

**A MOBILE APP FOR SUPERMARKET
CHECKOUT**

SIEW SHUN YAO

UNIVERSITI TUNKU ABDUL RAHMAN

A MOBILE APP FOR SUPERMARKET CHECKOUT

SIEW SHUN YAO


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

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

September 2022

DECLARATION

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

Signature :  _____

Name : SIEW SHUN YAO _____

ID No. : 1905076 _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**A MOBILE APP FOR SUPERMARKET CHECKOUT**” was prepared by **SIEW SHUN YAO** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Hons) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Ms Chean Swee Ling

Date : 30 Sept 2022

Signature : _____

Co-Supervisor : _____

Date : _____

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

© 2022, Siew Shun Yao. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Ms Chean Swee Ling for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement throughout the process of completing the report. There would be impossible to complete this project without their patient and supporting given by them.

Lastly, I would like to thank all the authors and contributor of the documentation and books I referred. Their research, findings and contributions had provided me solutions when I encountered obstacles along with whole project.

ABSTRACT

Existing supermarket's self-service technology requires a high cost when come to installing a self-service machine together with maintaining cost in the future. Besides, checkout manually at cashier counter in supermarket has resulted time consuming especially during the peak hour and vacation day. This led to overall of time operation consuming has caused stressful and contradiction due to be productivity which can provide the best service to consumer's need. Therefore, the goal of proposed project aims to examine and investigate the situation's underlying causes to offer a remedy. The proposed system that involved ionic framework to build web application for supermarket merchant and mobile application to checkout items themselves. Angular js is used to build the frontend component while backend server has involved AWS EC2 server to be API endpoints to communicate with frontend client and store data into PostgreSQL database. Furthermore, iteration and incremental model methodology was practiced throughout the project life cycle to achieve manage project timeline and follow up the system progress. In addition, the proposed project involved various of testing such as unit testing, integration testing and UAT testing, and all testing cases is accepted and passed. The proposed system reduced the queuing time and operation time of scanning item by cashier worker when mobile application provides self-service of scanning items and checkout through online especially consumer with less items.

TABLE OF CONTENTS

DECLARATION	ii
APPROVAL FOR SUBMISSION	iii
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	xii
LIST OF FIGURES	xiv
LIST OF SYMBOLS / ABBREVIATIONS	xxii
LIST OF APPENDICES	xxiii

CHAPTER

1	INTRODUCTION	1
	1.1 Introduction	1
	1.2 Background of the project	1
	1.3 Problem Statement	2
	1.3.1 Supermarket checkout app are not convenient for senior consumers	2
	1.3.2 Technical Issues	3
	1.3.3 Threat of Theft	3
	1.4 Project Objectives	3
	1.5 Project Solution	4
	1.6 Project Approach	6
	1.7 Scope of the Project	7
	1.7.1 Target User Scope	7
	1.7.2 Target Merchant administrators Scope	7
	1.7.3 Target security administrators Scope	7
	1.7.4 System Scope	8
	1.7.5 Modules Covered	8
	1.7.6 Module Not Covered	10

2	LITERATURE REVIEW	12
2.1	Introduction	12
2.2	Review on Similar Self-checkout system	12
2.3	Review on Project Methodology	18
2.3.1	Waterfall	18
2.3.2	Spiral	19
2.3.3	Iterative and incremental model	19
2.3.4	Comparison of Methodology	20
2.4	Review on Backend Server Framework	21
2.4.1	Laravel	22
2.4.2	Node.js	22
2.4.3	Comparison of Backend Framework	23
2.5	Review on Frond-end Web Application Framework	24
2.5.1	Angular.js	24
2.5.2	Vue.js	24
2.5.3	Comparison of Front-end Web Application Framework	25
2.6	Review on Cross-platform Mobile Application Framework	26
2.6.1	Ionic Framework	26
2.6.2	React Native	27
2.6.3	Comparison of Cross-platform Mobile Application Framework	27
2.7	Review on Cloud Computing Services	28
2.7.1	Firebase (Google cloud platforms)	28
2.7.2	AWS EC2	29
2.7.3	Comparison of Cloud Computing Services	29
3	METHODOLOGY AND WORK PLAN	31
3.1	Introduction	31
3.2	Development Methodology	31
3.3	Proposed Workplan	33
3.4	Technology and Development Tools Involved	39
3.4.1	Nodejs	39
3.4.2	Angular.js	39

3.4.3	Ionic Framework	39
3.4.4	PostgreSQL	39
3.4.5	AWS EC2	40
3.4.6	Firebase	40
3.4.7	Visual Studio Code	40
3.4.8	Git	40
3.4.9	Figma	40
3.4.10	Trello	41
4	PROJECT INITIAL SPECIFICATION	42
4.1	Introduction	42
4.2	Fact-finding	42
4.2.1	Observation	42
4.2.2	Questionnaire	45
4.3	Requirement Specification	53
4.3.1	Customer-side mobile application	53
4.3.2	Merchant-side web application	54
4.3.3	Administrative web application	55
4.3.4	Non-Function Requirements	55
4.4	Use Case Modelling	56
4.4.1	Use Case Diagram	56
4.5	Use Case Description	64
4.5.1	Mobile Application for Supermarket Self-Checkout	64
4.5.2	Web-based application for Merchant Users	68
4.5.3	Web-based application for Administrator	73
5	SYSTEM DESIGN	77
5.1	Introduction	77
5.2	System Architecture	77
5.3	System Design Pattern	79
5.4	Database Design	81
5.4.1	Physical Entity Relationship Diagram	81
5.4.2	Logical Entity Relationship Diagram	82

	5.4.3	Data Dictionary	83
5.5		User Interface Design	91
	5.5.1	Web Application for Merchant	91
	5.5.2	Supermarket Normal User	97
	5.5.3	Supermarket senior user	107
6		SYSTEM IMPLEMENTATION	112
6.1		Backend Server	112
	6.1.1	Overview backend server	112
	6.1.2	Controller Layer	114
	6.1.3	Service Layer	132
	6.1.4	Model Layer	136
	6.1.5	Other Integration	139
	6.1.6	Available Endpoints	145
6.2		Mobile application for supermarket self-checkout	150
	6.2.1	Overview of Mobile Application	150
	6.2.2	Pages Hierarchy	153
	6.2.3	Deployment	154
6.3		Web application for merchant side managements	155
	6.3.1	Overview of Web Application	155
	6.3.2	Pages Hierarchy	156
	6.3.3	Deployment	157
6.4		Web application for administrative side managements	158
	6.4.1	Overview of Web Application	158
	6.4.2	Pages Hierarchy	159
7		SYSTEM TESTING	160
7.1		Introduction	160
7.2		Unit Testing	160
	7.2.1	User Module	160
	7.2.2	Merchant Module	164
7.3		Integration Testing	167
7.4		User Acceptance Testing	173

8	CONCLUSIONS AND RECOMMENDATIONS	196
8.1	Conclusions	196
8.2	Limitations	197
8.3	Recommendations for future work	198
	REFERENCES	199
	APPENDICES	203

LIST OF TABLES

Table 2.1: Table of comparison between various software methodologies	20
Table 2.2: The comparison between Laravel and Node js backend framework	23
Table 2.3: The comparison between Angular and Vue.js frontend framework	25
Table 2.4: Table of comparison in Ionic framework and React Native framework.	27
Table 2.5: Table of comparison between AWS EC2 and firebase.	30
Table 3.1: Project Schedule Summary	33
Table 6.1 Mobile Endpoints Listing	146
Table 6.2 Merchant Endpoints Listing	148
Table 7.1 Unit testing of user module	160
Table 7.2 Unit testing of merchant module	164
Table 7.3 Integration testing	167
Table 7.4: Register module for supermarket user	173
Table 7.5: Login module for supermarket user	174
Table 7.6: Profile module for supermarket user	175
Table 7.7: Notification module for normal user	176
Table 7.8: Cart module for supermarket user	177
Table 7.9: Order module for supermarket user	179
Table 7.10: Favourite module for normal user	179
Table 7.11: Feedback module for normal user	181
Table 7.12: Search module for normal user	182
Table 7.13: Login module for merchant user	183

Table 7.14: Manage product module for merchant user	184
Table 7.15: Feedback module for merchant user	186
Table 7.16: Notification module for merchant user	188
Table 7.17: Profile module for merchant user	190
Table 7.18: Advanced search module for merchant user	191
Table 7.19 User Acceptance Testing Summary Result (Supermarket normal user)	193
Table 7.20 User Acceptance Testing Summary Result (Supermarket senior user)	193
Table 7.21 User Acceptance Testing Summary Result (Merchant user)	194

LIST OF FIGURES

Figure 1.1: Senior-Aged user interface in low fidelity.	4
Figure 1.2: Senior-Aged user interface in low fidelity.	5
Figure 1.3: Firebase. Figure 1.4: PostgreSQL.	6
Figure 1.5: Iterative and Incremental Model.	7
Figure 2.1: Lotus' self-checkout Counter.	13
Figure 2.2: Lotus' self-checkout hardware.	14
Figure 2.3: Scan with user QR code to sign into the store.	16
Figure 2.4: Make request for refund for eaten food.	17
Figure 2.5: Waterfall Model.	18
Figure 2.6: Spiral Model.	19
Figure 2.7: Iterative and incremental model.	20
Figure 2.8: Laravel framework.	22
Figure 2.9: Node js.	23
Figure 2.10: Angular js.	24
Figure 2.11: Vue.js.	25
Figure 2.12: Ionic Framework.	26
Figure 2.13: React Native.	27
Figure 2.14: Firebase.	29
Figure 2.15: Amazon EC2.	29
Figure 3.1: SDLC.	31
Figure 3.2: Gantt Chart for the project.	35
Figure 3.3: Gantt Chart for the project.	36
Figure 3.4: Work breakdown diagram of project.	37

Figure 4.1: Pie chart of Gender of Respondents.	45
Figure 4.2: Pie chart of age group of Respondents.	46
Figure 4.3: Pie chart of paying at cashier counter related question.	46
Figure 4.4: Pie chart of experiencing self-checkout related question.	47
Figure 4.5: Pie chart of experiencing self-checkout related question.	47
Figure 4.6: Bar chart of place that done self-checkout related question.	48
Figure 4.7: Bar chart of rating self-checkout related question.	49
Figure 4.8: Pie chart of self-checkout related question.	50
Figure 4.9: Bar chart of self-checkout related question.	51
Figure 4.10: Pie chart of self-checkout related question.	51
Figure 4.11: Bar chart of features of self-checkout related question.	52
Figure 4.12: User checkout use case diagram	56
Figure 4.13: User activity history use case diagram	57
Figure 4.14: User account related use case diagram	58
Figure 4.15: Notification related use case diagram	59
Figure 4.16: Feedback related use case diagram	60
Figure 4.17: Manage products use case diagram	61
Figure 4.18: advance search function use case diagram	62
Figure 4.19: Manage merchant use case diagram	62
Figure 4.20: Manage customer news use case diagram	63
Figure 4.21: Location use case diagram	63
Figure 5.1: Overview of System Architecture	77
Figure 5.2: MVC architecture.	79

Figure 5.3: Physical Entity Relationship Diagram.	81
Figure 5.4: Logical Entity Relationship Diagram.	82
Figure 5.5: Merchant user login page screen.	91
Figure 5.6: Merchant user manage product page screen.	92
Figure 5.7: Merchant user add new product page screen.	92
Figure 5.8: Merchant user add new product continue page screen.	93
Figure 5.9: Merchant user manage feedback list page screen.	93
Figure 5.10: Merchant user manage specific feedback page screen.	94
Figure 5.11: Merchant user Reply specific feedback page screen.	94
Figure 5.12: Merchant user manage notification page screen.	95
Figure 5.13: Merchant user create notification modal.	95
Figure 5.14: Merchant user edit profile page screen.	96
Figure 5.15: Merchant user edit notification page screen.	96
Figure 5.16: Login page screen.	97
Figure 5.17: Register page screen.	97
Figure 5.18: Home page screen.	98
Figure 5.19: Notification page screen	98
Figure 5.20: Cart page screen.	99
Figure 5.21: Profile page screen.	99
Figure 5.22: Edit profile page screen.	100
Figure 5.23: Order History page screen.	100
Figure 5.24: Favorite page screen.	101
Figure 5.25: Feedback page screen.	101
Figure 5.26: Feedback form page screen.	102

Figure 5.27: Feedback details page screen.	102
Figure 5.28: Scan item screen.	103
Figure 5.29: Add item page screen.	103
Figure 5.30: Order details page screen.	104
Figure 5.31: News details page screen.	104
Figure 5.32: Merchant List page screen.	105
Figure 5.33: Settings page screen.	105
Figure 5.34: Recover Password page screen.	106
Figure 5.35: Notification details page screen.	106
Figure 5.36: Senior user login page screen.	107
Figure 5.37: Senior user register page screen.	107
Figure 5.38: Senior user home page screen.	108
Figure 5.39: Senior user edit profile page screen.	108
Figure 5.40: Senior user cart page screen.	109
Figure 5.41: Senior user scan screen.	109
Figure 5.42: Senior user order history page screen.	110
Figure 5.43: Senior user order details page screen.	110
Figure 5.44: Senior user profile page screen.	111
Figure 5.45: Senior user settings page screen.	111
Figure 6.1: Overview the RESTful API request from mobile application.	112
Figure 6.2: Overview the RESTful API request from web application for merchant side.	113
Figure 6.3: Login Controller Authentication for User.	114
Figure 6.4: Login Controller Login Function for User.	114
Figure 6.5: Login Authentication for Merchant.	115

Figure 6.6: Register Controller Source Code.	116
Figure 6.7: Register for merchant account in Admin panel Source Code.	117
Figure 6.8: Notification Controller code segment.	117
Figure 6.9: Create Notification function from merchant side notification controller.	118
Figure 6.10: Cart Controller code segment.	119
Figure 6.11: Cart Controller checkout function source code.	120
Figure 6.12: Profile Controller code segment in user side.	121
Figure 6.13: Profile Controller code segment in user side.	121
Figure 6.14: Update function in edit profile controller in user side.	122
Figure 6.15: Edit Profile Controller code segment in merchant side.	122
Figure 6.16: Edit Profile Controller update function for merchant.	123
Figure 6.17: Feedback Controller code segment for user.	124
Figure 6.18: Feedback Controller create function in user side.	124
Figure 6.19: Feedback Controller code segment for merchant user.	125
Figure 6.20: Feedback Details Controller code segment for merchant user.	125
Figure 6.21: Feedback Details Controller function for merchant user.	126
Figure 6.22: Feedback Details Controller function for merchant user.	126
Figure 6.23: Order History Controller code segment for user.	126
Figure 6.24: Display Store Controller function.	127
Figure 6.25: Display Store Controller code segment.	128

Figure 6.26: Display Store Controller favorite function source code.	128
Figure 6.27: Tabs Controller code segment.	129
Figure 6.28: Tabs Controller function.	129
Figure 6.29: Tabs Controller Source Code.	130
Figure 6.30: Read news code segment in customer side.	130
Figure 6.31: Create news function code segment in admin side.	131
Figure 6.32: User service source code.	132
Figure 6.33: Feedback service source code.	133
Figure 6.34: Order service source code.	133
Figure 6.35: Product service source code.	134
Figure 6.36: Notification service source code.	134
Figure 6.37: News service source code.	135
Figure 6.38: User model source code.	136
Figure 6.39: Order model source code.	137
Figure 6.40: Feedback model source code.	137
Figure 6.41: Product model source code.	137
.	137
Figure 6.42: Notification model source code.	138
Figure 6.43: News model source code.	138
Figure 6.44: billplz api endpoints at server side.	139
Figure 6.45: billplz api code segment call from server side.	139
Figure 6.47: Firebase cloud messaging code segment in app.component.ts.	141
Figure 6.48: Firebase cloud messaging fcm function in app.component.ts.	141

Figure 6.49: Notification send to topic 'user' when merchant created a notification.	142
Figure 6.50: AWS S3 source code on backend.	142
Figure 6.51: s3.js on backend.	143
Figure 6.52: Upload file API endpoint on backend.	143
Figure 6.53: Edit profile's upload image code segment.	144
Figure 6.54: AWS EC2 server.	145
Figure 6.55: Mobile application normal mode screen layout.	150
Figure 6.56: Mobile application simple mode screen layout.	151
Figure 6.57: Verify user code segment.	152
Figure 6.58: Switch case between normal mode and simple mode.	152
Figure 6.59: Login file.	152
Figure 6.60: Normal mode page hierarchy.	153
Figure 6.61: Simple mode page hierarchy.	154
Figure 6.62: Mobile application deployment.	154
Figure 6.63: Web application screen layout.	155
Figure 6.64: Home.page.html source code in collapse.	156
Figure 6.65: Pages Hierarchy of merchant side web application.	156
Figure 6.66: Pages Hierarchy of merchant side web application.	157
Figure 6.67: Administrative web application screen layout.	158
Figure 6.68: Home.page.html source code in collapse.	158
Figure 6.69: Pages hierarchy of administrative side.	159
Figure 7.1: Jasmine Framework V3.8.0 – Karma test runner V6.3.20.	162
Figure 7.2: Testing providers and imports for unit testing.	162

Figure 7.3: Code segment of display news http test in news service.	163
Figure 7.4: Unit test results of mobile user.	163
Figure 7.5: Jasmine Framework V3.8.0 – Karma test runner V6.3.20.	165
Figure 7.6: Testing providers and imports for unit testing.	165
Figure 7.7: Dummy product in unit testing for product service.	166
Figure 7.8: Code segment of create product http test in product service.	166
Figure 7.9: Unit test results of merchant module.	167
Figure 7.10: Karma framework test debug result.	170
Figure 7.11: Integration test suite source code collapse.	171
Figure 7.12: Feedback test suite code segment.	172
Figure 7.13: Integration test performance.	172

LIST OF SYMBOLS / ABBREVIATIONS

SST	self-service technology
SSM	maximum allowable pressure, kPa
PIC	person in charge
SDLC	software development life cycle
SKU	stock keeping unit
ORM	object-relational mapping
MVC	model view controller
MVVM	model view view-model
DOM	document object model
OOPS	object-oriented programming
UI	user interface
UX	user experience
HTML	hyperlink text markup language
SCSS	sassy cascading style
IT	information technology
API	application programming interface
REST	representational state transfer
UID	unique identifier
VH	view height
VW	view width
SDK	software development kit
SQL	structured query language
CLI	command line interface
SRS	software requirement specification
HTTP	hypertext transfer protocol
CRUD	create, read, update, delete
URL	uniform resource locator
AWS	amazon web services
FCM	firebase cloud messaging
UAT	user acceptance testing
QR	quick response

LIST OF APPENDICES

APPENDIX A: Observation	203
APPENDIX B: Questionnaire Form	205
APPENDIX C: Supervisor and Moderator Comments on Project Plan	208
APPENDIX D: Project Progress (Trello Web Application)	211

CHAPTER 1

INTRODUCTION

1.1 Introduction

The purpose of designing a supermarket checkout mobile application is to reduce the waiting time or queuing time. It also purposed to reduce of manpower and training time. However, this mobile application is also managed stock categories and stock control. It provides bulk upload services which can greatly reduce product creation time.

1.2 Background of the project

In this urban life, technology is evolved rapidly as it can reduce the engagement of manpower which brings the focus to business flow. In common, as we all know a hypermarket have cashier which is the service provider that can serve consumers (Hassan, Sade and Rahman, 2013). From the report, the researcher has stated that a hypermarket cashier is able to handle items which between five hundred to one thousand and fill more than 80 bags per hour (Hassan, Sade and Rahman, 2013).

In a hypermarket, cashier should follow standard working flow and it come to repetitive. The cashier might lead to stressful and contradiction due to be productivity which can provide the best service to consumer's need. The time-consuming during products checkout is taking long when there are many consumers. While time is perceived differently by various customers, time restrictions cause consumers to be more concerned with the duration of the activity. (Chetthamrongchai and Davies, 2000). In the report, it stated that, there is a waiting time which is include, selection time, queue time and transaction time. The operation of cashier activities taken 60% of overall activities which one is scan the items that occupied 20%, to packing the purchased items which in 18% and payment is 22%. The others time is accounted for other service such as changing items, asking direction and so on. As a result, the cost of service is out of balance compared to cost of waiting in hypermarket. (Hassan, Sade and Rahman, 2013).

A marketing checkout is a mobile application which a SST (self-service checkout technology). The purpose of this mobile application is to reduce the manpower in hypermarket which also to reduce the service time and waiting time. It also can improve the checkout efficiency. In the perspective of merchant, this

application helps to manage stock categories and stock control. It provides bulk upload services which can greatly reduce product creation time.

In this case, consumers can have a self-service for items checkout which means it can be reduced the physical checkout counter and replace with items verify counter. Items verify counter is to verify consumer's checkout items is match with the cart checkout list to prevent dishonest behaviour during checkout process. In the physical cashier service counter, it possible to have long queuing or some consumer would carry a lot of items in their trolley which could taking long processing time. With this application, it can reduce the waiting time and processing time which designed for impatient consumer and urgent consumer to have a self-checkout and leave. For the consumer that carry a lot of items, they can choose to wait for physical counter to checkout, by this, the physical checkout operation will be smooth, and both is having a balanced time. The verify checkout counter has standardised the workflow and allows hypermarkets to hire non-skilled staff who can be trained in a short amount of time. (Bernard, 2007).

1.3 Problem Statement

1.3.1 Supermarket checkout app are not convenient for senior consumers

Supermarket checkout application might be a complication and trouble for those who cannot adopt a new technology in this new era (Hassan, Sade and Rahman, 2013). Although there are several applications available that cater to the diverse demands of senior citizen users, there is a lack of awareness of how the usability functions of these apps impact their adoption among senior consumer (Thamutharam, Mustafa, Musthafa, Tajudeen, 2021). Some of senior consumer might not educated in their era unlike nowadays children at least to graduate in secondary school in Malaysia. As a result, the senior age has a negative inclination for using self-service technology owing to a lack of confidence, fewer human interaction, and the features of self-service technology. In the report, the researcher found that degree to which human interaction is missed for middle-aged customer is occupied 4.95% in total of 234 persons and elder age customer is occupied 5.98% in total of 251 persons, by this, it can be concluded that elder customer is having few confidences to using self-checkout technology while middle-aged customer is comfortable to have less human interaction. (Dean, 2008)

1.3.2 Technical Issues

Some technical issues that might not be avoided which an unreadable barcode on a product is possible. In the report, the researcher found that barcode unreadable or not able to be scanned for students occupied 63.16% which 12 persons among 20 persons, while non-students occupied 79.49% which 31 persons among 39 persons. It also stated that scanning the products with unusual forms and textures can be problematic. Sometimes it will need attendant to assist those customers who face scanning issues (Mendat and Mayhorn, 2007). Self-checkout service might not be stable on hardware and customer behaviors. Reason of unstable system is depending on the hardware response speed. When implementing self-checkout service might have a high investment cost for purchasing hardware, it might not be able to have any extra hardware as possible to prevent any situations happen. Self-checkout service would cause a lot of confusion as it will chain up his impatient and affecting others consumer. A consumer, on the other hand, who encounters checkout failure may struggle to deal with the issue, especially if there are no personnel there. (iStrategy Conference, 2021)

1.3.3 Threat of Theft

Self-checkout service might not be secure to merchant. Main reason is the system relying on the behavior of the consumer when it is checking out his cart, as they should be responsible to the purchasing flow. This makes chance to shoplift while the shopper has a dishonest behavior, it might be purposely not to scan some of the item or directly perform a shoplift. When consumer has carries large amount of items system will not be able to verify the consumer had paid according to his/her checkout list. (Dwyer, 2019) The researcher found out that at merchants in the United States, the United Kingdom, and Europe and discovered that retailers who use self-checkout service had a loss rate of around 4%, which is approximately double the industry average. (Insider Intelligence, 2016).

1.4 Project Objectives

1. To investigate the current problem of supermarket checkout process time when dealing a huge number of customers.
2. To analyze the public acceptance of self-checkout service in supermarket.

3. To develop a supermarket self-checkout mobile application which provides self-payment service that helps supermarket to reduce the labor costs and checkout process time.

1.5 Project Solution

To resolve the problem stated above, there are a few solutions can be executed. Creating two types of user interface which can be differentiate with elder citizens and normal user. Elder citizen interface can be designed with easy step to checkout which has fewer functions can be executed. By creating this interface whoever user if he/she can understand the language, he/she should be able to operate the system very easily. Physically adding verify counter, it is used to verify each receipt that items is matched with the consumer's cart to prevent any dishonest behavior, the checkout flow is faster than a normal physical counter as they need to scan, packing and payment all by the counter. When it comes to unreadable barcode, the proposed solution is that system will use QR code instead of using barcode. The barcode sometimes not suited with phone scanner as there are various types of barcodes whereas QR code is more easily to scan with phone scanner. QR code is a kind of barcode that a digital device can read easily and that encodes data as a series of pixels in a grid of squares. Besides, when consumer encountered other kind of technical issues, they can write feedback to merchant side through mobile app or consult physically with the personnel around the validation counter.

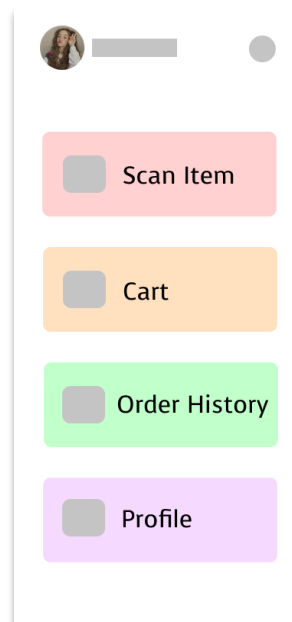


Figure 1.1: Senior-Aged user interface in low fidelity.

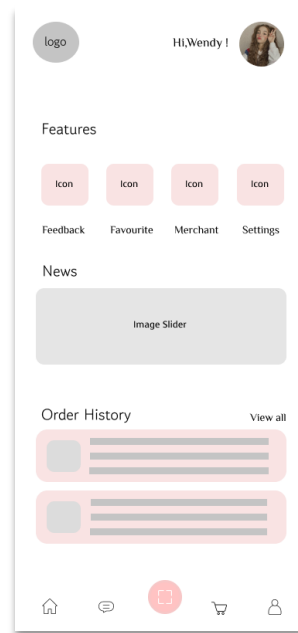


Figure 1.2: Senior-Aged user interface in low fidelity.

As diagram shown, simplified interface which has lesser functions, but it is easy to understand. This is suitable for all age users which include elder citizens. If they need any supports while operating the system, supermarket employee can explain each button easy. For the full functions interface, there is much more functions user can be discovered, like checking history, all item list, or merchant nearby. To consider elder citizen doesn't really use for the functions, in the simplify interface will not display the functions, but if some of the citizen wish to use the function, they can change the interface with a button by time to time.

All the information will store on our PostgreSQL Database. Comparing to Firebase Database which provide by Google, costs will greatly reduce while there are many users and data with Postgres. Some of the module will use the Firebase to make it more flexible. Firebase will mainly use on the authentication, FCM (Firebase Cloud Messaging), Website hosting and Cloud Server hosting. By the combination of 2 services, the system can take most of the scenarios and even secure for any further upgrade.



Figure 1.3: Firebase.



PostgreSQL

Figure 1.4: PostgreSQL.

1.6 Project Approach

In this project, the software development approach applies iterative and incremental model. Iterative and incremental model is defined as cyclical process model. Each iterative process is initial with a simple planning and implementation of the software requirements. It supports iteratively enhancement of the system versions until the completed versions is implemented. The iterative model supports modifications at each iterative level which can add new functional capabilities or changes of system design. This model is basically to develop a system within a cycling flow with a smaller portion at a time (Airbrake, 2016). It called as evolutionary acquisition approach since more than one iterative may be processed in once. During the incrementation, whole process is run separately. Each iteration will consider requirement, analysis, design, development, and validation which individually brings along with a mini project (Kananke, 2020). Each succeeding module release adds functionality to the preceding iteration. The method is repeated until the entire system meets the requirements. (tutorialspoint, n.d.)

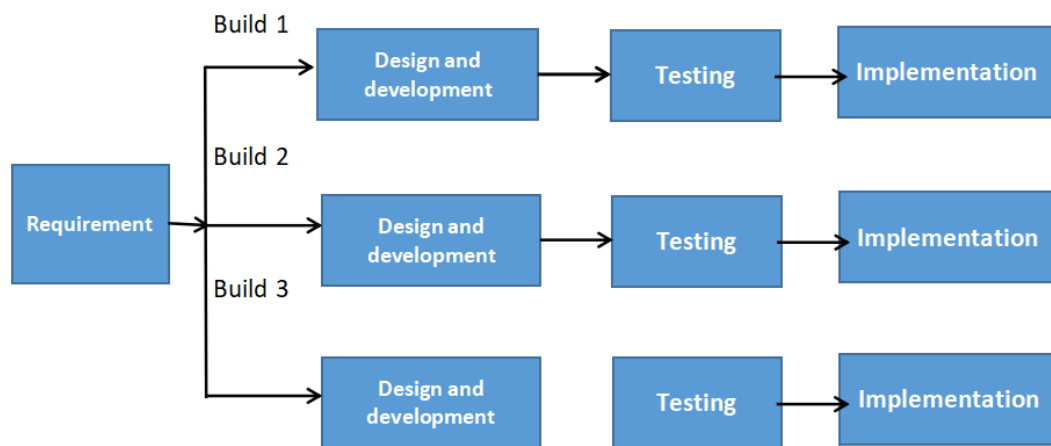


Figure 1.5: Iterative and Incremental Model.

1.7 Scope of the Project

This mobile application is limited to supermarket customer, supermarket merchant, and application administrator only. This mobile application can access using mobile devices on android platform only. This project will deliver three interfaces which customer-side is using a mobile application, while merchant-side and administrator-side is using a web application.

1.7.1 Target User Scope

In this project, supermarket customer play role as end-users which they are allowed to scan and pay through self-checkout application. They also allowed to search and view history and get notification if any update events in the application.

1.7.2 Target Merchant administrators Scope

Supermarket manager and staff play role as the merchant admin which they are administrator that can manage upload categories of items, organize the products, or add price for each product.

1.7.3 Target security administrators Scope

Security administrations play role as the security management which they can manage merchant accounts such as manage merchant side security information and also view list of all registered merchant.

1.7.4 System Scope

System target on android devices. Programming language will mainly use on IONIC which is a hybrid platform, this will secure further upgrade when it expands to iOS application. Visual Studio Code will take part as the development tools to develop the full functions web application. This web application allows user to scan & pay through the application which will have 2 interface to separate elders and youth. Merchant dashboard shall be able to manage supermarket products and create notification to users.

1.7.5 Modules Covered

The modules listed below are to meet the project's objectives. Basically, split into 2 web application and 1 mobile application.

Customer-side mobile application

1.7.5.1 User Scan & Cart Module (Mobile Application)

User can open camera and scan the product QR code. After scanning the codes, it will display the product details and add it into cart. By adding into cart, user may checkout current item available in cart and proceed to payment.

1.7.5.2 User Activities log & Transaction History (Mobile Application)

User can track the item that they scanned before from activities log. This can be ensured users doesn't duplicate scan on the items, may also prevent any argument on the verify table with duplicate scan. Transaction history displays user purchase history.

1.7.5.3 User profile (Mobile Application)

User can be easily registered their account with personal information, name, contact, date of birth, email, address etc. User can update their profile information in applications.

1.7.5.4 Product Specific on location Module (Mobile Application)

User can detect nearby merchant by using locations. Once the user entered a shop with the service subscriptions, product list will auto display based on the merchant nearby. Inside application will display a list of the merchant's registered product

(SKU). If there are multiple merchants nearby, the system will pop up a list that asked user to select the merchant that he/she entered.

1.7.5.5 Search Function (Mobile Application)

User can search the specific item or specific merchant by merchant name. It is also to create a chance for merchant to expose itself to the market, so user may notice the shop and the brand. This provide a chance for mini market to be known lately.

1.7.5.6 Notification (Mobile Application)

User can receive notification while merchant publish a promotion or an event. When user is interested on the promotion or the event, user can go to the shop made an order. User can easily receive feedback respond when the notification will always notify user immediate when merchant user has replied to the feedback. This helps merchant to do marketing on its own strategy.

1.7.5.7 Feedback (Mobile Application)

User can send feedback for complain with images and text. The feedback will send to merchant side. User can view feedback history and delete the feedback anytime.

Merchant-side web application

1.7.5.8 Manage supermarket products with CRUD operation (Web Application)

Supermarket checkout merchant is allowed to upload supermarket products with price, product's details, and product images. Each of products will need to input a product SKU which represent a barcode so that when customer scan the product barcode, it will identify which product be scanned and display the information. The product information will then display on customer-side application. The supermarket checkout merchant application is also allowed admin to create category for each product. Through this function, admin can perform search function with categories for all uploaded products. After creating a product, merchant is also allowed to update product details or even to delete the products.

1.7.5.9 Listing and delisting product (Web Application)

Except a CRUD operation in merchant-side web application, it also provides listing and delisting function. Supermarket checkout merchant can make the product in live

mode and to make the product in non-live mode. The reason of this functions is allowed merchant to delist the product without deleting it, the product is still can be modified without any effect of all information but in customer-side, the product is not shown.

1.7.5.10 Advanced Search Function (Web Application)

Supermarket checkout merchant can use advanced search function to search uploaded products. Merchant will easily search all uploaded products with category or even one alphabet character to search any uploaded products.

1.7.5.11 Merchant Profile (Web Application)

Merchant will need to contact admin to creating an account with subscription fees. This will store merchant information example like email, name, SSM, PIC name, PIC contact, subscription package, etc. Merchant will also need to get its own location by using geolocation, it let user to detect nearby merchant.

1.7.5.12 Manage Customer Feedback (Web Application)

Merchant can receive customer's feedback and reply to the feedback through the application. Customer-side will receive the merchant reply through the checkout mobileapp.

Administrator-side web application

1.7.5.13 Manage Merchant (Web Application)

Admin can create & edit merchant account so that merchant can access the web app by using the account. Admin having the authority to direct ban the merchant account.

1.7.5.14 Manage News (Web Application)

Admin can create, edit, delete customer news through the web app. Customer-side will update news on home page at mobile application and is able to read the news description.

1.7.6 Module Not Covered

There is a module cannot be complete due to business model and time constraints. Stock control which known as the warehouse management system, this system

should be able to trace the quantity of the item by the orders. However, there is a scenario that can cause the stock control not accurate. For the scenario, when the consumer is paid by cash so the order is never created on our system, it is not logical if we asked the merchant to key in the order and the items inside the order, by this means the stock will be not accurate while this scenario happen. This may cause user to saw this product available while they are comparing prices. But when the time they reach the merchant places and found out the item out of stocks. This will affect merchant's repudiations also the same with the system repudiations. To resolve this problem, merchant should implement full set of the systems which include the payment machine set, this will be connected to the same database and so that each record can be stock accurately and performing a perfect stock control. But the full system will need a big amount of development time, so this module will not be covered in this project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

A supermarket checkout system is a self-service technology for consumer to save time for queue up in traditional way and if with a less item to purchase. Apart from this, with the pandemic in the past few years, a global slogan “social distancing” spread all over in marketplace. A self-checkout system will be more suitable and benefit into nowadays’ situation since it resulted more hygiene and more efficient. However, there are a lot of bad feedback on the system which the system possible brings slower process than a traditional payment cashier due to external technical issues such as bar code not scanning, network failure cause slow payment processing etc. In this chapter, multiple existing self-checkout system has been discussed and reviewed to obtain inspirations and ideas. Besides, an evaluation and comparison between various software development methodology had been discuss in below and the final decision of methodology is iterative model. There are more system architecture and framework has been discussed and compared such as backend, frontend, cloud services provider.

2.2 Review on Similar Self-checkout system

Several evaluations and reviewed will be conducted on similar self-checkout system accessible through online to develop a supermarket self-checkout mobile application that meets and satisfy the system criteria and features. There are three titles been discussed in this section.

2.2.1 Local-based System

Most of the self-checkout system with physical hardware it takes a huge amount of development fund to complete and yet to make it secure. By the generation now, most of the young generation prefer checkout by themselves rather than wait the slow queue on the counter. Hence the physical hardware must be very powerful to make everything smooth and functions, this will rise a problem which to too hard to use, and some of the instruction is not clear on the checkout flow.

2.2.2 Lotus's checkout system

Lotus's Malaysia provides Self-Checkout machine which allow consumer conduct items checkout at self-checkout counter. Inside the Self-Checkout areas, there is only 2 employees to giving control and support for the whole area. This is greatly reduced the cost of Lotus's labour cost, the time for consumer to checkout and the stress of the checkout lane. Although this service seems like no-cons at all, but there is still some scenario that some of the customer may easily shoplifting while in the self-checkout lane.



Figure 2.1: Lotus' self-checkout Counter.

For the user-friendly part, it appears the instruction is not clear for the service interface. Although it supports multiple languages interface, most of the citizen may not really know how to operate the machine for the first time. This may cause the checkout queuing time is longer than the normal checkout counter.



Figure 2.2: Lotus' self-checkout hardware.

By the last field visits, we have observed the citizen which is willing to use self-checkout machine for the 2 hours. Most of the customer appears age between 18 ~ 50. Between these customers, there is only few people with many items to pay in the checkout area. By this situation can observe, most of customer think that self-checkout small number of items is way faster than normal checkout lane.

Through the field visit at Lotus self-checkout technology on 27 February 2022, the feature and flow has been identified:

Features:

- **Scan** – Scan items barcode through scanning device
- **Weight detector** – System will require the scanned items place on the weight detector platform. If doesn't, the system cannot be proceeded.
- **Payment** – The payment method can be E-wallet, card payment by card reader, and cash.
- **Cart** – Cart contain scanned items and able to edit items.

Flow:

- User wants to check out the items.
- User goes to the self-checkout counter.
- User selects the language.
- User starts scan items.
- User uses the barcode scanner device to scan the barcode.
- Monitor shows to put the scanned item to the left weight platform.
- User continue the remain items.
- User selects checkout method.
- User complete payment by the checkout method.
- Machine generates receipt to the user.
- User collects the item from the weight machine and leave.

2.2.3 Amazon's self-checkout store

Amazon Go Grocery is a cashier-less store launch by Amazon on January 22, 2018. It is an e-commerce app that allows consumer to 'Grab n Go'. Basically, it is conceptualized with no queuing line, no checkout with register. It has sensor technology and machine vision to capture human motion to detect whether it is taking off or putting back to the item rack. The system uses machine learning (Artificial Intelligence) to recognition and uses video surveillance technology which capture a human activity by time to time. By the review of 2 youtuber videos, shoplifting is allowed by default in Amazon Go. As the video can be observed, both youtubers have been trying to perform shop lifting in the Amazon Go, but both youtubers failed to steal anything from the store. One of the youtubers made a bold choice, he finished the food after walking out the grocery and made a request for a refund after 3 hours, surprisingly, the requested refund action has been approved. The reviewer stated that Amazon Go Grocery business model is totally based on customer loyalty and behaviour. On the other hand, it provides the environment for shoplifters.

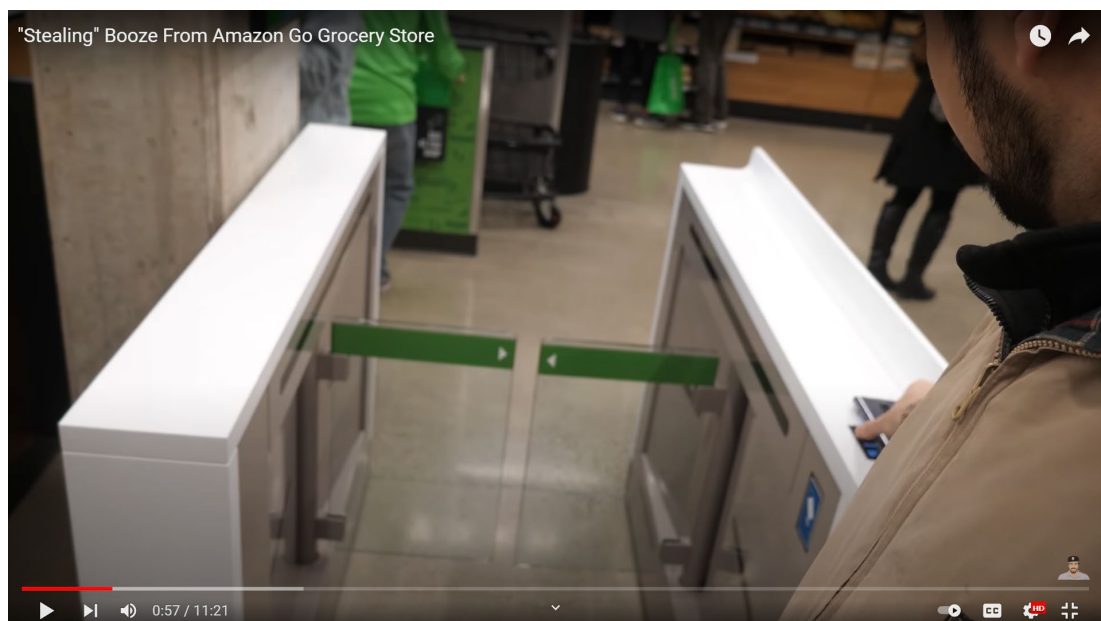


Figure 2.3: Scan with user QR code to sign into the store.

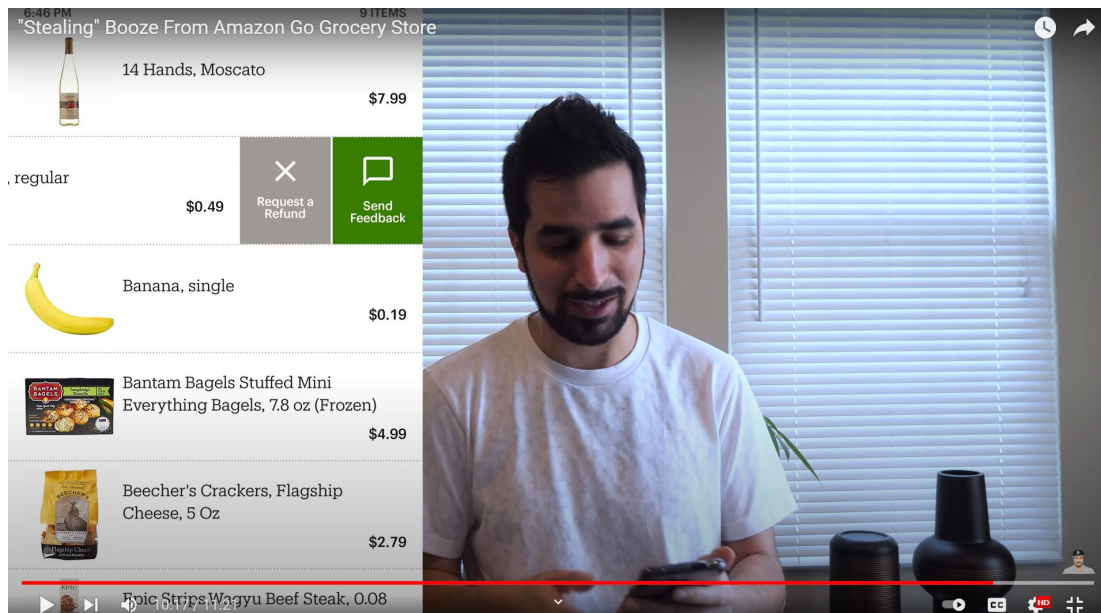


Figure 2.4: Make request for refund for eaten food.

After reviewed on both Amazon go grocery review video, the features and flow has been identified at following:

Features:

- Scan to login
- Human recognition
- Motion Sensor
- Purchase history

Flow:

- User open his/her user QR code and place on scanner to identify his/her identity.
- User grab items.
- User walks out the store.
- Order completes within an hour.
- User can make refund when wrong item added in payment lists.

2.3 Review on Project Methodology

Choose a suitable project methodology is a critical for every success software project. A software project can be small, medium, or large. A software projects can be expanded to large from a small project. In this case, to maintain or building for project incrementally in a success way, a right methodology is needed. SDLC (Software Development Life Cycle) models is the methodology that called as techniques for design, struct, or to maintain for a software project (Alshamrani and Bahattab 2015). SDLC models includes waterfall model, Scrum, V-Shape Model, Iterative and incremental Models and Spiral Models etc. SDLC has 7 process flow which is planning stage, analysis stage, design stage, development stage, testing stage, integration stage and maintenance stage.

2.3.1 Waterfall

Waterfall methodology is a well-known development methodology. It is the first methodology which invented by Dr Winston W. Royce in 1970 (Hughey, 2009). This development methodology is goes by sequential flow, it usually used in large project, and government project. In waterfall models, all steps are required goes by sequentially and without backward to previous stage if there are any mistake encounter. This usually costly and time consuming as it required quality control for every intensive documentation and planning at the beginning of stage which to ensure the software project goes according to plan and requirement.

