

**APPLICATION DEVELOPMENT OF VCARE CIRCLE FOR UTARIAN**

**BY**

**TAN ZHI YUAN**

**A REPORT**

**SUBMITTED TO**

**Universiti Tunku Abdul Rahman**

**in partial fulfillment of the requirements**

**for the degree of**

**BACHELOR OF COMPUTER SCIENCE (HONOURS)**

**Faculty of Information and Communication Technology**

**(Kampar Campus)**

**JANUARY 2023**

## REPORT STATUS DECLARATION FORM

**Title:** APPLICATION DEVELOPMENT OF VCARE CIRCLE FOR UTARIAN

\_\_\_\_\_  
\_\_\_\_\_

**Academic Session:** JANUARY 2023

I TAN ZHI YUAN  
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in  
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



\_\_\_\_\_  
(Author's signature)



\_\_\_\_\_  
(Supervisor's signature)

**Address:**

107, Taman Desa Bintang 2,

32000 Sitiawan, Perak

\_\_\_\_\_

Dr Tan Joi San

Supervisor's name

**Date:** 21/4/2023

**Date:** 21/4/2023

<b>Universiti Tunku Abdul Rahman</b>			
Form Title : <b>Sample of Submission Sheet for FYP/Dissertation/Thesis</b>			
Form Number: <b>FM-IAD-004</b>	Rev No.: <b>0</b>	Effective Date: <b>21 JUNE 2011</b>	Page No.: <b>1 of 1</b>

**FACULTY/INSTITUTE\* OF INFORMATION AND COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)**

**UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 21/4/2023

**SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS**

It is hereby certified that Tan Zhi Yuan (ID No: 19ACB03046) has completed this final year project/ dissertation/ thesis\* entitled “Application Development of VCare Circle For Utarian” under the supervision of Dr Tan Joi San (Supervisor) from the Department of Computer Science, Faculty/Institute\* of Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis\* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,




(TAN ZHI YUAN)

\*Delete whichever not applicable

## DECLARATION OF ORIGINALITY

I declare that this report entitled “**APPLICATION DEVELOPMENT OF VCARE CIRCLE**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  \_\_\_\_\_

Name : Tan Zhi Yuan

Date : 21/4/2023

## **ACKNOWLEDGEMENTS**

I would like to express my thanks to my supervisor, Dr. Tan Joi San, for giving me this project idea to do as my Final Year Project (FYP). This is the first big project in my road of IT degree studies, which can be my starting point to learn how to deal with the development and management of a project before going to work in IT careers. From here, I am able to have the opportunity to work on mobile application development, and also to study on Firebase and Flutter, which these skills and knowledge are to be useful in upcoming future for my future career.

I would like to also express my thanks to my academic advisor, Dr. Ng Hui Fuang, who have been guiding me throughout the years in my IT degree, while also giving me advice on how to do proper time management to finish my work in time. Lastly, to my parents, I would like to thank them for always be there for me when I need them in my hard times, and always support me in my studies, encouraging me to work hard and continue to finish my IT course.

## **ABSTRACT**

For this project, it is done for the purpose of helping UTAR university students in UTAR Kampar. Asking for help is a good method to solve the issues faced by anyone as it is time efficient and effective, while also promoting one's growth to adjust in different environments easily. However, not all people are capable of doing so as when they face difficulty, they are in a state of confusion on who to ask and how to approach someone for help. For example, when a student is facing difficulty in the university, there is no people around for them to ask for help as they don't know who can help them to solve this problem. Problems such as unable to locate particular buildings are mostly faced by new students when they are trying to get familiar with the new university life and environment. Thus, a mobile application called VCare Circle as an online assist platform is developed to allow people to ask for help when they need it. For example, when a university student does not know the direction of heritage hall of UTAR, he or she can utilize the project mobile application by login through the registered account to be automatically assigned to a random registered volunteer or helper. The project mobile application allows him or her to get contact with the anonymous helper in order to get advice to determine the exact location of the hall. For this project, the fields of study involved application development, mobile application, and data analysis. In the development of the project, the system methodology used to develop this project is the Rapid Application Development (RAD) methodology to ensure the complexity of framework design and smoothness in the development process. The techniques and equipment involved are the use of Android Studio, Visual Studio Code, Firebase, a mobile device and laptop, Dart programming language, and Google Flutter. Self-manual stimulation testing is done to conduct testing on the project mobile application based on its required functional and non-functional requirements. In conclusion, this system can help students to get familiar with the surroundings of UTAR Kampar, while being able to ask for help anytime and anywhere when they are in need.

# TABLE OF CONTENTS

<b>TITLE PAGE</b>	<b>i</b>
<b>REPORT STATUS DECLARATION FORM</b>	<b>ii</b>
<b>FYP THESIS SUBMISSION FORM</b>	<b>iii</b>
<b>DECLARATION OF ORIGINALITY</b>	<b>iv</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF TABLES</b>	<b>xvii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xix</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Overview	1
1.2 Problem Statement and Motivation	2
1.3 Project Objectives	4
1.4 Project Scope and Direction	5
1.5 Proposed Approach/Study	9
1.6 Highlights of What Have Been Achieved	11
1.7 Report Organization	12
1.8 Summary	12
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>14</b>
2.1 Overview	14
2.2 Previous Work on University Community	14
2.2.1 Hi-Hive Community	14
2.2.2 Uni Course Chat	18
2.2.3 Bronx CC	21
2.2.4 Differ Chat	25
2.2.5 Lewis University	29
2.3 Summary	33

<b>CHAPTER 3 SYSTEM DESIGN</b>	<b>36</b>
3.1 Overview	36
3.2 Design Specifications	36
3.2.1 Use Case Diagram	36
3.2.2 Use Case Descriptions	38
3.3 Project Development	72
3.3.1 Firebase Setup and Connect to Project	72
3.3.2 Register Account Function Development	74
3.3.3 Login Account Function Development	78
3.3.4 Profile Module Development	81
3.3.5 Online Communication Function Development	90
3.3.6 QR Identification Function Development	98
3.3.7 Helper Finder Function Development	100
3.3.8 Offline Module Development	102
3.3.9 Administrator Module Development	103
3.3.9.1 Administrator Control Development	104
3.3.9.2 Application Data Analysis Development	111
3.4 Summary	125
<b>CHAPTER 4 METHODOLOGY AND TOOLS USED</b>	<b>126</b>
4.1 Overview	126
4.2 Methodology	126
4.2.1 Requirement Planning Phase	127
4.2.2 Design Phase	127
4.2.3 Construction Phase	128
4.2.4 Cutover Phase	128
4.3 Development Tools Used	129
4.3.1 Hardware Requirements	129
4.3.2 Software Requirements	130
4.4 Project Timeline	131
4.4.1 Overview	131
4.4.2 Gantt Chart	133
4.5 Summary	135



<b>CHAPTER 5 SYSTEM IMPLEMENTATION</b>	<b>135</b>
5.1 Overview	135
5.2 Login Page	137
5.3 Sign Up Page	137
5.4 Reset Password Page	138
5.5 Verify Email Page	139
5.6 Verify Admin Password Page	140
5.7 Inactive Status Warning Page	141
5.8 Home Page	142
5.9 Profile Page	145
5.10 Settings Page	148
5.11 Offline Page	149
5.12 Online Chat Page	150
5.13 Online Voice Call Page	152
5.14 Users and Sessions Management Pages	155
5.15 Application Data Analysis Pages	156
5.16 Summary	
<b>CHAPTER 6 CONCLUSION</b>	<b>157</b>
6.1 Project Review, Discussion and Conclusion	157
6.2 Novelties and Contributions	158
6.3 Future Work	159
<b>REFERENCES</b>	<b>161</b>
<b>APPENDIX A</b>	
<b>FINAL YEAR PROJECT WEEKLY LOGS</b>	A-1
<b>POSTER</b>	A-6
<b>PLAGARISM CHECK RESULT</b>	A-7
<b>FYP2 CHECKLIST</b>	A-9

## LIST OF FIGURES

<b>Figure Number</b>	<b>Title</b>	<b>Page</b>
Figure 1.5.1	Flowchart of VCare Circle Mobile Application	9
Figure 2.2.1.1	Main Page of Hi-Hive	15
Figure 2.2.1.2	Hi-Hive Menu Options	16
Figure 2.2.1.3	Hi-Hive Profile Edit Page	16
Figure 2.2.1.4	Hive Store Options in Hi-Hive	17
Figure 2.2.2.1	Hub Page of Uni Course Chat	19
Figure 2.2.2.2	Discover Page of Uni Course Chat	19
Figure 2.2.2.3	Group Chat Page of Uni Course Chat	20
Figure 2.2.2.4	Chat Function in Uni Course Chat	20
Figure 2.2.3.1	Main Visitor Page of Bronx CC	23
Figure 2.2.3.2	Type of User Login of Bronx CC	23
Figure 2.2.3.3	News Page of Bronx CC	24
Figure 2.2.3.4	Social Media Platform of University of Bronx CC	24
Figure 2.2.4.1	Main Page of Differ.Chat	27
Figure 2.2.4.2	Community List of Differ.Chat	27
Figure 2.2.4.3	Chat Page of Differ.Chat	28
Figure 2.2.5.1	Main Page of LewisU	30
Figure 2.2.5.2	Event List of LewisU	31
Figure 2.2.5.3	Group and Clubs List of LewisU	31
Figure 2.2.5.4	Club Details Page of LewisU	32
Figure 3.2.1.1	Use Case Diagram for VCare Circle Mobile Application Part A	36
Figure 3.2.1.2	Use Case Diagram for VCare Circle Mobile Application Part B	37
Figure 3.3.1.1	Registering of Project in Firebase Console	73
Figure 3.3.1.2	Firebase Dependencies in App Level Build Gradle File	73
Figure 3.3.1.3	Initialize Firebase Services in main.dart	74
Figure 3.3.2.1	Example of TextEditingController Validator for Username	75
Figure 3.3.2.2	TextEdtitingController Validator for User Phone Number	75

Figure 3.3.2.3	Validation Check for User Personal Information in sign_up_page1.dart	76
Figure 3.3.2.4	signUp Function in sign_up_page2.dart	76
Figure 3.3.2.5	uploadUserDetails Function in sign_up_page2.dart	77
Figure 3.3.2.6	checkVerification Function in verify_email_page.dart	77
Figure 3.3.2.7	sendVerification Function in verify_email_page.dart	77
Figure 3.3.3.1	loginApp Function in login_page.dart	79
Figure 3.3.3.2	saveTokenForMessaging Function in push_notification_model.dart	79
Figure 3.3.3.3	sendNotificationMessage Function in push_notification_model.dart	80
Figure 3.3.3.4	searchUserStatus Function in checkUserStatus.dart	80
Figure 3.3.3.5	Verify Admin Password Code Snippet in admin_password_veirfy_screen.dart	81
Figure 3.3.4.1	setUserDetails Function in profile_page.dart	83
Figure 3.3.4.2	getUserInfo Function in user.dart	83
Figure 3.3.4.3	editComplete Function in edit_profile_page.dart	84
Figure 3.3.4.4	Toggle Switch Function in helper_details.dart To Toggle Helper Status	84
Figure 3.3.4.5	showConfirmOnOffHelperStatus Function to Display Dialog	85
Figure 3.3.4.6	updateHelperOnOffStatus Function in user.dart	85
Figure 3.3.4.7	removeDutyPeriodHelper Function in user.dart	85
Figure 3.3.4.8	Code Snippet To Set Helper Duty Period in helper_details.dart	86
Figure 3.3.4.9	updateUserInformation in user.dart	86
Figure 3.3.4.10	changeAccountType Function in user.dart	86
Figure 3.3.4.11	setRequiredInformation Function in admin_nav.dart	87
Figure 3.3.4.12	Code Snippet of Function to Edit Admin Profile in edit_admin_profile_info.dart Part A	87
Figure 3.3.4.13	Code Snippet of Function to Edit Admin Profile in edit_admin_profile_info.dart Part B	87

Figure 3.3.4.14	Code Snippet of Function to Edit Admin Profile in edit_admin_profile_info.dart Part C	88
Figure 3.3.4.15	updateAdminInfo Function in admin.dart	88
Figure 3.3.4.16	checkEmailInServer Function in admin.dart	88
Figure 3.3.4.17	Code Snippet of Function to Update Admin Email in edit_admin_profile_info.dart	89
Figure 3.3.4.18	updateAdminUser in admin.dart	89
Figure 3.3.4.19	sendUpdateEmail Function in edit_admin_profile_info.dart	89
Figure 3.3.5.1	Project Added in Agora Console	92
Figure 3.3.5.2	initializeCallEngine Function in in_call_screen.dart	93
Figure 3.3.5.3	joinCall Function and leaveCall function in in_call_screen.dart	93
Figure 3.3.5.4	sendCallMessageInChatRoom in chat_page.dart	94
Figure 3.3.5.5	buildChatRoom Function in chatroom_model.dart	94
Figure 3.3.5.6	Extended Functions for buildChatRoom Function in chatroom_model.dart	94
Figure 3.3.5.7	Stream Builder Code Snippet in chat_page.dart and helper_chat_page.dart	95
Figure 3.3.5.8	sendMessage Function in chat_page.dart and help_chat_page.dart	95
Figure 3.3.5.9	sendImage Function in chat_page.dart and help_chat_page.dart	96
Figure 3.3.5.10	setMessageToChatRoom Extended Function in sendMessage and sendImage Function	96
Figure 3.3.5.11	endChatRoomSession Function in chatroom_model.dart	96
Figure 3.3.5.12	setHistoryandClearChatRoom Function in history_model.dart	97
Figure 3.3.5.13	calculateDurationBetweenTime Function in history_model.dart	97
Figure 3.3.5.14	createHistory Function in history_model.dart	97
Figure 3.3.5.15	setRatingForChatRoom in rating_model.dart	98
Figure 3.3.6.1	createHelperQr Function in user.dart	99
Figure 3.3.6.2	retrieveHelperQr Function in user.dart	99

Figure 3.3.6.3	QR Scanner Main Code Snippet in qr_scanner_user.dart	99
Figure 3.3.6.4	QR Generator Main Code Snippet in qr_generator_helper.dart	99
Figure 3.3.7.1	getHelper Function in user.dart	101
Figure 3.3.7.2	assignHelper Function in home_page.dart	101
Figure 3.3.8.1	checkInternetConnection Function in main.dart and offline_page.dart	103
Figure 3.3.8.2	URL Launcher Function in offline_page.dart	103
Figure 3.3.9.1.1	Stream Builder to Fetch List Of Users in list_of_users.dart	106
Figure 3.3.9.1.2	Stream Builder to Fetch List Of Helpers in list_of_helpers.dart	106
Figure 3.3.9.1.3	Stream Builder to Fetch List of Inactive Users in list_of_banned_users.dart	107
Figure 3.3.9.1.4	Stream Used to Fetch List of Inactive Users in list_of_banned_users.dart	107
Figure 3.3.9.1.5	getInfo Function in admin_view_user_page.dart	107
Figure 3.3.9.1.6	getUserInfo Function in user.dart	107
Figure 3.3.9.1.7	moveUserToInactive Function in inactive_user.dart	108
Figure 3.3.9.1.8	setUserToActive Function in inactive_user.dart	108
Figure 3.3.9.1.9	Code Snippet to Direct Call Account User in view_user_info_body.dart and inactive_user_info_page.dart	108
Figure 3.3.9.1.10	sendUpdateEmail Function in send_email_in_background.dart	109
Figure 3.3.9.1.11	getUserRequestHistory Function in admin.dart	109
Figure 3.3.9.1.12	getHelperHelpHistory Function in admin.dart	109
Figure 3.3.9.1.13	Stream Builder for Retrieving Chat History in admin_view_chatroom_history.dart	110
Figure 3.3.9.1.14	Stream Set For Retrieving Chat History in admin_view_chatroom_history.dart	110
Figure 3.3.9.1.15	deleteHistoryDocument Function in history_model.dart	110
Figure 3.3.9.2.1	getNewUserByMonth Function in app_data_controls.dart	114
Figure 3.3.9.2.2	getNewUserByYear Function in app_data_controls.dart	114

Figure 3.3.9.2.3	Code Snippet Example To Input Data to Chart in app_user_chart.dart	114
Figure 3.3.9.2.4	Code Snippet Example To Find Maximum Y-Axis Value in app_user_chart.dart	114
Figure 3.3.9.2.5	returnHighestNumber in data_model.dart	115
Figure 3.3.9.2.6	Code Snippet Example To Find Interval Value in app_user_chart.dart	115
Figure 3.3.9.2.7	Code Snippet For Bar Chart in app_user_chart.dart	115
Figure 3.3.9.2.8	getNumberOfUsesByMonth Function in app_data_controls.dart	116
Figure 3.3.9.2.9	getNumberOfUsesByYear Function in app_data_controls.dart	116
Figure 3.3.9.2.10	Code Snippet Example To Input Data to Chart in app_usage_chart.dart	116
Figure 3.3.9.2.11	Code Snippet Example To Find Maximum Y-Axis Value in app_usage_chart.dart	116
Figure 3.3.9.2.12	Code Snippet Example To Find Interval Value in app_usage_chart.dart	117
Figure 3.3.9.2.13	Code Snippet For Line Chart in app_usage_chart.dart	117
Figure 3.3.9.2.14	getOverallUseTimeByMonth in app_data_controls.dart	118
Figure 3.3.9.2.15	getOverallUseTimeByYear in app_data_controls.dart	118
Figure 3.3.9.2.16	Code Snippet Example To Input Data To Chart in app_overall_session_duration.dart	118
Figure 3.3.9.2.17	Code Snippet Example To Find Maximum Y-Axis Value in app_overall_session_duration.dart	119
Figure 3.3.9.2.18	Code Snippet Example To Find Interval Value in app_overall_session_duration.dart	119
Figure 3.3.9.2.19	Code Snippet For Line Chart in app_overall_session_duration.dart	120
Figure 3.3.9.2.20	getListOfHighestSession Function in app_data_controls.dart	121
Figure 3.3.9.2.21	getLongestDuration Extension Function in app_data_controls.dart	121

Figure 3.3.9.2.22	getListOfShortestSessions Function in app_data_controls.dart	122
Figure 3.3.9.2.23	getShortestSessionDuration Extension Function in app_data_controls.dart	122
Figure 3.3.9.2.24	getListOfHighestSessionsInYear Function in app_data_controls.dart	123
Figure 3.3.9.2.25	getHighestSessionDurationInYear Function in app_data_controls.dart	123
Figure 3.3.9.2.26	getListOfShortestSessionsInYear Function in app_data_controls.dart	124
Figure 3.3.9.2.27	getShortestSessionDurationInYear Extension Function in app_data_controls.dart	124
Figure 4.2.1	Rapid Application Development Methodology	127
Figure 5.1.1	Admin Navigation Drawer	136
Figure 5.1.2	User and Helper Navigation Drawer	136
Figure 5.1.3	Welcome Page	136
Figure 5.2.1	Login Page	137
Figure 5.3.1	Sign Up Page One	138
Figure 5.3.2	Sign Up Page Two	138
Figure 5.4.1	Reset Password Page	139
Figure 5.5.1	Verify Email Page	140
Figure 5.6.1	Verify Admin Password Page	141
Figure 5.7.1	Inactive Status Warning Page	142
Figure 5.8.1	Admin Home Page	143
Figure 5.8.2	User Home Page	144
Figure 5.8.3	Helper Home Page	144
Figure 5.8.4	User Home Page in Session	144
Figure 5.8.5	Helper Home Page in Session	144
Figure 5.9.1	User Profile Page	145
Figure 5.9.2	Helper Profile Page	145
Figure 5.9.3	User Account Details Page	146
Figure 5.9.4	Helper Account Details Page	146
Figure 5.9.5	Helper Details Page	146

Figure 5.9.6	User Edit Profile Page	147
Figure 5.9.7	Helper Edit Profile Page	147
Figure 5.9.8	Admin Profile Page	147
Figure 5.9.9	Admin Edit Profile Page	147
Figure 5.10.1	Settings Page	148
Figure 5.11.1	Offline Page	149
Figure 5.12.1	User Chat Page	150
Figure 5.12.2	Helper Chat Page	150
Figure 5.13.1	User Incoming Call Screen	151
Figure 5.13.2	Helper Incoming Call Screen	151
Figure 5.13.3	User Call Screen	151
Figure 5.13.4	Helper Call Screen	151
Figure 5.14.1	List of Users Page	153
Figure 5.14.2	List of Helpers Page	153
Figure 5.14.3	List of Inactive Users Page	153
Figure 5.14.4	User Account Details Page	153
Figure 5.14.5	Helper Account Details Page	153
Figure 5.14.6	Inactive User Account Details Page	153
Figure 5.14.7	List of User Request History Sessions	154
Figure 5.14.8	List of Helper Duty History Sessions	154
Figure 5.14.9	Session Chat History Page	154
Figure 5.15.1	Bar Chart for Number of New Registered Users	155
Figure 5.15.2	Line Chart for Application Usage	155
Figure 5.15.3	Line Chart for Overall Session Duration	155
Figure 5.15.4	List of Result Sessions for Longest Duration	156
Figure 5.15.5	List of Result Sessions for Shortest Duration	156



## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 1.4.1	Project Scope Function Modules and Description	5
Table 2.3.1	Comparison of the Strengths and Weaknesses of the 5 Existing Application	35
Table 3.2.2.1	Login Use Case Description	38
Table 3.2.2.2	Sign Up Use Case Description	39
Table 3.2.2.3	Verify Admin Password Use Case Description	40
Table 3.2.2.4	Reset Admin Password Use Case Description	40
Table 3.2.2.5	Verify Email Use Case Description	42
Table 3.2.2.6	Reset Password Use Case Description	42
Table 3.2.2.7	View Profile Details Use Case Description	43
Table 3.2.2.8	Edit Admin Email Use Case Description	44
Table 3.2.2.9	View User or Helper List Use Case Description	45
Table 3.2.2.10	View User or Helper Account Use Case Description	46
Table 3.2.2.11	Edit User or Helper Account Use Case Description	47
Table 3.2.2.12	Toggle Account Active Status Use Case Description	48
Table 3.2.2.13	Contact User Use Case Description	49
Table 3.2.2.14	View Inactive User List Use Case Description	51
Table 3.2.2.15	View Inactive User Information Use Case Description	51
Table 3.2.2.16	View List of History Records Use Case Description	52
Table 3.2.2.17	View App Data Analysis Use Case Description	53
Table 3.2.2.18	View Number of New Users Use Case Description	54
Table 3.2.2.19	View Overall Session Duration Use Case Description	56
Table 3.2.2.20	View App Usage Count Description	57
Table 3.2.2.21	Search Email Account Description	58
Table 3.2.2.22	Request Help Use Case Description	59
Table 3.2.2.23	End Session Use Case Description	60
Table 3.2.2.24	Rate Session Use Case Description	60
Table 3.2.2.25	Contact UTAR Use Case Description	61
Table 3.2.2.26	Online Chat Communication Use Case Description	62

Table 3.2.2.27	Online Call Communication Use Case Description	63
Table 3.2.2.28	Change Account Type Use Case Description	64
Table 3.2.2.29	Modify Helper Status Use Case Description	64
Table 3.2.2.30	QR Scanning Use Case Description	65
Table 3.2.2.31	QR Code Generator Use Case Description	66
Table 3.2.2.32	Edit Profile Use Case Description	66
Table 3.2.2.33	Delete Account Use Case Description	67
Table 3.2.2.34	Create Session Use Case Description	68
Table 3.2.2.35	SMS Communication Use Case Description	69
Table 3.2.2.36	Phone Call Communication Use Case Description	69
Table 3.2.2.37	Email Communication Use Case Description	70
Table 3.2.2.38	Toggle Helper Status Use Case Description	71
Table 3.2.2.39	Set Helper Duty Period Use Case Description	72
Table 4.3.1.1	Specifications of Laptop for Development of Project	129
Table 4.3.1.2	Specifications of Mobile Device for Testing of Project	129
Table 4.3.2.1	Software Requirements for The Project	130
Table 4.4.2.1	Gantt Chart of The VCare Circle Mobile Application Project Timeline	133

## LIST OF ABBREVIATIONS

<i>Covid-19</i>	Coronavirus Disease
<i>Email</i>	Electronic Mail
<i>FYP</i>	Final Year Project
<i>ID</i>	Identification
<i>IDE</i>	Integrated Development Environment
<i>JSON</i>	JavaScript Object Notation
<i>QR</i>	Quick Response
<i>RAD</i>	Rapid Application Development
<i>SDK</i>	Software Development Kit
<i>SDLC</i>	System Development Life Cycle
<i>SHA</i>	Secure Hash Algorithm
<i>SMS</i>	Short Message Service
<i>SMTP</i>	Simple Mail Transfer Protocol
<i>UI</i>	User Interface
<i>URL</i>	Uniform Resource Locator
<i>UTAR</i>	Universiti Tunku Abdul Rahman

# Chapter 1

## Introduction

### 1.1 Overview

Social skills are the fundamental skills learned by university students to hold conversation with one another and to build relationships with the people around, whether personal or professional. This set of skills serves as a bridge to connect between people for more understanding of each other, and also serve the purpose of getting some benefit while building relationships, with the main benefit of being able to get help when one has faced difficulty in some situation and required some guidance [1].

Asking for help is what people should do when they are facing difficulties. This action also helps in building relationships with others as when one asks for help, they can build connections with the helper, no matter if they are of the same age or professionals [2]. For university students, asking for help not only can help with getting guidance when facing difficulty, but also to get familiarize with the university surroundings. For example, there are always new intake students entering the university, and they may face issues getting themselves familiar with the surroundings, such as getting lost in the way to their faculties, what to take note of as a new intake student of UTAR, and the procedure to submit documents to the faculty office. Thus, asking for help is also a very useful way for new intake students to adjust the change of environment for when they know what and who to ask when they are in need [3].

However, not many people are able to do it, especially university students nowadays who are living in modern society. For example, based on the survey conducted in mid-South in 2017, there are some factors that have become the barriers for university students to ask for help from their lecturers or professors [4]. The factors are shyness and being embarrassed to ask for help, and the issue of university students not knowing the specific issue they want to consult. Due to being unable to get help, they may face issues not only in academic matters, but also they may feel stressed out and end up being tangled with the problems at hand and unable to solve them. New intake students also face the issue of being unable to get familiar with the new environment with no

help for them and thus takes more time than usual to just get familiar with their new university life.

Therefore, the development of the VCare mobile application for university students to ask for help is done to improve the situation by giving the students an online communication platform to ask for help and ask for the information they required. The system consists of the function to allow UTAR students to ask for help by anonymous people who have registered as helpers in the system. For example, when the user is in need of help but doesn't know who to approach, they are able to contact a helper in the application in order to ask for help. Apart from that, if the user and helper agree to meet physically, QR identity verification of the helper is provided for identity recognition to avoid any safety issue. As for the registration as the user of the application, the user is able to either choose to register as a user or as a helper, as the helpers have an additional setting of getting requests for help from users. For helpers, they are also able to set their time of availability of help and toggle the helper status in the application.

### **1.2 Problem Statement and Motivation**

Asking for help undeniably gives university students more opportunity to grow and mainly to solve their issues at hand. However, communication and social skills are the crucial key in asking for help and unfortunately not all university students possess strong social skills and handle physical interaction with people, especially the new intake students. This is because they faced some barriers when wanting to ask for help. The problems are unable to specify an issue properly, feeling of anxiety and nervousness, and assuming asking for help as an act of shame. Based on these problems, the project tries to come out with solutions to overcome this problem by contributing a new way of asking for help.

#### **i. Unable to Specify an Issue Properly**

Although university students may know the issue or hardship they have faced, they may have problems in specifying the problem due to the fact of not knowing some specific terms or way of explaining what they are facing. For example, the student may know the documents need to be submitted to one of the people in charge in the office, however they may not know how to ask for the location of the office to submit the document and

who to mention when they are asking for help. Especially new intake students, as they are not familiar with the new environment, more difficulty is faced by them as they mostly won't know anyone around and what to do as a new UTAR student to fit into this new environment. Thus, the limitation of being unable to express the help they required also has become one of the factors university students slowly become less desired in asking for help.

### **ii. Feeling of Anxiety and Nervousness When Asking for Help**

There are times when university students are unable to voice out their thoughts, especially when they are in a state of confusion. For example, there are times when university students are confused about things such as the exact venue for the document submission or how to apply for vehicle stickers, and there are no familiar people around them to help. The feeling of anxiety and nervousness accumulated caused them to not know how to approach someone on the spot. This issue is further elevated due to the lockdown of Covid-19 pandemic, which has greatly decreased university students' communication skills. For example, based on the survey conducted in American University of Madaba, it has shown that most university students have agreed that during this Covid-19 pandemic they have negatively affected their communication due to the feeling of isolation in attending online class [5]. From here, this factor also has become one of the barriers faced by university students in asking for help due to the feeling of anxiety and nervousness upon trying to ask for help.

### **iii. Assume Asking for Help as Act of Shame**

In modern society, people start to bring more attention to self-image and personal status in a social circle, which they started to focus more on how people think about themselves [6]. With this thought, they think that asking for help gives people the image of them being weak and helpless. This self-assumed thought confuses their decision to ask for help from people around them. For example, when a student tries to ask for help from people around in the university on issues that they think are minor problems, they may think this may result in people around in university looking down on them when remembering or seeing them in the university. This ends up with them cancelling the thought of asking for help and keeping the issues to themselves, which leaves them to have the issues hard to overcome or cannot be solved. Therefore, this is also a barrier

for them to reach out to ask for help due to the hesitation of considering their self-images and assuming asking for help as an act of shame.

The motivation of developing this project is to allow an easier way of asking help for university students, which is by creating an online mobile platform to let people from either professionals or seniors from the university be able to give help to the university students when they are in need of guidance or help. To improvise the issue of students having issues talking to strangers, the identity of the helpers is hidden from the user who asked for help in the mobile application so that the user does not have to worry on the status or position of person he or she is asking and treat the helper as in the same position as them. For example, when a university student asks for help in the mobile application, random helper without knowing his or her identity is assigned to the student. Therefore, the motivation of developing this project not only allow university students to be able to ask for help efficiently but also allows university students to have more understanding in the university surroundings and help other people as helpers in the future when other university students face similar issues.

### **1.3 Project Objectives**

The main objective of the project is to develop a mobile application to improve the effectiveness and availability in giving help to university students when they are in need, whether academic issues or other minor issues. This next purpose is to allow new intake students to quickly adapt to the new environment, while improving the relationship among the university student community.

#### **i. To Improve the Effectiveness of University Students Asking for Help**

This mobile application allows university students to immediately ask for help to explain the situation they faced through having an online chat or online voice call conversation with the helpers, so that the help or advice they received are more effective and accurate as the helpers know exactly what the difficulty the requester is facing when they can take their time explaining through online chat or voice call conversation. On the other hand, the mobile application also has a QR code generating and scanning module for users in case they would like to have a physical meeting with their assigned

helper. As the helper is assigned automatically by the application, this also help improve time efficiency as the university students do not need to try approach to someone for help or wait for help.

## ii. To Reduce Future Disturbance and Worry for the Helpers and Users

The project aims to reduce the issue of identity exposed for the helper and the prevention of overthinking for users when asking for help. As mentioned in the application the helper's identity is hidden to reduce further disturbance between helpers and users as they only wanted to do an act of kindness. Thus, by having the helpers' identity hidden, they do not have to worry the user comes back for them or start to look for them in the university, while not offending the users to reject their approach for building unwanted relationships as their identity are kept secret from the user in the application. On the other hand, for users of the application, this also benefits them to not required to worry on their self-images when asking for help as asking for help is done only among the helpers with hidden identity and the users, which they can freely voice out their issues as they do not have to worry other people noticing them asking for help and think that other people look down on them, while also do not know who they have asked for help and end up having an awkward encounter in the university.

## 1.4 Project Scope and Direction

As the main purpose of this project is to enhance the communication of university students with people around asking for help, the project scope covers the functional modules that can help solve the problems stated in Chapter 1.2. In the project mobile application, features such as login and sign up, account profile, online chat and voice call communication, QR code generator and scanner function, helper finder function, and also administrator controls with application data analysis are added into the project.

Table 1.4.1 Project Scope Function Modules and Description

Functionality Modules	Description
<b>Login and Sign-Up Module</b>	In the mobile application, a sign-up function is provided to allow users to create account and set the account type as either "User" or "Helper". A login page



	<p>is created to allow users to login into the application with the account created. The purpose of having this module is to ensure different privileges and functionality of the application assigned to users based on the account type such as users, helpers, or administrator before proceeding to use the application. Before login into the application, the system also ensures that the user's email is verified, account is in active status, and to check if the account belongs to the administrator account or not.</p>
<b>Profile Module</b>	<p>The users can view their profile page with different information based on whether the user is logged in as a user, helper, or administrator. In the profile page, they can view and update the information in their profile if required and to edit information saved in the account such as profile picture, mobile phone number and status, and for helpers the time of availability to help and manually set on or off their helper status. For administrator, he or she can also choose to reset the administrator password and update the administrator email account.</p>
<b>Online Communication Module</b>	<p>When a user is asking for help, the user is connected to an online chatroom to ask for help from the helper assigned to the user. Here as the application can automatically assign helpers to the users, the user doesn't have to save any contacts in the application to use the function. Not only online messaging is provided in the project mobile application, but also the online calling function is also equipped for the purpose of allowing real-time interaction between the user and helper. Besides, when requesting help from helpers, the helper's and user's identity are hidden for privacy purpose.</p>

<b>QR Identification Module</b>	When there is an active online session, a random QR code is generated for the helper to show to the user for QR scanning in case if they have personally agreed to have a physical meeting for identification purpose. This is for the purpose of allowing users to identify their helpers assigned to them securely. This QR code is then discarded after use to ensure the helper or user cannot be tracked after the session in the future.
<b>Helper Finder Module</b>	For the ease of users seeking help, the mobile application automatically assigns helpers for users whenever the user chooses to ask for help. When the user requests help from the mobile application, the mobile application then tries to find available helpers to be assigned to help the user so that the user does not require to initially have contacts in their mobile application to ask for help and this also make asking for help much easier as only a short session of interaction with helper is required to ask for the help they need.
<b>Offline Module</b>	An offline module is added for the mobile application, which when the application user is not connected to the internet, they can choose to contact to UTAR help centre through their mobile phone, which is either through phone call or SMS messaging. This is to ensure the users can still ask for help in situation where they have with bad internet connection or no available online connections.
<b>Administrator Module</b>	The administrator module is for the purpose of adding administrator controls to the mobile application by allowing the administrator to log into the mobile application and have full control on the backend data and services such as to delete history record or to check

	<p>user's account information. Besides, data analysis on the project mobile application data such as the application usage by users and helpers, overall session duration of users and helpers, and the number of new users registered in the application, are showed in the administrator home page for the purpose of allowing admin to have a quick overall view on the project mobile application usage and analyse on the application user's uses in the project mobile application.</p>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1.5 Proposed Approach/ Study

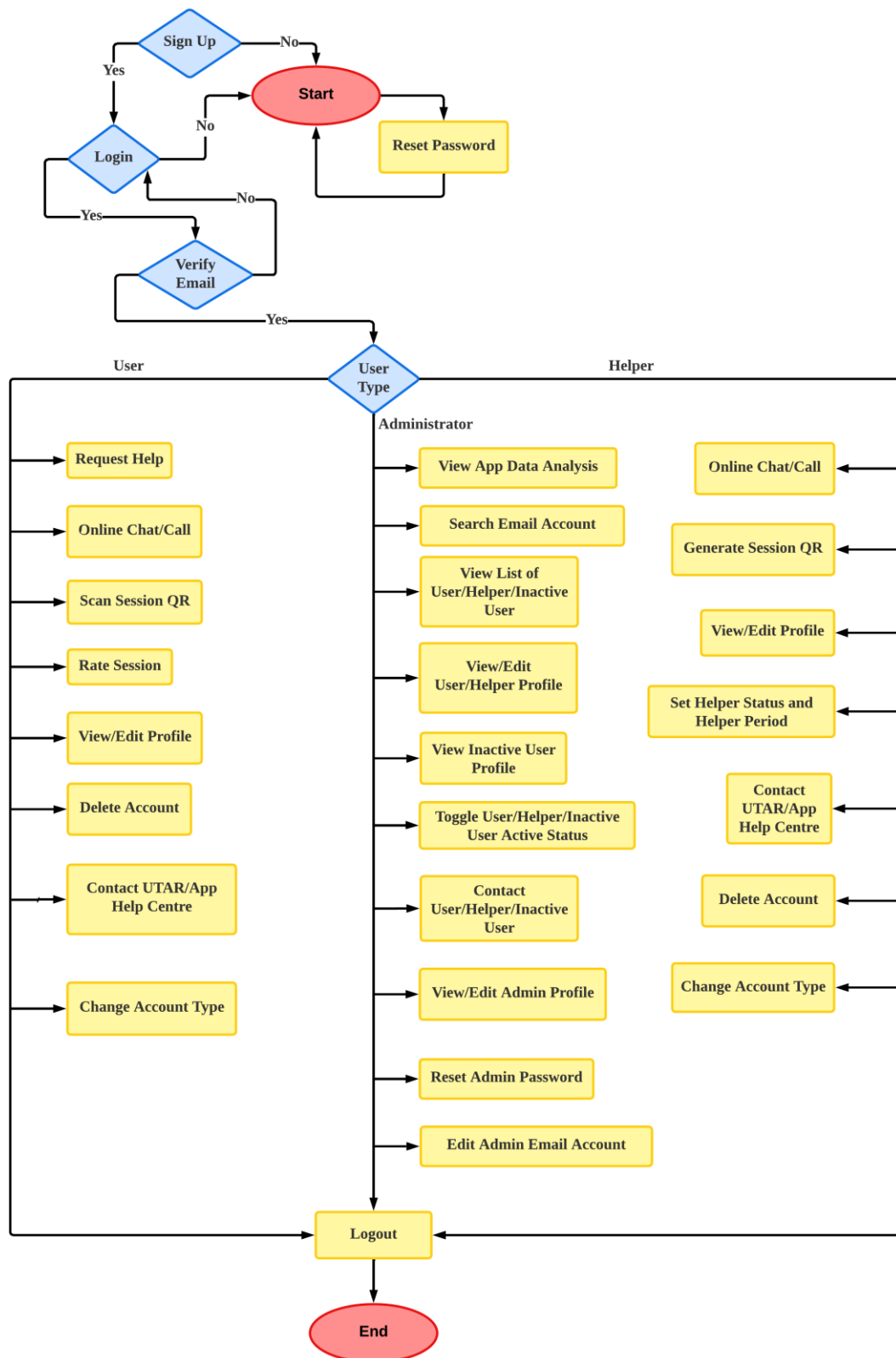


Figure 1.5.1 Flowchart of VCare Circle Mobile Application

Figure 1.5.1 shows the flowchart of the VCare Circle Mobile Application, which visualizes the overall flow of actions that can be done through the project mobile application. First, there are three options to be chosen before proceeding to use the project mobile application. The first option is the application user chooses to login and the project mobile application checks the verification, active status, and account type of the user's email account before allowing user to login. If the user has no registered account, he or she can choose the sign-up option to create and register an account in the project mobile application server to proceed login to use the project mobile application. If the user forgot his or her account password, he or she can choose to reset the account password through email reset password link. Only when one has succeeded in login into the project mobile application, the project mobile application allows the user to proceed using the project mobile application, else he or she will be prompted by the project mobile application to try login again. The three types of user accounts are user, helper, and administrator, which each of them has access to perform different actions in the project mobile application based on the privilege given.

For users, the main functions given are first and foremost to request help from any helper available from the system server and having online voice call or chat communication to communicate with the assigned helper during a session. The scanning of session QR code can also be done by the user in case he or she want to have an offline meeting with the helper to verify the helper identity. When done, the user can also choose to rate the session based on the help quality received. At the profile page, one can also proceed with viewing and editing the user profile details. In the settings page, the user can also choose to delete account if one does not want to use the account anymore, and to contact with the project mobile application help centre for technical help and UTAR help centre in case there is no available helper or in offline situation. Furthermore, in case the user wants to be a helper, he or she can choose to modify the account type to "Helper" at the navigation drawer of the project mobile application.

For administrator, he or she can view on the application data analysis such as the application usage, number of new registered users, and the overall session duration done between the users and helpers. At the home page, the administrator can also choose to search for specific email accounts registered in the system. The administrator

can also view the list of users, helpers, and inactive users. Besides, the administrator can also choose to view the account profile of users, helpers, and inactive users and toggle the account active status, but he or she can only edit the account profile of users and helpers. If the administrator wants to contact any users of the project mobile application, the admin can choose to either contact via email communication or direct phone call through the project mobile application. Apart from that, the administrator can choose to view and edit the admin profile, edit or reset admin password, and edit the admin email account.

For helpers, this type of user account basically has most of the similar functionalities as the user account type, but with some different main functions provided. When the helper is assigned to a user that has requested for help, he or she can communicate with the user through online chat or online call communication. In case there is a physical meeting between helper and user, the helper can also generate out the session QR for the user to scan as identity verification of the helper. The helper can also view and edit the account profile at the profile page, which here the helper is allowed to toggle the helper status on or off and also to set the helper period to set the duration where he or she is open for requests. Besides, the helper can also choose to contact the project mobile application help centre or the UTAR help centre in case he or she requests technical help. The helper can also choose to change the account type if he or she wishes to ask for help instead.

