

DEVELOP AGENT AND RESELLER SYSTEM IN GATHER DEAL

WONG YONG JIE


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

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

September 2023

DECLARATION

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

Signature :  _____

Name : WONG YONG JIE _____

ID No. : 2003560 _____

Date : 20/5/2024 _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**DEVELOP AGENT AND RESELLER SYSTEM IN GATHER DEAL**” was prepared by **WONG YONG JIE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Dr Tham Mau Luen

Date : 20 May 2024

Signature : _____

Co-Supervisor : Dr Khor Kok Chin

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.

© 2023, Wong Yong Jie. All right reserved.

ABSTRACT

The Agent and Reseller System is a web application developed within the Gather Deal framework to simplify commission management processes for resellers and administrators. This project aims to solve the complexities encountered while calculating and managing commissions by providing a centralised system that integrates with Gather Deal's existing infrastructure. The system provides effective commission calculation, sales monitoring, and reporting capabilities through real-time data processing and comprehensive functionality. By using cutting-edge web technologies like Django, React, and Node.js, the system provides resellers with an easy-to-use interface to monitor sales performance, retrieve commissions, and handle their accounts. The system allows administrators to set commission rates, monitor reseller activity, and produce extensive data that may be reviewed further. Ensuring easy adoption and utilisation of the system through comprehensive testing, validation, and documentation improves operational efficiency, decision-making skills, and satisfaction with users across the Gather Deal ecosystem.

TABLE OF CONTENTS

ABSTRACT		5
TABLE OF CONTENTS		i
LIST OF TABLES		iii
LIST OF FIGURES		v
LIST OF APPENDICES		ix
CHAPTER 1	10	
1	INTRODUCTION	10
	1.1 General Introduction	10
	1.2 Problem Statement	12
	1.3 Aim and Objectives	13
	1.4 Proposed Solution	14
	1.5 Proposed Approach	14
	1.6 Scope and Limitations of the Project	16
CHAPTER 2	18	
2	LITERATURE REVIEW	18
	2.1 Introduction	18
	2.2 Commission Type	18
	2.3 Cloud-based Systems	20
	2.4 User Experience and Interface Design	25
	2.5 Software Development Methodology	29
	2.6 Similar Existing Web Applications	33
	2.7 Overall Summary	36
CHAPTER 3	38	
3	METHODOLOGY AND WORK PLAN	38
	3.1 Introduction	38
	3.2 Software Development Methodology	38
	3.3 Project Planning and Scheduling	41
	3.4 Technologies and Development Tools	48

3.5	Summary	52
CHAPTER 4	53	
4	PROJECT SPECIFICATION	53
4.1	Introduction	53
4.2	Fact Finding	53
4.3	Functional Requirements	54
4.4	Non-Functional Requirements	57
4.5	Use Case Modelling	59
CHAPTER 5	71	
5	SYSTEM DESIGN	71
5.1	Introduction	71
5.2	System Architectural Design	71
5.3	Database Design	73
5.4	System Flow Design	78
CHAPTER 6	85	
6	SYSTEM IMPLEMENTATION	85
6.1	Introduction	85
6.2	Database Setup and Management	85
6.3	Application Programming Interfaces (APIs)	87
6.4	Web Application Implementation	89
CHAPTER 7	124	
7	SYSTEM TESTING AND EVALUATION	124
7.1	Introduction	124
7.2	Unit Testing	124
7.3	System Usability Testing	140
CHAPTER 8	145	
8	CONCLUSION AND DISCUSSION	145
8.1	Conclusions	145
8.2	Limitations and Future Enhancement	146
REFERENCES	148	
APPENDICES		151

LIST OF TABLES

Table 2-1: AWS's Benefits and Limitations	21
Table 2-2: GCP's Benefits and Limitations	22
Table 2-3: Azure's Benefits and Limitations	24
Table 2-4: Benefits and Limitations of Waterfall Software Development Methodology	30
Table 2-5: Benefits and Limitations of DevOps Software Development Methodology	31
Table 2-6: Benefits and Limitations of Rapid Application Development Methodology	32
Table 2-7: Benefits and Limitations of Performio Application	33
Table 2-8: Benefits and Limitations of CaptivateIQ Application	34
Table 2-9: Benefits and Limitations of Everstage Application	35
Table 4-1: User Management Module Functional Requirements	54
Table 4-2: Agent and Reseller Registration Module Functional Requirements	54
Table 4-3: Agent and Reseller Tier Structure Module Functional Requirements	55
Table 4-4: Commission Management Module Functional Requirements	55
Table 4-5: Sales Tracking, Analytics, and Reporting Module Functional Requirements	56
Table 4-6: Non-Functional Requirements	57
Table 4-7: Mapping between Functional Requirement ID and Use Case ID	58
Table 4-8: Use Case Description of Login Reseller Account	60
Table 4-9: Use Case Description of Edit Reseller List	61
Table 4-10: Use Case Description of Send Invitation Request	62
Table 4-11: Use Case Description of Register Reseller Account	63

Table 4-12: Use Case Description of Set Tier Naming	64
Table 4-13: Use Case Description of View Reseller Network Diagram	65
Table 4-14: Use Case Description of Allocate Commission Rate	66
Table 4-15: Use Case Description of Withdraw Commission	67
Table 4-16: Use Case Description of View Commission Transaction History	68
Table 4-17: Use Case Description of Upload Withdraw Transaction Slip Number	69
Table 4-18: Use Case Description of View Sales Analysis	70
Table 5-1: Data Dictionary (ResellerDetail)	74
Table 5-2: Data Dictionary (Setting)	74
Table 5-3: Data Dictionary (DirectCommission)	75
Table 5-4: Data Dictionary (IndirectCommission)	75
Table 5-5: Data Dictionary (LeadershipCommission)	75
Table 5-6: Data Dictionary (PoolCommission)	76
Table 5-7: Data Dictionary (CommissionRecord)	76
Table 5-8: Data Dictionary (WithdrawRecord)	77
Table 6-1: List of Function Endpoints	88
Table 7-1: Test Cases of User Management Model	125
Table 7-2: Test Cases of Agent and Reseller Registration Module	127
Table 7-3: Test Cases of Agent and Reseller Tier Structure Module	131
Table 7-4: Test Cases of Commission Management Module	131
Table 7-5: Test Cases of Sales, Analytics, and Reporting Module	138
Table 7-6: Usability Testing Scenario (Admin Side)	141
Table 7-7: Usability Testing Scenario (Reseller Side)	142
Table 7-8: SUS Score Interpretation (Bangor, et al., 2009)	144

Table 7-9: SUS Score of Admin Side Application	144
Table 7-10: SUS Score of Reseller Side Application	144

LIST OF FIGURES

Figure 1-1: CRM in Gather Deal	11
Figure 1-2: System Structure in Gather Deal	11
Figure 1-3: Structure of the Proposed Solution	14
Figure 1-4: DevOps Methodology (Business Transformation Career Advice Database DevOps Observability, 2022)	16
Figure 2-1: Three Elements of User Experience (Berni & Borgianni, 2021)	26
Figure 2-2: Waterfall Software Development Methodology's phases (Mohammad, et al., 2019)	29
Figure 2-3: DevOps Methodology's phases (Business Transformation Career Advice Database DevOps Observability, 2022)	30
Figure 2-4: Rapid Application Development Methodology's phases (kissflow, 2023)	32
Figure 3-1: DevOps Methodology (Business Transformation Career Advice Database DevOps Observability, 2022)	38
Figure 3-2: Project Schedule Overview	42
Figure 3-3: Project Initialization Schedule	43
Figure 3-4: System Development Schedule	43
Figure 3-5: System Testing Schedule	44
Figure 3-6: Project Closure Schedule	44
Figure 3-7: Project Initialization Gantt Chart	45
Figure 3-8: System Development Gantt Chart	46
Figure 3-9: System Testing Gantt Chart	46

Figure 3-10: Project Closure Gantt Chart	47
Figure 4-1: Use Case Diagram	59
Figure 5-1: Overview of System Architecture Design	71
Figure 5-2: Entity Relationship Diagram of System	73
Figure 5-3: Reseller side, Invitation Message	78
Figure 5-4: Reseller side, Register Account Interface	78
Figure 5-5: Reseller side, Agreement Interface	79
Figure 5-6: Reseller side, Login Interface	79
Figure 5-7: Reseller side, Commission Withdraw Interface	80
Figure 5-8: Reseller side, Sales Report Interface	80
Figure 5-9: Administrator side, Analyse Sales Interface	81
Figure 5-10: Administrator side, Send Invitation Request Interface	81
Figure 5-11: Administrator side, List of Reseller Interface	82
Figure 5-12: Administrator side, Commission Distribution Interface	82
Figure 5-13: Administrator side, Update Commission Withdraw Interface	83
Figure 5-14: Administrator side, Network Diagram of Reseller Interface	83
Figure 5-15: Administrator side, Tier Settings Interface	84
Figure 5-16: Administrator side, Commission Settings Interface	84
Figure 6-1: Gather Deal Repository	85
Figure 6-2: Gather Deal Existing Database Schema	86
Figure 6-3: Modifications of Database Schema	86
Figure 6-4: Version Control System	87
Figure 6-5: Reseller Register Page	89
Figure 6-6: Reseller Agreement Page	90
Figure 6-7: handleRegisterClick function	91

Figure 6-8: signUp endpoint	91
Figure 6-9: Reseller LogIn Page	92
Figure 6-10: handleLoginClick function	93
Figure 6-11: logIn endpoint	93
Figure 6-12: Reseller LogOut Button	94
Figure 6-13: LogOut function	94
Figure 6-14: Reseller List Page	95
Figure 6-15: getResellerList function	95
Figure 6-16: getResellerDetail endpoint	96
Figure 6-17: handleStatusChange function	96
Figure 6-18: updateResellerStatus endpoint	97
Figure 6-19: Invite Reseller Module	97
Figure 6-20: inviteReseller function	98
Figure 6-21: createResellerDetail endpoint	99
Figure 6-22: Reseller Network Page	99
Figure 6-23: useEffect function (Reseller Network Page)	100
Figure 6-24: getResellerTree endpoint	100
Figure 6-25: Tier Settings Page	101
Figure 6-26: getSetting function	101
Figure 6-27: getSettings endpoint	102
Figure 6-28: createSetting function	102
Figure 6-29: creatSetting endpoint	103
Figure 6-30: updateSetting function	103
Figure 6-31: updateSetting endpoint	104
Figure 6-32: Commission Settings Page	104

Figure 6-33: useEffect function (Commission Settings Page)	106
Figure 6-34: getDirectCommission, getIndirectCommission, getLeadershipCommission and getPoolCommission endpoints	107
Figure 6-35: save function	108
Figure 6-36: createDirectCommission & updateDirectCommission function	108
Figure 6-37: createIndirectCommission & updateIndirectCommission function	109
Figure 6-38: createLeadershipCommission & updateLeadershipCommission function	109
Figure 6-39: createPoolCommission & updatePoolCommission function	110
Figure 6-40: createDirectCommission & updateDirectCommision endpoints	111
Figure 6-41: createIndirectCommission & updateIndirectCommision endpoints	111
Figure 6-42: createLeadershipCommission & updateLeadershipCommision endpoints	112
Figure 6-43: createPoolCommission & updatePoolCommision endpoints	112
Figure 6-44: Reseller Dashboard Page	113
Figure 6-45: handleWithdraw function	113
Figure 6-46: makeWithdrawRequest endpoint	114
Figure 6-47: Sales Report Page	114
Figure 6-48: useEffect Function (Sales Report Page)	115
Figure 6-49: getCommissionList endpoint	115
Figure 6-50: Commission Management Page	116
Figure 6-51: Update Commission Withdraw Module	116
Figure 6-52: handleTransactionNumber Function	117
Figure 6-53: updateWithdrawStatus endpoint	117

Figure 6-54: makepaymentcommission function	117
Figure 6-55: makePaymentCommission endpoint	119
Figure 6-56: Administrator Dashboard Page	120
Figure 6-57: useEffect Function (Administrator Dashboard Page)	121
Figure 6-58: getResellerCount endpoint	121
Figure 6-59: getTotalCommission endpoint	122
Figure 6-60: getCommissionList endpoint	123
Figure 7-1: System Usability Test	140

LIST OF APPENDICES

Appendix A: Stakeholder Consultations Discussion	151
Appendix B: System Usability Test	157

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Agent and Reseller system is a web application system created to connect and manage resellers and the data such as commission and sales. The main focus of this system will be helping the organization calculate the commission of the reseller. This is because reseller commissions may come to a complex work when it has different commission rates and types based on their performance, seniority, or other criteria set by the organization. It works as a centralized system that contains several levels of reseller and sales information, works together, and simplifies processes. Through the provision of real-time data into numerous components of an organization's reseller, it could help efficiency, productivity, and decision-making. Business-type systems are often customized to meet the specific needs and requirements of each organization. Therefore, the price for a business-type system is in the cost range from ten thousand to billions of dollars for implementation, customization, infrastructure, support, and ongoing maintenance. It is only affordable for large and multinational organizations. In order to build a business-type system, it will be more effective with the help of a foundation system, such as inventory management or customer relationship management (CRM) system. Building a new system without any foundation can be a complex task that requires careful planning and coordination to avoid data inconsistencies or disruptions in operations. In addition, systems require continual technical maintenance and support to minimize disruptions and resolve any possible issues.

Agent and Reseller System in Gather Deal can solve the problem of complex calculation of commission, high cost, system foundation, and technical maintenance and support. The main idea is to create a web-based Agent and Reseller system in Gather Deal that helps the organization to process the complex commission calculation process. Gather Deal is a service platform that currently provides Customer Relationship Management (CRM) system software applications on a subscription basis through the Internet as shown in Figure 1-1. Users can access and utilize the application using a web browser rather than

installing and executing it locally on a computer or server. Therefore, it only required a subscription fee for the users to use the system. The main virtual facilities required by Agent and Reseller System include cloud databases to store the data, and Amazon Web Services (AWS) servers to host the web application. On the other hand, this does not require any physical facilities.

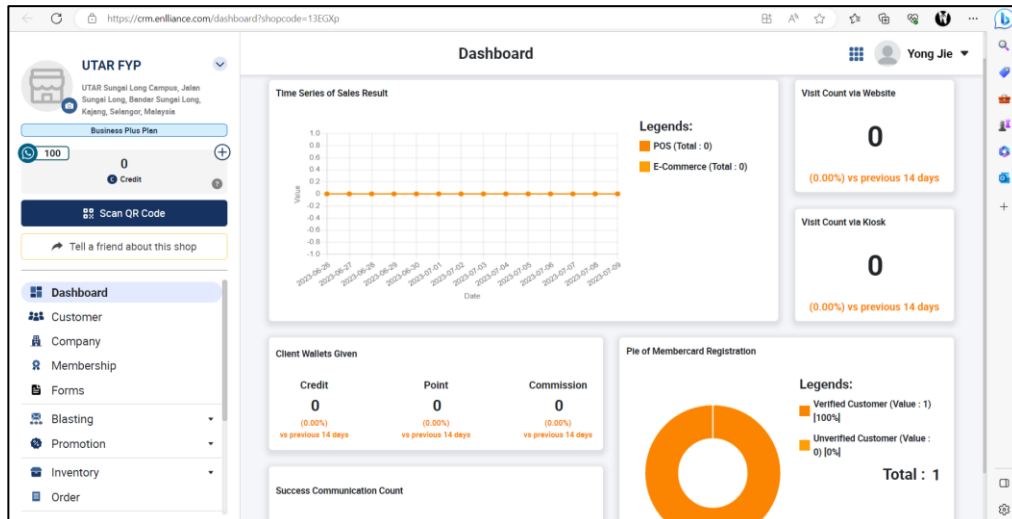


Figure 1-1: CRM in Gather Deal

To create the system on top of Gather Deal will use the Node.js environment, React.js library, Next.js, and Django framework. Therefore, Gather Deal will contain two systems for the user to subscribe to as shown in Figure 1-2.

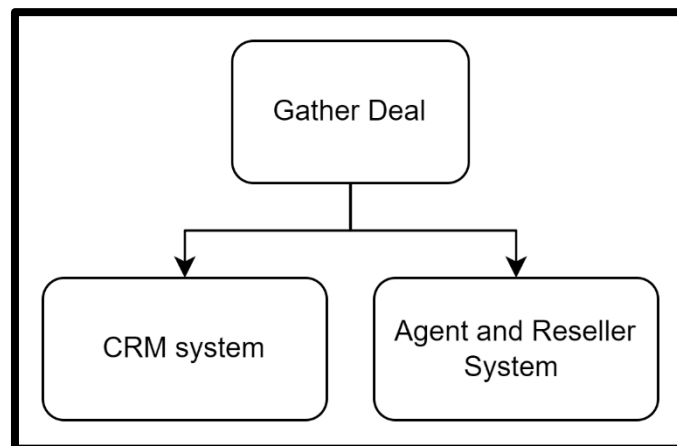


Figure 1-2: System Structure in Gather Deal

In this project, there are 5 steps for the user to use the Agent and Reseller System, each step is listed below:

- (i) Access to Gather Deal Website

- (ii) Register as a user in Gather Deal
- (iii) Subscribe to the Agent and Reseller System
- (iv) Setup properties in the Agent and Reseller System
- (v) Start adding resellers, and use the system

1.2 Problem Statement

1.2.1 Complexity of Commission Calculation

Due to the many variables that must be taken into consideration, the commission calculation process for the reseller can be very complex. First off, depending on their performance, level of seniority, or other standards established by the organization, various resellers may have varied commission rates. Second, the commission could change based on the kind of sale, the product category, or the amount of sales. Thirdly, to guarantee that resellers are paid the proper amount, the system must reliably track and compute commissions in real-time. The process should also consider any adjustments to commission rates, sales data, or reseller information. Last but not least, the commission process must be open and available to the business and the reseller, offering thorough reports and breakdowns of commission profits.

1.2.2 High Cost

To implement a business-based system, there is a need for business knowledge and specialized skills. Organizations frequently consult system integrators or IT consultants to help with the installation process. These services cover tasks including gathering requirements, configuring the system, migrating data, testing the system, training, and change management. According to Podmurnyi (2022), simple software will cost around 80,000 dollars to build, medium software will cost around 160,000 dollars, and complicated software will cost around 250,000 dollars. The cost of these services may increase the overall cost. Organizations may need to invest money in hardware and software for on-premises systems, including servers, storage, and networking tools. These infrastructure expenses may add up, especially for larger organizations with more demanding IT requirements. Therefore, this will be very costly for small and medium-sized enterprises (SMEs).

1.2.3 System Foundation

A system creation runs the risk of failing to satisfy the demands of the organization without an in-depth understanding of current business processes and a strong foundation. The processes may become ineffective and inconsistent, which may reduce productivity and confuse users. A system foundation gives the chance to standardize procedures, data layouts, and processes throughout various divisions and functions. Without a solid foundation, it can be difficult to create uniformity and standardization, which can result in inconsistent data, duplication of work, and problems integrating systems.

1.2.4 Technical Maintenance and Support

If the system is created by the organization itself, the organization will require constant support and maintenance. This will require the organization to allocate a number of technical staff to continuously monitor to help find and fix any flaws or defects in real-time. As Widestadh (2021) points out the maintenance cost may cost up to 40% to over 90% of the total cost of development. To guarantee optimal system operation, monitoring may include proactive alerts, error tracking, log analysis, and performance monitoring. This process also included maintaining the system operating properly and fixing any bugs, security vulnerabilities, or performance concerns, regular updates, and upgrades are necessary. Installing improvements, software upgrades, and newly released versions.

1.3 Aim and Objectives

The main aim of this project is to provide small and medium-sized enterprises (SMEs) with a system that can provide real-time data of an organization reseller's activities, and help efficiency, productivity, and decision-making easily. It can also facilitate the process of commissions and tiers of the resellers in the organization.

Specifically, the objectives of this project are:

- (i) To develop a real-time web reseller application to monitor the organization reseller's activities.
- (ii) To facilitate the process of commissions and tiers of the resellers in the organization.

- (iii) To deploy web applications using cloud services.

1.4 Proposed Solution

The proposed solution attempts to solve the problems highlighted above, the client which is the SMEs will require access to the Gather Deal website through the internet that is generated by the Vercel. In the fronted development, React will be used to build reusable UI components, and Next.js to provide server-side rendering, routing, and performance optimizations. JavaScript ES6 code to enhance interactivity and implement specific functionalities. On the other hand, the backend development Django will be used for server-side logic, database management, and Node.js with frameworks to provide server-side development. Lastly, the data storage will use one or more AWS EC2s that are manage by AWS Elastic Beanstalk. All the data will be stored in AWS RDS and AWS S3. GitHub to promote CICD flow with Python and JavaScript ES6 programming languages. Figure 1-3 is the overview of the proposed solution.

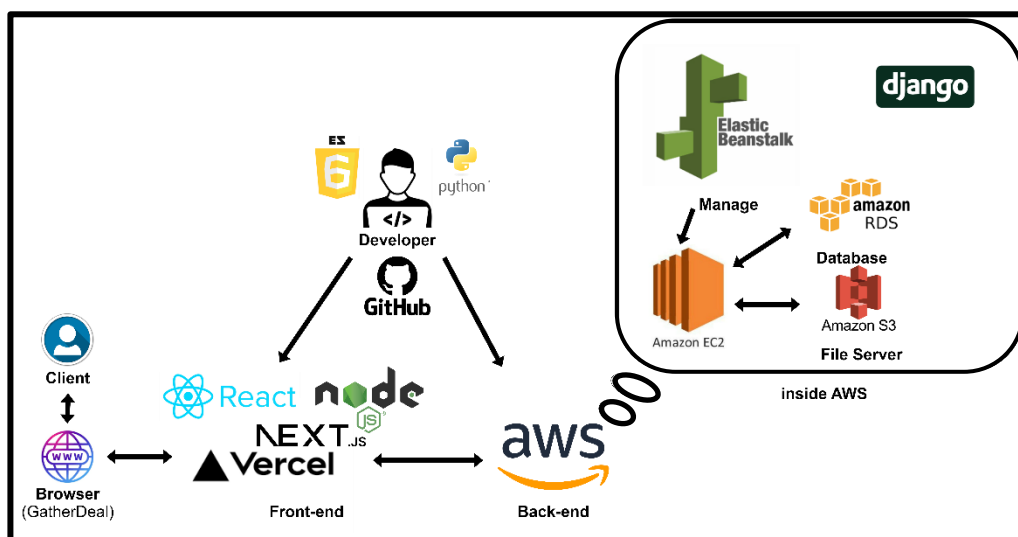


Figure 1-3: Structure of the Proposed Solution

1.5 Proposed Approach

This project proposed the implementation of the DevOps approach. DevOps is the combination of cultural philosophies, practices, and tools that can improve the project capacity to deliver applications and services at high velocity: evolving and improving products more quickly than using conventional infrastructure management and software development procedures. (Pérez, et al., 2015) DevOps could benefit the project in various ways, including the following:

- (i) Increase the reliability**

DevOps help to ensure the project quality of application updates and infrastructure adjustments so that the delivery can be consistently at a faster rate while keeping the end-user experience favourable. Test each change to ensure it is secure and functioning using techniques like continuous integration and continuous delivery.

- (ii) Increase Speed**

DevOps may assist the project in adopting a continuous delivery strategy, in which software updates are sent to clients in brief, regular batches. This reduces the risk of mistakes, problems, and disputes and enables the project to more rapidly and successfully adapt to customer needs and market needs.

- (iii) Enhance Resource Efficiency**

DevOps assists in the project's resource and infrastructure efficiency maximizing and the elimination of unnecessary or redundant tasks. The cost and difficulty of maintaining several environments, tools, and platforms are also decreased through DevOps. Last but not least, DevOps enables the project to do more with less and to provide the client with greater value.

Hence, DevOps is a useful approach that assists the project in achieving its objectives more quickly, better, and more effectively. The software development and delivery process may be made more effective, high-quality, agile, and innovative with the help of DevOps.

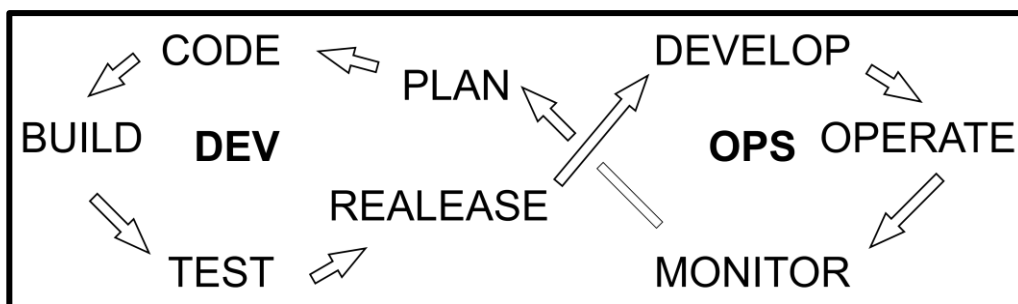


Figure 1-4: DevOps Methodology (Business Transformation Career Advice Database DevOps Observability, 2022)

1.6 Scope and Limitations of the Project

The project's scope is to deploy the online Agent and Reseller system in Gather Deal. The organization Enlliance Management Sdn. Bhd. has developed the Gather Deal online platform. This project mainly focuses on integrating the existing Gather Deal online platform with the Agent and Reseller system created by the Node.js environment, React.js library, Next.js, and Django framework. The limitation of the project will not be able to provide a very niche situation for each organization. This is because the system will be a general Reseller and Agent system that can suit the majority of the organization. The organization will need to make some additional settings to customize their needs.

1.6.1 Targeted User

The targeted user for this project is the Small and medium-sized enterprises (SMEs) mainly companies at contain large amounts of resellers in Malaysia. The system will be useful and affordable for the SME and the system is designed based on the behaviour and culture of local Malaysian organizations. However, the system has role and permission functions, and the SME can make some pages of the web accessible for the reseller according to their need.