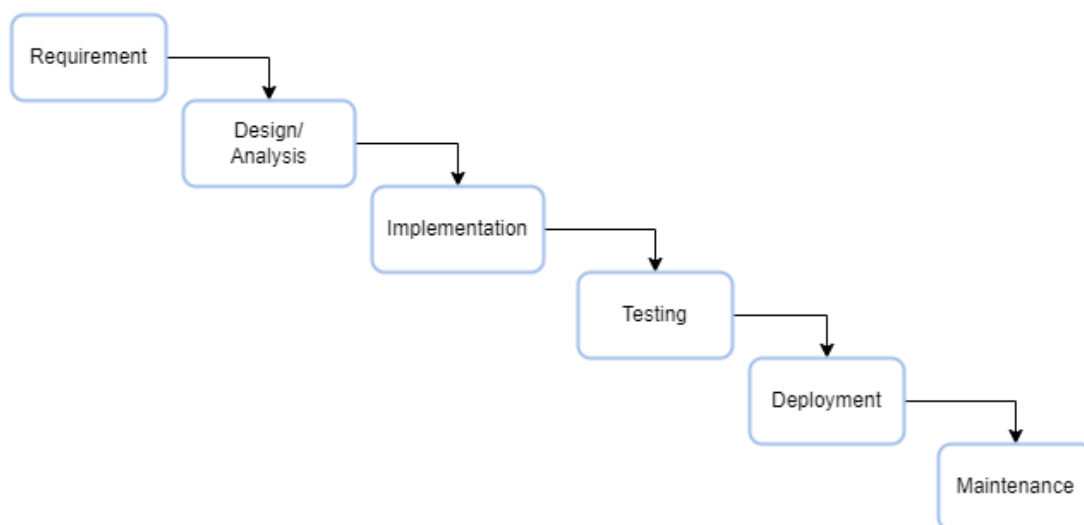


Figure 2.5: Waterfall Model.

2.3.2 Spiral

A spiral model is not the first invented SDLC models. It has a combination of iterative development and small element of waterfall development for a software project which focus on risk assessment and risk management (Alshararani and Bahattab, 2015). This model can be used whenever meet an unclear requirement from client as all projects can be breaking into a small element of easier for change process. The spiral models begin with centre point then it goes traversals with clockwise (Rastogi, 2016). Despite that each point is unclear defined deliverable, the first traversals might be a requirement phase. Spiral model is also defined as meta-model as mentioned above it combine both waterfall model and iterative model (GeeksforGeeks, 2022). In spiral model, it uses prototyping model to deal when risk occurs. Each risk management will rely on prototype at each stage with the defined scale in the spiral model of SLDC.

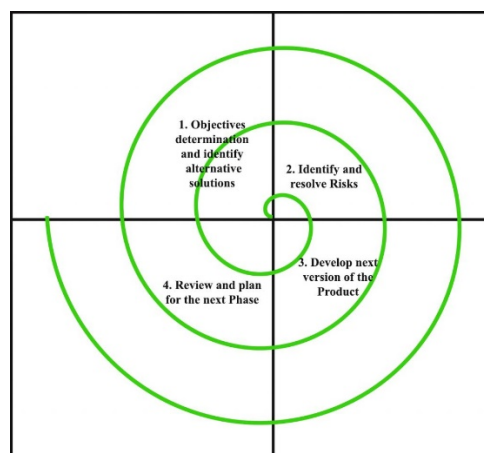


Figure 2.6: Spiral Model.

2.3.3 Iterative and incremental model

The iterative model is using several iteration or version to compose the SDLC. For the initial level, a clearly defined requirement is required which to perform the stage using iterative model (Kananke, 2020). An iterative model has a waterfall model element which it contains a sequential procedure and allow further incremental. For the very first iteration, it usually conducting the basic modules or functionality. In this condition, iterative model is allowed to add more and more function which comes to iteration 2, iteration 3 etc. Each of the previous version will contain a new function for a new release. After each iteration is deployed, feedback is considered

vital information to proceed a next iteration (Rastogi, 2015). This model is turning all project into a small element and process accordingly, hence, this can be applied when software developer team is learning the new technology during taking part in the project.

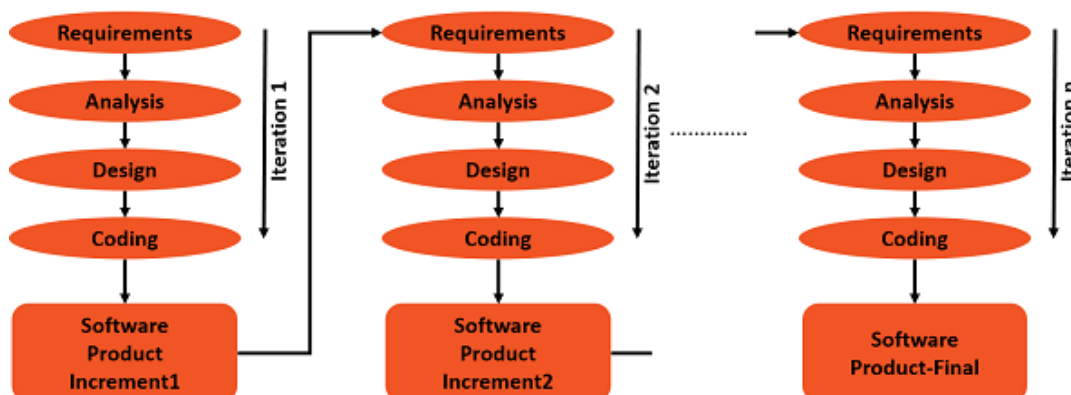


Figure 2.7: Iterative and incremental model.

2.3.4 Comparison of Methodology

To ensure the quality control of software development, it always need a formulation process model to control the process of every project stage. A suitable methodology with well-defined process will result a software project resulted to success. Every methodology model has its pros and cons. The model feature some is time consuming, or some is high cost to apply in the pathway of using the model. The study has done by Chandra (2015) perform comparison for different SLDC methodologies. Based on the study of Chandra (2015), in this project, present the parameter with various of methodologies. The table below shows the comparison of different software methodologies models which considered in this project:

Table 2.1: Table of comparison between various software methodologies

Model Feature	Waterfall	Spiral Model	Iterative Model
Clear Requirement and Specification	Initial Level	Low	Intermediate
Client feedback	No	Yes	Yes
Time needed	Low	High	Low
Risk Factor	High	No	High

Cost	Low	High	Low
Usability	Basic	Less	Normal
Resource needed	Less	Many	Less
Flexibility	No	Less	No

According to the data from the table above, Iterative methodology is defined to be more suitable for supermarket checkout mobile application. There are reasons to choose this methodology in this project, which only prioritized requirement needed. Besides, Iterative model may turn into incremental model if add more functionality that required by user lately. In this case, this model is easier to implement for required change process and always need feedback for user. This will lead to user satisfactory since each deploy will involve user activity and user feedback so that each version will involve more interaction between user and technical team. Due to its low cost and less time consuming if the project is not considered large, the iterative model is suitable for the project. As a result, after considering the model feature, iterative model is chosen to be the software methodologies in this supermarket checkout project.

2.4 Review on Backend Server Framework

Backend server is a must to all the applications and software, it is the connection of worldwide developers. Before building a backend server, framework is the key to make backend server more functions and flexible. Server is mainly contained the APIs (application programming interface) which to make connection between Database, Web Application, Mobile Application. By choosing the right framework to develop the backend server, it should be able to handle a thousand request in a second. Framework choosing can be flexible, it based on the needs of the projects and developers.

Framework can be differentiated with the programming language, and the features/plugins they provide. Different language provides different handle HTTP request in a mean time, some of it provide more features and plugins but with slower handling.

2.4.1 Laravel

Laravel is one of the best PHP Framework that developed smooth and stable web application which can be a difficult project. Advantage of Laravel as it can perform simplification to the complicated tasks. Laravel has the latest features of PHP that reduce the limitations of the web application. It has the great documentation makes Laravel developer-friendly, all Laravel update comes with standard documentations, you can explore the world of Laravel by yourself with all the detail explanations. Besides, it also has a large community. One of the Laravel community name LARACASTS, on the community there is a lot of expert developer always sending tutorial videos for various functions.

The following is features of Laravel framework:

- Easy to do authentication
- ORM in Laravel easy to do query with PHP only.
- Library supports Bcrypt hashing, CSRF protection.
- Provide unit testing.
- It a MVC Architecture.



Figure 2.8: Laravel framework.

2.4.2 Node.js

Node.js is a JavaScript based framework. JavaScript known as server-side only concept before the introduction of Node.js. Node.js expand JavaScript concept to web app development, it is extremely simple, light cost and efficiency. Node.js has its advantages of scalability and multitasking to web application. For example, Node.js can handle a huge amount of request at the same time while it can still proceed all the API request on a significant speed. There is a big community in Github which

providing NPM installer package plugins that makes programming more efficient and light weight. The vibrant community is backed up with Amazon, Google, Facebook, and Netflix. Nodejs also easy to learn as JavaScript is one of the most common programming languages in community.

The following is features of node js:

- NPM (Node package manager) contains a large library in “./node_modules/”. It required to call the package with command line for package installation.
- It can combine with Angular js to create a full stack of typescript application.
- High speed performs which can run several requests in the same time.



Figure 2.9: Node js.

2.4.3 Comparison of Backend Framework

Table 2.2: The comparison between Laravel and Node js backend framework

Parameters	Laravel	Node js
Popularity	Yes	Yes
Performance	Average	Excellent
Development Tools and Package Management	Average	Excellent
ORM	Eloquent	Sequelize
Framework Architecture	MVC architecture - PHP	Open-Source platform-V8 engine
Community	Average	Excellent

According to the above table matrix shown, Laravel and Nodejs has their own characteristics which some is good, and some is consider average. Nodejs has the

high-speed performance and for a checkout app, it requires a fast performance platform and smooth to provide a flexibility and efficiency when perform task. Despite that Laravel has a strong database management backup itself but it is strong and capable for web application only. Nodejs also has an open-source platform which allow to combine other framework to create a full stack application no matter in web app or mobile app. In this case, Nodejs is more suitable in this project.

2.5 Review on Frond-end Web Application Framework

2.5.1 Angular.js

Angular.js is a JavaScript based open-source framework initially introduced by Google Corporation in 2012, it is using MVC (Model-View-Controller) concept. Angular allows user to create Directives which use to create custom tags in HTML (sitepoint, 2018). Compared to original HTML5, Angular.js have the advantage of send asynchronous HTTP request by using unique component FormControl. It is very developer friendly and easy to learn comparing to others front-end framework. For example, it has the unique Ng directive which can interact with HTML and TS, it is way more convenient than the normal Script Writing way (sitepoint, 2018).

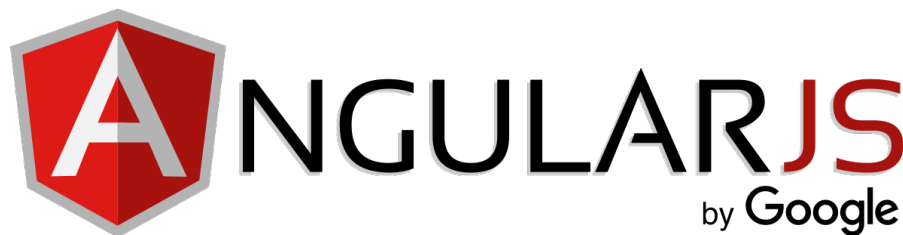


Figure 2.10: Angular js.

2.5.2 Vue.js

Vue was invented by Evan You which is an ex-Google employee. It invented in 2014 and it is also called as a progressive framework (SYSTANGO, 2018). Vue itself is fast, lightweight, and simple to work with, it is a framework for who want to work for building user interfaces. Vue has a similarity with Angular in syntax, but the use case is totally different. Vue is using MVVM (Model-View-ViewModel) architecture while Angular is using MVC(Model-View-Controller) as its architecture. ViewModel is consider the middleware which a connection between view and model (ButterCMS, 2019). It is more efficient if work for single page web app since the core of its libraries is focus on the view case (HBuilder, n.d).



Figure 2.11: Vue.js.

2.5.3 Comparison of Front-end Web Application Framework

Table 2.3: The comparison between Angular and Vue.js frontend framework

Parameters	Angular	Vue.js
Community	Large	Small
Architecture	MVC	MVVM
Code Scalability	Yes	No
Performance	High	High
Built-in libraries	Many	Less

According to the table above, few parameters and feature between Angular js and Vue.js is in a tie condition. Despite that Angular is restricted to use TypeScript and OOPS, but compared to Vue.js, Angular has more built-in libraries. The results shown that Angular has large community worldwide while Vue.js has a smaller community worldwide that shows reliability range is small. The weakness of Angular is it has a regular DOM while compared to Vue.js it has a virtual DOM which faster than Angular. However, the code scalability of Angular is better since it is a mature and it support for a large project. In the nutshell, Angular will be chosen as the front-end development framework for the Merchant and admin web app in this project.

2.6 Review on Cross-platform Mobile Application Framework

Cross-platform framework can call as a hybrid framework. Hybrid platform performance might slow a little bit compared to native framework which is a single-target platform framework. Hybrid framework can build web application, android application, iOS application and computer software with a same source code and dependency.

Ionic and React Native typically similar. Both is the common and popular JavaScript Based framework in Global Developer. The different between 2 framework is Ionic have way more programming language choices to develop an application, for example like Angular.js, React, Vue.js. But React Native have only one choice which is react. By comparing the library, both framework shas almost the same plugin and features, but the UI component library may be various, both have owned their own style on UI designs.

2.6.1 Ionic Framework

Ionic is a hybrid multiple framework acceptance platform which can be write in Angular.js, Vue.js, React. It is built with Apache Cordova. Cordova is an open-source framework that allows user to build mobile application on various mobile platform, for example like Android, iOS (Dunka, Emmanuel and Oyerinde, 2017). Ionic creator can build a wonderful user interface that combines animations, intuition interaction which can brings a benefit for this framework. The different between react native and ionic is react native is run based on dynamic runtime while ionic is run based on web view wrapper. Ionic uses HTML and SCSS to render the user interface which is benefit for basic learner. As mentioned above, ionic framework can be written in angular, react, and Vue.js, this has become one of the advantages of ionic framework itself.



Figure 2.12: Ionic Framework.

2.6.2 React Native

React Native is a hybrid JavaScript Based platform that make hybrid application rendered like native application for Android and iOS. Developer can create an application to both platform with the same source code. React Native was founded by Facebook in 2015 as an open-source project. After a few years, the world trending mobile apps like Facebook, Instagram, Skype was developed in React Native. React Native has its own JavaScript library which is built on top of React. Although React Native is a JavaScript Based Platform, but it is not easy to learn. On the UI side, the HTML code was needed to write in JavaScript format.

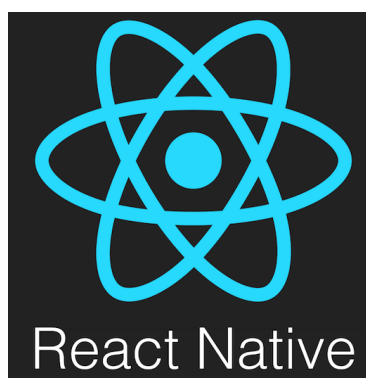


Figure 2.13: React Native.

2.6.3 Comparison of Cross-platform Mobile Application Framework

There is a difference between both hybrid mobile application framework no matter in characteristics or even the features and structural. To have a better choice of hybrid mobile development framework in developing a supermarket self-checkout mobile application, a comparison between Ionic and React native is discussed in below table.

Table 2.4: Table of comparison in Ionic framework and React Native framework.

Characteristics	Ionic	React Native
Learning Curve	Easier to learn	Need to familiar with CSS.
Stack Structure	Hybrid framework	Native framework
Performance benchmarking	Lower than React Native	Perform better
Developer Convenient	Convenient	Need prior knowledge on React js
Application Size	3.2 MB	8.5 MB

Time consuming	Lesser time	Time consumed than ionic
Third party libraries	Easy to integrate with multiple third-party libraries	Need to rely with mind-boggling
Cost of development	Cheaper	More expensive

According to the table above, Ionic is easier to learn and time saver compared to React native. There is the reason of Ionic is easy to install with Cordova and it automatically provide a rich pack of collection of Angular.js extensions and its services. Besides, ionic has online academy lesson which provide rich information and ionic lesson in steps which makes learning and implementation easier. Ionic build is 3.2 MB while React Native is 8.5 MB which ionic having a minimal size application to be created compared to react native. In drawback, Ionic has a slower performance speed when comes to develop a cross-platform app. However, ionic has a cheaper cost when comes to development of mobile application and more convenient for developer since react native will need to be more experience in React js and need to be more familiar with CSS. In nutshell, after considering the features of both frameworks, Ionic will be used to develop the supermarket self-checkout mobile application.

2.7 Review on Cloud Computing Services

Cloud Computing Service is a service that provide extra features for your applications. It includes Database, Website Hosting, Deep Linking, Mobile notification, etc... Comparing to the old method for creating everything by the developers, it appears convenient ways to manage external service to your applications. All the costs for the services are pay-as-you-go, it means the cost is totally based on the user usage. There are only a few providers which for the cloud service.

2.7.1 Firebase (Google cloud platforms)

Firebase was founded by James Templin and Andrew in 2011, after 2014 it officially acquired by Google. Firebase become a unified platform for mobile application in 2016. Firebase has the service of Real-Time Database, Google Ads, Catalytic, Website Hosting, Function Hosting, Cloud Messaging which is totally all suitable for

mobile applications (javaTpoint, n.d). Firebase is developer-friendly and easy to implement into your application.



Figure 2.14: Firebase.

2.7.2 AWS EC2

AWS (Amazon web service) introduced in 2006, it begins with offering IT infrastructure services. It provides highly reliable, scalability and low-cost service. For example, services that require an IT infrastructure like Website Hosting, Server Hosting, Database Storage, Cloud Messaging etc. It provides virtual servers, developer may create APIs with any server language and make it host to public (Amazon, 2022).



Figure 2.15: Amazon EC2.

2.7.3 Comparison of Cloud Computing Services

Cloud computing services has become indispensable for every type of business. In the worldwide, Cloud services is used to store data and is utilized 85 percent of businesses. Cloud computing services includes documentation form editing sharing, calendars sharing, and messaging service are all used in most of business model. In the development of software application, each of cloud computing services has its suitability for each software that wish to be developed. A comparison between AWS EC2 and firebase be discussed below.

Table 2.5: Table of comparison between AWS EC2 and firebase.

Parameters	AWS EC2	Firebase
Cost	Less expensive	More expensive
Strength	Application is flexible	Quick, easy to setup and integration
Core features	Virtual Machine	Realtime Database
API	REST & GraphQL	REST
Documentation	Well-written	Well-written

According to the table above, AWS EC2 is more flexible and experience that brings solution and has more complete services to business organization compared to Firebase service provider. Besides, compare to the origin price, AWS EC2 reduces the price point up to 80 percent (SFAppWorks, 2021). However, both are using REST for API but AWS EC2 has more option which also uses GraphQL. Anyway, firebase also have its benefit which have a quick setup, easy to integrate and provides wide range services and features (GLOWID, 2021). There is no final best option for projects when it comes to cloud computing service. In this case, both cloud computing services can be utilized with their services and features that might fit and suitable for the project. After the comprehensive comparison of both cloud computing service, AWS EC2 will be used to store all the data information and firebase will be served as get a real-time data service in this project. Firebase also will be served as server hosting in this project.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, the methodology of the project and project activities along a timeline will be discussed. SDLC (software development lifecycle) helps to result a successful software project. In choosing the right software methodology, it needs to identify which methodology are suitable to according to software project with plenty of software development methodology framework that will be implemented by flows with SDLC. Hence, this project will utilize iterative models for the software development.

Besides, there are Gantt chart and work breakdown structure diagram has been described in this chapter which to provide outline and guidance along with the project development. Lastly, it has also mentioned the project development tools involved in this project and briefing with all tools.

3.2 Development Methodology

Methodology that I've selected for this project is the SDLC (Systems development life cycle) with Iterative and Incremental Model.

SDLC is a development process that use to developer and design quality software. Whereas agile methodology aims to develop software that fulfilled or exceeds the client expectations within the estimated times and costs.



Figure 3.1: SDLC.

SDLC development methodology is focus on primary, simple execution, which will rise a profits complication and increase a system quality to the ultimate execution. It has the idea of given a huge development techniques, methodologies and tools, to solve each of software project complication that carry out the best solution till the end of project. These methodologies given different processes that allow software developer to follow accordingly are planning phase, defining phase, design phase, implementing phase, deployment phase and maintenance phase. SLDC usually provide solution to which project is consider large and final objective is carry out a high-quality product to the client.

By applying Iterative Model from SDLC, Iterative Incremental Model have the strengths on fast development for the whole flow of the project. As the customer can get the main function early by comparing to other methodology. This model is main perform module by module, so it won't be stuck in the middle for recap the whole development flow. Iterative model which can also called as Incremental Model, it performs break down all large project task into the smaller module to develop. It creates the less error chaining by breaking all task into smaller module for each process.

3.3 Proposed Workplan

In this section, the proposed workplan for this project be analyzed and designed. This will provide an overview to all task with detailed in this project.

Table 3.1: Project Schedule Summary

Name	Start Date	End Date	Duration
▼ Planning Iteration	Feb 07, 2022	Apr 15, 2022	50 days
▼ Preliminary and Planning Phase	Feb 07, 2022	Feb 25, 2022	15 days
Introduction of chapter	Feb 07, 2022	Feb 07, 2022	1 day
Background of project	Feb 07, 2022	Feb 08, 2022	2 days
Problem Statement	Feb 08, 2022	Feb 10, 2022	3 days
Project Objectives	Feb 10, 2022	Feb 11, 2022	2 days
Project Solution	Feb 11, 2022	Feb 14, 2022	2 days
Project Approach	Feb 15, 2022	Feb 18, 2022	4 days
Scope of project	Feb 18, 2022	Feb 25, 2022	6 days
Preliminary and Planning Phase	Feb 07, 2022	Feb 25, 2022	15 days
▼ Literature Review and Research Phase	Feb 28, 2022	Mar 25, 2022	20 days
Introduction of chapter	Feb 28, 2022	Mar 01, 2022	2 days
Review on similar self-checkout system	Feb 28, 2022	Mar 04, 2022	5 days
Review on project methodology	Mar 04, 2022	Mar 08, 2022	3 days
Review on backend server framework	Mar 08, 2022	Mar 11, 2022	4 days
Review on front-end web application frame...	Mar 11, 2022	Mar 15, 2022	3 days
Review on cross-platform mobile applicatio...	Mar 16, 2022	Mar 21, 2022	4 days
Review on cloud computing services	Mar 21, 2022	Mar 25, 2022	5 days
▼ Methodology and Work Plan Phase	Mar 25, 2022	Apr 04, 2022	7 days
Introduction of chapter	Mar 25, 2022	Mar 25, 2022	1 day
Development methodology	Mar 28, 2022	Apr 01, 2022	5 days
Proposed workplan	Mar 25, 2022	Apr 01, 2022	6 days
Technology and development tools involved	Mar 30, 2022	Apr 04, 2022	4 days
Methodology and Work Plan Phase	Mar 25, 2022	Apr 04, 2022	7 days
▼ Project Specification Phase	Mar 14, 2022	Apr 08, 2022	20 days
Introduction of chapter	Mar 14, 2022	Mar 15, 2022	2 days
Fact finding	Mar 16, 2022	Mar 21, 2022	4 days
Use Case Modelling	Mar 28, 2022	Apr 08, 2022	10 days
Requirement Specification	Mar 21, 2022	Mar 28, 2022	6 days
▼ System Design Phase	Mar 14, 2022	Apr 15, 2022	25 days
Introduction of chapter	Mar 28, 2022	Mar 29, 2022	2 days
Logical Database Diagram	Mar 14, 2022	Apr 15, 2022	25 days
User Interface Design	Mar 28, 2022	Apr 15, 2022	15 days
Completion of Iteration 1	Apr 15, 2022	Apr 15, 2022	1 day

▼ Iteration 2 (Development)	Jun 13, 2022	Jul 15, 2022	25 days
▼ Plan iteration	Jun 14, 2022	Jun 17, 2022	4 days
Analysis	Jun 14, 2022	Jun 17, 2022	4 days
▼ System Design Phase	Jun 13, 2022	Jul 01, 2022	15 days
System Architecture	Jun 13, 2022	Jun 16, 2022	4 days
Software Design Pattern	Jun 14, 2022	Jun 17, 2022	4 days
Database Design	Jun 17, 2022	Jul 01, 2022	11 days
▼ Design Mobile Application	Jun 17, 2022	Jun 22, 2022	4 days
Customer side - Design User Interface	Jun 17, 2022	Jun 21, 2022	3 days
Struct database	Jun 20, 2022	Jun 22, 2022	3 days
Implement Software Component	Jun 22, 2022	Jul 14, 2022	17 days
Debugging & Testing Phase	Jun 30, 2022	Jul 15, 2022	12 days
Release deliverables	Jul 15, 2022	Jul 15, 2022	1 day
▼ Iteration 3 (Development)	Jul 15, 2022	Aug 08, 2022	17 days
▼ Plan Iteration	Jul 15, 2022	Jul 19, 2022	3 days
Analysis	Jul 15, 2022	Jul 19, 2022	3 days
▼ Design Web Application	Jul 20, 2022	Jul 26, 2022	5 days
Merchant Side - Design User Interface	Jul 20, 2022	Jul 25, 2022	4 days
Struct Database	Jul 22, 2022	Jul 26, 2022	3 days
Implement Software Component	Jul 26, 2022	Aug 05, 2022	9 days
Debugging & Testing Phase	Jul 22, 2022	Aug 08, 2022	12 days
▼ Iteration 4 (Development)	Aug 09, 2022	Aug 31, 2022	17 days
▼ Plan Iteration	Aug 09, 2022	Aug 10, 2022	2 days
Analysis	Aug 09, 2022	Aug 10, 2022	2 days
▼ Design Application	Aug 10, 2022	Aug 15, 2022	4 days
Administrative - Design User Interface	Aug 10, 2022	Aug 12, 2022	3 days
Struct Database	Aug 12, 2022	Aug 15, 2022	2 days
Implement Software Component	Aug 15, 2022	Aug 29, 2022	11 days
Debugging & Testing Phase	Aug 17, 2022	Aug 31, 2022	11 days
▼ Deployment Phase	Sep 02, 2022	Sep 02, 2022	1 day
Deploy Android Version	Sep 02, 2022	Sep 02, 2022	1 day

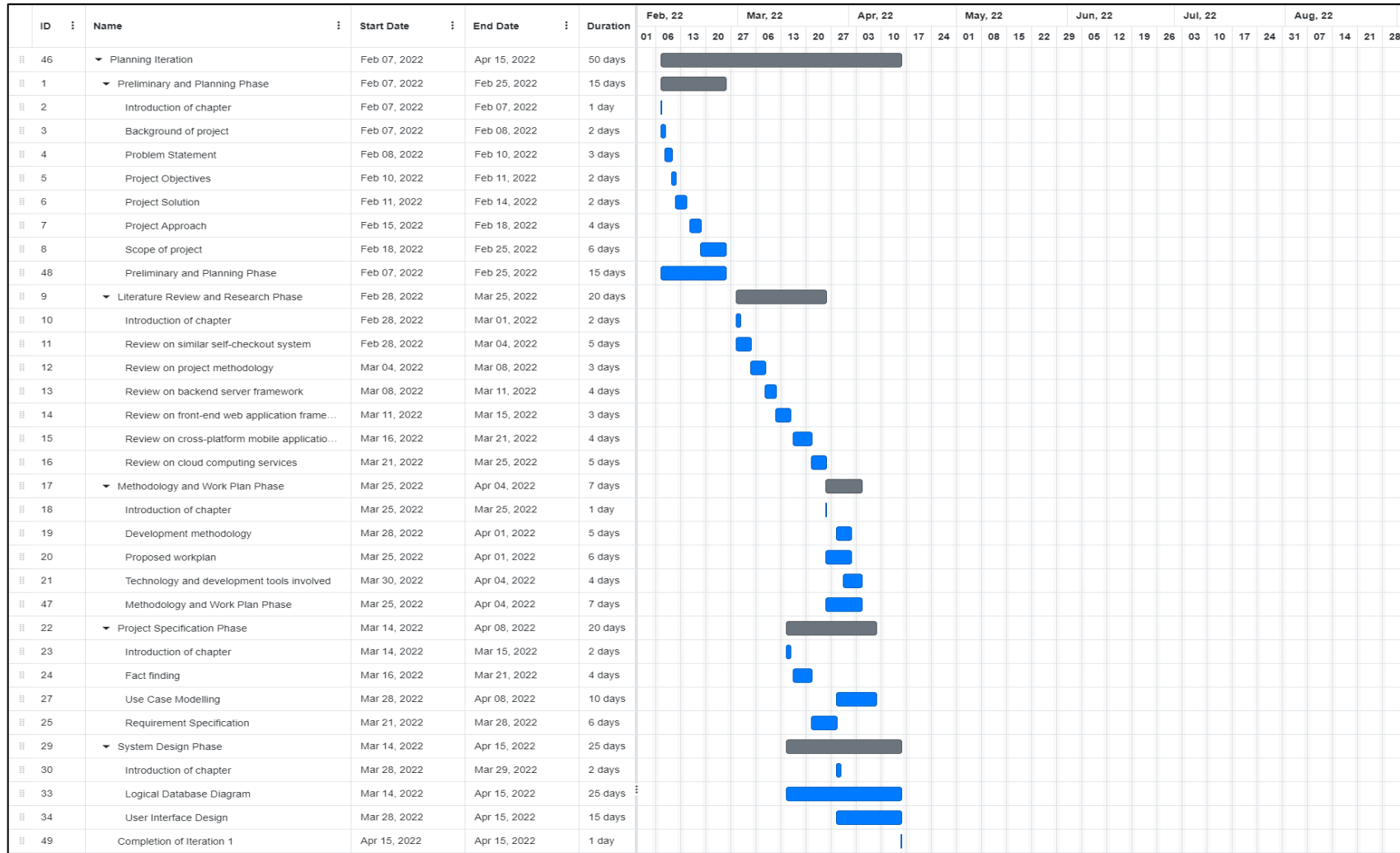


Figure 3.2: Gantt Chart for the project.

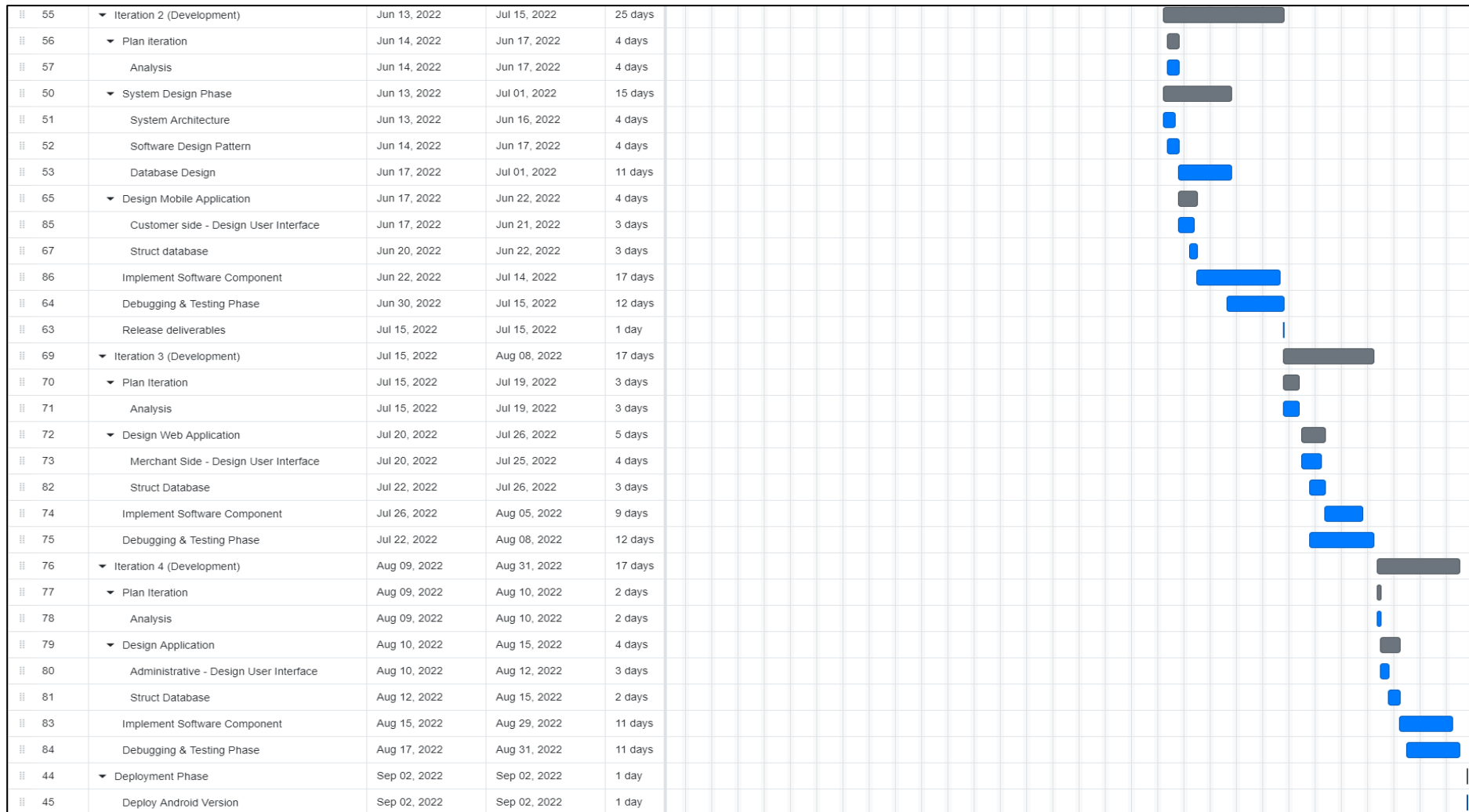


Figure 3.3: Gantt Chart for the project.

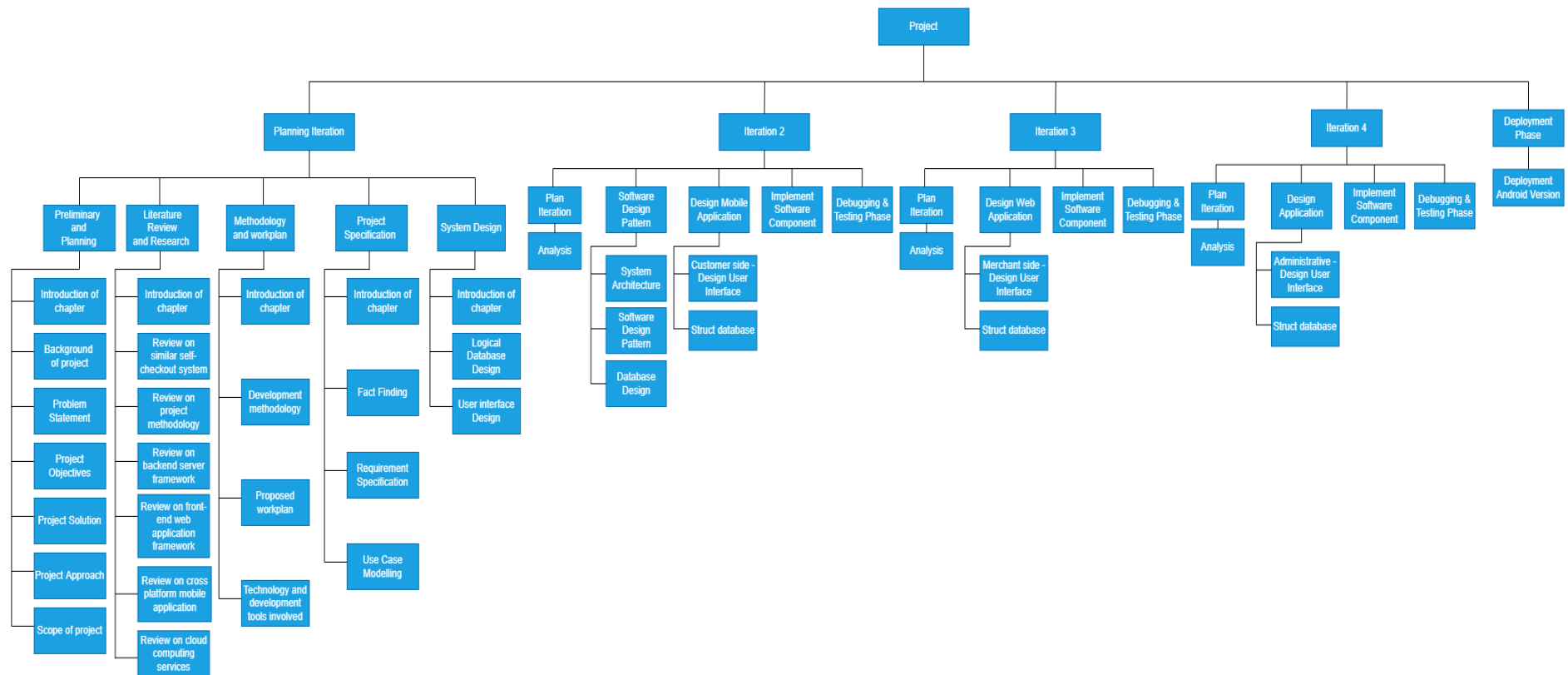


Figure 3.4: Work breakdown diagram of project.

Iterative model will be the development methodology for this project, it follows the 5 process to complete the primary module and repeating the process to make the flow more stable and consistent. Every module can be added into the project if there is more requirement from the client. Each workflow will be explained below:

1. Requirement

In the first phase of the model, the benefit of the iterative model is it can be enhanced the module in next iteration after the main flow was done. Sometimes, client may not clarify their concern or their needs at the initial state. This may constraints in the business model and along with the development flow. Developer will collect all the information from the client side and make a requirement list. Developer can also gather other information like public acceptance for this software by using questionnaires, public interview etc. The person in charge of this role must be agile and good communication skill as it will relate the whole development afterwards.

2. Analysis

After gather all requirements and information from the client and online survey, developer will conduct an analysis regarding how the application will be work as public acceptance such as module, and UI/UX of the application. Developer follows the rules of the model to plan which requirement prioritized in developing the application.

3. Design

After the analysis report was carried out, this phase based on the report which to construct database, the system workflow, and the UI/UX design based on the report suggest. Developer will need to draw several diagrams towards their team in understanding the flow of the application. For example, UML diagram, data flow diagram, use case diagram etc. In this phase, developer should identify all the requirement specification and module planning will be conducted in analysis phase. Developer should understand all the module needs according to the requirements that had been gathered in the early stage.

4. Coding

After all system design diagrams and required module was defined, it proceeds to the implementation phase, which is code with programming language in coupling the whole application by following the information gathered. By this phase, developer team will need to keep updating the progress to project manager. In the end of this phase, to ensure all features has implemented in the application is compulsory.

3.4 Technology and Development Tools Involved

3.4.1 Nodejs

Node.js will take part as a middleware for the project, it will be used to create APIs and make database connection to the application. Node.js has many plugins inside the NPM, it can build a lot of features of APIs with the plugins. Inside node.js will take include Firebase to provide the create account authentication, FCM (Firebase Cloud Message) for the project. For example, it can build an API that use `firebase.auth().createUserWithEmailAndPassword()` to make a UID and put the data into the database.

3.4.2 Angular.js

Angular.js will be the front-end language of the web application and mobile application. Angular.js has the benefit of combination with NPM which is Node.js, with the NPM it reduced a lot of limitations inside the application. Angular.js is easy to code and the source code is totally reusable for every module and pages. Angular.js can cater responsive element easily with the unique `ngStyle`, `ngClass`. The output of the UI may follow the VW (View Width) and VH (View Height) changed by size to size.

3.4.3 Ionic Framework

Ionic will be the main platform that contain Angular.js source code and build android and iOS application with Apache Cordova. Ionic has the benefit of accepting any SDK and any executable source code with a correct plugin. With Ionic application further upgrade or adding in new module will not be a problem unless it matched its limitations. For example, Ionic can accept Angular.js, Vue.js, React source code at a time, but it still can build Mobile Application through Apache Cordova.

3.4.4 PostgreSQL

PostgreSQL will be the database language in the project, PostgreSQL is one of the SQL languages that with a great interface PgAdmin, and it is easy linking to middleware. Compare with other SQL, PostgreSQL is easy to learn, and it has a very clean sight data view.

3.4.5 AWS EC2

AWS EC2 will take part as database server in this project. EC2 is a virtual server that allows developer to install any programming tools. This server will contain Database, Function Server (Middleware), AWS EC2 database cost is a lot cheaper if compared to Firebase Database. It can also mock the URL to the public APIs like api.xxxxx.com by using other domain service.

3.4.6 Firebase

Firebase will take part as Service Provider in this project. Firebase has the service of FCM (Firebase Cloud Messaging) which is sending notification to the targeted mobile applications. it will use for hosting the web application. Firebase has the benefits of many service that can use for a mobile application development. For example, firebase authentication for user signs up and login, firebase deep link which is used to click the URL and links you into the target application.

3.4.7 Visual Studio Code

Visual Studio Code will take part as the main tools of coding in this project. VS Code is the one of the top tools for coding in the world. It has the advantage of extension in the tools. For examples, developer can install snippets for more convenient coding while in the development process. VS code could edit any programming language if it had installed the language CLI.

3.4.8 Git

Git will take part as the project management tools in this project. Git can store source code and merge it by the git editor. Git editor will automatically detect the constraints inside the source code and declare whether it can be merge by automatically or need to manually resolve the constraints. It can be split branch for each user, it can be used to split module by module for each developer who are participate in this project.

3.4.9 Figma

Figma will take part as the interface design tools in this project. It provides tools such as design frame templates with various of device size. It available for phone, tablet, desktop, watch etc. It given a friendly interface for UI/UX designer to assist to

design wireframes, prototype with multiple type of image file extension, as for website wireframes, figma brings a benefit in exporting svg file. Besides, it also provides interaction tools which each can animate or interact between each frame to become a prototype. As for this project, Figma is utilized as UI design role which to draw the prototype to display the plan of UI design for the supermarket self-checkout mobile application.

3.4.10 Trello

Trello served as the collaborative project and work management tools which bring a purpose on tracking team projects or self-reminder project. For team project, Trello allows team lead to assign the team member to their role and show their details of works to achieve completion. Trello is relying to Kanban principles that visualize all tasks in one sight. It visualizes with board, cards or even list. The board created with a title then it indicates list which are the progress details. In this project, it used to create specific tasks to act as backlog to show which module has been done in time and which does not. It is also can reminds developers to follow along with tasks and submit before deadlines which to avoid do things at eleven o'clock. It also can ensure customer satisfaction as product can be delivered on time. It is one of the good practices when come to a real working environment.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

This chapter describes fact-finding which uses online survey questionnaire and observation to identify and comprehend the requirements. It obtains the information and suggestions of supermarket consumers. Supermarket consumers can be anyone, it classified in three group of age which is youth age, middle age and senior age. The questionnaire focused to gather the acceptance of implementation of self-checkout technology/system nowadays. The online survey is applied google form which to collect data from supermarket consumer. There are 30 of respondents filled in the survey and 30 responses has been collected and be evaluated below. There are use case diagrams and use case descriptions conducted in this section. Lastly, screen flow and screen capture of prototype will be displayed.

4.2 Fact-finding

Fact-finding in this chapter will be conducted by data gathering method in questionnaire and observation.

4.2.1 Observation

Date Conducted: 27-2-2022

Time: 2 pm to 4 30 pm

Location: Lotus's Puchong

The Purposes of data gathering is to collect data information ensuring data that be collected has a sufficient information to perform the project and system that will be implemented. In this project, the data collection type is qualitative data collection which include questions that provide to those respondents and to collect each opinion that resulted what is primary and what is secondary features in the proposed system.

During the field visit of lotus in Puchong, through the observation, it found that number of consumers that uses self-checkout machine is consider average. The majority of self-checkout user is youth-aged user and middle-aged user. The self-checkout area is monitored by at least one assistant worker which to assist the

customer when they encounter any problem during self-checkout. During the observation, there are out of 7 middle-aged users will need guide and assist during the self-checkout.

The lotus is considering a large organizational supermarket which currently have 64 stores located in Malaysia. The approximate year of Lotus introduced the self-checkout is in 2015. According to the report, the purpose of introduce the self-checkout system in Lotus, is to propose a solution for customer who complaint and feedback that long queuing time to open cashier.

SST in Lotus having total of 6 of the self-checkout machines. During the first observation in Lotus, 2 out of 6 self-checkout machines cannot be perform due to machine errors and unknown problem which cause time consuming and bring inconvenient than paying at the cashier counter. To maintain and fix the self-checkout machine whenever it broke which is always a high budget in long term. It has been avoided in this project which is the objective and goals that has been stated in early stage.