### **1.6 Highlights of What Have Been Achieved**

The VCare Circle mobile application is developed to provide an easier online platform for university students to ask for help without any time-bookings and worrying about who to seek help from. Throughout the online chat conversation or online voice call during the helping session, the helper's and user's identity are hidden so users can express their problem more freely. Next, this project also gives a more effective way of letting new intake students get more familiarized with the campus and surroundings. As asking for help able to let one grow and adapt to changes, new intake students can always ask for help through the project mobile application when facing issues in the new environment, to quickly solve their problems and adapt to the environment, while have a chance to widen their social field and give help to others in need in future.

The benefit of able to ask for help anytime and anywhere is also one of the achievements for the project as there are not always people around that can provide help to the ones in need. Thus, this project allows users who require help to be able to contact helpers anytime they want from the mobile application, so that they no need to wait at any place or time for the help and advice they need. The application data analysis implemented in the project can also allow administrator to view on the users and helpers' behaviour on the sessions made, to get an overview on what is troubling the university students nowadays and what is the optimal answers given by helpers, so that suggestions can be made to the university to take action in trying to solve the troubles to prevent university students facing the same trouble in the future.

### **1.7 Report Organization**

For this report, in Chapter 1, the introduction, problem statement, motivation, project objective, project scope, highlights of what have been achieved, report organization and Chapter 1 summary are included to introduce the overview of the project. In Chapter 2, related work and applications are reviewed to understand the basic requirements for the project mobile application, while comparing their strengths and weaknesses to think of ideas to improve the project mobile application. Chapter 3 includes the use case diagram with use case descriptions of the project, together with explanation on project development using project code. In Chapter 4, the system methodology of Rapid Application Development (RAD) is reviewed and discussed how it can be applied in the project, together with the declaring of the software and hardware development tools used. In Chapter 5, the overall implementation of the project is discussed and displayed in use case diagrams and use case descriptions. Finally, the chapter 6 includes the conclusion for the project, together with the novelties and contribution of the project. The future works that can improve the project are also included in Chapter 6.

### **1.8 Summary**

In summary, the introduction, problem statement and motivation, project objectives, project scope, project flowchart and highlights of what have been achieved for this project are discussed to have an overview on the project content and nature, which this chapter is served as an outline for the project for easy understanding. The importance

of asking for help should be taken into account as it brings many benefits, which the project is developed to contribute a new way for university students to ask for help, which focuses on time efficiency and privacy concerns. This project contains seven functional modules, which are login and sign-up module, profile module, QR identification module, online communication module, offline module, and the administrator module. The project's main aim is to try resolving the issue barriers when asking for help through the introducing of a new method, to improve the situation where university students cannot always get the help they need.



# Chapter 2

## Literature Review

### 2.1 Overview

This chapter includes the review on similar previous works on university community application. The purpose of reviewing the similar works is to get an overview on the functionality to see the strengths and weaknesses of the application systems, so that these factors are considered to propose solutions that can overcome the limitations while solving the problems mentioned in Chapter 1.2.

### 2.2 Previous Works on University Community

#### 2.2.1 Hi-Hive Community [7]

HI-Hive is an online application used by Universiti Tunku Abdul Rahman (UTAR) for the main purpose of letting students to scan the QR attendance given from the lecturer in class and enable students to check the history of the attendance taken for each course. This application is developed by the Silverlake Lifestyle Community Sdn Bhd development team and is still currently in the state of improving the mobile application. This application was first released on 22<sup>nd</sup> of December 2020 to get user feedback on the released version. This mobile application is mobile responsive and able to work on both Android and iOS mobile platforms. The user interface used for this application is simple and clean, with subpages included for functions such as feed page, menu page and QR scanner page for easy view.

First, to use this mobile application, the user must first login into the system with the university email account for verification as a university student or staff member. Here the user can see their information in the profile page, as well as the community blog that shows events organized in the university. Besides, there is also the messaging function on the top right of the application, which allows users to contact people through online messages. The application also allows users to receive and see the notification from the application on announcements and messages. There is also a QR

code scanner included in the application to allow students take attendance, and for all university staff and students to record the places they went to in the university.

In the profile icon at the top right of the application, it leads to the user page of the application, which shows the user's profile which contains information of the user and the community group they have joined. It also shows the activities joined by the user and the online market set if applicable. At the bottom right of the application, it also has the menu option for the application that consists of the settings for the application and also the hive store option, which is also one of the main options of the application. Inside the hive store, there are options such as to let users check the attendance record and location record, advisor and advisee meeting record, helping hand for requesting help in the application, online marketplace, e-wallet function if applicable, and also personal QR and business card that can let other users get their information.

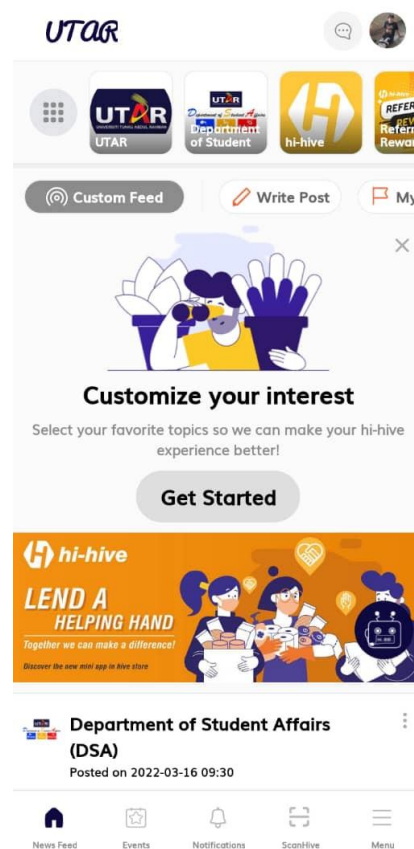


Figure 2.2.1.1 Main Page of Hi-Hive

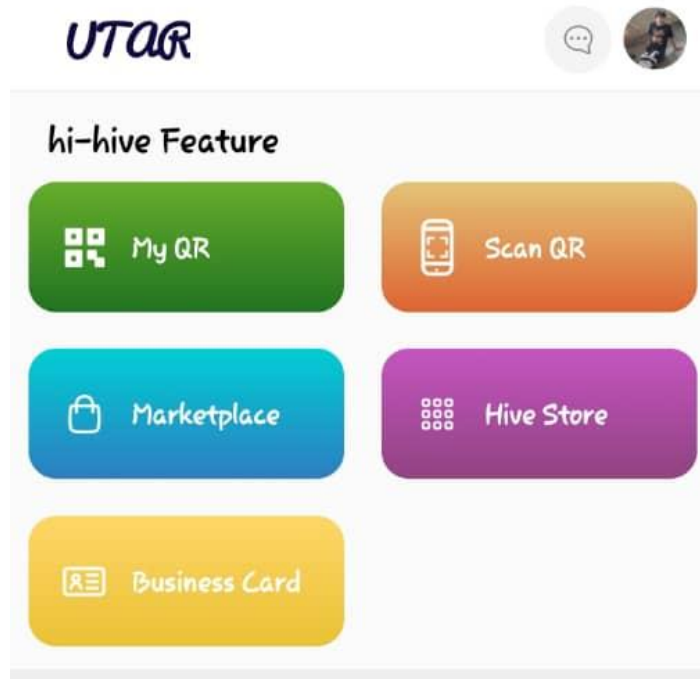


Figure 2.2.1.2 Hi-Hive Menu Options

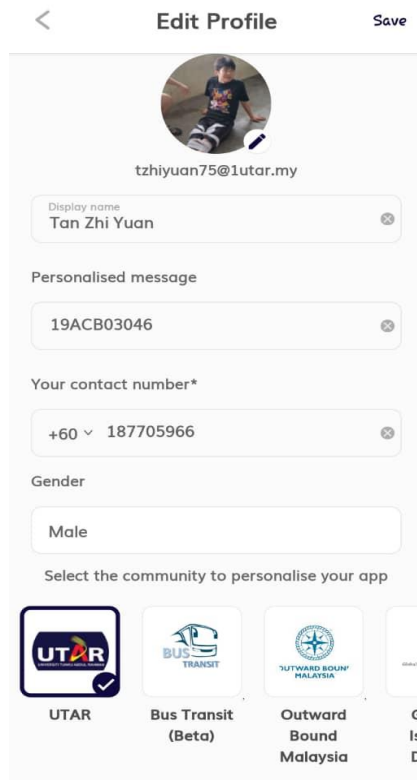


Figure 2.2.1.3 Hi-Hive Profile Edit Page



Figure 2.2.1.4 Hive Store Options in Hi-Hive

**Strength:**

- Enable fast recording and saving of students' attendance data using QR code method.
- Enable students to get the latest information of events with the availability to register with one click.
- Able to see previous record of meetings with advisor done for history checking purpose.
- Consist of its own messaging platform for communication with people from the same community.

**Weakness:**

- Unable to book sessions with advisors unless they consist of the advisor's personal contact number.
- No online call function included for real time interaction.
- Other user's contact details of the application can only be seen if they are being saved as contacts in the user's mobile contacts.

- If a meeting is required, it must be done physically as no online interaction aside messaging is provided.

### **2.2.2 Uni Course Chat [8]**

Uni Course Chat is an online mobile application developed by Unifi, which has its first version released on 24<sup>th</sup> of January 2020 and currently is in the active state of having updates to improve the system. This mobile application supports both Android and iOS platforms and requires the connection to the Internet in order to use the features in the application. This mobile application is developed for the purpose of letting university students build communities to communicate and help each other via this online platform.

In the application, simple yellow themed user interface design is used for a clean layout, with subpages for different functions such as discover function, hub function and chat function for clear divider and easy view. The user of this application is required to input the university name, student email address and password as the main information required, along with information such as the degree course taken, starting year of degree name and goals. The first page of the app shows the campus hub of the university with posts on things happening in university, including announcements from the universities and ongoing event posts.

In the second page of the application, it shows the discover page which has some recommended people to be added as friends for users to widen their social circle. When the user selects one of the user's profiles shown, it shows the photo, name, degree course taken, and short description of the person to let the user check user information before deciding whether to add the person as friends.

Then, the mobile application also consists of the online messages and group chat function for users to communicate with each other. Here the application also shows the group chats created by students in order to let the user choose to enter any preferable group chat that they want to have more interaction with other students from the same university. Apart from that, the user is also able to set the notification settings of the

application to receive message notification, and the application also allows modification of the user's information and interest if required.

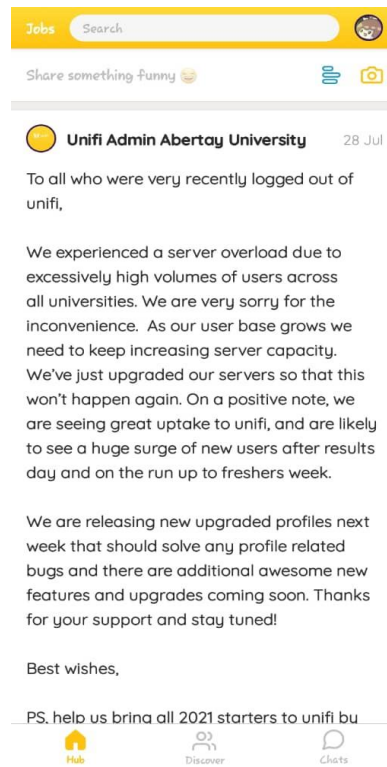


Figure 2.2.2.1 Hub Page of Uni Course Chat



Figure 2.2.2.2 Discover Page of Uni Course Chat

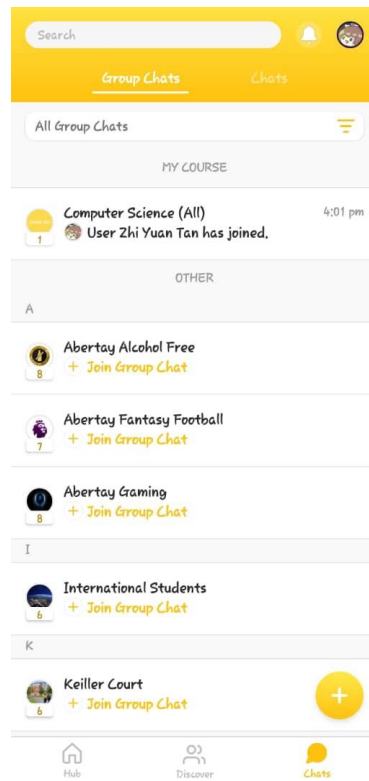


Figure 2.2.2.3 Group Chat Page of Uni Course Chat



Figure 2.2.2.4 Chat Function in Uni Course Chat

**Strength:**

- Easy-to-use simple UI design is used in this application to not confuse the user.

Bachelor of Computer Science (Honours)

Faculty of Information and Communication Technology (Kampar Campus), UTAR

- Enable easy finding of other users in the application based on similar interests.
- Consist of searching functions to filter posts and search other added users in the application.
- Able to configure on and off the notification settings based on the user's preference.

### **Weakness:**

- No professionals or lecturer contacts included if they require help from higher ups.
- No other methods such as QR code scanning were included for easier friend adding and sharing of personal account information instead of the traditional method of searching a user's name.
- No online calling function is included in the application for easy interaction.
- Unable to locate any available helper if required asking for help.
- No showing of available status of contacts in the application.

### **2.2.3 Bronx CC [9]**

The Bronx Community College application is an online mobile application developed by Ready Education Inc. on 17<sup>th</sup> of April 2017. This application is developed for the purpose of letting people to communicate and get familiar with the surroundings of Bronx Community College. This application can support both Android and iOS platforms and it is still currently in the state of improving. Simple UI design is used in this application for a clean layout to ensure the easy navigating of the application for the users.

To use this application, one must possess a Bronx account to login as student, faculty, and staff. As those who do not have a Bronx account, they can choose to login as visitors, friends, and alumni of the university to use the application. When the user login without a Bronx account, they can only access to see information about the university, such as a map of the university, ongoing events, directory of the university, news and announcements and so on.

When a student login using the Bronx account, they can get full access to the features given in the application. First of all, the same as when login without an account, the



students can view information, news and events happening, and the map that marks the places in the university. Next, the student can also receive notification on academic matters and assignment deadlines in the notification list. The mobile application also has the function of allowing students to post feeds in the community that they have joined to ask questions or to make new friends. Online messaging function is also provided for the users when the users has added friends in the application through the searching of friend's user ID in the application.

Apart from that, the application also allows the function of searching available services in the campus that let students to fully utilize the use of university facilities. The joining of groups and clubs is also included in the application that allows students to join so that they can be in the circle of people with the same interests and goals to widen their social circle in university. By joining the clubs and groups, the student is also able to receive new notifications in the notification list on the updates of posts released by the groups and clubs.

When one login as a university student in the mobile application, the student is able to use the application to manage classes, homework and assignments during their studies and also to stay updated on the important news and reminders regarding to their ongoing work and tasks, to ensure that they do not miss out any important tasks and updates that affects their academic performance.



Figure 2.2.3.1 Main Visitor Page of Bronx CC

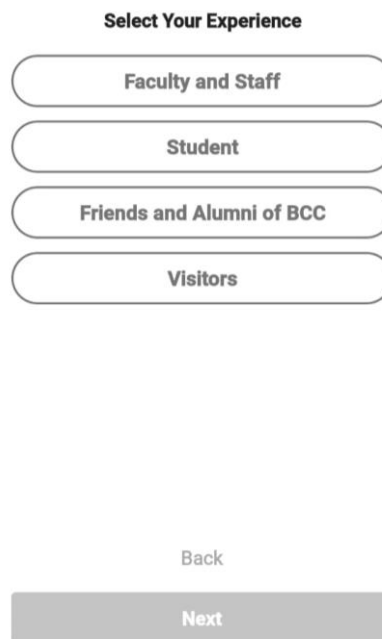


Figure 2.2.3.2 Type of User Login of Bronx CC

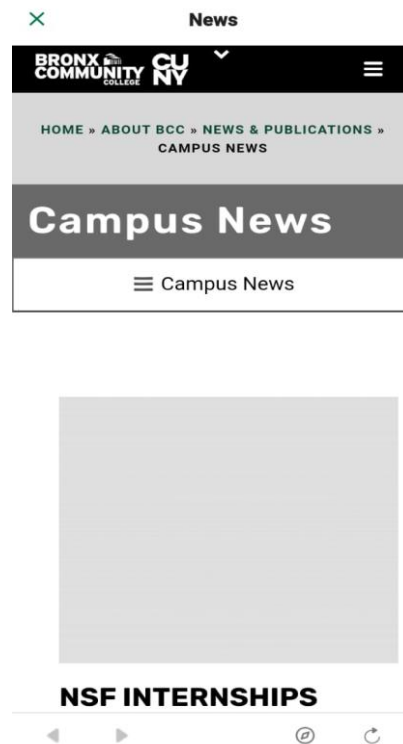


Figure 2.2.3.3 News Page of Bronx CC



Figure 2.2.3.4 Social Media Platform of University of Bronx CC

**Strength:**

- Consist of multiple user login privilege in using the mobile application features.
- Has a calendar function for university students to add their study schedules and reminders.
- Has a map function that shows the places around the university.
- Able to show all information regarding the university including the bus services and library services.

- Able to show users the important contacts that can be used when they have faced issues.

### **Weakness:**

- Does not have any online communication methods to other users unless they are added as friends to use messaging.
- Does not have subpages to separate the functions into categories as all the functions are shown as a list in the mobile application.
- No function for lecturers giving professional help regarding more major matters such as academic advice is given in this mobile application.
- There is no immediate way of asking for help aside from posting the request in the community feed.

### **2.2.4 Differ.Chat [10]**

Differ.Chat is an online application that is developed for the purpose of improving the lifestyle of university students by allowing them to make more friends in their university life so that they can ask for help easily and have a sense of belonging in their university. This application was developed by Differ starting on 25<sup>th</sup> of June 2016 and their last update was on 15<sup>th</sup> of November 2021. While this application allows both Android and iOS mobile platform, it also supports Mac OS and Windows 10 for desktop and laptop online browsing and download. Here the application uses a green themed UI design to have a beautiful and clean layout interface for easy navigation in the application. Subpages are used to divide the main functions of the application, such as the main page, discover page, online chat page, and community list page.

To use this application, the user starts off by entering their phone number and must be invited to any community in the application beforehand to use the application. The main feature provided in the application for university students is enabling online chat messaging with other users in the application, while allowing the university staff to manage the community activities. They can have informal chatting with people without sharing their phone number to keep their privacy before deciding to make friends with them. Notification is provided for the user to keep track of new messages and announcements given by the university lecturers in the community entered.

Bachelor of Computer Science (Honours)

Faculty of Information and Communication Technology (Kampar Campus), UTAR

Not only does this application allow online chatting among university students, but this application also allows university lecturers to create specific course related communities for students to join in. Here the students can see the information and updates for the course posted by the lecturers. University students can also find help from people in the community and also possibly the teaching assistant and lecturer when they face questions in their studies.

Apart from that, this application also has a social chatbot function which helps university students to recommend friends with the same courses or same interests and goals. The purpose of this function is to improve the relationship between university students to build a strong student community that can help each other when there are people in need.

Furthermore, this application also allows students to give their opinions on a certain matter anonymously to the higher ups or university lecturers. This function allows students to bravely speak out their opinion without worrying about identity being exposed, which allows them to be true to themselves when voicing out to people with higher authority in the university.

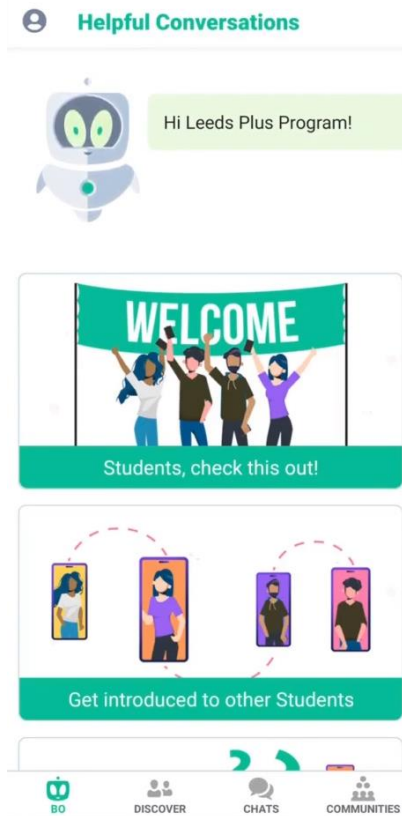


Figure 2.2.4.1 Main Page of Differ.Chat [11]

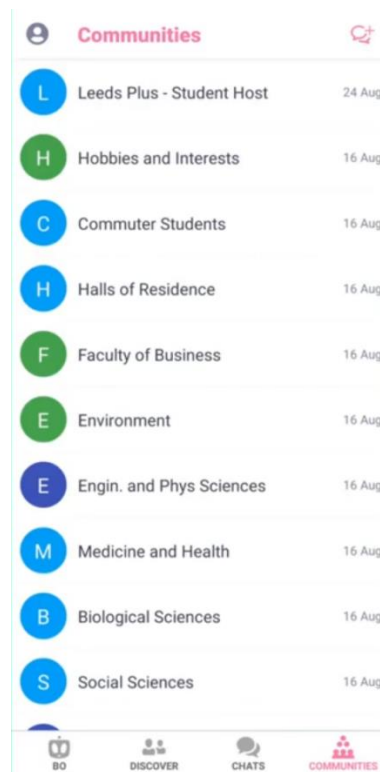


Figure 2.2.4.2 Community List of Differ.Chat [11]

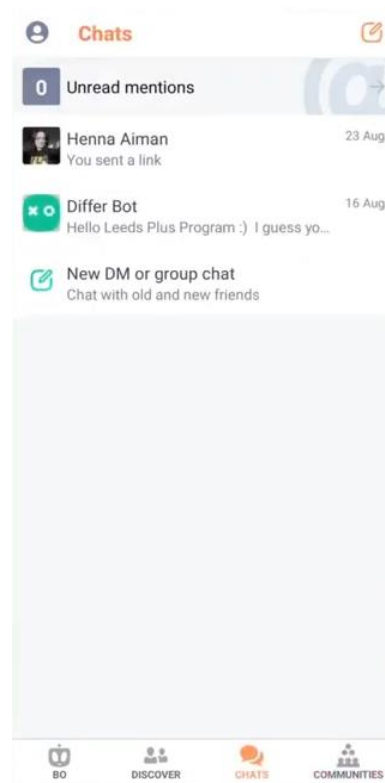


Figure 2.2.4.3 Chat Page of Differ.Chat [11]

**Strength:**

- Simple and clear UI design is used in the application.
- Mobile responsive to different screen sizes and devices to have proper layout when using the application.
- Allow the support of multiple platforms with different operating systems.
- Allow university students to get familiar with their university surroundings with the joining of multiple communities and chatbot recommendation.
- Allowing the university students to voice out their opinion to people easily without worrying upsetting the higher ups.

**Weakness:**

- New users cannot start using the application if they are not invited to any community.
- No online calling function is provided for real-time interaction.
- No map function for searching places around the university.
- Important information of the university such as emergency number and office contact is not given in the application.

- No contacts of university lecturers are shown besides communicating through the application messaging.

### **2.2.5 Lewis University [12]**

Lewis University (LewisU) is a mobile application for Lewis University students developed by Ready Education Inc. The first version was released on 11<sup>th</sup> of August 2014 and the latest update was on 3<sup>rd</sup> of November 2020. The purpose of this application is to allow Lewis University students to gain full access to the news, services and facilities of the university, and also for the purpose of allowing community communication for the university students. The application uses a simple UI design for the application for easy navigation to increase the user's experience.

To gain full access to all the features of the application, the user must first login as student or as staff using the Lewis account. As for the users without the account, they can still login into the application and use some of the features in the application. First of all, when login as a student, the university student is able to see the classes they registered for the semester and is able to do managing such as adding reminders and stay up to date on updates relating to the tasks and assignments to be done.

Next, the students can also see notification and information on the events in the notification list. The services provided by the university can also be browsed by the student in the mobile application for the purpose of letting university students know about them and use them when they have the needs for the services. The deals function in the application also shows information on the discounts ongoing in the university, such as research books and online courses provided.

Besides, the mobile application also has the function of allowing students to join groups and clubs to view information of the clubs and groups provided. The mobile application also provides a tour function that allows users to have a brief explanation of the important places around the university, while also having a map function that can help the students by showing the routes to get to the places or buildings required.



Apart from that, there is also the function of campus feed to allow students to share their opinions and voices by joining the discussion in the feed page. To allow the university students to find new friends in the university, there is also a student list function that allows the university students to find new friends with similar courses or interests and goals and proceed to call them through mobile phone calls.

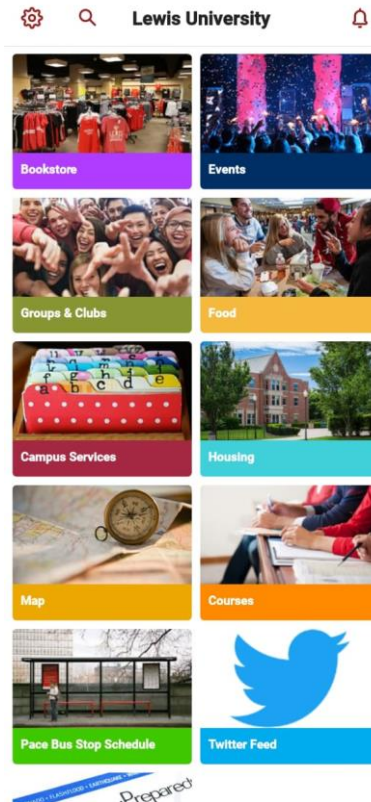


Figure 2.2.5.1 Main Page of LewisU



Figure 2.2.5.2 Event List of LewisU

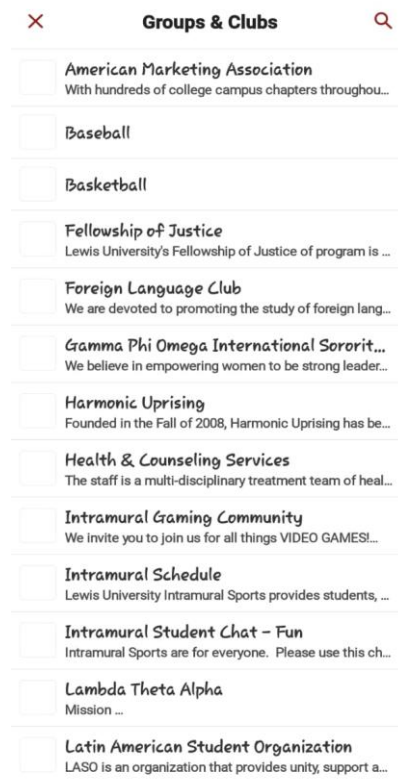


Figure 2.2.5.3 Group and Clubs List of LewisU

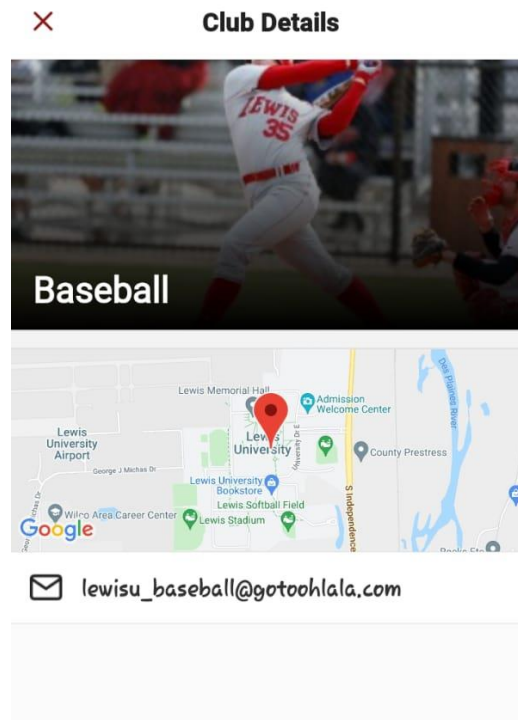


Figure 2.2.5.4 Club Details Page of LewisU

**Strengths:**

- Consist of tour guide function to help university students to get an understanding around the university.
- Allow the searching of new friends from the student lists.
- Able to do check-in and check-out of events with QR scanning.
- Has a map function to allow students to find places around the university.
- Help centre is provided to help students who have trouble when using the application.

**Weakness:**

- No online calling function is found in this application.
- Online messaging can only be done if both sides of the users accept to communicate with each other.
- No helper function is provided to request doing physical tour guides for students.
- No professionals or lecturers are in the application to answer students' questions regarding more serious matters such as questions for final year project (FYP) and academic matters.

- All function icons are all listed in the same page, which has affected the clean layout of the application.

### **2.3 Summary**

From this chapter, five existing software applications have been reviewed to identify the overall functionality, strengths, and weaknesses in the software application systems. By viewing the comparison between the strengths and weaknesses of each software system, the list of important functionalities and design for a university community application can be identified and able to be used in the project as reference, such as the user login function, online chat and voice call function and user profile function. From here, new ideas to solve the limitation of the previous works can also be identified, such as helper finding module and QR identity verification to improve the limitation of the previous work and contribute as a new method of asking for help for the ones in need.

Table 2.3.1 Comparison of the Strengths and Weaknesses of the 5 Existing Application

	<b>Hi-Hive Community</b>	<b>Uni Course Chat</b>	<b>Bronx CC</b>	<b>Differ.Chat</b>	<b>Lewis University</b>
<b>User Login</b>	Yes	Yes	Yes	Yes	Yes
<b>User Profile Roles</b>	Student and staff	Student	Student, staff, visitors	Student and staff	Student, staff, visitors
<b>Online Communication Method</b>	Online messaging	Online messaging	Online messaging	Online messaging	Online messaging
<b>Map Function</b>	Not provided.	Not provided.	Provided to show places around the university.	Not provided.	Provided to show places around the university.
<b>QR Code Scanning Function</b>	For recording attendance, places went, and to show user information.	Not provided.	Not provided.	Not provided.	For registering and doing check-ins and check-outs of events.

CHAPTER 2 LITERATURE REVIEW

<b>Discover New Friends</b>	Not provided.	Provided in the discover page or search function.	Provided in the search function.	Provided in the chatbot function.	Provided in the search function or student list.
<b>Application UI Design</b>	Simple design with subpages.	Simple design with subpages.	Simple design with no subpages.	Simple design with subpages.	Simple design with no subpages.
<b>Notification Function</b>	Provided for announcement and chat.	Provided for chat.	Not provided.	Provided for announcement and chat.	Not provided.
<b>Helper Function</b>	Allow students to request help in the application.	Not provided.	Not provided.	Not provided.	Not provided.
<b>Announcements From the University</b>	Provided in the feed page and notification.	Provided in the feed page.	Provided in the list of announcements.	Not provided.	Provided in the list of announcements.
<b>Platform Supported</b>	Android, iOS mobile platform.	Android, iOS mobile platform.	Android, iOS mobile platform.	Android, iOS mobile platform and Windows and Mac OS desktop platform.	Android, iOS mobile platform.

# Chapter 3

## System Design

### 3.1 Overview

This chapter discusses on the project’s system design, which includes the user diagram along with use case descriptions. The project’s coding work including brief explanation to explain on the development process is also discussed in this chapter.

### 3.2 Design Specification

#### 3.2.1 Use Case Diagram

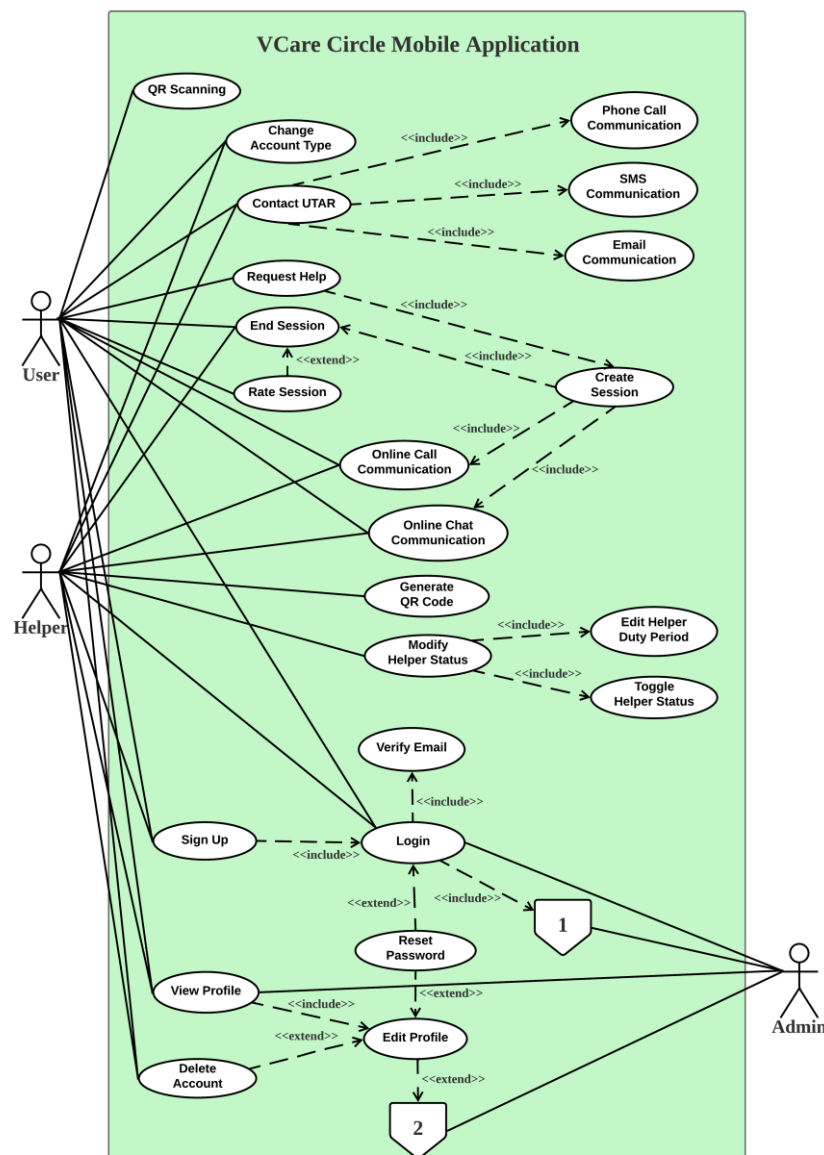


Figure 3.2.1.1 Use Case Diagram for VCare Circle Mobile Application Part A

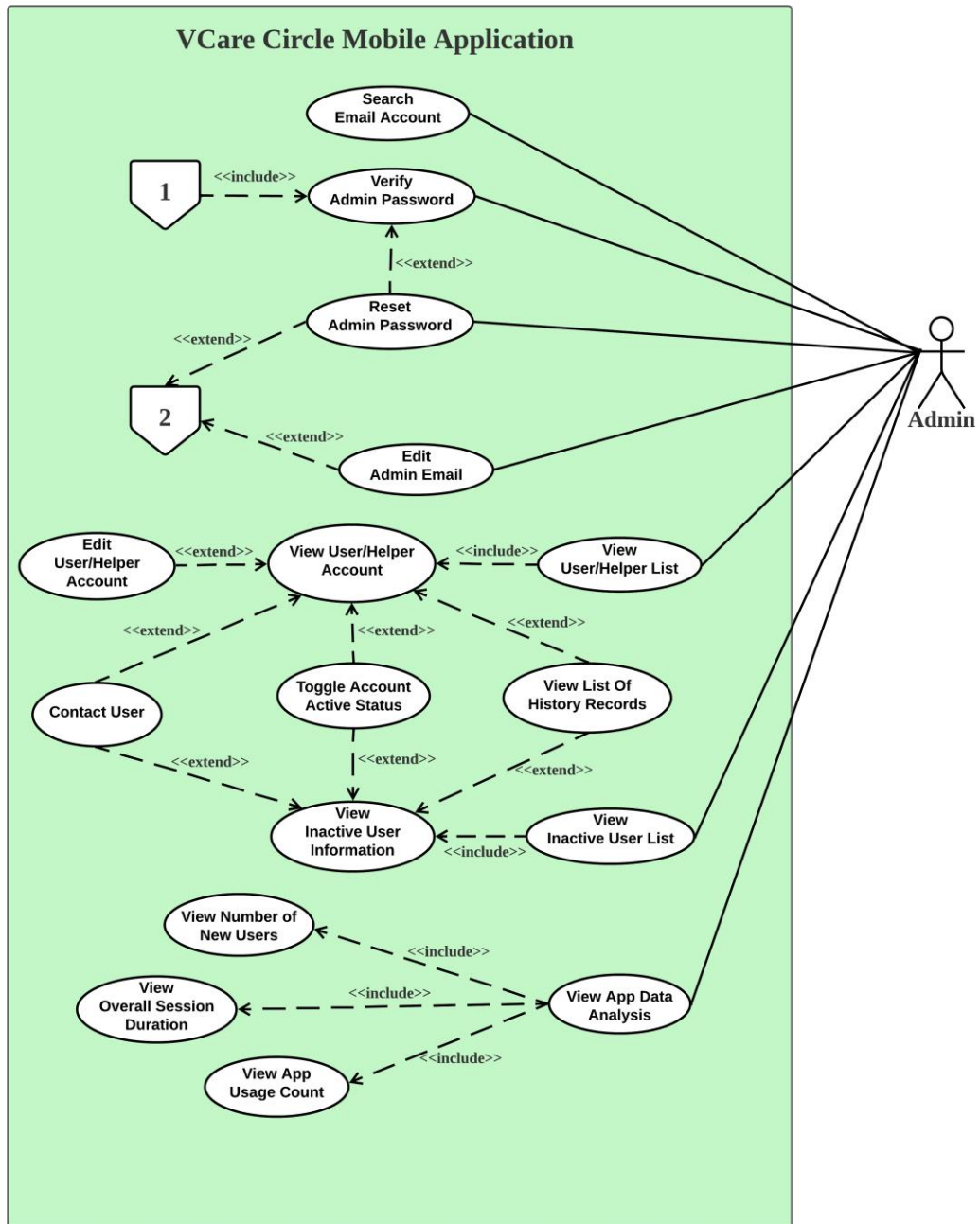


Figure 3.2.1.2 Use Case Diagram for VCare Circle Mobile Application Part B



## 3.2.2 Use Cases Descriptions

Table 3.2.2.1 Login Use Case Description

<b>Use Case ID</b>	00001
<b>Use Case Name</b>	Login
<b>Brief Description</b>	User or helper login to his or her account to gain access to functionalities in the mobile application, or administrator login to gain access to the administrator controls of the mobile application.
<b>Actor</b>	User, Helper, Admin
<b>Trigger</b>	When user, helper or admin enters the login page of the mobile application.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User, helper, or admin account must already be registered in the application server.</li> <li>- User or helper's account need to be in active status.</li> <li>- Admin requires to have the admin password.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User, helper, or admin enters their account email and key in the account password.</li> <li>2. The system then checks if the account exists, and password is correct.</li> <li>3. The system continues to check if the account is verified.</li> <li>4. The system registers the notification token of the user, helper, or admin's device to the server.</li> <li>5. The mobile application saves the login information of the user, helper, or admin, such as account and ID.</li> <li>6. The user, helper, or admin are directed to the respective home page of the mobile application.</li> </ol>
<b>Sub Flows</b>	<ol style="list-style-type: none"> <li>1a. User, helper, or admin does not need to login again to the application if they did not choose the log out of the application.</li> <li>3a. User or helper does not need to verify their email account again if they have done so previously when signing into the app.</li> <li>3b. Admin does not need to verify email as it is confirmed to be verified in the system server.</li> <li>6a. Admin are being directed to the verify admin password page before gaining access to login as admin and directed to the admin home page.</li> </ol>

<b>Alternative Flows</b>	<p>2a. If the input email account does not exist in the system or the inputted password is invalid, an error message is shown to notify the user or helper on the invalid login.</p> <p>3c. The email account has been set to inactive status by admin, error message is shown to the user or helper, and he or she is not allowed to login into the project mobile application.</p>
--------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 3.2.2.2 Sign Up Use Case Description

<b>Use Case ID</b>	00002
<b>Use Case Name</b>	Sign Up
<b>Brief Description</b>	User or helper signs up an account with their email before getting access to use the mobile application.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When user or helper enters the sign-up page of the mobile application.
<b>Precondition</b>	User or helper need to have a valid and usable email account.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper enters their personal information and select user type.</li> <li>2. The user proceeds to enter their email account and password.</li> <li>3. The system saves the user information to the server and direct user or helper to the verify email page to verify their email via link send to their email inbox.</li> <li>4. After verifying the email account, the user or helper is directed to the home page of mobile application.</li> </ol>
<b>Sub Flows</b>	<p>1a. When account type of “Helper” is selected, user is asked to confirm their selection and if they want to add in helper duty period.</p> <p>3a. The system checks if the email account information was previously saved in the inactive user collection, if got record then remove it from the inactive user collection in project mobile application server.</p>
<b>Alternative Flows</b>	2a. If the inputted email account is not valid, the password is in the wrong format, and the confirm password input is not the same as the password set, error message is shown to the user or helper until all information are valid.