1.6.2 Modules Covered

The list below states the modules covered in this project:

- (i) **User Management Module**
This module handles user accounts, permissions, resellers, and administrators.
- (ii) **Agent and Reseller Registration Module**

This module enables individuals or organizations to register as the organization's resellers.

(iii) Agent and Reseller Tier Structure Module

This module controls downline tracking and multi-tiered commissions.

(iv) Commission Management Module

This module calculates and tracks the commissions earned by resellers based on their sales and referrals.

(v) Sales, Analytics, and Reporting Module

This module records and monitors sales made by resellers, allowing for performance analysis and reporting.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter which is the literature review for the project “Develop Agent and Reseller System in Gather Deal”, a cloud-based platform accessible to anyone through subscription, aims to explore and analyze existing literature related to various aspects of the system. This review will delve into research surrounding the commission type, cloud-based systems, user experience and interface design, and software development methodologies. The review looks through relevant works in these areas in an effort to learn more about the field's innovations, issues, and best practices. It will advance knowledge of agent and reseller systems' functions, advantages, and difficulties in a cloud-based environment. Additionally, the evaluation will cover important elements that are crucial to the success of the Agent and Reseller System in Gather Deal, such as user experience, and software development methodologies.

2.2 Commission Type

Businesses commonly employ a variety of commission plans for resellers in the competitive business environment to increase sales and draw in more resellers. (Carter, 2023) These commission structures are intended to encourage resellers to aggressively market and sell the business's products or services, boosting sales and broadening the market. Direct commissions, indirect commissions, leadership commissions, and pool commissions are a few examples of popular commissions kinds. Resellers are encouraged to push for higher sales volumes by the commissions, which give them a set proportion of the overall sales they produce. Also, commissions encourage resellers to aim for larger sales milestones by rewarding them with increasing commission rates when they meet predetermined sales objectives.

2.2.1 Direct Commission

A reseller commission is a financial incentive provided to resellers based on the sales they generate for the organization. Resellers who purchase products or

services from the organization and then sell them to their customers at a higher price receive the commission from the organization as their profit. (Anderson, 2019) The commission paid to the reseller often represents a portion of their overall sales. The possibility of making more money with bigger sales volumes encourages resellers to actively market and sell the products or services of the organization. By providing a competitive and alluring reseller commission, a business may entice additional resellers to join its distribution network, increasing its market presence and reach.

2.2.2 Indirect Commission

Indirect commission for resellers refers to a commission structure where a reseller not only earns commissions from their direct sales but also receives additional commissions from other resellers that they have referred to them. (Hayes, 2023) A multi-level reseller structure is used to describe this kind of commission structure. In this approach, resellers are encouraged to hire and train new resellers, building a network or "downline" of resellers, in addition to making their own sales. A multi-tiered commission system is created when these recruited resellers generate sales and bring in other resellers while paying a commission to the original reseller on their own sales. As they stand to gain from both their own sales as well as the sales of their downline, this indirect compensation arrangement can provide resellers a strong motivation to grow and expand their network. It is a concept that organizations seeking to increase their market presence and reach find appealing since it may result in exponential development in sales and distribution.

2.2.3 Leadership Commission

A leadership commission compensation system pays people according to how well they can motivate and support a team in reaching goals in sales or business. With a portion of the commission going to the reseller who closed the deal and the bulk going to the team members who helped with the sale, this incentive model highlights the value of cooperation and teamwork. The leadership commission develops a collaborative culture and harmonizes individual ambitions with the overarching organizational goals by encouraging a sense of shared responsibility and encouraging team members to collaborate. It not only

rewards individual sales achievement but also emphasizes the need for teamwork in closing agreements with larger margins and achieving overall corporate success.

2.2.4 Pool Commission

Pool commission is a strategic commission structure that prioritizes team collaboration and overall success rather than just individual sales performance. A commission pool is created, with a portion of the money going to the salesperson and the bulk going to the team members who helped make the deal successful. (Forbes Business Development Council, 2019) This promotes teamwork and encourages collaboration among diverse departments and team members to successfully conclude agreements. Recognizing everyone on the team for their contributions, promotes a sense of shared ownership and responsibility, encouraging everyone to remain involved and committed to attaining favourable results. The aim of higher-margin sales that can help the organization, in the long run, is encouraged since the commission pool is often established based on the gross margin amounts earned by the sale.

2.2.5 Summary

In summary, in order to reward resellers, boost sales, and broaden the organization's market reach, this Agent and Reseller System will provide a variety of compensation options. This may encourage resellers to aggressively market and sell goods or services, thus giving Direct Commission, Indirect Commission, Leadership Commission, and Pool Commission. Resellers will get compensation from Direct Commission depending on their sales, which will motivate them to increase their sales volumes. A multi-level agent and reseller structure made possible by the indirect commission will encourage a rapid increase in sales and distribution.

2.3 Cloud-based Systems

2.3.1 Amazon Web Services (AWS)

Amazon Web Services (AWS) is a comprehensive and widely used cloud computing platform provided by Amazon. It provides a wide range of cloud services, tools, and services to meet the demands of companies, programmers,

and organizations of all sizes. AWS eliminates the need for physical on-premises infrastructure by enabling customers to access computing power, storage, and other resources via the Internet.

According to Mukherjee (2019, pp. 2-11), there are various benefits of using the AWS cloud services including cost effectiveness, auto scaling, and automatic maintenance, however, it also comes with a few limitations which are cost prohibitive, and usage is not facile. Table 2-1 shows the benefits and limitations of AWS in detail.

Table 2-1: AWS's Benefits and Limitations

Benefits	Limitations
<ul style="list-style-type: none"> <li data-bbox="304 808 786 1283"> <p>• Cost Effectiveness: With the Pay-As-You-Go pricing model available from AWS, businesses only pay for the resources they use. This flexibility enables organizations to scale up or down in response to demand, maximizing savings and preventing overprovision.</p> <li data-bbox="304 1305 786 2007"> <p>• Auto scaling and Automatic maintenance: In order to ensure optimal resource utilization, auto-scaling helps manage bandwidth use by scaling up during periods of high demand and scaling down during periods of low traffic. To ensure application performance, the AWS auto-scaling function monitors and modifies computing resources. By enabling auto-scaling for specific services and</p> 	<ul style="list-style-type: none"> <li data-bbox="813 808 1294 1503"> <p>• Cost Prohibitive: One of the primary drawbacks of AWS is that it might be too expensive for some companies, particularly smaller ones or startups with tight resources. Although the pay-as-you-go pricing approach might be reasonable in some circumstances, it can rapidly become expensive for resource-intensive apps or often-visited websites.</p> <li data-bbox="813 1525 1294 2007"> <p>• Usage is Not Facile: AWS is a strong platform with a vast selection of services, but it also has a high learning curve. The complexity of AWS might be challenging for consumers who are new to cloud computing or lack experience managing cloud infrastructure. Correct service</p>

optimizing resource allocation depending on traffic variations, it distinguishes it from other cloud platforms.	setup and configuration need technical expertise and a grasp of the AWS architecture, which may not be readily available to all users.
---	--

2.3.2 Google Cloud Platform (GCP)

Google Cloud Platform (GCP) is a suite of cloud computing services provided by Google. It provides a full range of tools and options for building, launching, and scaling applications, storing and processing data, and controlling cloud infrastructure. A trustworthy and secure infrastructure, a worldwide network, and cutting-edge machine-learning capabilities are all provided by GCP. GCP enables companies and developers to effectively use Google's technology to create, collaborate, and accelerate digital transformation thanks to its user-friendly interface and pay-as-you-go pricing model.

There are various benefits of using the GCP cloud services including well-authorized cloud computing, and live migration of virtual machines, however, it also comes with a few limitations which are safety, privacy bounded control, and flexibility. (Gupta, et al., 2021) Table 2-2 shows the benefits and limitations of GCP in detail.

Table 2-2: GCP's Benefits and Limitations

Benefits	Limitations
<ul style="list-style-type: none"> Well-Authorized in Cloud Computing: GCP reflects Google's reputation for constant innovation by providing state-of-the-art cloud computing services and solutions. The industry has long recognized and largely accepted GCP as a 	<ul style="list-style-type: none"> Safety and Privacy: Due to GCP employing robust security procedures, some businesses might be concerned about committing their sensitive information to a different cloud provider. Particularly in highly regulated businesses, they might wish to have complete control

<p>dependable and trustworthy cloud platform.</p> <ul style="list-style-type: none"> • Live Migration of Virtual Machines: Virtual machines may be shifted between physical hosts utilizing GCP's live migration feature without experiencing any downtime. This increases dependability and reduces interruptions during upkeep or hardware updates. 	<p>over their data storage and security.</p> <ul style="list-style-type: none"> • Bounded Control and Flexibility: The managed services and pre-configured environments offered by GCP are useful, but they could restrict the degree of customization and control needed by some organizations for particular workloads. It may be difficult to specifically customize GCP services to the demands of businesses with unusual requirements.
---	--

2.3.3 Microsoft Azure

Microsoft Azure, often referred to simply as Azure, is a comprehensive cloud computing platform offered by Microsoft. For creating, deploying, and administering applications and services in the cloud, it offers a broad range of services. Businesses can host their apps and data in various countries throughout the world thanks to Azure's global network of data centres, which improves performance and dependability. Azure meets the demands of developers, IT specialists, and businesses alike with a wide range of services, including computation, storage, databases, artificial intelligence, Internet of Things (IoT), and more. With its hybrid features, on-premises and cloud environments may be seamlessly integrated, giving organizations flexibility and continuity as they embark on their digital transformation journeys. For businesses of all sizes searching for a scalable and reliable cloud computing solution, Azure is a popular option because of its emphasis on security, compliance, and enterprise-grade support.

There are various benefits of using the Azure cloud services including Platform-as-a-Service (PaaS), and high-level availability, however, it also comes with a few limitations which are consequences with documentation, and high cost. (Maurya, et al., 2021) Tables 2-3 show the benefits and limitations of Azure in detail.

Table 2-3: Azure's Benefits and Limitations

Benefits	Limitations
<ul style="list-style-type: none"> <li data-bbox="304 645 788 1283"> <p>• Platform-as-a-Service (PaaS): With the help of Azure's PaaS solutions, developers can build, deploy, and scale applications in a secure and controlled environment without worrying about the underlying infrastructure. This streamlines the development process and frees developers from server administration to concentrate on creating apps.</p> <li data-bbox="304 1305 788 1783"> <p>• High-Level Availability: Azure has a solid and redundant infrastructure that guarantees high application and data availability. Microsoft is a dependable choice for essential workloads because of its Service Level Agreements (SLAs), which ensure high availability.</p> 	<ul style="list-style-type: none"> <li data-bbox="813 645 1286 1171"> <p>• Consequences with Documentation: Difficulties with Azure documentation might arise for some users. It could be insufficient or unclear, making it more difficult for IT teams or developers to comprehend and successfully implement particular functionalities.</p> <li data-bbox="813 1193 1286 1619"> <p>• High Cost: Even while Azure offers affordable solutions, some setups or resource-intensive applications could have greater expenses. Infrastructure needs to be properly planned by businesses to prevent unforeseen costs.</p>

2.3.4 Summary

In summary, cloud-based solutions play a crucial role in modern business operations, offering significant advantages such as cost-effectiveness, scalability, and flexibility. Among the leading cloud service providers, AWS stands out with its cost-effectiveness through pay-as-you-go pricing and efficient auto-scaling and maintenance features. But other businesses could find it prohibitively expensive and have trouble using it at first. Google Cloud Platform (GCP), on the other hand, is a well-known name in cloud computing and provides the advantage of live virtual machine migration, but potential restrictions in safety, privacy, and control may have an impact on some enterprises. Microsoft Azure dazzles with its solid Platform-as-a-Service (PaaS) and a high degree of availability, but consumers could have trouble with the documentation and think it's quite pricey. The choice of AWS ultimately comes down to individual organization requirements and goals, guaranteeing a well-informed decision to maximize the advantages of cloud-based services for long-term success and growth.

2.4 User Experience and Interface Design

2.4.1 User Experience (UX)

User experience (UX) is a comprehensive evaluation and interaction of people, systems, and the environment in which they are utilized, which might be a part of a group or individual phenomenon. According to Berni & Borgianni (2021, pp. 2-4) a study conducted to define the user experience, the literature identifies three basic kinds of experience—ergonomic, cognitive, and emotional—to define both individual and communal UXs.

Ergonomic Experience:

The ergonomic experience places significant value on a system's usability, efficiency, and affordances. The system must be simple to use and navigate in order for users to complete jobs effectively. While affordances relate to the system's design components that point users in the direction of particular interactions, effectiveness measures the system's capacity to accomplish its intended goals.

Cognitive Experience:

The cognitive experience involves how users interpret a system's appearance and exteriority. When interacting with the system, it takes into account the user's cognitive functions, including attention, memory, and problem-solving. The cognitive experience is significantly shaped by perception and aesthetic elements, which affect users' overall happiness and engagement with the system.

Emotional Experience:

Emotional experience covers the emotional elements of UX and takes into account the sentiments and emotions of users when they engage. This dimension takes both the emotional responses induced by the technology and the emotional characteristics that users emphasize. It includes hedonic elements like how enjoyable a system is to use and how strongly people feel about it. A system develops a stronger connection and sense of worth when it arouses good feelings in its users and enhances their personal development.

Figure 2-1 displays a comprehensive perspective of the UX based on the pillars or focal factors and experience type forms.

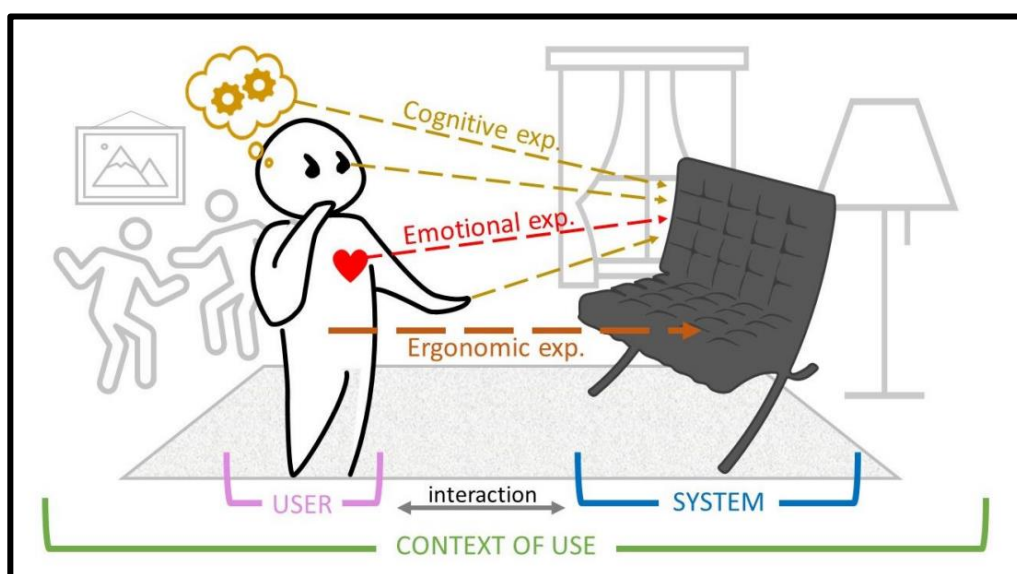


Figure 2-1: Three Elements of User Experience (Berni & Borgianni, 2021)

2.4.2 Interface Design Factors to Improve User Experience

User experience (UX) is heavily influenced by interface design in the competitive digital world of today. Designers may produce user interfaces that encourage effective interactions and arouse pleasant feelings by taking into account elements like usability, accessibility, visual appeal, and responsiveness. A user-friendly experience is aided by intuitive navigation, unambiguous call-to-action buttons, and inclusive design features, and the aesthetic appeal is improved by the careful use of colours and typefaces. Additionally, guaranteeing responsiveness across diverse platforms and devices provides smooth interactions for consumers. In this post, we'll examine the importance of these interface design elements and how they affect providing users with a memorable user experience that promotes organization success. Several factors may affect the user experience in interface design including general design factors, adaptive design factors, and individual design factors. (Li, et al., 2022)

General Design Factors:

This study's eye-tracking experiment provided insightful information on how users interact with website interfaces. The website logo had great relevance, according to the analysis of the heat maps, and immediately drew visitors' attention when they were given a job. The logo, which served as the main identifier, automatically attracted people's attention. Additionally, even if visitors frequently intentionally ignore copyright information, it is nevertheless crucial for legal reasons and to maintain the professionalism of the website as a whole. Website designers may optimize interface components to boost user engagement and provide a more enjoyable user experience by knowing these broad design principles and their effect on user attention. These conclusions offer helpful pointers for enhancing website design and making sure customers' preferences are taken into account, which will improve website usability and customer happiness.

Adaptive Design Factors:

Control components that are constrained and unable to be directly varied are referred to as adaptive design elements in the interface. Due

to changing screen sizes, resolutions, use circumstances, and user behaviours across various devices, these aspects are constrained in responsive website interfaces. To create a distinct design for each device while preserving consistency in visual recognition and user experience across responsive interfaces on the same website, designers must take these considerations into account. Numerous adaptable design elements, including the search tool, navigation bar, login tool, banner, content block, rapid navigation, interface layout, and interface density, were revealed by the testing findings. As an illustration, the design of the search tool changed from a complete style to a landscape icon with text style, then to a vertical icon with text style as the horizontal resolution.

Individual Design Factors:

Individual design factors are characteristics of a terminal's design that cannot simply be applied to other devices. These components must be applied in fundamentally different ways across distinct terminals due to these considerations. Only the tab bar control for smartphones was found in the T01 series interface in the evaluation of six representative samples, making it difficult to extract rich design features directly from the static interface. The researchers examined 40 first-generation samples to overcome this issue, then integrated their findings with information gleaned from subject and site designer interviews. Through this method, personalized design elements particular to each terminal were found, resulting in a more specialized and improved user experience on those devices.

2.4.3 Summary

In summary, interface design has a significant impact on user experience (UX) in the digital world. By taking usability, accessibility, aesthetic appeal, and responsiveness into account, designers may improve interactions and arouse good emotions. A user-friendly experience is facilitated by intuitive navigation, unambiguous call-to-action buttons, inclusive design, and aesthetic decisions. User interactions are made easy by ensuring responsiveness across platforms.

The eye-tracking experiment highlighted how crucial it is for websites to include a logo and copyright information to draw people in. In order to improve interface components and increase user engagement and happiness, it might be helpful to understand general, adaptable, and specific design considerations. In order to deliver a memorable and pleasurable user experience, which ultimately contributes to the success of organizations and websites, interface design aspects must be carefully considered.

2.5 Software Development Methodology

2.5.1 Waterfall Software Development Methodology

The Waterfall software development methodology is a linear and sequential approach to software development. Each phase must be finished before going on to the next in a planned and disciplined procedure. Because it moves gradually downward through the many phases, the process is called "Waterfall" because that is how waterfalls flow. The typical phases in the Waterfall methodology are included in Figure 2-2:

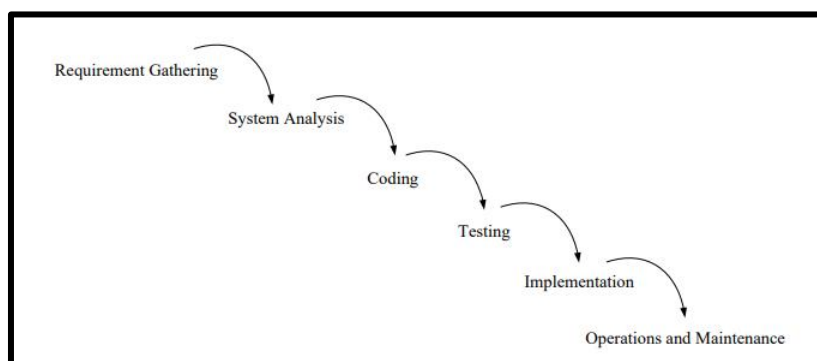


Figure 2-2: Waterfall Software Development Methodology's phases
(Mohammad, et al., 2019)

In software development, the waterfall methodology is one of the most recognized and used SDLC methods. Clear structure and non-overlapping are the benefits of waterfall software development methodology, while lengthy development time over budget, and deep research required are the limitations of waterfall software development methodology. (Rachma & Muhlas, 2022) Table 2-4 are the benefits and limitations of the waterfall.

Table 2-4: Benefits and Limitations of Waterfall Software Development Methodology

Benefits	Limitations
<ul style="list-style-type: none"> • Clear Structure: It gives the software development process a very clear, well-defined framework. The specified deliverables and milestones for each phase make it simpler to plan and keep track of development. • Non-Overlapping: It does not contain any processes to cross across. 	<ul style="list-style-type: none"> • Lengthy Development Time and Over Budget: It's possible that faults or problems with the program won't be discovered until testing, or even later. Testing takes place at the end of the development cycle, therefore fixing these problems may require time-consuming and expensive rework. • Deep Research Required: Early on in the phases, it gathers comprehensive data and makes informed decisions.

2.5.2 DevOps Software Development Methodology

DevOps is a software development methodology that aims to enhance teamwork and communication between teams working on development and operations. In order to facilitate quicker and more dependable software development and deployment, it seeks to establish a culture of shared accountability, continuous integration, and continuous delivery. The typical phases in the DevOps software development methodology are included in Figure 2-3:

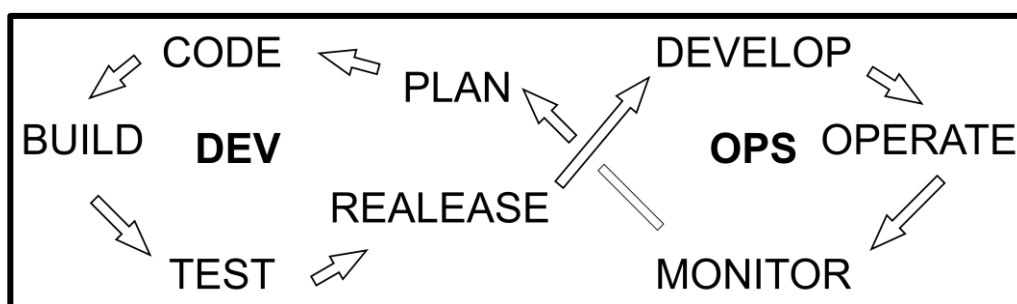


Figure 2-3: DevOps Methodology's phases (Business Transformation Career Advice Database DevOps Observability, 2022)

Faster time to market, and increased deployment frequency are the benefits of waterfall software development methodology, while cultural shift challenges, initial investment, and resource requirements are the limitations of the DevOps software development methodology. (Mayank & Raju, 2021) Table 2-5 are the benefits and limitations of the DevOps software development methodology.

Benefits	Limitations
<ul style="list-style-type: none"> • Faster Time to Market: Automation, collaboration, and continuous delivery are emphasized in DevOps practices. Therefore, reduced the time to bring new features to the market • Increased Deployment Frequency: Continuous Integration and Continuous Delivery (CI/CD) practices in DevOps. Developers may quickly deliver software upgrades to production environments and often integrate and test code changes. 	<ul style="list-style-type: none"> • Cultural Shift Challenges: Many organizations have established, deeply ingrained procedures that may not be compatible with the collaborative, iterative nature of DevOps. • Initial Investment and Resource Requirements: The lack of enough automation to support continuous testing.

Table 2-5: Benefits and Limitations of DevOps Software Development Methodology

2.5.3 Rapid Application Development Methodology

Rapid Application Development (RAD) is a software development methodology that focuses on rapid prototyping and iterative development to deliver software solutions quickly. It focuses a strong emphasis on user participation, iterative feedback, and adaptable development techniques. By reducing planning and increasing real development time, RAD attempts to

shorten the development lifecycle. The flow of the Rapid Application Development software development methodology is included in Figure 2-4:

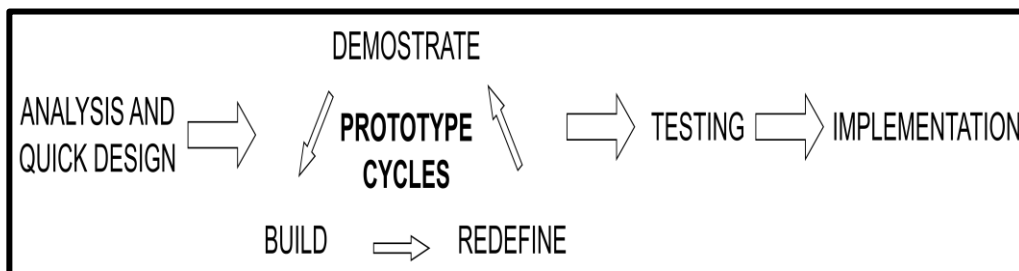


Figure 2-4: Rapid Application Development Methodology's phases (kissflow, 2023)

There are several benefits and limitations to the Rapid Application Development Methodology, the main focus of benefits will be enhanced user involvement, flexibility, and adaptability, on the other hand, the limitation that focus is the lengthy development time over budget and high dependency on user availability. (Hasani, et al., 2023) Table 2-6 is the detail of the benefits and limitations of the Rapid Application Development Methodology.

Table 2-6: Benefits and Limitations of Rapid Application Development Methodology

Benefits	Limitations
<ul style="list-style-type: none"> • Enhanced User Involvement: Direct user participation is emphasized by RAD throughout the development process. Users get the chance to offer feedback, confirm requirements, and mould the system to suit their needs through repeated prototyping and feedback sessions. • Flexibility and Adaptability: RAD is intended to be flexible to changing requirements. Each iteration enables modifications 	<ul style="list-style-type: none"> • Lengthy Development Time and Over Budget: Since RAD is iterative, scope creep is a possibility if effective scope control is not included. Iterations that often update and add to the original requirements might have an impact on deadlines and resources. • High Dependency on User Availability: RAD primarily depends on user feedback and participation. Users

and improvements, ensuring that the system develops in response to user input and changing business requirements.	must thus be available and involved actively throughout the development process.
---	--

2.5.4 Summary

DevOps benefits include faster time to market and more frequent deployments, which are in line with the objective of time and cost control. Waterfall's extensive development time, cost overruns, and need for in-depth research may make it difficult to manage time and money. Although the flexibility and user engagement of RAD are advantageous, the lengthy development period and heavy reliance on user availability might be risky. In order to facilitate collaboration, automation, and iterative development for quicker releases, lower cost risks, and more adaptability, DevOps achieves a balance. DevOps is a good option for time and cost management in the project, but certain long-term challenges need to be addressed, such as culture change challenges and early investment challenges.