The purpose observation is to view and having a reference of the process and workflow of existing SST in Malaysia. It also contributes to the problem identified in the proposed system. Threat of theft is one of the problems that been identified in this project. The project solution is focused to eliminate the risk toward the lowest possibility rate. The observation of checkout event in Lotus which has provide inspiration for solution of the problem. There are at least one or two workers will be patrol around at self-checkout machine. One of the purposes is to assist those customers who encounter problem when using the self-checkout machine. The most important is to monitor those who having dishonest behavior during the checkout process.

In this fact-finding, observation has been conducted through second field visit at lotus supermarket on 2 April 2022. The processes/step found in the observation is following:

1. Customer with items queue for the self-checkout machine slot.
2. Customers go to the self-checkout machine slot.
3. Customers click scan item button.
4. Customers start scan their items.

5. Customer put the scanned item to the weight detector,
6. Customers repeat the scanning.
7. Customers click checkout button
8. Customers select payment method for checkout.
9. Customers use the payment method to checkout his items.
10. Customer pack his item from the weight detector.
11. Customer leaves the area.

Above are the second-round observations of SST at Lotus. The workflows and process are being determined through the second field visit. There are some features and service are being referred to the proposed project. The main features are scanning the items barcode, items cart, payment method and customer service. These features are important and will be implemented in proposed project.

The field visit at Lotus of self-checkout related photos is attached in appendix part as “Appendix A”.

4.2.2 Questionnaire

In this section, questionnaire is conducted using google forms which given a list of questions with a purpose of data gathering from supermarket consumer. The total question of the questionnaire is 14 questions.

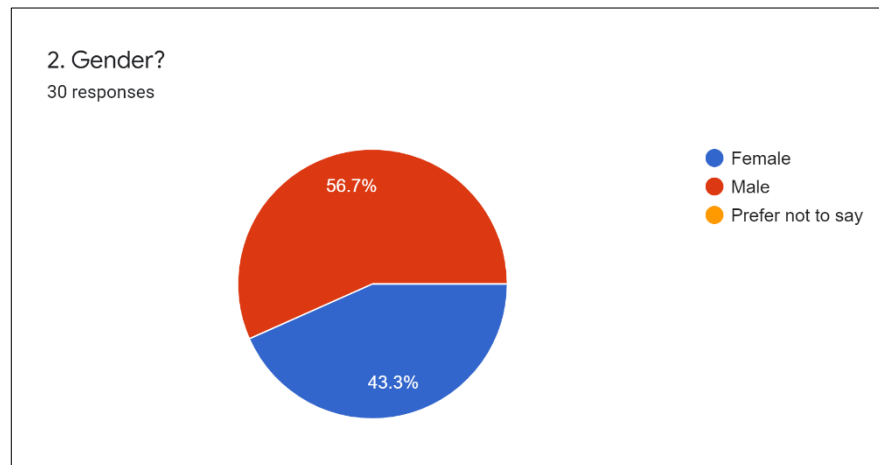


Figure 4.1: Pie chart of Gender of Respondents.

The chart above shows the gender type of respondents which from 30 responses in pie chart visualization.

According to the pie chart shows in figure 4.1, there are 56.7% of respondents is male. Besides, there are 43.3% of respondents is female.

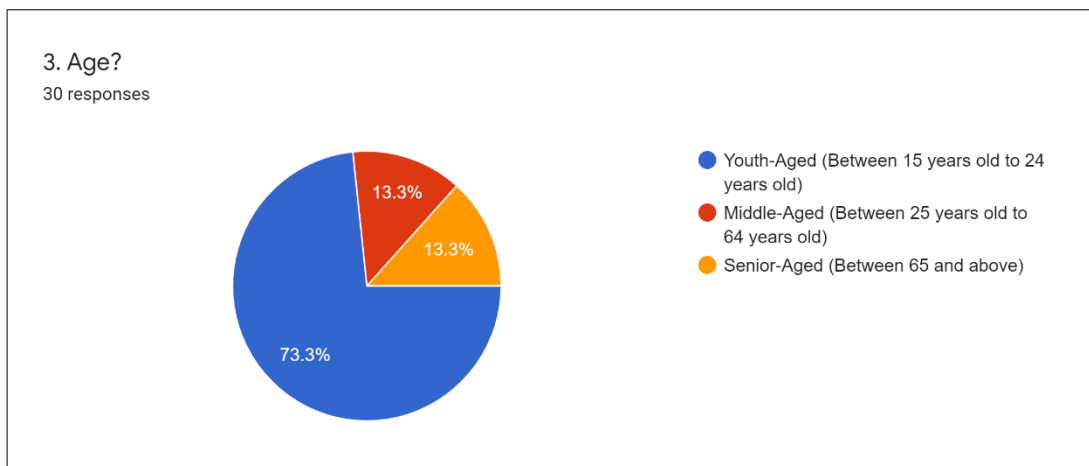


Figure 4.2: Pie chart of age group of Respondents.

The above pie chart displayed the percentage of age group of respondents in visualization. The age group classified in three group which is youth-aged that between 15 years old to 24 years old, middle-aged between 25 years old to 64 years old, and senior-aged is 65 above. According to the pie chart, the majority respondents are from youth group which occupied 73.3%. Middle-aged and senior-aged has equally shown in pie chart.

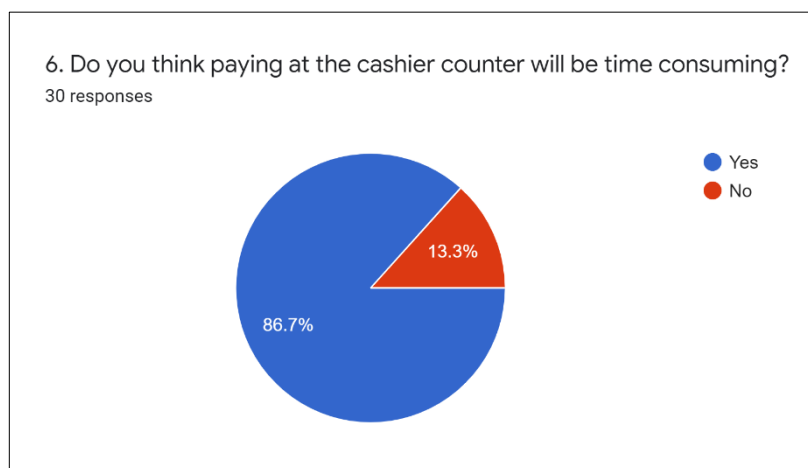


Figure 4.3: Pie chart of paying at cashier counter related question.

The above pie chart visualised the percentage of paying at cashier counter related question. There are 86.7% respondents think that paying at cashier counter will be more time consuming. However, there are respondents (13.3%) has no objection towards time taken during whole process of physical cashier counter from queuing up until paying at cashier counter.

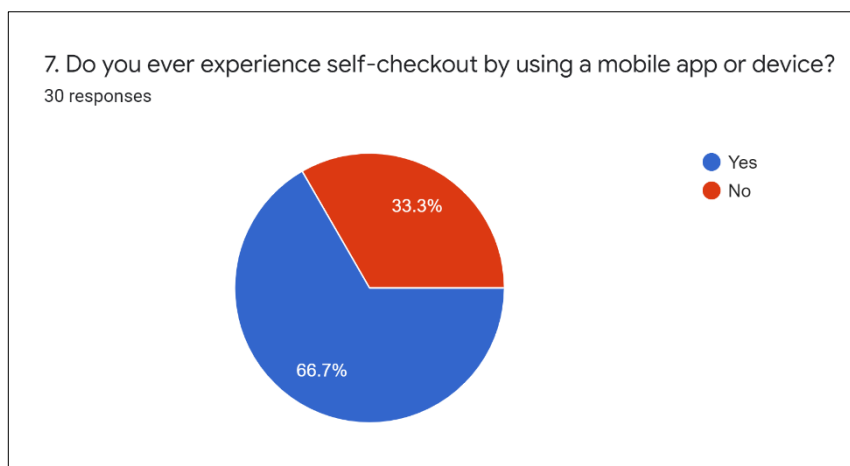


Figure 4.4: Pie chart of experiencing self-checkout related question.

The above pie chart visualized the percentage of respondents who ever experience self-checkout by using a mobile app or device. There are 66.7% of respondents experienced self-checkout system by using a mobile app or device. However, there are 33.3% never experience any self-checkout system by using any mobile app or device.

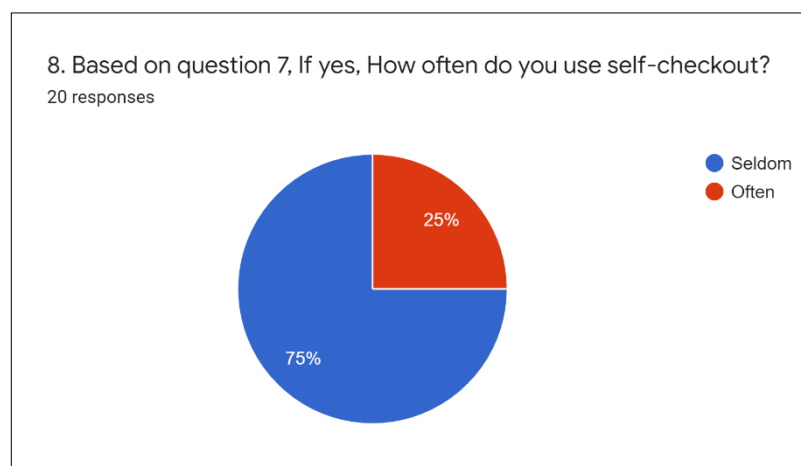


Figure 4.5: Pie chart of experiencing self-checkout related question.

The above pie chart visualized the percentage of respondent's result is based on question 7. According to the pie chart, there are 75% of respondents seldom use the self-checkout system to pay for their item. However, there are 25% of respondents are often use the self-checkout system. It can be concluded that the self-checkout system is still in developing form, it considers less popular in one of the payment methods.

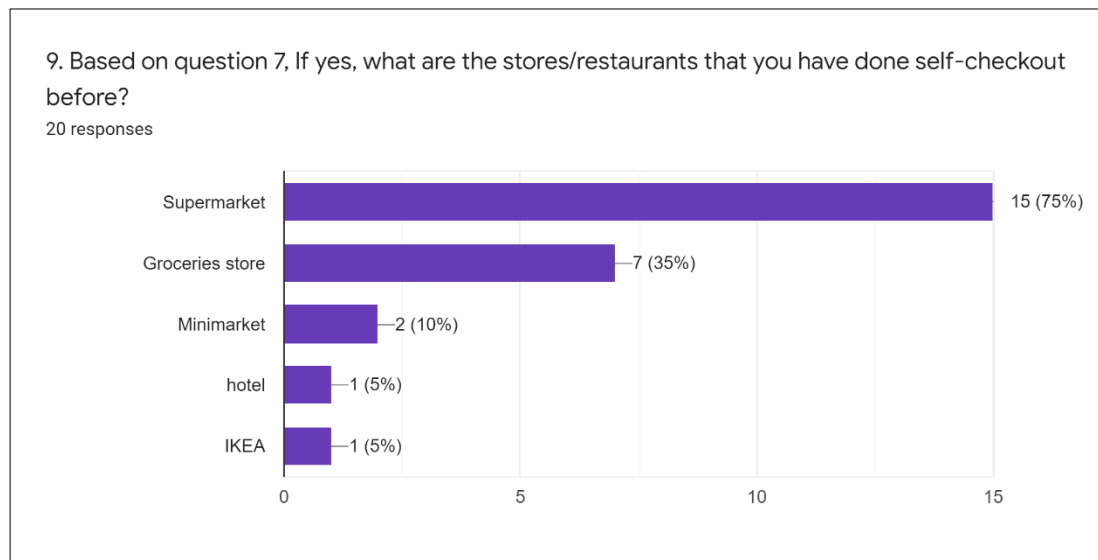


Figure 4.6: Bar chart of place that done self-checkout related question.

The above bar chart visualized the percentage of respondent's result is based on question 7. The above bar chart shown location that respondents has experience self-checkout before. There are 75% of respondents which 15 people out of 30 people has experienced self-checkout at supermarket. There are 35% of respondents which 7 people out of 30 people has experience self-checkout at groceries store. However, the other data shows 10% of respondents experience at minimarket, while there are each 5% experience self-checkout at hotel place and IKEA.

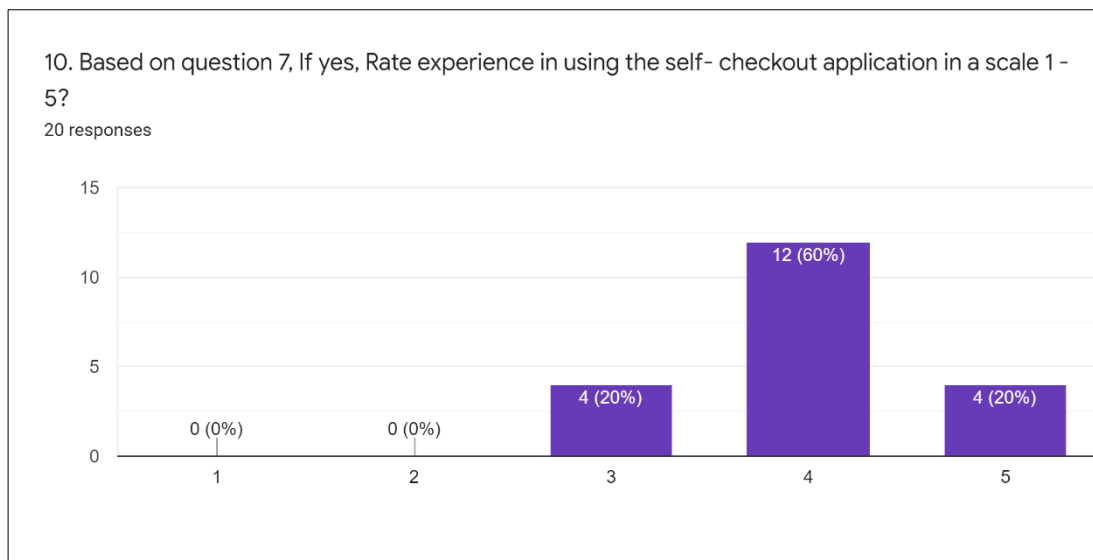


Figure 4.7: Bar chart of rating self-checkout related question.

The above bar chart visualized the percentage of respondent's result is based on question 7. The above bar chart of respondents gives a rating of experience in using the self-checkout application in a scale 1 to 5. The result shows there are majority (80%) of respondents is in a satisfactory range by experiencing self-checkout system. However, there are 20% of respondents are gives a normal rating in experience the self-checkout system. The self-checkout system is still can be improved and become known by everyone in the future.

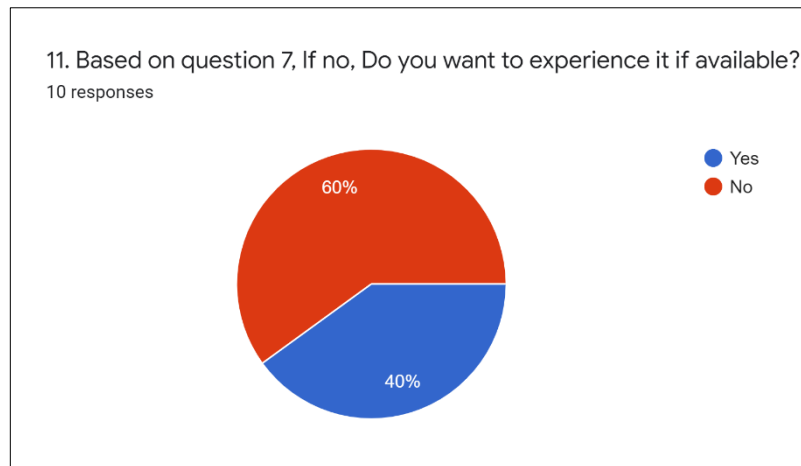


Figure 4.8: Pie chart of self-checkout related question.

The above pie chart visualized the percentage of respondent's result is based on question 7. There are 40% of respondents refused to experience the self-checkout system if available. This may bring a reason that user may not clear and understand what a self-checkout system is. Besides, the other reason may be that they are less confident to use the self-checkout system since they never used it before, and also that they will think that the self-checkout system will consume more time to learn and operate which it possibly takes longer time than queuing up at physical cashier counter. However, there are 60% of respondents are still wanting to experience the self-checkout system if available.

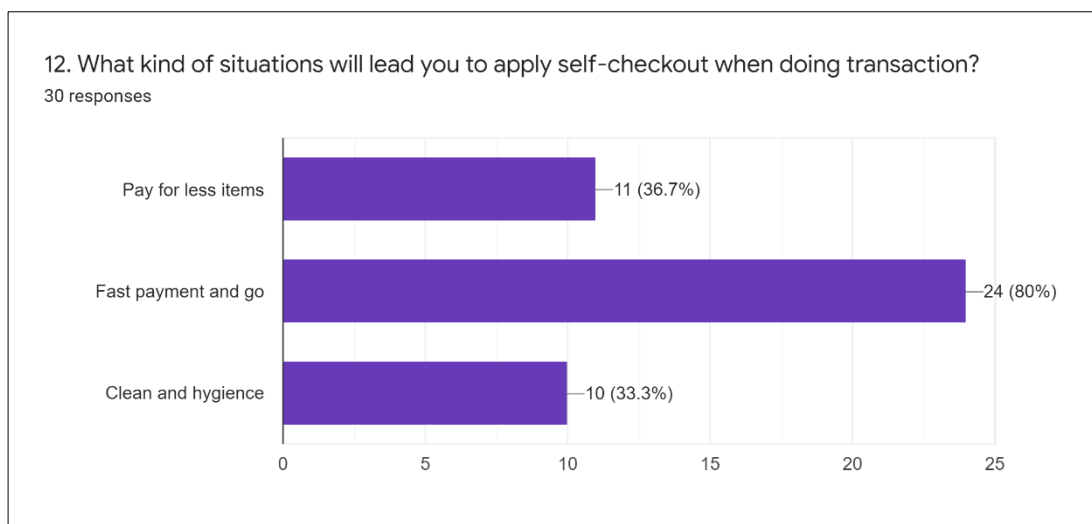


Figure 4.9: Bar chart of self-checkout related question.

The above bar chart visualized the percentage of respondent that situation led them to apply self-checkout when doing transaction. There is majority (80%) of respondents would apply the self-checkout system to do the transaction while it can allow them to pay their item faster than queue up at cashier counter. Besides, there are 36.7% of respondents would apply self-checkout system when pay for less items. Lastly, there are 33.3% of respondents would apply self-checkout system because they think that is cleaner and hygiene compared to physical cashier counter.

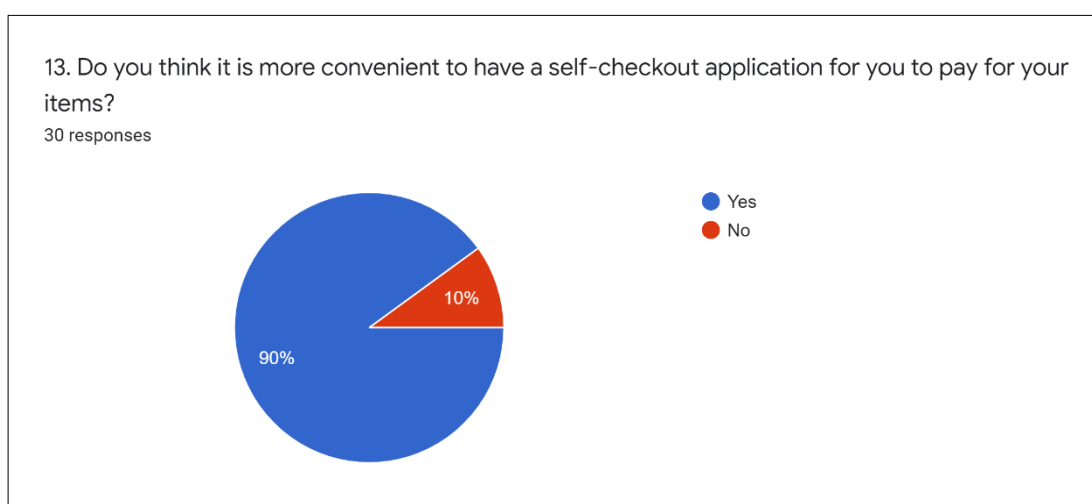


Figure 4.10: Pie chart of self-checkout related question.

The above pie chart visualized the 90% of respondents is agree that self-checkout application is more convenient to pay for their items. Besides, there are 10% of respondents is disagree.

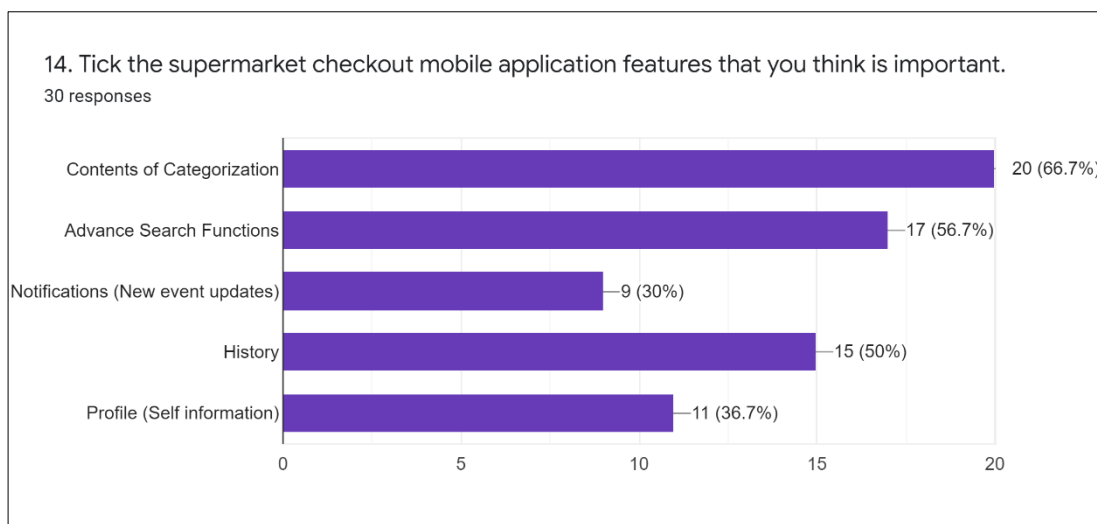


Figure 4.11: Bar chart of features of self-checkout related question.

The above bar chart shows that content of categorization is majority (66.7%) needs for respondents. The second features that might needs from respondents is advance search function which occupied 56.7%. History will be the third (50%) needs from respondents that shows in the bar chart. However, profile is dispensable for respondents (36.7%) which is not the necessary features for the application. Lastly, there are 70% of respondents think that notification might be not necessary to apply in the application.

In the other question, an open-ended question has been conducted with a purpose of collect the suggestion of respondents regarding the other features of self-checkout application that respondents would like to provide. An opinion and suggestion are important from a supermarket user to carry out a user friendly and user experience self-checkout application. Their suggestions are vital which may be possible to generate some inspiration of what is primary and secondary features that user might need throughout the implementation of application. Based on the response, some respondents may think that customer service is needed. Besides, some respondents provide suggestion of poster, news and chat features may need to implement in the self-checkout application.

In nutshell, above questionnaire practice is to determine whether the supermarket self-checkout application will bring a benefit and convenience to supermarket consumer. The practice also identifies the extent of acceptance in

supermarket self-checkout application from the respondents. Besides, the questionnaire practice also helps to analyse primary modules and features that expect to be implemented in the supermarket self-checkout application.

4.3 Requirement Specification

In this section, software requirement specification (SRS) will be conducted. The SRS documentation describes the details of what will be implemented in the application and deliver it for approval.

4.3.1 Customer-side mobile application

This section describes the software requirements for modules of customer-side mobile application.

4.3.1.1 User checkout

1. Supermarket user shall be able to scan product with barcode into cart.
2. Supermarket user shall be able to remove unwanted product from the cart.
3. Supermarket user shall be able to checkout all products in the cart by using online transaction.
4. Supermarket user shall be able to receive the payment e-receipt.

4.3.1.2 User activity history

1. Supermarket user shall be able to view payment history.
2. Supermarket user shall be able to request for refund.
3. Supermarket user shall be able to add previous set of orders into cart from user activity history.

4.3.1.3 Account

1. Supermarket user shall be able to log in to the application with user credential.
2. Supermarket user shall be able to edit their profile information such as name, address, phone number etc.
3. Supermarket user shall be able to log out from the application.
4. Supermarket user shall be able to recover their application password.

4.3.1.4 Notification

1. Supermarket user shall be able to receive notification with new event, poster, promotion, feedback reply etc.
2. Supermarket user shall be able to view details of notification.
3. Supermarket user shall be able to mute the notification.

4.3.1.5 Feedback

1. Supermarket user shall be able to send feedback.
2. Supermarket user shall be able to update feedback.
3. Supermarket user shall be able to receive feedback retrieved from merchant user.
4. Supermarket user shall be able to delete specific feedback.

4.3.2 Merchant-side web application

4.3.2.1 Manage products

1. Merchant user shall be able to add supermarket products into application.
2. Merchant user shall be able to edit supermarket products.
3. Merchant user shall be able to remove supermarket products.
4. Merchant user shall be able to update supermarket products.
5. Merchant user shall be able to live and unlive specific product or category of products with listing and delisting.

4.3.2.2 Advanced search function

1. Merchant user shall be able to search all products in the application.
2. Merchant user shall be able to search with product name or category in the application.
3. Merchant user shall be able to search products with only one single alphabet in the application.
4. Merchant user shall be able to choose specific product category to search the product.

4.3.2.3 Feedback

1. Merchant user shall be able to view all feedbacks from supermarket user-side.
2. Merchant user shall be able to view details of each feedback.
3. Merchant user shall be able to reply feedback to specific user.

4.3.2.4 Notification

1. Merchant user shall be able to view all created notification.

2. Merchant user shall be able to create notification by insert title, content and image.
3. Merchant user shall be able to delete specific notification.

4.3.3 Administrative web application

4.3.3.1 Manage Merchant Account

1. Administrator shall be able to add new merchant user account.
2. Administrator shall be able to update merchant user account information.
3. Administrator shall be able to switch status of specific merchant user account.
4. Administrator shall be able to view list of all merchant user account.

4.3.3.2 Manage Customer News

1. Administrator shall be able to add new news into application.
2. Administrator shall be able to update news details into application.
3. Administrator shall be able to delete news from application.
4. Administrator shall be able to view list of all news from application.

4.3.4 Non-Function Requirements

1. The mobile application shall allow users to access when there is having an internet connection only.
2. The mobile application shall allow android users to access the application only.
3. The mobile application shall be able to access by users at any time.
4. The mobile application shall be available a specific user layout for senior citizen.