Table 3.2.2.3 Verify Admin Password Use Case Description

<b>Use Case ID</b>	00003
<b>Use Case Name</b>	Verify Admin Password
<b>Brief Description</b>	The admin requires to enter the admin password before gaining access to use the administrator controls of the mobile application.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin has successfully login into his or her account.
<b>Precondition</b>	Admin has successfully login into the admin account.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin is directed to the verify admin password page after successfully login with the email account and account password.</li> <li>2. Admin enters the admin password in the provided text field.</li> <li>3. The system checks if the admin password is correct.</li> <li>4. The system saves the admin login key information into the project mobile application.</li> <li>5. After verifying the admin password login, admin is directed to the admin home page.</li> </ol>
<b>Sub Flows</b>	1a. Admin does not need to verify admin password again if previously admin did not choose to log out of the application.
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>2a. The admin password is incorrect or empty when submitted, error message is shown to notify admin on the wrong password and system prompts for password input again.</li> <li>2b. Admin chooses to reset the admin password by selecting the “Forget Password” option.</li> </ol>

Table 3.2.2.4 Reset Admin Password Use Case Description

<b>Use Case ID</b>	00004
<b>Use Case Name</b>	Reset Admin Password
<b>Brief Description</b>	Admin chooses to reset or update the admin password used to login as admin to use the administrator controls of the project mobile application.
<b>Actor</b>	Admin

<b>Trigger</b>	When admin selects the “Forget Password” option at the verify admin password page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has successfully login into the admin account.</li> <li>- Admin is using the main administrator mobile device.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the “Forget Password” option at the verify admin password page.</li> <li>2. The system checks if the admin email is correct and identify if the current device is the admin’s main device.</li> <li>3. The system displays a dialog that requests admin to input new admin password.</li> <li>4. The admin inputs the new admin password at the input text form field given.</li> <li>5. The admin re-enters the new admin password at the “Confirm Password” text form field provided to double check the new admin password and presses the “Submit” button.</li> <li>6. The system checks if the new admin password is valid and proceed to update the admin password in the project mobile application server.</li> <li>7. The admin is directed back to the verify admin password page to login again with new admin password.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>1a. The admin chooses the “Reset Password” option at the edit admin profile page to reset admin password instead.</li> <li>2a. The admin email is incorrect, or the device used is not the admin’s main device, error message is shown to admin to notify that admin is not authorized to reset password.</li> <li>4a. The admin enters an invalid new admin password or empty value, error message is shown to the admin to notify him or her to try again.</li> <li>5a. The re-entered new admin password is incorrect or empty value, error message is shown to the admin to notify him or her to try again.</li> <li>5b. Admin presses the “Cancel” button to cancel the reset of admin password, no changes made.</li> </ol>

Table 3.2.2.5 Verify Email Use Case Description

<b>Use Case ID</b>	0005
<b>Use Case Name</b>	Verify Email
<b>Brief Description</b>	User and helper needs to verify their email via link provided to proceed using the mobile application.
<b>Actor</b>	User, Helper
<b>Trigger</b>	After user or helper completes the account sign up process in the mobile application before being directed to the respective home pages.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper need to have a valid email.</li> <li>- User or helper completes the account sign up process.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper is directed to the verify email page.</li> <li>2. A message is shown to user or helper that an email verification link is sent to his or her email inbox.</li> <li>3. User or helper is required to check the email inbox to click the link to verify their email accounts.</li> <li>4. After verifying the email, user or helper is directed to their respective home page to gain access to the mobile application's functionalities.</li> </ol>
<b>Sub Flows</b>	2a. The user or helper also can get the verification link again by pressing the "Email Verification Link" button.
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>3a. If there is no email verification link found in the email inbox, one can go check the email spam folder for the link.</li> <li>4a. If user or helper does not verify the email through email verification link, user or helper remains in the email verification page.</li> </ol>

Table 3.2.2.6 Reset Password Use Case Description

<b>Use Case ID</b>	0006
<b>Use Case Name</b>	Reset Password
<b>Brief Description</b>	User, helper, or admin can choose to reset their account password by pressing the "Forgot Password" option at the login page.
<b>Actor</b>	User, Helper, Admin

<b>Trigger</b>	When the user, helper or admin selects the “Forgot Password” option in login page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- The user, helper, or admin has a valid email account.</li> <li>- User, helper, or admin has a registered account in the system.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User, helper, or admin selects the “Forgot Password” option at the login page.</li> <li>2. User, helper, or admin is directed to the reset password page to enter their email account.</li> <li>3. User, helper, or admin press the send email link button to receive a reset password email in the entered email account.</li> <li>4. User, helper, or admin enters the new password in the provided link.</li> <li>5. User, helper, or admin can login with the new password.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	1a. User or helper selects the “Change Password” option in the settings page to reset the account password if they preferred a different password instead.

Table 3.2.2.7 View Profile Details Use Case Description

<b>Use Case ID</b>	00007
<b>Use Case Name</b>	View Profile Details
<b>Brief Description</b>	User, helper, or admin can view and edit their profile details in the mobile application.
<b>Actor</b>	User, Helper, Admin
<b>Trigger</b>	When user, helper, or admin enters the profile page specified for them in the mobile application.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User, helper, or admin need to have an account registered in the system.</li> <li>- User, helper, or admin has logged in the project mobile application.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. From the navigation drawer, the user, helper, or admin chooses to enter the respective profile page.</li> </ol>

	<p>2. User, helper, or admin can view on the profile information saved in the project mobile application system.</p> <p>3. User, helper, or admin can choose to edit the profile by selecting the edit profile icon button.</p> <p>4. Helper can view and edit the helper status and helper duty period in helper details section.</p> <p>5. User, helper, or admin can choose to log out his or her account by selecting the log out option.</p>
<b>Sub Flows</b>	<p>3a. When the edit profile button is selected, the user, helper, or admin is directed to the edit profile page to edit their account personal details.</p> <p>4a. Helper can choose to toggle on or off for the helper status, and to edit or remove helper duty period.</p>
<b>Alternative Flows</b>	<p>4a. If a user account type “User” tries to enter the helper details page, the user cannot be directed to helper details page because he or she is not a helper.</p> <p>5a. If user or helper is in an active session, the log out option is considered an invalid action, user or helper cannot proceed to log out.</p>

Table 3.2.2.8 Edit Admin Email Use Case Description

<b>Use Case ID</b>	00008
<b>Use Case Name</b>	Edit Admin Email
<b>Brief Description</b>	Admin can choose to edit the email account used as the administrator email account.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin edits the email account at the edit admin profile page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- The new email account must have been previously registered in the system server.</li> </ul>
<b>Normal Flow of Events</b>	<p>1. Admin enters the edit admin profile page to edit the admin information.</p>

	<ol style="list-style-type: none"> <li>2. Admin enters a new email account at the email account text form field provided and press the “Save” button.</li> <li>3. The system checks if the new email is in the correct format and the new email is previously registered in the system server.</li> <li>4. The system displays a confirmation dialog to request for the admin to enter the admin password.</li> <li>5. The admin enters the admin password and presses the submit button.</li> <li>6. The system checks if the admin password is valid and proceeds to update the admin email account in the project mobile application server.</li> <li>7. An email is sent to the new email account notifying the new email account is now the admin email account.</li> <li>8. The admin is logged out of the project mobile application to login into the project mobile application with the new email account instead.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>3a. The new email account is in invalid format, or not found under the registered list of emails in system server, error message is shown to the admin and request admin to try again.</li> <li>5a. Admin chooses the “Cancel” button to cancel the edit admin email operation, no changes made.</li> <li>6a. The admin password entered is incorrect or empty value, error message is shown to admin to notify admin to key in the correct admin password.</li> </ol>

Table 3.2.2.9 View User or Helper List Use Case Description

<b>Use Case ID</b>	00009
<b>Use Case Name</b>	View User or Helper List
<b>Brief Description</b>	Admin chooses to view the list of users or helpers registered in the project mobile application server to view on the list of registered accounts and any specific account’s details.
<b>Actor</b>	Admin



<b>Trigger</b>	When admin selects the “User List” or “Helper List” at the admin navigation drawer.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin opens the admin navigation drawer and selects the “User List” or “Helper List” option to direct to the list of users or helpers’ page.</li> <li>2. The system retrieves the list of registered users or helpers from the user collection in system server.</li> <li>3. The system displays out the retrieved list of user or helper to the admin.</li> <li>4. Admin can view on the summarized details of the users or helpers from the list of user or helpers displayed.</li> </ol>
<b>Sub Flows</b>	4a. Admin can select on any account from the list of users or helpers to direct to the user or helper account details page to view on the full details of the account.
<b>Alternative Flows</b>	1a. Admin selects the user or helper counter at the admin home page to direct to the list of users or helpers page instead.

Table 3.2.2.10 View User or Helper Account Use Case Description

<b>Use Case ID</b>	000010
<b>Use Case Name</b>	View User or Helper Account
<b>Brief Description</b>	Admin selects one of the user or helper accounts from the list of users or helpers to view the full details of the user or helper account.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects one of the account tiles from the list of users or helpers in the list of users or helpers’ page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> </ul>

<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects one of the account tiles in the list of users or helpers page to view the full details of the user or helper account.</li> <li>2. The system directs admin to the view user or helper account details page.</li> <li>3. The system retrieves the user or helper information from the project mobile application server.</li> <li>4. The system displays the user or helper information to the admin in the account details page.</li> </ol>
<b>Sub Flows</b>	<ol style="list-style-type: none"> <li>4a. Admin can choose to view the list of history sessions at the list of history sessions page involving the user or helper by selecting either the user request counter or the helper duty counter.</li> <li>4b. Admin can choose to edit the user or helper account information by selecting the fields or icons at the information display provided.</li> <li>4c. Admin can choose to toggle the user or helper account's active status at the top right corner of the user or helper account details page.</li> <li>4d. Admin can choose to contact the user or helper through either direct phone call or email communication by selecting the icon buttons provided.</li> </ol>
<b>Alternative Flows</b>	-

Table 3.2.2.11 Edit User or Helper Account Use Case Description

<b>Use Case ID</b>	000011
<b>Use Case Name</b>	Edit User or Helper Account
<b>Brief Description</b>	Admin selects one of the user or helper account from the list of users or helpers to edit of the user or helper account.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects one of the account tiles from the list of users or helpers in the list of users or helpers' page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> </ul>

<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects one of the account tiles in the list of users or helpers page to direct to the view user or helper account details page.</li> <li>2. The system retrieves the user or helper information from the project mobile application server and displays to the admin.</li> <li>3. Admin chooses which information of the user or helper account to be updated.</li> <li>4. Admin saves the changes made and the system updates the changes made for the account in the project mobile application server.</li> </ol>
<b>Sub Flows</b>	<ol style="list-style-type: none"> <li>3a. Admin inputs the new name or phone number of the user or helper account in the text form field provided.</li> <li>3b. Admin chooses to switch the gender of the user or helper account by tapping on the gender logo that shows the confirmation dialog before proceeding update.</li> <li>3c. Admin chooses to update the status of the user or helper account by selecting one of the options at the drop-down menu button provided, a confirmation dialog is shown to confirm with admin to proceed update the status.</li> <li>3d. Admin chooses to set the user or helper account's account status to inactive by selecting the icon button at the top right corner of the account details page.</li> </ol>
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>3a. The new input value of name or phone number is empty value or invalid format, error message is shown to admin to notify admin to try again.</li> <li>3b. Admin cancels the decision to switch the user or helper account gender, no changes made.</li> <li>3c. Admin cancels the decision to switch the user or helper account status, no changes made.</li> <li>3d. Admin cancels the decision to set the user or helper account's account status to inactive, no changes made.</li> </ol>

Table 3.2.2.12 Toggle Account Active Status Use Case Description

<b>Use Case ID</b>	000012
<b>Use Case Name</b>	Toggle Account Active Status

<b>Brief Description</b>	Admin wants to set the user, helper, or inactive user's account status to active or inactive to decide whether to allow him or her to have access in using the project mobile application.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects the icon button to toggle the user, helper, or inactive user active status to active or inactive.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> <li>- There are inactive user records in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin navigates to the view user, helper, or inactive user account details page to toggle the active status of user, helper, or inactive user account.</li> <li>2. The systems check the current active status of the selected account.</li> <li>3. The systems display a dialog to get admin's confirmation to change the account's active status.</li> <li>4. The system proceeds to update the active status of the account to inactive or active.</li> </ol>
<b>Sub Flows</b>	4a. The inactive user or helper account will be moved to the inactive user collection in project mobile application server and is moved to the inactive user list.
<b>Alternative Flows</b>	4b. If inactive user account active status is set to active, the account is moved back to the user collection in the project mobile application server, and the account is moved to the user or helper list based on previous record.

Table 3.2.2.13 Contact User Use Case Description

<b>Use Case ID</b>	000013
<b>Use Case Name</b>	Contact User

<b>Brief Description</b>	Admin wants to contact user, helper, or inactive user through either direct phone call from mobile device or email communication through project mobile application.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects the icon buttons to contact a user, helper, or inactive user through either direct phone call or send email through the project mobile application.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> <li>- There are inactive user records in the project mobile application server.</li> <li>- Admin has internet access and phone call service when using the project mobile application.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin navigates to a specific account of user, helper, or inactive user account details page to contact user, helper, or inactive user with the contact information saved.</li> <li>2. Admin selects the “Email” icon button to send an email to the user, helper, or inactive user.</li> <li>3. The system displays a dialog to allow admin to enter the message to be delivered.</li> <li>4. Admin enters the email message and press the “Ok” button to submit.</li> <li>5. The system checks if the message is filled in properly by the admin.</li> <li>6. The system sends out the email through the SMTP server service added to the project mobile application.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>2a. Admin selects the “Phone Call” icon button to make direct phone call with user, helper, or inactive user through mobile device instead.</li> <li>5a. Admin fills in an empty message for the email, error message is shown to the admin to notify admin to fill in the message properly.</li> </ol>

Table 3.2.2.14 View Inactive User List Use Case Description

<b>Use Case ID</b>	000014
<b>Use Case Name</b>	View Inactive User List
<b>Brief Description</b>	Admin chooses to view the list of inactive users saved in the project mobile application server to view on the list of inactive users records and any specific account's record details.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects the "Inactive User List" at the admin navigation drawer.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered inactive user account records in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin opens the admin navigation drawer and selects the "Inactive User List" option to direct to the list of inactive users' page.</li> <li>2. The system retrieves the list of inactive user records from the inactive user collection in system server.</li> <li>3. The system displays out the retrieved list of inactive users to the admin.</li> <li>4. Admin can view on the summarized details of the inactive users from the list of account records displayed.</li> </ol>
<b>Sub Flows</b>	4a. Admin can select on any account from the list of inactive users to direct to the inactive user account details page to view on the full details of the account.
<b>Alternative Flows</b>	-

Table 3.2.2.15 View Inactive User Information Use Case Description

<b>Use Case ID</b>	000015
<b>Use Case Name</b>	View Inactive User Information
<b>Brief Description</b>	Admin chooses to view the detailed saved records of an inactive user account registered in the project mobile application server.

<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects one of the inactive user account tiles in the list of inactive user page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered inactive user accounts in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects a specific inactive user account from the list of inactive user page to navigate to the inactive user account details page.</li> <li>2. The system retrieves the inactive user account record saved in the inactive user collection of project mobile application server.</li> <li>3. The system displays the inactive user account information, and the date of account status became inactive to the admin.</li> </ol>
<b>Sub Flows</b>	<p>3a. Admin can choose to view the list of history sessions at the list of history session page involving the inactive user by selecting either the user request counter or the helper duty counter.</p> <p>3b. Admin can choose to toggle the inactive user account's active status at the top right corner of the inactive user account details page.</p> <p>3c. Admin can choose to contact inactive user through either direct phone call or email communication through selecting the icon buttons provided.</p>
<b>Alternative Flows</b>	-

Table 3.2.2.16 View List of History Records Use Case Description

<b>Use Case ID</b>	000016
<b>Use Case Name</b>	View List of History Records
<b>Brief Description</b>	Admin chooses to view the list of history records of the user, helper, or inactive user on the usage of the mobile application done.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin selects one of the usage counters at the user, helper, or inactive user account page.

<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user, helper, or inactive user accounts in the project mobile application server.</li> <li>- There are history records of sessions involving the user, helper or inactive user saved in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the user request counter at the user, helper, or inactive user account page to view the list of history record sessions involving the account holder as the requester for help.</li> <li>2. The system retrieves the list of history sessions that have the account holder as the user who requested help in the session.</li> <li>3. The system displays the list of history sessions with the summarized details to the admin.</li> <li>4. Admin can choose to open any of the history sessions to view the conversation recorded in the history sessions by tapping on the history session tiles.</li> </ol>
<b>Sub Flows</b>	4a. The admin can choose to delete the history session selected to view the conversation recorded by selecting on the delete icon button at the top right corner of the page.
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>1a. Admin selects the helper request counter instead to view on the history record involving the account holder as the helper.</li> <li>2a. There are no applicable history sessions found at the project mobile application server, error message is shown to admin after admin selects one of the usage counters to notify admin that there are no applicable history sessions found.</li> <li>4a. If there are no conversation found, an error message is shown to the admin to notify the history session has no chat conversation.</li> </ol>

Table 3.2.2.17 View App Data Analysis Use Case Description

<b>Use Case ID</b>	000017
<b>Use Case Name</b>	View App Data Analysis
<b>Brief Description</b>	The system displays out the summary of the application data analysis such as the number of new users registered, application usage count



	by users of the application, and the overall session duration based on current month to the admin at the admin home page to give out the summarized data analysis for the current month.
<b>Actor</b>	Admin
<b>Trigger</b>	When the admin enters the admin home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user, helper, or inactive user accounts in the project mobile application server.</li> <li>- There are history records of sessions saved in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin navigates to the admin home page after login into the project mobile application or any operation done.</li> <li>2. The system retrieves the required data from the project mobile application server to generate the summary of the application data analysis.</li> <li>3. The system displays out the summarized data to the admin.</li> <li>4. The admin can choose to tap on the displayed data column to be navigated to view the detailed application data analysis.</li> </ol>
<b>Sub Flows</b>	<p>4a. Admin taps on the number of new registered user data column, system directs the admin to the view new registered user chart page.</p> <p>4b. Admin taps on the application usage count data column, system directs the admin to the view application usage count chart page.</p> <p>4c. Admin taps on the overall session duration data column, system directs the admin to the view overall session duration chart page.</p>
<b>Alternative Flows</b>	-

Table 3.2.2.18 View Number of New Users Use Case Description

<b>Use Case ID</b>	000018
<b>Use Case Name</b>	View Number of New Users
<b>Brief Description</b>	Admin chooses to view more detailed data on the number of new registered users of the project mobile application in the form of bar chart.

<b>Actor</b>	Admin
<b>Trigger</b>	When admin taps on the number of new registered user data column at the admin home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user or helper accounts in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the number of new registered user data column to enter the view number of new registered user chart page.</li> <li>2. The system retrieves the required data from the project mobile application server and categorise the data based on recent three months.</li> <li>3. The system displays out the finalized bar chart to the admin showing the number of new registered users for the recent three months.</li> <li>4. Admin can choose to tap on the bars to enable the showing of the detail value of the bars in the bar chart.</li> <li>5. Admin can choose a different view on the bar chart from the list of option provided in the drop-down menu button at the top right corner.</li> </ol>
<b>Sub Flows</b>	<p>5a. Admin selects the “By Years” option at the drop-down menu button, the system displays the number of new registered user for the recent three years instead.</p> <p>5b. Admin selects the “Expand This Year” option at the drop-down menu button, the system displays the number of new registered user for this year categorized by months instead.</p> <p>5c. Admin selects the “Expand View” or “Minimize View” option at the drop-down menu button, the system expands or minimize the view of the current displaying bar chart instead.</p>
<b>Alternative Flows</b>	-

Table 3.2.2.19 View Overall Session Duration Use Case Description

<b>Use Case ID</b>	000019
<b>Use Case Name</b>	View Overall Session Duration
<b>Brief Description</b>	Admin chooses to view more detailed data on the overall session duration of the project mobile application in the form of line chart.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin taps on the overall session duration data column at the admin home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are history session records in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the overall session duration data column to enter the view overall session duration chart page.</li> <li>2. The system retrieves the required data from the project mobile application server and categorise the data based on recent three months.</li> <li>3. The system displays out the finalized line chart to the admin showing the overall session duration for the recent three months.</li> <li>4. Admin can choose to long press on the chart to enable the trackball to show the data values at the value points on the line in the line chart.</li> <li>5. Admin can choose a different view on the line chart from the list of option provided in the drop-down menu button at the top right corner.</li> </ol>
<b>Sub Flows</b>	<p>4a. Admin long presses the value points on the line chart to show a pop-up dialog displaying the overall duration, longest duration, shortest duration at the value point, and admin can choose to view on the involved sessions for the longest duration and shortest duration.</p> <p>5a. Admin selects the “By Years” option at the drop-down menu button, the system displays the overall session duration for the recent three years instead.</p>

	<p>5b. Admin selects the “Expand This Year” option at the drop-down menu button, the system displays the overall session duration for this year categorized by months instead.</p> <p>5c. Admin selects the “Expand View” or “Minimize View” option at the drop-down menu button, the system expands or minimize the view of the current displaying line chart instead.</p>
<b>Alternative Flows</b>	4a. Admin presses the cancel button to close the dialog instead.

Table 3.2.2.20 View App Usage Count Description

<b>Use Case ID</b>	000020
<b>Use Case Name</b>	View App Usage Count
<b>Brief Description</b>	Admin chooses to view more detailed data on the application usage count data of the project mobile application in the form of line chart.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin taps on the application usage count data column at the admin home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are history session records in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the application usage count data column to enter the view application usage count chart page.</li> <li>2. The system retrieves the required data from the project mobile application server and categorise the data based on recent three months.</li> <li>3. The system displays out the finalized line chart to the admin showing the application usage count for the recent three months.</li> <li>4. Admin can choose to long press on the chart to enable the trackball to show the data values at the value points on the line in the line chart.</li> <li>5. Admin can choose a different view on the line chart from the list of option provided in the drop-down menu button at the top right corner.</li> </ol>

<b>Sub Flows</b>	<p>5a. Admin selects the “By Years” option at the drop-down menu button, the system displays the application usage count for the recent three years instead.</p> <p>5b. Admin selects the “Expand This Year” option at the drop-down menu button, the system displays the application usage count for this year categorized by months instead.</p> <p>5c. Admin selects the “Expand View” or “Minimize View” option at the drop-down menu button, the system expands or minimize the view of the current displaying line chart instead.</p>
<b>Alternative Flows</b>	-

Table 3.2.2.21 Search Email Account Description

<b>Use Case ID</b>	000021
<b>Use Case Name</b>	Search Email Account
<b>Brief Description</b>	Admin chooses to search for an email account from the project mobile application to view on the email account information.
<b>Actor</b>	Admin
<b>Trigger</b>	When admin taps on the search bar provided at the admin home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- Admin has login into the mobile application with his or her account and verified identity with admin password.</li> <li>- There are registered user accounts in the project mobile application server.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Admin selects the search bar at the admin home page to navigate to the search delegate page.</li> <li>2. The system retrieves the list of email accounts from the project mobile application server and display out some recommendation email.</li> <li>3. Admin inputs some character into the search field provided and the system displays the email accounts that has the similar characters.</li> <li>4. Admin selects one of the search results in the search page to direct to the selected email account details page.</li> </ol>

<b>Sub Flows</b>	2a. Admin selects one of the email recommendations to direct to the selected email account details page.
<b>Alternative Flows</b>	4a. Admin clears the search input and close the search page instead.

Table 3.2.2.22 Request Help Use Case Description

<b>Use Case ID</b>	000022
<b>Use Case Name</b>	Request Help
<b>Brief Description</b>	User can request help from a random available helper from the system in the mobile application.
<b>Actor</b>	User
<b>Trigger</b>	When user press the “Request Help” button in the mobile application.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User needs to have an active account and logged into the mobile application.</li> <li>- User is currently not in an active session with a helper.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User selects the “Request Help” button on the user home page to request help from a helper.</li> <li>2. The system finds available helpers in the server and randomly assign one of them to the user.</li> <li>3. The system creates a chatroom session and generates a helper QR.</li> <li>4. The system notifies the helper that there is a client waiting through notification and pop-up dialog message.</li> <li>5. The user is then directed to the chat page linking to the same chatroom as the helper to start communicating with the helper.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>1a. If there is no internet connection, the user is directed to an offline page that still allows user to ask for help through contacting UTAR either via SMS messaging or direct phone call.</li> <li>2a. If there are no available helpers, the system notifies the user that the request has failed through error message in the mobile application.</li> </ol>

Table 3.2.2.23 End Session Use Case Description

<b>Use Case ID</b>	000023
<b>Use Case Name</b>	End Session
<b>Brief Description</b>	User or helper can end the session after the user have gained enough help.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When the user or helper selects the “End Session” option.
<b>Precondition</b>	User or helper needs to be in the active session in the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. When the user has gained the help, he or she needs, the user selects the “End Session” option.</li> <li>2. A pop-up dialog message is shown to the user to confirm his or her decision.</li> <li>3. A rating dialog is then showed to the user to give rating to the helper in the session.</li> <li>4. After giving the rating, the user is directed back to the user home page.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>1a. The helper decides to end the session on his or her side instead, no ratings are given.</li> <li>2a. The user chooses to cancel the decision, the session continues.</li> <li>3a. The user chooses not to give rating to the helper in the session, the session is ended, and no rating is given.</li> </ol>

Table 3.2.2.24 Rate Session Use Case Description

<b>Use Case ID</b>	000024
<b>Use Case Name</b>	Rate Session
<b>Brief Description</b>	User can give rating on how good the help is received from the helper in the session.
<b>Actor</b>	User

<b>Trigger</b>	When the user confirms to end the current active session at the pop-up dialog.
<b>Precondition</b>	User is currently in an active session.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. When user ends a session, a rating pop-up dialog is showed to user to ask for a rating.</li> <li>2. User can give a rating of one to five stars based on the help received from the helper in the session.</li> <li>3. The system saves the rating in the server accordingly in the history and rating collection.</li> <li>4. The user is then directed to the user home page.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	2a. The user chooses to not give rating to the session and the session is ended, which then the user is directed back to the user home page.

Table 3.2.2.25 Contact UTAR Use Case Description

<b>Use Case ID</b>	000025
<b>Use Case Name</b>	Contact UTAR
<b>Brief Description</b>	User and helper can contact UTAR for help if required.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When the user or helper selects the “Contact UTAR” or “Contact App Help Centre” options in settings page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper has mobile connection to make communications.</li> <li>- User or helper has internet access on his or her mobile device.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper goes to the settings page and select either “Contact UTAR” via email, SMS, or direct phone call.</li> <li>2. The user or helper is directed to the default mobile SMS application, mobile default email application or mobile phone call dial screen to proceed make communication to UTAR.</li> <li>3. The user or helper can then return to the home page.</li> </ol>
<b>Sub Flows</b>	1a. The permission to access to mobile phone call and SMS service is asked before proceeding to use the direct phone call or SMS service.



<b>Alternative Flows</b>	<p>1a. The user or helper is in an offline situation and thus is directed to the offline page to suggest contact UTAR via SMS communication or direct phone call for help.</p> <p>2a. The user does not give permission to the application to access mobile's phone call services or SMS service and thus cannot be directed to mobile phone call dial screen or mobile SMS application.</p>
--------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 3.2.2.26 Online Chat Communication Use Case Description

<b>Use Case ID</b>	000026
<b>Use Case Name</b>	Online Chat Communication
<b>Brief Description</b>	User and helper can communicate with each other through online chat communication for user to ask for help and helper to response user's request.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When user request help from helper, and both are in the same session.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User and helper are logged into the mobile application.</li> <li>- User has been assigned to a helper in the system.</li> <li>- User and helper are currently in an active session.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User requests help in the application and joins an active session with the assigned helper.</li> <li>2. A chatroom session is created in the system to save information of the user and helper, while also record down the chats.</li> <li>3. Chat page is opened for user and helper to join the chatroom.</li> <li>4. User and helper can communicate with each other for user to ask for help, while helper providing help to user.</li> </ol>
<b>Sub Flows</b>	<p>4a. User and helper can send images from gallery or camera by selecting the add icon button at the bottom right of the chat page to upload image from mobile device.</p> <p>4b. User can send text messages in the chat by typing in the text field provided and select the send button at the right side of the text field.</p> <p>4c. User and helper can save images from the chat to the mobile device by long pressing on the image in full view.</p>

	4d. User and helper can copy the text messages in the chat by long pressing the chat tile.
<b>Alternative Flows</b>	4a1. If user or helper does not permit the permission to use camera or phone storage, user or helper is unable to communicate using images. 4c1. If user or helper does not permit the access to phone storage, image cannot be saved to mobile device.

Table 3.2.2.27 Online Call Communication Use Case Description

<b>Use Case ID</b>	000027
<b>Use Case Name</b>	Online Call Communication
<b>Brief Description</b>	User and helper can communicate with each other through online voice call communication for real-time interaction between user and helper.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When user or helper selects the “Call” button at the chat page during an active session.
<b>Precondition</b>	- User and helper are logged into the mobile application. - User has been assigned to a helper in the system. - User and helper are currently in an active session
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User requests help in the application and joins an active session with the assigned helper.</li> <li>2. User or helper selects the “Call” icon button to start an online voice call.</li> <li>3. The system enables the user or helper to join the online call channel and directs the user or helper to the call page.</li> <li>4. The system sends a call notification to the other side that shows the incoming call page for the other user or helper to join in the call channel.</li> <li>5. User or helper ends the online call by pressing the “End Call” button after he or she has finished talking to the other user or helper.</li> </ol>
<b>Sub Flows</b>	2a. The other side user or helper starts the online call for the user or helper to join in instead.

<b>Alternative Flows</b>	<p>4a. The other side user or helper refuses to join the online voice call, no real-time interaction can be established.</p> <p>5a. If one of the user or helper side has ended the call after five minutes, the system automatically ends the online call instead.</p>
--------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 3.2.2.28 Change Account Type Use Case Description

<b>Use Case ID</b>	00028
<b>Use Case Name</b>	Change Account Type
<b>Brief Description</b>	User and helper can choose to change account type based on their preferences.
<b>Actor</b>	User, Helper
<b>Trigger</b>	When user or helper selects the toggle switch to change account type.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper is logged into the mobile application.</li> <li>- User or helper is not in an active session.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper directs to the navigation drawer of the application.</li> <li>2. User or helper chooses to change their account type based on their preferences.</li> <li>3. The system saves the changes made by user or helper.</li> </ol>
<b>Sub Flows</b>	<p>2a. If “Helper” account type is chosen, the specific user can modify helper status and accept requests.</p> <p>2b. If “User” account type is chosen, the specific user can request help from other helpers, while the helper’s information is hidden if have.</p>
<b>Alternative Flows</b>	2c. If the helper or user tries to change the account type while in a session, an error pop-up dialog is shown to notify the invalid action.

Table 3.2.2.29 Modify Helper Status Use Case Description

<b>Use Case ID</b>	00029
<b>Use Case Name</b>	Modify Helper Status
<b>Brief Description</b>	Helper can edit their helper status and add helper duty period if required.
<b>Actor</b>	Helper

<b>Trigger</b>	When helper toggle the helper status switch, edit or remove helper duty period.
<b>Precondition</b>	Helper is logged into the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Helper goes to the helper details page to view helper information.</li> <li>2. Helper can turn off or on the helper status to decide whether to be open for requests or not.</li> <li>3. Helper can add in or remove helper duty period to toggle on or off the helper status when the helper duty starts or ends.</li> </ol>
<b>Sub Flows</b>	<p>1a. If helper is not in an active session, the helper can view and modify helper information at the home page.</p> <p>2a. If helper has helper duty period, a pop-up dialog message is shown to double confirm with helper before removing helper duty period.</p> <p>3a. Helper can edit the helper period for the start and end time individually.</p>
<b>Alternative Flows</b>	2b. If a helper refuses to remove helper duty period, the helper status is remained unchanged.

Table 3.2.2.30 QR Scanning Use Case Description

<b>Use Case ID</b>	00030
<b>Use Case Name</b>	QR Scanning
<b>Brief Description</b>	User can use the QR scanner to identify assigned helper if physical meeting arranged between user and helper.
<b>Actor</b>	User
<b>Trigger</b>	When user opens the QR scanner at the home page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User is logged into the mobile application.</li> <li>- User is in an active session with an assigned helper.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User goes to the home page to open the mobile camera for QR scanning.</li> <li>2. The system obtains the helper QR from the system server as the QR code value to be scanned by the user.</li> </ol>

	3. When a valid helper QR is scanned, a success message is displayed over the QR scanner.
<b>Sub Flows</b>	1a. Before getting access to the QR scanner, the user must allow the permission for the application to gain access to mobile device camera.
<b>Alternative Flows</b>	1b. If no permission is allowed to use the mobile device camera, the QR scanner cannot be opened. 3a. If an invalid QR is scanned, an error message is shown to the user to notify that it is not the assigned helper QR.

Table 3.2.2.31 QR Code Generator Use Case Description

<b>Use Case ID</b>	00031
<b>Use Case Name</b>	QR Code Generator
<b>Brief Description</b>	Helper can get an QR code from the system to display for their assigned client user to scan to verify his or her identity.
<b>Actor</b>	Helper
<b>Trigger</b>	When helper tap on the icon button to show QR.
<b>Precondition</b>	- Helper is logged into the mobile application. - Helper is in an active session with an assigned client user.
<b>Normal Flow of Events</b>	1. Helper goes to the helper home page to show the QR code. 2. Helper selects the “Show QR” icon button to show QR. 3. The system retrieves the created helper QR from the server to return to the helper. 4. The QR code is then deleted after the active session is ended.
<b>Sub Flows</b>	2a. The helper can choose to show or hide the QR code through tapping on the QR icon button.
<b>Alternative Flows</b>	-

Table 3.2.2.32 Edit Profile Use Case Description

<b>Use Case ID</b>	00032
<b>Use Case Name</b>	Edit Profile

<b>Brief Description</b>	User, helper, or admin can edit the personal information at the respective edit profile page.
<b>Actor</b>	User, Helper, Admin
<b>Trigger</b>	When the user, helper, or admin selects the “Edit Profile” option at the account details page.
<b>Precondition</b>	User, helper, or admin is logged into the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User, helper, or admin selects the “Edit Profile” icon button at the respective profile page.</li> <li>2. User, helper, or admin is directed to the edit profile page that has the initial values of the personal information saved in the system.</li> <li>3. User, helper, or admin can choose to edit the information he or she wants to save to his or her account in the system.</li> <li>4. When edit complete, the user, helper or admin can select the “Save” icon button to save the changes made.</li> </ol>
<b>Sub Flows</b>	<p>3a. If the user, helper, or admin decided to revert the values back to the original ones before saving, he or she selects the “Revert Changes” icon at the edit profile changes to return the original value previously saved.</p> <p>4a. If admin edits the email account value, the system will pop out a confirmation dialog for the admin to enter the admin password to proceed updating the email account.</p>
<b>Alternative Flows</b>	<p>1a. User or helper selects the “Edit Profile” option at the navigation drawer instead.</p> <p>4a. User, helper or admin decided not to save the changes made and pop back out of the edit profile page, no changes is done.</p>

Table 3.2.2.33 Delete Account Use Case Description

<b>Use Case ID</b>	00033
<b>Use Case Name</b>	Delete Account
<b>Brief Description</b>	User or helper wants to remove the account registered in the system and clear personal information saved in the system server.
<b>Actor</b>	User, Helper

<b>Trigger</b>	User or helper selects the “Delete Account” option in the settings page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper is logged into the mobile application.</li> <li>- User or helper is not in an active session.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper selects the “Delete Account” option in settings page.</li> <li>2. A pop-up confirmation dialog is shown to the user or helper to confirm their decision.</li> <li>3. The system removes the user account and personal information from the server.</li> <li>4. The user or helper is then directed back to the welcome page of the mobile application.</li> </ol>
<b>Sub Flows</b>	3a. The system saves the account important personal information in the inactive user collection in project mobile application server in case of emergencies requiring that piece of information.
<b>Alternative Flows</b>	2a. The user or helper cancels the decision to delete account, account and personal information remains.

Table 3.2.2.34 Create Session Use Case Description

<b>Use Case ID</b>	00034
<b>Use Case Name</b>	Create Session
<b>Brief Description</b>	User requests help in the mobile application to create an active session with a random assigned available helper.
<b>Actor</b>	User
<b>Trigger</b>	User selects the “Request Help” button at the user home page.
<b>Precondition</b>	User is logged into the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User selects the “Request Help” button at the user home page.</li> <li>2. The system creates session chatroom for the user and helper after the user requests for help.</li> <li>3. The session records the information of the user and helper, along with the chatroom information such as the chatroom id and date.</li> </ol>

	4. The user is directed to the created chatroom session to start communicating with the helper.
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	2a. There is no available helper to be assigned to the user, no session is created.

Table 3.2.2.35 SMS Communication Use Case Description

<b>Use Case ID</b>	00035
<b>Use Case Name</b>	SMS Communication
<b>Brief Description</b>	User or helper wants to communicate with UTAR or app help centre through SMS communication when offline.
<b>Actor</b>	User, Helper
<b>Trigger</b>	User or helper selects the “Message” button on the offline page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper has no internet connection to use the functionalities of the application.</li> <li>- User or helper has mobile service to use SMS communication.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper is directed to the offline page due to no internet connection to use the functions in the application.</li> <li>2. User or helper selects the “Message” button on the offline page.</li> <li>3. A permission dialog to use mobile SMS service is shown to the user or helper to ask for permission.</li> <li>4. The user or helper is directed to the mobile default messaging platform to proceed contacting with UTAR.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	3a. If the permission to use the mobile SMS service is not permitted, the user or helper cannot proceed to SMS contact UTAR.

Table 3.2.2.36 Phone Call Communication Use Case Description

<b>Use Case ID</b>	00036
<b>Use Case Name</b>	Phone Call Communication



<b>Brief Description</b>	User or helper wants to communicate with UTAR or app help centre through direct phone call communication when offline.
<b>Actor</b>	User, Helper
<b>Trigger</b>	User or helper selects the “Call” button on the offline page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User has no internet connection to use the functionalities of the application.</li> <li>- User or helper has mobile service to use the phone call communication.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper is directed to the offline page due to no internet connection to use the functions in the application.</li> <li>2. User or helper selects the “Call” button on the offline page.</li> <li>3. A permission dialog to use mobile phone call service is shown to the user or helper to ask for permission.</li> <li>4. The user or helper is directed to the mobile default phone dial screen to proceed contacting with UTAR.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	3a. If the permission to use the mobile phone call service is not permitted, the user cannot proceed to direct phone call to contact UTAR.

Table 3.2.2.37 Email Communication Use Case Description

<b>Use Case ID</b>	00037
<b>Use Case Name</b>	Email Communication
<b>Brief Description</b>	User or helper wants to communicate with UTAR or the app help centre through email communication in the settings page.
<b>Actor</b>	User, Helper
<b>Trigger</b>	User or helper selects the email icon of the UTAR or the app help centre in the settings page.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>- User or helper is logged into the mobile application.</li> <li>- User or helper has a valid email account.</li> </ul>
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. User or helper enters the settings page of the mobile application.</li> </ol>

	<ol style="list-style-type: none"> <li>2. User or helper selects to communicate UTAR or app help centre through email communication.</li> <li>3. User or helper is directed to the mobile default email communication platform to proceed contact with UTAR or app help centre.</li> </ol>
<b>Sub Flows</b>	-
<b>Alternative Flows</b>	-

Table 3.2.2.38 Toggle Helper Status Use Case Description

<b>Use Case ID</b>	00038
<b>Use Case Name</b>	Toggle Helper Status
<b>Brief Description</b>	Helper wants to modify the helper status in the mobile application to decide whether to be open for requests or not.
<b>Actor</b>	Helper
<b>Trigger</b>	Helper toggles the helper status switch in helper home page or helper details page.
<b>Precondition</b>	Helper is logged into the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Helper goes to the helper home page to view the helper status.</li> <li>2. Helper toggles the helper status switch on the helper home page.</li> <li>3. The system saves the helper's updated status in the system server to allow the settings of being open or closed for requests.</li> </ol>
<b>Sub Flows</b>	2a. If the helper duty period of helper is not null, a confirmation dialog is shown to get helper confirmation to proceed modify helper status and remove helper duty period.
<b>Alternative Flows</b>	<ol style="list-style-type: none"> <li>1a. Helper goes to the helper details page to view the helper status.</li> <li>2a1. If helper cancels the decision to modify helper status, the helper status remains unchanged, and the helper duty period is not removed.</li> </ol>

Table 3.2.2.39 Set Helper Duty Period Use Case Description

<b>Use Case ID</b>	00039
<b>Use Case Name</b>	Set Helper Duty Period
<b>Brief Description</b>	Helper wants to set or edit the helper duty period to set when the helper is open for requests.
<b>Actor</b>	Helper
<b>Trigger</b>	Helper taps on the empty border to add helper duty period or select the start time or end time to edit the helper duty period.
<b>Precondition</b>	Helper is logged into the mobile application.
<b>Normal Flow of Events</b>	<ol style="list-style-type: none"> <li>1. Helper goes to the helper home page to view the helper duty period.</li> <li>2. Helper taps on the empty border to add in helper duty period.</li> <li>3. The system saves the modification done to the helper duty period in the helper account.</li> <li>4. The mobile application updates the helper status based on the helper duty period set.</li> </ol>
<b>Sub Flows</b>	2a. A time range picker dialog is shown to the helper to set the helper duty time duration wanted.
<b>Alternative Flows</b>	<p>2b. Helper selects the start time or end time of the helper duty period to edit the time.</p> <p>2a1. If an invalid time duration is set, an error message is shown to the helper on the invalid time duration set until the user sets a valid time duration.</p> <p>2b1. If invalid duration is found between the start and end time set for the helper duty period, an error message is shown to the helper on the invalid time set, no update is done.</p>

### 3.3 Project Development

#### 3.3.1 Firebase Setup and Connect to Project

Before starting the development of the project, the project is first connected to a server for backend services to support the project mobile application in data handling. Here the Firebase is used as the project mobile application's backend server services due to its compatibility with Flutter project and good and easy-to-use backend services

provided for developers. First, the SHA-1 and SHA-256 fingerprints are obtained from the project to be used to register the project in the Firebase console as shown in Figure 3.3.1.1. Then, the dependencies of Firebase libraries are added to the app level Build Gradle file for enabling the use of Firebase in the project mobile application, which is showed in Figure 3.3.1.2. After registering the project to the Firebase, a google-service JSON file is then generated and is added to the Android app level directory as it is the verification key for the project mobile application to use the Firebase services after registration. The main Firebase services used for the project are Firebase Authentication, Firebase Firestore, and Firebase Storage. For the project mobile application to start using the Firebase services, `WidgetsFlutterBinding.ensureInitialized` and `Firebase.initializeApp` functions as shown in Figure 3.3.1.3 must be called at the main file of the project mobile application to ensure Firebase services are initialized and started to use the Firebase services.