2.6 Similar Existing Web Applications

2.6.1 Performio Application

Performio is a tool aimed at assisting companies in optimising their sales commission process and enabling data-driven decision-making. The commission management software, which has a 93% customer retention rate, provides fast-growing enterprises with deep data to propel robust sales success. Table 2-7 is the detail of the benefits and limitations of the Performio Application.

Table 2-7: Benefits and Limitations of Performio Application

Benefits	Limitations
<ul style="list-style-type: none"> Efficiency: By automating sales performance management procedures, Performio helps sales managers and 	<ul style="list-style-type: none"> Complexity: Performio can be difficult to implement and configure, particularly for businesses with unusual pay or sales systems.

<p>administrators save time and handle-less administrative work.</p> <ul style="list-style-type: none"> • Accuracy: Performio assists in ensuring accuracy in commission payment and performance reporting by centralising sales data and computations. Lowering the possibility of mistakes and conflicts promotes openness and trust among the sales team. 	<p>Significant time and money may be needed for customisation, and continuing maintenance may be required to keep up with changes in the corporate environment.</p> <ul style="list-style-type: none"> • Integration Challenges: It may appear difficult to integrate Performio with other systems, such as accounting software or CRM platforms. Maintaining synchronisation may need further development effort and continuous assistance in order to ensure smooth data flow between systems.
--	--

2.6.2 CaptivateIQ Application

CaptivateIQ is an innovative platform that helps companies automate the administration, calculation, reporting, analysis, and management of even the most complex, variable-based commission plans. It centralizes and enriches data from disparate data sources into one powerful platform while also providing unbeatable flexibility and scalability to meet the needs of the organization. Table 2-8 is the detail of the benefits and limitations of the CaptivateIQ Application.

Table 2-8: Benefits and Limitations of CaptivateIQ Application

Benefits	Limitations
<ul style="list-style-type: none"> • Ease of Use: The straightforward design and user-friendly interface of CaptivateIQ are well-known. It facilitates the setup and administration of intricate commission systems, hence 	<ul style="list-style-type: none"> • Integration Complexity: It can occasionally be difficult to integrate CaptivateIQ with other systems, like as accounting software or CRM platforms. Additional development effort and continuing maintenance may

<p>simplifying the system's navigation and configuration for sales managers and administrators.</p> <ul style="list-style-type: none"> • Flexibility: The platform provides a great deal of customisation options for commission plans. Companies may modify commission structures to fit various sales roles, performance measures, and incentive schemes in order to better meet their own sales goals and objectives. 	<p>be necessary to ensure smooth data flow and synchronisation across systems.</p> <ul style="list-style-type: none"> • Scalability Challenges: Although CaptivateIQ is appropriate for a wide range of firms, scalability issues may arise for larger organisations with sizable sales teams and intricate commission arrangements. System performance might be strained while handling massive data quantities and performing computations for several sales representatives, necessitating careful optimisation.
--	---

2.6.3 Everstage Application

Everstage is a sales commission software that uses incentives to help the organization achieve the correct business results. Even while sales commissions are a significant corporate expenditure, it frequently underutilised as a revenue lever. Everstage supports profitable growth and helps the organization improve its incentive programme. Table 2-9 is the detail of the benefits and limitations of the Everstage Application.

Table 2-9: Benefits and Limitations of Everstage Application

Benefits	Limitations
<ul style="list-style-type: none"> • Improved Sales Motivation: Everstage offers timely transparency about commissions and performance statistics. Sellers are kept motivated and goal-focused by being able to 	<ul style="list-style-type: none"> • Learning Curve: Even though Everstage has a no-code interface, administrators may need technical knowledge or training to configure intricate incentive schemes or integrate them with current CRM systems.

<p>observe exactly how their efforts convert into revenue.</p> <ul style="list-style-type: none"> • Increased Sales Effectiveness: Everstage enables companies to customise their pay structure to encourage particular sales behaviours by providing a variety of incentive schemes and transparent progress reporting. 	<ul style="list-style-type: none"> • Customization: Even though Everstage gives designers considerable latitude in creating incentive schemes, it might not be appropriate for really distinctive commission structures with very particular specifications.
--	--

2.6.4 Summary

Performio, CaptivateIQ, and Everstage are three sales commission management applications designed to streamline and optimize the sales performance process for businesses. Performio provides commission management with precision and efficiency, however integration and execution may be more difficult. Although CaptivateIQ is flexible and easy to use, bigger organisations may find it difficult to integrate and scale. Although there may be a learning curve and it may not be appropriate for very specialised commission structures, Everstage focuses on increasing sales motivation and performance through transparent commission reporting and customisable incentive schemes. All things considered, these apps provide useful resources to businesses hoping to increase sales through efficient commission administration and informed decision-making.

2.7 Overall Summary

In summary, in this project Agent and Reseller System will capitalize on the Agent and Reseller System and will incorporate a diverse range of commission types, including the Direct Commission, Indirect Commission, Leadership Commission, and Pool Commission, to incentivize and reward resellers. These commission structures are strategically designed to encourage resellers to actively promote and sell the products or services, ultimately boosting sales and expanding the market presence. Cloud-based solutions, in particular AWS, GCP, and Microsoft Azure, provide a number of advantages, including cost-

effectiveness, scalability, and high availability. However, this project is going to use AWS. By emphasizing usability, accessibility, aesthetics, and responsiveness, interface design plays a significant part in improving user experience. The DevOps technique gives project development advantages in terms of time and money. This project seeks to create effective sales processes, outstanding user experiences, and long-term organization success through the use of Special Agents, cloud solutions, interface design optimization, and DevOps practices. Through optimised commission management and data-driven decision-making strategies, the integration of sales commission management applications such as Performio, CaptivateIQ, and Everstage further augments the efficacy of sales processes and contributes to long-term organisational success.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This chapter begins with going through each step in the DevOps process. In order for the project to be completed successfully on schedule, a Work Breakdown Structure (WBS) and Gantt Chart are also produced. This chapter concluded with a discussion of the employed development tools.

3.2 Software Development Methodology

DevOps was chosen for this project as the software development methodology due to it placing a high value on automation and continuous delivery. DevOps seeks to produce high-quality software more quickly and consistently than traditional methods. The phase of the DevOps methodology is shown in Figure 3-1.

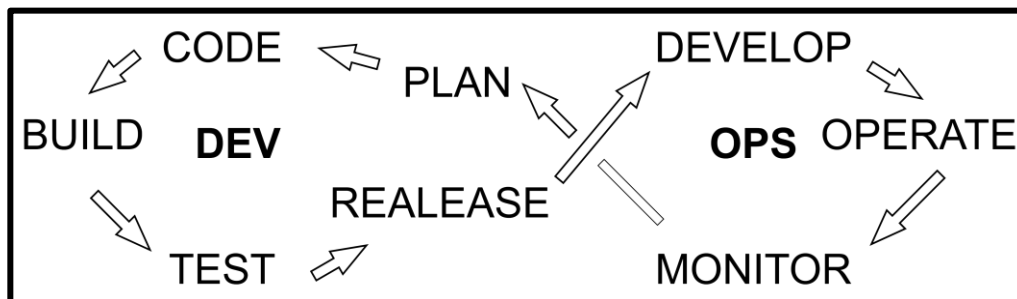


Figure 3-1: DevOps Methodology (Business Transformation Career Advice Database DevOps Observability, 2022)

3.2.1 Plan

This project will be building the system on top of the Gather Deal system. Therefore, the collection of data and requirements from the Enlliance Management Sdn. Bhd which is the owner of the Gather Deal system uses the information to form the functional and non-functional requirements for this project. Then, use case modelling can be developed according to the functional requirements. The relevant development tools are also decided in this phase, such as Node.js environment, React.js library, Next.js, and Django framework. Related

services which are AWS and GitHub services. Last but not least, WBS and Gantt charts are made to guarantee that the project can be completed within the allotted time.

3.2.2 Code

This project's code development phase may be split into three primary sections. The development and execution of web apps are covered in the first section utilizing React, Next.js, and JavaScript ES6. The project started applying AWS services and GitHub in the second stage to construct the complete CI/CD process so that developers could just issue the command `git push` to redeploy the full web project on AWS EC2, Elastic Beanstalk with Django and Node.js. The development of the storage for Amazon RDS and Amazon S3 to house the data is the final step. At this point, the key components of online and mobile apps are also being created, including the User Management Module, Agent and Reseller Registration Module, Agent and Reseller Tier Structure Module, Commission Management Module, and Sales, Analytics, and Reporting Module.

3.2.3 Build

This web project uses Django and Next.js as the backend servers to serve the React web application. The main build tools are `node.js`. React project will generate static resources such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript ES6 after using the `run build` command, and these static resources can be hosted on the backend server. The Django server can then be built into a standalone web application and integrated into a larger system architecture depending on the specific requirements. This project utilizes GitHub services to automate the building process in order to support CI/CD flow. As long as the developer has a new code update in the GitHub repository, the GitHub workflow will be automatically run, and the web project will be built into Vercel and manually updated by the developer to the backend server.

3.2.4 Test

To ensure the system satisfies user requirements and functions beyond the expected working environment smoothly, the testing phase consists of unit testing, system usability testing, and on-site testing.

3.2.5 Release and Deploy

GitHub is used throughout the release and deployment development process to automatically create and deploy web applications. The exact process is designating AWS EC2 servers as GitHub Runners so that they may carry out Github's workflow. Then, define a CI/CD pipeline by Github, utilize the updated project code to rebuild the code, and execute it in the EC2 environment. By entering the project's domain name in the web browser, the user may access the web application. Additionally, this cloud service may automatically update the portion of the GitHub repository after utilizing Vercel to develop the React webpage. Lastly, since the project will be used in commercials the release of the project will be released starting with the beta version phase by phase.

3.2.6 Operate

In the operate phase, to make sure ensure everything is operating as planned, EC2 machines and the infrastructure they operate on are monitored using AWS' CloudWatch services. CloudWatch will attempt to reboot the failed server if the EC2 server's status check is unsuccessful. Furthermore, Vercel leverages a global content delivery network (CDN) to distribute the web application to edge locations worldwide. This helps improve website performance by serving content from the nearest location to the user, reducing latency.

3.2.7 Monitor

Data and feedback from users from the monitoring stage are useful for enhancing the apps over time. An email will be sent to the developer and the EC2 service will be discontinued when the project's budget

exceeds a predetermined level, according to the project's implementation of the AWS Budgets service.

3.3 Project Planning and Scheduling

3.3.1 Work Breakdown Structure (WBS)

1. Develop a system for agents and resellers in GATHER DEAL project
 - 1.1. Project Initialization
 - 1.1.1. Preliminary Planning
 - 1.1.1.1. Understand Project Background
 - 1.1.1.2. Identify Existing Issues
 - 1.1.1.3. Identify Project Objective
 - 1.1.1.4. Determine Project Approach
 - 1.1.1.5. Identify Project Scope
 - 1.1.1.5.1. Define Targeted User
 - 1.1.1.5.2. Determine Modules Covered
 - 1.1.2. Literature Review
 - 1.1.2.1. Review Type of Agent
 - 1.1.2.2. Review Type of Commission
 - 1.1.2.3. Review Cloud-based System
 - 1.1.2.4. Study User Experience and Interface Design
 - 1.1.2.5. Review Software Development Methodology
 - 1.1.3. Methodology and Work Plan
 - 1.1.3.1. Identify Project's Software Development Methodology
 - 1.1.3.2. Develop Work Breakdown Structure
 - 1.1.3.3. Develop Gantt Chart
 - 1.1.3.4. Determine Software Development Tools
 - 1.1.4. Requirement Identification
 - 1.1.4.1. Identify Requirement Specification
 - 1.1.4.1.1. Define Functional Requirements

- 1.1.4.1.2. Define Non-Functional Requirements
- 1.1.4.2. UML Modelling
 - 1.1.4.1.1. Develop use case Diagram
 - 1.1.4.1.2. Develop use case Description
- 1.1.5. Develop Prototype
- 1.2. System Development
 - 1.2.1. System Design
 - 1.2.1.1. Database design
 - 1.2.1.2. Web Application Architecture Design
 - 1.2.1.3. Cloud Architecture Design
 - 1.2.2. System Implementation
 - 1.2.2.1. User Management Module
 - 1.2.2.2. Agent and Reseller Registration Module
 - 1.2.2.3. Agent and Reseller Tier Structure Module
 - 1.2.2.4. Commission Management Module
 - 1.2.2.5. Sales Tracking, Analytics, and Reporting Module
- 1.3. System Testing
 - 1.3.1. Unit Testing
 - 1.3.2. System Usability Testing
 - 1.3.3. Beta Testing
- 1.4. Project Closure
 - 1.4.1. Live System Monitoring
 - 1.4.2. Project Document Finalize
 - 1.4.3. Project Evaluation

3.3.2 Gantt Chart

Tasks	Duration	Start Date	End Date
1. Develop a system for agents and resellers in GATHER DEAL project	162d	06/19/23	04/19/24
+ 1.1. Project Initialization	81d	06/19/23	09/08/23
+ 1.2. System Development	46d	01/29/24	03/15/24
+ 1.3. System Testing	20d	03/16/24	04/05/24
+ 1.4. Project Closure	14d	04/06/24	04/19/24

Figure 3-2: Project Schedule Overview

Tasks	Duration	Start Date	End Date
▣ 1.1. Project Initialization	82d	06/19/23	09/08/23
▣ 1.1.1. Preliminary Planning	12d	06/19/23	06/30/23
1.1.1.1. Understand Project Background	5d	06/19/23	06/23/23
1.1.1.2. Identify Existing Issues	5d	06/19/23	06/23/23
1.1.1.3. Identify Project Objective	2d	06/24/23	06/25/23
1.1.1.4. Determine Project Approach	2d	06/24/23	06/25/23
▣ 1.1.1.5. Identify Project Scope	5d	06/26/23	06/30/23
1.1.1.5.1. Define Targeted User	2d	06/26/23	06/27/23
1.1.1.5.2. Determine Modules Covered	4d	06/27/23	06/30/23
▣ 1.1.2. Literature Review	25d	07/01/23	07/25/23
1.1.2.1. Review Type of Agent	10d	07/01/23	07/10/23
1.1.2.2. Review Type of Commission	10d	07/01/23	07/10/23
1.1.2.3. Review Cloud-based System	10d	07/11/23	07/20/23
1.1.2.4. Study User Experience and Interface Design	10d	07/11/23	07/20/23
1.1.2.5. Review Software Development Methodology	5d	07/21/23	07/25/23
▣ 1.1.3. Methodology and Work Plan	15d	07/26/23	08/09/23
1.1.3.1. Identify Project's Software Development Methodology	2d	07/26/23	07/27/23
1.1.3.2. Develop Work Breakdown Structure	6d	07/28/23	08/02/23
1.1.3.3. Develop Gantt Chart	2d	08/03/23	08/04/23
1.1.3.4. Determine Software Development Tools	5d	08/05/23	08/09/23
▣ 1.1.4. Requirement Identification	30d	08/10/23	09/08/23
▣ 1.1.4.1. Identify Requirement Specification	14d	08/10/23	08/23/23
1.1.4.1.1. Define Functional Requirements	9d	08/10/23	08/18/23
1.1.4.1.2. Define Non-Functional Requirements	5d	08/19/23	08/23/23
▣ 1.1.4.2. UML Modelling	16d	08/24/23	09/08/23
1.1.4.2.1. Develop use case Diagram	8d	08/24/23	08/31/23
1.1.4.2.2. Develop use case Description	10d	08/30/23	09/08/23
1.1.5. Develop Prototype	5d	09/03/23	09/08/23

Figure 3-3: Project Initialization Schedule

Tasks	Duration	Start Date	End Date
▣ 1.2. System Development	48d	01/29/24	03/17/24
▣ 1.2.1. System Design	10d	01/29/24	02/07/24
1.2.1.1. Database design	5d	01/29/24	02/02/24
1.2.1.2. Web Application Architecture design	4d	02/03/24	02/06/24
1.2.1.3. Cloud Architecture Design	3d	02/05/24	02/07/24
▣ 1.2.2. System Implementation	38d	02/08/24	03/18/24
1.2.2.1. User Management Module	12d	02/08/24	02/19/24
1.2.2.2. Agent and Reseller Registration Module	12d	02/08/24	02/19/24
1.2.2.3. Agent and Reseller Tier Structure Module	12d	02/20/24	03/04/24
1.2.2.4. Commission Management Module	16d	02/20/24	03/07/24
1.2.2.5. Sales Tracking, Analytics and Reporting Module	12d	03/07/24	03/18/24

Figure 3-4: System Development Schedule

Tasks	Duration	Start Date	End Date
1.3. System Testing	20d	03/18/24	04/06/24
1.3.1. Unit Testing	5d	03/18/24	03/22/24
1.3.2. System Usability Testing	5d	03/23/24	03/27/24
1.3.3. Beta Testing	10d	03/28/24	04/06/24

Figure 3-5: System Testing Schedule

Tasks	Duration	Start Date	End Date
1.4. Project Closure	14d	04/07/24	04/20/24
1.4.1. Live System Monitoring	7d	04/07/24	04/13/24
1.4.2. Project Document Finalize	7d	04/14/24	04/20/24
1.4.3. Project Evaluation	5d	04/16/24	04/20/24

Figure 3-6: Project Closure Schedule

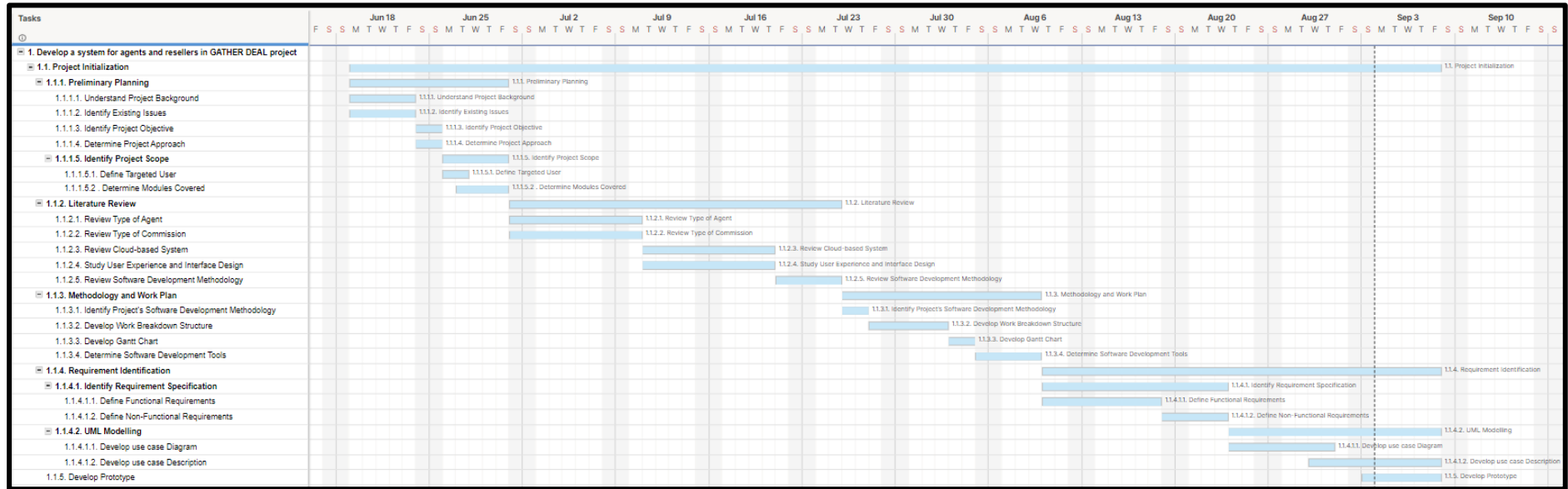


Figure 3-7: Project Initialization Gantt Chart

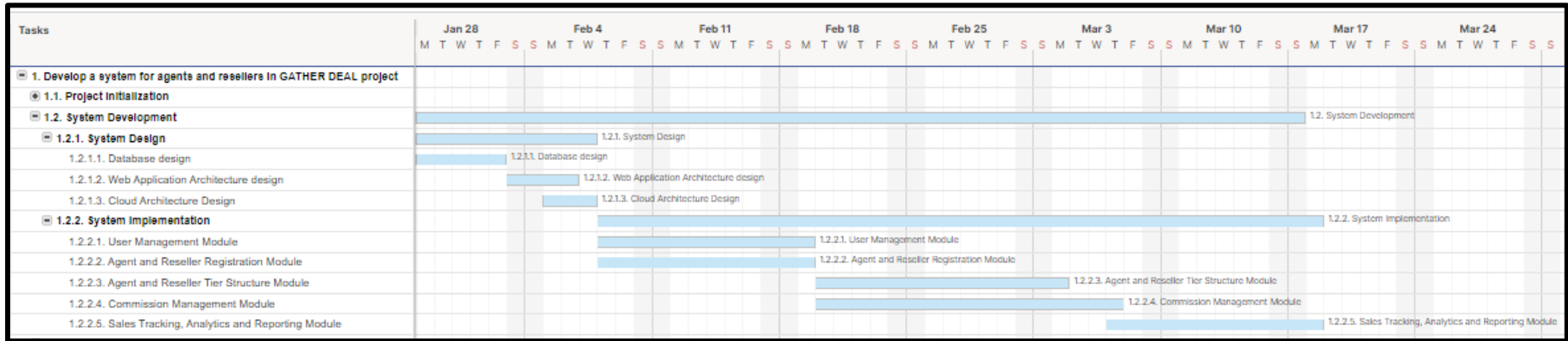


Figure 3-8: System Development Gantt Chart

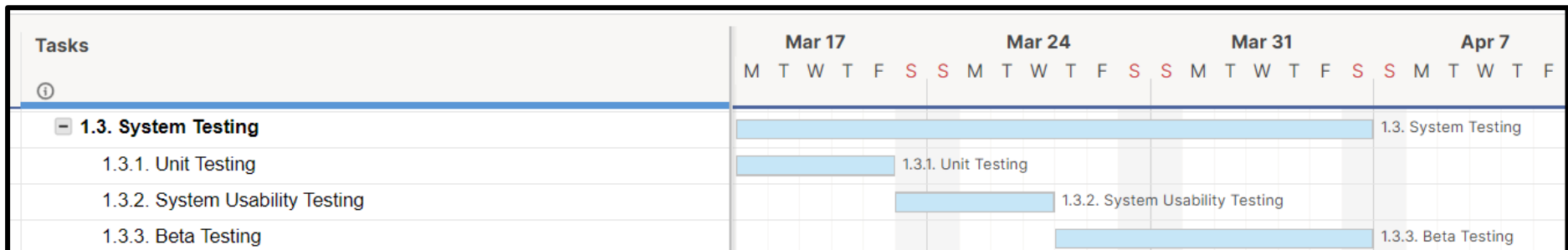


Figure 3-9: System Testing Gantt Chart

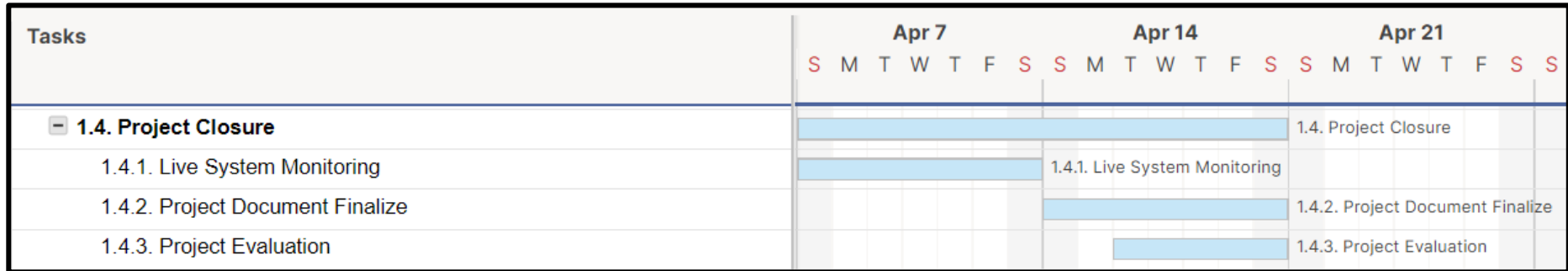


Figure 3-10: Project Closure Gantt Chart

3.4 Technologies and Development Tools

3.4.1 Vercel

Vercel is a cloud-based web development platform that makes deployment and hosting easier. It uses a worldwide CDN for quick content delivery, simplifies deployment, and enhances performance with tools like static site creation and server-side rendering. Vercel delivers real-time analytics for monitoring and troubleshooting, is scalable quickly, interacts with well-known frameworks, offers collaboration tools like preview deployments, and offers collaboration tools like preview deployments. It's a strong and convenient platform for efficient web development.

3.4.2 React

React is a powerful and popular JavaScript library used for building user interfaces in web applications. By establishing a component-based design that enables developers to construct reusable and modular UI elements, Facebook's React has completely changed front-end development. React offers great efficiency and responsiveness with its effective virtual DOM rendering and one-way data flow, making it the perfect solution for creating interactive and dynamic online apps. Its declarative syntax streamlines the development process and encourages code reuse by making it easier to manage the state and update the user interface as data changes. React has benefited from significant community support and acceptance as a result, becoming the preferred choice for developers aiming to create cutting-edge, scalable, and user-friendly online apps.

3.4.3 Node.js

Node.js is a JavaScript runtime environment that allows developers to run JavaScript code on the server side using Node.js, a JavaScript runtime environment. Building scalable network applications is facilitated by its event-driven, non-blocking I/O approach. Developers may create server-side apps using Node.js utilizing JavaScript, which

is frequently used for front-end web development. It is a potent tool for creating web servers, APIs, real-time applications, and more since it has a vast selection of built-in modules and a robust ecosystem of third-party libraries. Node.js is well-suited for high-performance and scalable applications due to its capacity to manage several concurrent connections and its effective handling of asynchronous activities.