4.4 Use Case Modelling

4.4.1 Use Case Diagram

4.4.1.1 User Checkout

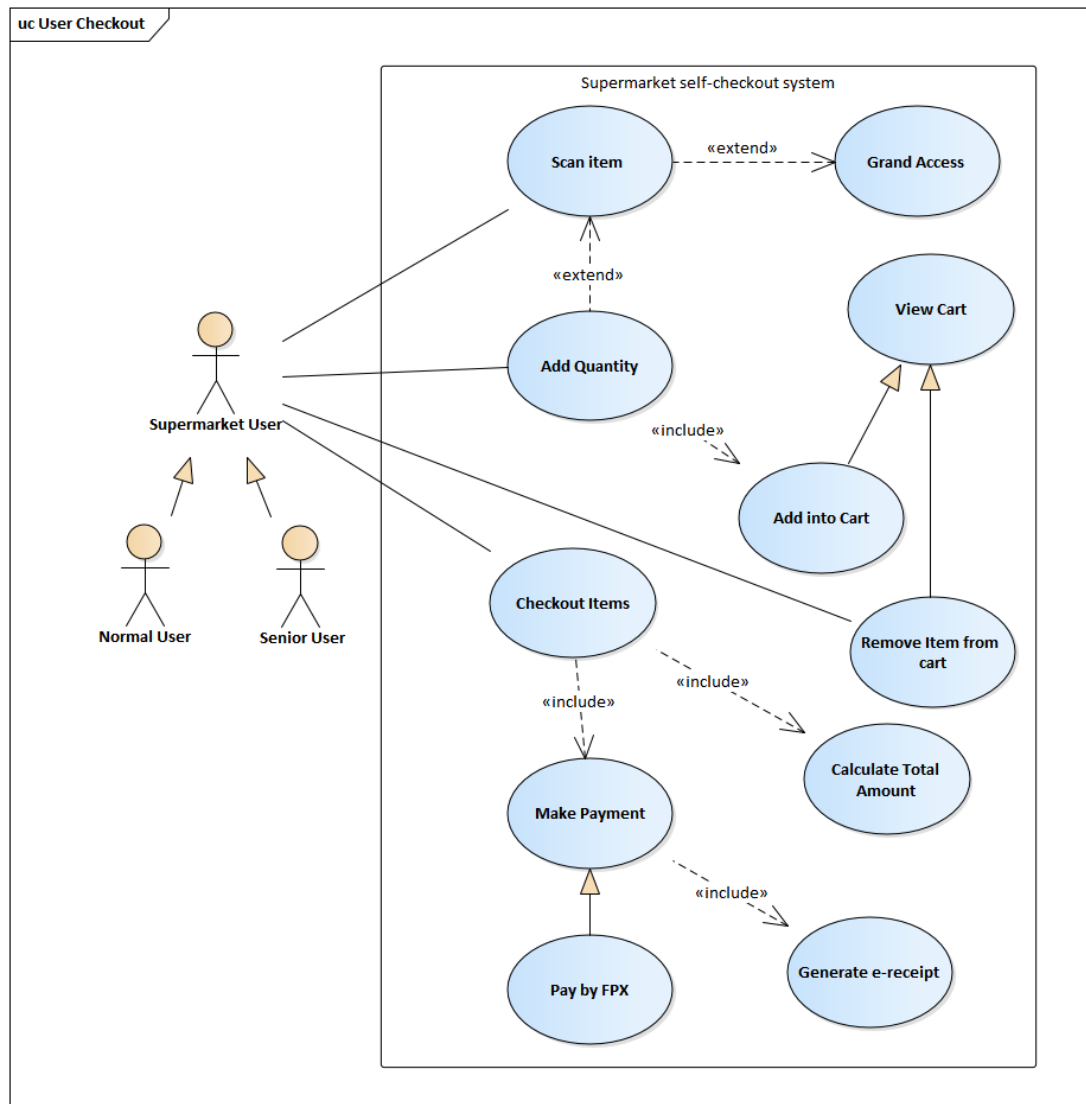


Figure 4.12: User checkout use case diagram

4.4.1.2 User Activity History

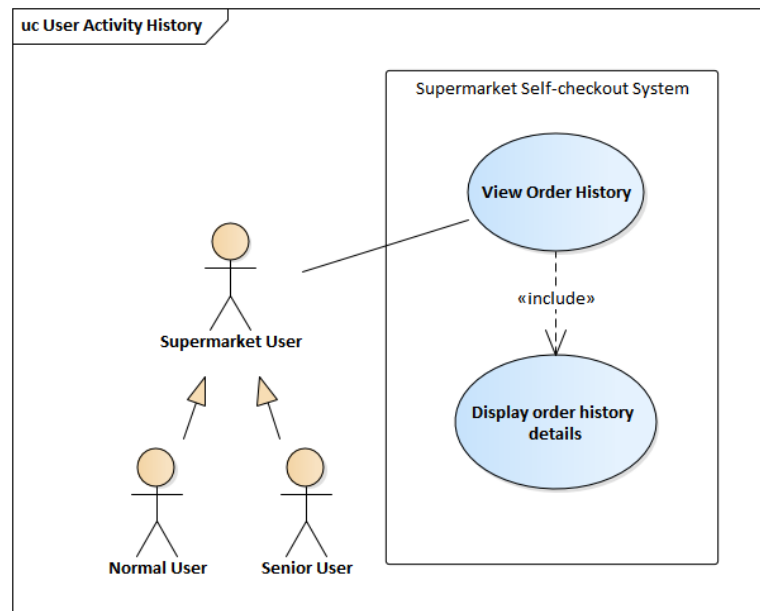


Figure 4.13: User activity history use case diagram

4.4.1.3 User Account Related

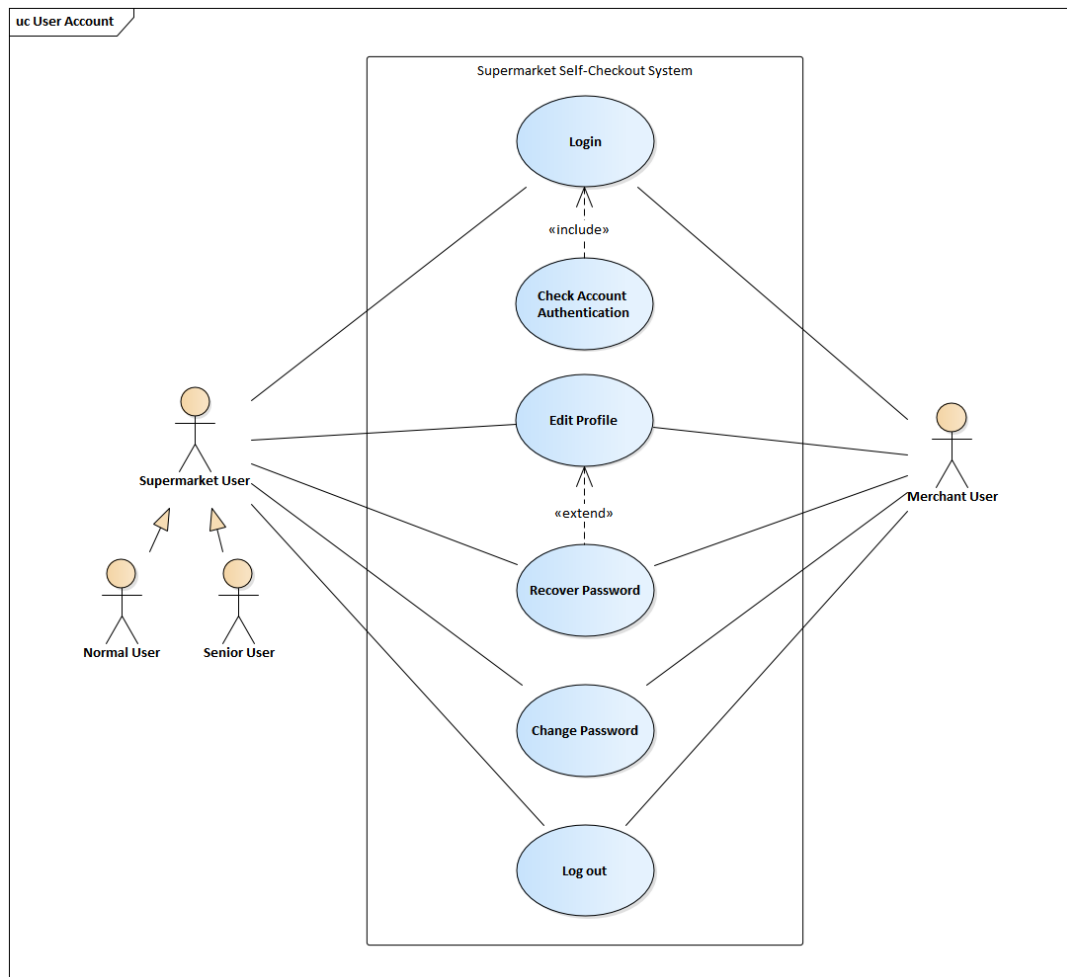


Figure 4.14: User account related use case diagram

4.4.1.4 Notification Related

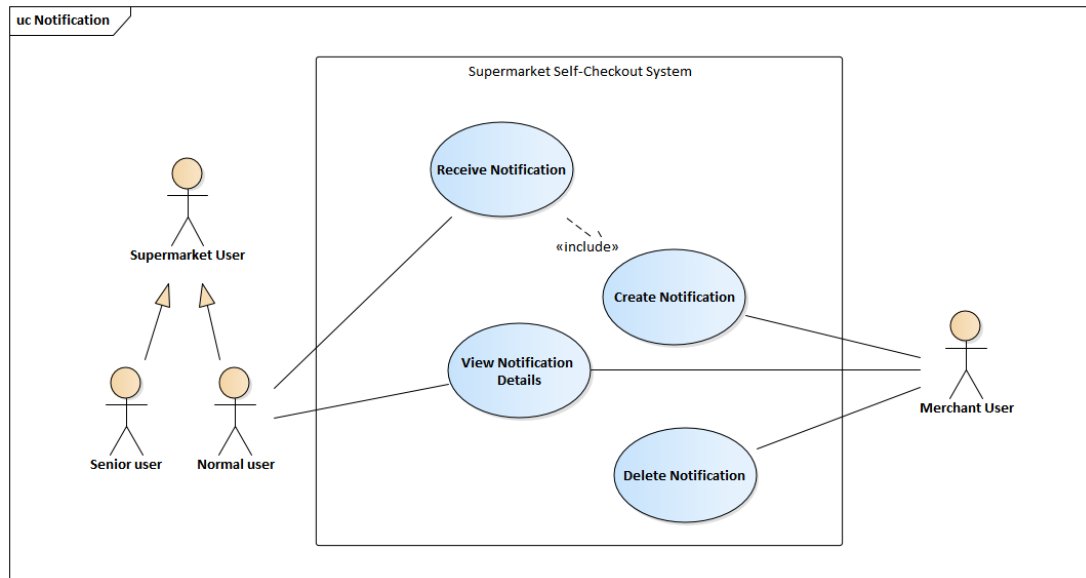


Figure 4.15: Notification related use case diagram

4.4.1.5 Feedback Related

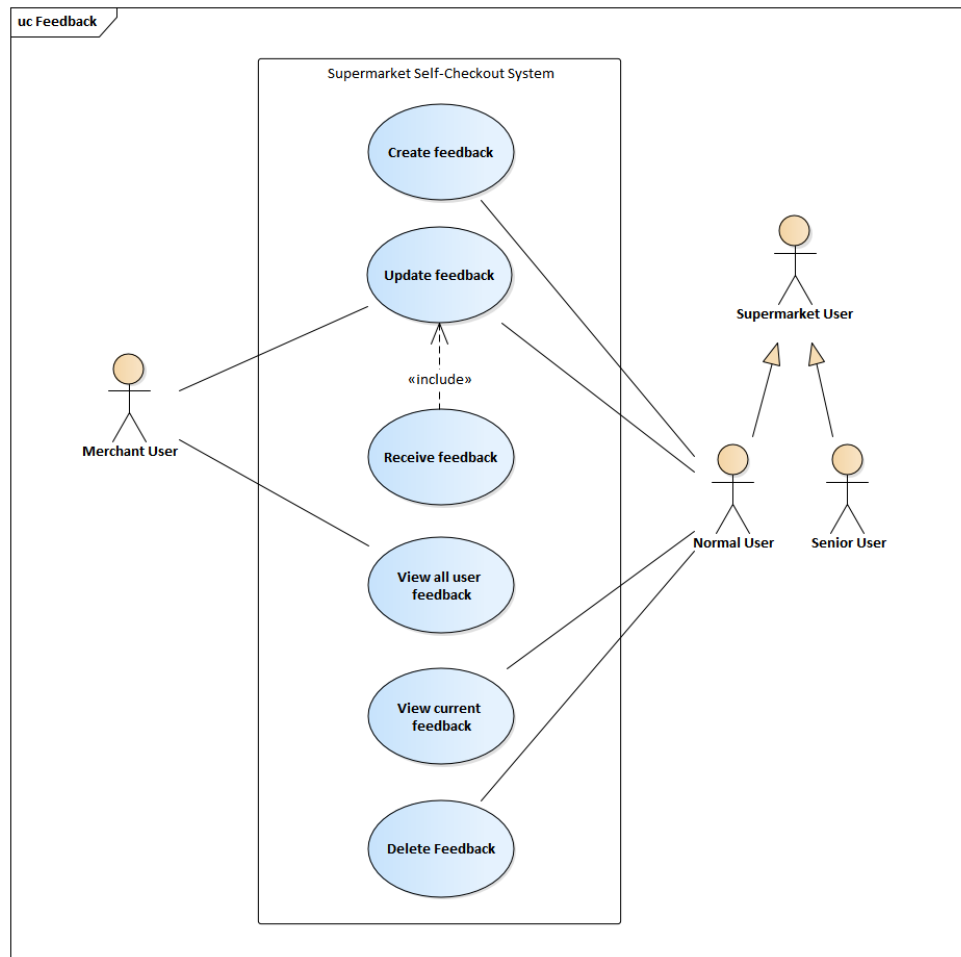


Figure 4.16: Feedback related use case diagram

4.4.1.6 Manage Products

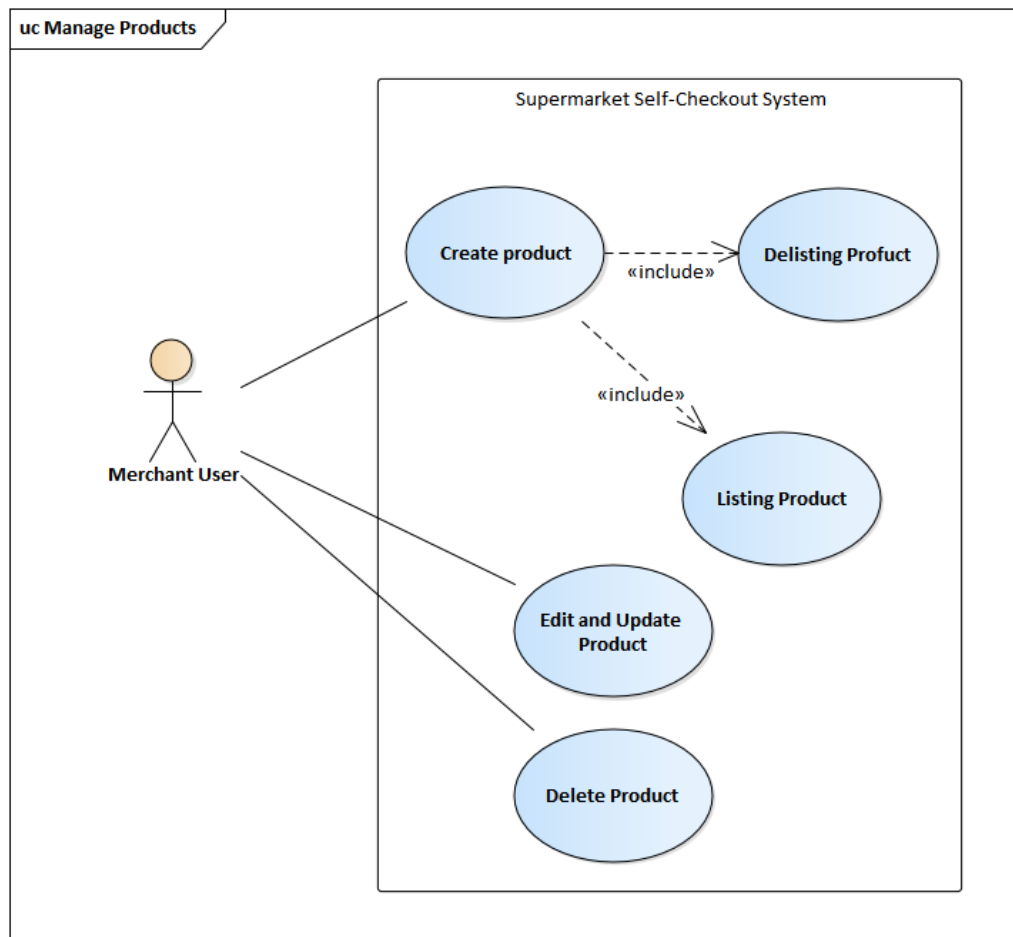


Figure 4.17: Manage products use case diagram

4.4.1.7 Advanced Search Functions

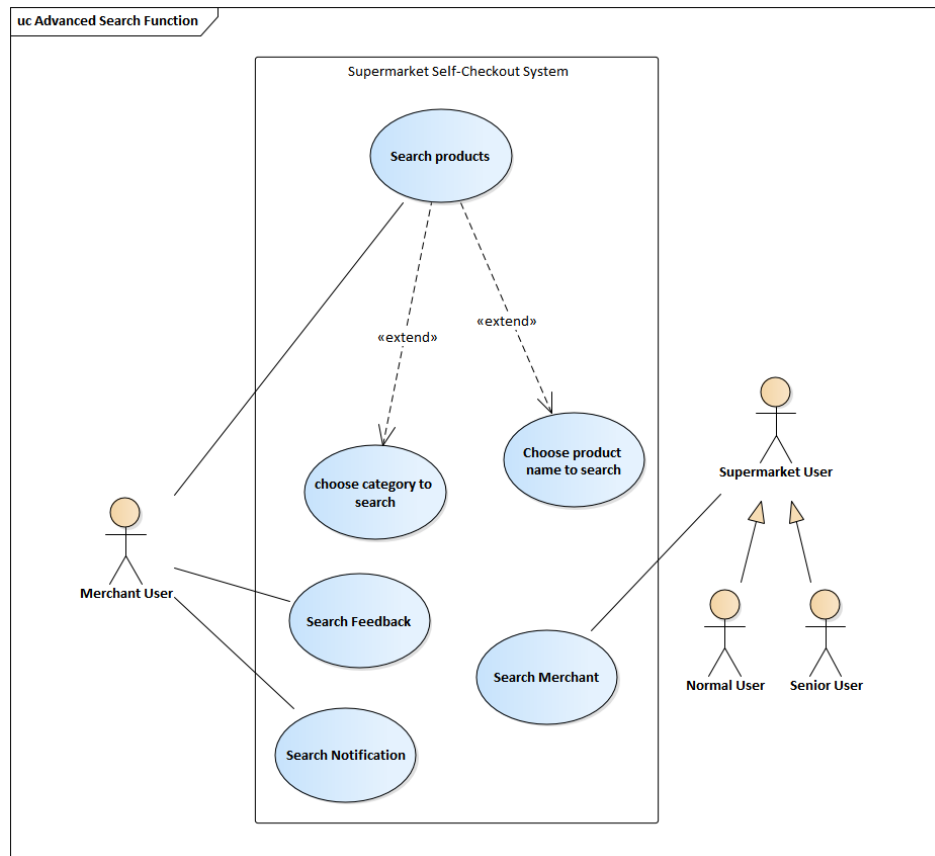


Figure 4.18: advance search function use case diagram

4.4.1.8 Manage Merchant

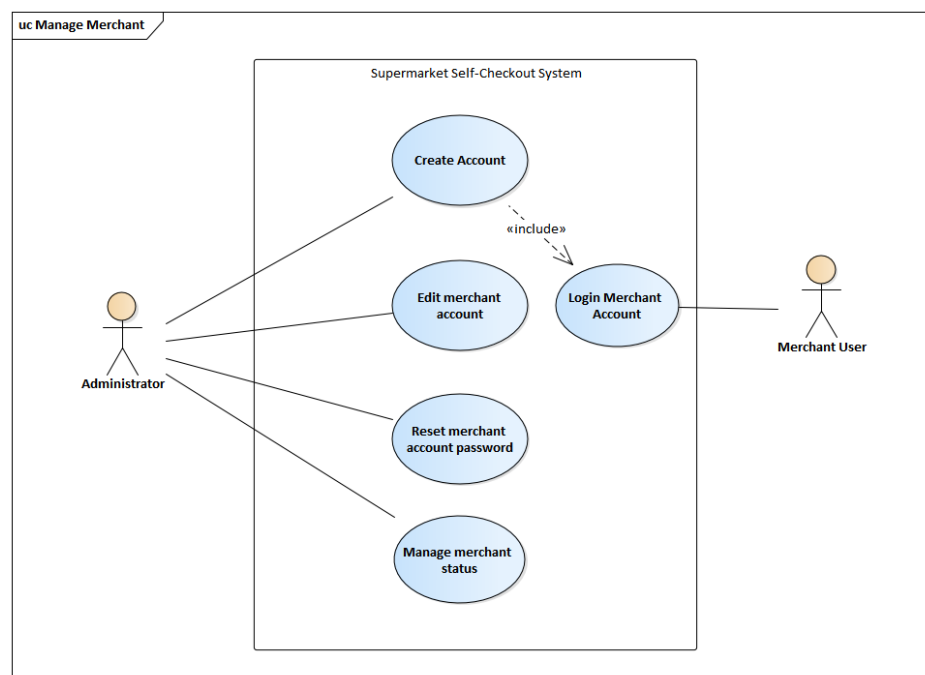


Figure 4.19: Manage merchant use case diagram

4.4.1.9 Manage Customer News

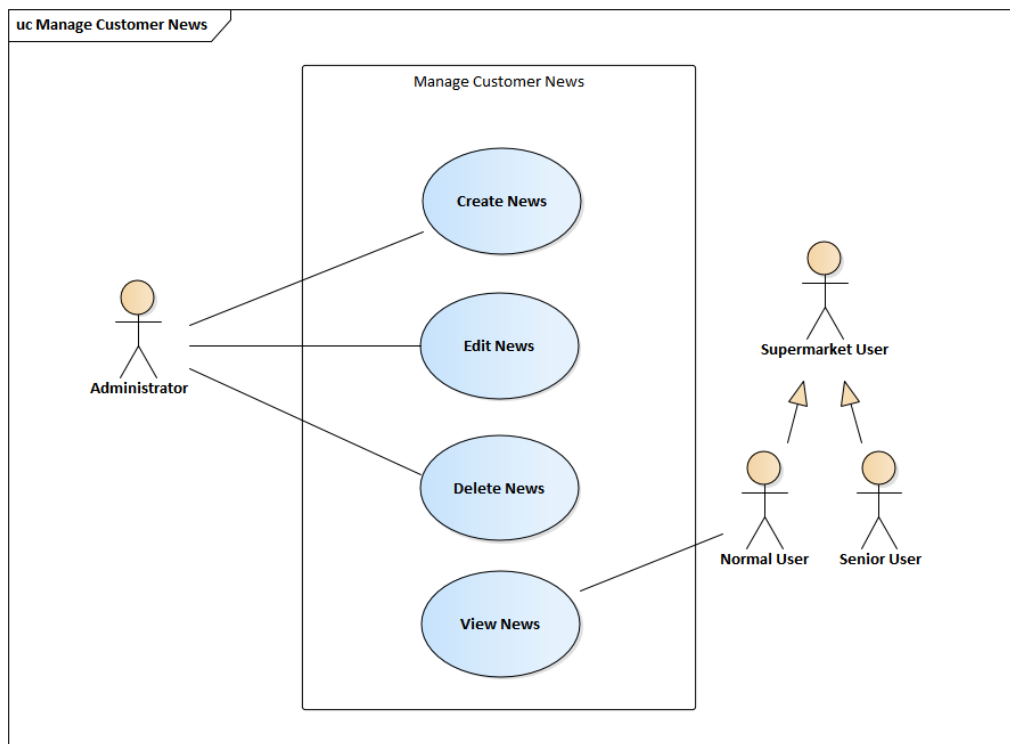


Figure 4.20: Manage customer news use case diagram

4.4.1.10 Location

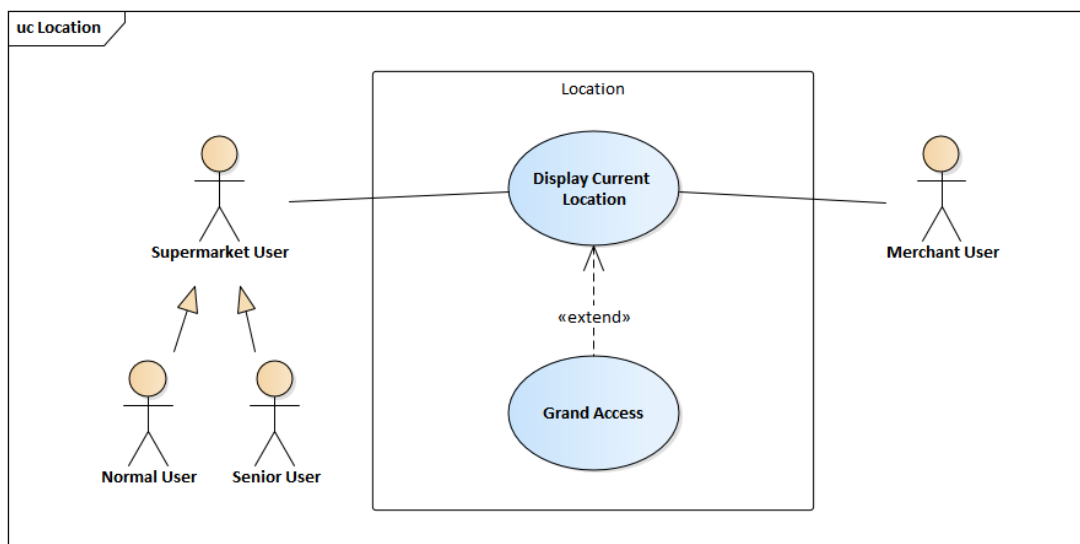


Figure 4.21: Location use case diagram

4.5 Use Case Description

4.5.1 Mobile Application for Supermarket Self-Checkout

Use case name	Scan item
Actor	Supermarket user
Description	Supermarket user scan item using the phone scanner integrated into the mobile application.
<p>Flow of Events</p> <ol style="list-style-type: none"> 1. Supermarket user accepts grand access for scanning and current location. 2. Supermarket user scan item with item barcode. 3. Supermarket user adds quantity for the existing scanned item. 4. Supermarket user confirm add item into cart. 5. System display add item succeeded. 	
<p>Alternative Flow of Events:</p> <p>2.1 QR scan failed</p> <p>2.1.1 Scan process terminates.</p>	

Use case name	Checkout items
Actor	Supermarket user
Description	Supermarket user checkout all items in the cart.
<p>Flow of Events</p> <ol style="list-style-type: none"> 1. System calculates total amount of items in the cart. 2. Supermarket user select pay for all items in the cart. 3. Supermarket user selects payment method to pay for the items. 4. System sends request to server to verify the payment. 5. System display payment is succeeded. 6. System generates e-receipt. 7. System displays generated e-receipt. 	
<p>Alternative Flow of Events:</p> <p>4.1 Payment failed</p> <p>4.1.1 Payment process terminates.</p>	

Use case name	Log in
Actor	Supermarket user
Description	Supermarket user login to the system.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user inserts email and password to log in to the system. 2. System verifies the supermarket user authentication. 3. System display supermarket user login succeeded. 	
Alternative Flow of Events:	
2.1 Invalid email or password.	
<ol style="list-style-type: none"> 2.1.1 Supermarket user authentication failed. <ol style="list-style-type: none"> 2.1.1.1 System login terminated. 	

Use case name	Log out
Actor	Supermarket user
Description	Supermarket user logout from the system.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user presses the log out button. 2. System display notice of confirmation of log out process. 3. System display log out is succeeded. 	

Use case name	Recover password
Actor	Supermarket user
Description	Supermarket user recover their account forgotten password.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user click the forgot password at login page. 2. Supermarket user input the email of their account. 3. System sends a recovery verify code to the email account. 4. Supermarket user copy the verify code and paste to recovery the account password. 5. Supermarket user redirect to web page to reset the password. 6. System display password is changed successfully. 	

Use case name	Change password
Actor	Supermarket user
Description	Supermarket user changes their account password.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user is entering profile page. 2. System displays the personal information. 3. Supermarket user selects change password button. 4. System displays new password and confirm new password to merchant user. 5. Supermarket user fills in the new password and confirm new password. 6. Supermarket user select confirm change password button. 7. System display confirmation message to supermarket user. 8. System displays new password updated successfully. 	

Use case name	Update profile
Actor	Supermarket user
Description	Supermarket user updates their personal information
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user enters profile page. 2. System displays the personal information. 3. Supermarket user edits their information. 4. Supermarket user confirms edited information. 5. System displays personal information updated successfully. 	

Use case name	View notification
Actor	Supermarket user
Description	Supermarket user updates their personal information
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user clicks notification tab. 2. System displays the list of notifications. 3. Supermarket user selects a notification detail. 4. System displays the notification details. 	

Use case name	Create feedback
Actor	Supermarket user
Description	Supermarket user creates and submit feedback.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user selects feedback on home page. 2. Supermarket user creates new feedback. 3. Supermarket user fills in the information which include feedback, details, subject etc. 4. Supermarket user press create button to create current feedback. 5. System display confirmation message for creating the feedback. 6. Supermarket user agrees to create feedback by pressing the confirm button. 7. System display feedback is submitted successfully. 	

Use case name	Update feedback
Actor	Supermarket user
Description	Supermarket users update specific feedback.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user selects feedback on homepage. 2. Supermarket user selects specific feedback to update. 3. Supermarket user selects update button. 4. System display confirmation message for update the feedback. 5. Supermarket user agrees update action by pressing the confirm button. 6. System display feedback is updated successfully. 	

Use case name	Delete feedback
Actor	Supermarket user
Description	Supermarket user deletes specific feedback.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user selects feedback on homepage. 2. Supermarket user selects specific feedback to delete. 3. Supermarket user slides the feedback to left. 4. System display delete button. 5. Supermarket user press delete button. 	

6. System display confirmation message for deleting the feedback.
7. Supermarket agrees the delete action by pressing the confirm button.
8. System display feedback is deleted successfully.

Use case name	View Order History
Actor	Supermarket user
Description	Supermarket user view order history.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user views all order history on homepage. 2. System display array of order history. 3. Supermarket user presses specific order history to view details. 4. System displays specific order history details. 	

Use case name	Grand Access Location
Actor	Supermarket user
Description	Supermarket user's location is granted to access.
Flow of Events	
<ol style="list-style-type: none"> 1. Supermarket user login into application with credential. 2. System request location permission from supermarket user. 3. Supermarket user accepts the permission. 4. System track user current location to find nearest merchant store. 5. System display nearest merchant profile picture. 	

4.5.2 Web-based application for Merchant Users

Use case name	Create product
Actor	Merchant user
Description	Merchant user creates products to the supermarket self-checkout application.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user select add product button. 2. Merchant user inserts all information by inputting product name, product category, product description, product promotion price, product original thumbnail, image etc. 	

<ol style="list-style-type: none"> 3. Merchant user chooses to publish with delisting the product. 4. Merchant user confirms all the information is correct then choose to publish live product. 5. System display product is uploaded successfully.
<p>Alternative Flow of Events:</p> <ol style="list-style-type: none"> 3.1 Publish failed <ol style="list-style-type: none"> 3.1.1 Display error message. 3.1.2 Merchant user input empty field. 3.1.3 Repeat again from flow 3.

Use case name	Edit and update product
Actor	Merchant user
Description	Merchant user edits and updates product in the supermarket self-checkout mobile application.
<p>Flow of Events</p> <ol style="list-style-type: none"> 1. Merchant user selects specific product. 2. System display product details. 3. Merchant user select edit button. 4. Merchant user edits existing product information. 5. Merchant user chooses to update with delisting the product. 6. Merchant users confirm all the information is correct then choose to publish live product. 7. System display product is updated successfully. 	
<p>Alternative Flow of Events:</p> <ol style="list-style-type: none"> 3.2 Publish failed <ol style="list-style-type: none"> 3.2.1 Display error message. 3.2.2 Merchant user input empty field. 3.2.3 Repeat again from flow 3. 	

Use case name	Delete product
Actor	Merchant user
Description	Merchant user deletes product from the supermarket self-checkout mobile application.

Flow of Events
<ol style="list-style-type: none"> 1. Merchant user selects specific product. 2. System display product details. 3. Merchant user select delete button. 4. System display confirmation message. 5. Merchant user confirms to delete the product. 6. System display product is deleted successfully.

Use case name	Log in
Actor	Merchant user
Description	Merchant user login to the system.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user inserts email and password to log in to the system. 2. System verifies the supermarket user authentication. 3. System display merchant user login succeeded. 	
Alternative Flow of Events:	
<ol style="list-style-type: none"> 1.1 Invalid email or password. <ol style="list-style-type: none"> 1.1.1 Merchant user authentication failed. 1.1.2 System login terminated. 	

Use case name	Log out
Actor	Merchant user
Description	Merchant user logout from the system.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user presses the log out button. 2. System display notice of confirmation of log out process. 3. System display log out is succeeded. 	

Use case name	Recover password
Actor	Merchant user
Description	Merchant user recover their account forgotten password.
Flow of Events	

7. Merchant user click the forgot password at login page.
8. Merchant user input the email of their account.
1. System sends a recovery verify code to the email account.
2. Merchant user copy the verify code and paste to recovery the account password.
3. Merchant user redirect to web page to reset the password.
4. System display password is changed successfully.

Use case name	Update profile
Actor	Merchant user
Description	Merchant user updates their personal information
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user is entering profile page. 2. System displays the personal information. 3. Merchant user is editing their information. 4. Merchant user confirms edited information. 5. System displays personal information updated successfully. 	

Use case name	Change password
Actor	Merchant user
Description	Merchant user change their account password.
Flow of Events	
<ol style="list-style-type: none"> 9. Merchant user is entering profile page. 10. System displays the personal information. 11. Merchant user selects change password button. 12. System displays new password and confirm new password to merchant user. 13. Merchant user fills in the new password and confirm new password. 14. Merchant user select confirm change password button. 15. System display confirmation message to merchant user. 16. System displays new password updated successfully. 	

Use case name	Create notification
---------------	---------------------

Actor	Merchant user
Description	Merchant user creates a notification to supermarket user.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user goes to manage user tab. 2. Merchant user select create notification 3. Merchant user fills in the information by inputting title and description. 4. Merchant user click create notification to fire up the notification to all users. 5. System display notification has been sent successfully. 	

Use case name	Update notification
Actor	Merchant user
Description	Merchant user update existing notification to supermarket user.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user goes to manage user tab. 2. Merchant user selects an existing notification. 3. Merchant user edits the information. 4. Merchant user click update notification and send to all users. 5. System display notification has been sent successfully. 	

Use case name	Delete notification
Actor	Merchant user
Description	Merchant user deletes existing notification.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user goes to notification tab. 2. Merchant user selects an existing notification. 3. Merchant user deletes the existing notification. 4. System display notification has been deleted successfully. 	

Use case name	View feedback
Actor	Merchant user
Description	Merchant user view existing feedback from

	supermarket user.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user goes to feedback tab. 2. Merchant user selects user feedback. 3. System displays a list of user feedback. 4. Merchant user clicks specific user feedback. 	

Use case name	Update feedback
Actor	Merchant user
Description	Merchant user update/follow up the user feedback.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user enter feedback tab. 2. Merchant user selects specific notification. 3. Merchant user select follow up button. 4. Merchant user input all information. 5. Merchant user click send to the user. 6. System display feedback has been sent successfully. 	

Use case name	Advanced Search
Actor	Merchant user
Description	Merchant user uses advanced search to full searching for products.
Flow of Events	
<ol style="list-style-type: none"> 1. Merchant user manages product page. 2. Merchant user selects search bar. 3. Merchant user search products by inputting keyword under product name, category, listing and delisting etc. 4. System displays all the related products by search results. 	

4.5.3 Web-based application for Administrator

Use case name	Create merchant account
Actor	Administrator
Description	Administrator creates account for merchant user.

Flow of Events
<ol style="list-style-type: none"> 1. Administrator requests email address from merchant user. 2. Merchant user provides email address to administrator. 3. Administrator select manage merchant tab. 4. Administrator select create account. 5. Administrator fills in all information by inputting merchant name, merchant contact, merchant email, merchant address, merchant default password etc. 6. System displays merchant account in the merchant list.

Use case name	Edit Merchant Account
Actor	Administrator
Description	Administrator edits merchant account.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator selects the merchant details. 2. Administrator edits merchant details. 3. Administrator selects update button. 4. System updated merchant information and display on the merchant list. 	

Use case name	Manage Merchant Status
Actor	Administrator
Description	Administrator manages merchant subscription status.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator selects the merchant details. 2. Administrator switches merchant status. 3. Administrator selects the update button. 4. System updated the merchant status. 	

Use case name	Reset Password
Actor	Administrator
Description	Administrator reset merchant account password.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator selects the merchant details. 2. Administrator selects the reset password button. 	

3. System prompt confirmation for the reset to default password process.
4. Administrator agrees the reset password process.
5. System reset merchant password into default password.

Use case name	Create News
Actor	Administrator
Description	Administrator creates supermarket user side's news.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator press create button. 2. System pops up create news page modal. 3. Administrator key in news details by inputting news title, news description and news photo. 4. Administrator press create button in create news page. 5. System prompt confirmation for creating the current news. 6. Administrator agrees create news action. 7. System displays the added a news. 	

Use case name	Edit News
Actor	Administrator
Description	Administrator edits supermarket user side's news.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator press edit button on specific news. 2. System pops up edit news page modal. 3. Administrator edits news details. 4. Administrator press edit button in edit news page. 5. System prompt confirmation for updating the current news details. 6. Administrator agrees editing news action. 7. System updates the current news details. 	

Use case name	Delete News
Actor	Administrator
Description	Administrator deletes supermarket user side's news.
Flow of Events	
<ol style="list-style-type: none"> 1. Administrator press edit button on specific news. 	

2. System pops up edit news page modal.
3. Administrator press delete button at bottom of the modal.
4. System prompt confirmation for deleting the current news.
5. Administrator agrees delete action.
6. System deletes the current news.

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

This chapter focused on system design phase which include system architecture, system design pattern and database design. Besides, there are three user interfaces will be visualized in this chapter which are senior user interface, normal user interface and also the merchant side user interface.

5.2 System Architecture

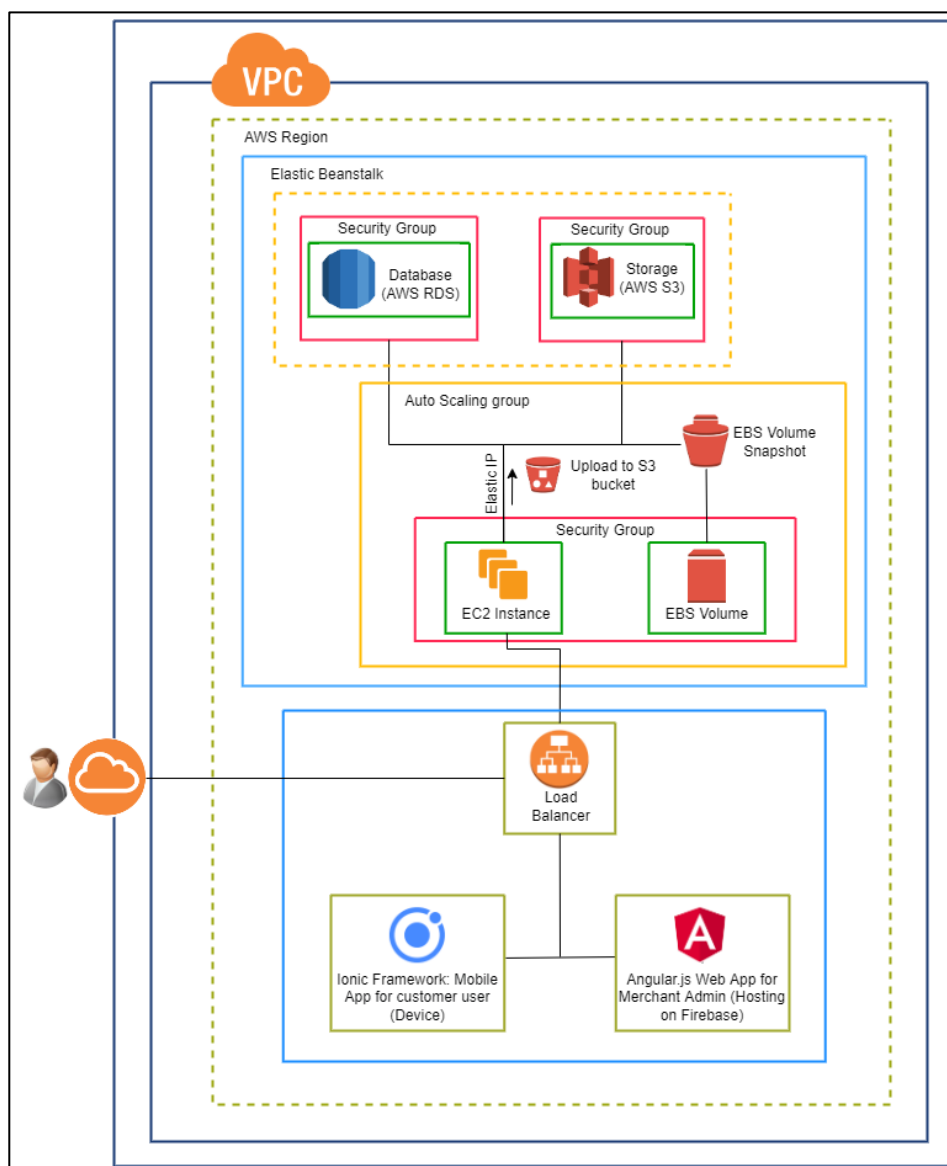


Figure 5.1: Overview of System Architecture

In this project, the system architecture involved multiple subnets and layers of its services. The overview of system architecture that shows above, subnet 1 have back-end server and subnet have two frond-end clients.

AWS Elastic Beanstalk is a framework which provide cloud service within AWS for deployment and auto scaling the web applications. Elastic Beanstalk will handle deployment from capacity provisioning, load balancing and auto scaling when developer uploading the software code to the provided services environment as it acts as a service (PaaS). In short, a central or fully stacked panel for hosting the applications, AWS Elastic Beanstalk is taking part on it. Other than this, there is no cost for creating an Elastic Beanstalk environment, but it cost for using their resources such as run application with their ES3 storage services. In this project, it will be using their storage services in the application to store JSON files, thumbnail, images, merchant data and also customer data.

Auto Scaling group act as logical grouping that helps auto scaling and manage services which contains collection of EC2 instances. It is provided more convenient to developer to maintain their application which search for the availability and auto scale such as add and remove EC2 instances which performed by the conditions of developer defined. Additionally, developer can add or remove EC2 instances using the auto scaling features such as dynamic and predictive scaling features. In shorts, auto scaling group helps developer with their server which are always accessible which it ensures efficiently and dynamically using the least amount of resources and reduce the overall cost.

Security group is one of important part must be involved in this project. It can include user personal information, merchant personal information within this application. The security group served as a virtual firewall and monitoring the traffic of server. In EC2 instance, the security group handles inbound and outbound traffic and request. A default security group is built if create a VPC. Rules and Regulations can be added to control the traffic based on the networks and there are inbound and outbound which the sets of rules have been separated.

5.3 System Design Pattern

The system design pattern in this project is apply using MVC (Model-View-Controller) architecture. The concept of MVC design pattern is separates into three logical components which is model, view and controller. Each of components is handle its important task which results an application. The presentation layer and business logic are separated by MVC design patterns. Historically, it was applied for desktop GUIs (Graphical User Interfaces). Nowadays, it is quite popular to be applied in developing a web application and also developing a mobile application.

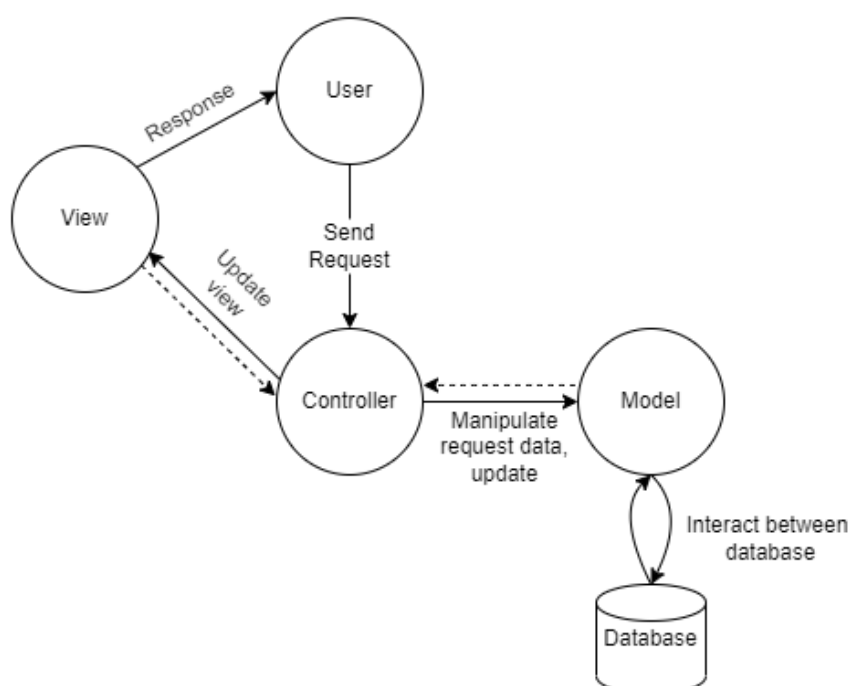


Figure 5.2: MVC architecture.

The figure above visualizes the MVC architecture with flows. First of all, model play an important role as component that stores data and data-oriented logic that operated by user. It is a representation of any business logic or data being passed between controller components. The view component is served as the presentation of data. The view can be represented the data from charts, diagram and also tables which all is display as customer view that include all UI components such as icon, buttons, toggles or even sliders. The role of controller component is to handles the request from client side and manipulate request data or even update data from model. The

controller is the input from user that uses mouse and keyboard to inform model and view to perform accordingly.

MVC architecture has bring up the benefit when come to the software development. It has easy code maintenance when the project is required to extend or maintain. In MVC architecture, each of component is allowed to separately test by developer. When developer wanted to test specific software component, this architecture has allowed them to test separately and would not affect each other. Besides, MVC also advantages in developing an object-oriented software since it facilitates rapidly and can be performed parallely that helps to avoid complexity. Additionally, developer can rely on MVC which are widely recognised as solution to recurrent issues and are employed to create scalable, reusable, and modular system. It works fine for web development which supports a larger team of developer to work together.

In this project, to build an application of supermarket checkout, it uses the framework of Ionic which also build with MVC design pattern. In ionic framework, AngularJS will detect the changes of model and updates the view which represent a two-way data binding. The view is interacted with user which to input data to controller and controller will manipulate the model and interact API then render to view.

5.4 Database Design

In this section, database design is including physical entity relational diagram and logical relational diagram. Both ERD diagram shows the relationship between entities. Besides, data dictionary for each of database table are describes which to show the relationship of attributes with each of table.

5.4.1 Physical Entity Relationship Diagram

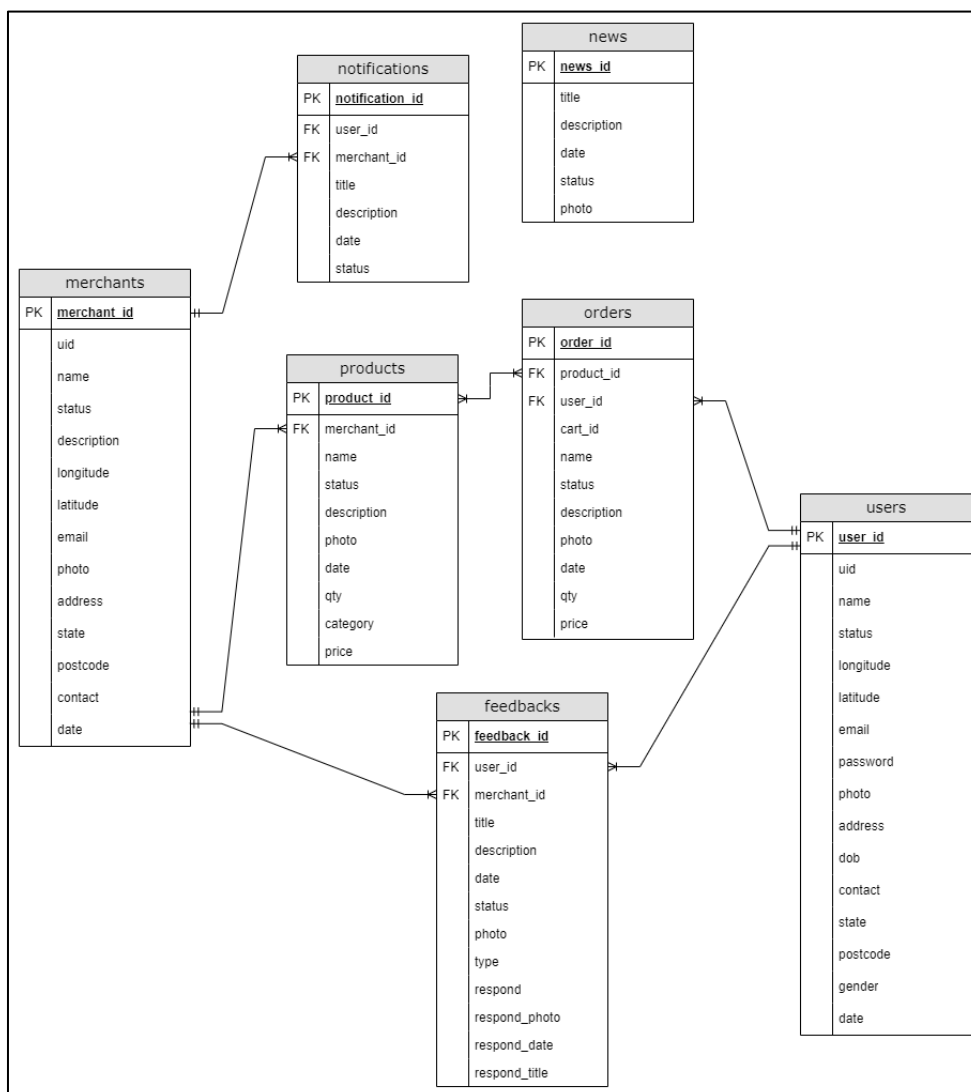


Figure 5.3: Physical Entity Relationship Diagram.

5.4.2 Logical Entity Relationship Diagram

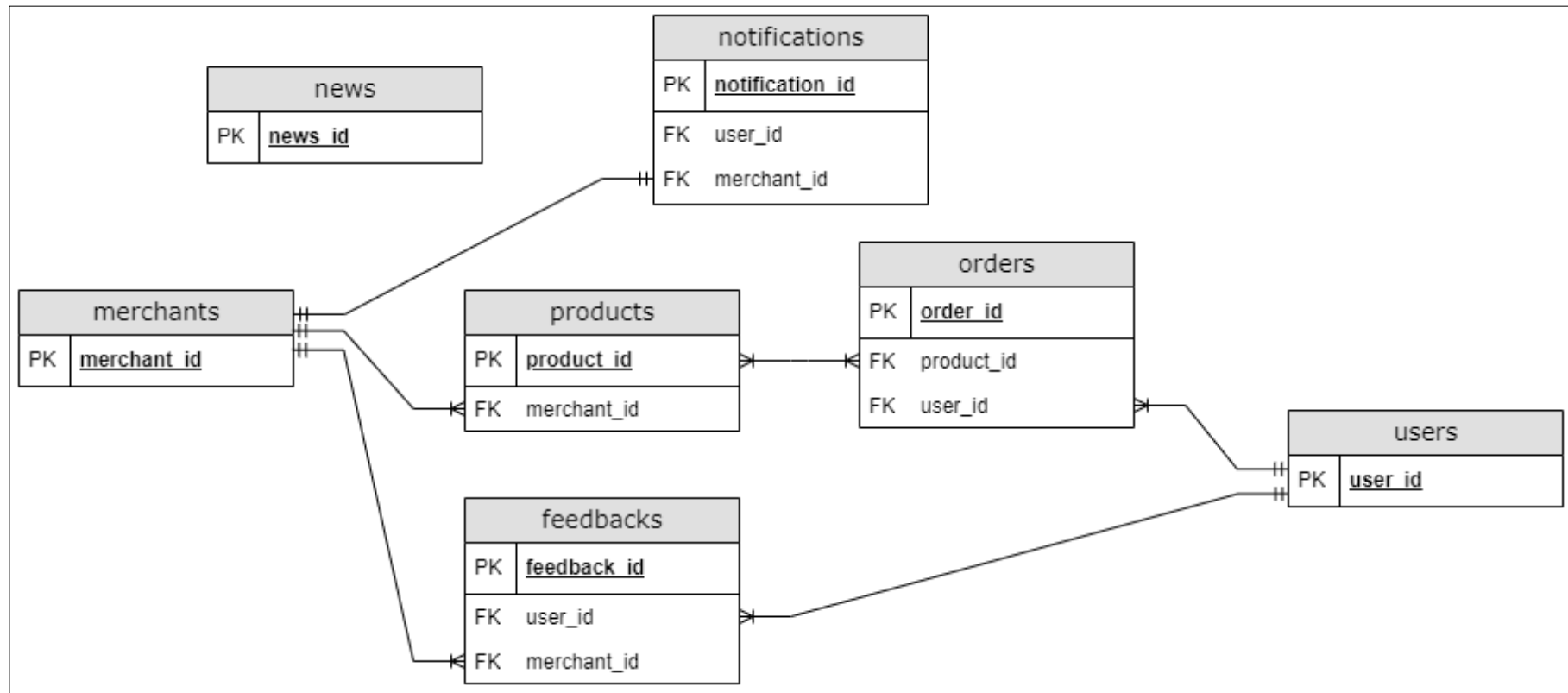


Figure 5.4: Logical Entity Relationship Diagram.

5.4.3 Data Dictionary

5.4.3.1 Merchants

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
merchant_id	SERIAL	Primary Key	Unique identifier of merchant user	0-9999999	1
uid	VARCHAR (16)	Not null	Authentication for merchant	Firestore generate	jBvq9xEqhHON79ivyHMJN6S0kaX2
name	VARCHAR (50)	Not null	Name of merchant user	*	KayShop
status	BOOLEAN	Not null	The status of merchant user	boolean	True, False
description	TEXT	Null	The description of merchant user	*	(.*?)
longitude	FLOAT	Not null	The longitude of the merchant location	number	101.12312312
latitude	FLOAT	Not null	The latitude of the merchant location	number	101.12312312
email	VARCHAR	Not null	The email of merchant user	Character and number	Testing3251@gmail.com
password	TEXT	Not null	The password of merchant user	*	Hello!2345

address	VARCHAR	Null	The address of merchant user	A-Z, 0-999999	No 1, jln friend
state	VARCHAR	Null	The state of merchant user	A-Z	Selangor
postcode	BIGINT	Null	The postcode of merchant user	0-999999	37410
contact	VARCHAR (20)	Not null	The contact number of merchant user	0-9	60123456789
date	BIGINT	Not null	The merchant account created date	System generates	16000000000

5.4.3.2 Products

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
product_id	SERIAL	Primary Key	Unique identifier of product	0-9999999	234673473214
merchant_id	VARCHAR (30)	Foreign Key	Merchant id	Foreign key	1
name	VARCHAR (120)	Not null	Name of product	A-Z	KayShop
status	BOOLEAN	Not null	The status of product	boolean	True, False
description	TEXT	Null	The description of product	*	New Fresh Vege
photo	JSON	Not null	Photo of products	Array to json	“[{‘name’: ‘link’}]”
date	BIGINT	Not null	Date product input	System generates	160000000
qty	BIGINT	Not null	Product capacity	0-9	2000
price	NUMERIC (30,2)	Not null	Product price	0-9	200
category	VARCHAR (100)	Not null	Product category	A-Z	meat

5.4.3.3 Orders

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
order_id	SERIAL	Primary Key	Unique identifier of order	0-99999999	234673473214
product_id	BIGINT	Foreign Key	Id of the product	Foreign key	1
user_id	BIGINT	Foreign Key	Id of the user	Foreign key	1
cart_id	VARCHAR	Not null	Id of the cart	Firestore generates	- N5a6KEt85lv8CTpD4mv
name	TEXT	Not null	Name of product	A-Z	KayShop
status	BOOLEAN	Not null	The status of order	boolean	True, False
description	TEXT	Null	The description of product	*	New Fresh vege
photo	JSON	Not null	The photo of product	Array to JSON	"[{ 'name': 'link' }]"
date	BIGINT	Not null	The date of order	System generates	1600000000
qty	BIGINT	Not null	The qty of order	0-9	5
price	NUMERIC (30,2)	Not null	The price of product	0-9	30

5.4.3.4 Users

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
user_id	INT	Primary Key	Unique identifier of user	0-9999999	234673473214
uid	VARCHAR	Not null	Firestore generated user id	0-9999999, a-z	BHJF34SA45Q2J
name	TEXT	Not null	Name of user	A-Z	KayShop
email	TEXT	Not null	The email of user	*	user@gmail.com
status	BOOLEAN	Not null	The status of user	boolean	True, False
description	TEXT	Null	The description of user	*	Hello World,
photo	JSON	Not null	The profile photo of user	Array to JSON	“[{‘name’: ‘link’}]”
date	BIGINT	Not null	The user account created date	System generates	160000000
dob	BIGINT	Not null	The date of birth of user	Timestamp	160000000
address	VARCHAR	Not null	Address of user	0-9999999, a-z	No 1, jln friend
state	VARCHAR	Not null	State of user	A-Z	Selangor
postcode	BIGINT	Not null	Postcode of user	0-999999	37410
contact	BIGINT	Not null	Contact number of user	0-9	012345689
gender	TEXT	Not null	The gender of user	*	Male
longitude	FLOAT	Not null	The longitude of user location	Number	101.11203333
latitude	FLOAT	Not null	The latitude of user location	Number	101.11203333

5.4.3.5 Feedbacks

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
feedback_id	SERIAL	Primary Key	Unique identifier of feedback	0-9999999	234673473214
user_uid	BIGINT	Foreign Key	Unique identifier of user	0-9999999	234673473214
merchant_uid	VARCHAR	Foreign Key	Unique identifier of merchant	Foreign key	1
status	BOOLEAN	Not null	The status of user	boolean	True, False
description	TEXT	Null	The description of user	*	Hi I am user
title	VARCHAR	Not null	The title of feedback	a-z,A-Z, 0-99999...	The product is broken
date	BIGINT	Not null	The feedback created date	Timestamp	02-12-2021
respond	TEXT	Not null	The respond details of feedback	*	Hi, user, thank your for your feedback!
photo	JSON	Null	The feedback photo	Array to JSON	"[{ 'name': 'link' }]"
type	VARCHAR	Not null	The type of feedback	*	general
respond_photo	JSON	Null	The merchant reply feedback photo	Array to JSON	"[{ 'name': 'link' }]"
respond_date	BIGINT	Not null	The feedback reply date	Timestamp	05-12-2021
respond_title	VARCHAR	Not null	The feedback reply subject	a-z,A-Z, 0-	Return and refund

				99999...	
--	--	--	--	----------	--

5.4.3.6 Notifications

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
notification_id	SERIAL	Primary Key	Unique identifier of notification	0-9999999	234673473214
user_id	INT	Foreign Key	Unique identifier of user	0-9999999	234673473214
merchant_id	VARCHAR	Foreign Key	Unique identifier of merchant	Foreign key	1
status	BOOLEAN	Not null	The status of notification	boolean	True, False
description	TEXT	Null	The description of notification	*	Hola, we got a big promotion today!
title	VARCHAR	Not null	The title of notification	A-Z, a-z ,0-99999	Promotion 11.11
date	BIGINT	Not null	The notification created date	timestamp	16000000
photo	JSON	Null	The notification photo	Array to JSON	"[{ 'name': 'link' }]"

5.4.3.7 News

Attribute	Data type	Constraint	Description	Acceptance Values	Example Values
new_id	SERIAL	Primary Key	Unique identifier of news	0-9999999	1
status	BOOLEAN	Not null	The status of news	boolean	True, False
description	TEXT	Null	The description of news	*	Today is promotion day!
title	VARCHAR	Not null	The title of news	A-Z,a-z,0-9	Follow our store to get discount!
date	BIGINT	Not null	The news created date	timestamp	16000000
photo	JSON	Not null	The photo of news	Array to JSON	“[{‘name’: ‘link’}]”

5.5 User Interface Design

In this section, shows the user interface design of proposed system. The user interface (UI) is drafted using the figma. The user interface design is present by screen prototyping which provide a blueprint for development stage. The user interface is drafted and meets the project expectations and support the software component. Besides, with the first draft of UI, it helps complete the project with shorter time. In this project, the total of user interfaces is 3 which one is web application for merchant-side. Mobile applications for normal supermarket user and mobile applications for senior supermarket user.

5.5.1 Web Application for Merchant

5.5.1.1 Merchant User Login Page

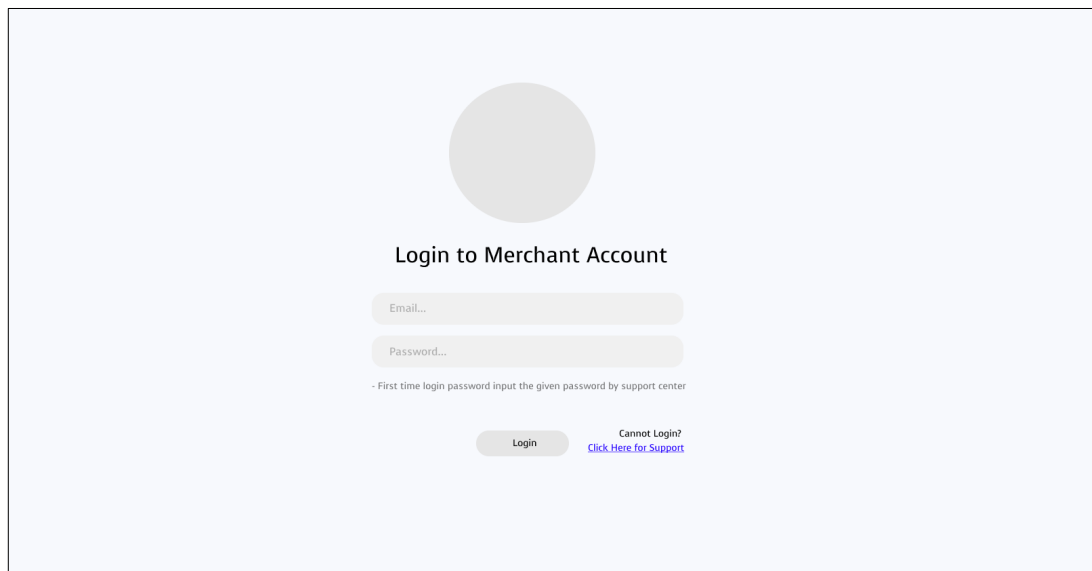


Figure 5.5: Merchant user login page screen.