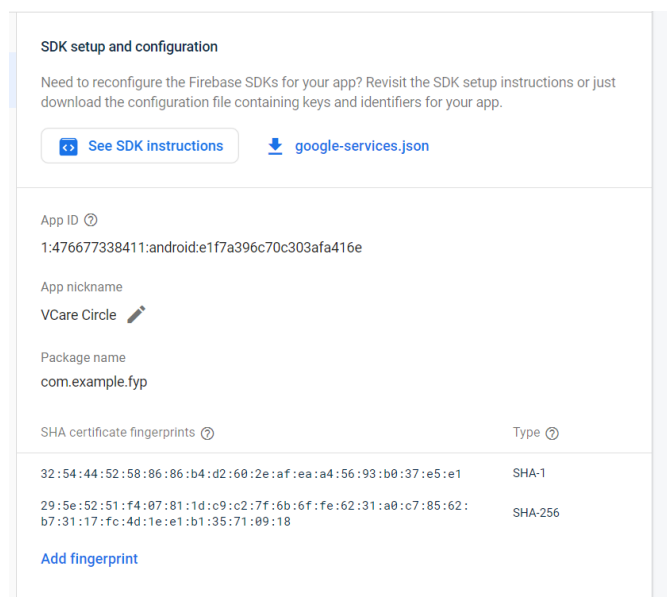


Figure 3.3.1.1 Registering of Project in Firebase Console

```
dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation platform('com.google.firebase:firebase-bom:30.4.1')
    implementation 'com.google.firebase:firebase-analytics'
    implementation 'com.android.support:multidex:1.0.3'
    implementation 'com.google.firebase:firebase-auth'
    // Also declare the dependency for the Google Play services library and specify its version
    implementation 'com.google.android.gms:play-services-auth:20.2.0'
    //Notification
    implementation 'com.google.firebase:firebase-inappmessaging-display:19.1.5'
    implementation 'com.android.support:multidex:1.0.3'
    //Try to add okio library
    implementation("com.squareup.okhttp3:okhttp:4.9.3")
    //Required
    coreLibraryDesugaring 'com.android.tools:desugar_jdk_libs:1.1.5'
}
```

Figure 3.3.1.2 Firebase Dependencies in App Level Build Gradle File

```

86 //Ensure initialize before start app
87 late WidgetsBinding widgetsBinding;
Run | Debug | Profile
88 void main() async {
89   widgetsBinding = WidgetsFlutterBinding.ensureInitialized();
90
91   //Native splash
92   FlutterNativeSplash.preserve(widgetsBinding: widgetsBinding);
93   ///Ensure to start firebase before all operations
94   await Firebase.initializeApp(
95     | options: DefaultFirebaseOptions.currentPlatform,
96   );
97   //notification
98   await firebaseMessaging.getInitialMessage().then((value) {
99     if (value != null) {
100       debugPrint("Initialize message${value.data}");
101     } else {
102       debugPrint("Value is null");
103     }
104   });
105   FirebaseMessaging.onBackgroundMessage(backgroundMessagesHandler);
106   //Set screen orientation to potrait
107   SystemChrome.setPreferredOrientations(
108     [DeviceOrientation.portraitUp, DeviceOrientation.portraitDown]);
109   runApp(const MyApp());
110 }

```

Figure 3.3.1.3 Initialize Firebase Services in main.dart

### 3.3.2 Register Account Function Development

The purpose of the register account function is to collect user's information and account details to be saved in the Firebase server. Dart language is used as the main language for functionality module development and Flutter as the main framework tool to build the user interfaces in the project mobile application. First, a user class is created together with a getter and setter to perform saving and retrieving of user information in the form of map data, as this format is preferred to conduct data exchange between project and Firebase server.

For the register account function, two forms are created for the register account function, which one of it records down the user's personal information, and the other records down the user's account and setting of account password. The forms are added with validation keys with the use of TextEditingController for the purpose of ensuring the input values for each field in the form are valid and not empty, such as for the email account, username, and password. In the first form of register account function, the form validator ensures that no values are empty and are in valid format for the user's information such as username and phone number before proceeding to the next page for adding email account details, which is shown in Figure 3.3.2.1, Figure 3.3.2.2, and Figure 3.3.2.3.

Next, the user is directed to the second page to enter the email account to be registered into the system and set the password according to the format defined. When complete, the user can then register the account by pressing on the register button. Here the signUp function in Figure 3.3.2.4 is called to register the new account into the Firebase server, while also creating a new document saving user information through the uploadUserDetails function in Figure 3.3.2.5. Here the function also checks if there are previous records saved in the inactive user collection in server and removes it. When the account is registered, the user is directed to the verification email page to verify the email account registered to continue using the project mobile application through the function checkVerification and sendVerification in Figure 3.3.2.6 and 3.3.2.7. On the other hand, if the sign up of account is unsuccessful, an error message is shown to the user.

```

title: TextFormField(
  keyboardType: TextInputType.name,
  autocorrect: false,
  controller: nameController,
  textAlign: TextAlign.center,
  style: const TextStyle(
    fontSize: 16,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 5, 72, 85),
  ),
  cursorColor:
    const Color.fromARGB(255, 5, 89, 106),
  textInputAction: TextInputAction.next,
  validator: (input) {
    if (input!.isEmpty) {
      return "Please Enter Your Name";
    } else if (!input
      .contains(RegExp(r'[A-Z|a-z]')) {
      return "Please only Enter Alphabetic Characters";
    }
    return null;
  },
),

```

Figure 3.3.2.1 Example of TextEditingController Validator for Username

```

title: TextFormField(
  keyboardType: TextInputType.phone,
  autocorrect: false,
  controller: phoneController,
  textAlign: TextAlign.center,
  style: const TextStyle(
    fontSize: 14,
    fontWeight: FontWeight.bold,
    color: Color.fromARGB(255, 5, 72, 85),
  ), // TextStyle
  cursorColor:
    const Color.fromARGB(255, 5, 89, 106),
  textInputAction: TextInputAction.next,
  validator: (input) {
    if (input!.isEmpty) {
      return "Please Enter Phone Number";
    } else if (!input.contains(RegExp(
      r'^(\+?6?01)[02-46-9][0-9]{7}$|^(+?6?01)[1][-[0-9]{8}$')) { // RegExp
      return "Please enter the correct format";
    }
    return null;
  },
),

```

Figure 3.3.2.2 TextEditingController Validator for User Phone Number

```

//Go to next page button
floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
floatingActionButton: showKeyboard
? null
: SizedBox(
  width: MediaQuery.of(context).size.width * 0.5,
  child: FloatingActionButton.extended(
    heroTag: null,
    onPressed: () {
      if (signUpkey.currentState!.validate()) {
        if (newUser.name != null &&
            newUser.gender != null &&
            newUser.phone != null) {
          Navigator.of(context).push(MaterialPageRoute(
            builder: (context) => SignUpPage2(
              getUserInfo: newUser,
            ));
        } else {
          ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
            content: Text(
              "Please fill up all required information!"));
        } else {
          return;
        }
      },
    ),
  ),

```

Figure 3.3.2.3 Validation Check for User Personal Information in sign\_up\_page1.dart

```

329 //After Getting the User Information, save to Firebase Server
330 Future signUp(String acc, String pass, UserN getUserInfo) async {
331   showDialog(
332     context: context,
333     barrierDismissible: false,
334     builder: ((context) => loadingWait());
335   if (signUpkey2.currentState!.validate()) {
336     try {
337       //when successful create account, the user is automatically signed in
338       await FirebaseAuth.instance
339         .createUserWithEmailAndPassword(email: acc, password: pass)
340         .then((value) {
341           uploadUserDetails(getUserInfo);
342           if (!mounted) return;
343           ScaffoldMessenger.of(context)
344             .showSnackBar(const SnackBar(content: Text("User is Created!")));
345           Navigator.of(context).pop();
346         });
347     } on FirebaseAuthException catch (e) {
348       String errorMessage = "";
349       if (e.code == "email-already-in-use") {
350         errorMessage = "Account is already been registered in the server!";
351       } else if (e.code == "invalid-email") {
352         errorMessage = "Invalid Email Account!";
353       } else {
354         errorMessage =
355           "Unable to create account, contact with admin for support!";
356       }
357       ScaffoldMessenger.of(context)
358         .showSnackBar(SnackBar(content: Text(errorMessage)));
359       Navigator.of(context).pop();
360     }
361   } else {
362     Navigator.of(context).pop();
363     return;
364   }

```

Figure 3.3.2.4 signUp Function in sign\_up\_page2.dart

```

370 //Upload user info to firebase firestore
371 void uploadUserDetails(UserN userInfoUpload) async {
372   /**At start set the user account to be as false for admin value
373   userInfoUpload.admin = false;
374   userInfoUpload.joinDate = DateHandler().setNewUserJoinDate();
375   await UserOld().checkBackOldHistory(userInfoUpload.email!);
376   FirebaseFirestore firestore = FirebaseFirestore.instance;
377   await firestore
378     .collection("user")
379     .doc(userInfoUpload.email)
380     .set(userInfoUpload.toMap())
381     .then((value) =>
382       PushNotification().saveTokenForMessaging(userInfoUpload.email!));
383   if (!mounted) return;
384
385   Navigator.pushAndRemoveUntil(
386     (context),
387     MaterialPageRoute(builder: (context) => const VerifyEmailPage()),
388     ModalRoute.withName(VerifyEmailPage.route));
389 }
390 }

```

Figure 3.3.2.5 uploadUserDetails Function in sign\_up\_page2.dart

```

195 //Check user email verification status
196 Future checkVerification() async {
197   try {
198     await FirebaseAuth.instance.currentUser!.reload();
199   } on FirebaseAuthException catch (error) {
200     ScaffoldMessenger.of(context)
201       .showSnackBar(SnackBar(content: Text(error.message!)));
202   }
203   if (!mounted) return;
204   setState(() {
205     emailVerified = FirebaseAuth.instance.currentUser!.emailVerified;
206   });
207   if (emailVerified) {
208     if (timer != null) {
209       timer!.cancel();
210     }
211   }
212 }
213 }
214 }

```

Figure 3.3.2.6 checkVerification Function in verify\_email\_page.dart

```

162 //Email verification link generator
163 Future sendVerification() async {
164   final currentUser = FirebaseAuth.instance.currentUser;
165   showDialog(
166     context: context,
167     barrierDismissible: false,
168     builder: ((context) => loadingWait());
169   if (!emailVerified) {
170     try {
171       await currentUser!.sendEmailVerification();
172       setState(() {
173         resendEmail = false;
174       });
175       await Future.delayed(const Duration(seconds: 5)).then((value) {
176         setState(() {
177           resendEmail = true;
178         });
179       });
180       if (!mounted) return;
181       ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
182         content: Text('An Email Verification Link Is Sent'))); // SnackBar
183       Navigator.of(context).pop();
184     } catch (error) {
185       ScaffoldMessenger.of(context)
186         .showSnackBar(SnackBar(content: Text(error.toString())));
187       Navigator.of(context).pop();
188     }
189   } else {
190     Navigator.of(context).pop();
191   }
192 }

```

Figure 3.3.2.7 sendVerification Function in verify\_email\_page.dart



### 3.3.3 Login Account Function Development

The login account function is for users to login their account registered in the project mobile application to proceed using the functionalities in the project mobile application. First, the login page allows the user to input their account and password, which here email and password TextEditingController validator are added to ensure valid email account is inputted and password is not null. Then the function of loginApp in Figure 3.3.3.1 is called to attempt login into the mobile application through the email and password typed by the user. At this point, the function also checks whether the current login email is the administrator email account, to determine the next page route for the user. The device messaging token is also obtained through the saveTokenForMessaging function which this token is saved to the user document in Firebase Firestore for receiving notification. The details of the saveTokenForMessaging function and sendNotification function can be seen in Figure 3.3.3.2 and Figure 3.3.3.3. When the user login is successful, a welcome notification message is shown to the user through the function of sendNotification using the device messaging token, and the searchUserStatus function shown in Figure 3.3.3.4 is called to check if the account status is active, to then direct user to the respective project mobile application's home page based on account type, else the user is directed to the inactive user page. On the other hand, if the login attempt is unsuccessful, exception handling is also added to the loginApp function to show error messages to the user. If the user's email account is yet to be verified in the system, the user is directed to the verify email page instead to continue verifying the email account before being directed to the project mobile application's homepage. For administrator account, after the successful login, the admin is required to verify the admin password at the verify admin password instead to verify his or her identity as the admin before entering the admin home page, which the function is shown in Figure 3.3.3.5.

```

299 //Login function
300 void loginApp(String acc, String pass, BuildContext context) async {
301   String adminEmail = await AdminControls().getAdminEmail();
302   showDialog(
303     context: context,
304     barrierDismissible: false,
305     builder: ((context) => loadingWait());
306   if (loginFormKey.currentState!.validate()) {
307     try {
308       await FirebaseAuth.instance
309         .signInWithEmailAndPassword(email: acc, password: pass);
310
311       //notification
312       await PushNotification().saveTokenForMessaging(acc).then((value) {
313         PushNotification().sendNotificationMessage(
314           FirebaseAuth.instance.currentUser!.email!, "Welcome");
315       });
316       if (!mounted) return;
317       ScaffoldMessenger.of(context)
318         .showSnackBar(const SnackBar(content: Text('Login Success!')));
319       loading(false);
320
321       //See if can hide the dialog or not
322       navigatorKey.currentState!.popUntil((route) => route.isFirst);
323       await Navigator.pushNamedAndRemoveUntil(
324         context,
325         AdminControls().checkAdmin(adminEmail, acc),
326         (Route<dynamic> route) => false);
327     } on SocketException {
328       loading(false);
329       if (!mounted) return;
330       ScaffoldMessenger.of(context).showSnackBar(
331         const SnackBar(content: Text('Internet Disconnected')));
332       Navigator.of(context).pop();
333     } on FirebaseAuthException catch (error) {
334       debugPrint(error.code);
335       loading(false);
336       if (!mounted) return;
337       Navigator.of(context).pop();
338       if (error.code == 'wrong-password') {
339         errorMessage = "Wrong Password!";
340       } else if (error.code == 'invalid-email') {
341         errorMessage = "Invalid Email!";
342       } else if (error.code == 'user-not-found') {
343         errorMessage =
344           'No account found for this app! Maybe try sign up an account!';
345       }
346       ScaffoldMessenger.of(context)
347         .showSnackBar(SnackBar(content: Text(errorMessage)));
348     }
349   } else {
350     if (!mounted) return;
351     Navigator.of(context).pop();
352     return;
353   }
354   FocusManager.instance.primaryFocus?.unfocus();
355 }

```

Figure 3.3.3.1 loginApp Function in login\_page.dart

```

8 class PushNotification {
9   //Get Token of device for messaging (Special case if user log in different device)
10  Future saveTokenForMessaging(String userEmail) async {
11    await firebaseMessaging.getToken().then((token) async {
12      debugPrint('token: $token');
13      await FirebaseFirestore.instance
14        .collection('user')
15        .doc(userEmail)
16        .update({'getMessageToken': token});
17    }).catchError((error) {
18      debugPrint("Unable to get Device Token: ${error.toString()}");
19    });
20  }
21
22  //get receiver token
23  Future<String?> retrieveUserMessagingToken(String userId) async {
24    var snapshot =
25      await FirebaseFirestore.instance.collection('user').doc(userId).get();
26    if (snapshot.exists && snapshot != null) {
27      return snapshot.data()!['getMessageToken'];
28    } else {
29      return null;
30    }
31  }
32 }

```

Figure 3.3.3.2 saveTokenForMessaging Function in push\_notification\_model.dart

```

33 //send notification to receiver
34 Future sendNotificationMessage(String receiverId, String message) async {
35   String? receiverToken = await retrieveUserMessagingToken(receiverId);
36   if (receiverToken != null) {
37     //Authorization is server key from project settings -> cloud messaging API **need add key=
38     try {
39       //Conduct cloud messaging via HTTP protocol
40       await http.post(Uri.parse('https://fcm.googleapis.com/fcm/send'),
41         headers: <String, String>{
42           'Content-Type': 'application/json',
43           'Authorization':
44             'key=AAAAbvuu9Ss:APA91bHApLv2MHERSghwDyGH5BKOAGCM2jHdk61ZuV6PRtA-RWtCt1GQuFV6n5Pfrxm7EugR9VzmOy1RE71WLJAL_
45         },
46         body: jsonEncode(
47           <String, dynamic>{
48             //if need action after notification click need adjust here
49             'priority': 'high',
50             'data': <String, dynamic>{
51               'click_action': 'FLUTTER_NOTIFICATION_CLICK',
52               'status': 'done',
53               'body': message,
54               'title': "VCare Circle",
55             },
56
57             //filling up notification details
58             'notification': <String, dynamic>{
59               'title': "VCare Circle",
60               'body': message,
61               'android_channel_id': "VCare Circle",
62             },
63             'to': receiverToken,
64           },
65         ));
66     } catch (e) {
67       if (kDebugMode) {
68         debugPrint(e.toString());
69       }
70     }
71   } else {
72     debugPrint("User $receiverId Token lost");
73   }
74 }
75 }
76

```

Figure 3.3.3.3 sendNotificationMessage Function in push\_notification\_model.dart

```

31 Future<bool> searchUserStatus() async {
32   setState(() {
33     loading = true;
34   });
35   var snapshot = await FirebaseFirestore.instance
36     .collection("inactiveUser")
37     .doc(FirebaseAuth.instance.currentUser!.email)
38     .get();
39
40   if (snapshot.exists && snapshot.data()!.isEmpty) {
41     setState(() {
42       loading = false;
43       isInActive = true;
44     });
45     return true;
46   } else {
47     setState(() {
48       loading = false;
49       isInActive = false;
50     });
51     return false;
52   }
53 }

```

Figure 3.3.3.4 searchUserStatus Function in checkUserStatus.dart

```

262         onPressed: () async {
263           //submit and login to admin home page
264           if (formKey.currentState!.validate()) {
265             debugPrint(passwordController.text);
266             debugPrint(adminPassword);
267             if (passwordController.text == adminPassword) {
268               SharedPreferences preferences =
269                 await SharedPreferences.getInstance();
270               await preferences.setBool("adminlogin", true);
271               if (!mounted) return;
272               Navigator.of(context).pushAndRemoveUntil(
273                 MaterialPageRoute(
274                   builder: (context) => const AdminHome(), // MaterialPageRoute
275                   ModalRoute.withName(AdminHome.route),
276                 );
277             } else {
278               if (!mounted) return;
279               ScaffoldMessenger.of(context).showSnackBar(
280                 const SnackBar(
281                   content: Text("Wrong Password!")); // SnackBar
282             }
283           }
284         },

```

Figure 3.3.3.5 Verify Admin Password Code Snippet in  
admin\_password\_veirfy\_screen.dart

### 3.3.4 Profile Module Development

The profile module is developed in the project to allow the application users to view and edit the account profile saved in the project mobile application server. In this module, the profile module consists of two type of profile management system, which is the system to handle the user or helper account profile, and the system to handle administrator profile.

In the profile management system for users and helpers' account, the setUserDetails function is used to retrieve the user or helper profile information from the Firebase to be displayed out to the user or helper when he or she enters the profile page, which this function is shown in Figure 3.3.4.1 and Figure 3.3.4.2. If the user or helper chooses to edit the profile information, the user or helper can change the changes made to be updated in the Firebase through calling the editComplete function by pressing the "Save Changes" button, which is shown in Figure 3.3.4.3. For the edit profile page, a form and TextEditingController for input text fields are used to validate the new value input of the user or helper before updating the profile in Firebase. As for helper account, the helper can also choose to toggle the helper status on or off through the toggle switch provided to call the function in Figure 3.3.4.4, Figure 3.3.4.5, and Figure 3.3.4.6, to update the helper status, which here the function checks if there is helper duty period set beforehand, a dialog is shown to helper to confirm if he or she wants to toggle helper status and remove helper duty period. Helper can also choose to set or update the helper duty period at the edit profile page using time range picker from Flutter to set the time duration and save the duty period to Firebase with the function code snippet shown in

Figure 3.3.4.8 and Figure 3.3.4.9. Error handling is added to the function to prevent error duration being saved into the Firebase server. Lastly, the user or helper can also choose to update the account type to either user or helper account by toggling the account type toggle switch to trigger the function of `changeAccountType` show in Figure 3.3.4.10 to save the changes made.

For the profile management used for administrator profile, the `setRequiredFunction` in Figure 3.3.4.11 is used to retrieve the admin profile information from the Firebase server for the project mobile application to display out admin information at admin profile page. For the editing of administrator profile, if the administrator has make changes in the administrator profile, the “Save Changes” button pressed after making changes will trigger the functions at the code snippet shown from Figure 3.3.4.12 to Figure 3.3.4.14, which the `updateAdminInfo` function in Figure 3.3.4.15 is called to update the administrator profile in Firebase server. Before saving the changes made, the email account value is checked if is modified. If no changes made, then the new profile information is updated through the function call of `updateAdminInfo` function. If modified, then the new email account is passed to the `checkifEmailInServer` function displayed in Figure 3.3.4.16 to ensure the email account is registered in the Firebase server, else an error message is displayed. Then, the reset email function, which involves the code snippet shown in Figure 3.3.4.17 is triggered to show a dialog requesting the input of administrator password to proceed updating the email account and changes made. If the admin password is valid, the `updateAdminUser` function in Figure 3.3.4.18 is triggered to update the administrator main email account to the new email and revert previous email account back to user or helper account, while the `updateAdminInfo` function is called to update the administrator profile information. When done, a success email message is delivered through the trigger of `sendUpdateEmail` showed in Figure 3.3.4.19 to send an email to the new email account through the SMTP server added in the project mobile application. When email changes are saved, the administrator is logged out of the project mobile application to then login with the new email account.

```

27 //Set up user details
28 void setUserDetails() async {
29   setState(() {
30     isLoading = true;
31   });
32
33   //Set up user values in the form
34   if (widget.user != null) {
35     user = widget.user;
36   } else {
37     var temp = await UserN().getUserInfo(currentUser!.email!);
38     if (temp != null) {
39       setState(() {
40         user = temp;
41       });
42       if (!mounted) return;
43       ScaffoldMessenger.of(context).showSnackBar(
44         const SnackBar(content: Text("User Information Set!")));
45       debugPrint("User Get");
46     } else {
47       if (!mounted) return;
48       ScaffoldMessenger.of(context).showSnackBar(
49         const SnackBar(content: Text("User Information Not Found!")));
50       debugPrint("User lost");
51     }
52   }
53   setState(() {
54     isLoading = false;
55   });
56 }

```

Figure 3.3.4.1 setUserDetails Function in profile\_page.dart

```

99 retrieve user info
100 Future<UserN?> getUserInfo(String email) async {
101   try {
102     var userInfo =
103       await FirebaseFirestore.instance.collection('user').doc(email).get();
104     if (userInfo.exists) {
105       UserN currentUser = UserN.fromMap(userInfo.data());
106       return currentUser;
107     } else {
108       var userInfo = await FirebaseFirestore.instance
109         .collection('inactiveUser')
110         .doc(email)
111         .get();
112       UserN currentUser = UserN.fromMap(userInfo.data());
113       return currentUser;
114     }
115   } catch (error) {
116     debugPrint(error.toString());
117     return null;
118   }
119 }

```

Figure 3.3.4.2 getUserInfo Function in user.dart

```

385  ///Update user information to firebase
386  Future editComplete(UserN user, BuildContext context) async {
387    showDialog(
388      context: context,
389      barrierDismissible: false,
390      builder: ((context) => loadingWait());
391    if (editProfileKey.currentState!.validate()) {
392      setState(() {
393        user.name = nameController.text;
394        user.phone = phoneController.text;
395        user.status = dropDownItem;
396        user.gender = groupValue;
397      });
398      await FirebaseFirestore.instance
399        .collection('user')
400        .doc(user.email)
401        .update(user.toMap())
402        .catchError((onError) {
403          debugPrint(onError.toString());
404          Navigator.of(context).pop();
405          ScaffoldMessenger.of(context)
406            .showSnackBar(SnackBar(content: Text(onError.toString())));
407        }).whenComplete(() {
408          navigatorKey.currentState!.popUntil((route) => route.isFirst);
409          ScaffoldMessenger.of(context)
410            .showSnackBar(const SnackBar(content: Text("Update Complete")));
411          Navigator.of(context).push(MaterialPageRoute(
412            builder: (context) => AccountDetails(user: user)); // MaterialPageRoute
413        });
414      } else {
415        Navigator.of(context).pop();
416        return;
417      }
418    }

```

Figure 3.3.4.3 editComplete Function in edit\_profile\_page.dart

```

638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661

```

```

onToggle: (index) async {
  debugPrint('switched to: $index');
  if (user.startTime != null &&
    user.endTime != null) {
    await showConfirmOnOffHelperStatus(
      user.email!);
  }
  if (index == 0) {
    setState(() {
      user.helperOn = true;
      choiceType = 0;
    });
    UserN().updateHelperOnOffStatus(
      user.email!, true);
    debugPrint('Helper Status On');
  } else {
    setState(() {
      user.helperOn = false;
      choiceType = 1;
    });
    UserN().updateHelperOnOffStatus(
      user.email!, false);
    debugPrint('Helper Status Off');
  }
}

```

Figure 3.3.4.4 Toggle Switch Function in helper\_details.dart To Toggle Helper Status

```

706 //confirmation dialog before toggle helper status if there is duty period
707 Future showConfirmOnOffHelperStatus(String helperId) async {
708   await showDialog(
709     context: context,
710     builder: (context) {
711       return AlertDialog(
712         title: const Text("Are you sure?"),
713         content: const Text(
714           "Once you manually on off the status, the duty period set will be removed"), // Text
715         actions: [
716           TextButton(
717             onPressed: () {
718               User().removeDutyPeriodHelper(helperId);
719               setState(() {
720                 user.startTime = null;
721                 user.endTime = null;
722               });
723               Navigator.of(context).pop();
724               if (!mounted) return;
725               setState(() {});
726             },
727             child: const Text("Yes")), // TextButton
728           TextButton(
729             onPressed: () {
730               Navigator.of(context).pop();
731             },
732             child: const Text("Cancel")), // TextButton
733         ],
734       ); // AlertDialog
735     });
736 }

```

Figure 3.3.4.5 showConfirmOnOffHelperStatus Function to Display Dialog

```

372 //toggle helper duty on off status
373 Future updateHelperOnOffStatus(String helperId, bool status) async {
374   try {
375     await FirebaseFirestore.instance
376       .collection('user')
377       .doc(helperId)
378       .update({'helperOn': status});
379   } catch (error) {
380     debugPrint("Cannot update helper OnOff Status :${error.toString()}");
381   }
382 }

```

Figure 3.3.4.6 updateHelperOnOffStatus Function in user.dart

```

427 //clear helper duty period
428 Future removeDutyPeriodHelper(String helperId) async {
429   Map<String, dynamic> map = <String, dynamic>{};
430   map['startTime'] = null;
431   map['endTime'] = null;
432   try {
433     await FirebaseFirestore.instance
434       .collection('user')
435       .doc(helperId)
436       .update(map);
437   } catch (error) {
438     debugPrint("Unable to remove duty period $error");
439   }
440 }

```

Figure 3.3.4.7 removeDutyPeriodHelper Function in user.dart



```

String start = DateFormat.jm()
    .format(DateTime(
        now.year,
        now.month,
        now.day,
        result.startTime.hour,
        result.startTime.minute)); // DateTime
String end = DateFormat.jm()
    .format(DateTime(
        now.year,
        now.month,
        now.day,
        result.endTime.hour,
        result.endTime.minute)); // DateTime
setState(() {
    user.startTime =
        start.toString();
    user.endTime = end.toString();
    startTimeController.text =
        start.toString();
    endTimeController.text =
        end.toString();
});
UserN()
    .updateUserInformation(user);
ScaffoldMessenger.of(context)
    .showSnackBar(SnackBar(
        content: Text(
            "From $start to $end")); // Text // SnackBar
if (!mounted) return;
setState(() {});
} else {
    ScaffoldMessenger.of(context)
        .showSnackBar(const SnackBar(
            content: Text(
                "No time is set!")); // Text // SnackBar

```

Figure 3.3.4.8 Code Snippet To Set Helper Duty Period in helper\_details.dart

```

121  ///update user info
122  Future updateUserInformation(UserN updateUser) async {
123      await FirebaseFirestore.instance
124          .collection('user')
125          .doc(updateUser.email)
126          .update(updateUser.toMap());
127  }

```

Figure 3.3.4.9 updateUserInformation in user.dart

```

129  ///change account type
130  Future changeAccountType(String userId, bool isHelper) async {
131      try {
132          var getUserInfo =
133              await FirebaseFirestore.instance.collection("user").doc(userId).get();
134          if (getUserInfo.exists) {
135              if (getUserInfo.data()["isHelper"] != isHelper) {
136                  await FirebaseFirestore.instance
137                      .collection('user')
138                      .doc(userId)
139                      .update({'isHelper': isHelper});
140              if (getUserInfo.data()["helperOn"] == null) {
141                  await FirebaseFirestore.instance
142                      .collection('user')
143                      .doc(userId)
144                      .update({'helperOn': true});
145              }
146          }
147      }
148      catch (error) {
149          debugPrint("Cannot update user account type:${error.toString()}");
150      }
151  }

```

Figure 3.3.4.10 changeAccountType Function in user.dart

```

31 //Fill up admin information
32 Future setRequiredInformation() async {
33   setState() {
34     loading = true;
35   });
36   if (currentAdmin!.email != null) {
37     admin = UserN.fromMap(
38       await AdminControls().getAdminInfo(currentAdmin!.email!)); // UserN.fromMap
39     setState() {
40       loading = false;
41     });
42   }
43 }

```

Figure 3.3.4.11 setRequiredInformation Function in admin\_nav.dart

```

595 FocusManager.instance.primaryFocus?.unfocus();
596 if (formKey.currentState!.validate()) {
597   if (emailController.text != initEmail) {
598     showDialog(
599       context: context,
600       barrierDismissible: false,
601       builder: (context) => loadingWait(),
602     );
603     //check for emails in server before allow to switch email
604     bool checkEmailInServer = await AdminControls()
605       .checkIfEmailInServer(emailController.text);
606     if (!mounted) return;
607     Navigator.of(context).pop();
608     if (checkEmailInServer) {
609       await resetEmail();
610     } else {
611       ///email is not registered in system
612       showDialog(
613         context: context,
614         barrierDismissible: false,
615         builder: (context) => AlertDialog(
616           backgroundColor: AdminColour.color2,
617           title: const Text(
618             "Email Not Exists in System!",
619             style: TextStyle(
620               fontSize: 18,
621               fontWeight: FontWeight.bold,
622               color: AdminColour.color3,
623             ), // TextStyle
624           ), // Text
625           content: Text(
626             "The Email ${emailController.text} is not found in the system server, please check again or proceed to
627             style: const TextStyle(
628               fontSize: 15,

```

Figure 3.3.4.12 Code Snippet of Function to Edit Admin Profile in edit\_admin\_profile\_info.dart Part A

```

629         fontWeight: FontWeight.bold,
630         color: AdminColour.color4,
631       ), // TextStyle
632     ), // Text
633     actions: [
634       TextButton(
635         onPressed: () =>
636           Navigator.of(context).pop(),
637         child: const Text(
638           "Ok",
639           style: TextStyle(
640             fontSize: 16,
641             fontWeight: FontWeight.bold,
642             color: AdminColour.color3,
643           ), // TextStyle
644         ), // Text // TextButton
645       ),
646     ], // AlertDialog
647   );
648 } else {
649   ///update success
650   showDialog(
651     context: context,
652     barrierDismissible: false,
653     builder: ((context) => loadingWait());
654   );
655   admin.name = nameController.text;
656   admin.gender = adminGender;
657   admin.phone = phoneController.text;
658   admin.status = adminStatus;
659   if (admin != temp) {
660     await AdminControls()
661       .updateAdminInfo(admin)
662       .then((value) async {

```

Figure 3.3.4.13 Code Snippet of Function to Edit Admin Profile in edit\_admin\_profile\_info.dart Part B

```

663 ScaffoldMessenger.of(context).showSnackBar(
664     const SnackBar(content: Text("Update Completed")));
665 Navigator.of(context).pop();
666 await Navigator.of(context).pushAndRemoveUntil(
667     MaterialPageRoute(
668         builder: (context) =>
669             AdminProfilePage(admin: admin), // MaterialPageRoute
670             ModalRoute.withName(AdminProfilePage.route));
671     );
672 } else {
673     //if there is no changes even admin press submit button
674     ScaffoldMessenger.of(context).showSnackBar(
675         const SnackBar(content: Text("No Changes Made!")));
676     Navigator.of(context).pop();
677     await Navigator.of(context).pushAndRemoveUntil(
678         MaterialPageRoute(
679             builder: (context) =>
680                 AdminProfilePage(admin: admin), // MaterialPageRoute
681                 ModalRoute.withName(AdminProfilePage.route));
682         );
683     }
684 } else {
685     return;
686 }
687 ), // FloatingActionButton
688 ); // Scaffold
689 }

```

Figure 3.3.4.14 Code Snippet of Function to Edit Admin Profile in  
edit\_admin\_profile\_info.dart Part C

```

264 //update the admin info to the firebase server in user collection
265 Future updateAdminInfo(UserN admin) async {
266     try {
267         await FirebaseFirestore.instance
268             .collection("user")
269             .doc(admin.email)
270             .update(admin.toMap());
271     } catch (error) {
272         debugPrint("Update Admin Info Error: ${error.toString()}");
273     }
274 }
275
276 //update user info (may remove cause similar function in user.dart)
277 Future updateUserInfo(UserN user) async {
278     try {
279         await FirebaseFirestore.instance
280             .collection("user")
281             .doc(user.email)
282             .update(user.toMap());
283     } catch (error) {}
284     debugPrint("Admin Update User Error: ${error.toString()}");
285 }
286 }

```

Figure 3.3.4.15 updateAdminInfo Function in admin.dart

```

326 // Returns true if email address is registered in server
327 Future<bool> checkIfEmailInServer(String emailAddress) async {
328     try {
329         // Fetch sign-in methods for the email address
330         final checkList =
331             await FirebaseAuth.instance.fetchSignInMethodsForEmail(emailAddress);
332         if (checkList.isNotEmpty) {
333             // True if email registered in project server
334             return true;
335         } else {
336             // False if email not registered in project server
337             return false;
338         }
339     } catch (error) {
340         debugPrint("Error in Checking Email From Firebase Authentication");
341         return false;
342     }
343 }

```

Figure 3.3.4.16 checkEmailInServer Function in admin.dart

```

788     await AdminControls()
789       .updateAdminUser(
790         FirebaseAuth.instance.currentUser!.email,
791         emailController.text)
792       .then((value) async {
793         admin.name = nameController.text;
794         admin.gender = adminGender;
795         admin.phone = phoneController.text;
796         admin.status = adminStatus;
797
798         //update the changes to new admin email user instead
799         admin.email = emailController.text;
800         await AdminControls()
801           .updateAdminInfo(admin)
802           .then((value) {
803             //first pop loading, then pop dialog
804             Navigator.of(context).pop();
805             Navigator.of(context).pop();
806             ScaffoldMessenger.of(context).showSnackBar(
807               const SnackBar(
808                 content: Text("Update Completed")); // SnackBar
809             ));
810         });
811       showDialog(
812         context: context,
813         barrierDismissible: false,
814         builder: (context) => loadingWait());
815       await sendUpdateEmail(emailController.text);
816       SharedPreferences preferences =
817         await SharedPreferences.getInstance();
818       await preferences.setBool("adminLogin", false);
819       AdminControls().saveSignOutDate();
820       UserN().logout(_scaffoldKey.currentContext!);

```

Figure 3.3.4.17 Code Snippet of Function to Update Admin Email in edit\_admin\_profile\_info.dart

```

114     //update admin user both in user collection and admin collection
115     Future updateAdminUser(String oldEmail, String newEmail) async {
116       //update admin email in admin collection
117       await FirebaseFirestore.instance
118         .collection("admin")
119         .doc("adminKey")
120         .update({"email": newEmail}).catchError((error) =>
121           debugPrint("Update Admin Email Error: ${error.toString()}"));
122
123       //set admin to true in the user collection of new user email
124       await FirebaseFirestore.instance
125         .collection("user")
126         .doc(newEmail)
127         .update({"admin": true});
128
129       //remove the admin field in the old admin of old email
130       await FirebaseFirestore.instance
131         .collection("user")
132         .doc(oldEmail)
133         .update({"admin": false});
134       await saveAdminToken();
135     }

```

Figure 3.3.4.18 updateAdminUser in admin.dart

```

854     //function to send email in background
855     Future sendUpdateEmail(String newEmail) async {
856       //use vcarecircle email for now (2 factor verification enabled to use app password)
857       const String senderEmail = "vcarecircle2023@gmail.com";
858
859       //set up account for sending email (vcare email with app password (auto-generated))
860       final SmtServer smtpServer = gmail(senderEmail, "hrnyqnoteeecffk");
861
862       //message content set to be send to the new updated email
863       final message = Message()
864         ..from = const Address(senderEmail, 'VCare Circle')
865         ..recipients.add(newEmail)
866         ..ccRecipients.addAll(['tzhuyuan75@1utar.my', 'tzhuyuan74@gmail.com'])
867         ..subject = 'Admin Email Updated on ${DateHandler().parseEmailDate()}'
868         ..html =
869         |<h1>Hi there $newEmail</h1>\n<p>The admin email has updated to the new email !</p>;
870
871       try {
872         //the command to send out an email
873         final sendReport = await send(message, smtpServer);
874         debugPrint("Email sent to $newEmail: $sendReport");
875       } on MailerException catch (e) {
876         debugPrint("Message not sent.");
877         debugPrint(e.message);
878         for (var p in e.problems) {
879           debugPrint('Problem: ${p.code}: ${p.msg}');
880         }
881       }
882     }

```

Figure 3.3.4.19 sendUpdateEmail Function in edit\_admin\_profile\_info.dart

### 3.3.5 Online Communication Function Development

There are two methods implemented under the online communication module, which are the online chat communication function, and the online voice call function. Both methods are usable by the user and helper during an active session for the purpose of communicating with each other for user to get the help required, while allow helper to understand the situation or problem faced by the user. Here, the online call function is developed with the use of Agora SDK, while online chat communication is utilizing the use of Firebase server and stream builder as the widget to get updates on the chat and images sent.

Agora is a free SDK compatible with Flutter that contributes to adding online communication functions such as online video call, online voice call, and online chat conversation to Flutter projects. In this project, the Agora SDK is implemented for the purpose of allowing online voice call between the user and helper. First of all, an Agora project is created at the Agora Developer console as shown in Figure 3.3.5.1 to gain access to its API server and services, and the installing of Agora SDK package in the project is done as the basic setup of the Agora SDK to the project.

When a user or helper starts an online voice call, the Agora engine is initialized by triggering the `initializeCallEngine` function in Figure 3.3.5.2 to first request the permission to use the mobile device microphone to allow the user or helper voice to be recorded in the call, then the Agora engine is created and setup with the basic settings for online voice call, together with the adding of event handler to allow the project having control over the voice call in any event cases happen. When the setup is completed, the project then enables the user or helper to join in the online voice call through calling the `joinCall` function displayed in Figure 3.3.5.3. When the user or helper is done having conversation over the voice call, the `leaveCall` function in Figure 3.3.5.3 is called after the user or helper selects the “End Call” button to trigger the function to end the online call and close the Agora engine. The `sendCallMessageInChatRoom` shown in Figure 3.3.5.4 is also called when a user or helper starts a call to send a chat message in the session as an indicator of the user or helper has joined the online voice call channel.

Next, the online chat communication is to allow user to communicate with the assigned helper during an active session. This function mainly focuses on using stream builder to communicate with Firebase server in getting the chat contents from the session document to display to the user and helper. This is done by first creating a chatroom map class containing the chatroom session information, such as the user ID, helper ID, session ID, date, and rating. A chat map class is also created to store the chat content values, such as message, sender ID, send time, content type, and image URL.

For the project mobile application, the chatroom is created through the `buildChatRoom` function in Figure 3.3.5.5 and Figure 3.3.5.6 whenever user requests help from a helper which both the user's and helper's email are stored as user and helper ID and used to generate the chatroom session ID. After that, the user and helper can start communicating with each other in the chatroom. The stream builder shown in Figure 3.3.5.7 is used for the chat page of user and helper to allow retrieving chat contents from the Firebase server in a continuous stream to minimize latency. The user's sent text messages are saved to the Firebase through the `sendMessage` function in Figure 3.3.5.8, while user's sent images from gallery or camera are saved through the `sendImages` function in Figure 3.3.5.9. The two functions work together with the help of the function shown in Figure 3.3.5.10.

