be able to use this system.				✓	
5. I found the various functions in this system were well integrated.					✓
6. I thought there was too much inconsistency in this system.				✓	
7. I would imagine that most people would learn to use this system very quickly.					✓
8. I found the system very cumbersome to use.	✓				

9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.	✓				

1. What do you like best about the system?

The dashboard design is nice, it provides various charts and analysis to display useful information.

2. What do you like least about the system?

The notification function is not designed very well, user cannot specifically set which channel notification to be open or closed. Besides, the notification is triggered using 2 buttons, yes or no, instead of a switch, so there is no indicator to let user know whether the notification is currently on or off.

3. Do you have any suggestions for improving the current system?

Make each channel to have an option to choose to open or close the notification for the channel. Besides, can replace the yes or no button with a switch, to make it look more user-friendly and informative.

Participant No: 3					
Name: Poey Wei Jun					
Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	1	2	3	4	5
1. I think that I would like to use this system frequently.				✓	
2. I found the system unnecessarily complex.	✓				
3. I thought the system was easy to use.			✓		
4. I think that I would need the support of a technical person to be able to use this system.				✓	
5. I found the various functions in this system were well integrated.				✓	
6. I thought there was too much inconsistency in this system.		✓			
7. I would imagine that most people would learn to use this system very quickly.					✓
8. I found the system very cumbersome to use.	✓				

9. I felt very confident using the system.				✓	
10. I needed to learn a lot of things before I could get going with this system.		✓			

1. What do you like best about the system?

The record action function has a calendar for user to choose the date to record activity. The calendar also have the marking of the dates to indicates activity carried out on that particular date. This calendar allows the user can have a overall and summarized view of all activities carried out in each month.

2. What do you like least about the system?

The activity created cannot be edit or deleted anymore, which means user cannot undo their operation when they make mistakes when adding new events.

3. Do you have any suggestions for improving the current system?

Make the calandar also have the ability to delete and edit the recorded activities.