5.5.1.2 Merchant User Manage Product Page

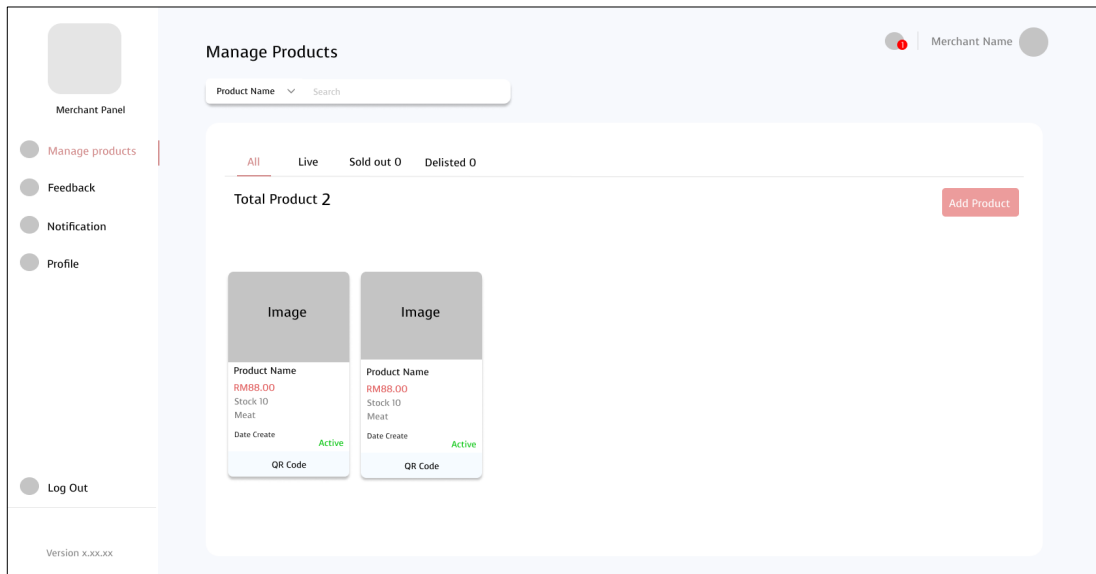


Figure 5.6: Merchant user manage product page screen.

5.5.1.3 Merchant User Add New Product Page

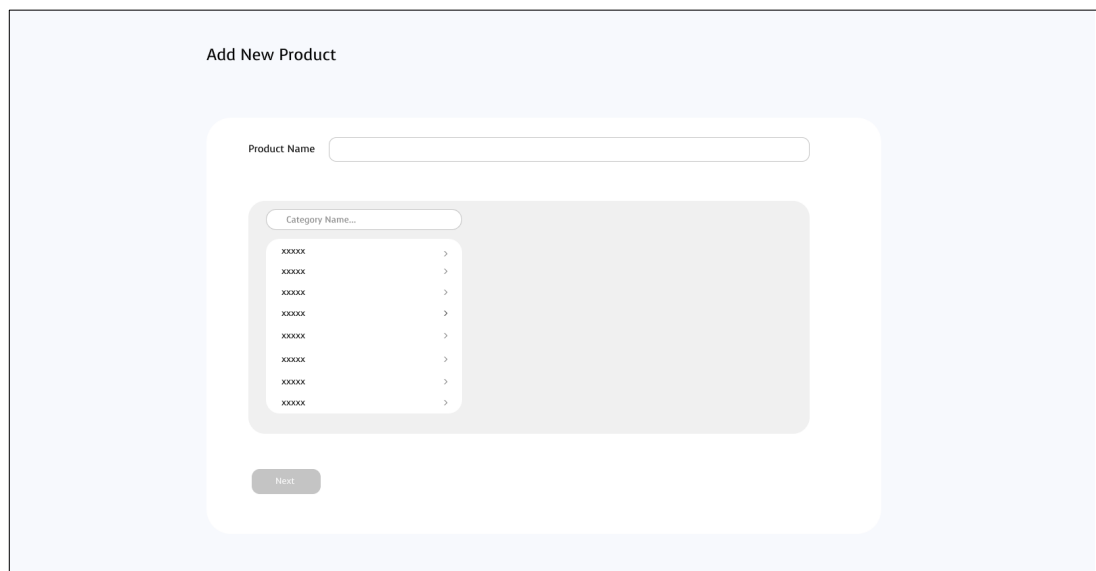


Figure 5.7: Merchant user add new product page screen.

5.5.1.4 Merchant User Add New Product continue Page

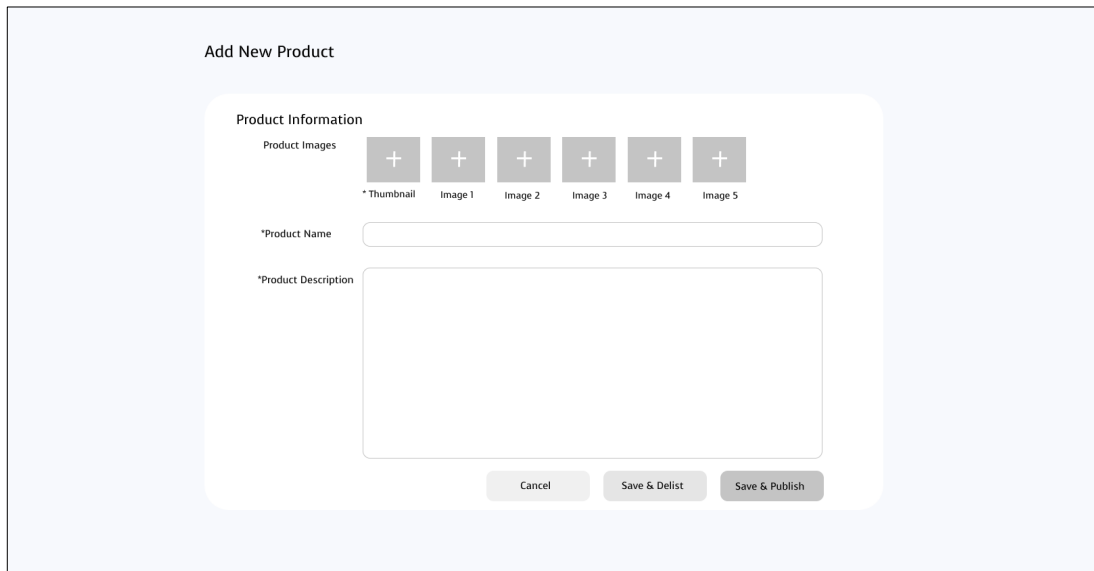


Figure 5.8: Merchant user add new product continue page screen.

5.5.1.5 Merchant User Manage Feedback List Page

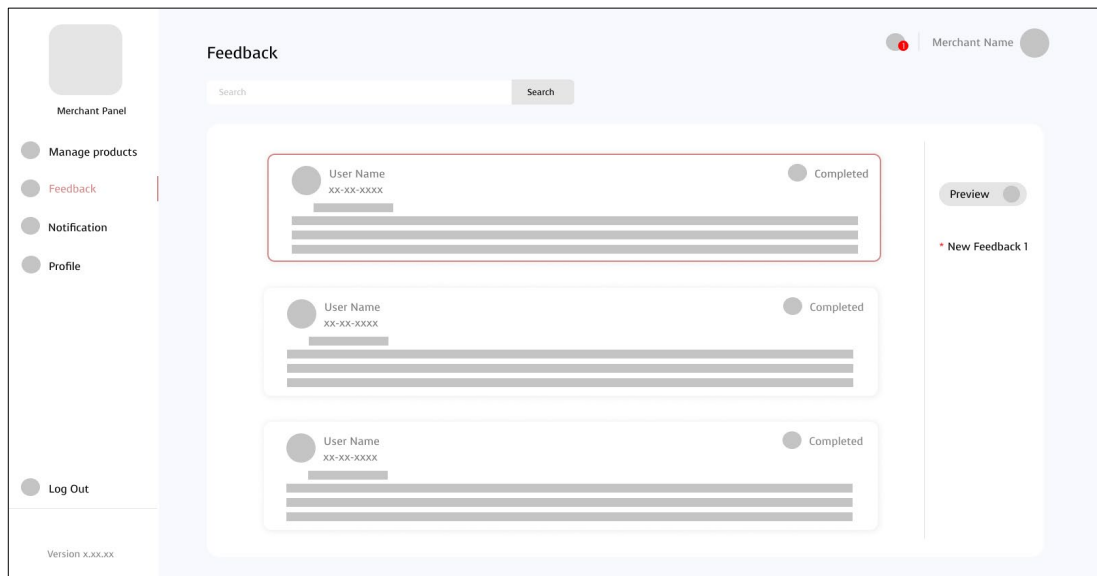


Figure 5.9: Merchant user manage feedback list page screen.

5.5.1.6 Merchant User Manage Specific Feedback Page

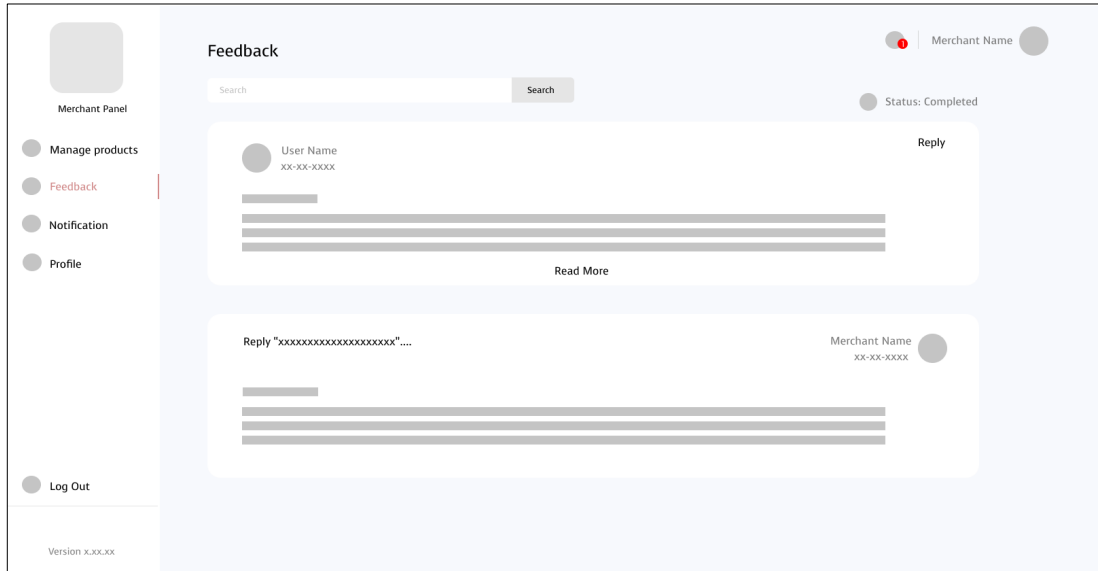


Figure 5.10: Merchant user manage specific feedback page screen.

5.5.1.7 Merchant User Reply Specific Feedback Page

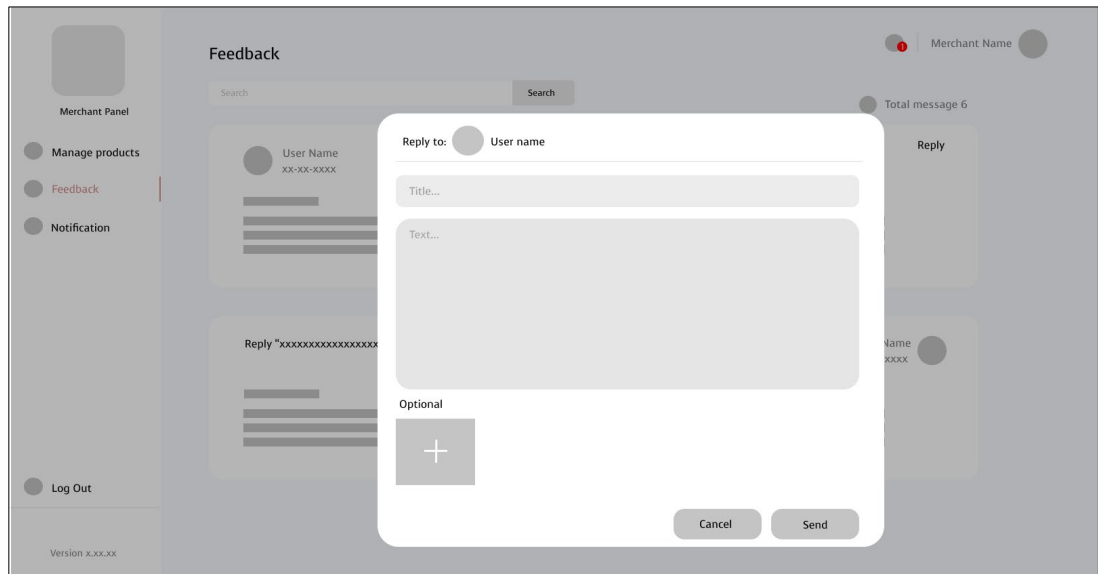


Figure 5.11: Merchant user Reply specific feedback page screen.

5.5.1.8 Merchant User Manage Notification Page

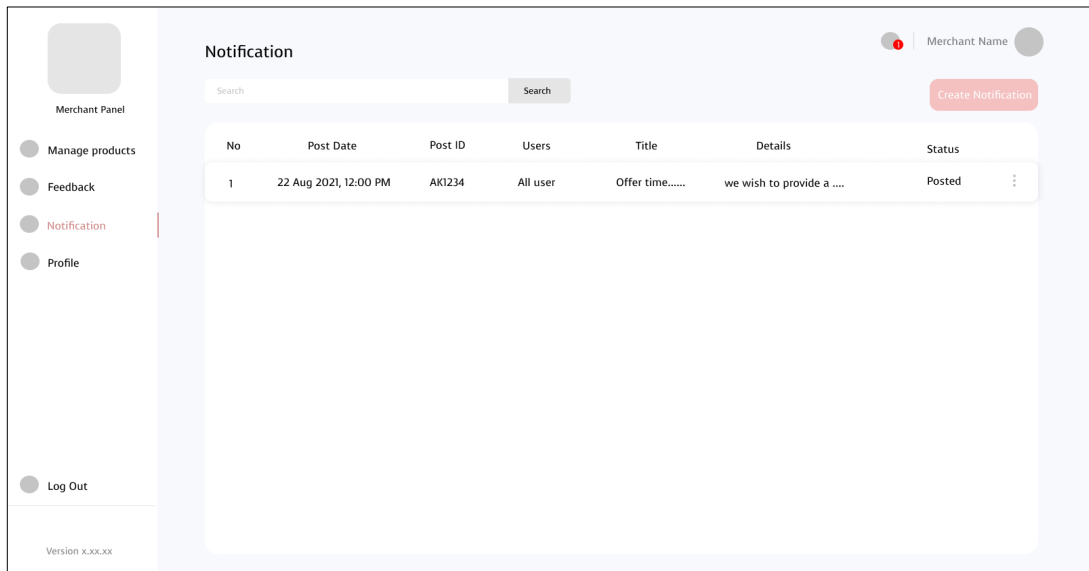


Figure 5.12: Merchant user manage notification page screen.

5.5.1.9 Merchant User Create Notification Modal

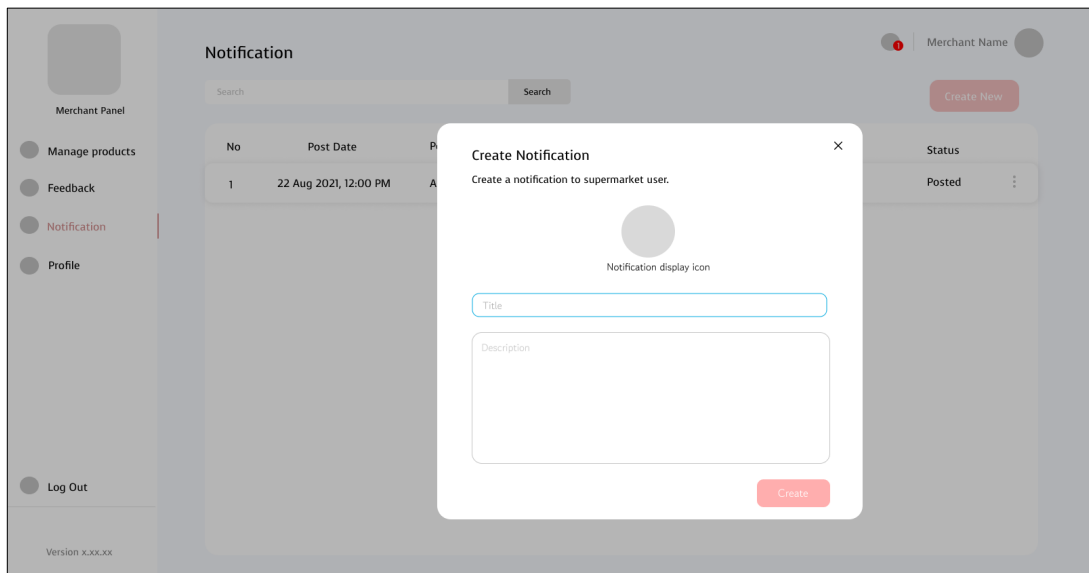


Figure 5.13: Merchant user create notification modal.

5.5.1.10 Merchant User Edit Profile Page

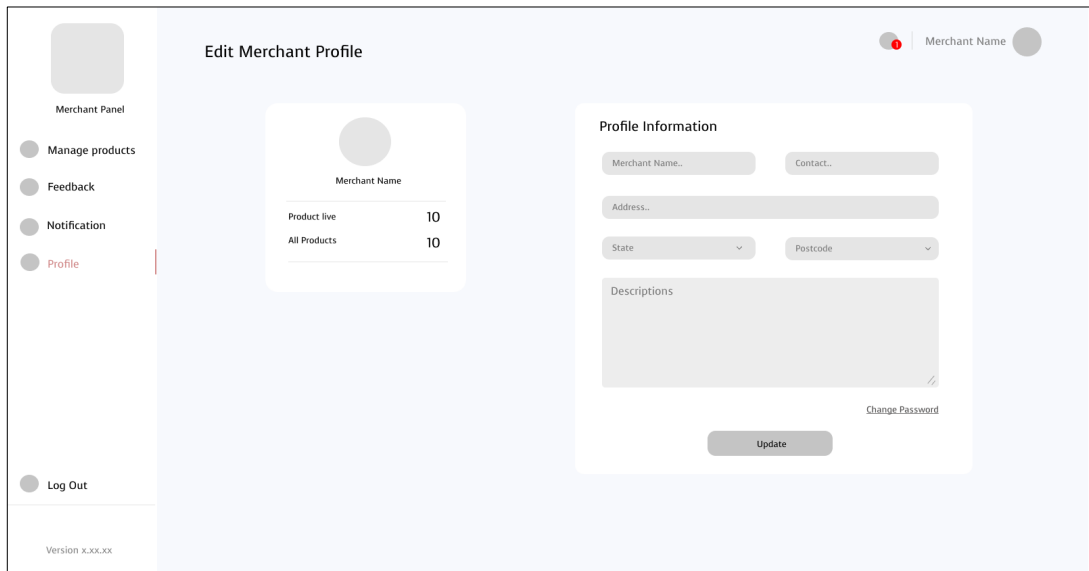


Figure 5.14: Merchant user edit profile page screen.

5.5.1.11 Merchant User Edit Notification Page

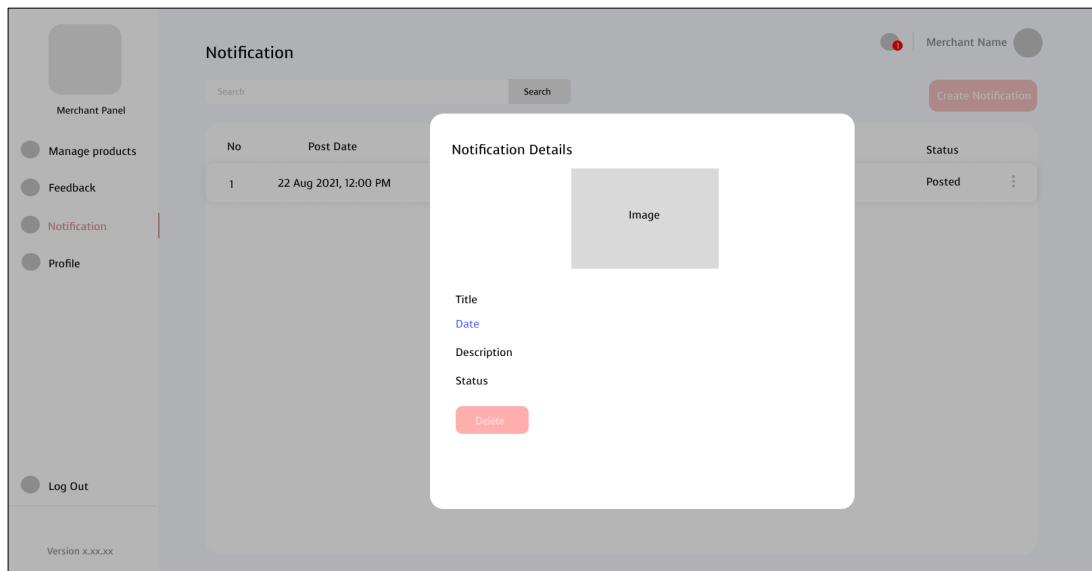
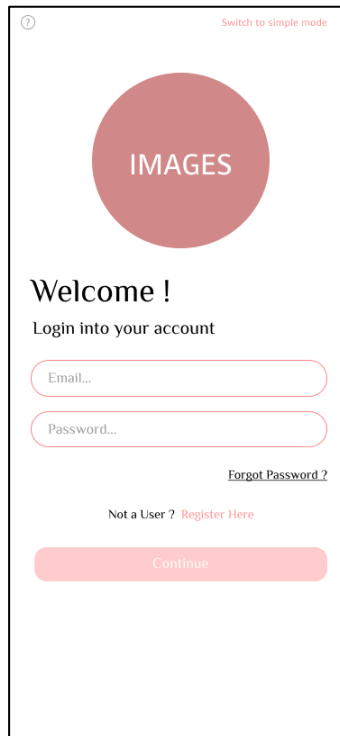


Figure 5.15: Merchant user edit notification page screen.

5.5.2 Supermarket Normal User

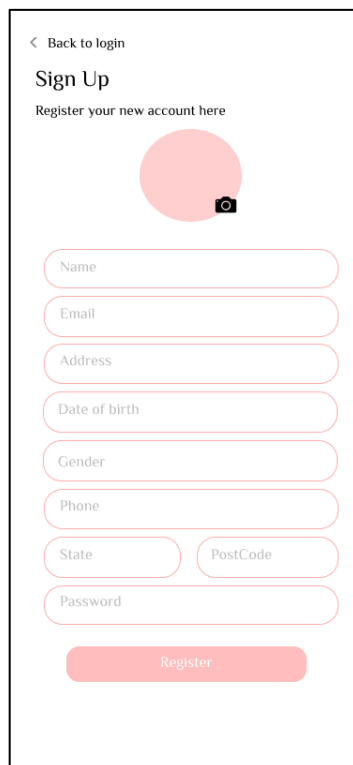
5.5.2.1 Supermarket User Login Page



The login page features a white background with a red circular logo at the top center containing the word "IMAGES". Above the logo is a small red question mark icon and the text "Switch to simple mode". Below the logo, the text "Welcome !" is displayed in a large, bold font, followed by "Login into your account" in a smaller font. The form includes two input fields: "Email..." and "Password...". Below the password field is a link for "Forgot Password?". At the bottom of the form, there is a link for "Not a User ? Register Here" and a large red "Continue" button.

Figure 5.16: Login page screen.

5.5.2.2 Supermarket User Sign Up Page



The register page has a white background with a red circular logo at the top center containing a camera icon. Above the logo is a back arrow and the text "Back to login". Below the logo, the text "Sign Up" is displayed in a large, bold font, followed by "Register your new account here" in a smaller font. The form includes several input fields: "Name", "Email", "Address", "Date of birth", "Gender", "Phone", "State", "PostCode", and "Password". At the bottom of the form, there is a large red "Register" button.

Figure 5.17: Register page screen.

5.5.2.3 Supermarket User Home Page

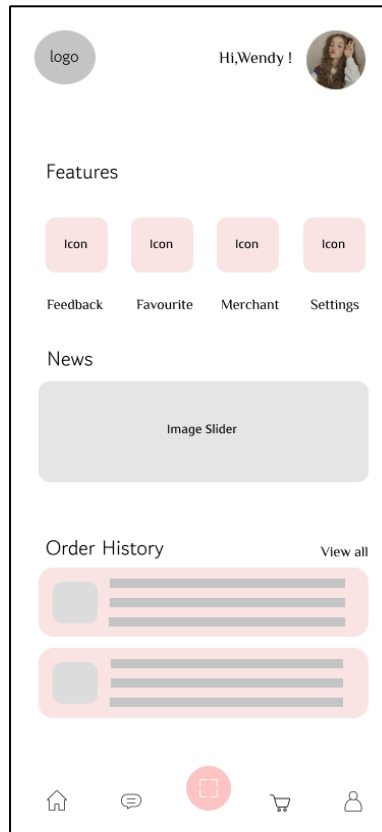


Figure 5.18: Home page screen.

5.5.2.4 Supermarket User Notification Page

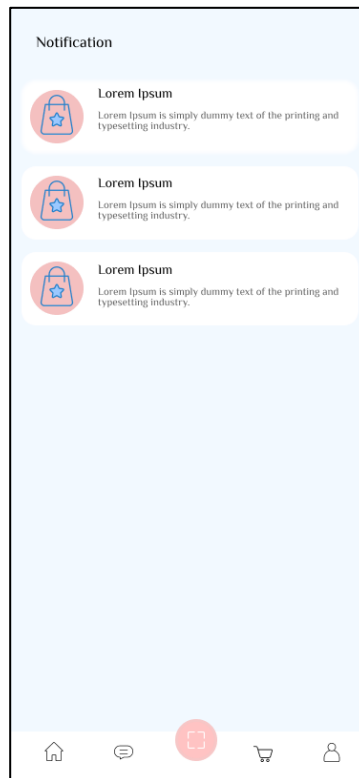


Figure 5.19: Notification page screen

5.5.2.5 Supermarket User Cart Page

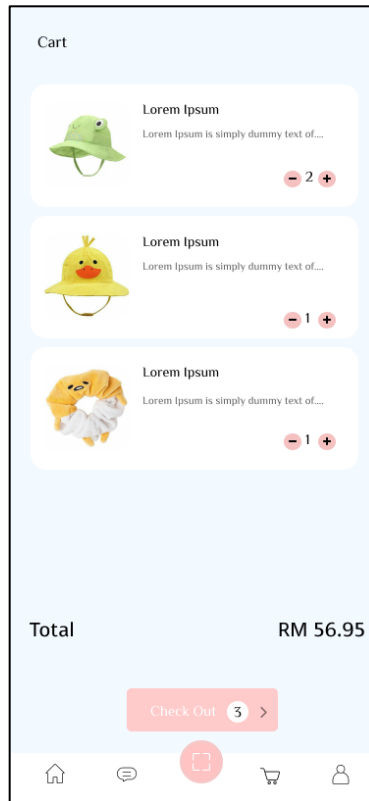


Figure 5.20: Cart page screen.

5.5.2.6 Supermarket User Profile Page

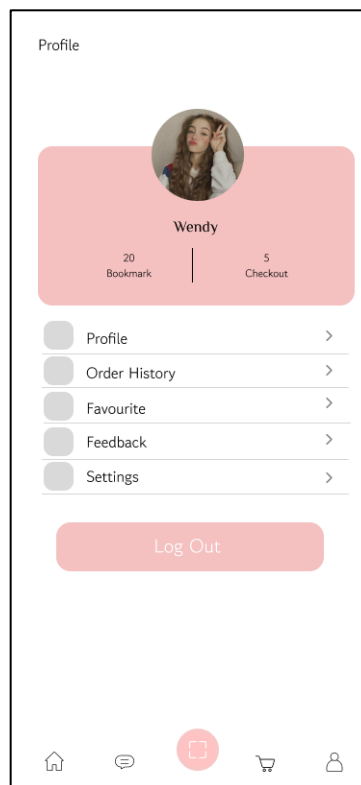


Figure 5.21: Profile page screen.

5.5.2.7 Supermarket User Edit Profile Page

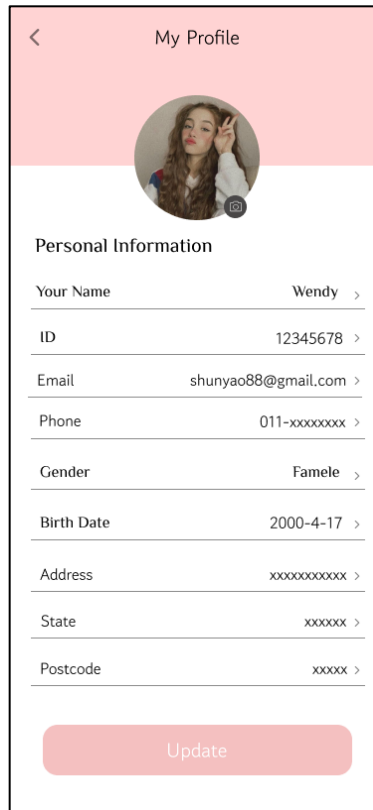


Figure 5.22: Edit profile page screen.

5.5.2.8 Supermarket User Order History Page

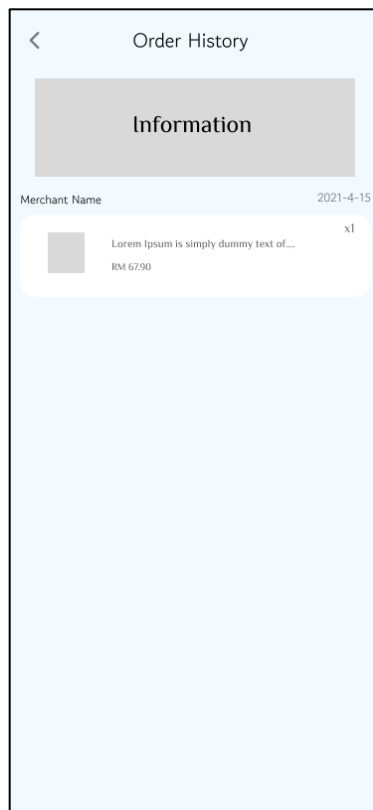


Figure 5.23: Order History page screen.

5.5.2.9 Supermarket User Favourite Page



Figure 5.24: Favorite page screen.

5.5.2.10 Supermarket User Feedback Page

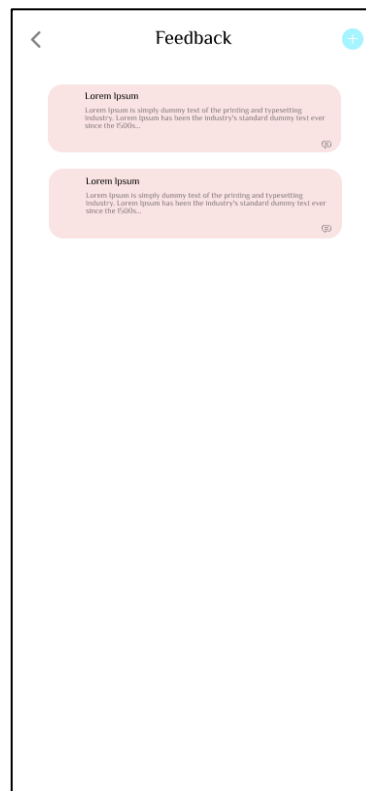


Figure 5.25: Feedback page screen.

5.5.2.11 Supermarket User Feedback Form Page

Create Feedback Cancel

Info

Which Merchant?
Select Merchant
Select Merchant

Enter a title
Title

Which type ?
Select Feedback Type
Type

Enter a title
Description

Attachment
Optional
+

Submit

Figure 5.26: Feedback form page screen.

5.5.2.12 Supermarket User Feedback Details Page

< Feedback Details

Subject: Testing
Status: Completed

User name
Date
Subject
Description

Merchant name
Date
Subject
Description

Figure 5.27: Feedback details page screen.

5.5.2.13 Supermarket User Scan Item Page

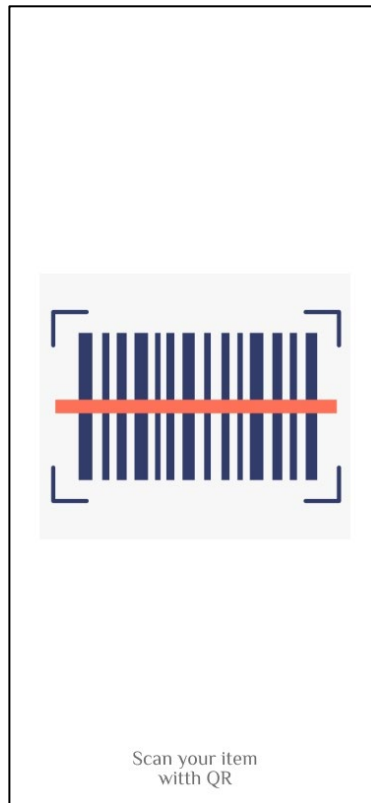


Figure 5.28: Scan item screen.

5.5.2.14 Supermarket User Add Item Page

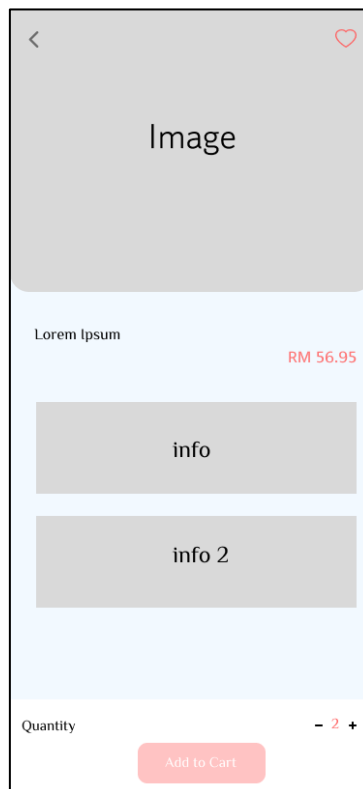


Figure 5.29: Add item page screen.

5.5.2.15 Supermarket User Order History Details Page



Figure 5.30: Order details page screen.

5.5.2.16 Supermarket User News Details Page



Figure 5.31: News details page screen.

5.5.2.17 Supermarket User Merchant List Page

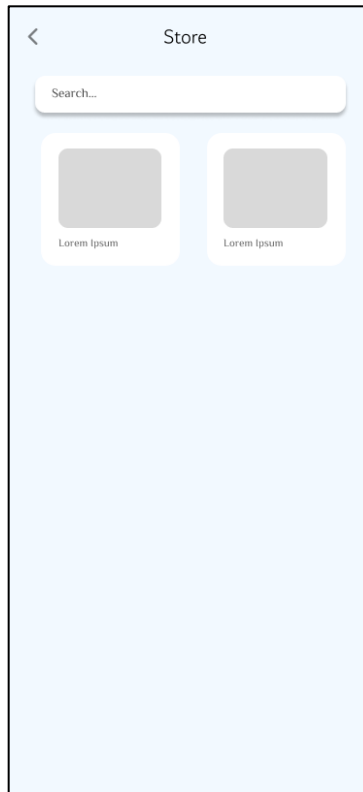


Figure 5.32: Merchant List page screen.

5.5.2.18 Supermarket User Settings Page

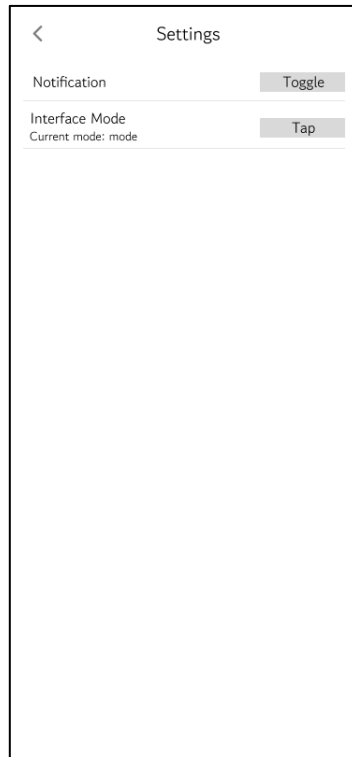


Figure 5.33: Settings page screen.

5.5.2.19 Supermarket User Recover Password Page

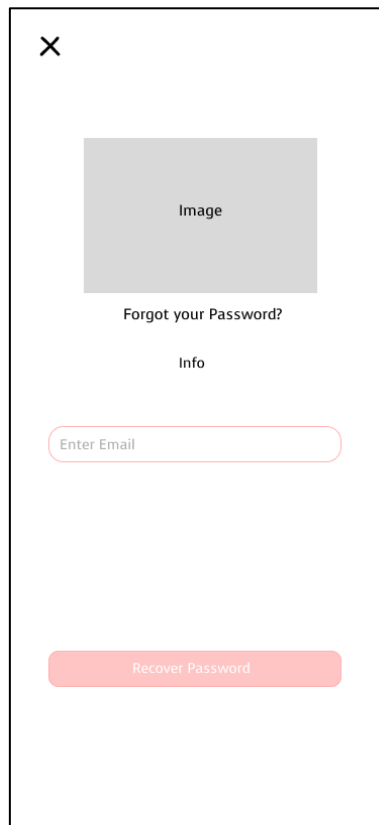


Figure 5.34: Recover Password page screen.

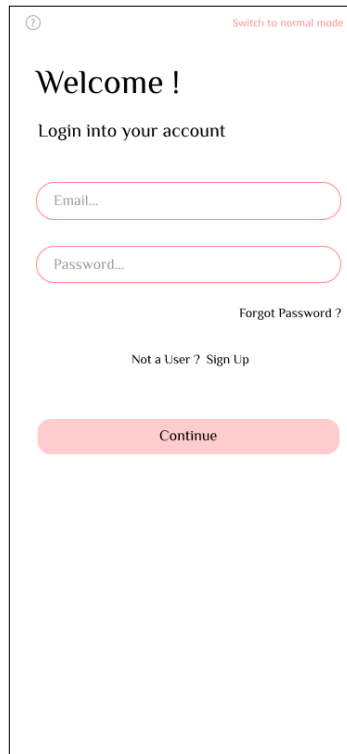
5.5.2.20 Supermarket User Notification Details Page



Figure 5.35: Notification details page screen.

5.5.3 Supermarket senior user

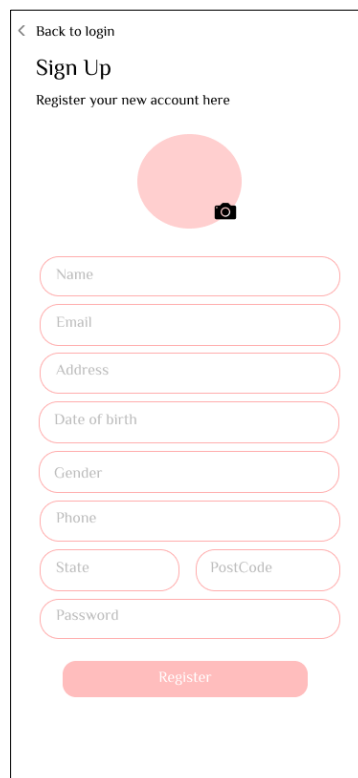
5.5.3.1 Supermarket Senior-Age User Login Page



The login page features a clean, minimalist design. At the top left, there is a question mark icon and a link to "Switch to normal mode". The main heading is "Welcome !", followed by the instruction "Login into your account". Below this are two input fields for "Email..." and "Password...". A "Forgot Password ?" link is positioned to the right of the password field. Below the input fields, there is a "Not a User ? Sign Up" link. At the bottom, a prominent red "Continue" button is centered.

Figure 5.36: Senior user login page screen.

5.5.3.2 Supermarket Senior-Age User Register Page



The register page has a "Back to login" link at the top left. The heading is "Sign Up" with the subtext "Register your new account here". A large red circle with a camera icon indicates a profile picture upload area. Below this are several input fields: "Name", "Email", "Address", "Date of birth", "Gender", "Phone", "State", "PostCode", and "Password". A red "Register" button is located at the bottom of the form.

Figure 5.37: Senior user register page screen.

5.5.3.3 Supermarket Senior-Age User Home Page

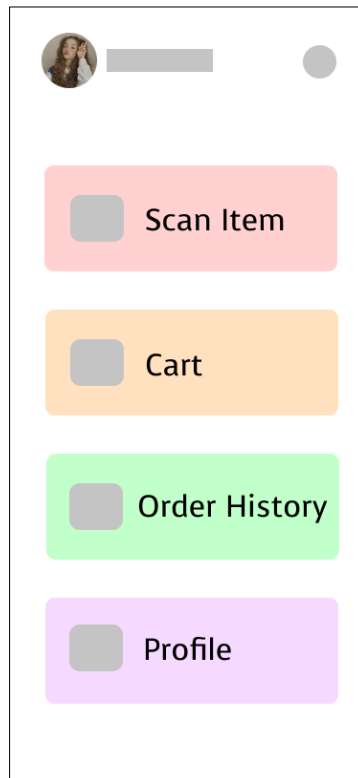


Figure 5.38: Senior user home page screen.

5.5.3.4 Supermarket Senior-Age User Edit Profile Page

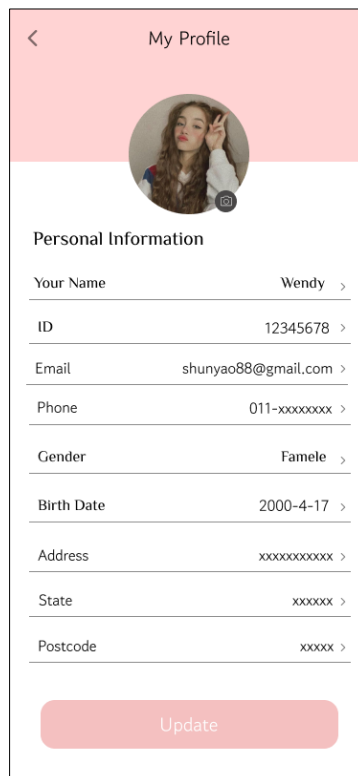


Figure 5.39: Senior user edit profile page screen.

5.5.3.5 Supermarket Senior-Age User Cart Page

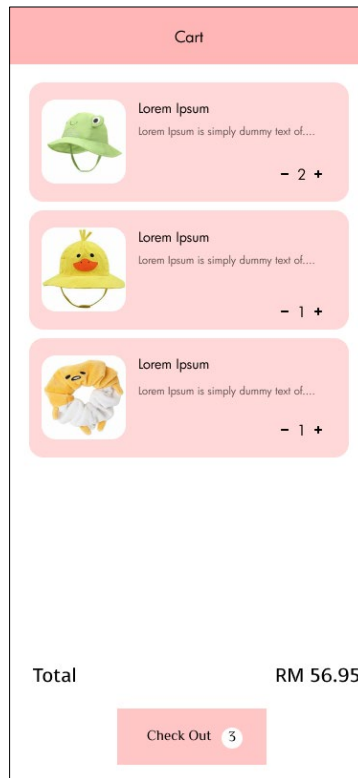


Figure 5.40: Senior user cart page screen.

5.5.3.6 Supermarket Senior-Age User Scan Page

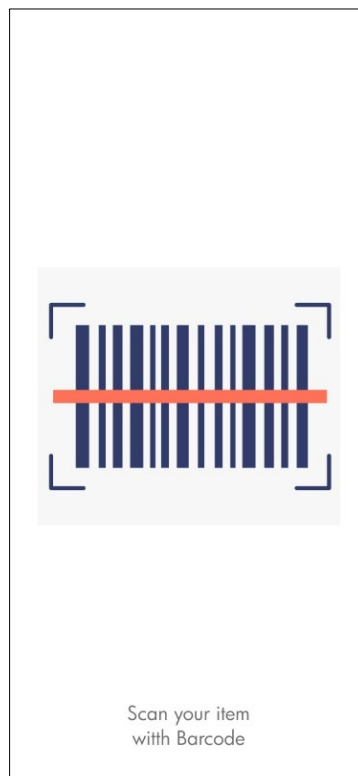


Figure 5.41: Senior user scan screen.