The concept here is the Firebase server saves the text messages and images sent by the user and helper, while the stream builder in the chat page continuously checks and retrieves any updates on the chat contents from the Firebase and shows the updates of the chat content in the chat page in descending order based on the time sent to have the proper order of conversation between the two parties, making the online chat communication possible.

When a chatroom session is complete, the chatroom is then removed for the purpose of privacy conceal for the helper and the user. Before deleting the chatroom, the chatroom session information and chat content is being transferred to a history class document which saves the contents such as helper ID, user ID, session ID, rating, session date, and duration of session, created in Firebase as a backup to save the data values. To remove the active chatroom session, first the user's and helper's respective assigned

helper and client id are removed, along with the helper QR through the function `endChatRoomSession` in Figure 3.3.5.11. Then, the `setHistoryAndClearChatRoom` function in Figure 3.3.5.12 and Figure 3.3.5.13 is called to create a history class for saving the chatroom session information and chats through the `createHistory` function in Figure 3.3.5.14. When the transfer of data is complete, the chatroom session is then deleted to clear the chatroom record in the Firebase.

Furthermore, rating function is also applied for the helping session which is done for the purpose of allowing user to optionally rate the help gained from the helper throughout the session from one to five based on satisfaction. After the user gives the rating for the helper, a rating class is created to save the rating and session information. Using the `setRatingForChatRoom` in Figure 3.3.5.15, the rating is saved to the chatroom before the data is transferred to the history class. If there is no rating given, then the rating is saved as zero.

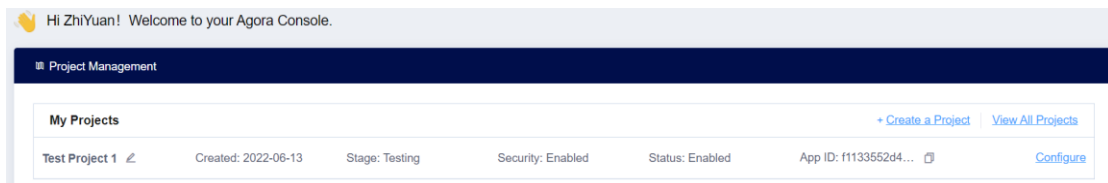


Figure 3.3.5.1 Project Added in Agora Console

```

83 //start agora engine settings
84 //issue with emulator testing (mobile phone test can proceed but cannot join server)
85 Future initializeCallEngine() async {
86 //check microphone permission
87 SharedPreferences preferences = await SharedPreferences.getInstance();
88 await preferences.setBool("inCallNow", true);
89 await Permission.microphone.request();
90 //start up agora
91 agoraEngine = createAgoraRtcEngine();
92 await agoraEngine
93   .initialize(RtcEngineContext(appId: await AgoraConstant().getAppId()));
94 //handling events occurring during call
95 agoraEngine.registerEventHandler(RtcEngineEventHandler(
96   onJoinChannelSuccess: (connection, elapsed) {
97     Fluttertoast.showToast(msg: "You have joined the call channel!");
98     setState() {
99       joinedCall = true;
100     });
101     updateStatus();
102   },
103   onUserJoined: (connection, remoteUid, elapsed) {
104     Fluttertoast.showToast(msg: "Helper has joined the call!");
105     if (!mounted) return;
106     setState() {
107       remoteUidGet = remoteUid;
108     });
109     updateStatus();
110     if (endCallTimer != null) {
111       endCallTimer!.cancel();
112     }
113   },
114   onUserOffline: (connection, remoteUid, reason) {
115     Fluttertoast.showToast(msg: "Helper has left the call!");
116     setState() {
117       remoteUidGet = remoteUid;
118     });
119     updateStatus();
120
121     //leave call upon leave the remote user
122     endCallTimer = Timer(const Duration(minutes: 5), () async {
123       debugPrint("trigger end call");
124       await leaveCall();
125       if (!mounted) return;
126       Navigator.of(context).pushAndRemoveUntil(
127         MaterialPageRoute(builder: (context) => const HomePage()),
128         ModalRoute.withName(HomePage.route));
129     }); // Timer
130   },
131   onAudioVolumeIndication:
132     (connection, speakers, speakerNumber, totalVolume) =>
133     debugPrint("On Audio Test Sound:$totalVolume"),
134   onLocalAudioStateChanged: (connection, state, error) =>
135     debugPrint("Check User Audio State ${state.name}"),
136 )); // RtcEngineEventHandler
137 if (!joinedCall) {
138   joinCall();
139 }
140 }

```

Figure 3.3.5.2 initializeCallEngine Function in in\_call\_screen.dart

```

167 Future joinCall() async {
168 //function to join channel
169 await agoraEngine.joinChannel(
170   channelId: await AgoraConstant().getChannelName(),
171   options: const ChannelMediaOptions(
172     clientRoleType: ClientRoleType.clientRoleBroadcaster,
173     channelProfile: ChannelProfileType.channelProfileCommunication,
174   ),
175   token: await AgoraConstant().getCallToken(),
176   uid: 101);
177 }
178
179 //leave call channel
180 Future leaveCall() async {
181 //reset variable to be
182 setState() {
183   joinedCall = false;
184   remoteUidGet = null;
185 };
186 SharedPreferences preferences = await SharedPreferences.getInstance();
187 await preferences.setBool("inCallNow", false);
188 await chatRoom().updateChatRoomCallStatus(
189   "${widget.userEmail}:${widget.helperEmail}", false);
190 await agoraEngine.leaveChannel();
191 }

```

Figure 3.3.5.3 joinCall Function and leaveCall function, in in\_call\_screen.dart



```

170 void sendCallMessageInChatRoom() {
171   setState(() {
172     chat.message = "User has entered the call channel!";
173     chat.photoUrl = null;
174     chat.contentType = 0;
175     chat.time = DateTime.now().millisecondsSinceEpoch;
176   });
177   HelpChat().setMessagesToChatRoom(chatRoomId!, chat);
178   PushNotification().sendNotificationMessage(widget.helperEmail,
179     chat.message ?? "User has entered the call channel!");
180
181   //try to set chat to max extent after sending messages
182   if (chatController.hasClients) {
183     setState(() {
184       WidgetsBinding.instance.addPostFrameCallback((_) {
185         chatController.animateTo(
186           chatController.position.maxScrollExtent,
187           duration: const Duration(milliseconds: 50),
188           curve: Curves.easeOut,
189         );
190       });
191     });
192   }
193 }

```

Figure 3.3.5.4 sendCallMessageInChatRoom in chat\_page.dart

```

73 ///Use to initiate chat room
74 Future buildChatRoom(String userId, String helperId) async {
75   ChatRoom room = ChatRoom();
76   String newChatRoomId = "$userId:$helperId";
77   bool checkPrevious = await searchPreviousChatRoom(userId, helperId);
78   if (checkPrevious != null) {
79     if (checkPrevious) {
80       debugPrint("Previous Record was not cleared");
81     } else {
82       room.userId = userId;
83       room.helperId = helperId;
84       room.chatRoomId = newChatRoomId;
85       //see if can get the correct format of string or not
86       room.createDate = DateFormat("dd/MM/yyyy").format(DateTime.now());
87       await createChatRoom(userId, helperId, room).catchError((onError) {
88         debugPrint(onError.toString());
89       });
90
91       debugPrint("Create New");
92     }
93   } else {
94     debugPrint("Error in Firebase Firesotre to search for chatroom records!");
95   }
96 }

```

Figure 3.3.5.5 buildChatRoom Function in chatroom\_model.dart

```

46 ///Create chat space
47 Future createChatRoom(
48   String userId, String helperId, ChatRoom chatRoom) async {
49   await FirebaseFirestore.instance
50     .collection('chatRoom')
51     .doc('$userId:$helperId')
52     .set(chatRoom.toMap())
53     .catchError((error) {
54       debugPrint(error.toString());
55     });
56 }
57
58 ///check if there was a record
59 Future<bool> searchPreviousChatRoom(String userId, String helperId) async {
60   var snapshot = await FirebaseFirestore.instance
61     .collection('chatRoom')
62     .doc("$userId:$helperId")
63     .get();
64   if (snapshot.exists) {
65     debugPrint("There was a record with helper!");
66     return true;
67   } else {
68     debugPrint("No Previous Record Found!");
69     return false;
70   }
71 }
72

```

Figure 3.3.5.6 Extended Functions for buildChatRoom Function in chatroom\_model.dart

```

237 //build chat page content
238 child: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
239   stream: FirebaseFirestore.instance
240     .collection('chatRoom')
241     .doc(chatRoomId)
242     .collection('chats')
243     .orderBy('time', descending: false)
244     .snapshots(),
245   builder: (context, snapshot) {
246     if (snapshot.hasData) {
247       if (snapshot.data!.docs.isNotEmpty) {
248         return ListView.builder(
249           controller: chatController,
250           physics: const BouncingScrollPhysics(),
251           itemCount: snapshot.data!.docs.length,
252           itemBuilder: (BuildContext context, int index) {
253             return ChatTiles(
254               messageContent: snapshot.data!.docs[index]
255                 .data()['contentType'] ==
256                 0
257                 ? snapshot.data!.docs[index]
258                   .data()['message']
259                 : snapshot.data!.docs[index]
260                   .data()['photoUrl'],
261               fromUser: snapshot.data!.docs[index]
262                 .data()['sendId'] ==
263                 widget.user.email
264                 ? true
265                 : false,
266               contentType: snapshot.data!.docs[index]
267                 .data()['contentType'] ??
268                 0,
269             ); // ChatTiles
270           }); // ListView.builder
271       } else {
272         //when chat content empty show label

```

Figure 3.3.5.7 Stream Builder Code Snippet in chat\_page.dart and helper\_chat\_page.dart

```

118 //send text messages
119 void sendMessage(TextEditingController message) {
120   if (message.text.isNotEmpty) {
121     setState(() {
122       chat.message = message.text;
123       chat.photoUrl = null;
124       chat.contentType = 0;
125       chat.time = DateTime.now().millisecondsSinceEpoch;
126     });
127     HelpChat().setMessagesToChatRoom(chatRoomId!, chat);
128     PushNotification()
129       .sendNotificationMessage(widget.helperEmail, message.text);
130     setState(() {
131       messageController.text = "";
132     });
133   }
134
135   //try to set chat to max extend after sending messages
136   if (chatController.hasClients) {
137     setState(() {
138       WidgetsBinding.instance.addPostFrameCallback((_) {
139         chatController.animateTo(
140           chatController.position.maxScrollExtent,
141           duration: const Duration(milliseconds: 50),
142           curve: Curves.easeOut,
143         );
144       });
145     });
146   }
147 }
148

```

Figure 3.3.5.8 sendMessage Function in chat\_page.dart and help\_chat\_page.dart

```

150  ///send images
151  void sendImage(String image) {
152    if (image != null) {
153      setState(() {
154        chat.message = null;
155        chat.photoUrl = image;
156        chat.contentType = 1;
157        chat.time = DateTime.now().millisecondsSinceEpoch;
158      });
159      HelpChat().setMessagesToChatRoom(chatRoomId!, chat);
160      PushNotification().sendNotificationMessage(widget.helperEmail, "[Image]");
161      setState(() {
162        image = "";
163      });
164    }
165
166    ///try to set chat to max extend after sending images
167    if (chatController.hasClients) {
168      setState(() {
169        WidgetsBinding.instance.addPostFrameCallback(_) {
170          chatController.animateTo(
171            chatController.position.maxScrollExtent,
172            duration: const Duration(milliseconds: 50),
173            curve: Curves.easeOut,
174          );
175        });
176      });
177    }
178  }

```

Figure 3.3.5.9 sendImage Function in chat\_page.dart and help\_chat\_page.dart

```

41  ///add messages to chatroom
42  Future setMessagesToChatRoom(String chatRoomId, HelpChat chat) async {
43    await FirebaseFirestore.instance
44      .collection('chatRoom')
45      .doc(chatRoomId)
46      .collection('chats')
47      .add(chat.toMap())
48      .catchError((onError) {
49        debugPrint(onError.toString());
50      });
51  }
52

```

Figure 3.3.5.10 setMessagesToChatRoom Extended Function in sendMessage and sendImage Function

```

116  ///end chatroom when done by clearing the user's current helper and helper's qr and current client
117  Future endChatRoomSession(String chatRoomId, double rating) async {
118    List<String> getEmails = chatRoomId.split(":");
119    String userId = getEmails[0], helperId = getEmails[1];
120    Map<String, dynamic> helperDataRemove = <String, dynamic>{};
121    helperDataRemove['getQR'] = null;
122    helperDataRemove['currentClient'] = null;
123    setRatingForChatRoom(chatRoomId, rating);
124    await FirebaseFirestore.instance
125      .collection('user')
126      .doc(userId)
127      .update({'currentHelper': null});
128    await FirebaseFirestore.instance
129      .collection('user')
130      .doc(helperId)
131      .update(helperDataRemove);
132  }

```

Figure 3.3.5.11 endChatRoomSession Function in chatroom\_model.dart

```

77 //move chatroom information to history and clear chatroom
78 Future setHistoryAndClearChatRoom(ChatRoom chatRoom) async {
79 //start calculate duration
80 String timeNow = DateFormat("yyyy-MM-dd HH:mm:ss").format(DateTime.now());
81 String chatRoomStartTime = chatRoom.createDate!;
82 int duration = calculateDurationBetweenTime(chatRoomStartTime, timeNow);
83 //calculate duration here
84 // int duration = calculateDurationBetweenTime(chatRoom.createDate, endTime)
85 await createHistory(chatRoom.helperId!, chatRoom.userId!,
86 chatRoom.chatRoomId!, chatRoom.rating!, timeNow, duration)
87 .then((value) async {
88 await FirebaseFirestore.instance
89 .collection('chatRoom')
90 .doc(chatRoom.chatRoomId)
91 .collection('chats')
92 .get()
93 .then((chats) async {
94 for (var element in chats.docs) {
95 HelpChat data = HelpChat.fromMap(element.data());
96 await FirebaseFirestore.instance
97 .collection('history')
98 .doc(value)
99 .collection('chats')
100 .add(data.toMap());
101 await FirebaseFirestore.instance
102 .collection('chatRoom')
103 .doc(chatRoom.chatRoomId)
104 .collection('chats')
105 .doc(element.id)
106 .delete();
107 }
108
109 //will delete after use for chatroom
110 await FirebaseFirestore.instance
111 .collection('chatRoom')
112 .doc(chatRoom.chatRoomId)
113 .delete();
114 });
115 });
116 }

```

Figure 3.3.5.12 setHistoryandClearChatRoom Function in history\_model.dart

```

69 int calculateDurationBetweenTime(String startTime, String endTime) {
70 //try to return in seconds
71 return (DateFormat("yyyy-MM-dd HH:mm:ss")
72 .parse(endTime)
73 .difference(DateFormat("yyyy-MM-dd HH:mm:ss").parse(startTime))
74 .inSeconds);
75 }

```

Figure 3.3.5.13 calculateDurationBetweenTime Function in history\_model.dart

```

48 //generate history record in firebase
49 Future<String> createHistory(String helperId, String userId, String chatId,
50 double? rating, String docIdGet, int duration) async {
51 String docId = docIdGet;
52 History history = History();
53 history.helperId = helperId;
54 history.userId = userId;
55 history.rating = rating ?? 0;
56 history.chatRoomId = chatId;
57 history.duration = duration;
58 history.date = DateHandler().getTodayDate();
59 await FirebaseFirestore.instance
60 .collection('history')
61 .doc(docId)
62 .set(history.toMap())
63 .onError((error, stackTrace) {
64 debugPrint("Error in Histroy: ${error.toString()}");
65 });
66 return docId;
67 }

```

Figure 3.3.5.14 createHistory Function in history\_model.dart

```

48 //upload chatroom ratings
49 Future uploadRating(RatingsForHelper record) async {
50   await FirebaseFirestore.instance
51     .collection('chatRoom')
52     .doc(record.chatRoomId)
53     .update({'rating': record.totalRating});
54 }

```

Figure 3.3.5.15 setRatingForChatRoom in rating\_model.dart

### 3.3.6 QR Identification Function Development

The purpose of QR identification function is to allow user to identify the random assigned helper if they have personally agreed to have a physical meeting for personal security. The concept behind QR identification method is to provide user a QR scanner with the assigned value of helper QR created when user requested help from a helper in the project mobile application. On the other hand, the assigned helper is provided with the QR generator to display the helper QR for the user to scan and identify whether is he or she is the correct one. The helper QR value is generated using the createHelperQr function in Figure 3.3.6.1 with the format of using the user ID and helper ID combined with 6 random numbers. The generated QR value is then saved in the Firebase at the helper's document. As for the QR scanner and QR generator, the Flutter plugin tools are used to add in the QR scanner and QR generator functionality into the project mobile application. When the user selects to open the QR scanner, the QR value supposed to be scanned is retrieved from the Firebase through the assigned helper document by using the retrieveHelperQR function as shown in Figure 3.3.6.2. Here the permission to use the mobile phone camera is asked before proceeding to show QR scanner interface as it involves accessing dangerous permission. If no permission is given the user is unable to use the QR code scanner. On the other hand, when helper selects the icon button to show the QR code, the project mobile application also uses the retrieveHelperQr function to retrieve the QR code from Firebase to add in the value to the QR code generator to create a QR for scanning. When user gets a successful scan, a success overlay messages is shown over the QR scanner. If user gets an invalid or wrong QR code scanned, an error message is shown to user instead. The main structure for user's QR scanner and helper's QR code generator can be seen in Figure 3.3.6.3 and 3.3.6.4.

```

215 //generate helper QR when user call
216 Future<String> createHelperQr(String userId, String helperId) async {
217   int randomEndNumber = Random().nextInt(900000) + 100000;
218   String createQr = "$userId:$helperId:$randomEndNumber";
219   await FirebaseFirestore.instance
220     .collection('user')
221     .doc(helperId.toString())
222     .update({"getQR": createQr.toString()})
223     .onError((error, stackTrace) => debugPrint(error.toString()))
224     .then((value) {
225       debugPrint("QR updated for helper $helperId");
226     });
227   return createQr;
228 }

```

Figure 3.3.6.1 createHelperQr Function in user.dart

```

231 //get QR from firebase when being requested
232 Future<String> retrieveHelperQr(String helperId) async {
233   String qrCode = "";
234   final snapshot =
235     await FirebaseFirestore.instance.collection('user').doc(helperId).get();
236   if (snapshot.exists && snapshot != null) {
237     if (snapshot.data()!.isEmpty) {
238       qrCode = snapshot.data()!["getQR"];
239       return qrCode;
240     } else {
241       debugPrint("QRCode null");
242       return qrCode;
243     }
244   } else {
245     return qrCode;
246   }
247 }

```

Figure 3.3.6.2 retrieveHelperQr Function in user.dart

```

85 //set camera at back and add key to save context state
86 child: QRView(
87   key: qrScannerKey,
88   cameraFacing: CameraFacing.back,
89   onQRViewCreated: (qrViewController) {
90     //when get information from QR
91     buildViewGetResults(qrViewController);
92   },
93   //crop out scanner view
94   overlay: QrScannerOverlayShape(
95     borderColor: Colors.grey.withOpacity(0.7),
96     overlayColor: Colors.white60,
97     borderRadius: 25,
98     borderLength: 30,
99     borderWidth: 1,
100    cutoutSize: widget.size,
101  ), // QrScannerOverlayShape
102  onPermissionSet: (controller, permission) {
103    permissionHandling(context, controller, permission);
104  },
105  )) // QRView // SizedBox
106

```

Figure 3.3.6.3 QR Scanner Main Code Snippet in qr\_scanner\_user.dart

```

48 ? QRImage(
49   size: widget.size,
50   data: qrCode!,
51   gapless: false,
52   version: QrVersions.auto,
53   padding: const EdgeInsets.all(15),
54   semanticsLabel: "QR code For Helper",
55   backgroundColor: Colors.white24.withOpacity(0.5),
56   foregroundColor: Colors.black,
57 )

```

Figure 3.3.6.4 QR Generator Main Code Snippet in qr\_generator\_helper.dart

### 3.3.7 Helper Finder Function Development

The helper finder function is developed to allow user to request for help from available helpers from the project mobile application, which the user or helper does not need to save any contacts or book any sessions in the project mobile application. This function is mainly developed with Dart language which focuses on the data communication between project mobile application and Firebase server as the browsing technique over the list of users from the Firebase Firestore is required. To search for available helper, the five main criteria are the user account type is “Helper”, the helper is not the user himself/herself, the helper status is on, the helper email is not the administrator email, and the helper is currently not in an active session. In Figure 3.3.7.1, the function `getHelper` is the main function used to search for available helper from the Firebase.

First of all, a helper list is created to store the list of available helpers’ emails that fullfills the five criteria mentioned. Based on the five criteria, the Firebase returns the list of available helpers’ emails to the helper list. Then, a random number index is created between zero and the maximum length of the list. The random number index is then used to select one of the available helpers’ emails from the list, to achieve the random assign system. When the available helper’s email is chosen, it is then used to retrieve the helper information from the Firebase to be returned as user class value for the function. On the other hand, if there is no available helper, the returned value is null.

Continuing from here, this concept is then moved on to be used in the `assignHelper` function shown in Figure 3.3.7.2, which is the main function to be used in the project mobile application to assign helper for the user. After a helper is found for the user, the QR session code for the helper is created and updated to the helper user document, together with the update of helper’s client and user’s helper assigned. Then, a session between the user and helper is created. When this process is complete, the user is properly assigned to an available helper and can start requesting the help needed. On the other hand, if the previous `getHelper` function returns null, the `assignHelper` function shows an error message to the user that currently there are no available helpers.

```

211 //look for available helper on firebase
212 Future<UserN?> getHelper(UserN? user) async {
213   UserN helperUser;
214   List<String> helperID = [];
215   try {
216     var listSnapshot = await FirebaseFirestore.instance
217       .collection('user')
218       .where('email', isNotEqualTo: user!.email)
219       .where('admin', isEqualTo: false)
220       .where('isHelper', isEqualTo: true)
221       .where('helperOn', isEqualTo: true)
222       .where('currentClient', isNull: true)
223       .get();
224     for (var snapshot in listSnapshot.docs) {
225       helperID.add(snapshot.data()['email']);
226       debugPrint(snapshot.data()['email']);
227     }
228     if (helperID.isNotEmpty) {
229       Random random = Random();
230       int max = helperID.length;
231
232       //generate random index between >= 0 and < max
233       int randomIndex = random.nextInt(max);
234       if (randomIndex >= 0) {
235         DocumentSnapshot snapshot = await FirebaseFirestore.instance
236           .collection('user')
237           .doc(helperID[randomIndex]).toString()
238           .get();
239         helperUser = UserN.fromMap(snapshot.data());
240         PushNotification().sendNotificationMessage(
241           helperUser.email!, "You have a request!");
242         return helperUser;
243       } else {
244         debugPrint("Error in Getting Helper in Helper Module");
245         return null;
246       }
247     } else {
248       debugPrint("Currently there is no available helper now!");
249       return null;
250     }
251   } on FirebaseAuthException catch (error) {
252     debugPrint("Helper Error: $error");
253     return null;
254   }
255 }

```

Figure 3.3.7.1 getHelper Function in user.dart

```

407 //Find available helpers
408 Future assignHelper(UserN user) async {
409   setState(() {
410     startFinding = true;
411   });
412
413   //set helper and user information on successful search
414   await UserN().getHelper(user).then((UserN? value) async {
415     if (value != null) {
416       setState(() {
417         helper = value;
418         user.currentHelper = helper!.email;
419         helper!.currentClient = user.email;
420       });
421
422       //generate session QR for helper
423       await UserN().createHelperQr(user.email!, helper!.email!).then((value) {
424         helper!.getQR = value;
425         UserN().updateUserInformation(helper!);
426         UserN().updateUserInformation(user);
427       });
428
429       //create chatroom (in case user straight end session after call)
430       await ChatRoom().buildChatRoom(user.email!, helper!.email!);
431       setState(() {
432         chatRoomId = "${user.email}:${helper!.email}";
433         debugPrint("User and Helper ${helper!.email} is Set!");
434       });
435     } else {
436       debugPrint("Unable to set Helper");
437       ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
438         content:
439           Text("Currently there are no available helper right now!")); //
440     }
441   });
442
443   //end find
444   Future.delayed(const Duration(seconds: 3), () {
445     setState(() {
446       startFinding = false;
447     });
448   }); // Future.delayed
449 }

```

Figure 3.3.7.2 assignHelper Function in home\_page.dart



### 3.3.8 Offline Module Development

The offline module is developed to listen to the internet connection changes when using the project mobile application. This is considered a crucial handling for the project mobile application as most of the functionality provided in the project mobile application requires internet connection, and thus this offline module is developed for the purpose of allowing user to continue ask for help through the project mobile application even in the situation where internet connection is not available. First of all, a Boolean variable is declared together with an internet connection listener function as shown in Figure 3.3.8.1, which this listener is created using the Flutter plugin libraries. This listener is added to two locations, which are the main file of the project mobile application to be able to listen on the changes on internet connection throughout the use of the project mobile application, and at the offline page file to listen to updates on internet connection when the user is back online. The Boolean variable is set to true if there is internet connection, which allows user to go back using the functionalities in the project mobile application, and it is set to false if there is no internet connection which the user is directed to the offline page.

When in the offline situation, the user is still able to ask for help through contacting UTAR through mobile SMS communication or direct mobile phone call, which this can be done using the URL launcher service from Flutter as shown in Figure 3.3.8.2. The user is directed to the default SMS communication platform if user chooses to communicate via SMS messaging and to the default mobile phone dialler screen if user chooses the phone call option. As this involves the use of mobile services, the user is asked to permit the permission to use the mobile SMS and phone call services before continuing. Therefore, unless the user permits the permission to use the SMS communications service and mobile phone call service, the mobile application does not have access to these services.

```

97  ///check internet connection listener
98  Future checkInternetConnection() async {
99    isOnline = await InternetConnectionChecker().hasConnection;
100   debugPrint("Is online?${isOnline}");
101   if (!isOnline) {
102     setState(() {
103       showLeading = false;
104     });
105   }
106
107   ///Check internet status
108   subscription =
109     Connectivity().onConnectivityChanged.listen((connection) async {
110       if (connection != ConnectivityResult.none) {
111         debugPrint("Has Connection?:$connection");
112         isOnline = await InternetConnectionChecker().hasConnection;
113         debugPrint("Is online?${isOnline}");
114         if (!isOnline) {
115           setState(() {
116             showLeading = false;
117           });
118         } else {
119           //see what to do if back online
120           setState(() {
121             showLeading = true;
122           });
123           if (!mounted) return;
124           ScaffoldMessenger.of(context)
125             .showSnackBar(const SnackBar(content: Text("Back Online!")));
126           setState(() {});
127         }
128       } else {
129         setState(() {
130           showLeading = false;
131         });
132       }
133     });

```

Figure 3.3.8.1 checkInternetConnection Function in main.dart and offline\_page.dart

```

333  ///run the sms or phone call command after button selected
334  Future makeContact(String link) async {
335    Uri url = Uri.parse(link);
336    if (!await launchUrl(url)) {
337      debugPrint("Error in making contact with UTAR Help Centre!");
338      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
339        content: Text("Error in making contact with UTAR Help Centre!"))); // SnackBar
340    }
341  }
342  }

```

Figure 3.3.8.2 URL Launcher Function in offline\_page.dart

### 3.3.9 Administrator Module Development

The administrator module contains the administrator controls to handle the data in the project mobile application server, while also includes the application data analysis to conduct analysis on the application data, such as the number of new registered users, application usage count, and the overall duration of sessions created between the users and helpers.

### 3.3.9.1 Administrator Control Development

For this project, administrator module is implemented to allow the administrator having full control over the backend data saved in the project mobile application Firebase server. This is for the purpose of bringing ease for the administrator in handling the data in the Firebase server through the project mobile application instead of opening the Firebase console to handle the data, which provides more interactive and simple user interface design for administrator to view on the data from the Firebase server in a clearer and more understandable way to not confuse the administrator. This also allows more simple handling of data as controls such as editing and deleting can be easier through simple taps on the mobile device, to make handling of data more efficient and effective. The main administrator controls provided for the admin are the user management controls and the history sessions management, which involves the viewing, editing, and deleting of data.

For the user management controls, the administrator can first of all view on the list of users, helpers and inactive users at the list of users, helpers, or inactive users page, which here the stream builder is used to fetch the list of related accounts based on criteria set for the stream, such as the account email not being the administrator account email and the account type preferred, and for inactive user the stream is connected to the inactive user collection instead of the user collection. The Figure 3.3.9.1.1 shows the stream builder for list of users, Figure 3.3.9.1.2 for the list of helpers and Figure 3.3.9.1.3, and Figure 3.3.9.1.4 with stream from Figure 3.3.9.1.5 for the list of inactive users. When the administrator chooses to view one of the accounts at the account details page, the `getInfo` function in Figure 3.3.9.1.5 and Figure 3.3.9.1.6 is called to retrieve the account details from the Firebase server. For the user and helper, the administrator also has privilege in editing the account information such as the account name, phone number, gender, and status, which `TextEditingController`s and forms are used to validate the new values before saving into the Firebase server. Here the administrator can also set the users and helpers' account status to inactive by selecting the icon button provided to trigger the `moveUserToInactive` function shown in Figure 3.3.9.1.7 to migrate the account information from the user collection to the inactive user collection, to not allow them gaining access to the project mobile application. On the other hand, the `setUserToActive` function in Figure 3.3.9.1.8 is used to set the inactive user account

status to active instead by migrating the account information from the inactive user collection to the user collection. Furthermore, the administrator can choose to contact users, helpers, and inactive users through either direct phone call with the phone number saved in the account, or through email communication to the email account using the SMTP service implemented in the project mobile application. If the administrator chooses to contact through direct phone call, the code snippet shown in Figure 3.3.9.1.9 is used to trigger the `launchUrl` function that allows the project to redirect administrator to the mobile device dial screen with the account phone number if the permission to access to mobile phone call is given, else the administrator is unable to be directed to the mobile device dial screen. On the other hand, if the email communication is chosen, a pop-up dialog is displayed to allow administrator to type in the message to be delivered and passed to the `sendUpdateEmail` function shown in Figure 3.3.9.1.10 to use the implemented SMTP service to send the email message from the project mobile application using the default email address and key password added without having to use external email communication platform.

As for the history session management, the administrator can view on the list of history session involving a specific account either being the user account or the helper account in the history session for the users, helpers, and inactive users. If the administrator selects on the user request counter on the account details page, the administrator will be directed to the list of history session page, which the function `getUserRequestHistory` in Figure 3.3.9.1.11 is triggered to retrieve the list of history sessions as list of document snapshots that have the account as the user ID and the order of the list is then set to reverse so that when it is displayed, the list of history session is arranged based on the time added to the server in descending order. On the other hand, if the helper duty counter on the account details page is selected instead, then the `getHelperHelpHistory` function in Figure 3.3.9.1.12 is triggered instead to retrieve the list of history session as list of document snapshots that have the account as the helper ID. At this point, the administrator can also view on the chat history in the history sessions by stream builder as shown in Figure 3.3.9.1.13 to listen to the stream in Figure 3.3.9.1.14 to fetch the history chat conversations in the history sessions. Furthermore, if the administrator decides to delete the history session, the administrator can select the “Delete” icon button provided to trigger the `deleteHistoryDocument` function in Figure 3.3.9.1.15 to

search for the history session in the Firebase server based on the reference ID to then remove it permanently from the Firebase server.

```

47     body: loading
48     ? loadingWait()
49     //use stream to get user list from Firebase
50     : StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
51       stream: FirebaseFirestore.instance
52         .collection('user')
53         .where("email", isNotEqualTo: adminEmail)
54         .where("isHelper", isEqualTo: false)
55         .snapshots(),
56       builder: (context, snapshot) {
57         if (snapshot.connectionState == ConnectionState.waiting) {
58           return loadingWait();
59         } else if (snapshot.connectionState ==
60           ConnectionState.active ||
61           snapshot.connectionState == ConnectionState.done) {
62           if (snapshot.hasData && snapshot.data!.docs.isNotEmpty) {
63             ///show user list tiles
64             return Container(
65               decoration: BoxDecoration(
66                 gradient: LinearGradient(
67                   begin: Alignment.topRight,
68                   end: Alignment.bottomLeft,
69                   stops: const [
70                     0.2,
71                     0.8
72                   ],
73                   colors: [
74                     AdminColour.color2.withOpacity(0.8),
75                     AdminColour.colors5.withOpacity(0.8),
76                   ]), // LinearGradient // BoxDecoration
77               child: ListView.builder(
78                 itemCount: snapshot.data!.size,
79                 itemBuilder: (context, index) => UserListTile(
80                   rawData: snapshot.data!.docs[index].data()));
81             } else if (snapshot.data!.size == 0) {
82               ///if there is no user display no user instead
83               return const NoAvailableUserAccounts();
84             }
85           }
86           return loadingWait();
87         }); // StreamBuilder // Scaffold

```

Figure 3.3.9.1.1 Stream Builder to Fetch List Of Users in list\_of\_users.dart

```

50     : StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
51       stream: FirebaseFirestore.instance
52         .collection('user')
53         .where("email", isNotEqualTo: adminEmail)
54         .where("isHelper", isEqualTo: true)
55         .snapshots(),
56       builder: (context, snapshot) {
57         if (snapshot.connectionState == ConnectionState.waiting) {
58           return loadingWait();
59         } else if (snapshot.connectionState ==
60           ConnectionState.active ||
61           snapshot.connectionState == ConnectionState.done) {
62           if (snapshot.hasData && snapshot.data!.docs.isNotEmpty) {
63             ///display list of helper tiles
64             return Container(
65               color: AdminColour.color1.withOpacity(0.4),
66               child: ListView.builder(
67                 itemCount: snapshot.data!.size,
68                 itemBuilder: (context, index) => HelperTiles(
69                   rawData: snapshot.data!.docs[index].data()));
70             } else if (snapshot.data!.size == 0) {
71               ///if no helper registered then show no helper
72               return const NoAvailableUserAccounts();
73             }
74           }
75           return loadingWait();
76         }); // StreamBuilder // Scaffold

```

Figure 3.3.9.1.2 Stream Builder to Fetch List Of Helpers in list\_of\_helpers.dart

```

108     body: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
109       stream: stream,
110       builder: (context, snapshot) {
111         if (snapshot.connectionState == ConnectionState.waiting ||
112             snapshot.connectionState == ConnectionState.none) {
113           return loadingWait();
114         } else if (snapshot.connectionState == ConnectionState.done ||
115             snapshot.connectionState == ConnectionState.active) {
116           if (snapshot.data!.docs.isNotEmpty &&
117               snapshot.data!.size != 0) {
118             //display list of tiles of inactive users
119             return Container(
120               decoration: const BoxDecoration(
121                 gradient: LinearGradient(
122                   begin: Alignment.bottomLeft,
123                   end: Alignment.topRight,
124                   colors: [
125                     Color.fromARGB(255, 82, 180, 145),
126                     Color.fromARGB(255, 6, 154, 102),
127                   ]), // LinearGradient // BoxDecoration
128               child: ListView.builder(
129                 itemCount: snapshot.data!.size,
130                 itemBuilder: (context, index) {
131                   return InactiveUserTile(
132                     mapData: snapshot.data!.docs[index].data(),
133                   ); // InactiveUserTile
134                 },
135               ); // ListView.builder // Container

```

Figure 3.3.9.1.3 Stream Builder to Fetch List of Inactive Users in `list_of_banned_users.dart`

```

21 Stream<QuerySnapshot<Map<String, dynamic>>> inactiveUserStream() {
22   return FirebaseFirestore.instance.collection('inactiveUser').snapshots();
23 }

```

Figure 3.3.9.1.4 Stream Used to Fetch List of Inactive Users in `list_of_banned_users.dart`

```

30 //get information of user
31 Future getUserInfo() async {
32   setState(() {
33     loading = true;
34   });
35   user = (await UserN().getUserInfo(widget.userEmail));
36   setState(() {
37     loading = false;
38   });
39 }

```

Figure 3.3.9.1.5 `getUserInfo` Function in `admin_view_user_page.dart`

```

99 //retrieve user info
100 Future<UserN?> getUserInfo(String email) async {
101   try {
102     var userInfo =
103       await FirebaseFirestore.instance.collection('user').doc(email).get();
104     if (userInfo.exists) {
105       UserN currentUser = UserN.fromMap(userInfo.data());
106       return currentUser;
107     } else {
108       var userInfo = await FirebaseFirestore.instance
109         .collection('inactiveUser')
110         .doc(email)
111         .get();
112       UserN currentUser = UserN.fromMap(userInfo.data());
113       return currentUser;
114     }
115   } catch (error) {
116     debugPrint(error.toString());
117     return null;
118   }
119 }

```

Figure 3.3.9.1.6 `getUserInfo` Function in `user.dart`

```

85 //when user delete account, save user information in "inactiveUser" collection
86 Future moveUserToInactive(UserN user) async {
87   UserOld setInactiveUser = convertImportantData(user);
88   try {
89     //move user information to "inactiveUser" collection
90     await FirebaseFirestore.instance
91       .collection('inactiveUser')
92       .doc(user.email)
93       .set(setInactiveUser.toMap())
94       .onError((error, stackTrace) =>
95         debugPrint("Set User Inactive Error :${error.toString()}"))
96       .then((value) =>
97         debugPrint("User ${setInactiveUser.email} moved to Inactive"));
98
99     //remove the user information in the
100    await FirebaseFirestore.instance
101      .collection("user")
102      .doc(user.email)
103      .delete()
104      .then((value) =>
105        debugPrint("User ${setInactiveUser.email} removed from active"))
106      .onError((error, stackTrace) =>
107        debugPrint("Error in delete active user: ${error.toString()}"));
108  } on FirebaseException catch (firebaseError) {
109    debugPrint("Set User Inactive Error :${firebaseError.toString()}");
110  }
111 }

```

Figure 3.3.9.1.7 moveUserToInactive Function in inactive\_user.dart

```

148 //set user back to active (only if the user request himself/herself)
149 Future setUserToActive(UserOld oldData, String email) async {
150   UserN user = UserOld().setOldDataToNewUser(oldData);
151   try {
152     await FirebaseFirestore.instance
153       .collection("user")
154       .doc(email)
155       .set(user.toMap())
156       .then((value) async {
157         await UserOld().removeInactiveUser(email);
158       });
159   } on FirebaseException catch (error) {
160     debugPrint("Error in setting user to active : ${error.toString()}");
161   }
162 }

```

Figure 3.3.9.1.8 setUserToActive Function in inactive\_user.dart

```

693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
onPressed: () async {
  bool call =
    await Permission.phone.isGranted;
  if (call) {
    Uri callUser = Uri.parse(
      "tel:${widget.user.phone}");
    if (await launchUrl(callUser)) {
      Fluttertoast.showToast(
        msg:
          "Redirecting to Phone Call");
    } else {
      debugPrint(
        "Error to call ${widget.user.name}");
      if (!mounted) return;
      ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(
          content: Text(
            "Error in trying to contact ${widget.user.name}!"));
    }
  } else {
    if (!mounted) return;
    ScaffoldMessenger.of(context)
      .showSnackBar(const SnackBar(
        content: Text(
          "Please allow phone call permission to continue!")); //
    await Permission.phone.request();
  }
}

```

Figure 3.3.9.1.9 Code Snippet to Direct Call Account User in view\_user\_info\_body.dart and inactive\_user\_info\_page.dart

```

7 Future sendUpdateEmail(
8   String newEmail, String title, String messageText) async {
9   //use vcarecircle email for now (2 factor verification enabled to use app password)
10  const String senderEmail = "vcarecircle2023@gmail.com";
11  String date = DateHandler().parseEmailDate();
12
13  ///set up account for sending email (vcare email with app password (auto-generated))
14  final SmtServer smtpServer = gmail(senderEmail, "hrnyqnhoteeefk");
15
16  ///message content set to be send to the new updated email
17  final message = Message()
18    ..from = const Address(senderEmail, 'VCare Circle')
19    ..recipients.add(newEmail)
20    ..ccRecipients.addAll(['tzhuiyuan75@1utar.my', 'tzhuiyuan74@gmail.com'])
21    ..subject = '$title $date'
22    ..html = "<h2>Hi there $newEmail!!</h2>\n<p>$messageText</p>";
23
24  try {
25    ///command to send out an email
26    final sendReport = await send(message, smtpServer);
27    debugPrint('Email sent to $newEmail: $sendReport');
28  } on MailerException catch (e) {
29    debugPrint('Message not sent. ');
30    debugPrint(e.message);
31    for (var p in e.problems) {
32      debugPrint('Problem: ${p.code}: ${p.msg}');
33    }
34  }

```

Figure 3.3.9.1.10 sendUpdateEmail Function in send\_email\_in\_background.dart

```

288 ///get user request history
289 Future<List<QueryDocumentSnapshot<Map<String, dynamic>>>>
290   getUserRequestHistory(String email) async {
291   var listOfChatRoom = await FirebaseFirestore.instance
292     .collection("history")
293     .where("userId", isEqualTo: email)
294     .get()
295     .catchError((error) {
296       debugPrint("Get Related ChatRoom Error: ${error.toString()}");
297       return null;
298     });
299   if (listOfChatRoom.docs.isNotEmpty) {
300     return listOfChatRoom.docs.reversed.toList();
301   } else {
302     debugPrint("NO Related ChatRoom Found!");
303     return null;
304   }
305 }

```

Figure 3.3.9.1.11 getUserRequestHistory Function in admin.dart

```

307 ///get helper request history
308 Future<List<QueryDocumentSnapshot<Map<String, dynamic>>>>
309   getHelperHelpHistory(String email) async {
310   var listOfChatRoom = await FirebaseFirestore.instance
311     .collection("history")
312     .where("helperId", isEqualTo: email)
313     .get()
314     .catchError((error) {
315       debugPrint("Get Related ChatRoom Error: ${error.toString()}");
316       return null;
317     });
318   if (listOfChatRoom.docs.isNotEmpty) {
319     return listOfChatRoom.docs.reversed.toList();
320   } else {
321     debugPrint("NO Related ChatRoom Found!");
322     return null;
323   }
324 }

```

Figure 3.3.9.1.12 getHelperHelpHistory Function in admin.dart



```

77 body: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
78   stream: stream,
79   builder: (context, snapshot) {
80     if (snapshot.connectionState == ConnectionState.waiting ||
81         snapshot.connectionState == ConnectionState.none) {
82       return loadingWait();
83     } else if (snapshot.connectionState == ConnectionState.active ||
84         snapshot.connectionState == ConnectionState.done) {
85       debugPrint("chats: ${snapshot.data!.docs.length.toString()}");
86
87       //display out the chats
88       return Container(
89         height: MediaQuery.of(context).size.height,
90         width: MediaQuery.of(context).size.width,
91         decoration: const BoxDecoration(
92           gradient: LinearGradient(
93             begin: Alignment.topRight,
94             end: Alignment.bottomLeft,
95             stops: [
96               0.2,
97               0.8
98             ],
99             colors: [
100              AdminColour.color2,
101              AdminColour.colors5,
102            ]), // LinearGradient // BoxDecoration
103         child: snapshot.data!.docs.isNotEmpty
104           ? ListView.builder(
105             controller: controller,
106             physics: const BouncingScrollPhysics(),
107             itemCount: snapshot.data!.docs.length,
108             itemBuilder: (BuildContext context, int index) {
109               return HistoryChatTiles(
110                 time: snapshot.data!.docs[index].data()["time"],
111                 messageContent: snapshot.data!.docs[index]
112                   .data()["contentType"] ==
113                   0
114                   ? snapshot.data!.docs[index].data()["message"]
115                   : snapshot.data!.docs[index]
116                     .data()["photoUrl"],
117                 contentType: snapshot.data!.docs[index]
118                   .data()["contentType"],
119                 sendEmail:
120                   snapshot.data!.docs[index].data()["sendId"],
121                 fromUser:
122                   snapshot.data!.docs[index].data()["sendId"] ==
123                   widget.email
124                   ? true
125                   : false); // HistoryChatTiles
126             }); // ListView.builder

```

Figure 3.3.9.1.13 Stream Builder for Retrieving Chat History in  
admin\_view\_chatroom\_history.dart

```

30 Stream<QuerySnapshot<Map<String, dynamic>>> chatStream(String uid) {
31   return Firestore.instance
32     .collection('history')
33     .doc(uid)
34     .collection('chats')
35     .orderBy("time", descending: false)
36     .snapshots();
37 }

```

Figure 3.3.9.1.14 Stream Set For Retrieving Chat History in  
admin\_view\_chatroom\_history.dart

```

118 //delete history session
119 Future deleteHistoryDocument(String uid) async {
120   try {
121     await Firestore.instance
122       .collection('history')
123       .doc(uid)
124       .delete()
125       .then((value) => debugPrint("History $uid deleted"))
126       .onError((error, stackTrace) =>
127         debugPrint("Delete history document error :${error.toString()}"));
128   } catch (error) {
129     debugPrint("Delete history document $uid failed: ${error.toString()}");
130   }
131 }

```

Figure 3.3.9.1.15 deleteHistoryDocument Function in history\_model.dart

### 3.3.9.2 Application Data Analysis Development

Apart from administrator controls, the administrator module also includes the application data analysis which shows the analysis done on the application data, such as the number of new registered users, application usage by users and helpers, and the overall sessions duration between the users and helpers based on different time intervals. This is to allow the administrator to quickly grasp on the overall user trends in the project mobile application, while also view on the summary of the sessions made between the users and helpers to understand the main factors or problems faced by university students nowadays and what are the high rated answers given by the helpers. Based on the overall session duration time needed, the administrator can also understand the criticality of impact of the problem faced as the longer the time needed to finish the session, the more critical the impact faced. For this purpose, three graphs are added into the project that visualizes these data, which are the bar chart for the number of new registered students, line chart for the application usage by users and helpers, and another line chart for the overall session duration. These charts can change between different time intervals at either for recent three months, recent three years, or the whole present year categorized by months.