3.4.4 Django

Django is a popular and powerful web framework for building web applications using the Python programming language. It offers a structured and effective approach to web development, allowing programmers to create reliable and scalable systems fast. Django adheres to the "batteries included" tenet by providing a broad variety of integrated features and capabilities to perform typical web development chores including URL routing, database administration, form handling, and user authentication. Django supports best practices, security, and code reuse with its clear and practical architecture. It's a great option for quickly and effectively creating web apps since it has a strong community, substantial documentation, and a tonne of third-party packages.

3.4.5 Next.js

Next.js is a popular framework for building server rendered React applications. With capabilities like server-side rendering, automated code splitting, and streamlined routing, it makes it easier to design complicated online apps. By pre-rendering pages and providing them as static HTML files or creating them on the server, Next.js improves speed. It provides an easy-to-use API, hot module replacement for rapid changes while developing, and seamless React component integration. By utilizing the advantages of React's component-based design and a large ecosystem of tools and modules, developers can quickly and simply build quick and scalable online apps with Next.js.

3.4.6 AWS EC2 (Elastic Compute Cloud)

Amazon EC2 is a cloud computing service provided by Amazon Web Services (AWS). It enables customers to lease cloud instances, which are virtual servers. Users of EC2 have complete control over the operating system, storage, and networking settings for their virtual server instances. Users may quickly scale up or down their computing resources using EC2's flexibility and scalability, depending on their needs. It is a well-known service for hosting apps, managing diverse workloads, and doing large-scale calculations in the cloud. For deploying virtual server instances in the cloud, Amazon EC2 offers a solid and adaptable foundation.

3.4.7 AWS Elastic Beanstalk

Elastic Beanstalk is a service provided by AWS that simplifies the deployment and management of web applications. Developers can concentrate on developing code rather than running infrastructure thanks to it. Uploading application code to Elastic Beanstalk, and the service will take care of deploying and configuring the required resources, including EC2 instances, load balancers, and databases. It offers a versatile environment for launching web applications and supports a number of different programming languages and platforms. Scaling, monitoring, and health checks are just a few of the services that Elastic Beanstalk provides, making it simple to manage traffic variations and guarantee high availability. It is a practical method for setting up and controlling web apps in the AWS cloud.

3.4.8 AWS RDS (Relational Database Service)

Amazon RDS is a managed database service provided by Amazon Web Services (AWS). Setting up, running, and growing relational databases in the cloud is made simpler. Developers may select from a variety of database engines with Amazon RDS, including MySQL, PostgreSQL, Oracle, and SQL Server, and AWS manages the infrastructure and operational chores needed to host a database. This contains features for

scalability, monitoring, and automated backups as well as software patching. Developers can concentrate on their applications without having to worry about the underlying database architecture thanks to Amazon RDS, which offers a trustworthy and highly available database solution.

3.4.9 AWS S3 (Simple Storage Service)

Amazon S3 is a cloud-based storage service provided by Amazon Web Services (AWS). Large volumes of data may be stored and retrieved by users in a safe, enduring, and highly scalable way. Users may upload and store a variety of file types using Amazon S3, including backups, pictures, videos, and documents. The S3 service's "buckets" are how these files are arranged. S3 delivers capabilities like versioning, access control, data encryption, and automated data replication across many locations, all through a user-friendly and straightforward API. It is frequently used as a backend storage option for online applications as well as for storing static files, distributing information, and backing up data. Designed to be extremely dependable and affordable, Amazon S3 offers companies and developers a scalable and trustworthy storage option for their data needs.

3.4.10 GitHub

GitHub is a platform for hosting and managing software projects using version control. It enables developers to manage project repositories, track changes, and collaborate on code. Developers may put their code in a centralized repository using GitHub, which makes it simple to monitor changes, work with others, and keep track of project developments. It provides tools for effective collaboration and code quality management, including bug tracking, pull requests, and code reviews. Developers may also promote their work on GitHub, donate to open-source software, and find new code repositories. In conclusion, GitHub is a crucial platform for developers to organize, communicate, and share their work.

3.4.11 ES6

The ECMAScript standard, which serves as the blueprint for JavaScript, has undergone six significant revisions. This version is known as ES6, or ECMAScript 6. When ES6 was released in 2015, a wide range of new features, syntactic changes, and language refinements were made, making the language more potent, expressive, and developer friendly. Arrow functions, classes, let and const for block-scoped variables, template literals for more adaptable string interpolation, and improved object literals are a few notable features. Modules were another feature of ES6, which made it simpler to arrange and manage JavaScript code. By enabling developers to create shorter, easier-to-maintain code, it has greatly improved the development experience, resulting in more productive developers and better functionality in current JavaScript apps.

3.5 Summary

The project uses a variety of technologies and development tools and sticks to the DevOps methodology. The DevOps process is broken down step by step in the first section of the chapter to ensure that the project is completed successfully and on schedule. A Work Breakdown Structure (WBS) and Gantt Chart are made to help with project management, giving an organized overview of the activities and their scheduling. The used development tools are covered in the chapter's conclusion, including GitHub, Amazon EC2, Elastic Beanstalk, React, Node.js, Django, Next.js, and ES6. Together, these technologies make it possible to deploy software quickly, create cross-platform mobile apps, design server-side and front-end applications, manage databases, and maintain code versions. The project's objectives and needs are met by their selection, guaranteeing the project's successful completion and delivery of the necessary functionality.

CHAPTER 4

PROJECT SPECIFICATION

4.1 Introduction

This section covers the use of stakeholder consultations to find out system user requirements. The main stockholder of this project is Enlliance Management Sdn. Bhd., therefore, requirements will be mainly based on their needs. The requirement specification, use case diagram, and use case description is created by analysing the data that has been gathered. The requirements document is further split into functional and non-functional requirements to accurately identify user requirements.

4.2 Fact Finding

The fact-finding of this project will be carried out in the form of stakeholder consultations.

4.2.1 Stakeholder Consultations

The findings from stakeholder consultations play a vital role in shaping the project requirements. These discussions offer priceless viewpoints and ideas from people and groups who will be directly touched by the project. The opinions and preferences of important stakeholders, such as users, clients, team members, and experts, have been gathered through a series of discussions and interactions.

These conversations have produced a number of remarkable observations. First and foremost, stakeholders highlighted the need for a user-friendly interface that facilitates quick navigation and natural interactions. This is in line with the larger objective of improving accessibility and user experience. The significance of real-time data access for agents and resellers to efficiently manage their sales and commissions was also emphasized by stakeholders. This emphasizes how important it is to get and integrate data in an effortless manner.

Stakeholders also showed a significant interest in varied compensation schemes that encourage resellers to boost sales and broaden their customer base. This comment emphasizes how important it is to have several incentive kinds, such as Direct Commissions, In-direct Commissions, Leadership Commission, and Pool Commissions to accommodate various motivating factors.

Overall, the project's objectives, scope, and suggested solution have all benefited from the conclusions from stakeholder engagements. Stakeholders' insightful comments helped shape the creation of a thorough Agent and Reseller System that meets their needs and is in line with the objectives of the organization.

Relevant discussion of stakeholder consultations in the appendix part as [Appendix A](#).

4.3 Functional Requirements

4.3.1 User Management Module

Table 4-1: User Management Module Functional Requirements

ID	Functional Requirement
FR01	The system shall allow the reseller to login before using the system.
FR02	The system shall allow the administrators to view a comprehensive list of resellers affiliated with the organization.
FR03	The system shall allow the administrators to be able to update the organization's reseller.

4.3.2 Agent and Reseller Registration Module

Table 4-2: Agent and Reseller Registration Module Functional Requirements

ID	Functional Requirement
FR04	The system shall allow the administrator to invite new resellers to join the platform.
FR05	The system shall allow the new reseller to register an account inside the system upon receiving an invitation link.

FR06	The system shall allow the new reseller to view the organization's reseller agreement.
FR07	The system shall allow the new reseller to respond (accept or reject) to the organization's reseller agreement.

4.3.3 Agent and Reseller Tier Structure Module

Table 4-3: Agent and Reseller Tier Structure Module Functional Requirements

ID	Functional Requirement
FR08	The system shall allow the administrator to set the naming of the tier for each reseller.
FR09	The system shall categorize each reseller based on their tier.
FR10	The system shall allocate each reseller downline.
FR11	The system shall allow the administrator to view all resellers in a diagram.

4.3.4 Commission Management Module

Table 4-4: Commission Management Module Functional Requirements

ID	Functional Requirement
FR12	The system shall allow the administrator to allocate different commission rates for each tier.
FR13	The system shall calculate the Direct Commission of each reseller based on tier and downline.
FR14	The system shall calculate the Indirect Commission of each reseller based on tier and downline.
FR15	The system shall calculate the Leadership Commission of each reseller based on tier and downline.
FR16	The system shall calculate the Pool Commission of each reseller based on tier, downline, and performance.

FR17	The system shall allow each reseller to view the total amount of the commission wallet.
FR18	The system shall allow each reseller to withdraw from the commission wallet to the withdrawer wallet.
FR19	The system shall allow each reseller to view the total amount of the withdrawer wallet.
FR20	The system shall allow each reseller to view the commission transaction history.
FR21	The system shall allow the administrators to view the commission withdraw request list of resellers.
FR22	The system shall allow the administrators to upload the transaction slip number for the commission withdraw request.

4.3.5 Sales Tracking, Analytics, and Reporting Module

Table 4-5: Sales Tracking, Analytics, and Reporting Module Functional Requirements

ID	Functional Requirement
FR23	The system shall allow each reseller to view their own sales overview, and record, which are linked to other systems in Gather Deal and commission distributed.
FR24	The system shall allow administrators to view the overview and statistics of the organization's resellers and sales.

4.4 Non-Functional Requirements

Table 4-6: Non-Functional Requirements

ID	Non-Functional Requirement	Category of NFR
NFR01	The system should be available every single second during working hours.	Availability
NFR02	The system's storage solution must have a minimum storage capacity of 20 GiB.	Reliability
NFR03	The system shall be easy to be used by users without training and understanding of designing.	Usability
NFR04	The system must be able to handle a minimum of 1 active instance and scale up to a maximum of 4 instances to meet performance demands	Performance
NFR05	The system must update the new data input by the user in real-time and must be reflected to the user within one second.	Performance
NFR06	The system must be able to encrypt the user's data using SSL/TLS certificates.	Security
NFR07	The website should be able to defend against distributed denial-of-service attacks.	Security
NFR08	The website should be able to defend against SQL injection.	Security
NFR09	The website should be able to defend against Cross-site scripting (XSS).	Security

Table 4-7 provides a mapping between the functional requirements for the system and their respective use case IDs.

Table 4-7: Mapping between Functional Requirement ID and Use Case ID

Functional Requirement ID	Use Case ID
FR01	UC01
FR02, FR03	UC02
FR04	UC03
FR05, FR06, FR07	UC04
FR08	UC05
FR11	UC06
FR12	UC07
FR17, FR18, FR19	UC08
FR20	UC09
FR21, FR22	UC10
FR23, FR24	UC11

4.5 Use Case Modelling

4.5.1 Use Case Diagrams

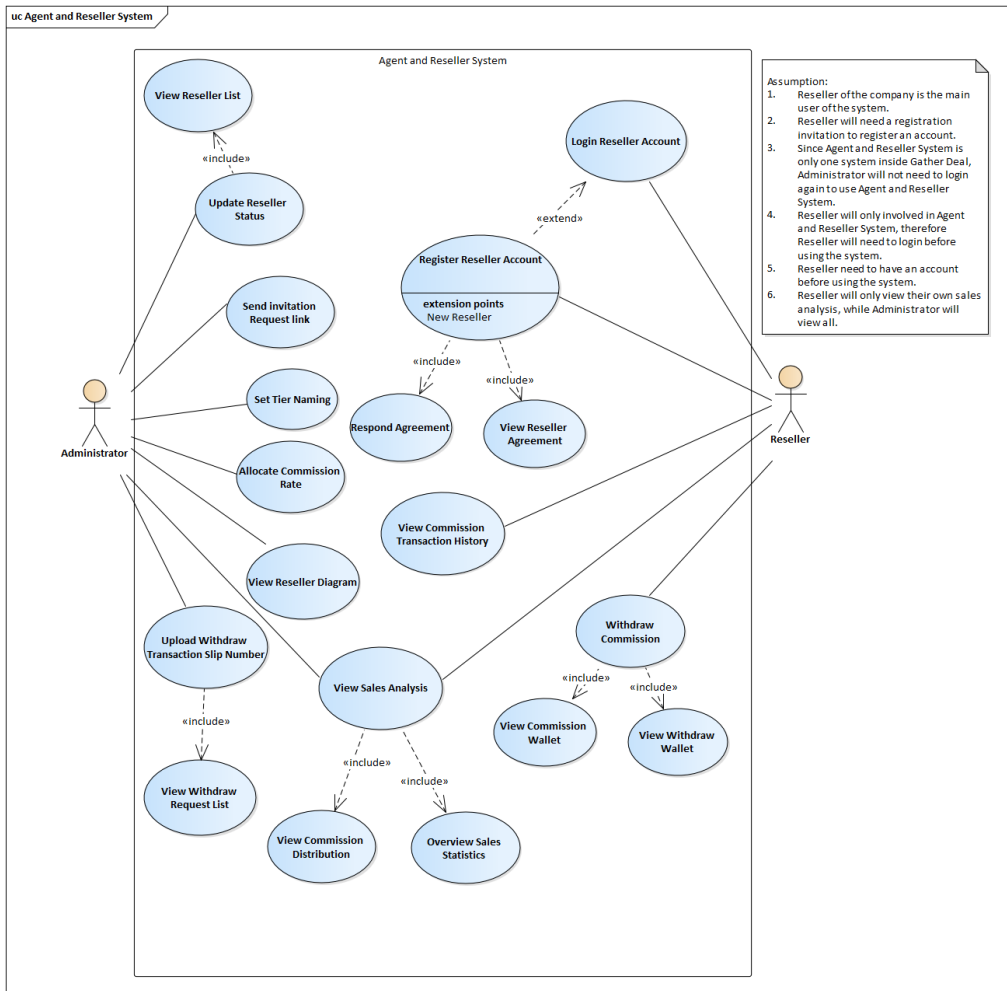


Figure 4-1: Use Case Diagram

4.5.2 Use Case Description

Table 4-8: Use Case Description of Login Reseller Account

Use Case Name: Login Reseller Account	ID: UC01	Importance Level: High
Primary Actor: Reseller	Use Case Type: Detail, Real	
Stakeholders and Interests: The Reseller -Wants to gain access to the Agent and Reseller System.		
Brief Description: This use case describes how the Reseller can input their Reseller email address and password to gain access to the application.		
Trigger: Reseller enters Reseller email address and password.		
Relationships: Association : Reseller Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The system displays a login page for the Reseller to enter a phone number and password. 2. The Reseller is able to log in with an existing account only. 3. The reseller inputs the Reseller's phone number and password into the system. 4. The system searches the database for the corresponding entry. If the username or password is invalid, Af-1 Invalid Entry is executed. 5. The system provides access to the following Reseller to the application. 		
Sub-flows: N/A		
Alternate/Exceptional Flows: Af-1: Invalid Registration Entry <ol style="list-style-type: none"> 1. The system states that the Reseller's phone number or password is invalid. 2. Return to normal flow 3. 		

Table 4-9: Use Case Description of Edit Reseller List

Use Case Name: Update Reseller Status	ID: UC02	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to edit the reseller status.		
Brief Description: This use case describes how the Administrator updates the reseller status in the system.		
Trigger: The administrator wants to edit the reseller that is no longer working with the organization.		
Relationships: Association : Administrator Include : View Reseller List Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays the reseller list and shows the related information such as the reseller's phone number, name, and tier. 2. The administrator clicks on the reseller to edit it. 3. The system will display a new Reseller List.		
Sub-flows: N/A		
Alternate/Exceptional Flows: N/A		

Table 4-10: Use Case Description of Send Invitation Request

Use Case Name: Send Invitation Request Link	ID: UC03	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to new reseller to join the system.		
Brief Description: This use case describes how the Administrator invites new resellers to the system.		
Trigger: The administrator wants to invite the new reseller who just joined the organization to use the system.		
Relationships: Association : Administrator Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The administrator inputs the phone number, referrer's phone number, and tier of the new reseller. 2. The system displays a confirmation message. a. If the administrator clicks "invite" i. The Sf-1: confirm invitation with "invite" is performed. b. If the administrator clicks "cancel" i. The Sf-2: confirm invitation with "cancel" is performed. 3. The system will remain on the invitation page.		
Sub-flows: Sf-1: confirm invitation with "invite" is performed. 1. The system sends an SMS including an invitation link to the new reseller. Sf-2: confirm invitation with "cancel" is performed 1. The system will cancel all the actions done and return to the invitation page.		
Alternate/Exceptional Flows: N/A		

Table 4-11: Use Case Description of Register Reseller Account

Use Case Name: Register Reseller Account	ID: UC04	Importance Level: High
Primary Actor: Reseller	Use Case Type: Detail, Real	
Stakeholders and Interests: The Reseller -Wants to register an account after receiving the invitation link.		
Brief Description: This use case describes how the Reseller registers a new account after receiving an invitation link to the system.		
Trigger: The reseller wants to create a reseller account to use the system.		
Relationships: Association : Reseller Include : View Agreement, Respond Agreement Extend : Login Reseller Account Generalization : N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The reseller clicks on the invitation link received through SMS. 2. The system requests the reseller input the phone number. If the reseller inputs a phone number other than the invited number, the system performs sub-flow 1.1. 3. The system requests the reseller input the password of the reseller. If the reseller inputs the password length is less than 8, the system performs sub-flow 2.1. 4. The system will display a Reseller Agreement, the reseller is required to respond to it. If the reseller accepts the agreement, the system performs sub-flow 3.1. If the reseller rejects the agreement, the system performs sub-flow 4.1. 		
Sub-flows: <ol style="list-style-type: none"> 1.1 Display error of the name <ol style="list-style-type: none"> 1.1.1) The system displays the error messages. 1.1.2) The system requests the user re-enter a valid phone number. 2.1 Display error of password <ol style="list-style-type: none"> 2.1.1) The system displays the error messages. 2.1.2) The system requests the user re-enter a valid password. 3.1 Direct login to the system. 4.1 Return to normal flow 3 		
Alternate/Exceptional Flows: N/A		

Table 4-12: Use Case Description of Set Tier Naming

Use Case Name: Set Tier Naming	ID: UC05	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to set new naming for each reseller tier level.		
Brief Description: This use case describes how the Administrator sets new names for each reseller tier level.		
Trigger: The Administrator wants to set new naming for each reseller tier level.		
Relationships: Association : Administrator Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The system shows the default naming for each tier level. 2. The system requests the Administrator input the naming. 3. The Administrator clicks on the “save” button. 4. The system saves and displays the latest tier-level naming. 		
Sub-flows: N/A		
Alternate/Exceptional Flows: N/A		

Table 4-13: Use Case Description of View Reseller Network Diagram

Use Case Name: View Reseller Diagram	ID: UC06	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to view the reseller diagram.		
Brief Description: This use case describes how the Administrator views the reseller diagram from the system.		
Trigger: The administrator wants to view the reseller diagram		
Relationships: Association : Administrator Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays all organization resellers in a diagram.		
Sub-flows: N/A		
Alternate/Exceptional Flows: N/A		

Table 4-14: Use Case Description of Allocate Commission Rate

Use Case Name: Allocate Commission Rate	ID: UC07	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to set or change the commission rate for each tier of reseller.		
Brief Description: This use case describes how the Administrator sets or changes the commission rate for different tiers of resellers in the system.		
Trigger: The administrator wants to set or change the rate of commission for each reseller tier in the system.		
Relationships: Association : Administrator Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: <ol style="list-style-type: none"> 1. The system requests the Administrator the commission rate of the Direct Commission, Indirect Commission, Leadership Commission, and Pool Commission for each tier. If the Administrator inputs a rate other than a number, the system performs sub-flow 1.1. 2. The Administrator clicks on the “save” button. 3. The system saves and displays the latest commission rate. 		
Sub-flows: 1.1 Display error of commission rate <ol style="list-style-type: none"> 1.1.1) The system displays the error messages. 1.1.2) The system requests the user re-enter a valid rate. 		
Alternate/Exceptional Flows: N/A		

Table 4-15: Use Case Description of Withdraw Commission

Use Case Name: Withdraw Commission	ID: UC08	Importance Level: High
Primary Actor: Reseller	Use Case Type: Detail, Real	
Stakeholders and Interests: The Reseller -Wants to withdraw the commission from the system.		
Brief Description: This use case describes how the Reseller withdraws commission from the system		
Trigger: The Reseller wants to withdraw commission from the system.		
Relationships: Association : Reseller Include : View Commission Wallet, View Withdraw Wallet Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays the total amount of Commission Wallet and Withdraw Wallet. 2. The system will require the reseller to input the number of withdrawers. If the reseller inputs an amount other than the number or larger than the total commission the reseller has, the system performs sub-flow 1.1. 3. The system will add the amount to Withdraw Wallet.		
Sub-flows: 1.1 Display error of withdraw amount 1.1.1) The system displays the error messages. 1.1.2) The system requests the user re-enter a valid amount.		
Alternate/Exceptional Flows: N/A		

Table 4-16: Use Case Description of View Commission Transaction History

Use Case Name: View Commission Transaction History	ID: UC09	Importance Level: High
Primary Actor: Reseller	Use Case Type: Detail, Real	
Stakeholders and Interests: The Reseller -Wants to view the commission transaction from the system.		
Brief Description: This use case describes how the Reseller view commission transaction from the system		
Trigger: The Reseller wants to view the commission transaction history from the system.		
Relationships: Association : Reseller Include : N/A Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays the date, task, and amount of each transaction.		
Sub-flows: N/A		
Alternate/Exceptional Flows: N/A		

Table 4-17: Use Case Description of Upload Withdraw Transaction Slip Number

Use Case Name: Upload Withdraw Transaction Slip Number	ID: UC10	Importance Level: High
Primary Actor: Administrator	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to upload the transaction slip number that is paid for reseller withdrawal to the system.		
Brief Description: This use case describes how the Administrator uploads transaction slip number to the system		
Trigger: The Administrator wants to upload the withdrawal transaction number to the system.		
Relationships: Association : Reseller Include : View Withdraw List Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays the Withdraw List 2. The system will require the reseller to upload the transaction slip number.		
Sub-flows: N/A		
Alternate/Exceptional Flows: N/A		

Table 4-18: Use Case Description of View Sales Analysis

Use Case Name: View Sales Analysis	ID: UC11	Importance Level: High
Primary Actor: Administrator, Reseller	Use Case Type: Detail, Real	
Stakeholders and Interests: The Administrator -Wants to view the organization's sales analysis from the system. The Reseller -Wants to view its own sales analysis from the system.		
Brief Description: This use case describes how the Administrator and Reseller view the sales analyses from the system		
Trigger: The administrator wants to analyse the organization's sales from the system.		
Relationships: Association : Administrator Include : Overview Sales Statistics, View Commission Distribution Extend : N/A Generalization : N/A		
Normal Flow of Events: 1. The system displays the Overview Sales Statistics If is Administrator performs sub-flow 1.1 If is Reseller performs sub-flow 1.2		
Sub-flows: 1.1 Display the total amount of sales and the total amount of commission distributed. 1.2 Display the reseller's own total amount of sales, and commission record.		
Alternate/Exceptional Flows: N/A		

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

In this chapter, an overview of the project's system architectural design will be covered first. Follow by the database design and system flow design for this project are discussed after the system architecture. The database design will discuss the table and data storing design of the system, the system flow design will discuss the user-system interaction of the system.

5.2 System Architectural Design

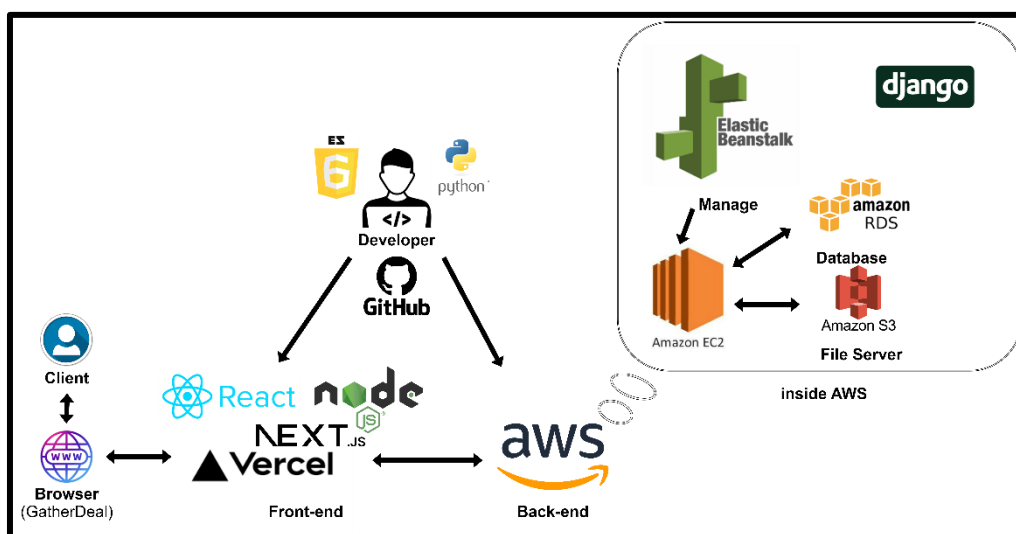


Figure 5-1: Overview of System Architecture Design

This project architecture design mainly has three components, developer, frontend, and backend. In the frontend layer, it has React, Node.js, Next.js, and Vercel play the most important role, it is responsible for handling the interface of the system.