5.5.3.7 Supermarket Senior-Age User Order History Page

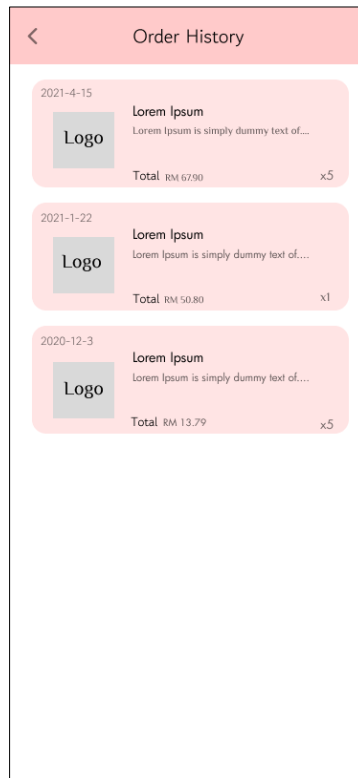


Figure 5.42: Senior user order history page screen.

5.5.3.8 Supermarket Senior-Age User Order Details Page

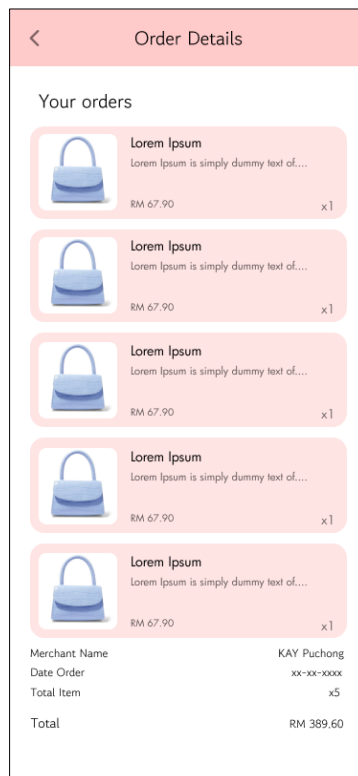


Figure 5.43: Senior user order details page screen.

5.5.3.9 Supermarket Senior-Age User Profile Page

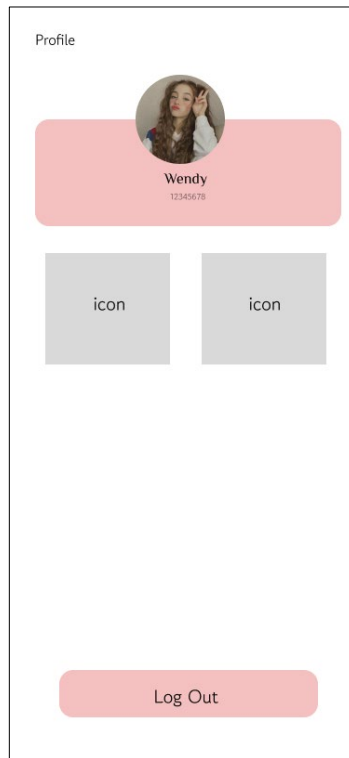


Figure 5.44: Senior user profile page screen.

5.5.3.10 Supermarket Senior-Age User Profile Page

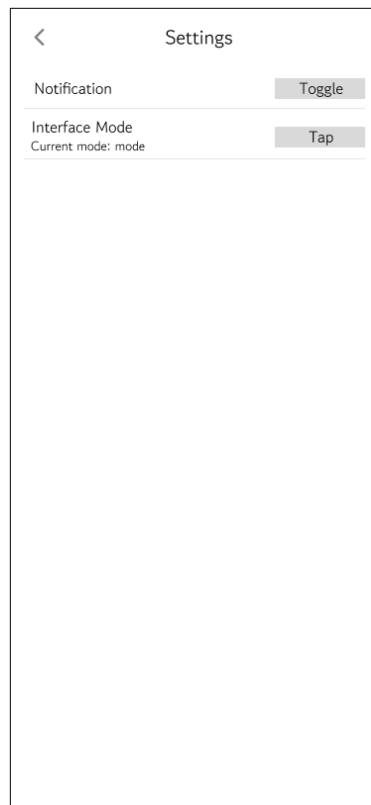


Figure 5.45: Senior user settings page screen.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Backend Server

The node.js backend server is resided in this project. In this backend system, there are three layers which is controller layer, model layer and service layer. As mentioned in previous chapter, the system is relied upon MVC architecture and when come to backend server, service is taking part as code reuse, it can inject single piece function across many controllers. This section will discuss the overview of backend system.

6.1.1 Overview backend server

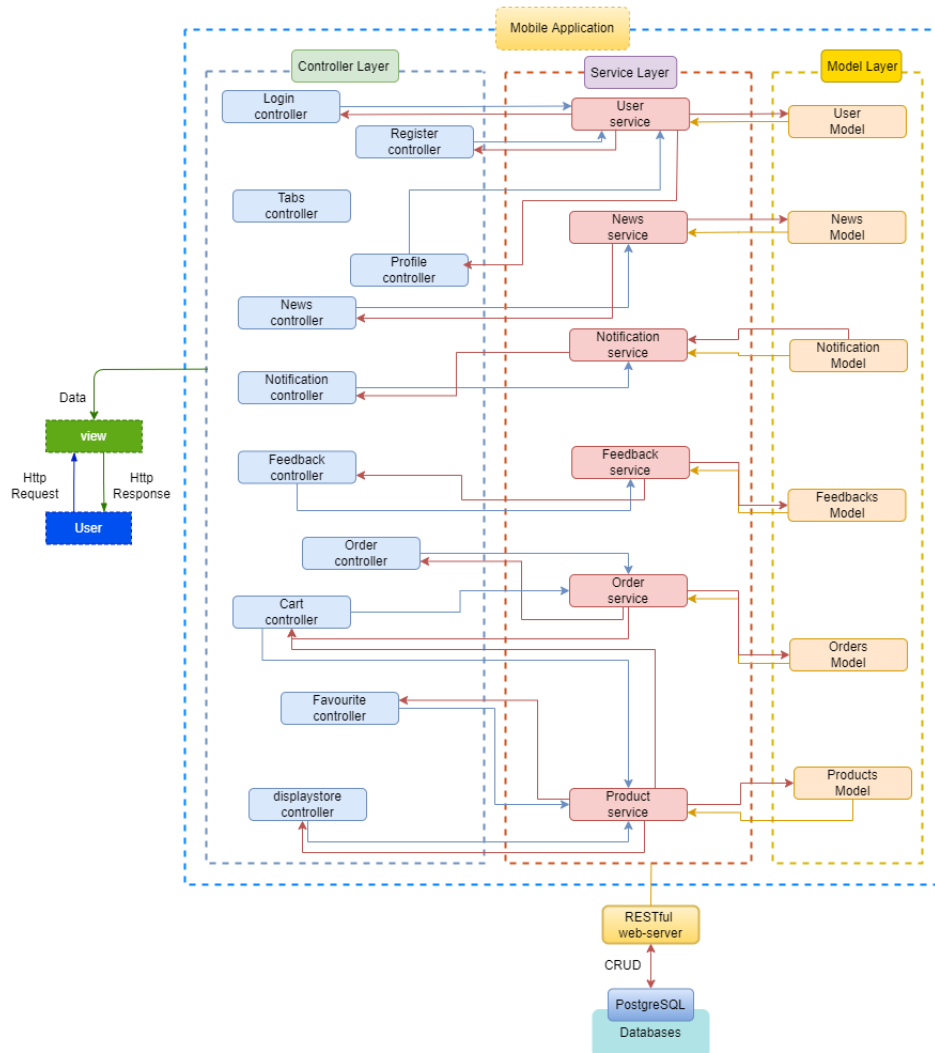


Figure 6.1: Overview the RESTful API request from mobile application.

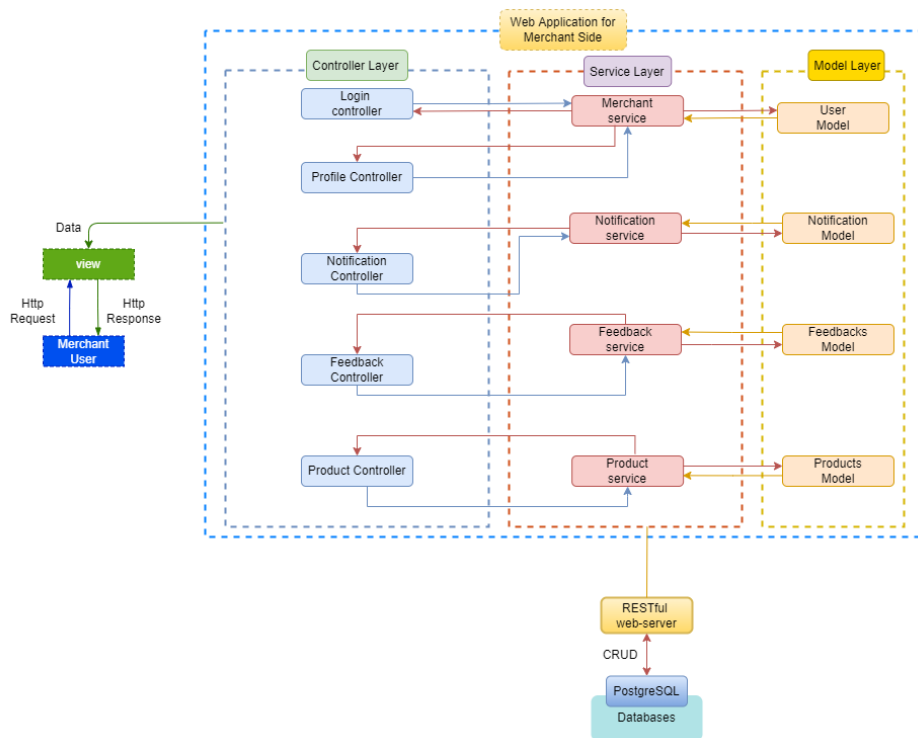


Figure 6.2: Overview the RESTful API request from web application for merchant side.

The figure shown above is regarding the HTTP request and HTTP response flow from client to the backend server. The HTTP request handled by controller and controller will then proceed the request by using model and update to Postgres database in AWS server. For example, login controller will receive request from view, and it will manipulate the data through user model and get data from the database. Once tasks are completed successfully, HTTP response shall be sent back from login controller and render the information to view and send the view result to client.

6.1.2 Controller Layer

In this project, the controller layer is generated automatically when the page is generated through the CLI. The controller itself can handle request from client and will process the data entry then return the data back to client. The controller using model to perform the business logic and declare the association from database to perform the CRUD operation.

6.1.2.1 Login Controller

Customer Side

```
ngOnInit() {
  firebase.auth().onAuthStateChanged(a => { //firebase authentication to identify the last login of current uid
    if (a) {
      swal({ //sweet alert message plugin
        title: 'Loading...',
        text: 'Please wait',
        buttons: [false]
      })
    }
    this.userservice.getuserverify(a.uid).subscribe(a => { //get http POST from user service
      if (a['data']) {
        swal({
          title: 'Login Successful (property) LoginPage.username: string',
          text: 'Welcome!' + this.username,
          icon: 'success',
        })
        console.log('logged in');
        this.nav.navigateForward(this.tab == 'normal' ? 'tabs/tab1' : 'home-senior');
      } else {
        // swal error
        swal({
          title: 'Wrong email or password',
          text: 'Please try again',
          icon: 'warning',
        })
      }
    })
    // this.http.post('http://52.77.247.60/getuserverify', { uid: a.uid }).subscribe(a => {
    //   console.log(a);
    // })
  } else {
    console.log('no log in');
  }
}
```

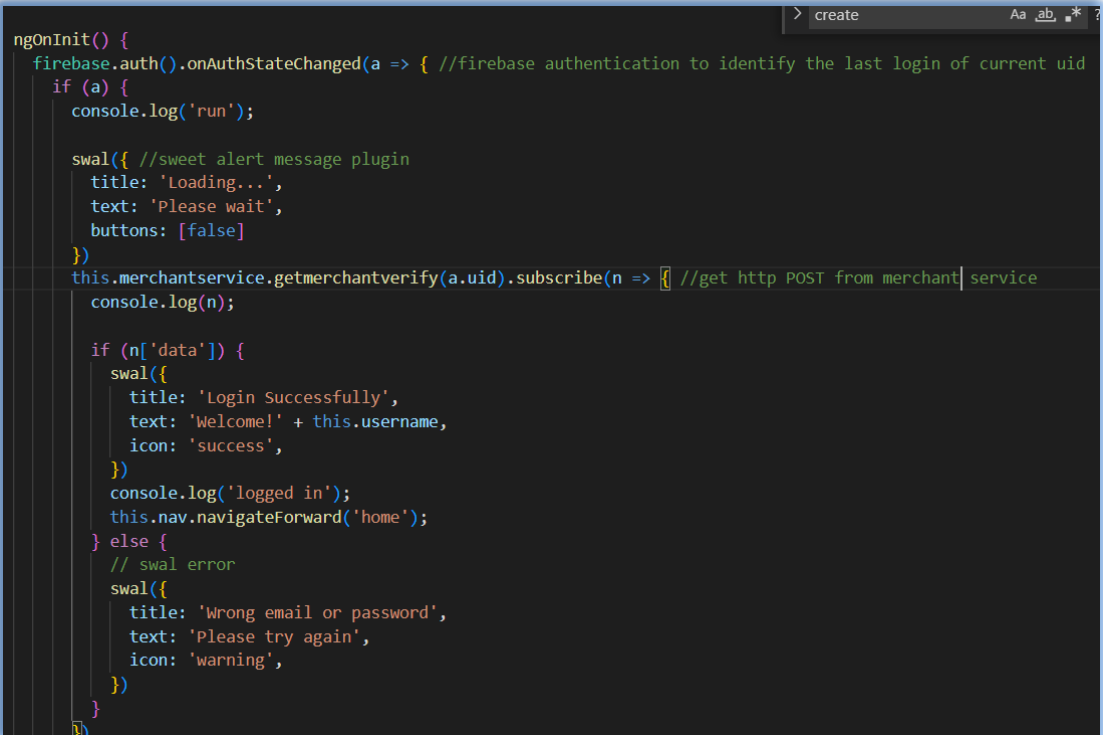
Figure 6.3: Login Controller Authentication for User.

```
login() {
  firebase.auth().signInWithEmailAndPassword(this.email, this.password).then(a => {
    console.log(a);
  }).catch(e => {
    swal({
      title: 'Wrong email or password',
      text: 'Please try again',
      icon: 'warning',
    })
    // this.nav.navigateForward('login')
  })
}
```

Figure 6.4: Login Controller Login Function for User.

In the “Login Controller”, the authentication function will check the user account whether that has been signed in before. After checking the user with recent sign in status, the system will check the interface (Normal or Simple) that user last signed in. The login function will work when user push the login button. The system will check whether user is filling every field, if not, the error dialog will pop up and show the error description to user.

Merchant Side



```
ngOnInit() {
  firebase.auth().onAuthStateChanged(a => { //firebase authentication to identify the last login of current uid
    if (a) {
      console.log('run');

      swal({ //sweet alert message plugin
        title: 'Loading...',
        text: 'Please wait',
        buttons: [false]
      })
    }
    this.merchantservice.getmerchantverify(a.uid).subscribe(n => { //get http POST from merchant service
      console.log(n);

      if (n['data']) {
        swal({
          title: 'Login Successfully',
          text: 'Welcome!' + this.username,
          icon: 'success',
        })
      }
      console.log('logged in');
      this.nav.navigateForward('home');
    } else {
      // swal error
      swal({
        title: 'Wrong email or password',
        text: 'Please try again',
        icon: 'warning',
      })
    }
  })
}
```

Figure 6.5: Login Authentication for Merchant.

In auth controller of merchant side is same with the user authentication. In the initial state, it will call merchant verify API from server side to identify the merchant credential.

6.1.2.2 Register Controller

Customer Side

```
create() {
  if (['name', 'email', 'gender', 'contact', 'address', 'state', 'postcode', 'password'].every(b => this.user[b])) {
    this.user.photo = 'https://cdn-icons-png.flaticon.com/512/847/847969.png';
    swal({
      title: 'Confirmation',
      text: 'Confirm create user?',
      buttons: { Cancel: true, Confirm: true },
      closeOnEsc: false,
      closeOnClickOutside: false,
    }).then(a => {
      if (a == 'Confirm') {
        let temp = {
          name: this.user.name,
          email: this.user.email,
          gender: this.user.gender,
          contact: this.user.contact,
          address: this.user.address,
          state: this.user.state,
          postcode: this.user.postcode,
          password: this.user.password,
          date: new Date().getTime(),
          photo: this.user.photo,
          status: true // boolean
        }
        // insertuser
        this.userservice.insertuser(temp).subscribe(a => {
          swal({
            icon: 'success',
            title: 'Success',
            text: 'Create successfully!',
            buttons: [false],
            timer: 2000,
          })
          this.nav.navigateRoot('login');
        }, e => {
          console.log(e);
          swal({
            icon: 'error',
            title: 'error',
            text: e.message,
            buttons: [false],
            timer: 2000,
          })
        })
      }
    })
  }
}
```

Figure 6.6: Register Controller Source Code.

The figure shown above is “Register Controller” create function. The function will check whether user is filling every field that required on the interface. If all has been filling, the user information will push to database and if not, the error will pop up to show the error description. The user can create their account by push the create button.

Admin Side

```
create() {
  if (['name', 'email', 'contact', 'address', 'state', 'postcode', 'password'].every(b => this.merchant[b])) {
    this.merchant.photo = 'https://cdn-icons-png.flaticon.com/512/847/847969.png';
    swal({
      title: 'Confirmation',
      text: 'Confirm create merchant?',
      buttons: { Cancel: true, Confirm: true },
      closeOnEsc: false,
      closeOnClickOutside: false,
    }).then(a => {
      if (a == 'Confirm') {
        let temp = {
          name: this.merchant.name,
          email: this.merchant.email,
          contact: this.merchant.contact,
          address: this.merchant.address,
          state: this.merchant.state,
          postcode: this.merchant.postcode,
          password: this.merchant.password,
          date: new Date().getTime(),
          photo: JSON.stringify(this.merchant.photo),
          status: true // boolean
        };
        this.service.insertmerchants(temp).subscribe(a => {
          swal({
            icon: 'success',
            title: 'Success',
            text: 'Create successfully!',
            buttons: [false],
            timer: 2000,
          })
          // this.nav.navigateRoot('home');
          this.modalController.dismiss()
        })
      }
    })
  }
}
```

Figure 6.7: Register for merchant account in Admin panel Source Code.

In admin panel, it manages merchant accounts. Merchant account is first created by security administrator side which include all merchant information and created date are recorded in administrative side. Administrative side can decide the merchant account status such as to off or on the merchant account in admin panel.

6.1.2.3 Notification Controller

Customer Side

```
ngOnInit() {
  this.notiservice.getAlltruenotification().subscribe(a => {
    this.notification = a['data'];
    console.log(a['data']);
  })
}
```

Figure 6.8: Notification Controller code segment.

In notification controller, it requests notification api through the notification service from the server side and return back to user interface.

Merchant Side

```
create() {  
  if (['title', 'content'].every(a => this.notification[a])) {  
    swal({  
      title: 'Create notification',  
      text: 'This will create a new notification, are you sure?',  
      icon: 'warning',  
      buttons: ['Cancel', 'Confirm']  
    }).then(ans => {  
      console.log(ans);  
      if (ans) {  
        let temp = {  
          title: this.notification.title,  
          status: true,  
          description: this.notification.content,  
          date: new Date().getTime(),  
          photo: this.notification.photo || 'https://img.freepik.com/free-vector/pus  
          merchant_id: this.merchant_id,  
        } as any;  
        this.notiservice.insertnotification(temp).subscribe(a => {  
          let temp2 = {  
            title: this.notification.title,  
            body: this.notification.content,  
            topic: 'user',  
            path: ''  
          };  
          this.notiservice.FirebaseCloudMessaging(temp2).subscribe(a => {  
            swal({  
              title: 'notification Created',  
              text: 'notification create successfully',  
              icon: 'success',  
              buttons: [false],  
              timer: 1500,  
            })  
            // this.back();  
            this.modalController.dismiss(1);  
          });  
        });  
      }  
    });  
  }  
}
```

Figure 6.9: Create Notification function from merchant side notification controller.

In merchant side notification controller, create function will bind the ngmodel from view to get both title and content input then check if both are filled. If all inputs are filled, the swal message will pop up for create confirmation. “temp” will push through notification service to server-side database and also “temp2” is push to api and user will receive a notification pop up.

6.1.2.4 Cart Controller

Customer Side

```
ngOnInit() {
  this.productservice.getproduct().subscribe(a => {
    console.log(a);
    let temp = (a['data'] || []).reduce((a, b) => ({ ...a, [b.product_id]: b }), {});
    this.products = temp;
    console.log(this.products);
  })
  this.storage.create();
  // this.storage.set('cart', null);
  firebase.auth().onAuthStateChanged(a => {
    if (a) {
      this.user_id = a['uid'];
      this.cart_id = a.uid.substring(0, 4) + new Date().getTime();
      this.actRoute.queryParams.subscribe(() => {
        this.storage.get('cart').then(a => {
          this.cart = a;
          console.log(a);
        })
      })
    }
  })
  // this.service.publishTrigger({ loading: false });
}
}
```

Figure 6.10: Cart Controller code segment.

The above figure shows code segment of cart controller. The “temp” is a formatter of changing array to object. The parent is change to product id. Besides, the cart items are store into local storage. The cart id is generated during each set of orders and cart id is generated using the first four user id with timestamps which to ensure cart id will never repeated.

```

checkout() {
  swal({
    icon: 'warning',
    title: 'Confirmation',
    text: 'Confirm to checkout?',
    buttons: { Cancel: true, Confirm: true }
  }).then(ans => {
    if (ans == 'Confirm') {
      console.log(ans);
      for (let i = 0; i < this.cart.length; i++) {
        let temp = {
          name: this.products[this.cart[i].name].name,
          description: this.products[this.cart[i].name].description,
          price: this.products[this.cart[i].name].price,
          qty: this.cart[i].qty,
          date: new Date().getTime(),
          status: false,
          photo: this.products[this.cart[i].name].photo,
          merchant_id: this.products[this.cart[i].name].merchant_id,
          cart_id: this.cart_id,
          product_id: this.products[this.cart[i].name].product_id,
          user_id: this.user_id,
        }

        this.orderservice.insertorder(temp).subscribe(a => {
          // firebase.database().ref('orders/' + key).update({ id: key });
        });
        // this.storage.set('cart', null);
        this.cart = []

        swal({
          icon: 'success',
          title: 'Success',
          text: 'Checkout Successfully',
        });
      }
    }
  });
}
}

```

Figure 6.11: Cart Controller checkout function source code.

The above figure shows the checkout function in cart page. The sweet alert confirmation message will pop up when user push the checkout button. “Swal” is abbreviation of sweet alert. The sweet alert component such as icon, title, text and buttons will display to user screen while button choice will affect the result. If user is press confirm button, the system will check the length of cart and proceed next step. The http called the order service and insert order into server-side database.

6.1.2.5 Profile Controller

Customer Side

```
signout() {
  swal({
    className: 'swaler',
    icon: 'warning',
    title: 'Confirmation',
    text: 'Confirm to Sign Out?',
    buttons: {
      Confirm: { value: 'Confirm' },
      Cancel: { value: 'Cancel' },
    }
  }).then(ans => {
    if (ans == 'Confirm') {
      console.log(ans);
      firebase.auth().signOut();
      localStorage.removeItem('user');
      this.nav.navigateRoot(['login']);
    }
  })
}
```

Figure 6.12: Profile Controller code segment in user side.

The above figure is code segment of profile controller. In the function, to sign out a user account is basically using firebase authentication backend service which is easy to implement and security.

```
ngOnInit() {
  firebase.auth().onAuthStateChanged(a => {
    if (a) {
      // getuser
      this.userservice.getSpecificUsers(a.uid).subscribe(a => {
        this.user = a['data'];
        console.log(this.user);
      })
    } else {
      this.nav.navigateRoot('login')
    }
  })
}
```

Figure 6.13: Profile Controller code segment in user side.

The above figure shown api call through user service from the server side which to request the specific user data by posting specific user id.

```

update() {
  console.log('update profile');
  swal({
    className: 'swaler',
    icon: 'warning',
    title: 'Confirmation',
    text: 'Confirm to update?',
    buttons: {
      Confirm: { value: 'Confirm' },
      Cancel: { value: 'Cancel' },
    }
  }).then(ans => {
    if (ans == 'Confirm') {
      let temp = {
        user_id: this.user.user_id,
        name: this.user.name,
        email: this.user.email,
        gender: this.user.gender,
        contact: this.user.contact,
        address: this.user.address,
        state: this.user.state,
        postcode: this.user.postcode,
        photo: this.user.photo,
      } as any
      console.log(ans);
      this.userservice.updateSpecificUsers(temp).subscribe(a => { //http request is called from user service
        swal({
          icon: 'success',
          title: 'Updated',
          text: 'Profile update successfully',
          timer: 2000,
          buttons: [false],
        })
      })
    } else {
      swal({
        icon: 'error',
        title: 'Not Saved',
        text: 'Profile did not update',
        timer: 2000,
        buttons: [false],
      })
    }
  })
}

```

Figure 6.14: Update function in edit profile controller in user side.

For profile controller in user side, the controller detects user input and execute the change action. “Ng model” is directive binding view into model and contact with controller. The controller gets the data from view and process the data then called a http from user service to update the data according to the user_id.

Merchant Side

```

this.merchantservice.getmerchants(b.uid).subscribe(a => {
  this.profile = a['data']
  console.log(a['data']);
})

```

Figure 6.15: Edit Profile Controller code segment in merchant side.

The above figure shown api call through merchant user service from the server side which to request the specific merchant user data by posting specific merchant id.

```
update() {
  console.log('update merchant profile');
  swal({
    className: 'swaler',
    icon: 'warning',
    title: 'Confirmation',
    text: 'Confirm to update?',
    buttons: {
      Confirm: { value: 'Confirm' },
      Cancel: { value: 'Cancel' },
    }
  }).then(ans => {
    if (ans == 'Confirm') {
      let temp = {
        merchant_id: this.merchant.merchant_id,
        name: this.merchant.name,
        email: this.merchant.email,
        contact: this.merchant.contact,
        address: this.merchant.address,
        state: this.merchant.state,
        postcode: this.merchant.postcode,
        photo: this.merchant.photo,
      } as any
      console.log(ans);
      this.merchantservice.updatemerchants(temp).subscribe(a => { //h
        swal({
          icon: 'success',
          title: 'Updated',
          text: 'Profile update successfully',
          timer: 2000,
          buttons: [false],
        })
        this.nav.pop();
      });
    }
  });
}
```

Figure 6.16: Edit Profile Controller update function for merchant.

For profile controller in merchant side, the controller detects merchant user input and execute the change action. “Ng model” is directive binding view into model and contact with controller. The controller gets the data from view and process the data then called a http from merchant service to update the data according to the merchant_id.

6.1.2.6 Feedback Controller

Customer Side

```
ngOnInit() {
  this.actRoute.queryParams.subscribe(() => {
    firebase.auth().onAuthStateChanged(a => {
      if (a) {
        this.feedbackservice.getuserfeedback(a).subscribe(a => {
          this.feedbacks = a['data']
          console.log(a['data']);
        });
      }
    })
  })
}
```

Figure 6.17: Feedback Controller code segment for user.

The figure above shows code segment of feedback controller for user side. The firebase authentication will check the status of current user. If it is verified, it will get http from feedback service to call the server side which to display feedback details according to feedback id.

```
feedbackcreate(x) {
  console.log(this.feedback);
  if ([ 'title', 'type', 'description' ].every(a => this.feedback[a]) && this.merchant_id && this.user.user_id) {
    swal({
      title: 'Create Feedback',
      text: 'This will create a new feedback, are you sure?',
      icon: 'warning',
      buttons: {
        Confirm: { value: 'Confirm' },
        Cancel: { value: 'Cancel' },
      }
    }).then(ans => {
      console.log(ans);
      if (ans == 'Confirm') {
        let temp = {
          title: this.feedback.title,
          status: 'true',
          description: this.feedback.description,
          photo: JSON.stringify(this.photo),
          date: new Date().getTime(),
          merchant_id: this.merchant_id,
          user_id: this.user.user_id,
          type: this.feedback.type,
        } as any;
        console.log(temp);
        this.feedbackservice.insertfeedback(temp).subscribe(a => { //http called from feedback service
          swal({
            title: 'Feedback Sent',
            text: 'Feedback sent successfully',
            icon: 'success',
            buttons: [false],
            timer: 1500,
          })
          this.nav.back();
        })
      } else {
        false
      }
    })
  }
}
```

Figure 6.18: Feedback Controller create function in user side.

The figure above shows code segment of feedback create controller. The “feedbackcreate” function will check the view whether it given all information that is necessary to fill in or it will return error message to view. The merchant id is according to the selection of user. After all is verified, it will call http from feedback service and push the data into database.

Merchant Side

```
this.feedbackservice.getallmerchantfeedback(b.uid).subscribe(a=>{
  this.feedbacks = a['data']
  this.select = a['data'][0]?.feedback_id
  console.log(a['data']);
})
```

Figure 6.19: Feedback Controller code segment for merchant user.

The figure above shows code segment of feedback controller for merchant side. The firebase authentication will check the status of current merchant user. If it is verified, it will get http from feedback service to call the server side which to display feedback details according to feedback id.

Merchant Side

```
ngOnInit() {
  firebase.auth().onAuthStateChanged(a => {
    if (a) {
      this.actRoute.queryParams.subscribe(b => {
        this.id = b['id']
        this.feedbackdata()
      })
    }
  })
}
```

Figure 6.20: Feedback Details Controller code segment for merchant user.

In this controller, to get the feedback details, it need get the feedback id before return the feedback details to user.

```

feedbackdata() {
  this.feedbackservice.getfeedback(this.id ).subscribe(a => {
    this.feedback = a['data'][0]
    this.photo = a['data'][0].photo
    console.log(this.feedback);
  });
}

```

Figure 6.21: Feedback Details Controller function for merchant user.

The above figure shown the feedback data function called after the feedback id is identified.

```

async presentModal(x) {
  const modal = await this.modalController.create({
    component: FeedbackReplyPage,
    componentProps: { id: x },
    cssClass: 'feedback'
  });

  await modal.present();
  const { data } = await modal.onDidDismiss();
  if (data) {
    this.feedbackdata()
  }
}

```

Figure 6.22: Feedback Details Controller function for merchant user.

The above figure shown modal controller to feedback reply page. The modal is called when the feedback id is passed.

6.1.2.7 Order Controller

Customer Side

```

ngOnInit() {
  firebase.auth().onAuthStateChanged(a => {
    if (a) {
      this.userservice.getSpecificUsers(a.uid).subscribe(a => {
        this.user = a['data']
      })
      this.orderservice.getalluserorders(a.uid).subscribe(a => {
        this.orders = a['data']
      })
    } else {
      this.nav.navigateRoot('login')
    }
  })
}

```

Figure 6.23: Order History Controller code segment for user.

The figure above shows code segment of order history controller. As mentioned above, the firebase authentication onAuthStateChanged function will check user account status. Both user service and order service are called from the controller which to post http to the view.

6.1.2.8 Display Store Controller

Customer Side

```
carter() {  
  swal({  
    icon: 'warning',  
    title: 'Confirmation',  
    text: 'Confirm add to cart?',  
    buttons: {  
      Cancel: true,  
      Confirm: true,  
    }  
  }).then(ans => {  
    if (ans == 'Confirm') {  
      this.cart.push({ name: this.product.product_id, qty: this.qty });  
      this.storage.set('cart', this.cart);  
      swal({  
        icon: 'success',  
        title: 'Added',  
        text: 'Add to cart successfully',  
        timer: 2000,  
        buttons: [false],  
      })  
      this.goback();  
    }  
  });  
}
```

Figure 6.24: Display Store Controller function.

The figure above shown the display store controller function. Once the carter function is called, the sequence will go with “swal” function which rendering confirmation message to user, when the system gets the response, it will proceed the responded action. If confirms, it will push current product into local storage and display into cart page.

6.1.2.9 Favourite Controller

Customer Side

```
ngOnInit() {
  this.storage.create();
  this.storage.get('cart').then(a => {
    this.cart = a || []
    console.log(this.cart);
  })
  this.actRoute.queryParams.subscribe(a => {
    this.productservice.getproducts({ product_id: a['id'] }).subscribe(a => {
      this.product = a['data']
      console.log(a);
    })
  })
  firebase.auth().onAuthStateChanged(b => {
    if (b) {
      this.userservice.getSpecificUsers({ uid: b['uid'] }).subscribe(c => {
        this.user = c['data']
        console.log(this.user);
        this.user.favorite = c['data']?.favorite || []
        this.favor = (c['data']?.favorite || []).findIndex(d => d == a.id) > -1
        console.log(this.user);
      })
    }
  })
}
```

Figure 6.25: Display Store Controller code segment.

The figure above shown code segment of display store controller. The storage named “cart” is called whenever the page is entered. Besides, it also called the products by product id. Lastly, there are also code segment of returning user data by verifying their id and display the favorite items according to user id.

```
favorite() {
  let temp = !this.favor ? (this.user.favorite.push(this.product.prodcut_id)) :
    this.user.favorite.splice(this.user?.favorite.findIndex(a => a['product_id'] == this.product.product_id), 1);
  this.userservice.updateUser({uid: this.user.uid, favorite: JSON.stringify(temp)
  }).subscribe(a => {
    this.favor = !this.favor;
    swal({
      title: 'Success',
      icon: 'success',
      text: (!this.favor ? 'Remove from' : 'Added into') + ' favorite',
      buttons: [false]
    })
  })
}
```

Figure 6.26: Display Store Controller favorite function source code.

The figure above shown the favorite function source code. The favorite action is depending on the product id to perform add favorite and remove from favorite. The

default display is unfavorite from product. If user favorite the product, dialog message will pop up and display success message.

6.1.2.10 Tabs Controller

```
ngOnInit(): void {  
    firebase.auth().onAuthStateChanged(a => {  
        if (a) {  
        } else {  
            this.nav.navigateRoot('login', { animationDirection: 'back' });  
        }  
    })  
    this.service.getTrigger().subscribe(a => {  
        console.log(a);  
        if(a){  
            // this.loading = a['loading'];  
        }  
    })  
}
```

Figure 6.27: Tabs Controller code segment.

In this controller, a loading function is injected from service component which the loading animation will pop up before the page is ready.

```
getSelectedTab(): void {  
    this.nowtab = this.tabs.getSelected()  
    this.nowtab2 = this.tabs.getSelected()  
    this.nowtab3 = this.tabs.getSelected()  
    this.nowtab4 = this.tabs.getSelected()  
    console.log(this.tabs.getSelected());  
}
```

Figure 6.28: Tabs Controller function.

The “Tabs Controller” will handle the tabs function (home tab, notification tab, cart tab and profile tab).

```

scan() {
  this.options = {
    preferFrontCamera: false,
    showFlipCameraButton: true,
    showTorchButton: true,
    torchOn: false,
    prompt: 'Place a barcode inside the scan area',
    resultDisplayDuration: 500,
    formats: 'EAN_13,EAN_8,QR_CODE,PDF_417 ',
    orientation: 'portrait',
  };
  this.barcodeScanner.scan(this.options).then(barcodeData => {
    this.nav.navigateForward('display-store?id=' + barcodeData.text + '&scan=true')
  }).catch(err => {
    this.nav.navigateForward('display-store?id=1&scan=true')
    console.log('Error', err);
  });
}
}

```

Figure 6.29: Tabs Controller Source Code.

The figure above shown the scan function from tabs controller. The scanner is ionic plugin, it can customize its settings with options. The scanner will detect QR code and display the relevant data and pass product id to “display store” page.

6.1.2.11 News Controller

Customer Side

```

ngOnInit() {
  this.actRoute.queryParams.subscribe(a => {
    this.http.post('http://52.77.247.60/getnews2', { news_id: a['id'] }).subscribe(b => {
      console.log(b['data']);
      this.news = b['data'][0]
    })
  })
}
}

```

Figure 6.30: Read news code segment in customer side.

The figure above shown read news code segment in customer side. The news http is called when the page is loaded.

Admin Side

```
create() {
  if (['title', 'description', 'photo'].every(b => this.news[b])) {
    swal({
      title: 'Confirmation',
      text: 'Confirm create merchant?',
      buttons: { Cancel: true, Confirm: true },
      closeOnEsc: false,
      closeOnClickOutside: false,
    }).then(a => {
      if (a == 'Confirm') {
        let temp = {
          title: this.news.title,
          description: this.news.description,
          date: new Date().getTime(),
          photo: JSON.stringify(this.news.photo),
          status: true // boolean
        }

        this.service.insertnews(temp).subscribe(a => {
          swal({
            icon: 'success',
            title: 'Success',
            text: 'Create successfully!',
            buttons: [false],
            timer: 2000,
          })
          // this.nav.navigateRoot('home');
          this.modalController.dismiss()
        }, e => {
          console.log(e);
          swal({
            icon: 'error',
            title: 'error',
            text: e.message,
            buttons: [false],
            timer: 2000,
          })
        })
      }
    })
  }
}
```

Figure 6.31: Create news function code segment in admin side.

The figure above shown the create news function in admin side. The admin required to enter title, description and image to create news for customer side. If successful, the dialog message will pop up and display success message.

6.1.3 Service Layer

During the whole code implementation, the service layer is one of the important layers which to carry api client that provides root url to controller. To get http response from the remote server, the module needs to import HttpClient provider to make sure the service is work. After injecting HttpClient, the communication between remote server and services layer is connected and are able to send Http Method such as POST, GET, PUT and DELETE requests. In this project, it only uses two http method which is POST and GET method and details is discussed as below.

6.1.3.1 User Service

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }
  user = [] as any

  getSpecificUsers(x): Observable<User[]> {
    return this.http.post<User[]>(`${this.ROOT_URL}/getuser`, { uid: x })
      .pipe(catchError(this.handleError<User[]>('getuser', [])))
  }
  updateSpecificUsers(temp): Observable<User[]> {
    return this.http.post<User[]>(`${this.ROOT_URL}/updateuser2`, temp)
      .pipe(catchError(this.handleError<User[]>('updateuser2', [])))
  }
  updateUser(temp): Observable<User[]> {
    return this.http.post<User[]>(`${this.ROOT_URL}/updateuser`,
      [{ uid: this.user.uid, favorite: JSON.stringify(temp) }])
      .pipe(catchError(this.handleError<User[]>('updateuser', [])))
  }

  getuserverify(x): Observable<User[]> {
    return this.http.post<User[]>(`${this.ROOT_URL}/getuserverify`, { uid: x })
      .pipe(catchError(this.handleError<User[]>('getuserverify', [])))
  }

  insertuser(temp): Observable<User[]> {
    return this.http.post<User[]>(`${this.ROOT_URL}/insertuser`, temp)
      .pipe(catchError(this.handleError<User[]>('insertuser', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);

      return of(result as T);
    };
  }
}
```

Figure 6.32: User service source code.

As figure shown above, one of the examples of user service is contained user http client which include getuser, updateuser, getuserverify and insertuser. There are

pipeable operators which is error catcher that is like mapping operators. If the http request is failed, the error object will catch and retrieve the source observable. “User []” is model which exported from interface models.

6.1.3.2 Feedback Service

```
@Injectable({
  providedIn: 'root'
})
export class FeedbackService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }

  insertfeedback(temp): Observable<Feedback[]> {
    return this.http.post<Feedback[]>(`${this.ROOT_URL}/insertfeedbacks`, temp)
      .pipe(catchError(this.handleError<Feedback[]>('insertfeedbacks', [])))
  }

  getfeedback(id): Observable<Feedback[]> {
    return this.http.post<Feedback[]>(`${this.ROOT_URL}/getfeedback`, {feedback_id: id})
      .pipe(catchError(this.handleError<Feedback[]>('insertfeedbacks', [])))
  }

  getuserfeedback(a): Observable<Feedback[]> {
    return this.http.post<Feedback[]>(`${this.ROOT_URL}/getuserfeedbacks`, {uid: a.uid })
      .pipe(catchError(this.handleError<Feedback[]>('getuserfeedbacks', [])))
  }

  deletefeedback(id): Observable<Feedback[]> {
    return this.http.delete<Feedback[]>(`${this.ROOT_URL}/deletefeedback?id=${id}`)
      .pipe(catchError(this.handleError<Feedback[]>('deletefeedback', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Figure 6.33: Feedback service source code.

6.1.3.3 Order Service

```
@Injectable({
  providedIn: 'root'
})
export class OrderService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }

  getorder2(x): Observable<Order[]> {
    return this.http.post<Order[]>(`${this.ROOT_URL}/getarrayorders`, {uid: x })
      .pipe(catchError(this.handleError<Order[]>('getarrayorders', [])))
  }

  getalluserorders(uid): Observable<Order[]> {
    return this.http.post<Order[]>(`${this.ROOT_URL}/getalluserorders`, {user_id: uid })
      .pipe(catchError(this.handleError<Order[]>('getalluserorders', [])))
  }

  insertorder(temp): Observable<Order[]> {
    return this.http.post<Order[]>(`${this.ROOT_URL}/insertorders`, temp)
      .pipe(catchError(this.handleError<Order[]>('insertorders', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Figure 6.34: Order service source code.

6.1.3.4 Product Service

```
@Injectable({
  providedIn: 'root'
})
export class ProductService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }

  getalltrueproduct(): Observable<Product[]> {
    return this.http.get<Product[]>(`${this.ROOT_URL}/getalltrueproducts`)
      .pipe(catchError(this.handleError<Product[]>('getarrayorders', [])))
  }

  getproducts(x): Observable<Product[]> {
    return this.http.post<Product[]>(`${this.ROOT_URL}/getproducts`, { product_id: x.product_id })
      .pipe(catchError(this.handleError<Product[]>('getarrayorders', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Figure 6.35: Product service source code.

6.1.3.5 Notification Service

```
export class NotificationService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }

  getalltruenotification(): Observable<Notification[]> {
    return this.http.get<Notification[]>(`${this.ROOT_URL}/getalltruenotification`)
      .pipe(catchError(this.handleError<Notification[]>('getalltruenotification', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Figure 6.36: Notification service source code.

6.1.3.6 News Service

```
@Injectable({
  providedIn: 'root'
})
export class NewsService {
  public ROOT_URL = 'http://52.77.247.60';
  constructor(private http: HttpClient) { }

  getnews(): Observable<News[]> {
    return this.http.get<News[]>(`${this.ROOT_URL}/getnews`)
      .pipe(catchError(this.handleError<News[]>('getnews', [])))
  }

  private handleError<T>(operation = 'operation', result?: T) {
    return (error: any): Observable<T> => {
      console.error(`${operation} failed: ${error.message}`);
      return of(result as T);
    };
  }
}
```

Figure 6.37: News service source code.

6.1.4 Model Layer

In this project, all of the user's data-related logic is represented by the Model component. The data that related to a business logic or data is required from controller to pass to view is manipulated by model component. Model also can interact with database server to retrieve data or update data into database storage. Besides, model also process data into data structure such as deserialization and serialization.

6.1.4.1 User Model

For example, the interface user is a model which all variable is declare with specific type for user service component which then return to controller component. To declare the type of each role and return to controller respectively instead of declare using "as any". Each model is exported, and it can be imported into service component when define http client method.