For the bar chart of the number of users, first the list of registered users is retrieved from Firebase based on criteria months or years given by the project mobile application whenever the administrator chooses to view the bar chart at different time intervals, for example for viewing months and whole present year, the list is retrieved based on month input, and for years the list is retrieved based on year input. The Figure 3.3.9.2.1 shows the `getNewUserByMonth` function used to retrieve the list by month input, and Figure 3.3.9.2.2 shows the `getNewUserByYear` function used to retrieve the list by year input. After the list is retrieved, the main function returns the list size as integer number indicating the number new registered user found based on the time criteria, which then the data is inputted into the table as shown in the example in Figure 3.3.9.2.3 for the bar chart tables involved. Then, the maximum y-axis value is calculated by comparing for the highest value between the data retrieved and used the `ceilToDouble` function to round up the value to the nearest tens, which then the value is passed to the `returnHighestNumber` function to return the final rounded whole number for the maximum Y-axis value. The example code snippet is shown in Figure 3.3.9.2.4 and

Figure 3.3.9.2.5. Besides, the minimum value is also obtained through comparing of data values to be used as the interval values of the chart, which then the value is passed to the `roundToDouble` function to round up the value to the nearest whole number, which an example is shown in Figure 3.3.9.2.6. When the data is prepared, the `SfCartesianChart` package is used to generate the bar chart with the data input values provided to the chart, which here the administrator can view on the detailed data on the number of new registered users based on different time intervals set. The Figure 3.3.9.2.7 shows the code snippet used to generate the bar chart with the `SfCartesianChart` package.

For the application usage of users and helper, first the list of history sessions is retrieved from Firebase based on the time interval set using the `getNumberOfUsesByMonth` for month intervals, and `getNumberOfUsesByYear` for year intervals, which the functions are shown in Figure 3.3.9.2.8 and Figure 3.3.9.2.9 to get the integer number of the number of usage done by the users and helpers to be passed into the line chart as shown in the example code snippet in Figure 3.3.9.2.10. Then, using the similar comparison method and `returnHighestNumber` function, the maximum Y-axis value is generated to be used in the line chart, as well as the comparison of minimum value and `roundToDouble` function to generate the interval value for the line chart. The Figure 3.3.9.2.11 and Figure 3.3.9.2.12 shows the example code snippet in getting the maximum y-axis value and interval value. Figure 3.3.9.2.13 shows the code snippet of the `SfCartesianChart` package used to generate the line chart.

As for the overall session duration, the list of history sessions is first being retrieved from Firebase based on time interval set, which here the calculation of duration is done by summing all the duration obtained from the list of history sessions to then be divided by the number of history sessions found, to obtain the average of time duration in seconds, which the Figure 3.3.9.2.14 shows the function for retrieving overall duration based on month input interval, and Figure 3.3.9.2.15 shows the retrieving overall duration based on year input interval. When the required overall session duration is obtained, the data is then converted into minutes before adding to the chart data as shown in Figure 3.3.9.2.16 for the purpose of more organized Y-axis value. As same as the previous chart, the comparison of the data values is done for getting the maximum

Y-axis value and also the minimum value as interval value for the line chart, which is shown in Figure 3.3.9.2.17 and Figure 3.3.9.2.18. Finally, the line graph for the overall session duration is generated after passing the values to the SfCartesianChart package widget, which the code snippet used is shown in Figure 3.3.9.2.19.

Furthermore, the capability to search for the related session based on the longest and shortest duration found based on either month or year input is added for the overall session duration analysis for the purpose of allowing administrator to investigate the history sessions involved. When the administrator long presses the value points of the line graph for the overall session duration line graph when displaying the month charts, the `getListOfHighestSession` function showed in Figure 3.3.9.2.20 is triggered is first search for the longest duration with the help of the `getLongestDuration` function in Figure 3.3.9.2.21 by comparing the duration of the session found based on the month selected for this year, which then the returned longest duration value in seconds is used to set a range of minimum the shortest duration minus thirty seconds to the maximum of shortest duration plus thirty seconds. The range is then used to locate the list of sessions which have the duration within the range set and saving the session ID. When completed, the list of session ID is returned to then be used to get the details of the sessions to display to the administrator. The same theory is applied in finding the list of session ID relating to the shortest duration, which the function `getListOfShortestSession` in Figure 3.3.9.2.22 is used to locate the list of sessions related to the shortest duration, together with the help of `getShortestSessionDuration` in Figure 3.3.9.2.23 to compare out the shortest session duration based on the month interval. On the other hand, if the value point of the line graph for the overall session duration line graph when displaying the years chart is long pressed, the `getListOfHighestSession` function in Figure 3.3.9.2.24 is triggered instead to retrieve the list of session related to the longest session based on the duration identified by the `getLongestDurationInYear` function shown in Figure 3.3.9.2.25 by comparing out the longest session duration found based on the year input, which the list of longest session ID is retrieved to use it to retrieve the sessions' information to be displayed to the administrator. Similarly for the shortest duration, the `getListOfShortestSessionInYear` in Figure 3.3.9.2.26 and the `getShortestSessionDurationInYear` in Figure 3.3.9.2.27 are used to locate the list of session ID related to the shortest duration based on year input.

```

78  ///get number of new registered user based on month input
79  Future<int> getNewUserByMonth(int month) async {
80    int count = 0;
81    var snapshot = await FirebaseFirestore.instance.collection("user").get();
82    if (snapshot.docs.isNotEmpty) {
83      for (var doc in snapshot.docs) {
84        if (DateHandler().parseHistoryDate(doc["joinDate"]).month == month) {
85          count += 1;
86        }
87      }
88      return count;
89    } else {
90      debugPrint("Error in getting user this month!");
91      return 0;
92    }
93  }

```

Figure 3.3.9.2.1 getNewUserByMonth Function in app\_data\_controls.dart

```

95  ///get number of new registered user based on year input
96  Future<int> getNewUserByYear(int year) async {
97    int count = 0;
98    var snapshot = await FirebaseFirestore.instance.collection("user").get();
99    if (snapshot.docs.isNotEmpty) {
100     for (var doc in snapshot.docs) {
101       if (DateHandler().parseHistoryDate(doc["joinDate"]).year == year) {
102         count += 1;
103       }
104     }
105     return count;
106   } else {
107     debugPrint("Error in getting user this month!");
108     return 0;
109   }
110 }

```

Figure 3.3.9.2.2 getNewUserByYear Function in app\_data\_controls.dart

```

54  ///get required data
55  getUserThisMonth = await AppDataControls()
56    .getNewUserByMonth(DateHandler().getThisMonthInInt());
57  getUserLastMonth = await AppDataControls()
58    .getNewUserByMonth(DateHandler().getLastMonthInInt());
59  getUserLast2Month = await AppDataControls()
60    .getNewUserByMonth(DateHandler().getLast2MonthInInt());
61  //set data to table chart
62  data = [
63    AppUserChartData(
64      label: DateHandler().getLast2Month(), count: getUserLast2Month), // AppUserChartData
65    AppUserChartData(
66      label: DateHandler().getLastMonth(), count: getUserLastMonth), // AppUserChartData
67    AppUserChartData(
68      label: DateHandler().getCurrentMonth(), count: getUserThisMonth), // AppUserChartData
69  ];

```

Figure 3.3.9.2.3 Code Snippet Example To Input Data to Chart in app\_user\_chart.dart

```

71  getMax = 0;
72  ///get highest value (as table highest y-value)
73  if (getUserThisMonth > getUserLastMonth) {
74    getMax = getUserThisMonth.ceilToDouble();
75  } else if (getUserLastMonth > getUserLast2Month) {
76    getMax = getUserLastMonth.ceilToDouble();
77  } else {
78    getMax = getUserLast2Month.ceilToDouble();
79  }
80  //return as highest value rounded to tens
81  getMax = returnHighestNumber(getMax);

```

Figure 3.3.9.2.4 Code Snippet Example To Find Maximum Y-Axis Value in app\_user\_chart.dart

```

24 //Round the value to the highest whole number (as ten)
25 double returnHighestNumber(double number) {
26   debugPrint("number is $number");
27   double a = (number / 10).ceilToDouble() * 10;
28   debugPrint("A is $a");
29   double b = a + 10;
30   debugPrint("B is $b");
31   double compare1 = number - a;
32   debugPrint("Compare 1: $compare1");
33
34   double compare2 = b - number;
35   debugPrint("Compare 2: $compare2");
36   double finalValue = (compare1 > compare2) ? b : a;
37   if (finalValue % 10 == 0) {
38     finalValue = finalValue + 10;
39     debugPrint(" (1)Final Value $finalValue");
40   } else {
41     finalValue = (finalValue + 5).ceilToDouble();
42     debugPrint(" (2)Final Value $finalValue");
43   }
44   return finalValue;
45 }

```

Figure 3.3.9.2.5 returnHighestNumber in data\_model.dart

```

83 //get lowest value (as table interval count)
84 if (getUserThisMonth < getUserLastMonth &&
85     getUserThisMonth < getUserLast2Month) {
86   intervalCount = getUserThisMonth.roundToDouble();
87 } else if (getUserLastMonth < getUserLast2Month &&
88     getUserLastMonth < getUserThisMonth) {
89   intervalCount = getUserLastMonth.roundToDouble();
90 } else {
91   intervalCount = getUserLast2Month.roundToDouble();
92 }
93 if (intervalCount == 0) {
94   intervalCount = 2;
95 } else if (intervalCount > 10) {
96   intervalCount = (intervalCount / 10).floor() * 10;
97 }

```

Figure 3.3.9.2.6 Code Snippet Example To Find Interval Value in app\_user\_chart.dart

```

379 //display chart
380 child: SfCartesianChart(
381   //display table title and legend
382   title: ChartTitle(text: "Registered Users Count"),
383   legend: Legend(isVisible: true),
384   //x-axis design
385   primaryXAxis: CategoryAxis(
386     labelStyle: const TextStyle(
387       fontSize: 16,
388       fontWeight: FontWeight.bold,
389       color: AdminColour.color5,
390     ), // TextStyle
391     edgeLabelPlacement: EdgeLabelPlacement.shift,
392     labelPosition: ChartDataLabelPosition.outside,
393     rangePadding: ChartRangePadding.auto), // Category
394
395   //y-axis design
396   primaryYAxis: NumericAxis(
397     labelStyle: const TextStyle(
398       fontSize: 14,
399       fontWeight: FontWeight.bold,
400       color: AdminColour.color3,
401     ), // TextStyle
402     minimum: 0,
403     decimalPlaces: 0,
404     maximum: getMax,
405     interval: intervalCount), // NumericAxis
406   tooltipBehavior: tooltip,
407   //bar chart
408   series: <ChartSeries<AppUserChartData, String>>[
409     ColumnSeries<AppUserChartData, String>(
410       dataSource: data,
411       xValueMapper: (AppUserChartData data, _) =>
412         data.label,
413       yValueMapper: (AppUserChartData data, _) =>
414         data.count,
415       name: "Number of Users",
416       color: AdminColour.color2,
417     ), // ColumnSeries
418   ], // <ChartSeries<AppUserChartData, String>>[] // Sf
419 ) // SizedBox

```

Figure 3.3.9.2.7 Code Snippet For Bar Chart in app\_user\_chart.dart

```

112 ///get number of session opened based on month input
113 Future<int> getNumberOfUsesByMonth(int month) async {
114   try {
115     int count = 0;
116
117     var snapshot =
118       await FirebaseFirestore.instance.collection("history").get();
119     if (snapshot.docs.isNotEmpty) {
120       for (var doc in snapshot.docs) {
121         if (DateHandler().parseHistoryDate(doc["date"]).month == month &&
122             DateHandler().parseHistoryDate(doc["date"]).year ==
123             DateHandler().getYearAsInt(0)) {
124           count += 1;
125         }
126       }
127       return count;
128     } else {
129       return 0;
130     }
131   } catch (error) {
132     debugPrint("Get Number of Uses ($month) Error ${error.toString()}");
133   }
134   return 0;
135 }

```

Figure 3.3.9.2.8 getNumberOfUsesByMonth Function in app\_data\_controls.dart

```

137 ///get number of session opened based on year input
138 Future<int> getNumberOfUsesByYear(int year) async {
139   try {
140     int count = 0;
141
142     var snapshot =
143       await FirebaseFirestore.instance.collection("history").get();
144     if (snapshot.docs.isNotEmpty) {
145       for (var doc in snapshot.docs) {
146         if (DateHandler().parseHistoryDate(doc["date"]).year == year) {
147           count += 1;
148         }
149       }
150       return count;
151     } else {
152       return 0;
153     }
154   } catch (error) {
155     debugPrint(
156       "Get Number of Uses (In Year $year) Error ${error.toString()}");
157   }
158   return 0;
159 }

```

Figure 3.3.9.2.9 getNumberOfUsesByYear Function in app\_data\_controls.dart

```

53 ///get required data
54 getThisMonth = await AppDataControls()
55   .getNumberOfUsesByMonth(DateHandler().getThisMonthInInt());
56 getLastMonth = await AppDataControls()
57   .getNumberOfUsesByMonth(DateHandler().getLastMonthInInt());
58 getLast2Month = await AppDataControls()
59   .getNumberOfUsesByMonth(DateHandler().getLast2MonthInInt());
60 debugPrint(
61   "Last 2 Month $getLast2Month, Last Month $getLastMonth, This Month $getThisMonth");
62
63 ///set data to table chart
64 data = [
65   AppUsageChartData(
66     label: DateHandler().getLast2Month(), count: getLast2Month), // AppUsageChartData
67   AppUsageChartData(
68     label: DateHandler().getLastMonth(), count: getLastMonth), // AppUsageChartData
69   AppUsageChartData(
70     label: DateHandler().getCurrentMonth(), count: getThisMonth), // AppUsageChartData
71 ];

```

Figure 3.3.9.2.10 Code Snippet Example To Input Data to Chart in app\_usage\_chart.dart

```

133 getMax = 0;
134 if (getThisYear > getLastYear) {
135   getMax = getThisYear.ceilToDouble();
136 } else if (getLastYear > getLast2Year) {
137   getMax = getLastYear.ceilToDouble();
138 } else {
139   getMax = getLast2Year.ceilToDouble();
140 }
141 //return highest value as rounded tens
142 getMax = returnHighestNumber(getMax);

```

Figure 3.3.9.2.11 Code Snippet Example To Find Maximum Y-Axis Value in app\_usage\_chart.dart

```

144 //get lowest value as the interval count
145 if (getThisYear < getLastYear && getThisYear < getLast2Year) {
146   intervalCount = getThisYear.roundToDouble();
147 } else if (getLastYear < getLast2Year && getLastYear < getThisYear) {
148   intervalCount = getLastYear.roundToDouble();
149 } else {
150   intervalCount = getLast2Year.roundToDouble();
151 }
152 if (intervalCount == 0) {
153   intervalCount = 2;
154 } else if (intervalCount > 10) {
155   intervalCount = (intervalCount / 10).floor() * 10;
156 }

```

Figure 3.3.9.2.12 Code Snippet Example To Find Interval Value in app\_usage\_chart.dart

```

387 child: SfCartesianChart(
388   //table title and legend
389   title: ChartTitle(text: "Total Usage Count"),
390   legend: Legend(isVisible: true),
391   //trackball design settings
392   trackballBehavior: TrackballBehavior(
393     enable: true,
394     activationMode: ActivationMode.longPress,
395     tooltipSettings: InteractiveTooltip(
396       enable: true,
397       textStyle: const TextStyle(
398         fontSize: 14,
399         color: Colors.white,
400         fontWeight: FontWeight.bold,
401       ), // TextStyle
402       color: AdminColour.color1.withOpacity(0.7)),
403   //x-axis design
404   primaryXAxis: CategoryAxis(
405     labelStyle: const TextStyle(
406       fontSize: 16,
407       fontWeight: FontWeight.bold,
408       color: AdminColour.color3,
409     ), // TextStyle
410     edgeLabelPlacement: EdgeLabelPlacement.shift,
411     labelPosition: ChartDataLabelPosition.outside,
412     rangePadding: ChartRangePadding.auto), // Category
413   //y-axis design
414   primaryYAxis: NumericAxis(
415     labelStyle: const TextStyle(
416       fontSize: 14,
417       fontWeight: FontWeight.bold,
418       color: AdminColour.color3,
419     ), // TextStyle
420     minimum: 0,
421     maximum: getMax,
422     interval: intervalCount,
423     edgeLabelPlacement: EdgeLabelPlacement.shift), // N
424   tooltipBehavior: tooltip,
425
426   //line chart
427   series: <ChartSeries<AppUsageChartData, String>>[
428     LineSeries<AppUsageChartData, String>(
429       animationDuration: 4000,
430       markerSettings: const MarkerSettings(
431         isVisible: true,
432         color: AdminColour.color1,
433         height: 12,
434         width: 12), // MarkerSettings
435       dataLabelSettings: const DataLabelSettings(
436         isVisible: true, showZeroValue: false), // Data
437       dataSource: data,
438       xValueMapper: (AppUsageChartData data, _) =>
439         data.label,
440       yValueMapper: (AppUsageChartData data, _) =>
441         data.count,
442       name: "Number of Sessions",
443       color: AdminColour.color3,
444     ) // LineSeries
445   ]) // <ChartSeries<AppUsageChartData, String>>[] // S
446 ]), // ListView // Container

```

Figure 3.3.9.2.13 Code Snippet For Line Chart in app\_usage\_chart.dart



```

20 Future<int> getOverallUseTimeByMonth(int month) async {
21   int total = 0;
22   int duration = 0;
23   int getAverage = 0;
24   String currentMonth = DateHandler().getMonthInString(month);
25   var snapshot = await FirebaseFirestore.instance.collection("history").get();
26   if (snapshot.size != 0) {
27     for (var doc in snapshot.docs) {
28       if (doc.data().isNotEmpty) {
29         DateTime getDate = DateHandler().parseHistoryDate(doc.data()["date"]);
30         String getHistoryMonth = DateHandler().changeToMonthString(getDate);
31         if (getHistoryMonth == currentMonth &&
32             getDate.year == DateTime.now().year) {
33           duration += int.parse(doc.data()["duration"].toString());
34           total += 1;
35         }
36       }
37     }
38     if (duration > 0) {
39       getAverage = (duration / total).round();
40     } else {
41       getAverage = 0;
42     }
43     return getAverage;
44   } else {
45     return 0;
46   }
47 }

```

Figure 3.3.9.2.14 getOverallUseTimeByMonth in app\_data\_controls.dart

```

50 Future<int> getOverallUseTimeByYear(int year) async {
51   int total = 0;
52   int duration = 0;
53   int getAverage = 0;
54   var snapshot = await FirebaseFirestore.instance.collection("history").get();
55   if (snapshot.size != 0) {
56     for (var doc in snapshot.docs) {
57       if (doc.data().isNotEmpty) {
58         DateTime getDate = DateHandler().parseHistoryDate(doc.data()["date"]);
59         if (getDate.year == year) {
60           duration += int.parse(doc.data()["duration"].toString());
61           total += 1;
62           debugPrint("Duration: $duration");
63         }
64       }
65     }
66     if (duration > 0) {
67       getAverage = (duration / total).round();
68       debugPrint("Total: $total");
69     } else {
70       getAverage = 0;
71     }
72     return getAverage;
73   } else {
74     return 0;
75   }
76 }

```

Figure 3.3.9.2.15 getOverallUseTimeByYear in app\_data\_controls.dart

```

70 //get the required data
71 getDurationThisMonth = changeDurationToMinutes(await AppDataControls()
72   .getOverallUseTimeByMonth(DateHandler().getThisMonthInInt()));
73 getDurationLastMonth = changeDurationToMinutes(await AppDataControls()
74   .getOverallUseTimeByMonth(DateHandler().getLastMonthInInt()));
75 getDurationLast2Month = changeDurationToMinutes(await AppDataControls()
76   .getOverallUseTimeByMonth(DateHandler().getLast2MonthInInt()));
77 //set the data into the table chart data
78 data = [
79   AppDurationChartData(
80     label: DateHandler().getLast2Month(), minutes: getDurationLast2Month),
81   AppDurationChartData(
82     label: DateHandler().getLastMonth(), minutes: getDurationLastMonth),
83   AppDurationChartData(
84     label: DateHandler().getCurrentMonth(),
85     minutes: getDurationThisMonth), // AppDurationChartData
86 ];

```

Figure 3.3.9.2.16 Code Snippet Example To Input Data To Chart in app\_overall\_session\_duration.dart

```

88     getMax = 0;
89     //get the highest value of the data (set as the table maximum value)
90     if (getDurationThisMonth > getDurationLastMonth) {
91         getMax = getDurationThisMonth.ceilToDouble();
92     } else if (getDurationLastMonth > getDurationLast2Month) {
93         getMax = getDurationLastMonth.ceilToDouble();
94     } else {
95         getMax = getDurationLast2Month.ceilToDouble();
96     }
97     //return the highest level after rounding up to the tens
98     getMax = returnHighestNumber(getMax);

```

Figure 3.3.9.2.17 Code Snippet Example To Find Maximum Y-Axis Value in app\_overall\_session\_duration.dart

```

99     //get the lowest value of the data (set as the table interval count)
100    if (getDurationThisMonth < getDurationLastMonth &&
101        getDurationThisMonth < getDurationLast2Month) {
102        intervalCount = getDurationThisMonth.roundToDouble();
103    } else if (getDurationLastMonth < getDurationLast2Month &&
104        getDurationLastMonth < getDurationThisMonth) {
105        intervalCount = getDurationLastMonth.roundToDouble();
106    } else {
107        intervalCount = getDurationLast2Month.roundToDouble();
108    }
109    if (intervalCount == 0) {
110        intervalCount = 2;
111    } else if (intervalCount > 10) {
112        intervalCount = (intervalCount / 10).floor() * 10;
113    }

```

Figure 3.3.9.2.18 Code Snippet Example To Find Interval Value in app\_overall\_session\_duration.dart

```

438 child: SfCartesianChart(
439
440     ///title and legend of table
441     title: ChartTitle(text: "Overall Session Duration"),
442     legend: Legend(isVisible: true),
443     ///the line tracker to view data
444     trackballBehavior: TrackballBehavior(
445       enable: true,
446       activationMode: ActivationMode.longPress,
447
448       ///set the viewing label design
449       tooltipSettings: InteractiveTooltip(
450         enable: true,
451         textStyle: const TextStyle(
452           fontSize: 14,
453           color: Colors.white,
454           fontWeight: FontWeight.bold,
455         ), // TextStyle
456         color: AdminColour.color1.withOpacity(0.7)),
457
458     ///x-axis design
459     primaryXAxis: CategoryAxis(
460       labelStyle: const TextStyle(
461         fontSize: 16,
462         fontWeight: FontWeight.bold,
463         color: AdminColour.color3,
464       ), // TextStyle
465       edgeLabelPlacement: EdgeLabelPlacement.shift,
466       labelPosition: ChartDataLabelPosition.outside,
467       rangePadding: ChartRangePadding.auto), // Category
468
469     ///y-axis design
470     primaryYAxis: NumericAxis(
471       labelStyle: const TextStyle(
472         fontSize: 14,
473         fontWeight: FontWeight.bold,
474         color: AdminColour.color3,
475       ), // TextStyle
476       decimalPlaces: 0,
477       minimum: 0,
478       maximum: getMax,
479       interval: intervalCount,
480       edgeLabelPlacement: EdgeLabelPlacement.shift), // N
481
482     tooltipBehavior: tooltip,
483     ///line chart
484     series: <ChartSeries<AppDurationChartData, String>>[
485       LineSeries<AppDurationChartData, String>(
486         animationDuration: 4000,
487         markerSettings: const MarkerSettings(
488           isVisible: true,
489           color: AdminColour.color1,
490           height: 12,
491           width: 12), // MarkerSettings
492
493         ///set the value point design
494         dataLabelSettings: const DataLabelSettings(
495           margin: EdgeInsets.all(20),
496           overflowMode: OverflowMode.shift,
497           labelAlignment: ChartDataLabelAlignment.auto,
498           isVisible: true,
499           showZeroValue: false,
500           textStyle: TextStyle(
501             fontSize: 15,
502             color: AdminColour.color4,
503             fontWeight: FontWeight.bold,
504           ), // TextStyle // DataLabelSettings
505
506         dataSource: data!,
507         xValueMapper: (AppDurationChartData data, _) =>
508           data.label,
509         yValueMapper: (AppDurationChartData data, _) =>
510           data.minutes,
511         name: "Session Duration",
512         color: AdminColour.color1,
513
514         ///view the details of the longest, shortest, overa
515         onPointLongPress: (ChartPointDetails
516           pointInteractionDetails) async {
517           debugPrint(pointInteractionDetails.pointIndex
518             .toString());
519           AppDurationChartData getData =
520             data![pointInteractionDetails.pointIndex!];
521           showDialog(
522             context: context,
523             barrierDismissible: false,
524             builder: (context) => loadingWait(),
525           );
526           await showDetailsOfPoint(
527             context, whichTable, getData);
528           if (!mounted) return;
529           Navigator.of(context).pop();
530         },
531       ) // LineSeries
532     ]) // <ChartSeries<AppDurationChartData, String>>[ /

```

Figure 3.3.9.2.19 Code Snippet For Line Chart in app\_overall\_session\_duration.dart

```

190 Future<List<String>> getListOfHighestSessions(int month) async {
191   try {
192     List<String> sessionList = [];
193     int longestDuration = await getLongestSessionDuration(month);
194     var snapshotList =
195       await FirebaseFirestore.instance.collection("history").get();
196     if (snapshotList.docs.isNotEmpty) {
197       for (var doc in snapshotList.docs) {
198         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
199
200         ///check for history in this year and input month
201         if (historyDate.year == DateHandler().getYearAsInt(0) &&
202             historyDate.month == month) {
203           int sessionDuration = int.parse(doc.data()["duration"].toString());
204           if (sessionDuration >= longestDuration - 30 &&
205               sessionDuration <= longestDuration + 30) {
206             sessionList.add(doc.reference.id);
207           }
208         }
209       }
210     }
211     return sessionList;
212   } else {
213     return [];
214   }
215 } catch (error) {
216   debugPrint(
217     "Get Longest Session List (In $month 2023) Error ${error.toString()}");
218   return [];
219 }

```

Figure 3.3.9.2.20 getListOfHighestSession Function in app\_data\_controls.dart

```

161 // get the longest duration that can be found this year based on month
162 Future<int> getLongestSessionDuration(int month) async {
163   try {
164     int highestDuration = 0;
165     var snapshotList =
166       await FirebaseFirestore.instance.collection("history").get();
167     if (snapshotList.docs.isNotEmpty) {
168       for (var doc in snapshotList.docs) {
169         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
170         if (historyDate.year == DateHandler().getYearAsInt(0) &&
171             historyDate.month == month) {
172           if (int.parse(doc.data()["duration"].toString()) >
173               highestDuration) {
174             highestDuration = int.parse(doc.data()["duration"].toString());
175           }
176         }
177       }
178     }
179     return highestDuration;
180   } else {
181     return 0;
182   }
183 } catch (error) {
184   debugPrint(
185     "Get Longest Session Details (In $month 2023) Error ${error.toString()}");
186   return 0;
187 }

```

Figure 3.3.9.2.21 getLongestDuration Extension Function in app\_data\_controls.dart

```

310 //get the list of history uid based on the range of duration similar to the short
311 Future<List<String>> getListOfShortestSessions(int month) async {
312   try {
313     List<String> sessionList = [];
314     int shortestDuration = await getShortestSessionDuration(month);
315     var snapshotList =
316       await FirebaseFirestore.instance.collection("history").get();
317     if (snapshotList.docs.isNotEmpty) {
318       for (var doc in snapshotList.docs) {
319         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
320
321         //check for history in this year and input month
322         if (historyDate.year == DateHandler().getYearAsInt(0) &&
323             historyDate.month == month) {
324           int sessionDuration = int.parse(doc.data()["duration"].toString());
325           if [(sessionDuration >= shortestDuration - 30 &&
326               sessionDuration <= shortestDuration + 30)] {
327             sessionList.add(doc.reference.id);
328           }
329         }
330       }
331       return sessionList;
332     } else {
333       return [];
334     }
335   } catch (error) {
336     debugPrint(
337       "Get Shortest Session List (In $month 2023) Error ${error.toString()}");
338     return [];
339   }
340 }
341

```

Figure 3.3.9.2.22 getListOfShortestSessions Function in app\_data\_controls.dart

```

277 // get the shortest duration found based on month input
278 Future<int> getShortestSessionDuration(int month) async {
279   try {
280     //set the largest int possible as a default constant to be compared for the shortest duration
281     int shortestDuration = 999999999999999999;
282     bool changed = false;
283     var snapshotList =
284       await FirebaseFirestore.instance.collection("history").get();
285     if (snapshotList.docs.isNotEmpty) {
286       for (var doc in snapshotList.docs) {
287         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
288         if (historyDate.year == DateHandler().getYearAsInt(0) &&
289             historyDate.month == month) {
290           if (int.parse(doc.data()["duration"].toString()) <
291               shortestDuration) {
292             shortestDuration = int.parse(doc.data()["duration"].toString());
293             changed = true;
294           }
295         }
296       }
297       if (!changed) return 0;
298       return shortestDuration;
299     } else {
300       debugPrint("No Shortest Session Found");
301       return 0;
302     }
303   } catch (error) {
304     debugPrint(
305       "Get Shortest Session Details (In $month 2023) Error ${error.toString()}");
306     return 0;
307   }
308 }

```

Figure 3.3.9.2.23 getShortestSessionDuration Extension Function in app\_data\_controls.dart

```

248 // get the list of history uid based on the range of duration similar to the long
249 Future<List<String>> getListOfHighestSessionsInYear(int year) async {
250   try {
251     List<String> sessionList = [];
252     int longestDuration = await getLongestSessionDurationInYear(year);
253     var snapshotList =
254       await FirebaseFirestore.instance.collection("history").get();
255     if (snapshotList.docs.isNotEmpty) {
256       for (var doc in snapshotList.docs) {
257         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
258         if (historyDate.year == year) {
259           int sessionDuration = int.parse(doc.data()["duration"].toString());
260           if (sessionDuration >= longestDuration - 30 &&
261               sessionDuration <= longestDuration + 30) {
262             sessionList.add(doc.reference.id);
263           }
264         }
265       }
266       return sessionList;
267     } else {
268       return [];
269     }
270   } catch (error) {
271     debugPrint(
272       "Get Longest Session List (In $year) Error ${error.toString()}");
273     return [];
274   }
275 }

```

Figure 3.3.9.2.24 getListOfHighestSessionsInYear Function in app\_data\_controls.dart

```

221 // get the longest duration that can be found based on year
222 Future<int> getLongestSessionDurationInYear(int year) async {
223   try {
224     int highestDuration = 0;
225     var snapshotList =
226       await FirebaseFirestore.instance.collection("history").get();
227     if (snapshotList.docs.isNotEmpty) {
228       for (var doc in snapshotList.docs) {
229         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
230         if (historyDate.year == year) {
231           if (int.parse(doc.data()["duration"].toString()) >
232               highestDuration) {
233             highestDuration = int.parse(doc.data()["duration"].toString());
234           }
235         }
236       }
237       return highestDuration;
238     } else {
239       return 0;
240     }
241   } catch (error) {
242     debugPrint(
243       "Get Longest Session Details (In $year) Error ${error.toString()}");
244     return 0;
245   }
246 }

```

Figure 3.3.9.2.25 getHighestSessionDurationInYear Function in app\_data\_controls.dart

```

372 // get the list of history uid based on the range of duration similar to the sho
373 Future<List<String>> getListOfShortestSessionsInYear(int year) async {
374   try {
375     List<String> sessionList = [];
376     int shortestDuration = await getShortestSessionDurationInYear(year);
377     var snapshotList =
378       await FirebaseFirestore.instance.collection("history").get();
379     if (snapshotList.docs.isNotEmpty) {
380       for (var doc in snapshotList.docs) {
381         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
382         if (historyDate.year == year) {
383           int sessionDuration = int.parse(doc.data()["duration"].toString());
384           if (sessionDuration >= shortestDuration - 30 &&
385               sessionDuration <= shortestDuration + 30) {
386             sessionList.add(doc.reference.id);
387           }
388         }
389       }
390       return sessionList;
391     } else {
392       return [];
393     }
394   } catch (error) {
395     debugPrint(
396       "Get Shortest Session List (In $year) Error ${error.toString()}");
397     return [];
398   }
399 }

```

Figure 3.3.9.2.26 getListOfShortestSessionsInYear Function in app\_data\_controls.dart

```

342 // get the shortest duration that can be found based on year
343 Future<int> getShortestSessionDurationInYear(int year) async {
344   try {
345     int shortestDuration = 999999999999999999;
346     bool changed = false;
347     var snapshotList =
348       await FirebaseFirestore.instance.collection("history").get();
349     if (snapshotList.docs.isNotEmpty) {
350       for (var doc in snapshotList.docs) {
351         DateTime historyDate = DateHandler().parseHistoryDate(doc["date"]);
352         if (historyDate.year == year) {
353           if (int.parse(doc.data()["duration"].toString()) <
354               shortestDuration) {
355             shortestDuration = int.parse(doc.data()["duration"].toString());
356             changed = true;
357           }
358         }
359       }
360       if (!changed) return 0;
361       return shortestDuration;
362     } else {
363       return 0;
364     }
365   } catch (error) {
366     debugPrint(
367       "Get Shortest Session Details (In $year) Error ${error.toString()}");
368     return 0;
369   }
370 }

```

Figure 3.3.9.2.27 getShortestSessionDurationInYear Extension Function in app\_data\_controls.dart

### **3.4 Summary**

In this chapter, the use case diagram which illustrates the functionalities available in using the project mobile application is included along with use case descriptions to allow easy understanding and clear overview of the overall functionalities provided in the application, and showing how the functionalities are connected to each other to perform as a functional module in the project mobile application. The project development process is also discussed to explain on the concepts behind developing the project mobile application and functional modules by showing code snippets from the project and explaining the important components used in developing the functional modules.



## CHAPTER 4

### Methodology and Tools Used

#### 4.1 Overview

This chapter includes the discussion of the system methodology used to develop this project, and the listing of the hardware and software development tools used in developing this project which includes the brief explanation of what the development tools chosen are used for. For effective and efficient time management in developing this project, the project timeline is also planned to allocate time to finish each task on time so that the project can be completed within deadline.

#### 4.2 Methodology

This section discusses on the methodology used to develop this project. For this project, the system methodology used is the Rapid Application Development (RAD) methodology. This agile type of project management methodology focuses on incremental and iterative development, which mainly focuses on developing workable prototypes for users to test the system and getting feedback from them to continue improving the working prototypes until the final working models are complete. This gives a better flow of development with better time management through the adjustment of the System Development Life Cycle (SDLC) and allow focus on user testing feedback for project during development of the project [13]. The advantages of applying this methodology in this project are it gives more flexibility, allow changes in each phase, encourages faster development time, able to quickly identify design flaws at early stage, and produce a better quality project that focuses on satisfying the users' requirement on project mobile application [14]. From the RAD methodology, the project is divided into 4 main phases, which are requirement planning phase, user design phase, construction phase and lastly the cutover phase.

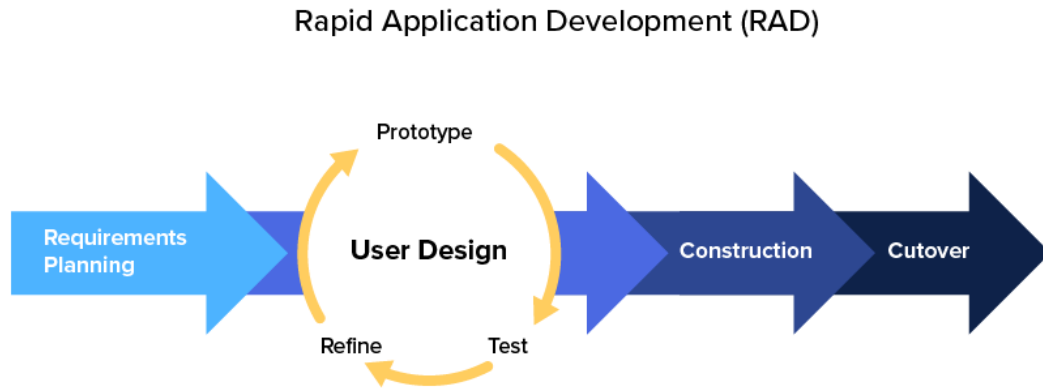


Figure 4.2.1 Rapid Application Development Methodology [15]

#### 4.2.1 Requirement Planning Phase

In the requirement planning phase, the idea of developing the mobile application of VCare Circle is planned and proposed. Literature review on similar works is done to have an overview of the basic requirements required for the project mobile application and what are the user expectations when using the application. From here, the comparison of the strength and weaknesses is done to have better accurate information for setting the project scope and formulate the project objectives that can satisfy the user requirements, and to avoid unwanted mistakes that affects user's experience to be included into the project based on the weaknesses. Multiple discussions are carried out to ensure the project's complexity and functional modules can be planned more systematically and effective, while conducting the time scheduling of the project development by creating a project timeline to ensure the project prototype can be done before deadline. The hardware and software development tools are also identified.