First, Next.js, by Vercel, serves as the foundation for server-rendered applications. This is because of its ability to render content server-side, split code automatically, and optimise routing, all of which facilitate the construction of complex online systems. JavaScript code is executed on the server side using the Node.js runtime environment. User interfaces in the system are built using

React, a JavaScript framework. This is because React offers reusable and modular user interface elements.

In addition, four major Amazon cloud services are used in this project's backend: Elastic Compute Cloud (EC2), Elastic Beanstalk, RDS (Relational Database Service), and S3 (Simple Storage Service). This project's web application is hosted on AWS EC2 instances, which users may access in order to view the web application. Elastic Beanstalk is used for cloud health checks, monitoring, and scalability. Relational databases are set up, maintained, and expanded in the cloud via RDS. Large volumes of data may be safely, permanently, and highly scalable stored and accessed by users using S3. On the other hand, the system was constructed on the backend using the Django framework. With the capability of providing a structured and effective approach to web development.

Lastly, this project used Github as the platform for hosting and managing the system code. It enables developers to manage project repositories and, track changes in code.

5.3 Database Design

5.3.1 Entity Relationship Diagram

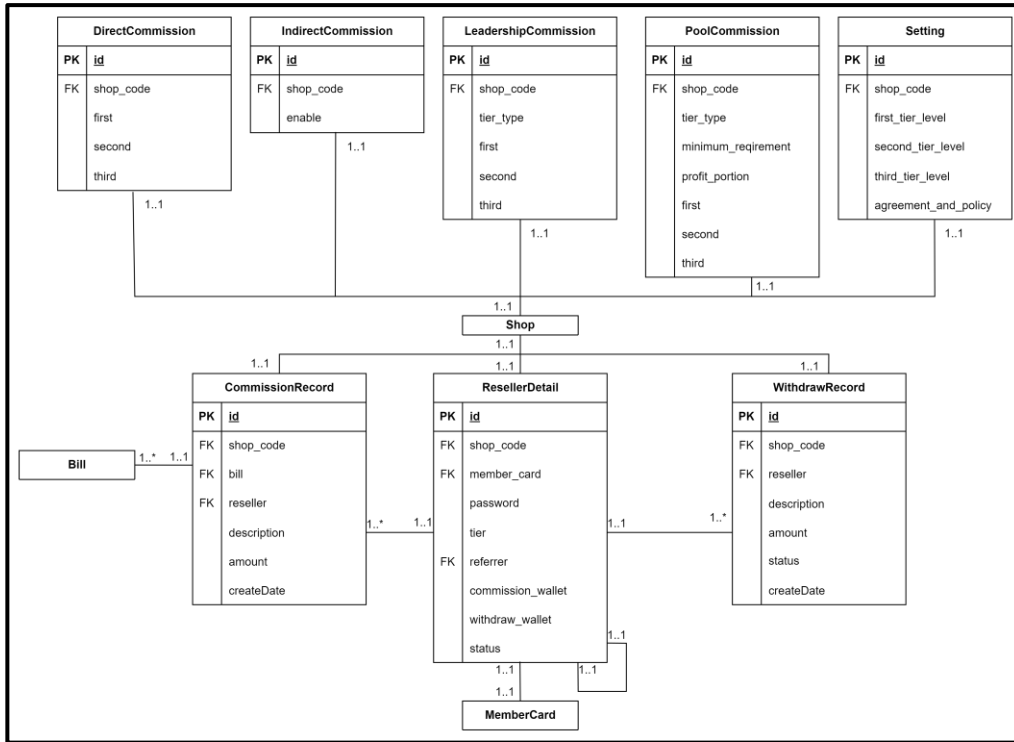


Figure 5-2: Entity Relationship Diagram of System

This project created eight new tables that will interact with each other and three existing tables of the Gather Deal system. The newly created tables are ResellerDetail, CommissionRecord, WithdrawRecord, DirectCommission, IndirectCommission, Leadership Commission, PoolCommission, and Setting table. The existing tables are the Shop, Bill, and MemberCard tables.

5.3.2 Data Dictionary

Table 5-1: Data Dictionary (ResellerDetail)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of ResellerDetail	int	PK	-
member_card	Unique Identification of MemberCard	int	FK	MemberCard
shop_code	Unique Identification of Shop	char	FK	Shop
password	Password of reseller	char	-	-
tier	tier of reseller	char	-	-
referrer	Unique Identification of ResellerDetail for referrer	int	FK	MemberCard
commission_wallet	Commission wallet of reseller	decimal	-	-
withdraw_wallet	Withdraw the wallet of the reseller	decimal	-	-
status	Status of reseller	boolean	-	-

Table 5-2: Data Dictionary (Setting)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of Setting	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
first_tier_level	First-tier level naming of reseller	char	-	-
second_tier_level	Second-tier level naming of reseller	char	-	-
third_tier_level	Third-tier level naming of reseller	char	-	-
agreement_and_policy	Agreement and policy of reseller	char	-	-

Table 5-3: Data Dictionary (DirectCommission)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of DirectCommission	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
first	First-tier level direct commission rate	decimal	-	-
second	Second-tier level direct commission rate	decimal	-	-
third	Third-tier level direct commission rate	decimal	-	-

Table 5-4: Data Dictionary (IndirectCommission)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of IndirectCommission	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
enable	Status of Indirect Commission	decimal	-	-

Table 5-5: Data Dictionary (LeadershipCommission)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of LeadershipCommission	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
tier_type	Minimum tier type requirement for Leadership Commission	char	-	-
first	First-level Leadership Commission rate	decimal	-	-

second	Second-level Leadership Commission rate	decimal	-	-
third	Third-level Leadership Commission rate	decimal	-	-

Table 5-6: Data Dictionary (PoolCommission)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of PoolCommission	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
tier_type	Minimum tier type requirement for Pool Commission	char	-	-
minimum_requirement	Minimum amount of sales for Pool Commission	decimal	-	-
profit_portion	Profit portion of total sales for Pool Commission	decimal	-	-
first	First-level Leadership Commission rate	decimal	-	-
second	Second-level Leadership Commission rate	decimal	-	-
third	Third-level Leadership Commission rate	decimal	-	-

Table 5-7: Data Dictionary (CommissionRecord)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of CommissionRecord	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop

bill	Unique Identification of Bill	int	FK	Bill
reseller	Unique Identification of ResellerDetail	int	FK	Reseller
description	Description of commission record	char	-	-
amount	Amount of commission record	decimal	-	-
createDate	Date and time of commission record	DateTime	-	-

Table 5-8: Data Dictionary (WithdrawRecord)

Field Name	Description	Type	Key	FK Referenced table
id	Unique Identification of WithdrawRecord	int	PK	-
shop_code	Unique Identification of Shop	char	FK	Shop
reseller	Unique Identification of ResellerDetail	int	FK	Reseller
description	Description of withdrawal record	char	-	-
amount	Amount of withdrawal record	decimal	-	-
status	Status of withdrawal record	boolean	-	-
createDate	Date and time of withdrawal record	DateTime	-	-

5.4 System Flow Design

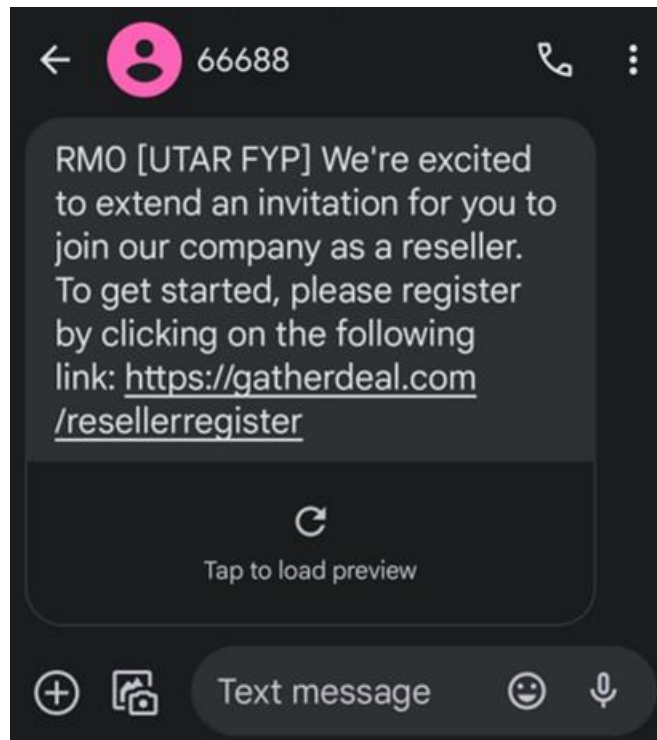
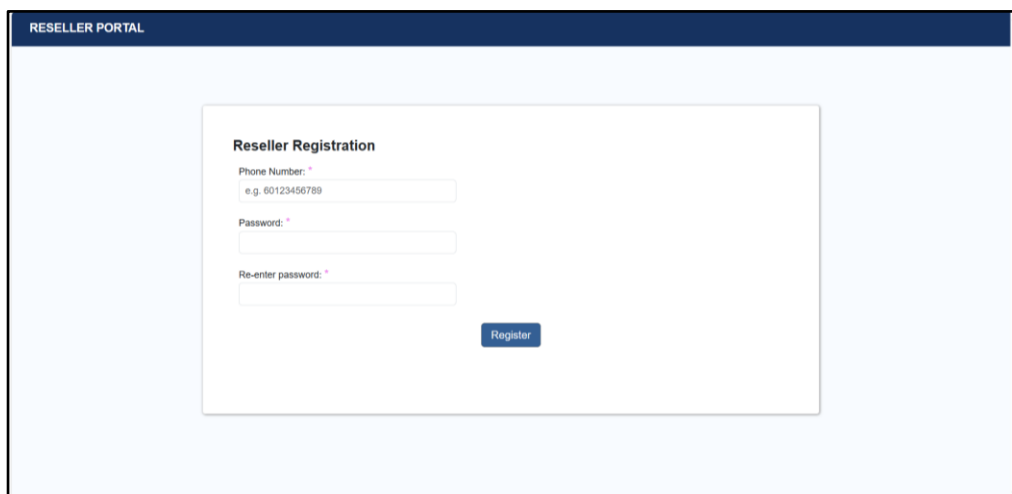


Figure 5-3: Reseller side, Invitation Message

After a new reseller clicks on the received invitation link in Figure 5-3 from the organization to join as the organization reseller, it will directly be brought to this interface in Figure 5-4. The new reseller will be required to fill in the phone number and password to register an account.



RESELLER PORTAL

Reseller Registration

Phone Number: *
e.g. 60123456789

Password: *

Re-enter password: *

Register

Figure 5-4: Reseller side, Register Account Interface

After the reseller fills all the input required, the system will bring the reseller to the interface in Figure 5-5. The reseller will be able to view the organization's reseller agreement, and the reseller will be able to accept or deny the agreement. If the reseller accepts the agreement the system will bring the reseller to the interface in Figure 5-7, else the reseller will return to the interface in Figure 5-4.

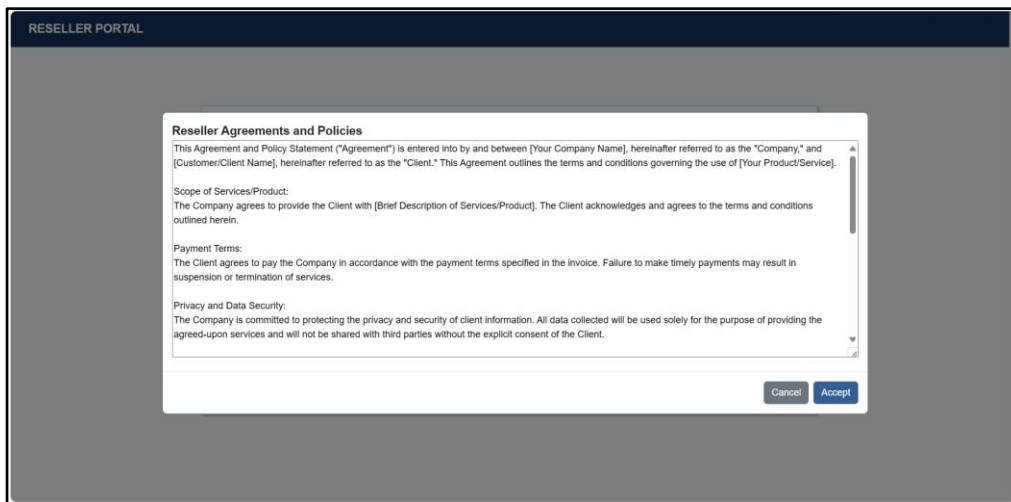


Figure 5-5: Reseller side, Agreement Interface

If the reseller already contains an account in the system, the reseller will be required to log in before using the system by filling in the phone number and password in the interface Figure 5-6.

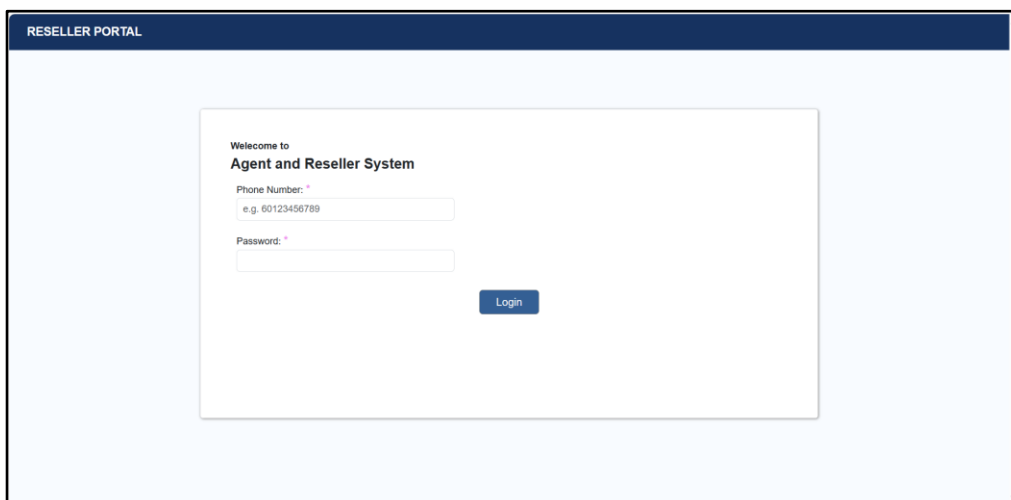


Figure 5-6: Reseller side, Login Interface

Figure 5-7 shows the user interface that will be displayed upon the reseller's successful login. The central area of Figure 5-7 features a comprehensive depiction of the commission wallet, positioned at the top, alongside the withdraw wallet. Below it, is the commission transaction history. Adjacent to this insightful overview lies a designated space on the right-hand side. Here, the reseller is empowered to initiate withdrawals by inputting the desired withdrawal amount.

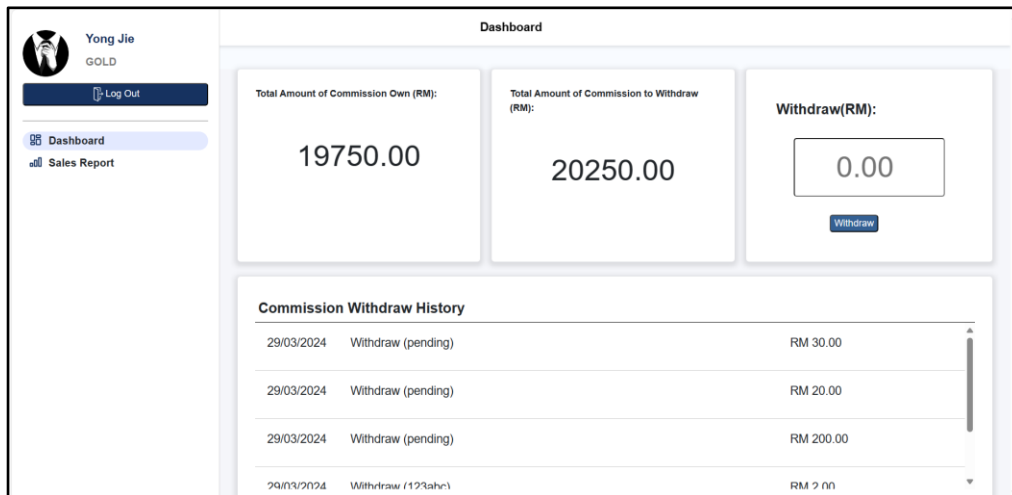


Figure 5-7: Reseller side, Commission Withdraw Interface

Figure 5-8 shows the user interface that will be displayed upon the reseller's selecting the “Sales Report” in the left section of Figure 5-8. The central top area of Figure 5-8 features a comprehensive depiction of the sales and commission chart. Below it, is the commission distribution record of the reseller.

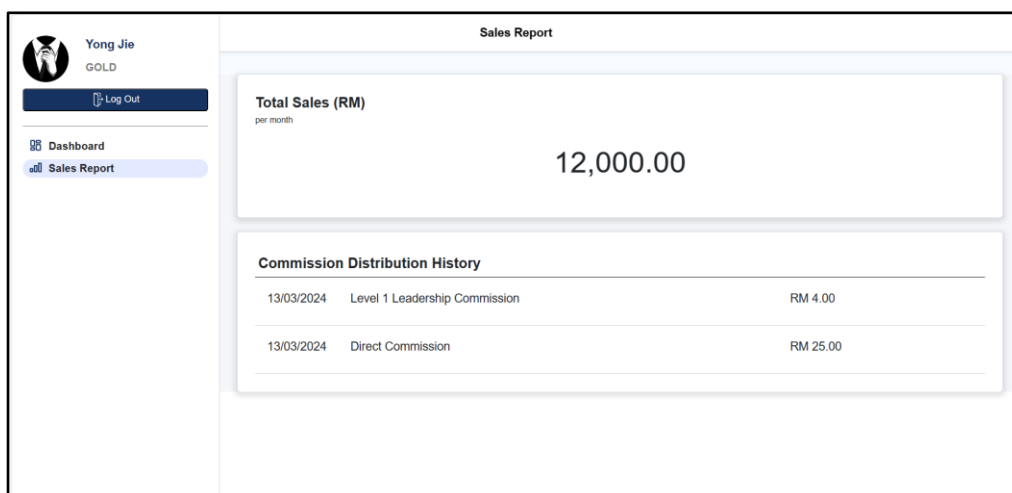


Figure 5-8: Reseller side, Sales Report Interface

In Figure 5-9, this is the interface of the administrator. The central area of Figure 5-9 features a comprehensive depiction of the overview of the organization which includes the total commission distribution, total number of resellers, total sales amount, and a general commission distribution for the current month.

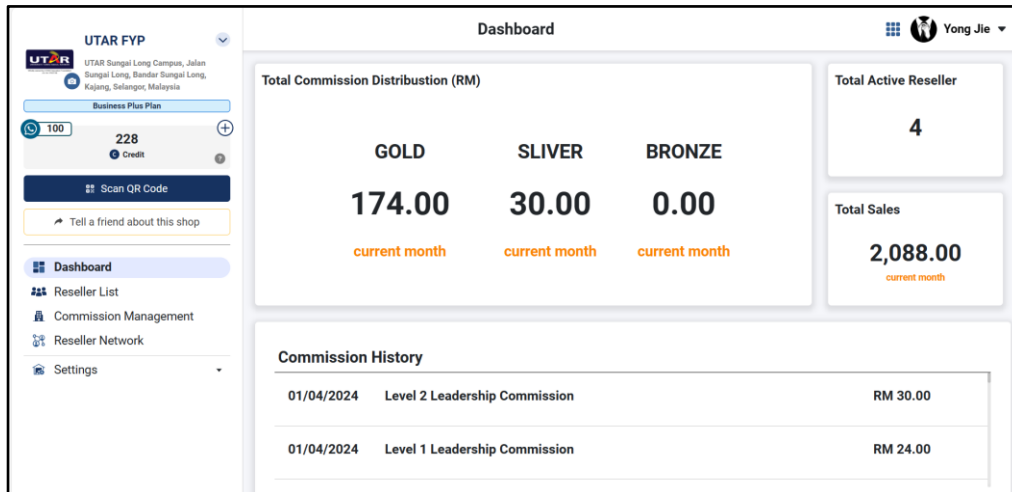


Figure 5-9: Administrator side, Analyse Sales Interface

In Figure 5-10, this is the interface of the administrator after clicking on the dark blue button “Invite Reseller” on the left of Figure 5-10. It requires the administrator to input the phone number, tier, and referrer of the reseller in order to invite a new reseller.

Fields are optional, but must filled at least one.

Phone* Tier Type*

Referrer Phone

Figure 5-10: Administrator side, Send Invitation Request Interface

In Figure 5-11, this is the interface of the administrator after clicking on the “Reseller List” on the left of Figure 5-9. It allows the administrator to view and edit the reseller that is in the system.

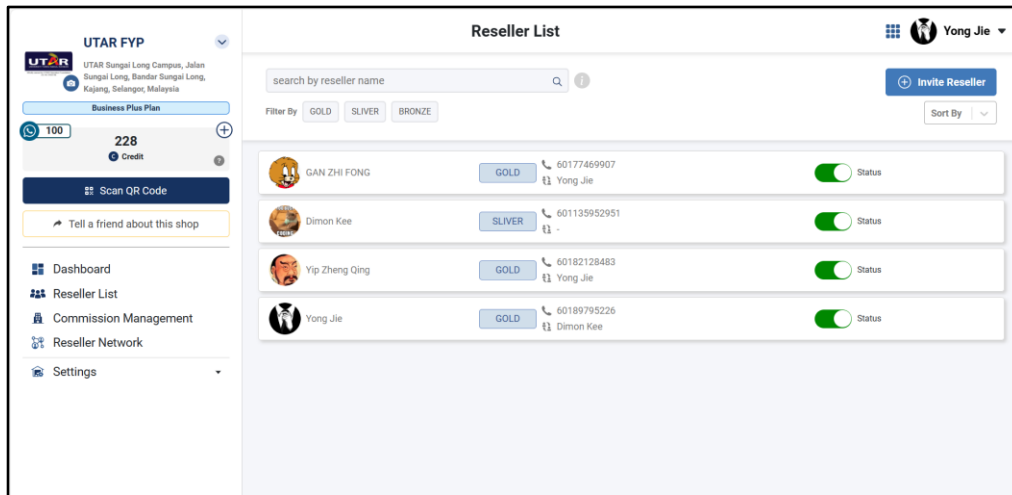


Figure 5-11: Administrator side, List of Reseller Interface

In Figure 5-12, this is the interface of the administrator after clicking on the “Commission Management” on the left of Figure 5-9. It allows the administrator to distribute the pool commission by clicking the “Distribute Pool Commission” button. Furthermore, it allows viewing commission distribution, viewing the withdrawer request list, and uploading the transaction number to the system in Figure 5-13.

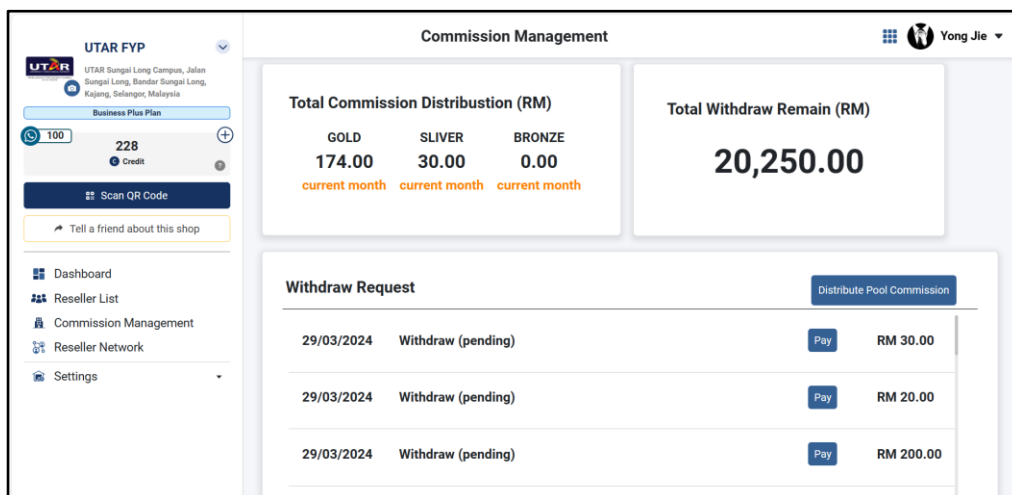


Figure 5-12: Administrator side, Commission Distribution Interface

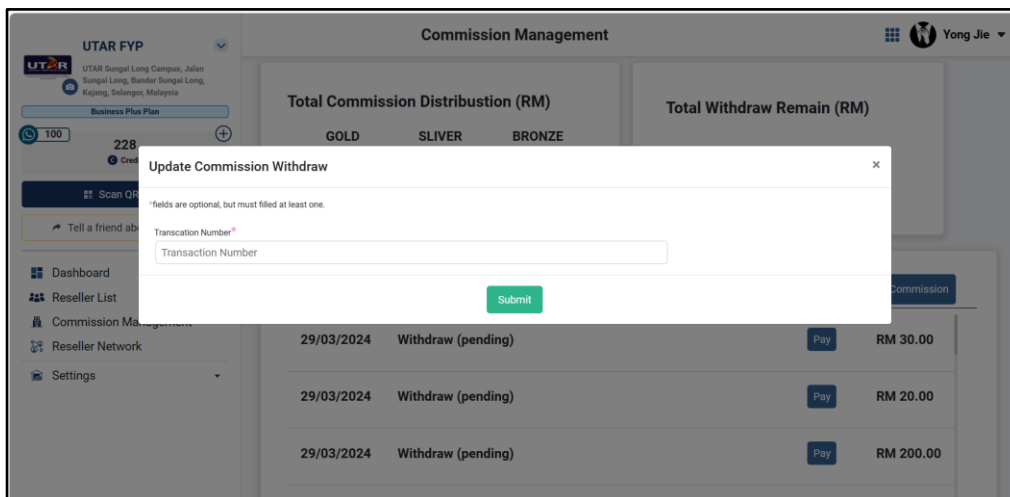


Figure 5-13: Administrator side, Update Commission Withdraw Interface

In Figure 5-14, this is the interface of the administrator after clicking on the “Reseller Network” on the left of Figure 5-9. It allows the administrator to view all the organization’s resellers in a network diagram form in the system.