```
export interface User {  
  user_id: number;  
  uid: string;  
  name: string;  
  status: boolean;  
  longitude: Float64Array;  
  latitude: Float64Array;  
  address: string;  
  state: string;  
  postcode: number;  
  dob: number;  
  ggender: string;  
  description: string;  
}
```

Figure 6.38: User model source code.

The below is models source code in this project which include order, feedback, product, notification and news.

6.1.4.2 Order Model

```
export interface Order {  
  order_id: number;  
  product_id: number;  
  user_id: string;  
  cart_id: string;  
  date: number;  
  price: number;  
  qty: number;  
  photo: JSON;  
  status: boolean;  
  merchant_id: string;  
  name: string;  
  description: string;  
}
```

Figure 6.39: Order model source code.

6.1.4.3 Feedback Model

```
export interface Feedback {  
  feedback_id: number;  
  merchant_id: number;  
  user_id: number;  
  title: string;  
  status: boolean;  
  name: string;  
  photo: JSON;  
  respond: string;  
  respond_date: BigInt;  
  respond_photo: JSON;  
  respond_title: Text;  
  date: BigInt;  
  type: string;  
  description: string;  
}
```

Figure 6.40: Feedback model source code.

6.1.4.4 Product Model

```
export interface Product {  
  product_id: number;  
  merchant_id: number;  
  name: string;  
  photo: JSON;  
  price: number;  
  date: BigInt;  
  description: string;  
}
```

Figure 6.41: Product model source code.

6.1.4.5 Notification Model

```
export interface Notification {  
  notification_id: number;  
  merchant_id: number;  
  user_id: number;  
  title: string;  
  status: boolean;  
  photo: JSON;  
  date: BigInt;  
  description: string;  
}
```

Figure 6.42: Notification model source code.

6.1.4.6 News Model

```
export interface News {  
  news_id: number;  
  title: string;  
  status: boolean;  
  photo: JSON;  
  date: BigInt;  
  description: string;  
}
```

Figure 6.43: News model source code.

6.1.5 Other Integration

There are two other integrations be integrated to the backend system which to reach more advancing and functionality in this system. The integrations are Bill Plz and Firebase Cloud Messaging.

6.1.5.1 Billplz

Billplz is a payment checkout platform that allow customer to perform checkout function and do transaction through online. For example, customer can check out their orders in the cart and pay through online transaction in mobile application.

```
//initialize the admin credential form firebase console admin sdk
//Sand Box tester
var billplzautho = "Basic YzNkZTMwMTUtODAxMy00M2U5LTNmYTEtMDM0MjFmZDU0YzBi"
var billplzlink = 'https://www.billplz.com/api/v3/bills'
var billcollid = "s8lziwfr"

app.post('/billplz', (req, res) => {
  var options = {
    method: 'POST',
    url: billplzlink,
    qs:
    {
      collection_id: billcollid,
      description: req.body.description,
      email: req.body.email,
      name: req.body.name,
      amount: req.body.amount, //100--RM1.00
      callback_url: 'http://52.77.247.60/callback?order_id=' + req.body.order_id,
      redirect_url: 'https://supermarket2.page.link/6RQi'
    },
    headers:
    {
      Authorization: billplzautho
    }
  };
```

Figure 6.44: billplz api endpoints at server side.

```
this.http.post('http://52.77.247.60/billplz', caller).subscribe(a => {
  console.log(a);
  window.open(a['url']);

  firebase.database().ref('orders/' + keyer).on('value', b => {
    if (b.exists()) {
      console.log('in progress');
    } else {
      firebase.database().ref('orders/' + keyer).off();
      this.storage.set('cart', null);
      this.cart = []
      swal({
        icon: 'success',
        title: 'Success',
        text: 'Checkout Successfully',
      });
    }
  })
})
```

Figure 6.45: billplz api code segment call from server side.

Figure above shown billplz api code segment call from server side. The code segment above is resided on checkout function which the billplz api will call once user checkout the items in cart. “window.open(a[‘url’])”, the “url” represent the billplz link that stated in server side that shown in figure 6.54.

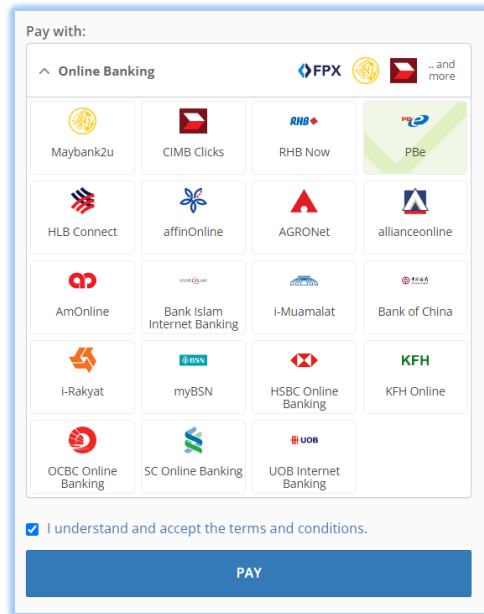


Figure 6.46: billplz payment gateway.

Figure above shown is an example of billplz payment gateway after checkout the cart items. The payment method is allowed online payment with various of bank.

6.1.5.2 Firebase Cloud Messaging

Firebase Cloud Message is a notification platform that be used in this project to send real time notification to all customer mobile application. The customer is able to see updates announcement, feedback respond, news etc from merchant site.

```
constructor(private fcm: FCM, private nav: NavController, private storage: Storage,
private toastController: ToastController, private platform: Platform) {}

firebase.initializeApp(firebaseConfig);

firebase.auth().onAuthStateChanged(a => {
  if (a) {
    this.storage.create();
    this.storage.get('notification').then(b => {
      this.notification = b !== null ? b : this.notification;

      console.log(this.notification);

      if (this.platform.is('android') && this.notification) {
        console.log(a.uid);
        this.fcm.subscribeToTopic(a.uid);
        this.fcm.subscribeToTopic('user');
      }
    });
  }
});
```

Figure 6.47: Firebase cloud messaging code segment in app.component.ts.

The figure above shown Firebase cloud messaging code segment in appcomponent. At initial stage, it will detect whether the notification is on, in the user settings, this lead to determine the notification will enter into individual or not. Besides, notification can only be received when user is using android devices, as the platform is declared to “android” only. This.fcm.subscribeToTopic is allowed this device to retrieve notification from specific topic.

```
fcmNotification() {
  this.fcm.onNotification().subscribe(async data => {
    if (data.wasTapped) {
      setTimeout(() => {
        this.nav.navigateForward(data.path);
      }, 2000);
    }
    else {
      this.presentToastWithOptions(data.title, data.message, data.id, data.path)
    }
  })
}
```

Figure 6.48: Firebase cloud messaging fcm function in app.component.ts.

As the figure above shown, once the notification is sent to a device, the notification will pop up to phone. If the notification is tapped, it will display the notification details on the application. When the notification is notified but not been tapped, it will be appeared until it be swapped away by user. The notification will display title and description on the phone.

```

this.notiservice.insertnotification(temp).subscribe(a => {
  let temp2 = {
    title: this.notification.title,
    body: this.notification.content,
    topic: 'user',
    path: ''
  };
  this.notiservice.FirebaseCloudMessaging(temp2).subscribe(a => {
    swal({
      title: 'notification Created',
      text: 'notification create successfully',
      icon: 'success',
      buttons: [false],
      timer: 1500,
    })
    // this.back();
    this.modalController.dismiss(1);
  });
});

```

Figure 6.49: Notification send to topic 'user' when merchant created a notification.

The figure above shown the notification send function's code segment when merchant created a notification. The notification details will go to a topic and push to all user once merchant created a notification on merchant dashboard.

6.1.5.3 Amazon S3 Storage Service

The abbreviation of Amazon S3 Storage Service is AWS S3 which is integrated into the backend system. It provides the service to store user data such as video, images etc. For example, the QR code that generated once merchant created a product will be store at AWS bucket.

```

const aws = require('aws-sdk')
const region = 'ap-southeast-1'
const bucketName = 'supermarket/'
const accessKeyId = "AKIA2EAJLXN0NAFX2D3Q"
const secretAccessKey = "0U15X0ooyeef2W88i0tx3V2FX0wfmKSkUvDAU5xc"

const s3 = new aws.S3({
  region,
  accessKeyId,
  secretAccessKey,
  signatureVersion: 'v4'
})

function uploadFile(imageData, folder, userid) {
  try {
    const base64Data = new Buffer.from(imageData.replace(/^data:image\/\w+;base64/, ''), 'base64')
    const type = imageData.split(';')[0].split('/')[1]
    const randomKey = Date.now().toString(36).substring(0, 5) + Math.random().toString(36).substr(2).substring(0, 4)

    const uploadParams = {
      Bucket: bucketName,
      Body: base64Data,
      //ACL: 'public-read',
      ContentEncoding: 'base64',
      ContentType: `image/${type}`,
      Key: `${folder}/${userid}${randomKey}.${type}`
    }
    return s3.upload(uploadParams).promise()
  } catch {
    return
  }
}

module.exports = { uploadFile };

```

Figure 6.50: AWS S3 source code on backend.

The figure above shown the AWS S3 upload image source code which resided on server side. The region, bucket name, API key and secret access key will generate once the service has created. All above is defined as stated above and upload function is written for user or merchant user to upload image into S3 bucket.

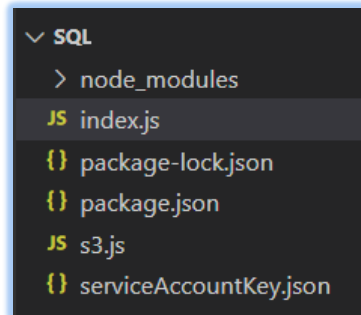


Figure 6.51: s3.js on backend.

```
// users //
app.post('/upload', async (req, res) => {
  console.log('upload');
  let imageData = req.body.image
  let folder = req.body.folder
  let userid = req.body.userid
  const result = await uploadFile(imageData, folder, userid)

  if (result) {
    res.status(200).send({ imageURL: result ? result.Location : "" })
  } else {
    res.status(400).send({ message: "Wrong Base 64 Format/ Wrong File" })
  }
})
```

Figure 6.52: Upload file API endpoint on backend.

The figure above shown the upload file API endpoint resided at server side. The upload module is exported from s3.js and imported into the index.js (API endpoint resided), the upload file function from s3.js is called to define the upload file API for client.

```

fileChange(event, maxsize) {
  if (event.target.files && event.target.files[0] && event.target.files[0].size < (10485768)) {
    const can = document.createElement('canvas');
    const ctx = can.getContext('2d');
    const thisImage = new Image;
    const maxW = maxsize;
    const maxH = maxsize;
    thisImage.onload = (a) => {
      const iw = thisImage.width;
      const ih = thisImage.height;
      const scale = Math.min((maxW / iw), (maxH / ih));
      const iwScaled = iw * scale;
      const ihScaled = ih * scale;
      can.width = iwScaled;
      can.height = ihScaled;
      ctx.save();
      ctx.drawImage(thisImage, 0, 0, iwScaled, ihScaled);
      ctx.restore();
      this.imagec = can.toDataURL();
      this.user.photo = 'https://i.pinimg.com/originals/a2/dc/96/a2dc9668f2cf170fe3efeb263128b0e7.gif';
      this.http.post('http://52.77.247.60/upload', { image: this.imagec, folder: 'market', userid: 'supermarket' }).subscribe((link) => {
        console.log(link['imageURL']);
        this.user.photo = link['imageURL'];
      });
    };
    thisImage.src = URL.createObjectURL(event.target.files[0]);
  } else {
    swal.close();
    alert('Your Current Image Too Large, ' + event.target.files[0].size / (10241024) + 'MB! (Please choose file lesser than 8MB)');
  }
}

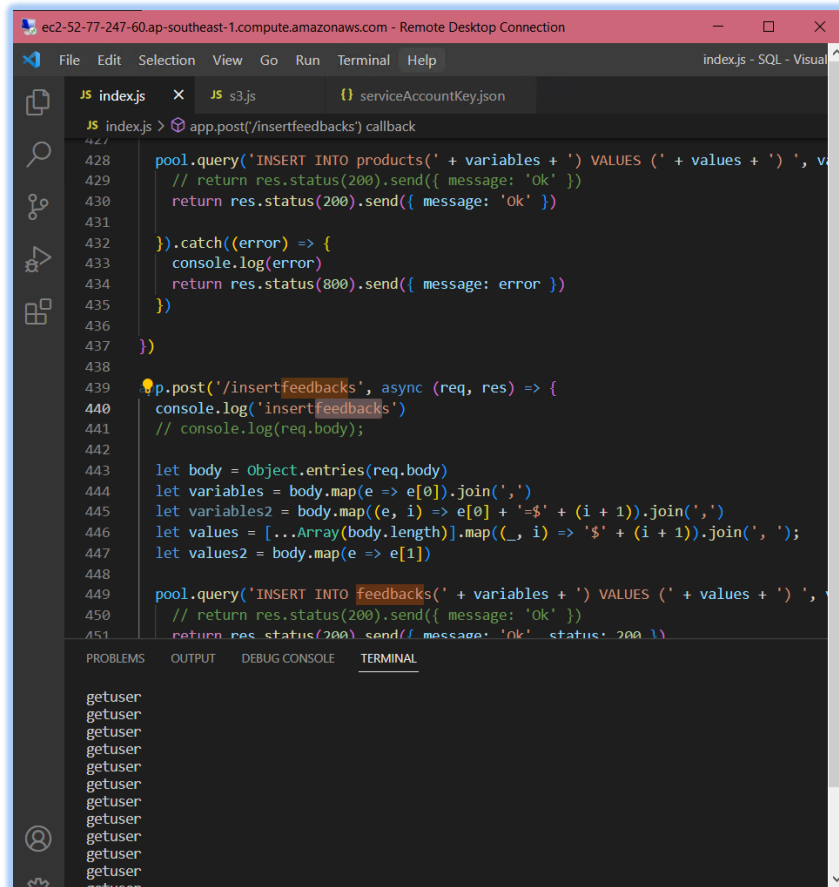
```

Figure 6.53: Edit profile's upload image code segment.

The figure above shown the upload image function resided in edit profile. The file change function (upload image) will open the local image gallery and allow user to upload image which image size is lower than 8MB to S3 bucket. The image file is JSON format and it post to http client and store into S3 bucket database.

6.1.6 Available Endpoints

API endpoints is the communication point that led two applications to communication and connect to each other. API is abbreviation of application program interface which will work when sending a request to grab specific information from a web server and if is successful, it will be receiving a success response. There are total of 24 API endpoint that in this project.



```
ec2-52-77-247-60.ap-southeast-1.compute.amazonaws.com - Remote Desktop Connection
indexjs - SQL - Visual Studio Code
File Edit Selection View Go Run Terminal Help
JS indexjs x JS s3.js {} serviceAccountKeyjson
JS indexjs > app.post('/insertfeedbacks') callback
428 pool.query('INSERT INTO products(' + variables + ') VALUES (' + values + ')', v
429 // return res.status(200).send({ message: 'Ok' })
430 return res.status(200).send({ message: 'Ok' })
431
432 }).catch((error) => {
433 console.log(error)
434 return res.status(800).send({ message: error })
435 })
436
437 })
438
439 p.post('/insertfeedbacks', async (req, res) => {
440 console.log('insertfeedbacks')
441 // console.log(req.body);
442
443 let body = Object.entries(req.body)
444 let variables = body.map(e => e[0]).join(',')
445 let variables2 = body.map((e, i) => e[0] + '=' + (i + 1)).join(',')
446 let values = [...Array(body.length)].map((_, i) => '$' + (i + 1)).join(', ');
447 let values2 = body.map(e => e[1])
448
449 pool.query('INSERT INTO feedbacks(' + variables + ') VALUES (' + values + ')', v
450 // return res.status(200).send({ message: 'Ok' })
451 return res.status(200).send({ message: 'Ok', status: 200 })
452
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
getuser
```

Figure 6.54: AWS EC2 server.

6.1.6.1 Mobile Endpoints

Mobile endpoints are written in a file of remote server which called “index js”, the server that is subscribed plan with Amazon EC2 server and region in southeast. There is total of 12 API endpoint in mobile side which to provide a point that mobile side user could request the information that perform CRUD operation and retrieve information from PostgreSQL database.

Table 6.1 Mobile Endpoints Listing

User Endpoint			
No	Method	Route	Descriptions
1	POST	upload	To upload image
2	POST	‘/getuser, uid’	To get specific user data with user uid
3	POST	‘/insertuser’	Register process for user
4	POST	‘/updateuser, uid’	To update specific user profile information with user uid
5	POST	‘/changepassword, uid’	Update changed password with user uid
6	POST	‘/getuserverify, uid’	Verified the user with user uid
7	POST	‘getalltruemerchant, status’	Retrieve all merchant with status is true
Feedback Endpoint			
No	Method	Route	Descriptions
8	POST	‘/getfeedback, feedback_id’	Display specific feedback
9	POST	‘/getuserfeedbacks, user_id’	Display user feedbacks with user id
10	POST	‘/insertfeedback’	Create new feedback
11	DELETE	‘/deletefeedback, feedback_id’	Delete specific feedback with feedback id
Order Endpoint			
No	Method	Route	Descriptions
12	POST	‘/getorders, order id’	Retrieve specific order history with order_id
13	POST	‘/getalluserorders, user_id’	Retrieve all user’s users with user_ids
14	POST	‘/insertorders’	User checkout product
Product Endpoint			
No	Method	Route	Descriptions
15	POST	‘/getproducts, product id’	Retrieve specific product with

			product id after scan the QR code
16	GET	‘/getalltrueproduct’	Retrieve all product with status is true
Notification Endpoint			
No	Method	Route	Descriptions
17	POST	‘getnotification, notification_id’	Retrieve specific notification with notification id
18	GET	‘/getalltruenotification’	Retrieve all notification with status is true
News Endpoint			
No	Method	Route	Descriptions
19	GET	‘/getnews’	Retrieve news

6.1.6.2 Merchant Endpoints

Merchant endpoints are written in a file of remote server which called “index js”, the server that is subscribed plan with Amazon EC2 server and region in southeast. There is total of 12 API endpoint in merchant side which to provide a point that merchant dashboard user could request the information that perform CRUD operation and retrieve information from PostgreSQL database.

Table 6.2 Merchant Endpoints Listing

Merchant Endpoint			
No	Method	Route	Descriptions
1	POST	upload	To upload image
2	POST	‘getmerchant, uid’	To get specific merchant data
3	POST	‘updatemerchants, merchant_id’	To update specific merchant profile information with merchant id
4	POST	‘getmerchantverify, uid’	Verify merchant with merchant uid
Feedback Endpoint			
No	Method	Route	Descriptions
5	POST	‘getfeedback, feedback_id’	Display specific feedback with feedback id
6	POST	‘/getallvendorfeedback, merchant_id’	Retrieve all merchant’s feedbacks with merchant id
7	POST	‘insertfeedbackrespond, feedback id’	Respond user feedback according feedback id
Product Endpoint			
No	Method	Route	Descriptions
8	POST	‘insertproducts’	Create new product for user mobile side
9	POST	‘updateproduct, product id’	Update specific product with product id
10	POST	‘getproducts, product id’	Retrieve specific product after scan the QR code
11	POST	‘getalltrueproduct, product id’	Retrieve product in cart

12	DELETE	'deleteproduct, product_id'	Delete specific product with product id
Notification Endpoint			
No	Method	Route	Descriptions
13	GET	'getnotification, notification_id'	Retrieve specific notification with notification id
14	POST	'Insertnotification'	Create new notification
15	POST	'getmerchantnotification, merchant_id'	Retrieve all merchant's notification with merchant id
16	DELETE	'deletenotification, notification_id'	Delete specific notification with notification id

6.2 Mobile application for supermarket self-checkout

In this project, mobile applications for supermarket self-checkout are implemented with angular js, it will become a customer user interface to checkout supermarket items. The main function of mobile application is for customer to scan item with QR code, add to cart and checkout efficiently. Angular js is taking part of front-end client within overall system.

6.2.1 Overview of Mobile Application

Building the mobile application is using Ionic framework with angular js as front-end client. The screen is separated into two part which is main screen and navigation tabs which stick at bottom. The main tabs are home tab which to display profile name, main features, news slider and also order history. Other tabs such as notification, cart and profile will discuss in this section. Besides, senior user interface will not contain navigation tabs as it mainly simple screen layout and only given main function. The simple mode can be switch through login screen, which to switch into simple mode and perform login, or can change in normal mode settings if user wish to use simple mode to perform checkout.

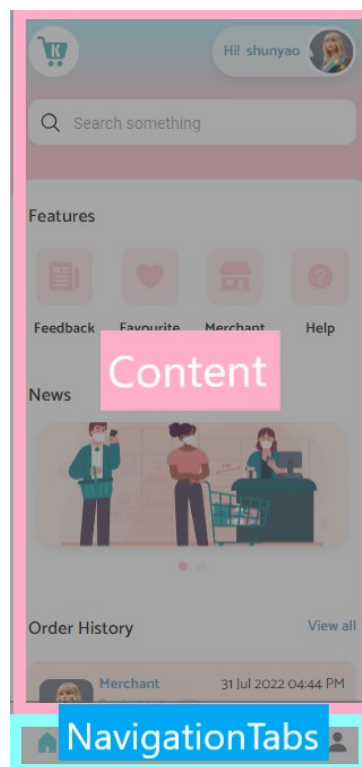


Figure 6.55: Mobile application normal mode screen layout.

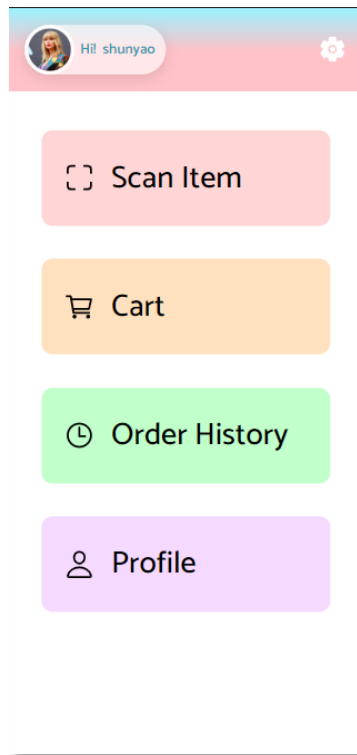


Figure 6.56: Mobile application simple mode screen layout.

There are two different screens for supermarket user as two figures shown above. Both home page will display only if the account is logged in or the screen will prompt to login screen. This action is to ensure that user having their credential and valid logging. The authentication will detect user verify status and last login interface. If the user registers the date of birth that is above 60 years old, the login screen will lead to simple mode and recommended for user to login with simple mode. If user wish to switch back to normal mode, there also can be changed in settings.

```
firebase.auth().onAuthStateChanged(a => { //firebase authentication to identify the last login of current uid
  if (a) {
    swal({ //sweet alert message plugin
      title: 'Loading...',
      text: 'Please wait',
      buttons: [false]
    })
    this.userservice.getuserverify(a.uid).subscribe(a => { //get http POST from user service
      if (a['data']) {
        swal({
          title: 'Login Successfully',
          text: 'Welcome!' + this.username,
          icon: 'success',
        })
        console.log('logged in');
        this.nav.navigateForward(this.tab == 'normal' ? 'tabs/tab1' : 'home-senior');
      } else {
        // swal error
        swal({
          title: 'Wrong email or password',
          text: 'Please try again',
          icon: 'warning',
        })
      }
    })
  }
})
```

Figure 6.57: Verify user code segment.

As mentioned above, verify user is done in login controller which verify user status.

```
1 > <ion-content *ngIf="tab == 'normal'">...
40 </ion-content>
41 > <ion-content *ngIf="tab == 'senior'">...
80 </ion-content>
```

Figure 6.58: Switch case between normal mode and simple mode.

The above figure shows the switch case between normal mode and simple mode using angular language “ngIf”.

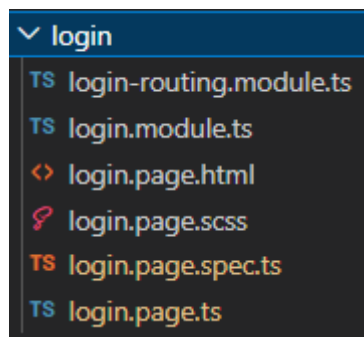


Figure 6.59: Login file.

The view folder of mobile application of ionic framework default is wrapped all of component together. For example, in login folder, it default generated routing module, module, page for html,scss, page for testing and also controller. It

automatically generates and wrapped up when we call from CLI, such as “ionic g page login”.

6.2.2 Pages Hierarchy

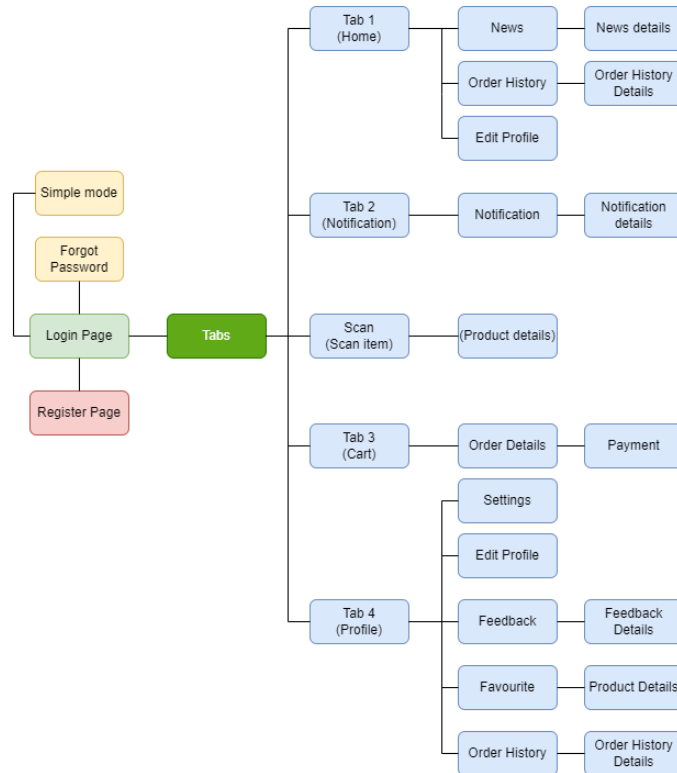


Figure 6.60: Normal mode page hierarchy.

At first phase, the mobile application entry screen is login screen. It will prompt to home page if account is logged in. It also can navigate to register screen through login screen. Besides, to switch to simple mode can also perform through login. After logging in, it will prompt to Tab 1 which is “Home page”. The home page contains news, edit profile, and order history. In tab 2 (Notification) page, it will get notification from merchant side, such as feedback respond. Scan is function that allow to scan using mobile camera on QR code of items. The screen then will prompt to product details. The products details are specific product and allow to add into cart with quantity. Tab 3 is cart page; it contains order details and payment. Lastly, Tab 4 is profile screen, it contains settings, edit profile, feedback, favorite and order history.

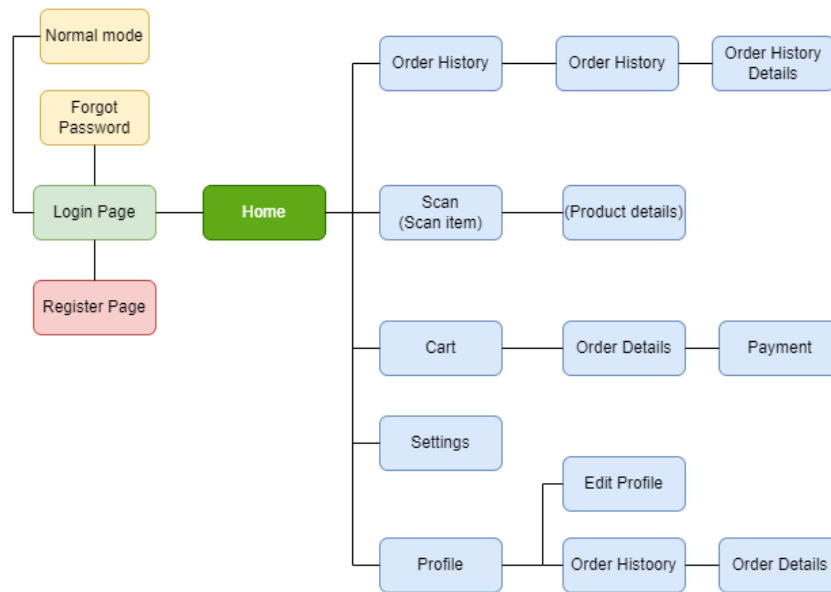


Figure 6.61: Simple mode page hierarchy.

In simple mode, the first screen will be login page. It also will verify user status and last login of user. It can also provide function to switch back to normal mode. After login to account, there are no navigation tabs for simple mode. The screen will only contain main function which is stated as important.

6.2.3 Deployment

```

> Configure project :app
WARNING: Configuration 'compile' is obsolete and has been replaced with 'implementation' and 'api'.
It will be removed in version 7.0 of the Android Gradle plugin.
For more information, see http://d.android.com/r/tools/update-dependency-configurations.html.
FCMPlugin: Support for Gradle v4 or lower is deprecated. Please upgrade to a newer version.
WARNING: Using flatDir should be avoided.
Warning: Mapping new ns http://schemas.android.com/repository/android/common/02 to old ns http://schemas.android.com/repository/android/common/01
Warning: Mapping new ns http://schemas.android.com/repository/android/generic/02 to old ns http://schemas.android.com/repository/android/generic/01
Warning: Mapping new ns http://schemas.android.com/sdk/android/repo/addon2/02 to old ns http://schemas.android.com/sdk/android/repo/addon2/01
Warning: Mapping new ns http://schemas.android.com/sdk/android/repo/repository2/02 to old ns http://schemas.android.com/sdk/android/repo/repository
Warning: Mapping new ns http://schemas.android.com/sdk/android/repo/sys-img2/02 to old ns http://schemas.android.com/sdk/android/repo/sys-img2/01

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.1.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 22s
50 actionable tasks: 3 executed, 47 up-to-date
Built the following apk(s):
  C:\Users\siewshunyao\Documents\Supermarket-self-checkout\platforms\android\app\build\outputs\apk\debug\app-debug.apk
  
```

Figure 6.62: Mobile application deployment.

The above figure shown the supermarket user mobile application deployment using ionic framework.

6.3 Web application for merchant side managements

In this project, web application for merchant side management is implemented with angular js too, it will become a merchant user interface to manage supermarket items or products for customer user to scan and checkout individually. The main function of web application is to create products, and each come out with QR code for customer to scan item, add to cart and checkout efficiently. Angular js is taking part of front-end client within overall system.

6.3.1 Overview of Web Application

In this project, web application play role as managing product for supermarket. The web application using Ionic framework with angular js as front-end development. The screen is separated into two part which is main screen and navigation bar which stick at left section. “Home.page.html” are home page which contains all content includes manage, feedback, notification and profile. For example, as figure shown below, “Home.page.html” include both navigation bar area and main content area. The content area is used to display content such as feedback-details, notification details and product create page, etc.

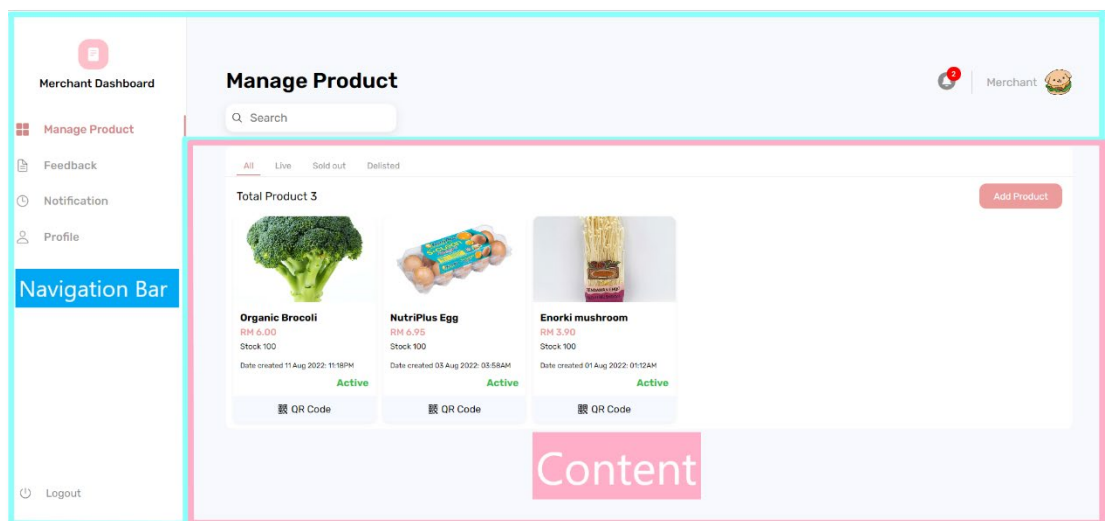


Figure 6.63: Web application screen layout.

```

<div style="width: 100%; margin-top: 50px; height: 100%; ">
  <div style="padding: 30px 45px; height: 100%; width: calc(100%); transition-duration: .2s; ">
    <!-- top -->
    <div style="display: flex; justify-content: space-between; width: 100%; flex-direction: column">...
    </div>
    <!-- manage -->
    <div *ngIf="tab == 'manage'">...
    </div>
    <!-- Feedback tab -->
    <div *ngIf="tab == 'feedback'" style="display: flex; width: 100%">...
    </div>
    <!-- Notification tab -->
    <div *ngIf="tab == 'notification'">...
    </div>
    <!-- profile -->
    <div *ngIf="tab == 'profile'" class="noscroll" style="overflow-x: scroll;">...
    </div>
  </div>
</div>
</div>
</ion-content>
<!-- left section -->
<div [ngClass]="menu ? 'slideInLeft' : 'slideOutLeft' " style="z-index: 100; border-right: 1px solid #236, 2...
</div>

<div (click)="menu = !menu" *ngIf="menu && width() < 1000"
  style="width: calc(100%); height: 100%; background-color: #000; position: fixed; z-index: 1px;
</div>

```

Figure 6.64: Home.page.html source code in collapse.

The figure above shown, the home.page.html contains two section which is content area and navigation bar section (left section). As mentioned above, navigation bars are used as menu to select tabs. The right section is to render tabs content. For example, if tab is feedback, the content area will render feedback content.

6.3.2 Pages Hierarchy

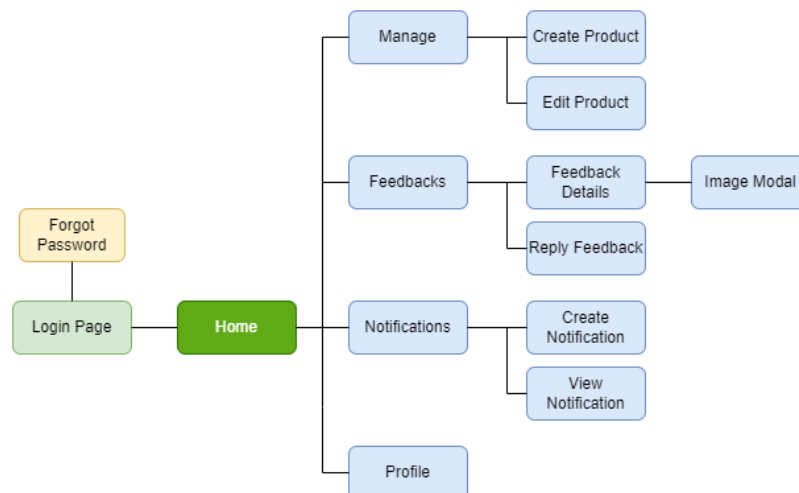


Figure 6.65: Pages Hierarchy of merchant side web application.

The login page is entrance of the page's hierarchy. After merchant account is verified, page will prompt to home page. In home page, manage, feedback, notification and profile are resided in content area. Besides, merchant user can navigate to other page such as create product, edit product through manage tab. Other tab such as feedback and notification are also can navigate to respective page to perform the tasks.

6.3.3 Deployment

```
PS C:\Users\siewshunyao\Documents\Supermarket-checkout-merchant> firebase deploy --only hosting

=== Deploying to 'supermarket-6ebd4'...

i  deploying hosting
i  hosting[supermarket-6ebd4]: beginning deploy...
i  hosting[supermarket-6ebd4]: found 1546 files in platforms/browser/www
+  hosting[supermarket-6ebd4]: file upload complete
i  hosting[supermarket-6ebd4]: finalizing version...
+  hosting[supermarket-6ebd4]: version finalized
i  hosting[supermarket-6ebd4]: releasing new version...
+  hosting[supermarket-6ebd4]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/supermarket-6ebd4/overview
Hosting URL: https://supermarket-6ebd4.web.app
```

Figure 6.66: Pages Hierarchy of merchant side web application.

The figure above shown the merchant side web application deployment using firebase hosting service.

6.4 Web application for administrative side managements

In this project, web application for administrative side management is implemented. It will become an admin user interface to manage merchant account and manage news in customer mobile interface. The main purpose of admin web application is to create, edit new merchant efficiently. Angular js is taking part of front-end client within overall system.

6.4.1 Overview of Web Application

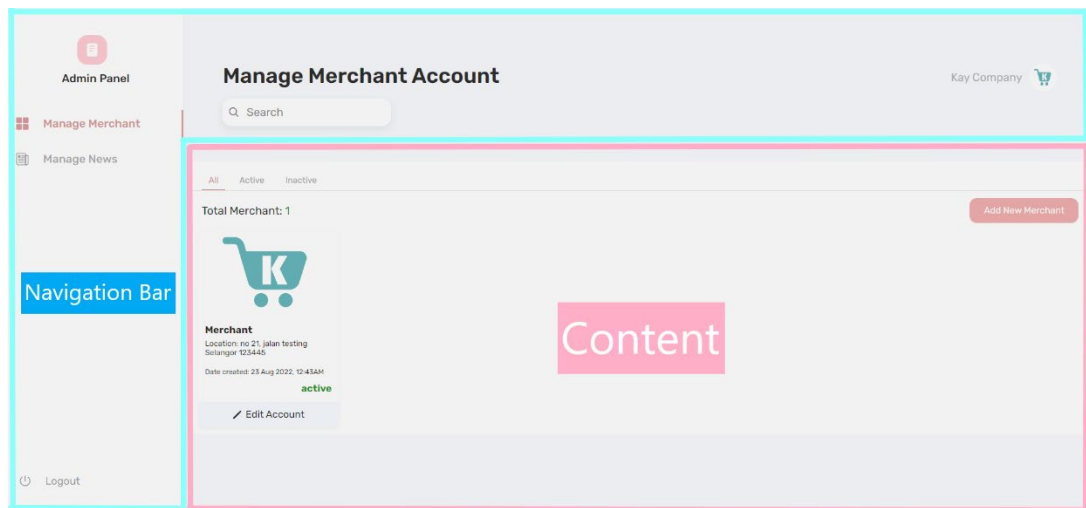


Figure 6.67: Administrative web application screen layout.

```
    <!-- manage merchant-->
  >   <div *ngIf="tab == 'manage'">...
     </div>

    <!-- news -->
  >   <div *ngIf="tab == 'news'">...
     </div>
  </div>
</div>
</ion-content>

<!-- left section -->
> <div [ngClass]="menu ? 'slideInLeft' : 'slideOutLeft' " style=
  </div>

<div (click)="menu = !menu" *ngIf="menu && width() < 1000"
  style="width: calc(100%); height: 100%; background-color: #
</div>
```

Figure 6.68: Home.page.html source code in collapse.

The figure above shown the home.page.html resided in administrative side in collapse. It has the same structure with merchant web application layout, which are separated in two sections, one is navigation bar, and one is rendering the content.

6.4.2 Pages Hierarchy

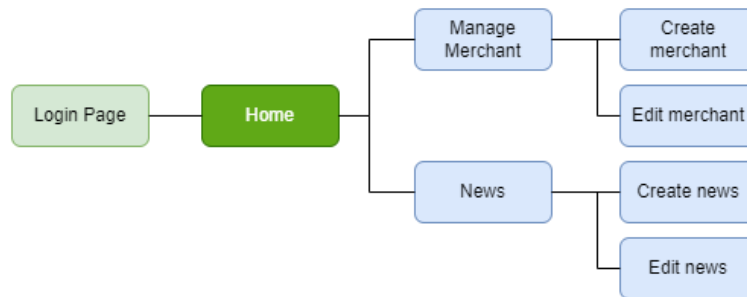


Figure 6.69: Pages hierarchy of administrative side.

The login page is entrance of the page's hierarchy. After admin account is verified, page will prompt to home page. In home page, manage and news are resided in content area. Besides, admin user can navigate to other page such as create merchant account, edit merchant account through manage merchant tab. Other tab such as news can navigate to respective page to perform the tasks.

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

The testing phase is one of the important phases in iterative modelling. There various types of testing strategic involved in this chapter such as unit testing, integration testing and user acceptance testing. Both unit test and integration test are performed with automated testing by using JASMINE testing framework with KARMA test runner. For unit testing, it is test for single piece of a module, it is using httpMock Object which to simulate object that mimic the real-world object in service. Besides, it also using fixture to wrap the component. For integration testing, it is deal with test suite which to ensure modules is work fine together with another module. Integration testing is using jest spyOn library and Fakeasync library to test the test suite in real time.

7.2 Unit Testing

This section will carry out unit testing for both user module and merchant module. Testing using JASMINE JavaScript testing framework. KARMA test runner is to debug the unit test results. The user module tests with service of “user” while merchant module tests with service of “merchant”.

7.2.1 User Module

There total of 11 test cases of unit testing involved in user module.

Table 7.1 Unit testing of user module

Auth Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
001	Login process	<ul style="list-style-type: none">• Email• Password	The warning message will display if entering wrong email or password.	Pass
002	Create user account	<ul style="list-style-type: none">• Name• Email• Gender• Contact• Address• State	The result should return success with 200 status code.	Pass

		<ul style="list-style-type: none"> • Postcode • Photo 		
Product Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
003	Display specific items data	<ul style="list-style-type: none"> • Product_id 	The result should return success with 200 status code and specific items data.	Pass
Notification Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
004	List all notifications		The result should return success with 200 status code and array of notification.	Pass
User Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
005	List user information accordingly	<ul style="list-style-type: none"> • Name • User id • Address 	The result should return success with 200 status code and user information.	Pass
006	Update user information	<ul style="list-style-type: none"> • Name • User id • Address 	The result should return success with 200 status code and updated user information.	Pass
Feedback Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
007	Get specific feedback	<ul style="list-style-type: none"> • User id • Merchant id • Feedback id 	The result should return success with 200 status code and specific feedbacks.	Pass
008	List all feedback	<ul style="list-style-type: none"> • Feedback id 	The result should return success with 200 status code and array of feedbacks.	Pass
009	Add new feedback	<ul style="list-style-type: none"> • User id • Title • Description 	The result should return success with 200 status code and new feedback.	Pass
Order Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
010	Display order history	<ul style="list-style-type: none"> • Order_id 	The result should return success with 200 status code and array of order history.	Pass
News Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
011	Display news	<ul style="list-style-type: none"> • news_id 	The result should return success with 200 status code and latest news.	Pass

The user module unit test is performed using Jasmine framework version 3.8.0. To initial the testing, it required to import `httpTestingModule`, `httpClient` and `HttpTestingController` before the unit test. Each of the unit test case will implement with automated test.

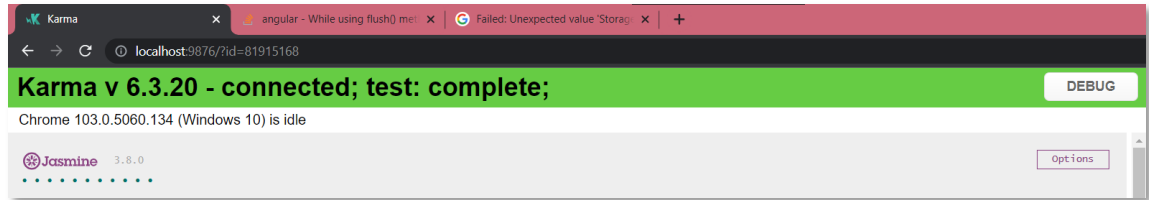


Figure 7.1: Jasmine Framework V3.8.0 – Karma test runner V6.3.20.

```
describe('OrderService', () => {
  let service: OrderService;
  let fixture: ComponentFixture<OrderService>;
  let httpMock: HttpClientTestingModule;

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [OrderService, { provide: HttpClientTestingModule, useExisting: HttpBackend }],
      imports: [HttpClientTestingModule, HttpClientTestingModule],
    });
    service = TestBed.inject(OrderService);
    httpMock = TestBed.inject(HttpClientTestingModule);
  });
});
```

Figure 7.2: Testing providers and imports for unit testing.

The below figure 7.3 is code segment of display product http test in news service. The mock test is using httpMock which declare at earlier. The mock test is setup with dummyNews which to mock the scenario.

```
const dummyNews = {
  news_id: 1,
  title: 'news',
  description: 'Cheeseee',
}

it('Display product, return 200 status code if success else return 800 status code',
inject([HttpClient, HttpTestingController], (http: HttpClient, httpMock: HttpTestingController) => {
  const result = 'news'
  const id = 1
  const description = 'Cheeseee'
  http.get(`${service.ROOT_URL}/getnews`).subscribe(data => {
    expect(dummyNews.news_id).toEqual(id)
    if (dummyNews.news_id == id) {
      expect(dummyNews.title).toEqual(result)
      expect(dummyNews.description).toEqual(description)
      expect(data['status']).toBe(200)
    } else {
      expect(data['status']).toBe(800)
    }
    console.log(data['status']);
  });
  const req = httpMock.expectOne(`${service.ROOT_URL}/getnews`);
  expect(req).toBeDefined();
  expect(req.request.method).toEqual('GET');
  if (dummyNews.title == result) {
    req.flush({ title: dummyNews.title, description: dummyNews.description, status: 200, statusText: 'success' });
  } else {
    req.flush({ title: dummyNews.title, description: dummyNews.description, status: 800, statusText: 'error' });
  }
  httpMock.verify();
  console.log(req);
});
});
```

Figure 7.3: Code segment of display news http test in news service.

The test scenario is to mock the display news scenario and return success status. The http mock will detect the method of http then send data to user. The http response will return success of 200 status code with the assert (expect) called. If http is able to call and get all data, the test is considered as pass.