#### 4.2.2 Design Phase

For the design phase, the development of the project mobile application prototype function modules is done such as the building of user interface and implementing basic functionalities to the project mobile application. For the development of this project mobile application, the tools used are the Visual Studio Code as the project's integrated development environment (IDE), Flutter as the framework tool for developing the user interface, and the use of Dart language to build the functionalities for the application. During the development of the project mobile application, the debugging tests of the project mobile application is done using the Android emulator from Android Studio

and real Android mobile phone device to stimulate real use case scenarios. Multiple iterations are carried out in this phase for continuous modifying and improving of the design of the project mobile application to meet the requirements [16]. This phase continues until the basic structure of the user interface and basic functionalities prototypes is finalized and completed.

### **4.2.3 Construction Phase**

In the construction phase, this is where the enhanced work on the prototype is done to develop actual working models for the project. From here, the Firebase is selected as the main backend server for the project, and the improvement of user interface design and functional modules is done with Flutter and Dart. This phase is done to ensure the complexity of the project and crucial functions of the application is suitable for actual use cases. In this process, modification to improve the working models is done based on the feedbacks and suggestions. This phase is carried out until all working modules are implemented to the project mobile application with reaching the user requirements and passed the correctness checking for the coding of the functionalities [16].

### **4.2.4 Cutover Phase**

Cutover phase is the final phase for the development of the project mobile application, which includes the writing of documentation and implementing of the project mobile application. During this phase, final checking on the project mobile application is done such as system level testing and compatibility testing to ensure the overall usability and complexity of the project mobile application's functional modules are stable and complete. After testing, the project's documentation is written to provide full definition of the project mobile application and explanation on the system design and functional modules included, along with the work done for the project mobile application.

### 4.3 Development Tools Used

#### 4.3.1 Hardware Requirements

Table 4.3.1.1 Specifications of Laptop for Development of Project

<b>Description</b>	<b>Specifications</b>
<b>Model</b>	Lenovo Legion Y545
<b>Processor</b>	Intel(R) Core i7-9750H CPU @ 2.60GHz
<b>Operating System</b>	Windows 11
<b>Graphic</b>	NVIDIA GeForce RTX 2060 4GB DDR4
<b>Memory</b>	8GB DDR4 RAM
<b>Storage</b>	1TB SSD

Table 4.3.1.2 Specifications of Mobile Device for Testing of Project

<b>Description</b>	<b>Specifications</b>
<b>Model</b>	Huawei Mate 20 Pro
<b>Chipset</b>	Kirin 980 (7nm)
<b>Operating System</b>	Android 10, EMUI 12.0.0
<b>Screen Resolution</b>	1440 x 3120 pixels (6.39 inches display)
<b>Memory</b>	6GB RAM
<b>Storage</b>	128GB

In the process of developing the project, the main machine used for coding and constructing the project is the Lenovo Legion Y545 laptop. On the other hand, the Huawei Mate 20 Pro mobile device is mainly used to test the project mobile application during the debugging process and conduct software testing, together with the use of Android emulator from the Android Studio for real case interactivity between devices when testing the project mobile application.

## 4.3.2 Software Requirements

Table 4.3.2.1 Software Requirements for The Project

Components	Description
<b>Tools</b>	<p><b><u>Visual Studio Code</u></b></p> <p>Visual Studio Code is a free coding editor that supports multiple programming languages such as Python, Java, and Dart through the adding of extensions to the editor. It also allows the user to conduct the checking and debugging of code.</p>
	<p><b><u>Android Studio</u></b></p> <p>Android Studio is an Integrated Development Environment (IDE) to develop android applications. It consists of a strong Android emulator function for programmers to test the application developed using emulator device while doing debugging of code.</p>
	<p><b><u>Firebase</u></b></p> <p>Firebase is a mobile application development platform developed by Google to allow programmers to improve their application software using the tools and services provided, such as maintaining backend components in the cloud database. In this project, Firebase Authentication, Firebase Storage, and Firebase Firestore are the main backend services used to support the project mobile application.</p>
<b>Language, Libraries and Framework</b>	<p><b><u>Dart</u></b></p> <p>Dart programming language is a programming language that is usually used together with Flutter framework to develop applications for any platform. This language includes null safety and core libraries that can support programmers in developing applications. The Dart language is the main language used in developing the functionalities modules for the project mobile application.</p>
	<p><b><u>Flutter</u></b></p> <p>Flutter is an open-source mobile UI framework developed by Google in 2017 to help programmers create mobile applications with one codebase, which means to use the same language and codebase to develop applications for multiple platforms. The Software Development Kit (SDK) and framework UI elements</p>

	<p>from Flutter allows the ease of programmers to use them as tools to increase their working effectiveness. For this project, Flutter is used to build the interactive user interface for the project mobile application.</p>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 4.4 Project Timeline

### 4.4.1 Overview

Based on the project methodology, the project timeline is planned carefully for effective time management and ensure that the project can be done on time before deadline. Before planning the project timeline, the requirements planning for the project is done, which includes determining the project scope, problem statement and project objectives based on the project's goals. Literature review on similar works is also done under requirement planning to obtain the basic requirements for the project mobile application and comparing the strengths and weaknesses of the similar works. After that, the project timeline is planned based on the finalized requirement for the project to ensure a proper and effective schedule is arranged.

Next for the user design phase, the project development tools such as VS Code IDE and the Android Studio are determined, followed by setting up Firebase as the project's main database. The project development is started by first implementing the basic functionalities for the project mobile application, such as the login and sign-up module, and the profile module. These modules are important to be done first as it is the starting point of the project functionalities, as the user needs to have an account to log into the project mobile application to use the available features.

From here, the project development has moved on to the construction phase, which mainly works on handing backend data server and important working model functions, which are the online communication module, helper finder module, QR identification module, and offline module. This is due to these are the important functional modules that interacts with each other to achieve the main goals of the project mobile application. From here, the Final Year Project 1 report is also finalized, and the Final Year Project 1 is presented.

## CHAPTER 4 METHODOLOGY AND TOOLS USED

Starting in Final Year Project 2, the improvement of the project is done based on the feedback received from Final Year Project 1, which then the online calling function is implemented to the project mobile application. Then, the administrator module is also developed to add in administrator controls, and the implementing of application data analysis into the administrator module. The data is then added manually to be analysed using the data model developed for the data analysis.

At this point, the project has started to enter the final phase, which is the cutover phase to start wrapping up the work done. Testing of the project mobile application as a whole system is done to ensure the project mobile application is working as expected without error. Finally, the Final Year Project 2 report is prepared, and the Final Year Project 2 will be presented to the supervisor and moderator in Week 14.

4.4.2 Gantt Chart

Table 4.4.2.1 Gantt Chart of The VCare Circle Mobile Application Project Timeline

Task Name	Duration (Week)	Week																					
		Project 1							8	Project 2													
		1	2	3	4	5	6	7		1	2	3	4	5	6	7	8	9	10	11	12	13	14
<b>Requirement Planning Phase</b>																							
Planning Project Scope, Problem Statement, and Project Objectives	1																						
Conduct Literature Review on Similar Work	2																						
Plan Project Timeline	2																						
Decide Project Tools and Project Database	1																						
<b>User Design, Construction Phase</b>																							
Login And Sign-Up Module	1																						
Profile Module	1																						
Online Communication Module	2																						
Helper Finder Module and QR Identification Module	1																						
Offline Module	1																						
Software Testing	1																						
Project Documentation	2																						
Final Year Project 1 Report Writing	2																						



CHAPTER 4 METHODOLOGY AND TOOLS USED

Final Year Project 1 Report Submission	1																					
Final Year Project 1 Presentation	1																					
Improvement of Project Based on Feedback	2																					
Online Call Function Module	2																					
Administrator Module	4																					
Application Data Analysis Function Module	5																					
<b>Finalize Product and Cut Over Phase</b>																						
Testing	2																					
Final Year Project 2 Report Preparation	2																					
Final Year Project 2 Presentation	1																					

### **4.5 Summary**

In summary, this chapter has included the explanation of the Rapid Application Development system methodology used for this project, which gives the main benefits of able to adapt changes in each phase, identify design flaws at early stages to be fixed as soon as possible, faster overall project development time, and to give out the final product that focus in satisfying the needs and requirements of the users. The hardware and software development tools used for this project is also discussed, together with the planning of project timeline for the purpose of ensuring the project can be finished on time, which has been displayed in the Gantt chart for better visualization.

## Chapter 5

# System Implementation

### 5.1 Overview

In this project, the functional modules implemented for this project for the purpose of achieving the project objectives and goals are the login and sign-up module, profile module, online communication module, helper finder module, QR identification module, offline module, and the administrator module. The project named VCare Circle, is a mobile application developed mainly to be used in Android devices, which consist of three themes, which are the dark blue theme for user, pink and purple theme for helper, and green theme for administrator. The project mobile application is fixed to be in portrait orientation, and to be mobile responsive to be able to be used in all screen sizes. Navigation drawer is added in this project mobile application for easy navigation, which here there are two types of navigation drawer, which are the navigation drawer for administrator as shown in Figure 5.1.1, and also the navigation drawer for users and helpers shown in Figure 5.1.2. Multiple pages are included in the project mobile application, which some of them being able to be accessed by all user account types, while are only accessible to specific user account type, such as for helper account have access to view the helper details page, and for administrator account type have access to the application data analysis pages. The project mobile application starts from the welcome page which is show in Figure 5.1.3.

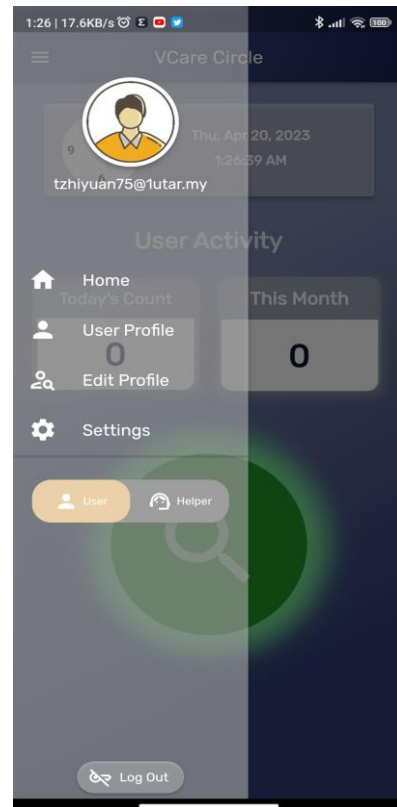
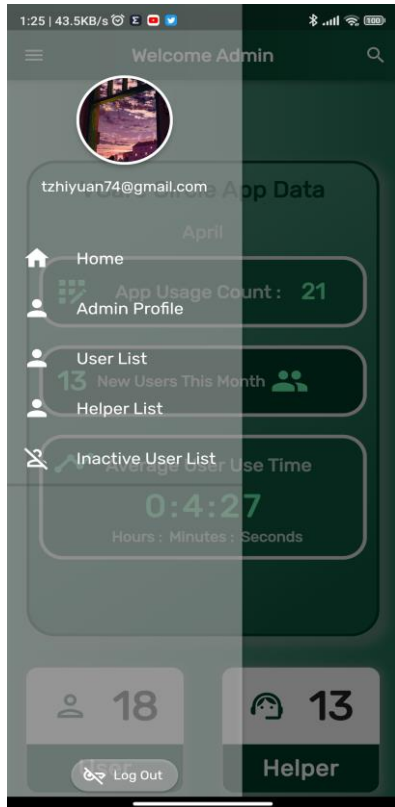


Figure 5.1.1 Admin Navigation Drawer (Left) and Figure 5.1.2 User and Helper Navigation Drawer (Right)

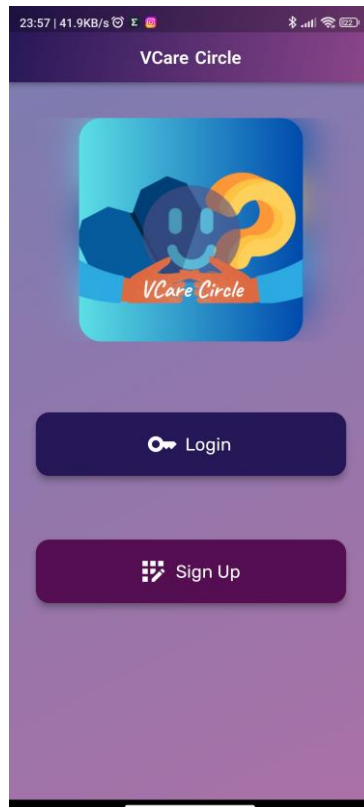


Figure 5.1.3 Welcome Page

## 5.2 Login Page

Figure 5.2.1 shows the login page of the project mobile application where a user is directed to this page if one selects the login button on the welcome page or select the login link on the first sign up page. The login page consists of the account and password field for user to enter their registered email account and password to login into the mobile application. There is also the option of forgot password link if the user wants to reset the password, and the sign up account link if user does not have a registered account and wants to sign up one instead.

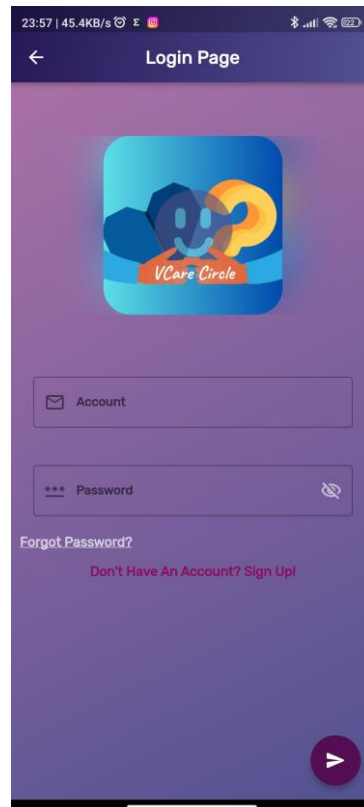


Figure 5.2.1 Login Page

## 5.3 Sign Up Page

There are two pages for the sign-up module, which the first page provides a form that allows a new user to input his or her personal information and the second page provides a form to input the email account and preferred password. The purpose is to allow new users to register an account into the system to proceed logging into the project mobile application. Figure 5.3.1 and Figure 5.3.2 shows the two pages for the sign-up module.

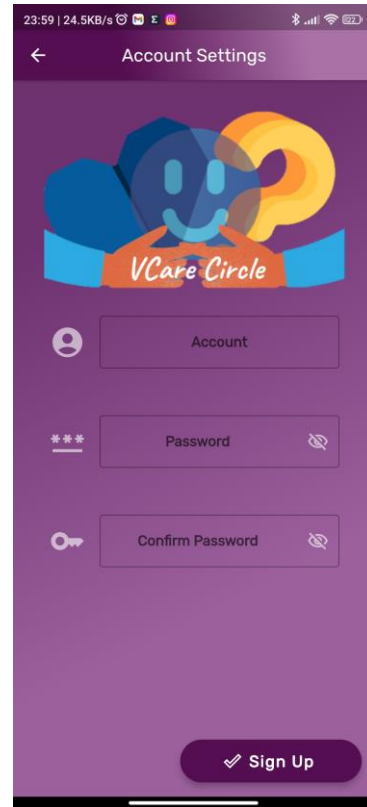
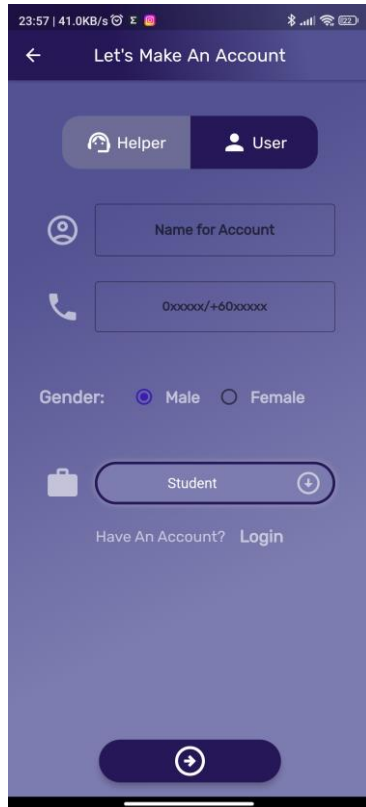


Figure 5.3.1 Sign Up Page One (Left) and Figure 5.3.2 Sign Up Page Two (Right)

#### 5.4 Reset Password Page

Figure 5.4.1 shows the forgot password page, which the user can input his or her email account used to register the account in the project mobile application, to then press on the “Get Reset Password Link” button to get the reset password link in the email inbox.

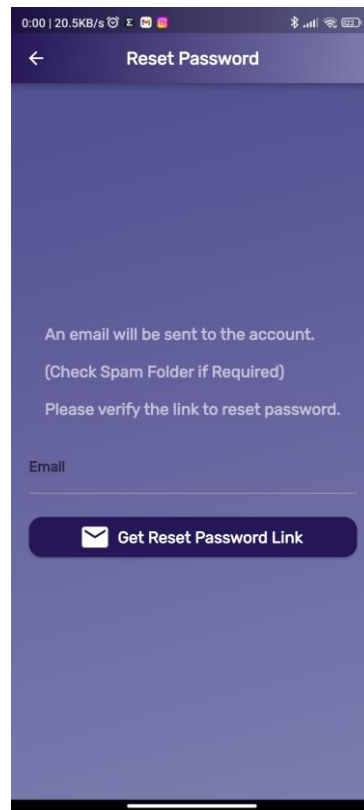


Figure 5.4.1 Reset Password Page

## 5.5 Verify Email Page

The verify email page is where the user with account type “Helper” or “User” can get the email verification link here to proceed verifying his or her email account to continue using the project mobile application. The user is directed to this page before entering the home page after login if user has not verified his or her email account yet or is directed after complete signing up an account. The user can also manually request for email verification link through pressing the “Resend Email Verification” button or press the cancel button to be directed back to the welcome page. The verify email page is shown in Figure 5.5.1.

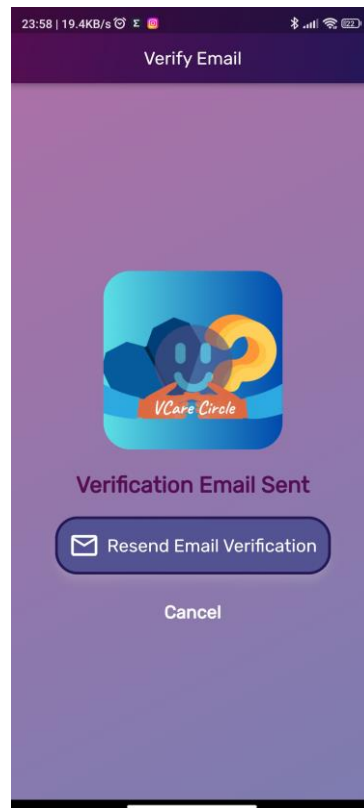


Figure 5.5.1 Verify Email Page

## 5.6 Verify Admin Password Page

The verify admin password page is for the administrator to login into the project mobile application to gain access to the administrator controls of the project mobile application and to view on the application data analysis. Here an input field is provided for the administrator to key in the admin password to be directed to the admin home page after verification. In case the administrator has forgotten the admin password, he or she can choose the forget password link below the input field to reset the admin password under the condition of he or she has login the project mobile application with the admin email account and using the main administrator mobile device. The administrator can also choose to select the icon button at the top right corner of the page if he or she chooses to log out instead The Figure 5.6.1 shows the design of the verify admin password page.



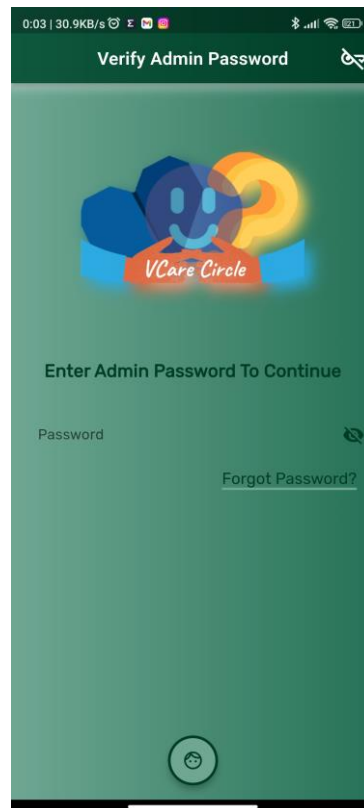


Figure 5.6.1 Verify Admin Password Page

## 5.7 Inactive Status Warning Page

Figure 5.7.1 shows the inactive status warning page, which is showed to users and helpers if his or her account status has been set to inactive by the administrator of the project mobile application. This page shows the error message of the account being inactive and unable to proceed login into the project mobile application and can choose to either select the icon button at the top right corner of the page to be directed to the mobile device email application to email help to the administrator or select the log out button to log out of the project mobile application instead.

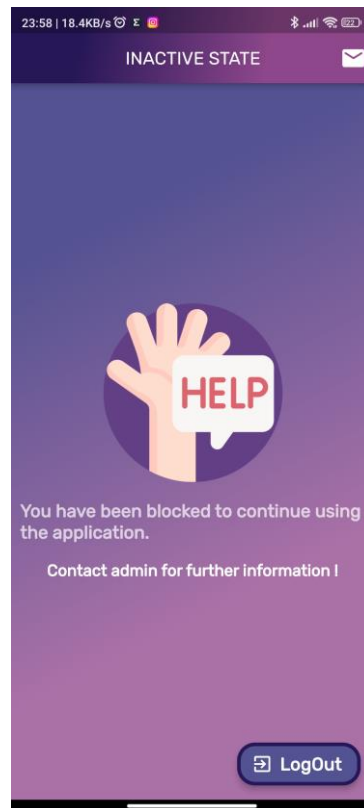


Figure 5.7.1 Inactive Status Warning Page

## 5.8 Home Page

The home page of the project mobile application is designed based on the account types, which are the user home page, the helper home page, and also the administrator home page. The Figure 5.8.1 shows the administrator home page, Figure 5.8.2 shows the user home page, and Figure 5.8.3 shows the helper home page. The reason for this is to allocate different functionalities based on the user account type, such as the home page for “User” type user has a daily and monthly activity count and an icon button to request for help, while the home page for “Helper” type user has a helper duty daily and monthly count and able to view and modify helper status and helper duty period. The administrator home page on the other hand shows the summary of the application data analysis, and also the counters for the total number of users and helpers registered in the project mobile application server. The admin home page also has a search icon at the top of the page, which allows admin to search for a specific email account.

When the user is in an active session, the home page also changes to another display which includes functionalities that support the user when in the session. For users, the home page shows the QR code scanner for scanning QR code, the session timer to view the duration of the

ongoing session, a “Message” button to enter the chat page, and also the slider for ending the active session when done. On the other hand, for helpers, the home pages also change to another display which has a QR code generator display, a session timer showing the duration of the ongoing session, and a “Message” button to enter the chat page. The changes in the home page can be seen in Figure 5.8.4 for user home page and Figure 5.8.5 for helper home page.

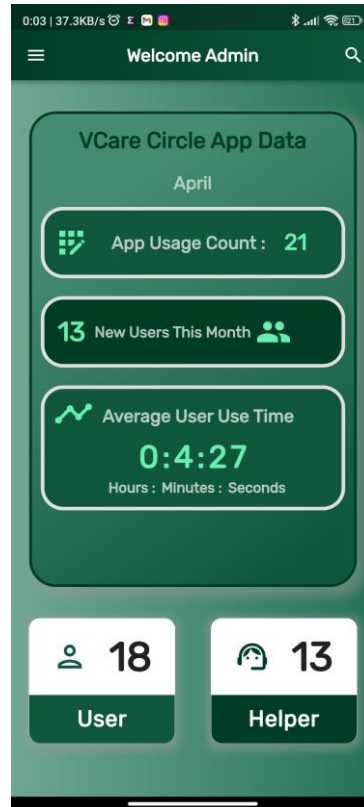


Figure 5.8.1 Admin Home Page

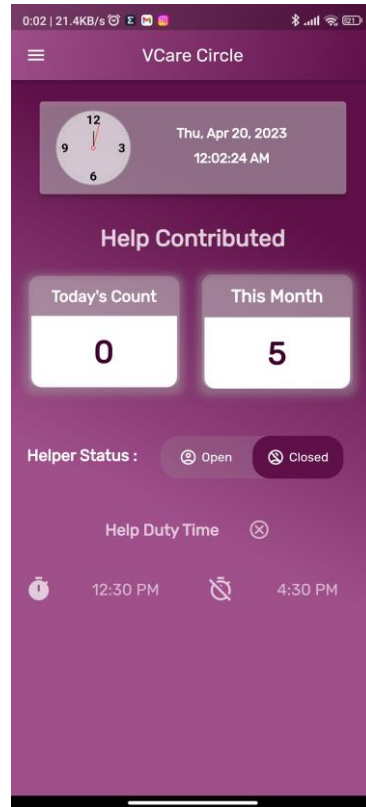
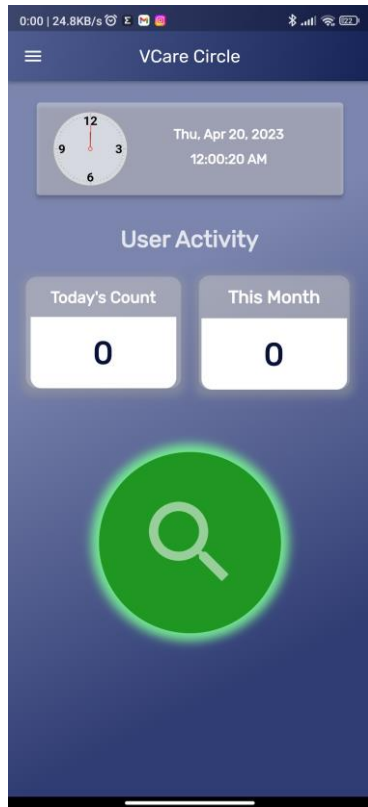


Figure 5.8.2 User Home Page (Left) and Figure 5.8.3 Helper Home Page (Right)

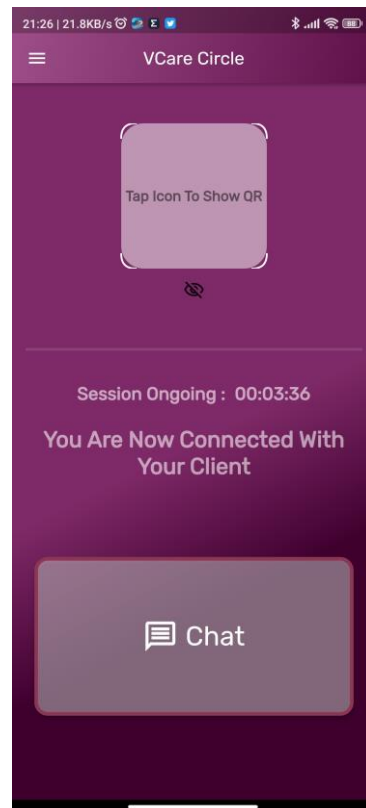
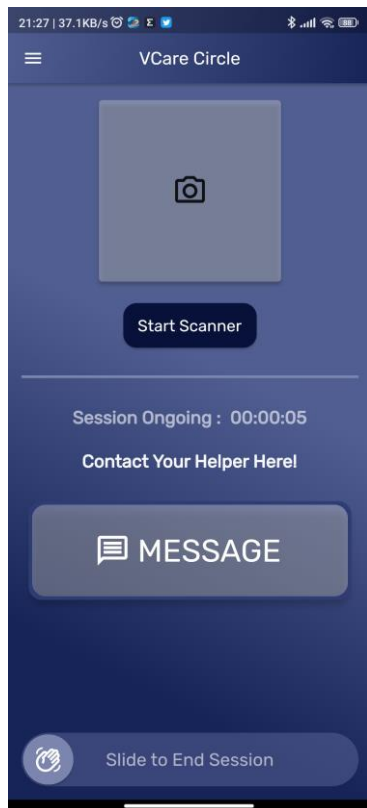


Figure 5.8.4 User Home Page in Session (Left) and Figure 5.8.5 Helper Home Page in Session (Right)

## 5.9 Profile Page

The project mobile application has profile pages prepared for users based on user account type, which shows the user account information saved in the Firebase server. Figure 5.9.1 shows the user profile page and Figure 5.9.2 shows the helper profile page. For users and helpers, there are two main subpages, which is the account details page showed in Figure 5.9.3 and Figure 5.9.4 showing the user personal information, and the helper details page in Figure 5.9.5 showing the helper's status and helper duty period, which is only accessible if the account type is "Helper". There is also edit profile pages provided as shown in Figure 5.9.6 and Figure 5.9.7 to allow modifying of user personal information. The profile page body for user and helper profile page also have the "Settings" tile which directs user to the settings page and the "Log Out" tile if he or she wishes to log out of the project mobile application.

There is also an administrator profile page prepared which shows the personal information of administrator, which is shown in Figure 5.9.8. The administrator edit profile page is also provided to not only edit the personal information, but also to edit the email account, and to reset the admin password at the reset password link at the bottom of the form in the page as shown in Figure 5.9.9.

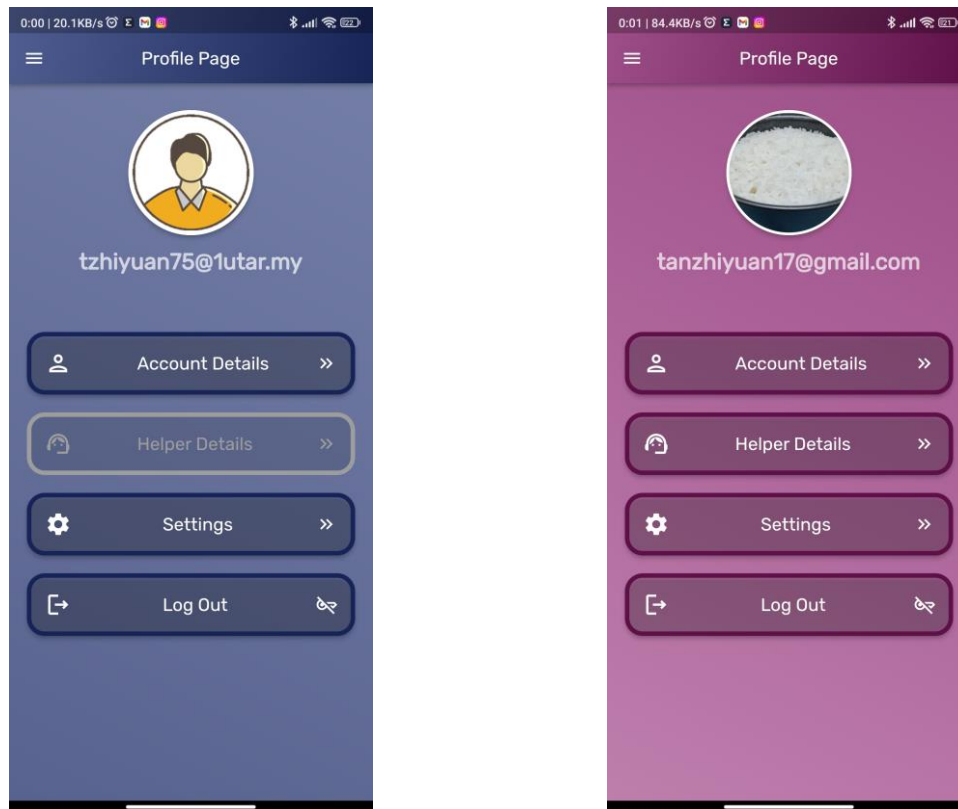


Figure 5.9.1 User Profile Page (Left) and Figure 5.9.2 Helper Profile Page (Right)

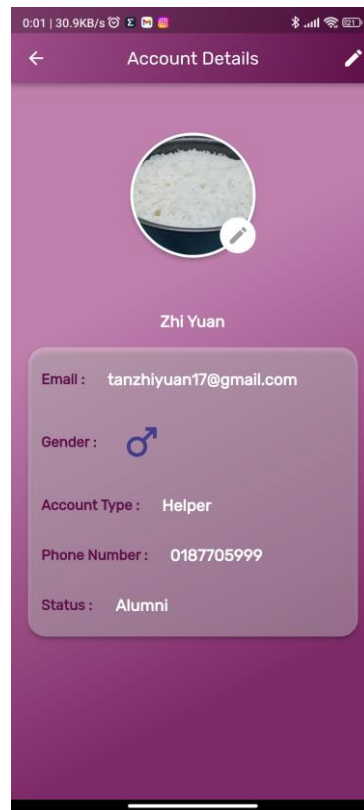
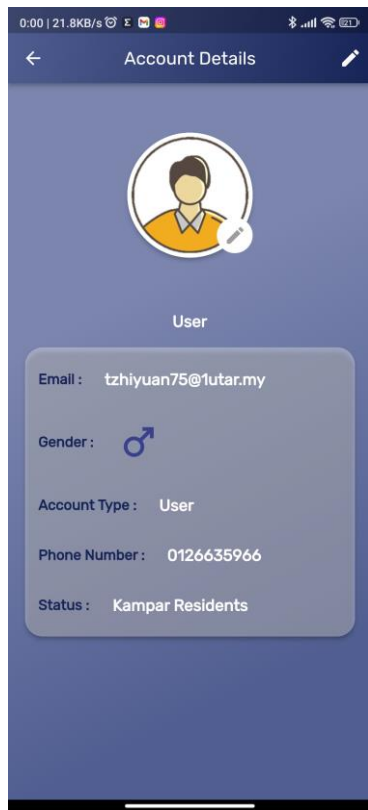


Figure 5.9.3 User Account Details Page (Left) and Figure 5.9.4 Helper Account Details Page (Right)



Figure 5.9.5 Helper Details Page

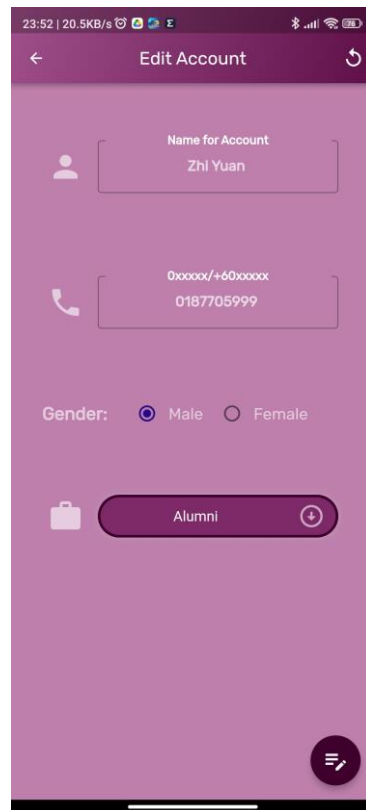


Figure 5.9.6 User Edit Profile Page (Left) and Figure 5.9.7 Helper Edit Profile Page (Right)

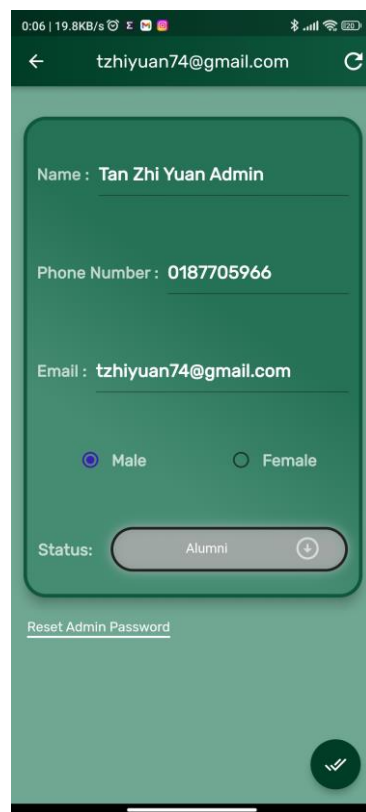
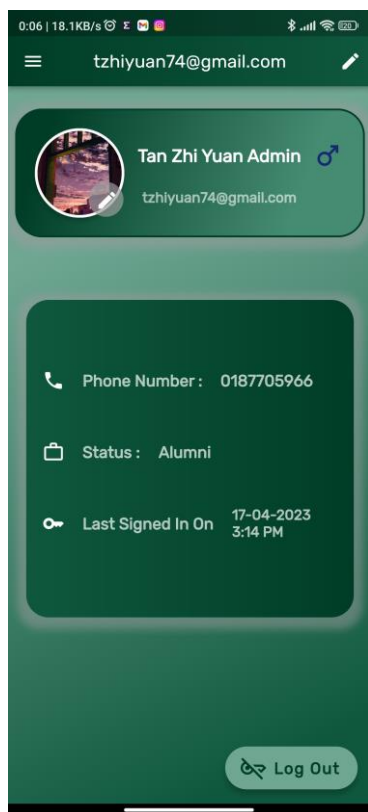


Figure 5.9.8 Admin Profile Page (Left) and Figure 5.9.9 Admin Edit Profile Page (Right)

### 5.10 Settings Page

In the project mobile application, there is a settings page which is only accessible by the user and helper as shown in Figure 5.10.1. In the page, there is an email banner of the user which directs to the user's account details page. The user is also able to conduct the changing of account password through the "Change Password" option, and to permanently delete the user account through the "Delete Account" option. Furthermore, icon buttons to contact the project mobile application help centre and UTAR are also provided under the "Other Settings" section, which the user can either contact using email or mobile phone call through the icon button options given. Lastly, there is also a log out button provided for user to sign out his or her account.

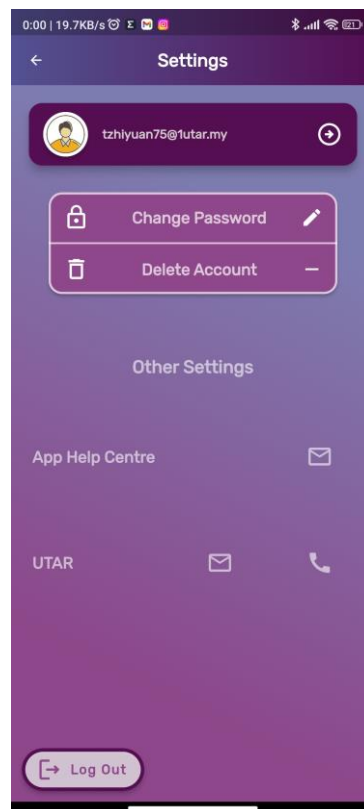


Figure 5.10.1 Settings Page

### 5.11 Offline Page

The offline page shown in Figure 5.11.1 is added to the project mobile application as an indicator of currently the application user is in an offline situation. When the project mobile application is running or starting, if the internet connection is lost, the user is directed to the offline page, where the user can choose to press the "Message" button to contact UTAR via



mobile SMS communication, which the user is then directed to the mobile default SMS communication platform. On the other hand, if the user presses the “Call” button to contact UTAR via mobile direct phone call, the user is then directed to the mobile phone call dialler screen.

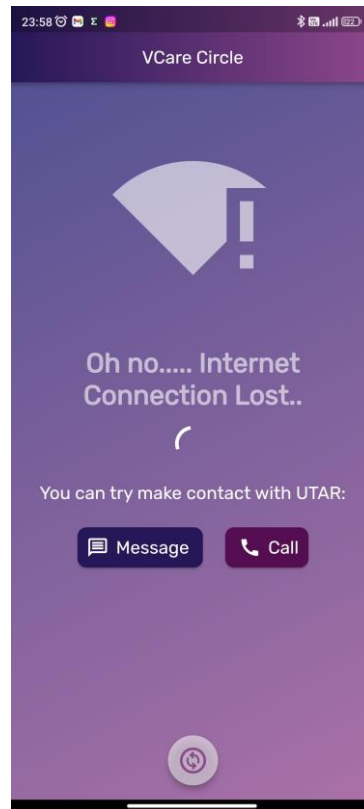


Figure 5.11.1 Offline Page

### 5.12 Online Chat Page

During an active session, the user can use online chat communication to interact with his or her helper or vice versa. The chat page has different colour themes based on the user’s account type. For users, the chat page has a dark blue theme, while for helpers the chat page has a pink and purple theme. The chat page designs can be seen in the Figure 5.12.1 and Figure 5.12.2. The chat pages for the two types of users have the same functionalities, which is to display chat contents with sender chat tiles on the right and the receiver chat tiles on the left. The chat page has an add icon button on the bottom left to allow sending images from gallery or camera, while the bottom center input text field allows user to type in text messages and send the text messages through pressing the send icon button on the bottom right.

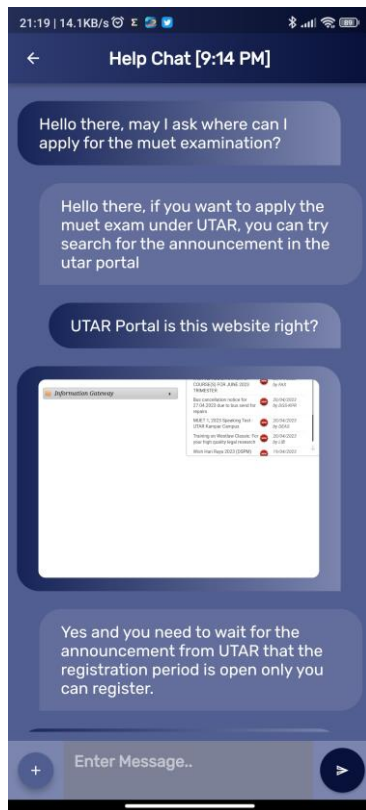


Figure 5.12.1 User Chat Page (Left) and Figure 5.12.2 Helper Chat Page (Right)

### 5.13 Online Voice Call Page

During an active session, the user can choose to have online voice call communication with the assigned helper and vice versa for real-time interaction. When one of the user or helper side initiates a call, the other side receiver is notified as the incoming call screen overlay is displayed to allow user or helper to choose to either accept or reject the call, which is shown in Figure 5.13.1 and Figure 5.13.2. When the user or helper is in the online voice call channel, he or she is directed to the respective call screen as shown in Figure 5.13.3 and Figure 5.13.4, where the top of the screen shows the status on the receiver side, and the bottom of the screen provides the microphone button to choose to open or mute the mobile device microphone, The end call button is also provided for the call screen to end the online voice call when the conversation is done.