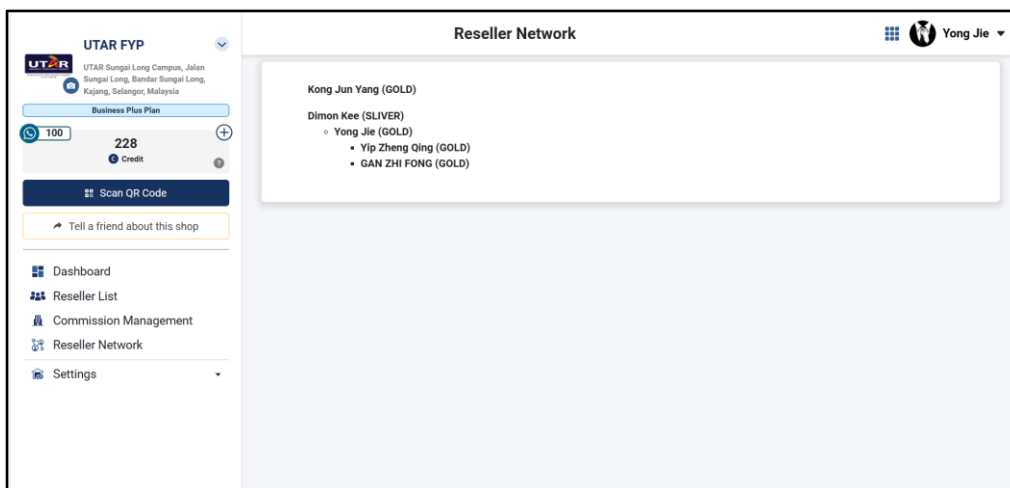


Figure 5-14: Administrator side, Network Diagram of Reseller Interface

In Figure 5-15, this is the interface of the administrator after clicking on the “Tier Settings” on the left of Figure 5-9. It requires the administrator to input the tier naming and reseller agreement in order to update the tier naming and the reseller agreement.

The screenshot displays the 'Tier Settings' interface for an administrator. On the left is a sidebar with the UTAR FYP logo and navigation menu items: Dashboard, Reseller List, Commission Management, Reseller Network, and Settings. The main content area is titled 'Tier Settings' and includes a user profile for 'Yong Jie'. The 'Tier Level Naming' section has three input fields: 'First Level' (GOLD), 'Second Level' (SLIVER), and 'Third Level' (BRONZE). Below this is the 'Reseller Agreement & Policy' section, which contains a text area with a placeholder agreement statement and a 'Save' button.

Figure 5-15: Administrator side, Tier Settings Interface

In Figure 5-16, this is the interface of the administrator after clicking on the “Commission Settings” on the left of Figure 5-9. It requires the administrator to input the percentage of each commission to reset all the commission percentages.

The screenshot displays the 'Commission Settings' interface for an administrator. On the left is a sidebar with the UTAR FYP logo and navigation menu items: Dashboard, Reseller List, Commission Management, Reseller Network, and Settings. The main content area is titled 'Commission Settings' and includes a user profile for 'Yong Jie'. The 'Direct Commission' section has three input fields for GOLD (25.00), SLIVER (18.00), and BRONZE (12.00). The 'Indirect Commission' section has a checkbox. The 'Leadership Commission' section has a dropdown for 'Tier Type' (SLIVER) and three input fields for 'First Level' (4.00), 'Second Level' (5.00), and 'Third Level' (6.00). The 'Pool Commission' section has three input fields for 'First Level' (50.00), 'Second Level' (30.00), and 'Third Level' (20.00), and a dropdown for 'Tier Type' (GOLD). Below this are input fields for 'Profit Portion' (5.00) and 'Minimum Requirement (RM)' (111.00). A 'Save' button is located at the bottom.

Figure 5-16: Administrator side, Commission Settings Interface

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter discusses the implementation details of the Agent and Reseller system in Gather Deal. Web application implementation, Application Programming Interfaces (APIs), and Database Setup and Management are all included. This chapter also includes a list of the API functions that the project implements. Overall, each page and feature of the programs will be shown in detail, along with a section code explanation of each function.

6.2 Database Setup and Management

In the implementation phase of the Agent and Reseller System within the Gather Deal environment, database setup and management are critical components to ensure the smooth functioning of the system. The process begins by cloning the Gather Deal repository to the local environment, granting access to the existing database schema and structure in Figure 6-1.

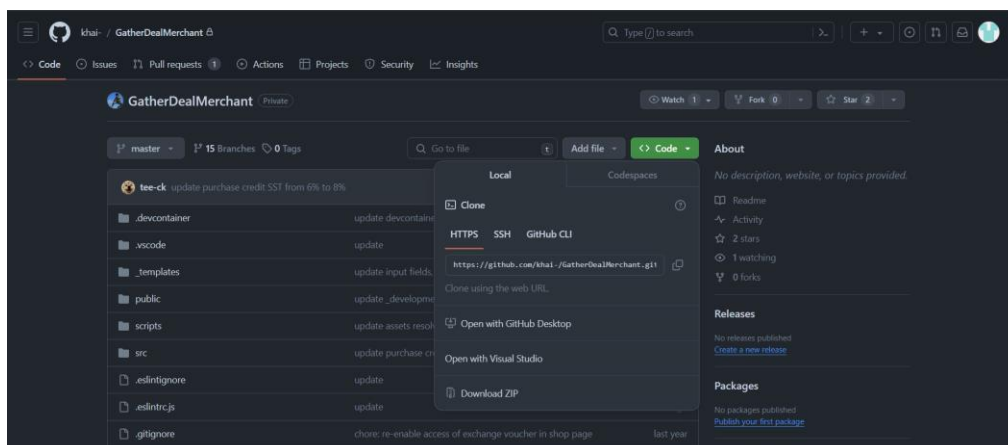


Figure 6-1: Gather Deal Repository

Subsequently, a dedicated branch is created for the modifications and developments specific to the Agent and Reseller System, allowing for independent management of changes. Reviewing the existing database schema of Gather Deal is essential to understanding its tables, relationships, and constraints in Figure 6-2. This understanding guides the identification of areas

requiring modifications or additions to accommodate the new functionalities of the Agent and Reseller System.

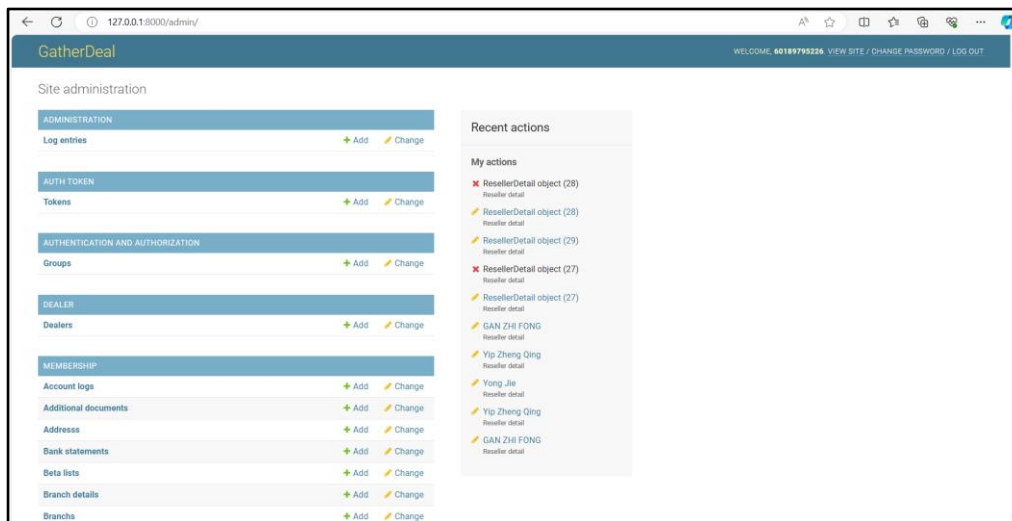


Figure 6-2: Gather Deal Existing Database Schema

Once identified, modifications to the database schema are made, which may include creating new tables, altering existing ones, defining relationships, and adding constraints to ensure data integrity in Figure 6-3.

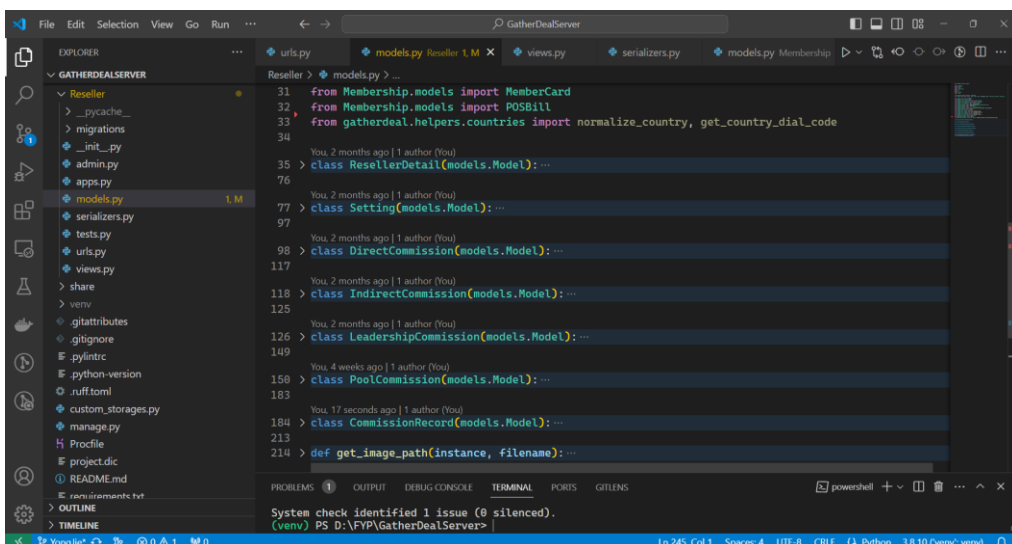


Figure 6-3: Modifications of Database Schema

Collaboration with team members and effective utilization of version control systems like Git ensure coordinated database setup and management

tasks, maintaining code integrity throughout the development process by using pull requests in Figure 6-4.

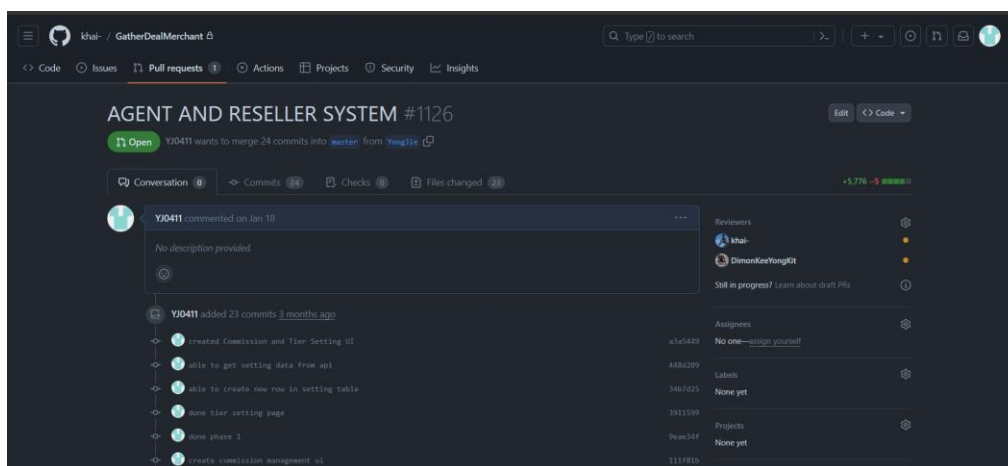


Figure 6-4: Version Control System

Through these steps, the database for the Agent and Reseller System within the Gather Deal framework is set up and managed effectively, enabling seamless integration and reliable data management capabilities.

6.3 Application Programming Interfaces (APIs)

In the implementation phase of the Agent and Reseller System within the Gather Deal framework, the development of Application Programming Interfaces (APIs) plays a pivotal role in facilitating seamless communication and interaction between different components of the system. APIs serve as the bridge between the frontend user interface and the backend database, enabling data retrieval, manipulation, and transmission.

The development process begins by identifying the specific functionalities and operations that need to be exposed through APIs to support the desired features of the Agent and Reseller System. This includes defining endpoints for accessing reseller information, commission calculations, sales data, and other relevant operations. Each API endpoint is designed to accept input parameters, perform the necessary operations, and return the appropriate response in a structured format, using JSON.

The implementation of APIs involves writing code to handle incoming requests, validate input data, execute the corresponding business logic, and generate the response. This code is typically developed using server-side programming languages using Python.

Table 6-1: List of Function Endpoints

Function	Description
getSetting	Retrieve setting detail
createSetting	Create setting detail
updateSetting	Update setting detail
getDirectCommission	Retrieve direct commission detail
createDirectCommission	Create direct commission detail
updateDirectCommission	Update direct commission detail
getIndirectCommission	Retrieve indirect commission detail
createIndirectCommission	Create indirect commission detail
updateIndirectCommission	Update indirect commission detail
getLeadershipCommission	Retrieve leadership commission detail
createLeadershipCommission	Create leadership commission detail
updateLeadershipCommission	Update leadership commission detail
getPoolCommission	Retrieve pool commission detail
createPoolCommission	Create pool commission detail
updatePoolCommission	Update pool commission detail
getResellerDetail	Retrieve reseller detail
updateResellerStatus	Update reseller detail
createResellerDetail	Create reseller detail
getResellerCount	Retrieve the total amount of reseller
getResellerTree	Retrieve company reseller in a tree format
signUp	Sign up reseller to the system
logIn	Log in reseller to the system
getResellerProfile	Retrieve reseller profile
makePaymentCommission	Record commission distribution (except pool commission)
makePoolCommission	Record pool commission distribution
getTotalCommission	Retrieve the total amount of commission distributed based on tier
getCommissionList	Retrieve the commission distribution record of a reseller
getAllCommissionList	Retrieve all commission distribution record
makeWithdrawRequest	Create a withdrawal record
getWithdrawList	Retrieves withdraw the record of a reseller
getAllWithdrawList	Retrieve all withdrawal record
updateWithdrawStatus	Update the withdrawal record with a transaction number
getTotalWithdraw	Retrieve the total withdrawal amount of all resellers

Through the development of robust and well-documented APIs, the Agent and Reseller System within the Gather Deal framework achieves seamless integration, efficient data exchange, and enhanced functionality, ultimately delivering a superior user experience for resellers and administrators alike.

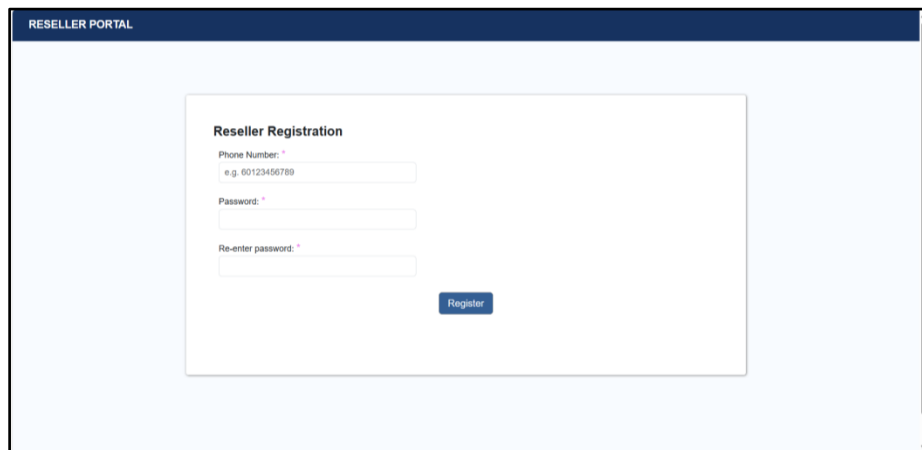
6.4 Web Application Implementation

This section will display each web application page and explain the system logic using section code.

6.4.1 Agent and Reseller Registration

The Agent and Reseller Registration module is divided into sign-up activity, login activity, and logout activity.

6.4.1.1 Sign Up Activity



The screenshot shows a web browser window titled "RESELLER PORTAL". Inside the browser, there is a registration form titled "Reseller Registration". The form contains three input fields: "Phone Number:" with a placeholder "e.g. 60123456789", "Password:", and "Re-enter password:". Below these fields is a blue "Register" button.

Figure 6-5: Reseller Register Page

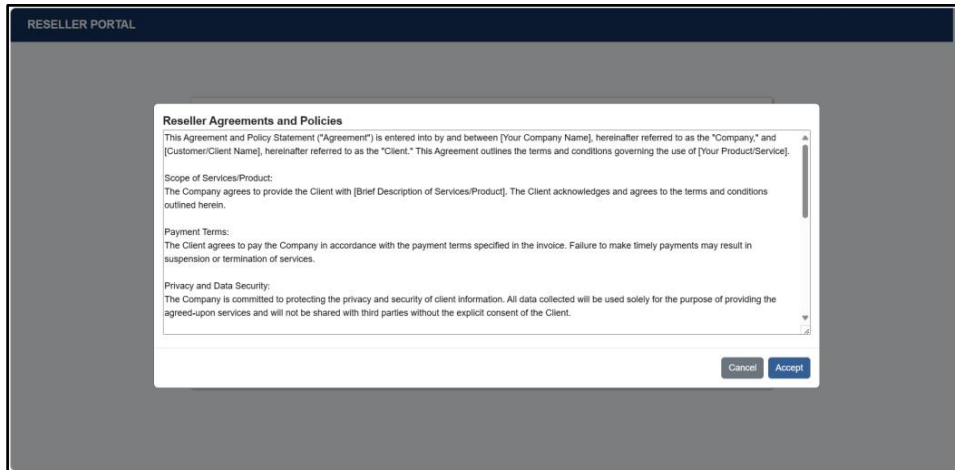


Figure 6-6: Reseller Agreement Page

When the reseller clicks on the register button in Figure 6-5, it will enter the `handleRegisterClick` function. In this function, it will check the password and the re-entered password. If both are the same, it will pass the phone number and password to the backend using the `signup` API in Figure 6-7. Then if the API returns a status of 200 it will display the agreement for the reseller to accept.

```

const handleRegisterClick = async () => {
  if (password === reenteredPassword) {
    if (!phoneNumber || !password || !reenteredPassword) {
      window.alert('Please fill in all required fields.');
```

```
      return;
    }

    try {
      const response = await axios.put('reseller/signup', {
        phone_number: phoneNumber,
        password: password,
      });

      if (response.status === 200) {
        setAgreement(response.data)
        setSuccessModal(true);
      }

    } catch (error) {
      window.alert(error.response.data.message);
    }
  } else {
    window.alert('Passwords do not match');
  }
};

```

Figure 6-7: handleRegisterClick function

In the backend, the signUp endpoint will validate the phone number that is passed in with the invited reseller list. If the phone number is valid, it will convert the password into a hash format and save it into the database in Figure 6-8.

```

# SignUp
@api_view(['PUT'])
@csrf_exempt
def signUp(request):
    if request.method == 'PUT':
        try:
            phone_number = request.data.get('phone_number')
            password = request.data.get('password')

            if not phone_number or not password:
                return JsonResponse(data={"message": "Phone number and password are required"}, status=400, safe=False)

            reseller_instance = ResellerDetail.objects.filter(member_card__user__globalUsername=phone_number).first()

            if not reseller_instance:
                return JsonResponse(data={"message": "Please contact company before proceeding with registration."}, status=404, safe=False)

            if reseller_instance.password is not None:
                return JsonResponse(data={"message": "Reseller account already registered."}, status=409, safe=False)

            hashed_password = make_password(password) # Hash the password securely
            reseller_instance.password = hashed_password
            reseller_instance.save()

            token, created = Token.objects.get_or_create(user=reseller_instance)

            setting_instance = Setting.objects.get(shop_code=reseller_instance.shop_code)

            return JsonResponse(data={"token": token.key, "agreement_and_policy": setting_instance.agreement_and_policy}, status=200, safe=False)

        except Exception as e:
            return JsonResponse(data={"error": str(e)}, status=500, safe=False)
    else:
        return JsonResponse(data={"message": "Invalid request method"}, status=405, safe=False)

```

Figure 6-8: signUp endpoint

6.4.1.2 Login Activity

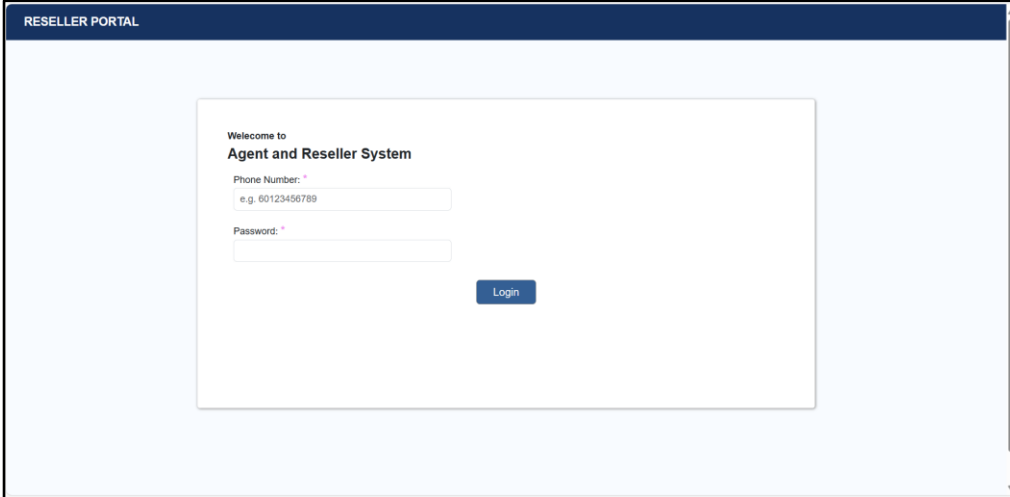
The image shows a screenshot of a web browser displaying the 'RESELLER PORTAL' login page. The page has a dark blue header with the text 'RESELLER PORTAL' in white. Below the header, there is a light blue background with a white rectangular login form centered on it. The form contains the following elements: a heading 'Welcome to Agent and Reseller System', a 'Phone Number:' label with a red asterisk, a text input field with the placeholder text 'e.g. 60123456789', a 'Password:' label with a red asterisk, another text input field, and a blue 'Login' button positioned below the password field. The browser's scrollbar is visible on the right side of the page.

Figure 6-9: Reseller LogIn Page

When the reseller clicks on the login button, it will enter the `handleLoginClick` function. In this function, it will check the presence of the phone number and the password. If both exist, it will pass the phone number and password to the backend using login API. Then if the API returns a true value it will save the phone number of the reseller to the session and redirect to the dashboard page in Figure 6-10.

```

const handleLoginClick = async () => {

  if (!phoneNumber || !password) {
    window.alert('Please fill in all required fields.');
```

```

    return;
  }

  try {
    const response = await axios.put('reseller/login', {
      phone_number: phoneNumber,
      password: password,
    });

    if (response.data.status === true) {
      const someData = { token: response.data.token };
      sessionStorage.setItem('myDataKey', JSON.stringify(someData));
      router.push('/resellerdashboard')
    }
  } catch (error) {
    window.alert(error.response.data.message);
  }
};

```

Figure 6-10: handleLoginClick function

In the backend, the logIn endpoint will validate the phone number that is passed in with the reseller list. If the phone number is valid, it will check the password that passed in with the passed store in the database in Figure 6-11.

```

# LogIn
def logIn(request):
    if request.method == 'PUT':
        try:
            phone_number = request.data.get('phone_number')
            password = request.data.get('password')
            if not phone_number:
                return JsonResponse(data={"message": "Phone number is required"}, status=400, safe=False)
            reseller_instance = ResellerDetail.objects.filter(member_card__user__globalUsername=phone_number).first()
            if not reseller_instance:
                return JsonResponse(data={"message": "Please Register before login."}, status=404, safe=False)
            if reseller_instance.password is None:
                return JsonResponse(data={"message": "Please Register before login."}, status=404, safe=False)
            if check_password(password, reseller_instance.password):
                token, created = Token.objects.get_or_create(user=reseller_instance)
                return JsonResponse(data={"token": token.key, "status":True}, status=200, safe=False)
            else:
                return JsonResponse(data={"message": "Incorrect username or login"}, status=401, safe=False)
        except Exception as e:
            return JsonResponse(data={"error": str(e)}, status=500, safe=False)
    else:
        return JsonResponse(data={"message": "Invalid request method"}, status=405, safe=False)

```

Figure 6-11: logIn endpoint

6.4.1.3 Logout Activity

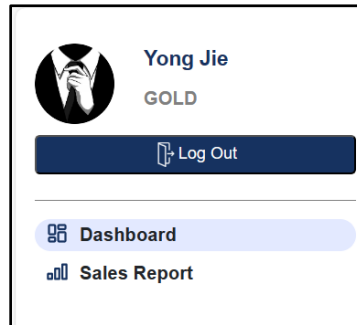


Figure 6-12: Reseller LogOut Button

When the reseller clicks on the logout button, it will remove the session that is saved during login or sign-up and redirect the reseller back to the login page in Figure 6-13.

```
<button style={{ backgroundColor: 'rgb(22, 50, 96)', width: '100%', textAlign: 'center', color: 'white',
onClick={() => { router.push('/resellerlogin'); sessionStorage.removeItem('myDataKey'); }}>
  <svg fill="#ffffff" height="24px" width="24px" version="1.1" id="Layer_1" xmlns="http://www.w3.org/20
    <g id="SVGRepo_bgCarrier" strokeWidth="0"></g>
    <g id="SVGRepo_tracerCarrier" strokeLinecap="round" strokeLinejoin="round"></g>
    <g id="SVGRepo_iconCarrier">
      <g id="Exit_1">
        <path d="M52.4501991,28.76785091-5-4.9990005c-0.3768997-0.3770008-0.9902-0.3770008-1.3671
        <path d="M40.2666016,39.4524498c-0.5527,0-1,0.4473-1,1v10.7900009c0,1.0429993-0.8310013,2
      </g>
    </g>
  </svg>
  Log Out
</button>
```

Figure 6-13: LogOut function

6.4.2 User Management

The User Management module is divided into viewing reseller list activity, updating reseller status activity, and sending invitation request link activity.