```
Chrome 104.0.5112.102 (Windows 10): Executed 11 of 11 SUCCESS (0.08 secs / 0.049 secs)
TOTAL: 11 SUCCESS
```

Figure 7.4: Unit test results of mobile user.

7.2.2 Merchant Module

There total of 10 test cases of unit testing involved in merchant module.

Table 7.2 Unit testing of merchant module

Product Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
001	Display all vendor product	<ul style="list-style-type: none"> • Product_id • Merchant_id 	The result should return success with 200 status code and vendor items data.	Pass
002	Create a product	<ul style="list-style-type: none"> • Name • Product_id • Price • Stock • Description • Category • Merchant id 	The result should return success with 200 status code and product data.	Pass
Notification Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
003	List all merchant notifications	<ul style="list-style-type: none"> • Notification_id 	The result should return success with 200 status code and all merchant notification.	Pass
004	Create a notification	<ul style="list-style-type: none"> • Notification_id • Title • Description 	The result should return success with 200 status code and created notification data.	Pass
005	Delete a notification	<ul style="list-style-type: none"> • Notification_id 	The result should return success with 200 status code.	Pass
Merchant Service				
TC ID	Test Case Descriptions	Parameters	Expected Results	Result
005	Get merchant profile data	<ul style="list-style-type: none"> • Name • Merchant_id • Email • Contact 	The result should return success with 200 status code and merchant profile data.	Pass
006	Update merchant information	<ul style="list-style-type: none"> • Name • Merchant id • Contact • Email 	The result should return success with 200 status code and updated merchant information.	Pass
007	Verify merchant	<ul style="list-style-type: none"> • Merchant_id 	The result should return success with 200 status code and merchant profile data.	Pass
Feedback Service				

TC ID	Test Case Descriptions	Parameters	Expected Results	Result
008	Get specific feedback	<ul style="list-style-type: none"> • User id • Merchant id • Feedback id 	The result should return success with 200 status code and specific feedbacks.	Pass
009	List all merchant feedback	<ul style="list-style-type: none"> • Feedback id 	The result should return success with 200 status code and array of feedbacks.	Pass
010	Update feedback	<ul style="list-style-type: none"> • User id • Feedback id • Merchant id • Title • Description 	The result should return success with 200 status code and updated feedback data.	Pass

The merchant module unit test is performed using Jasmine framework version 3.8.0. To initial the testing, it required to import httpTestingModule, HttpClient and HttpTestingController before the unit test. Each of the unit test case will implement with automated test.

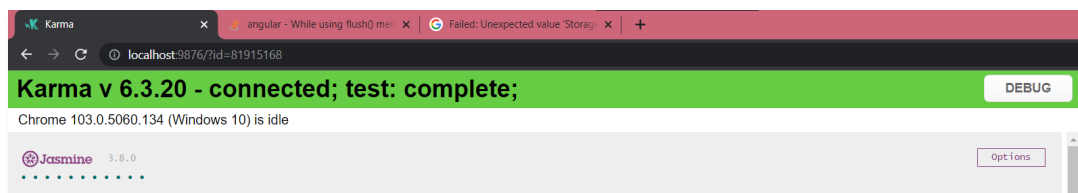


Figure 7.5: Jasmine Framework V3.8.0 – Karma test runner V6.3.20.

```
describe('FeedbackService', () => {
  let service: FeedbackService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [FeedbackService, { provide: HttpClientTestingModule, useExisting: HttpBackend }],
      imports: [HttpClientModule, HttpClientTestingModule],
    });
    service = TestBed.inject(FeedbackService);
  });
});
```

Figure 7.6: Testing providers and imports for unit testing.

The below figure 7.6 is code segment of display product http test in product service. The mock test is using httpMock which declare at earlier. The mock test is setup with dummyProduct which to mock the scenario.

```
const dummyProduct = {
  product_id: 1,
  merchant_id: 2,
  description: 'Helo testing',
  name: 'Milk',
  price: 5,
  category: 'grain',
}
```

Figure 7.7: Dummy product in unit testing for product service.

```
it('Create a product, should return success with 200 status code if error will get status code 800',
inject([HttpClient, HttpTestingController],
(http: HttpClient, httpMock: HttpTestingController) => {
  const temp = {
    product_id: 1,
    id: 2,
    description: 'Helo testing',
    name: 'Milk',
    price: 5,
    category: 'grain',
  }

  http.post(`${service.ROOT_URL}/insertproducts`, temp).subscribe(data => {
    expect(dummyProduct.merchant_id).toEqual(temp.id)
    if (dummyProduct.merchant_id == temp.id) {
      expect(data['name']).toEqual(temp.name)
      expect(data['description']).toEqual(temp.description)
      expect(data['category']).toEqual(temp.category)
      expect(data['price']).toEqual(temp.price)
      expect(data['status']).toBe(200)
    } else {
      expect(data['status']).toBe(800)
    }
    console.log(data['status']);
  });
  const req = httpMock.expectOne(`${service.ROOT_URL}/insertproducts`);
  expect(req).toBeDefined();
  expect(req.request.method).toEqual('POST');
  if (dummyProduct.merchant_id == temp.id) {
    req.flush({
      name: dummyProduct.name, merchant_id: dummyProduct.merchant_id,
      description: dummyProduct.description, category: dummyProduct.category, price: dummyProduct.price,
    });
  } else {

```

Figure 7.8: Code segment of create product http test in product service.

The test scenario is to mock the create merchant product scenario and return success status. The http mock will detect the method of http then send data to merchant user. The http response will return success of 200 status code with the assert (expect) called. If http is able to call and get all data, the test is considered as pass.

```

Chrome 105.0.0.0 (Windows 10): Executed 11 of 11 SUCCESS (0.341 secs / 0.153 secs)
TOTAL: 11 SUCCESS

```

Figure 7.9: Unit test results of merchant module.

7.3 Integration Testing

Integration testing in this project was using jasmine framework which same as unit testing. The methodology of an integration testing is using spyOn library and also fakeAsync library to perform test in real environments. Unlike unit testing, as mentioned above, unit testing is basically test for single modules within application in isolation, in short, no dependencies between other component such as login page's display UI, email and password verification token test and etc. While integration testing is testing the group of different modules that added up together by ensuring is working as expected. There is total 6 test suite tested in this section.

Table 7.3 Integration testing

Merchant Test Suite				
Test suite ID: 1				
Step no#	Step Details	Parameters	Expected Results	Result
1	Admin creates new merchant account	<ul style="list-style-type: none"> • Email • Password • Merchant Name • Address • Postcode • Contact 	The result should return success with 200 status code and merchant data.	Pass
2	Merchant gets verify into application with created account	<ul style="list-style-type: none"> • Merchant ID 	The result should return success with 200 status code when merchant is verified and return merchant data.	Pass
Notification Test Suite				
Test suite ID: 2				
Step no#	Step Details	Parameters	Expected Results	Result
1	Merchant creates new notification	<ul style="list-style-type: none"> • Title • Descriptions • Merchant Name • Merchant ID 	The result should return success with 200 status code when notification is created and return notification data.	Pass
2	Customer user read notification	<ul style="list-style-type: none"> • Notification ID 	The result should return success with 200 status code and notification data.	Pass
Feedback Test Suite				
Test suite ID: 3				
Step	Step Details	Parameters	Expected Results	Result

no#				
1	Customer creates new feedback	<ul style="list-style-type: none"> • Title • Description • Username • User ID • Date • Status • Merchant ID 	The result should return success with 200 status code and created feedback data.	Pass
2	Merchant update the feedback progress	<ul style="list-style-type: none"> • Feedback ID • Title • Description • Merchant ID • User ID • Date • Status 	The result should return success with 200 status code and updated feedback data.	Pass
Product Test Suite				
Test suite ID: 4				
Step no#	Step Details	Parameters	Expected Results	Result
1	Merchant create new product items	<ul style="list-style-type: none"> • Product ID • Merchant ID • Product Name • Price • Descriptions • Status • Date 	The result should return success with 200 status code and created product data.	Pass
2	Customer scan to read specific product	<ul style="list-style-type: none"> • Product ID • Merchant ID • Product Name • Price • Descriptions • Status 	The result should return success with 200 status code and scanned product data.	Pass
3	Customer add product into cart		The result should return success.	Pass
Order Test Suite				
Test suite ID: 5				
Step no#	Step Details	Parameters	Expected Results	Result
1	Customer read orders in cart		The result should return success	Pass

2	Customer read checkout orders	<ul style="list-style-type: none"> • Product ID • Order ID • User ID • Cart ID • Product Name • Price • Descriptions • Status • Quantity • Date 	The result should return success with 200 status code and array of all current orders in order history.	Pass
Profile Test Suite				
Test suite ID: 6				
Step no#	Step Details	Parameters	Expected Results	Result
1	Customer register a new account	<ul style="list-style-type: none"> • User ID • Username • Contact • Status • Email • Address • Postcode • State • Date 	The result should return success with 200 status code and created customer data.	Pass
2	Customer get verify into application with created account	<ul style="list-style-type: none"> • User ID 	The result should return success with 200 status code when is verified and customer data.	Pass
3	Customer update personal information	<ul style="list-style-type: none"> • Username • Contact • Email • Address • Postcode • State • User ID 	The result should return success with 200 status code and updated customer data.	Pass

There are total of 6 test suites are being tested. The main describe function is contained suite definition and injection module such as provider and imports. Each of test suite is tested under one aspect. Besides, each aspect is using consistent structure of a test to easily implement the test such as Arrange, Act and Assert.

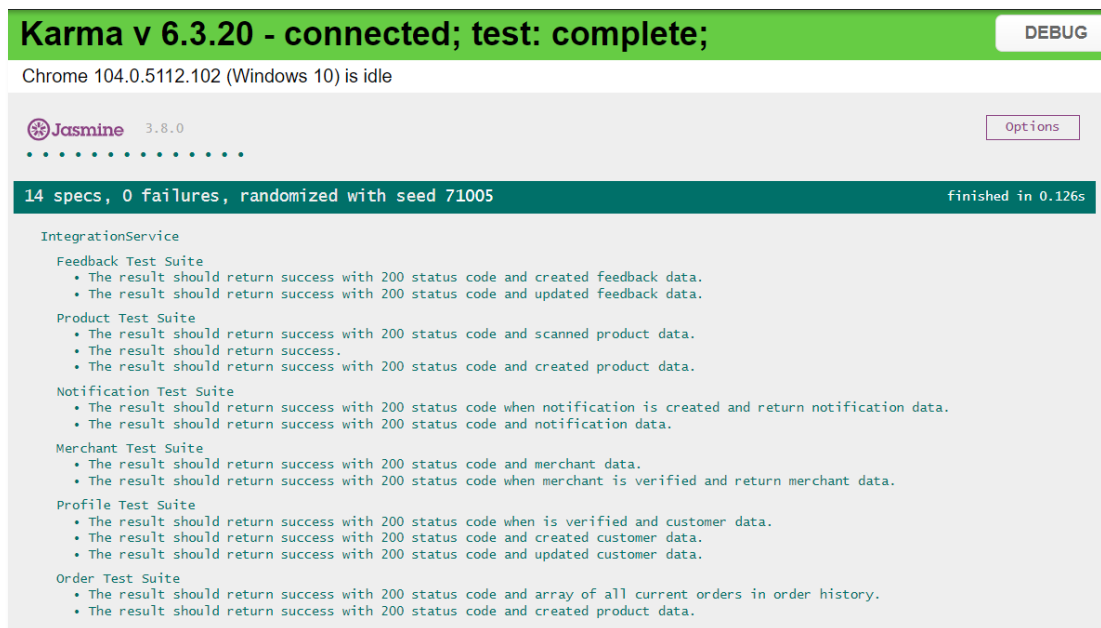


Figure 7.10: Karma framework test debug result.

The figure above shown the test debug result using jasmine framework. The jasmine is act as test-driven development that's supports proposed project to carry out testing practices. To run jasmine test manually is using KARMA test runner by refreshing the browser each time the test changes. The test debug using angular command line if "ng test". It can be defined specifically within one service or many with command line. For example, "ng test --include src/app/integration.service.spec.ts", which run only one single specification typescript page.

```

import { TestBed, inject, tick, flush, fakeAsync } from '@angular/core/testing';
import { HttpClient, HttpClientModule, HttpResponse } from '@angular/common/http';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { IntegrationService } from './integration.service';

describe('IntegrationService', () => {
  let service: IntegrationService;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [
        HttpClientModule,
        HttpClientTestingModule
      ],
      providers: [
        IntegrationService
      ]
    });
    service = TestBed.inject(IntegrationService);
  });

  afterEach(inject([HttpTestingController], (backend: HttpTestingController) => {
    backend.verify();
  }));

  // it('should be created', () => {
  //   expect(service).toBeTruthy();
  // });

  describe('Merchant Test Suite', () => { ...
  })
  describe('Notification Test Suite', () => { ...
  })
  describe('Feedback Test Suite', () => { ...
  })
  describe('Product Test Suite', () => { ...
  })
  describe('Order Test Suite', () => { ...
  })
  describe('Profile Test Suite', () => { ...
  })
});

```

Figure 7.11: Integration test suite source code collapse.

The figure above shown integration test suite source code in collapse. There are several libraries need to import before writing the test suite such as “Testbed”,HttpClientTestingModule, “HttpTestingController” and etc. All under test api is written on integration service, in this case, the spec environment must import or call before each http is used to test respectively. As mentioned above, there several tasks need to be done in structural way to carry out a complete test which is using AAA (Arrange, Act, Assert) pattern. Arrange is a setup at a start of test case such as object that will go under test, beforeEach, beforeAll, afterEach and afterAll function. Act is target on test behaviour which will have sort of response on test debug. For example, in jasmine framework there would be spyOn, httpresponse, event(), flush() and etc. Lastly, assert is kind of expect outcomes such as expect object to equal some result. This can determine whether the test case is success of fail.

```

describe('Feedback Test Suite', () => {
  afterEach(inject([HttpTestingController], (backend: HttpTestingController) => {
    backend.verify();
  }));
  const temp = { merchant_id: 4, user_id: 1, title: 'Hello', description: 'testing testing', username: 'test' };

  it('The result should return success with 200 status code and created feedback data.',
    fakeAsync(inject([IntegrationService, HttpTestingController],
      (service: IntegrationService, backend: HttpTestingController) => {

        service.insertfeedbacks(temp).subscribe(data => { expect(data).toEqual(temp), fail });
        const req = backend.expectOne(`${service.ROOT_URL}/insertfeedbacks`);
        expect(req.request.url).toBe(`${service.ROOT_URL}/insertfeedbacks`);
        expect(req.request.method).toEqual('POST');
        expect(req.request.body).toEqual(temp);
        const expectedResponse = new HttpResponse({ status: 200, statusText: 'Created', body: temp });
        req.event(expectedResponse);
        req.flush(temp);
        backend.verify();
        tick();
        console.log(expectedResponse)
        console.log(req);
      }));

  it('The result should return success with 200 status code and updated feedback data.',
    fakeAsync(inject([IntegrationService, HttpTestingController, HttpClient],
      (service: IntegrationService, backend: HttpTestingController, http: HttpClient) => {
        const temp1 = { feedback_id: 6, merchant_id: 4, user_id: 1, title: 'Hello', description: 'testing testing', username: 'test' };
        const id = 6;
        http.post(`${service.ROOT_URL}/updatefeedbacks`, id).subscribe(data => { expect(id).toEqual(temp1.feedback_id), fail });
        const req = backend.expectOne(`${service.ROOT_URL}/updatefeedbacks`);
        expect(req.request.url).toBe(`${service.ROOT_URL}/updatefeedbacks`);
        expect(req.request.method).toEqual('POST');
        expect(req.request.body).toEqual(id);
        const expectedResponse = new HttpResponse({ status: 200, statusText: 'Created', body: temp1 });
        req.event(expectedResponse);
        req.flush({ id: temp1.feedback_id });
        backend.verify();
        tick();
        console.log(expectedResponse)
        console.log(req);
      }));
});

```

Figure 7.12: Feedback test suite code segment.

The figure above shown integration test on feedback test suite code segment.

There are two test case be carried out respectively.

```

Chrome 104.0.5112.102 (Windows 10): Executed 14 of 14 SUCCESS (0.089 secs / 0.074 secs)
TOTAL: 14 SUCCESS

```

Figure 7.13: Integration test performance.

7.4 User Acceptance Testing

User acceptance testing will be conducted in this project. It is different compared to unit testing and integration testing as customer or merchant will involve in UAT. UAT does not require any programming or coding skill as it is an end user testing for ensure whether the application such as flow, performance and etc, that is meeting the business requirement. In this section, the user acceptance testing template is carried out. The user acceptance testing template is referenced from CSTE CBOOK which is a Certification Guide for testing.

Tables below shown lists for supermarket user and merchant of user acceptance testing test cases:

Table 7.4: Register module for supermarket user

Module 1: Register Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
1.1	Register Process	N/A	In login page, press “register here” to register the user account.	N/A	Prompt to register page.	
			Fill in all information (Name, contact, email, address, date of birth, password etc).		Input box can input text and selection box is able to select the choice.	
			Press “register” button.	N/A	Validation warning will display if (email not in	

					correct format) or (not fill all the field). Once above all is meet, it will display success message.	
			Check if prompt to login once successful register the account.	N/A	Go back to login page.	

Table 7.5: Login module for supermarket user

Module 2: Login Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
2.1	Login Process	Have an account	Fill in all information (Email and password).		Input box can input text.	
			Press "login" button.	N/A	Validation warning will display if (email not in correct	

					format) or (not fill all the field). Once above is meet, it will display welcome message	
			Check if prompt to homepage once successful login the account.	N/A	Prompt to home page.	

Table 7.6: Profile module for supermarket user

Module 3: Profile Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
3.1	Update Profile Process	Login into account	In home page, press the profile photo to enter account or press profile at tabs and press edit profile.	N/A	Enter the edit profile page.	
			Update profile information.		Input box can edit the text.	
			Press update button.	N/A	Validation warning will	

					display if (email not in correct format) or (not fill all the field). Once above is meet, it will display welcome message.	
			Check if profile is updated.	N/A	Updated the user profile information.	

Table 7.7: Notification module for normal user

Module 4: Notification Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
4.1	Read notifications	Login into account	In home page, press the notification tab at bottom of the screen.	N/A	Enter the notification page.	
			Check if notification is existed.	N/A	View array of notification.	
4.2	Read specific notification details	Login into account, at least one notification	In home page, press the notification tab at bottom of the screen	N/A	Enter the notification page.	
			Press any notification to enter the full page.	N/A	The full page of specific notification details will be	

					seen.	
--	--	--	--	--	-------	--

Table 7.8: Cart module for supermarket user

Module 5: Cart Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
5.1	Scan item	Login into account	In home page, press scanner icon at the bottom tab.	N/A	Enter scanning mode.	
			Scan QR code.	N/A	Scan successful.	
			Check if the item details page is display.	N/A	Item details page is display.	
5.2	Add item	Login into account	In home page, press scanner icon at the bottom tab.	N/A	Enter scanning mode.	
			Add and minus quantity.	N/A	Default quantity is 1 Add and minus must work fine.	
			Press add item.	N/A	Item added into cart and direct prompt to cart page.	

5.3	Delete item	Login into account	In home page, press cart tab to enter cart page.	N/A	Display cart page.	
			In cart page, select any item to perform delete, swipe to left to delete.	N/A	The delete icon display on each array of items.	
			Press delete icon to delete an item.	N/A	The current item is deleted.	
			Check if the total price and total checkout number is correct according to current state.	N/A	The total price and total checkout number is correct.	
5.4	Checkout item	Login into account	In home page, press cart tab to enter cart page.	N/A	Display cart page.	
		Have an online account/credit card/debit card	Press checkout button.	N/A	Display payment method page.	
			Select payment method.		Prompt to payment page.	
			Input all required bank information.		Verify the bank info.	
			Confirm to pay.		Payment successful and display payment successful message.	

Table 7.9: Order module for supermarket user

Module 6: Order Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
6.1	View all order history	Login into account, atleast checkout one item	In home page, at order history, press view all.	N/A	Enter order history page.	
			Check if the order history is display.	N/A	Order history are display.	
6.2	View specific order history details	Login into account, at least one checkout item	In home page, at order history, press view all.	N/A	Enter order history page.	
			Select specific order history.	N/A	Display specific order history details.	

Table 7.10: Favourite module for normal user

Module 7: Favourite Module						
Test	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result

ID						
7.1	Add to favorite	Login into account	In home page, press scanner icon at the bottom tab.	N/A	Enter scanning mode.	
			Scan QR code.	N/A	Scan successful.	
			Check if the item details page is display.	N/A	Item details page is display.	
			At top right corner, press “heart icon” to bookmark the item.	N/A	Pop up successful message.	
7.2	View favorite	Login into account, at least one favorite item	In home page, press favorite.	N/A	Enter favorite page.	
			Check if favorite items are displayed.	N/A	Favorite items are displayed.	

Table 7.11: Feedback module for normal user

Module 8: Feedback Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
8.1	Create feedback	Login into account	In home page, press feedback to enter feedback page.	N/A	Enter feedback page.	
			Press create button at top right corner.	N/A	Feedback form page is display.	
			Fill in required information (Choose merchant, enter title, choose type, enter description, add image(optional)).		Validation warning will display if required information is not filled.	
			Press submit button.	N/A	Pop up successful message.	
8.2	Read feedback	Login into account, at least one feedback	In home page, press feedback to enter feedback page.	N/A	Enter feedback page.	
			Check if feedback lists are displayed.	N/A	Feedback lists are displayed.	

			Press any feedback to view details.	N/A	Feedback details are displayed.	
8.3	Delete feedback	Login into account, at least one feedback	In home page, press feedback to enter feedback page.	N/A	Enter feedback page.	
			Select one feedback, swipe to left.	N/A	Delete icon displayed.	
			Press delete icon to delete feedback.	N/A	Pop up successful message.	

Table 7.12: Search module for normal user

Module 9: Search Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
9.1	Search by merchant name	Have a user account	Login into user account.	N/A	Enter home page.	
			Select merchant at home page.	N/A	Enter merchant store page.	
			Input merchant name at search bar.		Display merchant.	
			Check if input keyword is match with display merchant.	N/A	Keyword is match with display merchant.	

Table 7.13: Login module for merchant user

Module 1: Login Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
1.1	Login Process	Have an account	Fill in all information (Email and password).		Input box can input text.	
			Press “login” button.	N/A	Validation warning will display if (email not in correct format) or (not fill all the field). Once above is meet, it will display welcome message.	
			Check if prompt to homepage once successful login the account.	N/A	Prompt to home page.	

Table 7.14: Manage product module for merchant user

Module 2: Manage Product Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
2.1	Add Product	Have a merchant account	Login into merchant account.	N/A	Enter home page. Default tab (manage).	
			Press add button.	N/A	Go to add product first page.	
			Enter product name and choose category.		The input length is not more than 120 words The chosen product category will display at below	
			Press next button.	N/A	Display next page of create product.	
			Enter product image, price, stock and product description.	N/A	The input box can input text. The image can be	

					uploaded from device.	
			Press publish button.	N/A	Display successful message.	
			Check if the product is added at manage product tab. Check if the QR code can be displayed.	N/A	Product is added QR code can be displayed.	
2.2	Edit product	Have a merchant account. Have at least one product.	Login into merchant account.	N/A	Enter home page. Default tab (manage).	
			Choose specific product to edit.	N/A	Enter the product edit page.	
			Choose next page to continue edit.	N/A	Enter next page of edit page.	
			Press publish button.	N/A	Pop up successful message.	
			Check if the product is edited at manage product tab.	N/A	Product is edited.	
2.3	Delete product	Have a merchant	Login into merchant account and homepage.	N/A	Default tab is managed product tab.	

		account.	Select specific product to delete.	N/A	Enter edit product page.	
		Have at least one product.	Press delete button.	N/A	Pop up confirmation message.	
			Press confirm button.	N/A	Pop up successful message.	
			Check if the deleted product is removed.	N/A	Product has been removed.	

Table 7.15: Feedback module for merchant user

Module 3: Feedback Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
3.1	Preview feedback details	Have a merchant account and have user feedback	Login into merchant account.	N/A	Enter homepage. Default tab (manage).	
			Select feedback tab.	N/A	It shows preview button when tab on any feedback and prompt to feedback details page.	
			Press preview button at right of specific	N/A	Enter feedback details.	

			feedback.			
			Check if the feedback details is display Check if the photo can be displayed in modal.	N/A	Feedback is displayed Photo can be displayed in modal.	
3.2	Reply feedback	Have a merchant account and have user feedback	Login into merchant account.	N/A	Enter homepage. Default tab (manage).	
			Select feedback tab.	N/A	Enter feedback page.	
			Choose incomplete feedback.	N/A	Preview button is displayed.	
			Press preview button.	N/A	Enter feedback details page.	
			Press reply button.	N/A	Reply feedback form modal pop up.	
			Fill in title, description, or image (optional).		Input box can input text Image can upload from device.	
			Press send button.	N/A	Pop up confirmation message.	
			Press confirm button.	N/A	Pop up successful	

					message.	
			Check if feedback status become “Completed”.	N/A	Feedback status become “Completed”.	

Table 7.16: Notification module for merchant user

Module 4: Notification Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
4.1	Create notification	Have a merchant account	Login into merchant account and go to homepage.	N/A	Default tab is managed product tab.	
			Select notification tab.	N/A	Enter notification page.	
			Press create button.	N/A	A create notification modal is pop up.	
			Enter title, description.		The input box can input text. The image can be uploaded from device.	
			Press create button.	N/A	Display successful	

					message.	
			Check if the notification is added at notification page.	N/A	Notification is added.	
4.2	View notification details	Have a merchant account and one notification	Login into merchant account and go to homepage.	N/A	Default tab is managed product tab.	
			Select notification tab.	N/A	Enter notification page.	
			Press any notification.	N/A	A notification details modal is pop up.	
			Check if the notification details is correct.	N/A	Notification details is correct.	
4.3	Delete notification	Have a merchant account and one notification	Login into merchant account and go to notification page.	N/A	Enter notification page.	
			Select specific notification to delete.	N/A	Enter notification details page.	
			Press delete button.	N/A	Pop up confirmation message.	
			Press confirm button.	N/A	Pop up successful message.	
			Check if notification is removed from	N/A	Notification has been	

			notification page.		removed from notification page.	
--	--	--	--------------------	--	---------------------------------	--

Table 7.17: Profile module for merchant user

Module 5: Profile Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
5.1	Edit merchant profile	Have a merchant account	Login into merchant account and go to profile.	N/A	Enter profile page.	
			Edit profile information (any info).		Input box can input text. The image can be uploaded from device.	
			Press update button.	N/A	Pop up confirmation message.	
			Press confirm button.	N/A	Pop up successful message.	
			Check if profile information is updated.	N/A	Profile information is updated.	

Table 7.18: Advanced search module for merchant user

Module 6: Advanced Search Module						
Test ID	Test Description	Pre-conditions	Test Steps	Input Data	Expected Result	Actual Result
6.1	Search by product name	Have a merchant account, at least one product	Login into merchant account.	N/A	Enter home page. Default tab (manage).	
			Select product name at search option.	N/A	Enter search product name mode.	
			Input product name.		Display product.	
			Check if input keyword is match with display product's name.	N/A	Keyword is match with display product's name.	
6.2	Search by product category	Have a merchant account, at least one product	Login into merchant account.	N/A	Enter home page. Default tab (manage).	
			Select product category at search option.	N/A	Enter search category mode.	
			Input category name.		Display list of categories.	
			Select category.		Display product.	
			Check if selected category is match with	N/A	Selected category is match	

			display product.		with display product.	
--	--	--	------------------	--	-----------------------	--

Table 7.19 User Acceptance Testing Summary Result (Supermarket normal user)

No	Acceptance Requirement	Critical		Test Result (Number of people)	
		Yes	No	Accept	Reject
1.1	Register process	✓		3	0
2.1	Login process	✓		3	0
3.1	Update profile process		✓	3	0
4.1	Read notifications	✓		3	0
4.2	Read specific notification details	✓		3	0
5.1	Scan item	✓		3	0
5.2	Add item	✓		3	0
5.3	Delete item	✓		3	0
5.4	Checkout item	✓		3	0
6.1	View order history	✓		3	0
6.2	View specific order history details		✓	3	0
6.1	Add to favourite	✓		3	0
6.2	View favourite		✓	3	0
7.1	Create feedback	✓		3	0
7.2	Read feedback	✓		3	0
7.3	Delete feedback		✓	3	0
8.1	Search by merchant name		✓	3	0

Table 7.20 User Acceptance Testing Summary Result (Supermarket senior user)

No	Acceptance Requirement	Critical		Test Result (Number of people)	
		Yes	No	Accept	Reject
1.1	Register process	✓		2	0
2.1	Login process	✓		2	0
3.1	Update profile process		✓	2	0
5.1	Scan item	✓		2	0
5.2	Add item	✓		2	0

5.3	Delete item	✓		2	0
5.4	Checkout item	✓		2	0
6.1	View order history	✓		2	0
6.2	View specific order history details		✓	2	0

Table 7.21 User Acceptance Testing Summary Result (Merchant user)

No	Acceptance Requirement	Critical		Test Result (Number of people)	
		Yes	No	Accept	Reject
1.1	Login process	✓		2	0
2.1	Add product	✓		2	0
2.2	Edit product	✓		2	0
2.3	Delete product		✓	2	0
3.1	Preview feedback details	✓		2	0
3.2	Reply feedback	✓		2	0
4.1	Create notification	✓		2	0
4.2	View notification details		✓	2	0
4.3	Delete notification		✓	2	0
5.1	Edit merchant profile		✓	2	0
6.1	Search by product name		✓	2	0
6.2	Search by product category		✓	2	0

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Conclusions

The proposed project, “A mobile app for supermarket checkout” is developed based on the early planning and research to reach the project goals and objectives. Iterative methodology was practiced throughout the project life cycle to achieve manage project timeline and follow up the system progress. Besides, the combination of different framework such as angular.js and ionic framework to develop a mobile app and web app. Furthermore, the proposed system has done various of validation and testing which include unit testing, integration testing and UAT (user acceptance testing) to ensure is meeting the project requirements that stated earlier. The objectives achieved has been fulfilled that listed at below:

- i. To investigate the current problem of supermarket checkout process time when dealing a huge number of customers.
- ii. To analyze the public acceptance of self-checkout service in supermarket.
- iii. To develop a supermarket self-checkout mobile application which provides self-payment service that helps supermarket to reduce the labour costs and checkout process time.

According to the objective list above, the first objectives have been achieved in this project. The system invented to tackle the problem for customer of queuing up at cashier counter by just purchasing less items. The mobile application is allowed to checkout item individually by scanning the product using product QR code and add into cart in mobile application. Customer then is able to checkout their items by online transaction and leave the supermarket. The second objectives have been achieved on this project. There are several analyses and testing are accomplished such as user acceptance testing, questionnaire or survey, observation has been done to analyse the public acceptance of proposed system. Lastly, the third objectives achieved as there is reduces the labour costs and checkout process time when the mobile application can provide a self-service for customer to checkout products. In

addition, similar and existing system research and literature review were evaluated and come out main features as listed below:

Customer side:

- i. Supermarket user authentication and authorization
- ii. Scan module
- iii. Image upload
- iv. Item checkout
- v. Responsive user interface
- vi. Search function
- vii. Feedback module
- viii. Notification module
- ix. Favorite/ bookmark module
- x. Order activity
- xi. Profile
- xii. Geolocation of supermarket user
- xiii. Two interface available (senior and normal)

Merchant side:

- i. Merchant user authentication and authorization
- ii. Image upload
- iii. Responsive user interface
- iv. Advanced search function
- v. Manage product for supermarket user
- vi. Manage notification for supermarket user
- vii. Manage supermarket user feedback
- viii. Profile
- ix. Geolocation of merchant user

8.2 Limitations

There are some limitations found in this project. Although the main objectives are saved time during the checkout operation compared to pay items at cashier counter, novice user which perform scan and checkout may need to try out a few times to familiar with the operation, this may cause the time-consuming rate higher than pay items at cashier counter. Nevertheless, the high rate of relying on the behavior of customer when it is checking out their cart has raised up the possibility of threatening

of treat. The proposed solution is based on the merchant side decision, there is no way can be avoided from technical side when requirements are to reduce the installation cost.

8.3 Recommendations for future work

There are some future implementations in this project. This project is underlying with iterative model, and it can boost and work with incremental development model which can breaks down into pieces of workflow. Each of the refined work is built on previous version as it can improve step by step. The project can do adjustment and transform into service that provides a product management platform for supermarket, retail store or groceries.

For the merchant management side, the merchant has considered 1 role and separate into multiple staff account. The merchant manages product to perform bulk upload at the same time to manage staff status and view product sales in visualization. Each staff member has their account credentials and their position. The platform is also can have subscription plan which to join the platform as merchant with selected subscription plan, e.g., subscription premium plan to get more service. In organization, which is administrative side, it manages merchandise account and their subscription status.

In customer side, the mobile application can implement reward system which allow customers to claim points and discount voucher to apply it on specific merchant store. Besides, customer side mobile application also has a social share such as using referral code to refer a new friend to join the supermarket self-checkout mobile application. This action is beneficial to existing user and new user which both will get a reward point as act as rewards for active sharer and also for a new joiner user.

REFERENCES

Thamutharam, Y.N., Peer Mustafa, M.B, Musthafa, F.N and Tajudeen, F.P, 2021. Usability Features to Improve Mobile Apps Acceptance among the Senior Citizens in Malaysia. *Journal of Science*, [e-journal], vol. 16, pp. 1-3. <https://doi.org/10.32802/asmscj.2021.686>.

Hassan, H., Sade, A.B. and Rahman, M.S., 2013. Self-service Technology for Hypermarket Checkout Stations. *Journal of Asian Social Science*, [e-journal], vol. 10, no. 1, pp. 61-63. <http://dx.doi.org/10.5539/ass.v10n1p61>.

Mendat, C.C., Mayhorn, C.B., 2007. An Evaluation of Self-Checkout Systems. *Journal of proceedings of the Human Factors and Ergonomics Society Annual Meeting*, [e-journal] 51(17), pp. 2-4. 10.1177/154193120705101703.

Bwyer, B., 2019. Self-Checkout: Should You Implement It? [online] Available at: < <https://www.cardfellow.com/blog/self-checkout-should-you-implement-it/>> [Accessed 19 February 2022].

Insider Intelligence, 2016. Retailers face threat of theft with self-service checkouts. [online] Available at: < <https://www.businessinsider.com/retailers-face-threat-of-theft-with-self-service-checkouts-2016-8> > [Accessed 6 March 2022].

Dean, D.H, 2008. Shopper age and the use of self-service technologies. *Managing Service Quality: An International Journal*, [e-journal], vol. 18 no. 3, pp. 225-238. <https://doi.org/10.1108/09604520810871856>.

Bernard, S., 2007. Cashiers' work- time: Between a productivity mentality and a service mentality. *Journal of Sociology of Work*, [e-journal], 49(supp-S2), pp. e129-e144. <https://doi.org/10.1016/j.socotra.2007.10.001>.

iStrategy Conference, 2021. Self-checkout: Why Technology Still Has a Long Way to Go in Retail. [online] Available at: <<https://www.istrategyconference.com/self-checkout-technology-long-way-to-go-retail/>> [Accessed 6 March 2022].

Kananke, T.S., 2020. *A mobile application to enhance shopping experience by introducing self-service and self-checkout*. Master. University of Colombo School of Computing. Available at: <<http://dl.ucsc.cmb.ac.lk/jspui/handle/123456789/4493>> [Accessed 12 March 2022].

Airbrake, 2016. *Iterative Model: What Is It And When Should You Use It?* [online] Available at: < <https://airbrake.io/blog/sdlc/iterative-model#:~:text=The%20iterative%20model%20is%20a,the%20final%20system%20is%20complete.>> [Accessed 12 March 2022].

Genius Learner. 2020. [image online] Available at: < <https://genius-learner.medium.com/iterative-model-software-engineering-b11a76da7895>> [Accessed 12 March 2022]

Rastogi, V., 2015. Software development life cycle models-comparison, consequences. *International Journal of Computer Science and Information Technologies*, 6(1), pp.168-172. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.9896&rep=rep1&type=pdf>

Hijazi, H., Khmour, T. and Alarabeyyat, A., 2012. A review of risk management in different software development methodologies. *International Journal of Computer Applications*, 45(7), pp.8-12. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.6705&rep=rep1&type=pdf>

ButterCMS, 2019. Angular vs. Vue: Which Framework Is Best for You? [Online] Available at: < <https://buttercms.com/blog/comparing-angular-vs-vue>> [Accessed 24 March 2022].

Sharma, Brij & Dubey, Shivendra & Jain, Neelesh & Syed, Habeebullah Hussaini. (2020). A Review on Cloud Computing Services and issues. SSRN Electronic Journal. 9(4), pp 1-6.
https://www.researchgate.net/publication/349521421_A_Review_on_Cloud_Computing_Services_and_issues

SF AppWorks, 2021. COMPARING FIREBASE VS. AWS | SF APPWORKS. [Online] Available at: <<https://www.sfappworks.com/blogs/firebase-vs-aws-comparision>> [Accessed 27 March 2022].

Chandra, V., 2015. Comparison between various software development methodologies. *International Journal of Computer Applications*, 131(9), pp.7-10.
<https://www.ijcaonline.org/research/volume131/number9/chandra-2015-ijca-907294.pdf>

GLOWID, 2021. Firebase Vs AWS: Which One to Choose in 2022? [Online] Available at: < <https://aglowiditsolutions.com/blog/firebase-vs-aws/>> [Accessed 28 March 2022].

SIMFORM, 2021. Laravel vs Node.js: Everything You Should Know Before Ratifying Project Backend. [Online] Available at: <<https://www.simform.com/blog/laravel-vs-nodejs/>> [Accessed 26 March 2022].

SIMFORM, 2020. React Native vs Ionic: Which Framework is best and Why? [Online] Available at: < <https://www.simform.com/blog/react-native-vs-ionic/>> [Accessed 27 March 2022].

Alshamrani, A. and Bahattab, A., 2015. A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. *International Journal of Computer Science Issues (IJCSI)*, 12(1), p.106.

Sitepoint, 2018. Angular Introduction: What It Is, and Why You Should Use It. [Online] Available at: < <https://www.sitepoint.com/angular-introduction/>> [Accessed 25 March 2022]

Dunka, Bakwa & Emmanuel, Edim & Oyerinde, Yinka., 2017. HYBRID MOBILE APPLICATION BASED ON IONIC FRAMEWORK TECHNOLOGIES. *International Journal of Recent Advances in Multidisciplinary Research*. 4(12), pp 3121-3130.

javaTpoint, n.d. Firebase Introduction. [Online] Available at: <<https://www.javatpoint.com/firebase-introduction>> [Accessed 27 March 2022].

Amazon, 2022. Overview of Amazon Web Services. [Online] Available at: <<https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html>> [Accessed 28 March 2022].

Hava, 2021. What is AWS Elastic Beanstalk? [Online] Available at:<<https://www.hava.io/blog/what-is-aws-elastic-beanstalk>> [Accessed 29 June 2022].

Aws, n.d. Auto scaling groups. [Online] Available at: <<https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-groups.html>> [Accessed 29 June 2022].

Ionic DOCS, n.d. Testing. [Online] Available at:<<https://ionicframework.com/docs/angular/testing>> [Accessed 3 July 2022].

APPENDICES

APPENDIX A: Observation





APPENDIX B: Questionnaire Form

Section 1 of 4

Survey on youth-aged, middle-aged and senior-aged for a Supermarket Self-Checkout Mobile Application



I am Siew Shun Yao, a final year undergraduate student, pursuing Bachelor of Science(Hons) Software Engineering from University Tunku Abdul Rahman (UTAR) Sungai Long.

I am currently in the early stage of developing a Supermarket Checkout Mobile Application. The purpose of this study is to appraise whether the supermarket checkout application will bring a benefit and convenience to supermarket consumer. Besides, the findings is allowed to provide a support to my study claims. Your participation will greatly contribute to the success of the project. I truly appreciate your participation in this survey. Your responses will remain private and it will be used strictly for academic purpose only. Thank You.

2. Gender? *

- Female
- Male
- Prefer not to say

3. Age? *

- Youth-Aged (Between 15 years old to 24 years old)
- Middle-Aged (Between 25 years old to 64 years old)
- Senior-Aged (Between 65 and above)

5. Full Name *

Eg: Siew Shun Yao

Short answer text

6. Do you think paying at the cashier counter will be time consuming? *

Yes

No

7. Do you ever experience self-checkout by using a mobile app or device? *

Yes

No

Section 2 of 4

Section 2



Description (optional)

8. Based on question 7, If yes, How often do you use self-checkout? *

Seldom

Often

9. Based on question 7, If yes, what are the stores/restaurants that you have done self-checkout * before?

Supermarket

Groceries store

Minimarket

Other...

10. Based on question 7, If yes, Rate experience in using the self- checkout application in a scale 1 - 5? *

1

2

3

4

5

Bad experience

Excellent Experience

Section 3 of 4

Section 3



Description (optional)

11. Based on question 7, If no, Do you want to experience it if available? *

Yes

No

Section 4 of 4

Section 4



Description (optional)



12. What kind of situations will lead you to apply self-checkout when doing transaction? *

Pay for less items

Fast payment and go

Clean and hygiene

13. Do you think it is more convenient to have a self-checkout application for you to pay for your items? *

Yes

No

⋮

14. Tick the supermarket checkout mobile application features that you think is important. *

Contents of Categorization

Advance Search Functions

Notifications (New event updates)

History

Profile (Self information)

15. Any other features that you would like to suggest. (Optional)

Long answer text

APPENDIX C: Supervisor and Moderator Comments on Project Plan

Project title:	A mobile app for supermarket checkout
Student Name	SIEW SHUN YAO
Supervisor	Chean Swee Ling
Moderator	Hoo Meei Hao

Key Assessment for Project Proposal	Supervisor Comments/Remarks	Moderator Comments/Remarks
Project Description - Is the problem or need to be addressed clearly presented? - Is the proposed approach or solution clearly presented and justified?	No comment.	ok

<p>Project Scope and Objectives</p> <ul style="list-style-type: none"> - Is the scope of the project clearly defined? - Are the objectives of the project clearly specified? - Are the project scope and objectives appropriate for a final year project? 	<p>No comment.</p>	<p>These 2 objectives : To minimize the queuing time at physical cashier counter in existing supermarket; To improve overall smoothness on the checkout flow while purchasing a small number of items - are unable to measure its achievement. Consider to remove these 2 objectives because it is the expectation/ benefit after the solution is implemented.</p>
<p>Literature Review / Fact Finding for Benchmarking / Verification of Project</p> <ul style="list-style-type: none"> - Are sources for literature review / fact finding appropriate? - Is information from literature review / fact finding relevant and adequate? - Is information from literature review / fact finding clearly presented and discussed? 	<p>No comment.</p>	<p>ok</p>
<p>Research/Development Methodology and Development Tools- Is the methodology for the project clearly described and discussed?- Are the required development tools clearly described and discussed?- Are the stated methodology and development tools appropriate?</p>	<p>No comment.</p>	<p>avoid to use "I" in report writing. The selected methodology, iterative and incremental must reflect in the project schedule. Project schedule does not show which activities are in incremental mode. What are those activities involved in the incremental model?</p>
<p>Project Plan</p> <ul style="list-style-type: none"> - Are the phases and tasks of the project properly defined and planned? - Are the phases and tasks consistent with the methodology of the 	<p>No comment.</p>	<p>tasks in the project schedule is not consistent with the methodology selected. Tasks in the project schedule just shows structured way.</p>

project?		
<p>Initial Deliverables- Are deliverables (e.g. use case diagrams and descriptions) of initial phases of the project plan included in the report?</p>	No comment.	<p>Explain the purpose of the data gathering performed. observation and questionnaire were conducted with reported result. Relate the observation result contribute to the workflow in the proposed system or to support the problem identified. Some results do not show significant support to the project (eg The result shows there are majority (80%) of respondents is in a satisfactory range by experiencing selfcheckout system), and does not support the proposal. Just relate the result that deemed necessary. How are the attributes in the ERD being identified? from forms / receipt or existing system?</p>
<p>Report Structure and References - Is the report organised in a logical structure? - Are references listed in accordance to Harvard format?</p>	No comment.	
<p>Language and Clarity of Writing - Are the sentences concise and understandable? - Are there spelling and grammar issues?</p>	Grammar mistakes found in the report.	

APPENDIX D: Project Progress (Trello Web Application)