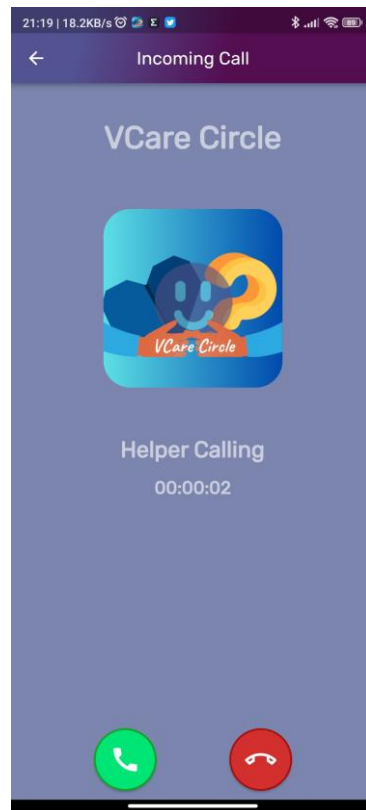
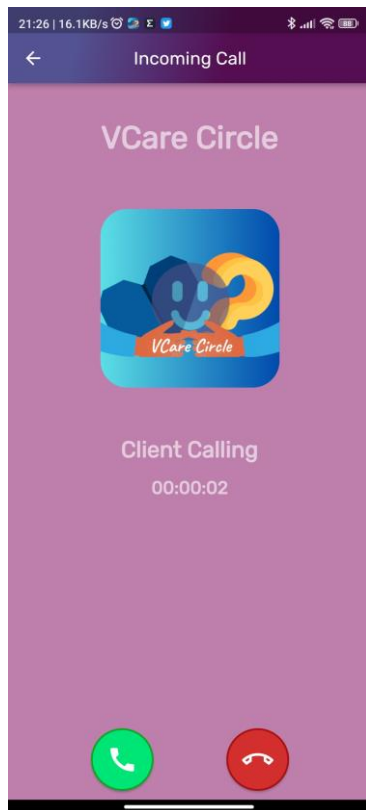


Figure 5.13.1 User Incoming Call Screen (Left) and Figure 5.13.2 Helper Incoming Call Screen (Right)

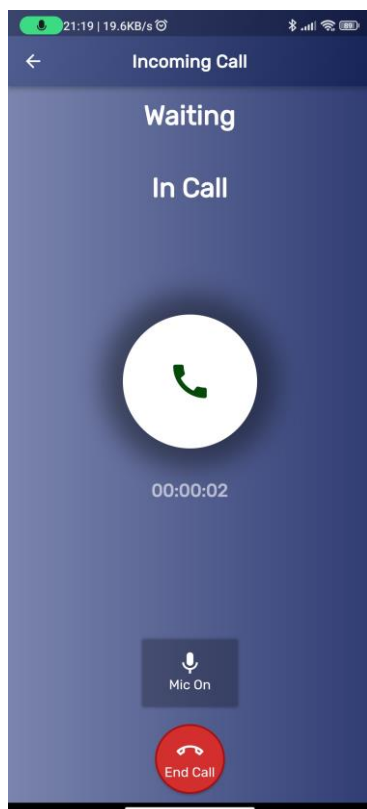


Figure 5.13.3 User Call Screen (Left) and Figure 5.13.4 Helper Call Screen (Right)

### 5.14 Users and Sessions Management Pages

For administrator of the project mobile application, multiple users and sessions management pages are added to provide easy and clean UI design interface in handling user and session data under the administrator controls. First, the list of users, helpers, and inactive users is created to show the summarized details of list of users, helpers, and inactive users registered and saved in the Firebase server. The Figure 5.14.1 shows the list of users page, Figure 5.14.2 shows the list of helpers page, and Figure 5.14.3 shows the list of inactive users page. Next, the administrator can also view on the account details of the users, helpers, and inactive users saved in the Firebase server, while also able to see the application usage of the user, helper, and inactive user at the counter widgets and for inactive user the inactive start date. The respective account details pages are shown in Figure 5.14.4, Figure 5.14.5, and Figure 5.14.6. However, only the users and helpers account details can be edited by the administrator by selecting on the fields provided. In the case where if the administrator wants to set the user or helper account status to inactive, he or she can select the icon button at the top right corner to set the current account status to inactive. For inactive user account details page, selecting the icon button at the top right corner will set the current account status back to active instead. Furthermore, by selecting on the application usage counter widget, the administrator can see the list of sessions involving the current account administrator is viewing, which either having the current account as the user or the helper in the session, which the pages are shown in Figure 5.14.7 and Figure 5.14.8. Chat history in the history session can also be seen by the administrator at the chat history page shown in Figure 5.14.9, while the administrator can also choose to delete the history session if required by selecting the icon button at the top right corner of the page.

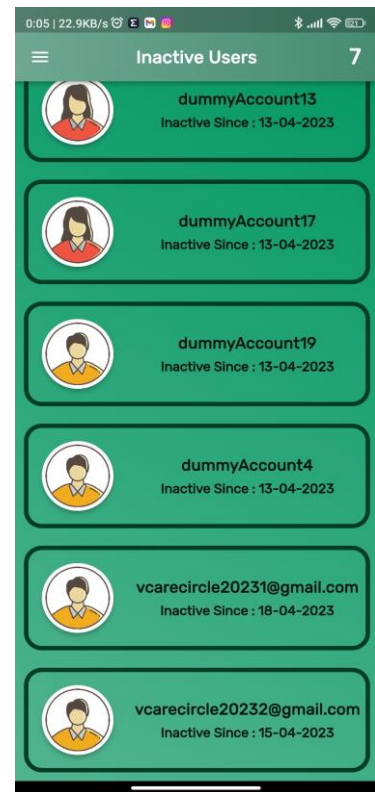
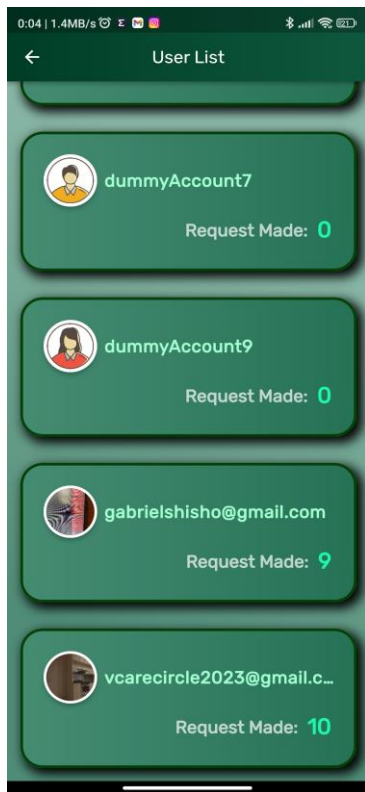


Figure 5.14.1 List of Users Page (Left), Figure 5.14.2 List of Helpers Page (Middle), and Figure 5.14.3 List of Inactive Users Page (Right)

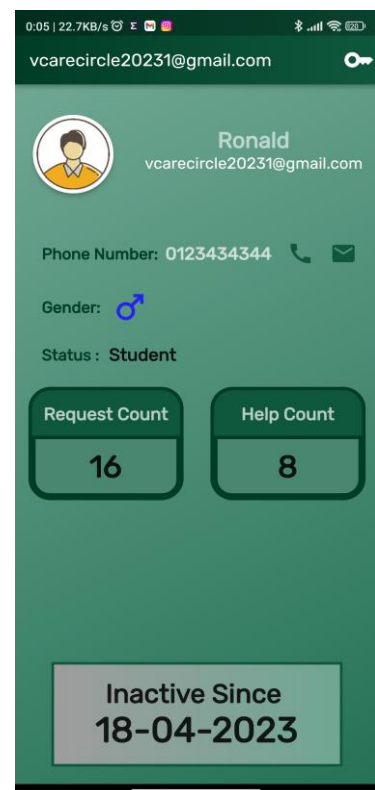
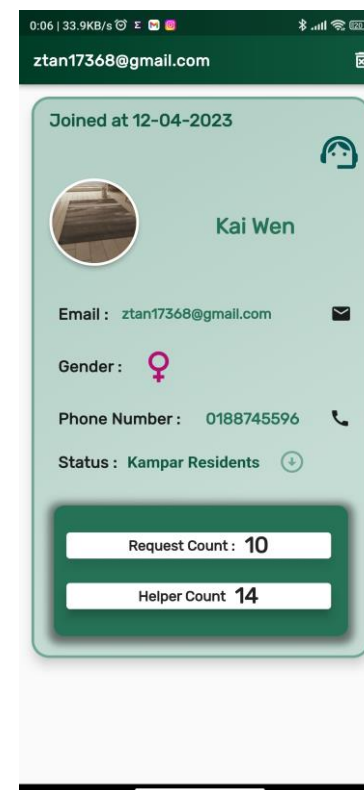
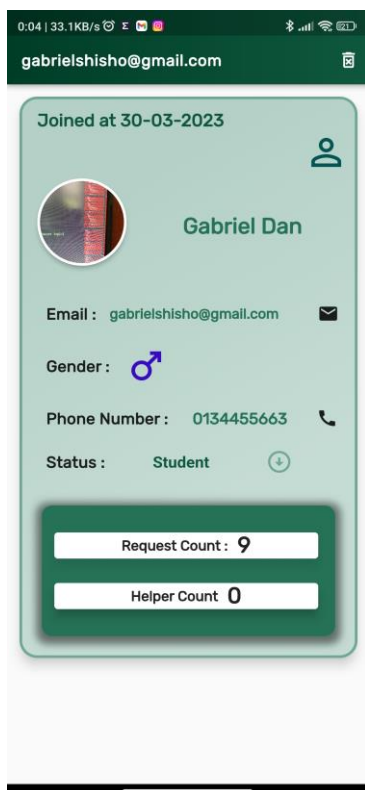


Figure 5.14.4 User Account Details Page (Left), Figure 5.14.5 Helper Account Details Page (Middle), and Figure 5.14.6 Inactive User Account Details Page (Right)

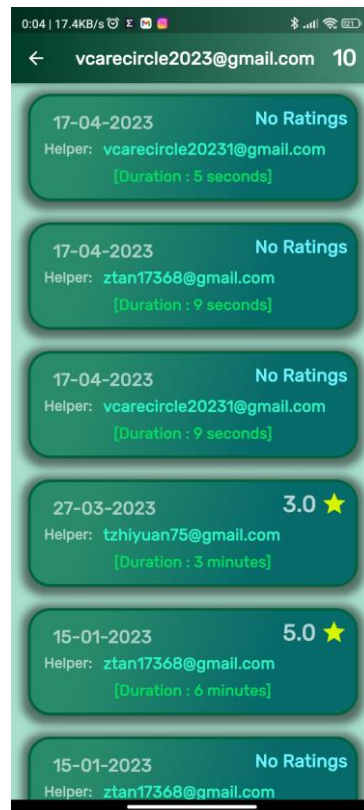
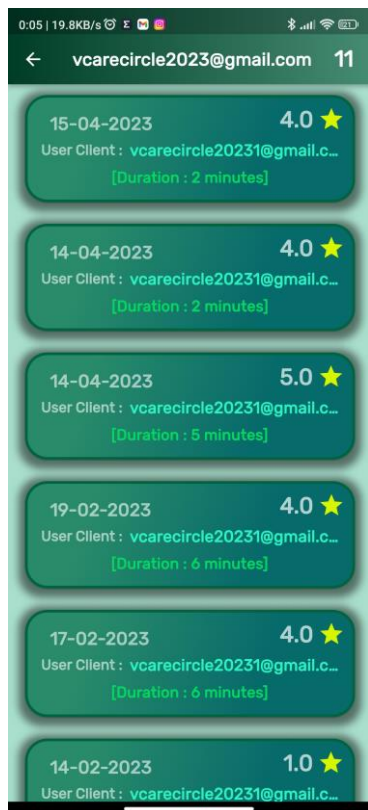


Figure 5.14.7 List of User Request History Sessions (Left) and Figure 5.14.8 List of Helper Duty History Sessions (Right)

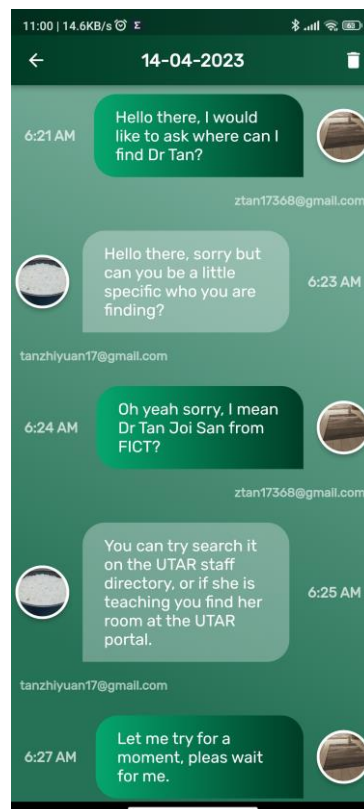


Figure 5.14.9 Session Chat History Page

### 5.15 Application Data Analysis Pages

In the application data analysis module, there are three graphs provided for the administrator to view on the tabulation of data such as on the number of new registered users, application usage by users and helpers, and the overall session duration done between users and helper based on different time intervals, which each of the graph designs are displayed in Figure 5.15.1, Figure 5.15.2, and Figure 5.15.3 respectfully. From here, the administrator is able to change the time interval of the graph and also the view size of the graph through selecting the three-dots icon at the top right corner of the pages of the graphs. Furthermore, from the overall session duration graph, the administrator can also choose to view on the list of result session based on the longest and shortest duration found by long pressing on the value points of the line graph, which then the administrator can also view on the chat history in the result sessions. Figure 5.15.4 and Figure 5.15.5 shows the examples of list of history session page used to show the summarized details of the history sessions.

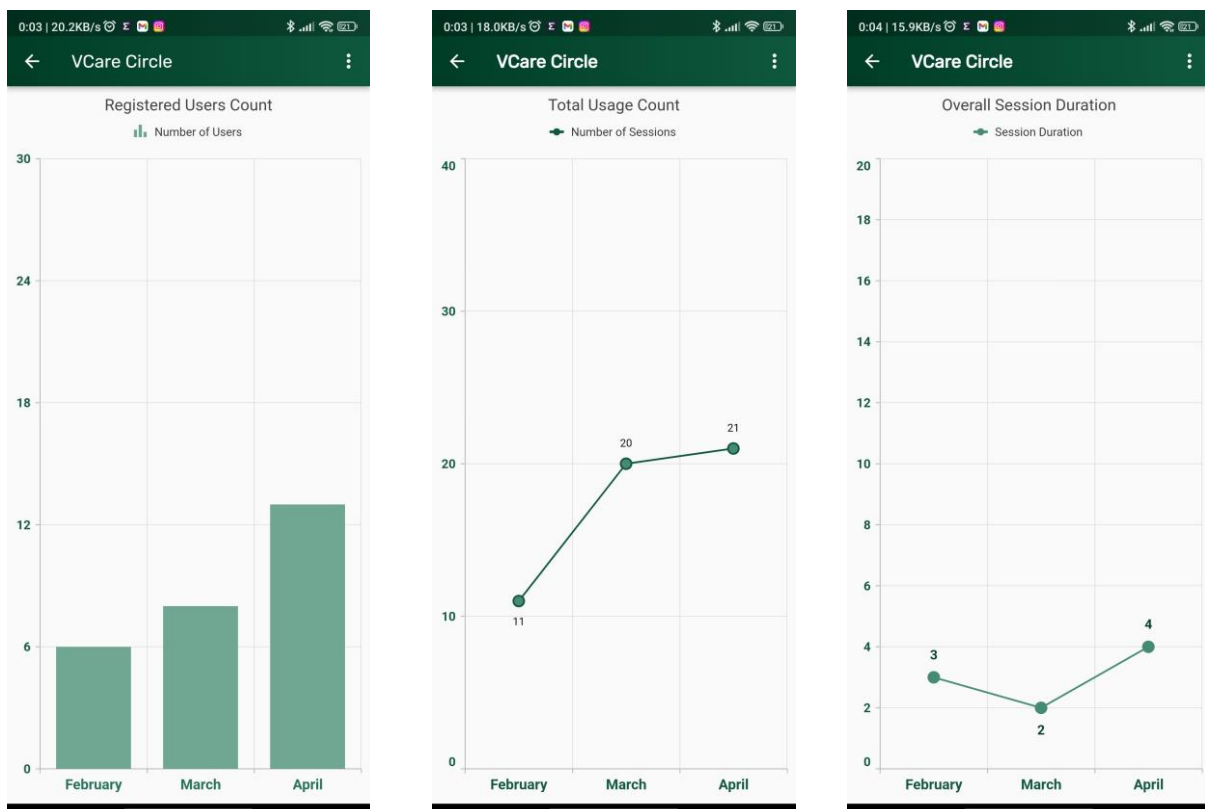


Figure 5.15.1 Bar Chart for Number of New Registered Users (Left), Figure 5.15.2 Line Chart for Application Usage (Middle), and Figure 5.15.3 Line Chart for Overall Session Duration (Right)

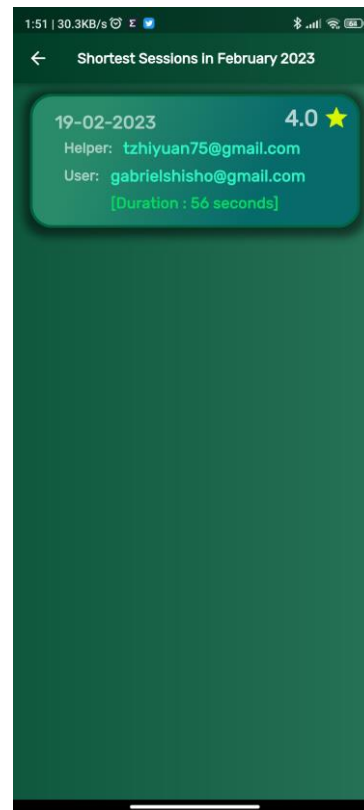
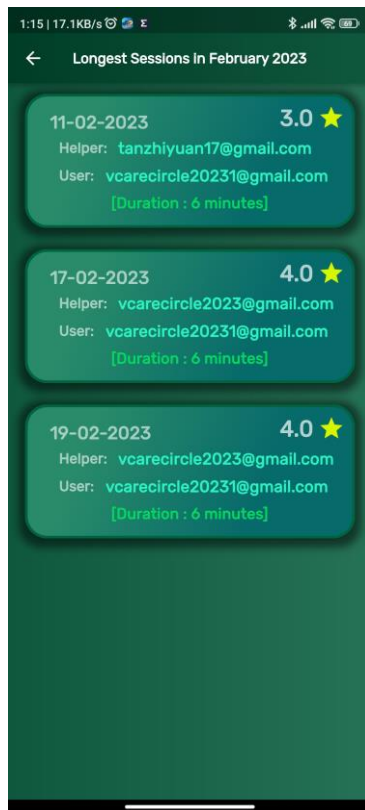


Figure 5.15.4 List of Result Sessions for Longest Duration (Left) and Figure 5.15.5 List of Result Sessions for Shortest Duration (Right)

## 5.16 Summary

In summary, the project mobile application consists of seven functional modules, which are the login and sign-up module, profile module, online communication module, helper finder module, QR identification module, offline module, and the administrator module. This chapter has showed and discussed on the design and overall functionalities of the pages of the project mobile application implemented, for the purpose of having a simple overview on the overall work done for the project mobile application.



## Chapter 6

### Conclusion

#### 6.1 Project Review, Discussion and Conclusion

Asking for help is an effective way of solving issues on hand, and also to be able to quickly adapt to new environment easily. It is important for one to be able to speak out their issues in order to not keep the issues to themselves that may be too much for them to handle or leave the issues unsolved that may bring more hassle to them in the future. However, not many people are able to do so due to certain barriers, such as unable to specify the issue specifically, assume this is an act of shame, and also the feeling of nervous and anxiety when asking for help. This problem is faced by many people around due to low self-esteem or being in a new environment without any support around, especially university students nowadays [2]. Therefore, based on the barriers mentioned, the project is developed to try resolve the barriers mentioned and provide an effective way for university students to ask for help.

The project aims to provide an effective way for university students to ask for help anywhere and anytime without having to physically look around for someone or to book a session with people, while keeping their privacy concealed for the users and helpers to prevent unwanted disturbance and awkwardness. To achieve the project's main objectives, multiple functional modules are planned carefully to achieve the goal of the project. Rapid Application Development (RAD) methodology is also chosen as the project's methodology for development. For the project first objective of providing an effective way of asking for help, the login and sign-up module, profile module, the online communication module, helper finder module, QR identification module, and lastly the offline module are done to achieve the objective. With these modules developed, the user can now easily find an available helper through the project mobile application to interact and seek help from helper without having to book a session or know someone. Besides, the project's second objective of privacy conceal is also achieved as during the process of asking for help using the project mobile application, the identity of the helper and user are hidden throughout the session. The administrator module is also developed to provide administrator controls for the project mobile application, and also to include the application data analysis to study the users and helpers' behaviours through the usage of project mobile application for research purpose.

The problem faced when developing this project is first of all the implementing of online communication module, which is due to the issue of not getting real-time updates properly on the online chat conversation. This is due to the major latency in obtaining the chat conversation from Firebase to be displayed in the chat page through fetching of data at time intervals. After spending time on online research, a workable solution is implemented by utilizing the use of stream builder. Next, the issue of allowing receiver to join in the online voice call channel is also one of the issues faced during the development of project. This is due to the receiver unable to get update on the call has been made by the caller, and thus unable to properly enter the online call channel. This has been resolved after spending some time configuring the receiver settings based on the online documentation of the Agora SDK.

### **6.2 Novelties and Contributions**

By developing this project to improve the situation of university students in asking for help, this will help in saving the time needed by them in trying to search for someone nearby or have to book a session to seek for help, which can be time consuming if one does not know anyone, or even in the situation where they required help and there are nobody nearby to ask for help. This project also helps new intake students in adapting to the new environment more quickly, as this project allows them to seek the help they need, even if they are not familiar with anyone yet. By being able asking for help easily, one can promote his or her self-growth, to not only able to handle the problems they are facing now, but to also build up their self-esteem so that in near future they can be more confident in facing problem, while also be capable of being the one who helps others instead.

The application data analysis in this project can help the admin in understanding the users' behaviour through the analysis done on the sessions conducted between the users and helpers. Here, not only the admin can understand what are the average duration of the sessions needed for users to be able to get help, but also to see the content of the sessions, to try understand what is the most questions being asked by the users and what are usually the answers being given by the helpers towards this question. By doing this, the admin can get a summary on what is mainly troubling the university students, to try to understand the situation and suggest the UTAR university to take action in trying to resolve the issues beforehand, to ensure the university students will not face similar problems in the future.

The online communication module included in this project mobile application allows online interaction between the helper and user, so that no external application or online platform is needed for the user or helper to communicate with each other. Two options of either online chat conversation or online voice call are available for the user and helper to interact during a session, while here not only the project mobile application does not require the saving of contacts to communicate to each other, but to ensure neither party's identity is exposed to prevent future disturbance for the helpers, while the users being able to fully express themselves without having to worry on the position or power of who he or she is asking help from.

### **6.3 Future Work**

For this project mobile application, there are some improvements that can be made in order to further improve the mobile application to achieve a higher goal. First, the chat data analysis can be added in order to have a trained model to find out what are the topics that are most asked or discussed by the users and helpers. This can allow the administrator to understand what the most questions are asked from the users, and what are the answers mostly given for those questions asked by the helpers. The purpose of this is to understand what are troubling the users most and try to figure out what can be done to improve this situation instead, so that less similar situation would be faced by them in the situation. By obtaining this data, the set of optimal questions and answers can be finalized, so that they can be posted out the questions for the users in a type of forum or Q&A section, so that before trying to ask or request for help from a helper through the project mobile application they can choose to search for the answers that may answer their situation or problem facing, which this can be the second future work that can be done for the project mobile application.

A forum or Q&A section page can be added for the users to display out what are the questions normally asked by the users, and also what the optimal answers are being given to the questions ask. This not only can reduce the demand of asking for help from a group of limited helpers, but also can increase the efficiency of the users in solving their situation facing now, as the accuracy of the answers from the Q&A section are rated by the previous users who have faced similar situation, so the top optimal answers are usually the ones who have solved the situation or problem previously faced, which can act as the best reference for them in case there are future users who have faced the same situation or problem again. The university can also use

## CHAPTER 6 CONCLUSION

these data as reference in order to improve their student affair management system and also in organizing open days, what advise should be given to the new intake students so that they won't be in a state of confusion when starting their student life in UTAR and have a harder kickstart in getting used to the new environment.

The last future work can be done for the project mobile application is to implement a GPS tracking system. This is mainly to allow a better efficiency in allowing the project mobile application to assign available helpers that are currently near to the requested users, so that in case the user requires a guide around the place, the helper assigned can find the user more easily. Besides, the GPS tracking system can also be used by the helper to pinpoint the user location to search for him or her in case they have agreed to have a physical meeting, as the identity of the helper and user are hidden, it can be hard to find each other on just based on personal characteristics alone.

## References

- [1] I. E. Team, "What are Social Skills? Definition and Examples," 9 December 2021. [Online]. Available: <https://www.indeed.com/career-advice/career-development/social-skills>. [Accessed 24 November 2022].
- [2] D. Butler, "Why is asking for help a good idea?," proactive, 30 June 2021. [Online]. Available: <https://www.proactivehm.com.au/why-is-asking-for-help-a-good-idea/#:~:text=Asking%20for%20help%20builds%20connections,skills%2C%20and%20appreciate%20their%20advice.&text=Asking%20for%20help%20allows%20for,and%20perhaps%20a%20new%20perspective..> [Accessed 24 November 2022].
- [3] K. Shives, "The Importance of Asking For Help," Inside Higher ED, 3 November 2015. [Online]. Available: <https://www.insidehighered.com/blogs/gradhacker/importance-asking-help>. [Accessed 24 November 2022].
- [4] W. J. Lammers, "Why Won't They Ask Us for Help?," Faculty Focus, 24 March 2017. [Online]. Available: <https://www.facultyfocus.com/articles/effective-classroom-management/wont-ask-us-help/>. [Accessed 24 November 2022].
- [5] M. & A.-T. L. & A.-S. G. Alawamleh, "The Effect of Online Learning On Communication Between Instructors and Students During Covid-19 Pandemic," Asian Education and Development Studies, 10 August 2020. [Online]. Available: [https://www.researchgate.net/publication/343897561\\_The\\_effect\\_of\\_online\\_learning\\_on\\_communication\\_between\\_instructors\\_and\\_students\\_during\\_Covid-19\\_pandemic](https://www.researchgate.net/publication/343897561_The_effect_of_online_learning_on_communication_between_instructors_and_students_during_Covid-19_pandemic). [Accessed 24 November 2022].
- [6] "Why Don't More Students Ask For Help," YUP, 19 October 2019. [Online]. Available: <https://yup.com/blog/students-ask-for-help/#:~:text=Educational%20psychologist%20Allison%20M.,have%20had%20lower%20academic%20achievement..> [Accessed 24 November 2022].
- [7] "Hi-Hive," Silverlake, [Online]. Available: <https://play.google.com/store/apps/details?id=com.slc.hihive.community&hl=en&gl=US>. [Accessed 24 November 2022].
- [8] "Uni Course Chat," Unifi, [Online]. Available: <https://www.unifi.app/>. [Accessed 24 November 2022].
- [9] "Bronx CC," Ready Education Inc, [Online]. Available: <http://www.bcc.cuny.edu/mobile/>. [Accessed 24 November 2022].
- [10] "Differ.Chat," Differ, [Online]. Available: <https://www.differ.chat/>. [Accessed 24 November 2022].

## REFERENCE

- [11] "Differ.Chat," Differ, [Online]. Available: [https://www.youtube.com/watch?v=etgYmM\\_Z8fY](https://www.youtube.com/watch?v=etgYmM_Z8fY). [Accessed 24 November 2022].
- [12] "Lewis University," Ready Education Inc, [Online]. Available: <https://play.google.com/store/apps/details?id=com.lewisu&hl=en&gl=US>. [Accessed 24 November 2022].
- [13] T. Cox, "What Is Rapid Application Development (RAD)?," Capterra, 14 January 2022. [Online]. Available: <https://www.capterra.com/resources/what-is-rapid-application-development/>. [Accessed 24 November 2022].
- [14] E. Egeonu, "The Advantages and Disadvantages of the RAD Model," Distant Job, 7 June 2022. [Online]. Available: <https://distantjob.com/blog/rad-model-advantages-and-disadvantages/>. [Accessed 24 November 2022].
- [15] "Rapid Application Development (RAD) Model: An Ultimate Guide For App Developers in 2022," Kissflow, 31 10 2022. [Online]. Available: <https://kissflow.com/application-development/rad/rapid-application-development/>. [Accessed 24 November 2022].
- [16] S. Stiner, "Rapid Application Development (RAD): A Smart, Quick And Valuable Process For Software Developers," Forbes, 24 August 2016. [Online]. Available: <https://www.forbes.com/sites/forbestechcouncil/2016/08/24/rapid-application-development-rad-a-smart-quick-and-valuable-process-for-software-developers/?sh=6f2edd8019e8>. [Accessed 24 November 2022].

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Trimester 3</b>	<b>Study week no.: 1</b>
<b>Student Name &amp; ID: Tan Zhi Yuan 19ACB03046</b>	
<b>Supervisor: Dr Tan Joi San</b>	
<b>Project Title: Application Development of VCare Circle for Utarian</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

From advises by supervisor and moderator, I have fixed some of the issues remained in the project mobile application during FYP 1, such as some of the performance issues of unable to delete account properly and also revise the code to remove unwanted parts that affects the overall performance of the project mobile application. I have also had my first physical meeting for this semester with my supervisor to discuss on the upcoming tasks that should be done and also gained some advice on what I should be focusing on to stay on track.

## 2. WORK TO BE DONE

I will continue fixing the performance issue and start on developing the administrator module for the project mobile application.

## 3. PROBLEMS ENCOUNTERED

I have faced issue on the previous widget used in the home page of the project mobile application as I have been using improper methods to receive updates from the Firebase server to update values in the home page widget. This problem is solved after conducting research on better widget such as the stream builder to use streams to listen to updates from Firebase.

## 4. SELF EVALUATION OF THE PROGRESS

My progress is slow, and I will keep on improving myself.



\_\_\_\_\_  
Supervisor's signature



\_\_\_\_\_  
Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Trimester 3</b>	<b>Study week no.: 3</b>
<b>Student Name &amp; ID: Tan Zhi Yuan 19ACB03046</b>	
<b>Supervisor: Dr Tan Joi San</b>	
<b>Project Title: Application Development of VCare Circle for Utarian</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Multiple testing on use cases of the application after changes on the codes for better performance are done to ensure the usability of the function codes. I have also proceeded with the developing of the administrator module, which I have completed the admin login, admin profile and edit admin profile functional modules.

## 2. WORK TO BE DONE

I will continue with the developing of admin control functions such as viewing and editing the list of users and chat history records. I will also continue with the testing of admin login and editing of admin user function.

## 3. PROBLEMS ENCOUNTERED

I have faced issue on the retrieving of history chats due to unable to allocate the document ids of the chats from Firebase server. After conducting research online, I have managed to resolve the issue through implementing a new function to retrieve the chat history.

## 4. SELF EVALUATION OF THE PROGRESS

My progress is slow, and I will try to keep up the pace of my work.



Supervisor's signature



Student's signature



# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Trimester 3</b>	<b>Study week no.: 5</b>
<b>Student Name &amp; ID: Tan Zhi Yuan 19ACB03046</b>	
<b>Supervisor: Dr Tan Joi San</b>	
<b>Project Title: Application Development of VCare Circle for Utarian</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

I have implemented the online calling function for the project mobile application through the adding of Agora SDK for real-time online interaction. I have also added the search user function for admin to easily find users from the Firebase server.

## 2. WORK TO BE DONE

I will continue with editing the UI design of the administrator module to allow simple and clean view on the pages used by the admin. I will also continue with the testing of online calling function to ensure the usability of the function for users and helpers.

## 3. PROBLEMS ENCOUNTERED

I have faced issue on allowing the receiver side to join the online call channel due to the receiver side cannot join the call properly such as allow speaking in call and to end the call. After conducting some research on the Agora documentation, I have realized that I have missed out some settings and have modified them to resolve the issue.

## 4. SELF EVALUATION OF THE PROGRESS

My progress is slow, and I will keep on improving myself.



\_\_\_\_\_  
Supervisor's signature



\_\_\_\_\_  
Student's signature

## FINAL YEAR PROJECT WEEKLY REPORT

*(Project II)*

<b>Trimester, Year: Year 3 Trimester 3</b>	<b>Study week no.: 7</b>
<b>Student Name &amp; ID: Tan Zhi Yuan 19ACB03046</b>	
<b>Supervisor: Dr Tan Joi San</b>	
<b>Project Title: Application Development of VCare Circle for Utarian</b>	

### 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

I have completed the testing of online calling function on the project mobile application, and I have also created an app icon for the project mobile application. I have also implemented a background sending email function for admin to send emails to users of the application without having to access the mobile phone default email app. I have also continued on the UI design for the administrator module pages with simple and clean design.

### 2. WORK TO BE DONE

I will continue on testing the overall implemented function to ensure no error will be found such as during the changes of user roles, the modifying of user data by admin and also the changing of admin email.

### 3. PROBLEMS ENCOUNTERED

During the implementing of the background email sender function, I have issues in getting the SMTP server to access the email account for sending emails. After some research done, I have resolved the issue by setting the email used as the sender email to enable the use of app password for authenticating the function's access to use the email account.

### 4. SELF EVALUATION OF THE PROGRESS

My progress is slow, and I will keep on improving myself.



Supervisor's signature



Student's signature

# FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

<b>Trimester, Year: Year 3 Trimester 3</b>	<b>Study week no.: 8</b>
<b>Student Name &amp; ID: Tan Zhi Yuan 19ACB03046</b>	
<b>Supervisor: Dr Tan Joi San</b>	
<b>Project Title: Application Development of VCare Circle for Utarian</b>	

## 1. WORK DONE

[Please write the details of the work done in the last fortnight.]

I have attended a meeting with supervisor this week to ask for some advice on my current progress and what to proceed at this point. I have also proceeded to conduct some modification of the code based on the suggestion of supervisor for better overall usability of program application.

## 2. WORK TO BE DONE

I will continue on modifying the code based on the supervisor's advice and continue with making changes on the administrator module to be more suitable in practical use case. I will also start on writing the Final Year Project 2 Report.

## 3. PROBLEMS ENCOUNTERED

During the process of allowing admin to change email account as the admin profile, I have faced issue of unable to conduct proper transferring of old admin email account profile back to normal account which causes defect in using the functions in the project mobile application. After conducting several modifications and also adjustment in the Firebase server, I have managed to solve this issue.

## 4. SELF EVALUATION OF THE PROGRESS

My progress is slow, and I will keep on improving myself.



\_\_\_\_\_  
Supervisor's signature



\_\_\_\_\_  
Student's signature

POSTER

**VCARE CIRCLE**

**INTRODUCTION**

Asking for help is an effective way for people to solve their issue on hand when they face difficulties in solving it alone.

**OBJECTIVES**

- Improve Effectiveness In Asking For Help
- Reduce Future Disturbance and Worry

**Methods**

- ❑ Methodology : Rapid Application Development
- ❑ Software: VS Code, Android Studio, Firebase
- ❑ Libraries: Dart, Flutter

**Function Modules**

- Login/Sign Up
- Profile
- Online Communication
- Helper Finder
- QR Identification
- Offline Help

**Conclusion**

A mobile application that automatically assigns helper to user is delivered , which contributes an effective way of asking for help, and solves the privacy concerns of users and helpers.

Universiti Tunku Abdul Rahman  
Bachelor Of Computer Science (Honours)  
Prepared by Tan Zhi Yuan (19ACB03046)

**UTAR**  
UNIVERSITI TUNKU ABDUL RAHMAN

## PLAGIARISM CHECK RESULT

19ACB03046\_FYP2

## ORIGINALITY REPORT

<b>2</b> %	<b>1</b> %	<b>1</b> %	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

## PRIMARY SOURCES

<b>1</b>	"Chapter 5 Communicating and Collaborating", Springer Science and Business Media LLC, 2007 Publication	<1%
<b>2</b>	etheses.bham.ac.uk Internet Source	<1%
<b>3</b>	hdl.handle.net Internet Source	<1%
<b>4</b>	Felicity Brand, Heather McNamee, Jeffrey A. McGuire. "The TYPO3 Guidebook", Springer Science and Business Media LLC, 2021 Publication	<1%
<b>5</b>	scholar.ppu.edu Internet Source	<1%
<b>6</b>	studentsrepo.um.edu.my Internet Source	<1%
<b>7</b>	www.ece.ucf.edu Internet Source	<1%
<b>8</b>	docplayer.net Internet Source	<1%

9	<a href="http://www.ijraset.com">www.ijraset.com</a> Internet Source	<1 %
10	Walaa H. Elashmawi, Ahmad Akram, Mohammed Yasser, Menna Hisham, Manar Mohammed, Noha Ihab, Ahmed Ali. "IOT Based Smart Parking System Using Ensemble Learning", Intelligent Automation & Soft Computing, 2023 Publication	<1 %
11	<a href="http://things.maths.cam.ac.uk">things.maths.cam.ac.uk</a> Internet Source	<1 %
12	<a href="http://utpedia.utp.edu.my">utpedia.utp.edu.my</a> Internet Source	<1 %
13	"Beginning ASP.NET 2.0 in C# 2005", Springer Science and Business Media LLC, 2006 Publication	<1 %
14	<a href="http://www.george.gov.za">www.george.gov.za</a> Internet Source	<1 %
15	<a href="http://zarthur.gitbooks.io">zarthur.gitbooks.io</a> Internet Source	<1 %
16	<a href="http://mafiadoc.com">mafiadoc.com</a> Internet Source	<1 %
17	<a href="http://essay.utwente.nl">essay.utwente.nl</a> Internet Source	<1 %
18	<a href="http://www.blm.gov">www.blm.gov</a>	

	Internet Source	<1 %
19	<a href="http://www.ramp-alberta.org">www.ramp-alberta.org</a> Internet Source	<1 %
20	<a href="http://github-wiki-see.page">github-wiki-see.page</a> Internet Source	<1 %
21	<a href="http://ir.unimas.my">ir.unimas.my</a> Internet Source	<1 %
22	<a href="http://www.geocities.ws">www.geocities.ws</a> Internet Source	<1 %
23	Nor Saradatul Akmar Zulkifli, Mohammad Ridwan Satrial, Mohd Zamri Osman, Nor Syahidatul Nadiah Ismail et al. "IoT-Based Smart Environment Monitoring System for Air Pollutant Detection in Kuantan, Pahang, Malaysia", IOP Conference Series: Materials Science and Engineering, 2020 Publication	<1 %
24	<a href="http://www.answerbag.com">www.answerbag.com</a> Internet Source	<1 %
25	<a href="http://www.oecd.org">www.oecd.org</a> Internet Source	<1 %
26	<a href="http://discourse.slicer.org">discourse.slicer.org</a> Internet Source	<1 %

27	"Chapter 5 Communicating with Customers", Springer Science and Business Media LLC, 2005 <small>Publication</small>	<1 %
28	David Powers. "Getting StartED with CSS", Springer Science and Business Media LLC, 2009 <small>Publication</small>	<1 %
29	Henry Lee, Eugene Chuvyrov. "Chapter 5 Packaging, Publishing, and Managing Applications", Springer Science and Business Media LLC, 2010 <small>Publication</small>	<1 %
30	usermanual.wiki <small>Internet Source</small>	<1 %
31	www.chulavistaca.gov <small>Internet Source</small>	<1 %
32	www.coursehero.com <small>Internet Source</small>	<1 %

Exclude quotes  On  
 Exclude bibliography  On

Exclude matches  < 8 words



<b>Universiti Tunku Abdul Rahman</b>			
<b>Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)</b>			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



**FACULTY OF INFORMATION AND COMMUNICATION  
TECHNOLOGY**

<b>Full Name(s) of Candidate(s)</b>	Tan Zhi Yuan
<b>ID Number(s)</b>	19ACB03046
<b>Programme / Course</b>	Bachelor of Computer Science (Honours)
<b>Title of Final Year Project</b>	Application Development of VCare Circle For Utarian

<b>Similarity</b>	<b>Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)</b>
<b>Overall similarity index: <u>2</u> %</b>  <b>Similarity by source</b> Internet Sources: <u>1</u> % Publications: <u>1</u> % Student Papers: <u>0</u> %	No comment
<b>Number of individual sources listed of more than 3% similarity: <u>0</u></b>	No comment
<b>Parameters of originality required and limits approved by UTAR are as Follows:</b> (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

*Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.*

\_\_\_\_\_  
Signature of Supervisor

Name: Dr Tan Joi San

Date: 21/4/2023

\_\_\_\_\_  
Signature of Co-Supervisor

Name: \_\_\_\_\_

Date: \_\_\_\_\_



## UNIVERSITI TUNKU ABDUL RAHMAN

### FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

#### CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	19ACB03046
Student Name	Tan Zhi Yuan
Supervisor Name	Dr Tan Joi San

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
N/A	Front Plastic Cover (for hardcopy)
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
N/A	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

\*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

\_\_\_\_\_  
(Signature of Student)

Date: 21/4/2023