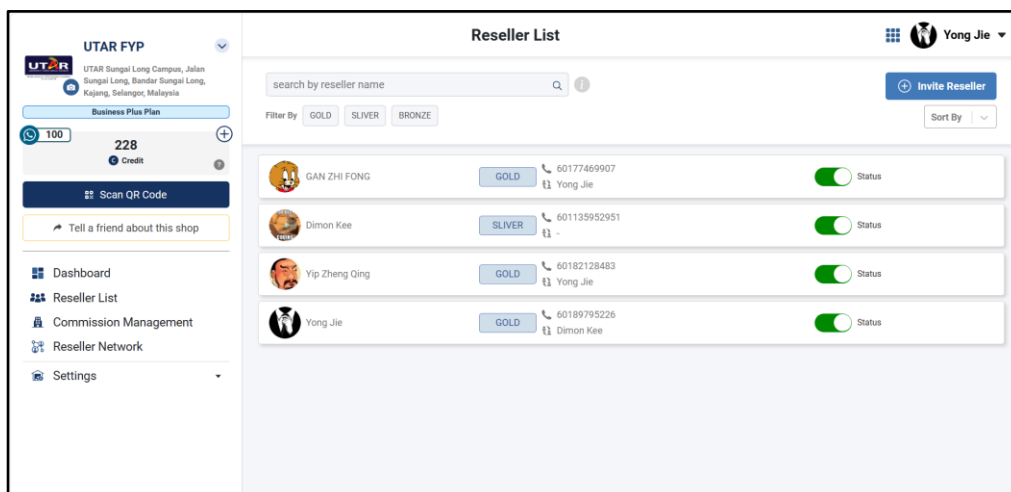


Figure 6-14: Reseller List Page

6.4.2.1 View Reseller List Activity

When the administrator performs a redirect action, filter action, sort action, or search action on the Reseller List Page in Figure 6-14, it will enter the `getResellerList` function by passing filter, sort, and search data to the backend using `getresellerdetail` API in Figure 6-15.

```
const getResellerList = async (shopCode, searchTerm = "", filterTerm = "", sortTerm = "") => {
  const payload = {
    shopcode: shopCode,
    search: searchTerm,
    filter: filterTerm,
    sort: sortTerm,
  };

  try {
    const response = await axios.get('reseller/getresellerdetail', { params: payload });
    return response.data;
  } catch (error) {
    window.alert("No Reseller found!");
    console.error('Error fetching reseller details:', error);
    throw error;
  }
};
```

Figure 6-15: `getResellerList` function

In the backend, the `getResellerDetail` endpoint will filter the reseller list based on the filter and sort data. It also arranges the filtered list based on sort data. Lastly, it will return the processed list of resellers as a response in Figure 6-16.

```

# reseller detail
@api_view(['GET'])
@csrf_exempt
def getResellerDetail(request):
    try:
        shop_code_param = request.GET.get('shopcode')
        search_term = request.GET.get('search')
        filter_term = request.GET.get('filter')
        sort_term = request.GET.get('sort')

        if not shop_code_param:
            return Response({"message": "Shop code parameter is required"}, status=400)

        reseller_instances = ResellerDetail.objects.filter(shop_code=shop_code_param)

        if search_term is not None:
            reseller_instances = reseller_instances.filter(member_card__user__name__icontains=search_term)

        if filter_term is not None:
            reseller_instances = reseller_instances.filter(tier__icontains=filter_term)

        if sort_term == 'ascending':
            reseller_instances = reseller_instances.order_by('member_card__nickName')

        if sort_term == 'descending':
            reseller_instances = reseller_instances.order_by('-member_card__nickName')

        if not reseller_instances.exists():
            return Response({"message": "Reseller not found"}, status=404)

        serializer = ResellerDetailSerializer(reseller_instances, many=True)
        return Response(serializer.data, status=200)

    except ResellerDetail.DoesNotExist:
        return Response({"message": "Reseller not found"}, status=404)

    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-16: getResellerDetail endpoint

6.4.2.2 Update Reseller Status Activity

When the administrator clicks on the status button next to each reseller row on the Reseller List Page in Figure 6-14, it will enter the `handleStatusChange` function by passing the phone number to the backend using the `updateresellerstatus` API in Figure 6-17.

```

const handleStatusChange = (event, phone_number) => {
    axios.put("reseller/updateresellerstatus", { phone_number: phone_number })
        .then(response => {
            console.log(response.data.message);
            handleResellersRefresh(searchTerm, 1, sortBy);
        })
        .catch(error => {
            console.error('Error updating status:', error);
        });
};

```

Figure 6-17: handleStatusChange function

In the backend, the updateResellerStatus endpoint will change the current status of the reseller based on the phone number passed in Figure 6-18.

```

@api_view(['PUT'])
@csrf_exempt
def updateResellerStatus(request):
    if request.method == 'PUT':
        try:
            phone_number = request.data.get('phone_number')

            if not phone_number:
                return JsonResponse(data={"message": "Phone number is required"}, status=400, safe=False)

            reseller_instance = ResellerDetail.objects.filter(member_card__user__globalUsername=phone_number).first()

            if not reseller_instance:
                return JsonResponse(data={"message": "Reseller not found"}, status=404, safe=False)

            reseller_instance.status = not reseller_instance.status
            reseller_instance.save()

            return JsonResponse(data={"message": "Status updated successfully"}, status=200, safe=False)

        except Exception as e:
            return JsonResponse(data={"error": str(e)}, status=500, safe=False)
    else:
        return JsonResponse(data={"message": "Invalid request method"}, status=405, safe=False)

```

Figure 6-18: updateResellerStatus endpoint

6.4.2.3 Send Invitation Request Link Activity

Figure 6-19: Invite Reseller Module

When the administrator clicks on the invite reseller button on the Reseller List Page in Figure 6-14, it will pop up the invite reseller module in Figure 6-19. If the administrator enters the detail and clicks on invite reseller in Figure 6-19, it will enter the inviteReseller function by passing data entered to the backend using createresellerdetail API in Figure 6-20. If the API does not

respond to any error, it will continue to send an SMS invitation to the reseller to the backend using merchantblastsmsfromsystem API in Figure 6-20.

```
const inviteReseller = async () => {
  const postData = {
    shop_code: shopCode,
    phone_number: resellerFormattedPhone,
    tier: selectedTier,
    referrer_phone_number: referrerFormattedPhone,
  };

  axios.post('reseller/createreSELLERdetail', postData)
    .then(response => {
      const payload = {
        shopCode: shopCode,
        smsData: JSON.stringify({ messages: ["We're excited to extend an invitation for you to join our team. We'll be in touch with you soon."], targetAudience: 'Custom Audiences', customAudiences: JSON.stringify([{ phone: resellerFormattedPhone }]), customVariables: {}, isTag: false, isFunnel: false, isCustom: true, selectedClient: null, version: reactLocalStorage.get("version"),
      };
      axios.post('merchant/merchantblastsmsfromsystem', qs.stringify(payload), { timeout: 90000 })
        .then(response => {
          openNotification('success', 'Invite successfully!');
          handleResellersRefresh(searchTerm, 1, sortBy);
        })
        .catch(error => {
          console.error('Error creating setting:', error);
          openNotification('error', 'Error Inviting. Please try again.');
```

Figure 6-20: inviteReseller function

In the backend, the createResellerDetail endpoint will create a reseller based on the passed-in data and store it in the ResellerDetail table in Figure 6-21.

```

@api_view(['POST'])
@csrf_exempt
def createResellerDetail(request):
    if request.method == 'POST':
        serializer = CreateResellerDetailSerializer(data=request.data)
        phone_number = request.data.get('phone_number')
        referrer_phone_number = request.data.get('referrer_phone_number')

        if serializer.is_valid():
            if phone_number:
                existing_member_card = MemberCard.objects.filter(user__globalUsername=phone_number).first()
                serializer.validated_data['member_card'] = existing_member_card

            if referrer_phone_number:
                existing_referrer = ResellerDetail.objects.filter(member_card__user__globalUsername=referrer_phone_number).first()
                serializer.validated_data['referrer'] = existing_referrer

            serializer.save()

            return Response({"message": "Reseller created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-21: createResellerDetail endpoint

6.4.3 Agent and Reseller Tier Structure

The Agent and Reseller Tier Structure module is divided into view reseller diagram existing member card and set tier naming activity.

6.4.3.1 View Reseller Diagram Activity

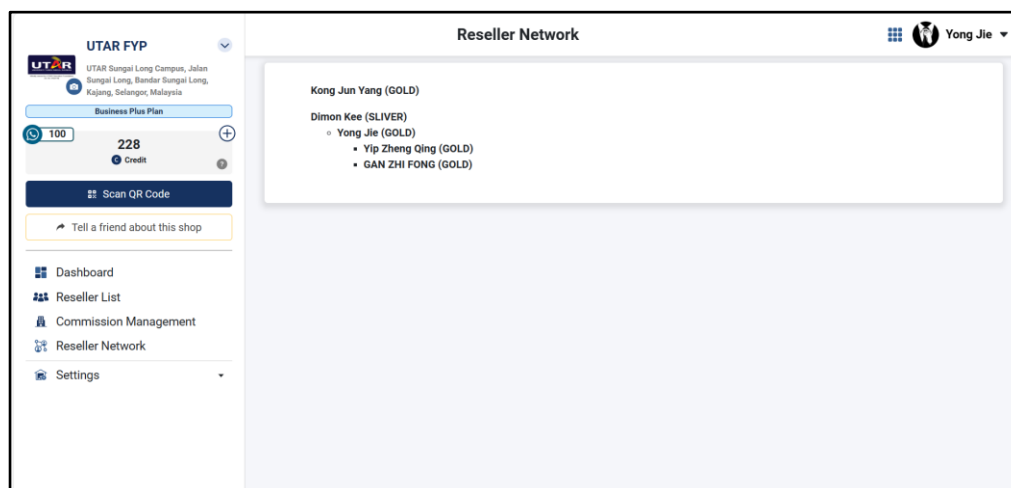


Figure 6-22: Reseller Network Page

When the administrator redirects to the Reseller Network Page in Figure 6-22, it will enter the useEffect function by passing shop code data to the backend using the getresellertree API in Figure 6-23.


```

useEffect(() => {
  setMounted(true);

  axios.get(`reseller/getresellertree?shopcode=${shopCode}`)
    .then(({ data }) => {
      console.log(data)
      setTreeData(data);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
}, []);

```

Figure 6-23: useEffect function (Reseller Network Page)

In the backend, the getResellerTree endpoint will respond to a JSON that is arranged based referrer in Figure 6-24.

```

@api_view(['GET'])
@csrf_exempt
def getResellerTree(request):
    try:
        shop_code_param = request.GET.get('shopcode')

        if not shop_code_param:
            return Response({"message": "Shop code parameter is required"}, status=400)

        def create_tree(label, tier):
            return {
                "label": label,
                "tier": tier,
                "parent": None,
                "children": [],
            }

        def find_and_add_child(node, new_node, tier):
            if node["label"] == new_node["parent"]:
                for child in node["children"]:
                    if child["label"] == new_node["label"]:
                        return # Node already exists, avoid duplicates
                node["children"].append(new_node)
            else:
                for child in node["children"]:
                    find_and_add_child(child, new_node, tier)

        reseller_instances = ResellerDetail.objects.filter(shop_code=shop_code_param)

        trees = []

        for reseller_instance in reseller_instances:
            label = reseller_instance.member_card.user.name
            tier = reseller_instance.tier
            parent_label = reseller_instance.referrer.member_card.user.name if reseller_instance.referrer else None

            if parent_label is None:
                new_tree = create_tree(label, tier)
                trees.append(new_tree)
            else:
                for tree in trees:
                    find_and_add_child(tree, {"label": label, "parent": parent_label, "children": [], "tier": tier}, tier)

        return Response(trees, status=200)

    except ResellerDetail.DoesNotExist:
        return Response({"message": "No Reseller"}, status=404)

    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-24: getResellerTree endpoint

6.4.3.2 Set Tier Naming activity

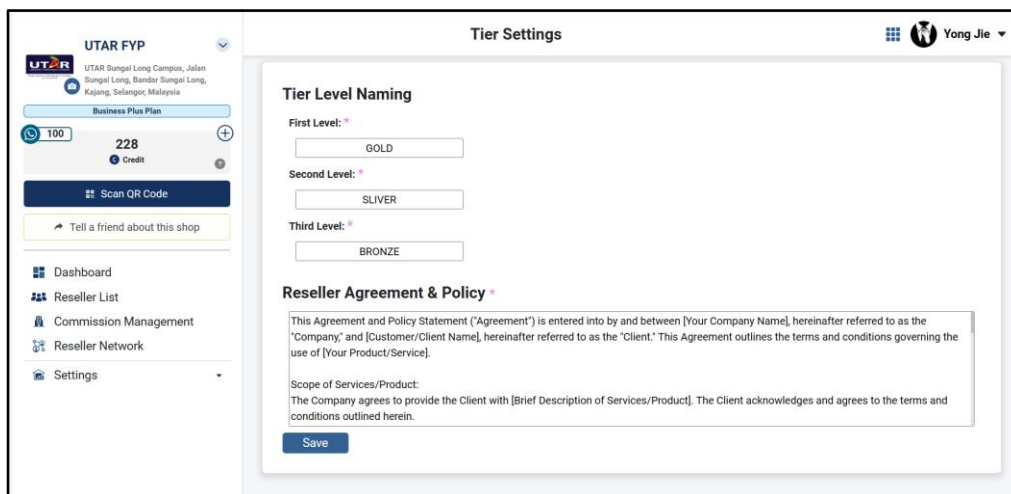


Figure 6-25: Tier Settings Page

When the administrator redirects to the Tier Setting Page in Figure 6-25, it will enter the `useEffect` function by passing shop code data to the backend using the `getsetting` API in order to retrieve the current setting data in Figure 6-26.

```
useEffect(() => {
  setMounted(true);
  axios.get(`reseller/getsetting?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setFirstTier(data.first_tier);
      setSecondTier(data.second_tier);
      setThirdTier(data.third_tier);
      setAgreement(data.agreement);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
}, []);
```

Figure 6-26: getSetting function

In the backend, the `getSetting` endpoint will respond to a JSON of setting data stored in the database in Figure 6-27.

```

@api_view(['GET'])
@csrf_exempt
def getSetting(request):
    shop_code_param = request.GET.get('shopcode')

    try:
        setting_instance = Setting.objects.get(shop_code=shop_code_param)
        response_data = {
            "first_tier": setting_instance.first_tier_level,
            "second_tier": setting_instance.second_tier_level,
            "third_tier": setting_instance.third_tier_level,
            "agreement": setting_instance.agreement_and_policy,
        }

        return JsonResponse(data=response_data, status=200)

    except Setting.DoesNotExist:
        return JsonResponse(data={"message": "Setting not found"}, status=404)
    except Exception as e:
        return JsonResponse(data={"error": str(e)}, status=500)

```

Figure 6-27: getSettings endpoint

When the administrator that is new to the system clicks on the save button on the Tier Setting Page in Figure 6-25, it will enter the createSetting function by passing data entered by the administrator to the backend using createsetting API in order to create the setting record in Figure 6-28.

```

const createSetting = () => {
    const postData = {
        shop_code: shopCode,
        first_tier_level: firstTier,
        second_tier_level: secondTier,
        third_tier_level: thirdTier,
        agreement_and_policy: agreement,
    };

    axios.post('reseller/createsetting', postData)
        .then(response => {
            console.log(response.data);
        })
        .catch(error => {
            console.error('Error creating setting:', error);
            setSaveStatus(false);
        });
};

```

Figure 6-28: createSetting function

In the backend, the createSetting endpoint creates a new record of data stored in the database in Figure 6-29.

```

@api_view(['POST'])
@csrf_exempt
def createSetting(request):
    if request.method == 'POST':
        serializer = SettingSerializer(data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-29: creatSetting endpoint

When the administrator clicks on the save button on the Tier Setting Page in Figure 6-25, it will enter the updateSetting function by passing data entered by the administrator to the backend using updatesetting API in order to update the setting record in Figure 6-30.

```

const updateSetting = () => {
    const updatedData = {
        first_tier_level: firstTier,
        second_tier_level: secondTier,
        third_tier_level: thirdTier,
        agreement_and_policy: agreement,
    };

    axios.put(`reseller/updatesetting/${shopCode}/`, updatedData)
        .then(response => {
            console.log(response.data);
        })
        .catch(error => {
            createSetting();
        });
};

```

Figure 6-30: updateSetting function

In the backend, the updateSetting endpoint updates the existing record of settings data stored in the database in Figure 6-31.

```

@api_view(['PUT'])
@csrf_exempt
def updateSetting(request, shop_code):
    if request.method == 'PUT':
        # Retrieve the existing setting instance
        setting_instance = get_object_or_404(Setting, shop_code=shop_code)

        # Update the setting instance with the new data
        serializer = SettingSerializer(instance=setting_instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting updated successfully"}, status=200)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-31: updateSetting endpoint

6.4.4 Commission Management

The Commission Management module is divided into allocating commission rate activity, withdrawing commission activity, viewing commission transaction history activity, uploading withdraw transaction slip number activity, and calculating commission activity.

6.4.4.1 Allocate Commission Rate Activity

Figure 6-32: Commission Settings Page

When the administrator redirects to the Commission Management Page in Figure 6-32, it will enter the useEffect function by passing shop code data to the backend using getdirectcommission, getindirectcommission, getleadershipcommission, and getpoolcommission API in order to retrieve the current commission data from the database in Figure 6-33.

In the backend, `getDirectCommission`, `getIndirectCommission`, `getLeadershipCommission`, and `getPoolCommission` endpoints will return the commission detail based on the shop code from the database in Figure 6-34.

When the administrator clicks on the save button on Tier Setting Page in Figure 6-32, it will enter the save function in Figure 6-35 by passing data entered by the administrator to the backend using `createDirectCommission`, `updateDirectCommission`, `createIndirectCommission`, `updateIndirectCommission`, `createLeadershipCommission`, `updateLeadershipCommission`, `createPoolCommission` and `updatePoolCommission` function in order to use the API in Figure 6-36, Figure 6-37, Figure 6-38, and Figure 6-39.

```

useEffect(() => {
  .then(({ data }) => {
    setThirdTier(data.third_tier);
  })
  .catch(error => {
    console.error('Error fetching settings:', error);
  });
  // Direct Commission
  axios.get(`reseller/getdirectcommission?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setFirstD(data.first);
      setSecondD(data.second);
      setThirdD(data.third);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
  // Indirect Commission
  axios.get(`reseller/getindirectcommission?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setEnable(data.enable);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
  // Leadership Commission
  axios.get(`reseller/getleadershipcommission?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setTierTypeL(data.tier_type);
      setFirstL(data.first);
      setSecondL(data.second);
      setThirdL(data.third);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
  // Pool Commission
  axios.get(`reseller/getpoolcommission?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setTierTypeP(data.tier_type);
      setMinimumRequirement(data.minimum_requirement);
      setProfitPortion(data.profit_portion);
      setFirstP(data.first);
      setSecondP(data.second);
      setThirdP(data.third);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });
}, []);

```

Figure 6-33: useEffect function (Commission Settings Page)

```

@api_view(['GET'])
@csrf_exempt
def getDirectCommission(request):
    shop_code_param = request.GET.get('shopcode')

    try:
        setting_instance = DirectCommission.objects.get(shop_code=shop_code_param)
        response_data = {
            "first": setting_instance.first,
            "second": setting_instance.second,
            "third": setting_instance.third,
        }

        return JsonResponse(data=response_data, status=200)

    except Setting.DoesNotExist:
        return JsonResponse(data={"message": "Setting not found"}, status=404)
    except Exception as e:
        return JsonResponse(data={"error": str(e)}, status=500)

@api_view(['GET'])
@csrf_exempt
def getIndirectCommission(request):
    shop_code_param = request.GET.get('shopcode')

    try:
        setting_instance = IndirectCommission.objects.get(shop_code=shop_code_param)
        response_data = {
            "enable": setting_instance.enable,
        }

        return JsonResponse(data=response_data, status=200)

    except Setting.DoesNotExist:
        return JsonResponse(data={"message": "Setting not found"}, status=404)
    except Exception as e:
        return JsonResponse(data={"error": str(e)}, status=500)

@api_view(['GET'])
@csrf_exempt
def getLeadershipCommission(request):
    shop_code_param = request.GET.get('shopcode')

    try:
        setting_instance = LeadershipCommission.objects.get(shop_code=shop_code_param)
        response_data = {
            "tier_type": setting_instance.tier_type,
            "first": setting_instance.first,
            "second": setting_instance.second,
            "third": setting_instance.third,
        }

        return JsonResponse(data=response_data, status=200)

    except Setting.DoesNotExist:
        return JsonResponse(data={"message": "Setting not found"}, status=404)
    except Exception as e:
        return JsonResponse(data={"error": str(e)}, status=500)

@api_view(['GET'])
@csrf_exempt
def getPoolCommission(request):
    shop_code_param = request.GET.get('shopcode')

    try:
        setting_instance = PoolCommission.objects.get(shop_code=shop_code_param)
        response_data = {
            "tier_type": setting_instance.tier_type,
            "minimum_requirement": setting_instance.minimum_requirement,
            "profit_portion": setting_instance.profit_portion,
            "first": setting_instance.first,
            "second": setting_instance.second,
            "third": setting_instance.third,
        }

        return JsonResponse(data=response_data, status=200)

    except Setting.DoesNotExist:
        return JsonResponse(data={"message": "Setting not found"}, status=404)
    except Exception as e:
        return JsonResponse(data={"error": str(e)}, status=500)

```

Figure 6-34: getDirectCommission, getIndirectCommission, getLeadershipCommission and getPoolCommission endpoints


```

const save = () => {
  updateDirectCommission();
  updateIndirectCommission();
  updateLeadershipCommission();
  updatePoolCommission();

  if (saveStatus) {
    openNotification('success', 'Settings saved successfully!');
  } else {
    openNotification('error', 'Error saving settings. Please try again.');
```

Figure 6-35: save function

```

// Direct Commission
const createDirectCommission = () => {
  const postData = {
    shop_code: shopCode,
    first: firstD,
    second: secondD,
    third: thirdD,
  };

  axios.post('reseller/createdirectcommission', postData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      console.error('Error creating setting:', error);
      setSaveStatus(false);
    });
};

const updateDirectCommission = () => {
  const updatedData = {
    first: firstD,
    second: secondD,
    third: thirdD,
  };

  axios.put(`reseller/updatedirectcommission/${shopCode}/`, updatedData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      createDirectCommission();
    });
};

```

Figure 6-36: createDirectCommission & updateDirectCommission function

```

// Indirect Commission
const createIndirectCommission = () => {
  const postData = {
    shop_code: shopCode,
    enable: enable,
  };

  axios.post('reseller/createindirectcommission', postData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      console.error('Error creating setting:', error);
      setSaveStatus(false);
    });
};

const updateIndirectCommission = () => {
  const updatedData = {
    enable: enable,
  };

  axios.put(`reseller/updateindirectcommission/${shopCode}/`, updatedData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      createIndirectCommission();
    });
};

```

Figure 6-37: createIndirectCommission & updateIndirectCommission function

```

// Leadership Commission
const createLeadershipCommission = () => {
  const postData = {
    shop_code: shopCode,
    tier_type: tier_typeL,
    first: firstL,
    second: secondL,
    third: thirdL,
  };

  axios.post('reseller/createleadershipcommission', postData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      console.error('Error creating setting:', error);
      setSaveStatus(false);
    });
};

const updateLeadershipCommission = () => {
  const updatedData = {
    tier_type: tier_typeL,
    first: firstL,
    second: secondL,
    third: thirdL,
  };

  axios.put(`reseller/updateleadershipcommission/${shopCode}/`, updatedData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      createLeadershipCommission();
    });
};

```

Figure 6-38: createLeadershipCommission & updateLeadershipCommission function

```
// Pool Commission
const createPoolCommission = () => {
  const postData = {
    shop_code: shopCode,
    tier_type: tier_typeP,
    minimum_requirement: minimum_requirement,
    profit_portion: profit_portion,
    first: firstP,
    second: secondP,
    third: thirdP,
  };

  axios.post('reseller/createpoolcommission', postData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      console.error('Error creating setting:', error);
      setSaveStatus(false);
    });
};

const updatePoolCommission = () => {
  const updatedData = {
    tier_type: tier_typeP,
    minimum_requirement: minimum_requirement,
    profit_portion: profit_portion,
    first: firstP,
    second: secondP,
    third: thirdP,
  };

  axios.put(`reseller/updatepoolcommission/${shopCode}/`, updatedData)
    .then(response => {
      console.log(response.data);
    })
    .catch(error => {
      createPoolCommission();
    });
};
```

Figure 6-39: createPoolCommission & updatePoolCommission function

In the backend, the createDirectCommission endpoint will create a new row of data in the DirectCommission table, while the updateDirectCommission endpoint will update the existing row of data into the DirectCommission table based on the shop code in Figure 6-40.

```

@api_view(['POST'])
@csrf_exempt
def createDirectCommission(request):
    if request.method == 'POST':
        serializer = DirectCommissionSerializer(data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

@api_view(['PUT'])
@csrf_exempt
def updateDirectCommission(request, shop_code):
    if request.method == 'PUT':
        # Retrieve the existing setting instance
        setting_instance = get_object_or_404(DirectCommission, shop_code=shop_code)

        # Update the setting instance with the new data
        serializer = DirectCommissionSerializer(instance=setting_instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting updated successfully"}, status=200)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-40: createDirectCommission & updateDirectCommmission endpoints

In the backend, the createIndirectCommission endpoint will create a new row of data into the IndirectCommission table, while the updateIndirectCommission endpoint will update the existing row of data into the IndirectCommission table based on the shop code in Figure 6-41.

```

@api_view(['POST'])
@csrf_exempt
def createIndirectCommission(request):
    if request.method == 'POST':
        serializer = IndirectCommissionSerializer(data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

@api_view(['PUT'])
@csrf_exempt
def updateIndirectCommission(request, shop_code):
    if request.method == 'PUT':
        # Retrieve the existing setting instance
        setting_instance = get_object_or_404(IndirectCommission, shop_code=shop_code)

        # Update the setting instance with the new data
        serializer = IndirectCommissionSerializer(instance=setting_instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting updated successfully"}, status=200)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-41: createIndirectCommission & updateIndirectCommmission endpoints

In the backend, the createLeadershipCommission endpoint will create a new row of data in the LeadershipCommission table, while the updateLeadershipCommission endpoint will update the existing row of data into the LeadershipCommission table based on the shop code in Figure 6-42.

```

@api_view(['POST'])
@csrf_exempt
def createLeadershipCommission(request):
    if request.method == 'POST':
        serializer = LeadershipCommissionSerializer(data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

@api_view(['PUT'])
@csrf_exempt
def updateLeadershipCommission(request, shop_code):
    if request.method == 'PUT':
        # Retrieve the existing setting instance
        setting_instance = get_object_or_404(LeadershipCommission, shop_code=shop_code)

        # Update the setting instance with the new data
        serializer = LeadershipCommissionSerializer(instance=setting_instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting updated successfully"}, status=200)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-42: createLeadershipCommission & updateLeadershipCommission endpoints

In the backend, the createPoolCommission endpoint will create a new row of data in the PoolCommission table, while the updatePoolCommission endpoint will update the existing row of data into the PoolCommission table based on the shop code in Figure 6-43.

```

@api_view(['POST'])
@csrf_exempt
def createPoolCommission(request):
    if request.method == 'POST':
        serializer = PoolCommissionSerializer(data=request.data)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting created successfully"}, status=201)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

@api_view(['PUT'])
@csrf_exempt
def updatePoolCommission(request, shop_code):
    if request.method == 'PUT':
        # Retrieve the existing setting instance
        setting_instance = get_object_or_404(PoolCommission, shop_code=shop_code)

        # Update the setting instance with the new data
        serializer = PoolCommissionSerializer(instance=setting_instance, data=request.data, partial=True)

        if serializer.is_valid():
            serializer.save()
            return Response({"message": "Setting updated successfully"}, status=200)
        else:
            return Response({"errors": serializer.errors}, status=400)

    return Response({"message": "Invalid request method"}, status=400)

```

Figure 6-43: createPoolCommission & updatePoolCommission endpoints

6.4.4.2 Withdraw Commission Activity

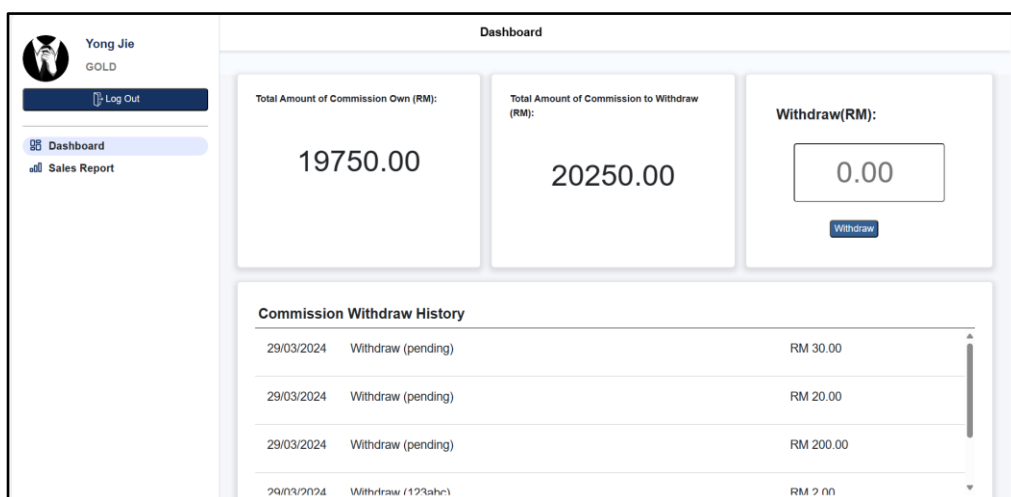


Figure 6-44: Reseller Dashboard Page

When the reseller enters the amount and clicks on the withdraw button in the Reseller Dashboard Page in Figure 6-44, it will enter the `handleWithdraw` function by passing data entered to the backend using `makewithdrawrequest` API in order to withdraw commission in Figure 6-45.

```
const handleWithdraw = () => {
  if (withdrawAmount < commissionWallet) {
    axios.post('reseller/makewithdrawrequest', {
      shopCode: shopCode,
      phone_number: id,
      amount: withdrawAmount,
    })
    .then(response => {
      openNotification('success', 'Withdraw Requested!');
      setWithdrawWallet((withdrawWallet + withdrawAmount).toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
      setCommissionWallet((commissionWallet - withdrawAmount).toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
      setWithdrawAmount(0);
    })
    .catch(error => {
      openNotification('error', 'Error Withdrawing. Please try again.');
```

Figure 6-45: `handleWithdraw` function

In the backend, `makeWithdrawRequest` endpoints will create a new withdraw record to the database based on the detail passed in Figure 6-46.

```

# withdraw
@api_view(['POST'])
@csrf_exempt
def makeWithdrawRequest(request):
    try:
        shop_code = request.data.get('shopCode')
        phone_number = request.data.get('phone_number')
        amount = request.data.get('amount')

        reseller_instance = ResellerDetail.objects.get(member_card__user__globalUsername=phone_number)

        withdraw_data = {
            "shop_code": shop_code,
            "reseller": reseller_instance.pk,
            "description": "Withdraw (pending)",
            "amount": Decimal(amount),
            "status": False,
        }

        serializer = WithdrawRequestSerializer(data=withdraw_data)
        serializer.is_valid(raise_exception=True)
        serializer.save()

        reseller_instance.withdraw_wallet += Decimal(amount)
        reseller_instance.commission_wallet -= Decimal(amount)
        reseller_instance.save()

        return Response({"message": "Withdraw record created successfully"}, status=200)

    except ResellerDetail.DoesNotExist:
        return Response({"error": "Reseller not found for the given ID"}, status=404)
    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-46: makeWithdrawRequest endpoint

6.4.4.3 View Commission Transaction History Activity

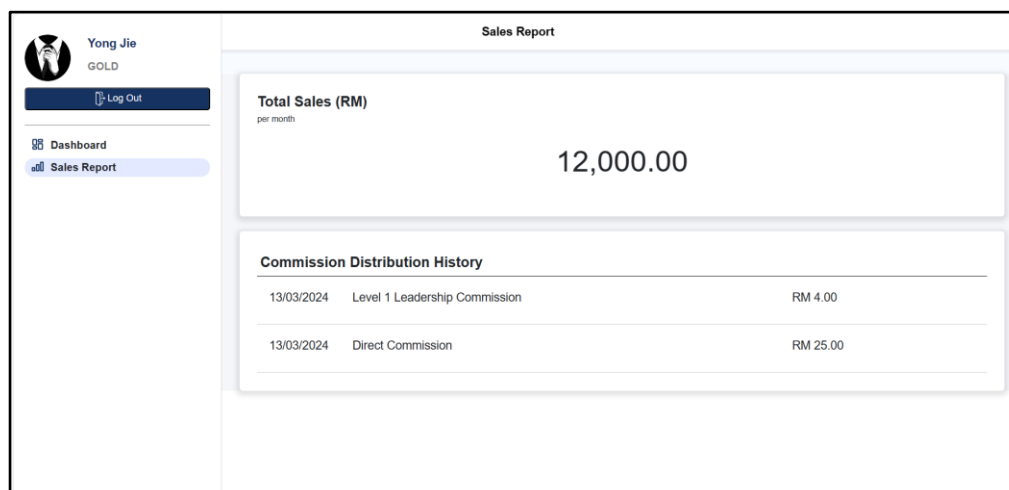


Figure 6-47: Sales Report Page

When the reseller redirects to the Sales Report Page in Figure 6-47, it will enter the `useEffect` function by passing the shop code to the backend using the `getcommissionlist` API in order to retrieve the commission list in Figure 6-48.

```

useEffect(() => {
  window.scrollTo(0, 0);
  startAxios();

  const retrievedData = JSON.parse(sessionStorage.getItem('myDataKey'));
  const hasPhoneNumber = retrievedData?.phoneNumber;
  console.log(hasPhoneNumber)
  if (hasPhoneNumber === undefined) {
    router.push('/resellerlogin');
  } else {
    setId(retrievedData.phoneNumber);
  }

  axios.get('reseller/getresellerprofile', {
    params: {
      phone_number: id,
    },
  })
  .then(response => {
    setProfile(response.data);
    setShopCode(response.data.shop_code);
    axios.get('reseller/getcommissionlist', {
      params: {
        phoneNumber: id,
        shopCode: response.data.shop_code,
      }
    })
    .then(response => {
      setCommissionList(response.data.commission_records);
      setTotalSales(response.data.total_amount_sales.toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
    })
    .catch(error => {
      console.error('Error fetching commission records:', error);
    });
  })
  .catch(error => {
    console.error(error);
  });

  setComponentMounted(true);
}, [id, router]);

```

Figure 6-48: useEffect Function (Sales Report Page)

In the backend, getCommissionList endpoints will retrieve the list of withdrawal records from the database based on the shop code passed in Figure 6-49.

```

@api_view(['GET'])
@csrf_exempt
def getCommissionList(request):
    try:
        shop_code = request.query_params.get('shopCode')
        phone_number = request.query_params.get('phoneNumber')
        current_month = timezone.now().month
        if not phone_number:
            return JsonResponse(data={"message": "Phone number and shop code are required"}, status=400, safe=False)

        if not shop_code:
            return JsonResponse(data={"message": "Phone number and shop code are required"}, status=400, safe=False)

        reseller_record_instance = CommissionRecord.objects.filter(
            reseller__member_card__user__globalUsername=phone_number,
            shop_code=shop_code
        ).order_by('-createDate')

        serializer = CommissionRecordListSerializer(reseller_record_instance, many=True)

        total_amount_sales = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__member_card__user__globalUsername=phone_number,
            shop_code=shop_code
        ).aggregate(Sum('bill__total'))['bill__total__sum'] or 0

        return Response({
            "commission_records": serializer.data,
            "total_amount_sales": total_amount_sales
        }, status=200)

    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-49: getCommissionList endpoint

6.4.4.4 Upload Withdraw Transaction Slip Number Activity

Figure 6-50: Commission Management Page

Figure 6-51: Update Commission Withdraw Module

When the administrator clicks on the pay button of each withdrawal request in the Commission Management Page in Figure 6-50, it will pop up the Update Commission Withdraw Module. After the administrator enters the transaction number and clicks on submit, it will enter the handleTransactionNumber function by passing the transaction number to the backend using updatewithdrawstatus API in order to update the transaction number in Figure 6-52.

```

const handleTransactionNumber = () => {
  axios.put("reseller/withdrawstatus", { id: commissionId, transactionNumber: transactionNumber })
    .then(response => {
      console.log(response.data.message);
      window.location.reload();
    })
    .catch(error => {
      console.error('Error updating status:', error);
    });
};

```

Figure 6-52: handleTransactionNumber Function

In the backend, updateWithdrawStatus endpoints will update the withdraw record by saving the transaction record in the database based on the transaction number passed in Figure 6-53.

```

@api_view(['PUT'])
@csrf_exempt
def updateWithdrawStatus(request):
    if request.method == 'PUT':
        try:
            id = request.data.get('id')
            transaction_number = request.data.get('transactionNumber')

            record_instance = WithdrawRecord.objects.get(id=id)

            record_instance.status = not record_instance.status
            record_instance.description = "Withdraw (No: "+transaction_number+)"
            record_instance.save()

            reseller = record_instance.reseller
            reseller.withdraw_wallet -= record_instance.amount
            reseller.save()
            return JsonResponse(data={"message": "Status updated successfully"}, status=200, safe=False)

        except Exception as e:
            return JsonResponse(data={"error": str(e)}, status=500, safe=False)

    else:
        return JsonResponse(data={"message": "Invalid request method"}, status=405, safe=False)

```

Figure 6-53: updateWithdrawStatus endpoint

6.4.4.5 Calculate Commission Activity

```

axios.post('reseller/makepaymentcommission', payload2, { timeout: 20000 })
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error making payment commission:', error);
  });

```

Figure 6-54: makepaymentcommission function

The makepaymentcommission function is added to the Gather Deal existing code base. When the administrator processes a bill, it will enter the makepaymentcommission function by passing payment data to the backend

using the `makePaymentCommission` API in order to calculate and save the commission of the bill in Figure 6-54.

```

@api_view(['POST'])
@csrf_exempt
def makePaymentCommission(request):
    try:
        shop_code = request.data.get('shopCode')
        receipt_id = request.data.get('receiptID')
        bill_no = request.data.get('billNo')

        if not shop_code or not receipt_id or not bill_no:
            return Response({"message": "Shop code, receipt ID, bill No are required"}, status=400)

        direct_commission_instance = DirectCommission.objects.filter(shop_code=shop_code).first()
        if direct_commission_instance is None:
            return Response({"message": "Shop code not register in reseller"}, status=400)

        indirect_commission_instance = IndirectCommission.objects.filter(shop_code=shop_code).first()
        if indirect_commission_instance is None:
            return Response({"message": "Shop code not register in reseller"}, status=400)

        leadership_commission_instance = LeadershipCommission.objects.filter(shop_code=shop_code).first()
        if leadership_commission_instance is None:
            return Response({"message": "Shop code not register in reseller"}, status=400)

        commission_settings_instance = Setting.objects.filter(shop_code=shop_code).first()
        if commission_settings_instance is None:
            return Response({"message": "Shop code not register in reseller"}, status=400)

        pos_bill_instance = POSBill.objects.select_related('owner__referrer').filter(billNumber=bill_no).first()

        reseller_instance = MemberCard.objects.filter(user=pos_bill_instance.owner.referrer, shop__shopCode=shop_code).first()

        first_layer_reseller = ResellerDetail.objects.filter(member_card=reseller_instance).first()
        second_layer_reseller = first_layer_reseller.referrer if first_layer_reseller.referrer else None
        third_layer_reseller = second_layer_reseller.referrer if second_layer_reseller.referrer else None

        first_tier = commission_settings_instance.first_tier_level
        second_tier = commission_settings_instance.second_tier_level
        third_tier = commission_settings_instance.third_tier_level

        if first_layer_reseller is None:
            return Response({"message": "no referrer"}, status=400)

    if commission_settings_instance is None:
        return Response({"message": "Shop code not register in reseller"}, status=400)

    # direct commission
    if first_layer_reseller.status:
        if first_layer_reseller.tier == first_tier:
            direct_commission_rate = direct_commission_instance.first
        elif first_layer_reseller.tier == second_tier:
            direct_commission_rate = direct_commission_instance.second
        elif first_layer_reseller.tier == third_tier:
            direct_commission_rate = direct_commission_instance.third

        direct_commission_record_data = {
            "shop_code": shop_code,
            "bill": pos_bill_instance.pk,
            "reseller": first_layer_reseller.pk,
            "description": "Direct Commission",
            "amount": (Decimal(str(pos_bill_instance.subtotal))/100*direct_commission_rate).quantize(Decimal('0.00'))
        }
        serializer = CommissionRecordSerializer(data=direct_commission_record_data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        first_layer_reseller.commission_wallet += (Decimal(str(pos_bill_instance.subtotal))/100*(direct_commission_rate)).quantize(Decimal('0.00'))
        first_layer_reseller.save()

    #indirect commission
    if indirect_commission_instance.enable and first_layer_reseller.tier != first_tier and second_layer_reseller is not None:
        if second_layer_reseller.tier == first_tier:
            indirect_commission_rate = direct_commission_instance.first
        elif second_layer_reseller.tier == second_tier:
            indirect_commission_rate = direct_commission_instance.second
        elif second_layer_reseller.tier == third_tier:
            indirect_commission_rate = direct_commission_instance.third

        indirect_commission_record_data = {
            "shop_code": shop_code,
            "bill": pos_bill_instance.pk,
            "reseller": second_layer_reseller.pk,
            "description": "Indirect Commission",
            "amount": (Decimal(str(pos_bill_instance.subtotal))/100*(indirect_commission_rate-direct_commission_rate)).quantize(Decimal('0.00'))
        }
        serializer = CommissionRecordSerializer(data=indirect_commission_record_data)
        serializer.is_valid(raise_exception=True)
        serializer.save()

```


In the backend, makePaymentCommission endpoints will calculate and save the record of direct, indirect, and leadership commissions in the database based on the bill detail passed in Figure 6-55.

6.4.5 Sales, Analytics, and Reporting

The Sales, Analytics, and Reporting module is divided into reseller view sales analysis activity, and administrator view sales analysis activity.

6.4.5.1 Administrator View Sales Analysis Activity

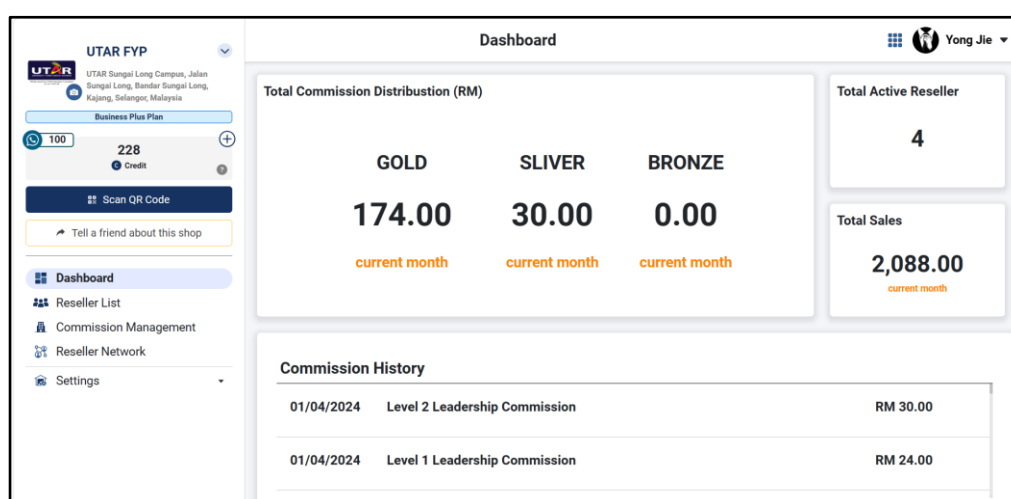


Figure 6-56: Administrator Dashboard Page

When the administrator redirects to the Administrator Dashboard Page in Figure 6-56, it will enter the useEffect function by shop code to the backend using getresellercount, gettotalcommission, and getcommissionlist API in order to retrieve sales analysis in Figure 6-52.

```

useEffect(() => {
  setMounted(true);

  // Total number of Reseller
  axios.get(`reseller/getresellercount?shopcode=${shopCode}`)
    .then(({ data }) => {
      setResellerCount(data.reseller_count);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });

  axios.get(`reseller/getsetting?shopcode=${shopCode}`)
    .then(({ data }) => {
      setFirstTier(data.first_tier);
      setSecondTier(data.second_tier);
      setThirdTier(data.third_tier);
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });

  axios.get(`reseller/gettotalcommission?shopcode=${urlShopCode}`)
    .then(({ data }) => {
      setFirstTotal(data.first.toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
      setSecondTotal(data.second.toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
      setThirdTotal(data.third.toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
      setTotalSales(data.total_sales.toLocaleString('en-US', { minimumFractionDigits: 2, maximumFractionDigits: 2 }));
    })
    .catch(error => {
      console.error('Error fetching settings:', error);
    });

  axios.get(`reseller/getallcommissionlist`, {
    params: {
      shopCode: shopCode,
    }
  })
    .then(response => {
      setCommissionList(response.data.commission_records);
    })
    .catch(error => {
      console.error('Error fetching commission records:', error);
    });
}, []);

```

Figure 6-57: useEffect Function (Administrator Dashboard Page)

In the backend, getResellerCount endpoints will calculate and return the total amount of resellers in the database based on the shop code passed in Figure 6-58.

```

@api_view(['GET'])
@csrf_exempt
def getResellerCount(request):
    try:
        shop_code_param = request.GET.get('shopcode')

        if not shop_code_param:
            return Response({"message": "Shop code parameter is required"}, status=400)

        reseller_instances = ResellerDetail.objects.filter(shop_code=shop_code_param, status=True)

        reseller_length = len(reseller_instances)

        if reseller_length == 0:
            return Response({"message": "No Reseller"}, status=404)

        return Response({"reseller_count": reseller_length}, status=200)

    except ResellerDetail.DoesNotExist:
        return Response({"message": "No Reseller"}, status=404)

    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-58: getResellerCount endpoint

In the backend, `getTotalCommission` endpoints will calculate and return the total amount of commission distribution based on each of the tiers in the database in Figure 6-59.

```
# display total commission distribution
@api_view(['GET'])
@csrf_exempt
def getTotalCommission(request):
    try:
        shop_code = request.GET.get('shopcode')
        if not shop_code:
            return Response({"message": "Shop code is required"}, status=400)

        commission_settings_instance = Setting.objects.filter(shop_code=shop_code).first()
        if commission_settings_instance is None:
            return Response({"message": "Shop code not register in reseller"}, status=400)

        first_tier = commission_settings_instance.first_tier_level
        second_tier = commission_settings_instance.second_tier_level
        third_tier = commission_settings_instance.third_tier_level
        current_month = timezone.now().month

        first_total_amount_resellers_commission = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__tier=first_tier,
            reseller__shop_code=shop_code
        ).aggregate(Sum('amount'))['amount__sum'] or 0

        second_total_amount_resellers_commission = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__tier=second_tier,
            reseller__shop_code=shop_code
        ).aggregate(Sum('amount'))['amount__sum'] or 0

        third_total_amount_resellers_commission = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__tier=third_tier,
            reseller__shop_code=shop_code
        ).aggregate(Sum('amount'))['amount__sum'] or 0

        total_amount_sales = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__shop_code=shop_code
        ).aggregate(Sum('bill__total'))['bill__total__sum'] or 0

        return Response({
            "first": first_total_amount_resellers_commission,
            "second": second_total_amount_resellers_commission,
            "third": third_total_amount_resellers_commission,
            "total_sales": total_amount_sales
        }, status=200)

    except Exception as e:
        return Response({"error": str(e)}, status=500)
```

Figure 6-59: `getTotalCommission` endpoint

In the backend, `getCommissionList` endpoints will calculate and return the total amount of sales based on the shop code in the database in Figure 6-60.

```

# list of commission
@api_view(['GET'])
@csrf_exempt
def getCommissionList(request):
    try:
        shop_code = request.query_params.get('shopCode')
        phone_number = request.query_params.get('phoneNumber')
        current_month = timezone.now().month
        if not phone_number:
            return JsonResponse(data={"message": "Phone number and shop code are required"}, status=400, safe=False)

        if not shop_code:
            return JsonResponse(data={"message": "Phone number and shop code are required"}, status=400, safe=False)

        reseller_record_instance = CommissionRecord.objects.filter(
            reseller__member_card__user__globalUsername=phone_number,
            shop_code=shop_code
        ).order_by('-createDate')

        serializer = CommissionRecordListSerializer(reseller_record_instance, many=True)

        total_amount_sales = CommissionRecord.objects.filter(
            createDate__month=current_month,
            reseller__member_card__user__globalUsername=phone_number,
            shop_code=shop_code
        ).aggregate(Sum('bill__total'))['bill__total__sum'] or 0

        return Response({
            "commission_records": serializer.data,
            "total_amount_sales": total_amount_sales
        }, status=200)
    except Exception as e:
        return Response({"error": str(e)}, status=500)

```

Figure 6-60: getCommissionList endpoint

CHAPTER 7

SYSTEM TESTING AND EVALUATION

7.1 Introduction

This chapter mainly discusses the testing method of the project. The testing of this project includes unit testing and using SUS to conduct system usability testing.

7.2 Unit Testing

This project uses unit testing to test every function of the web applications to ensure that the requirement specification is met. Below contains a total of 50 test cases.

7.2.1 User Management Module

Table 7-1: Test Cases of User Management Model

View Reseller List						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC01	Admin navigates to view reseller list	1. Click on the reseller list inside the sidebar		Display reseller list	Display reseller list	Successful
Update Reseller Status						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC02	When the admin updates the reseller status to inactive	1. Click on the status slider button		Reseller status changed to inactive	Reseller status changed to inactive	Successful
TC03	When admin updates the reseller status to activate	1. Click on the status slider button		Reseller status change to activate	Reseller status change to activate	Successful
Send Invitation Request Link						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC04	When admin invites a reseller with a referrer	<ol style="list-style-type: none"> 1. Enter the reseller's phone number 2. Select tier 3. Enter the reseller's referrer phone number 	<ul style="list-style-type: none"> • Reseller's phone number • Tier • Reseller's referrer phone number 	The reseller received an SMS invitation	The reseller received an SMS invitation	Successful

		4. Click invite button				
TC05	When admin invites a reseller without a referrer	<ol style="list-style-type: none"> 1. Enter the reseller's phone number 2. Select tier 3. Click invite button 	<ul style="list-style-type: none"> • Reseller's phone number • Tier 	The reseller received an SMS invitation	The reseller received an SMS invitation	Successful
TC06	When admin invites a reseller who is not a company member	<ol style="list-style-type: none"> 1. Enter the non-member's phone number 2. Select tier 3. Click invite button 	<ul style="list-style-type: none"> • Non-member's phone number • Tier 	Display error message	Display error message	Successful
TC07	When admin invites a reseller with the referrer's phone number that is not in the company	<ol style="list-style-type: none"> 1. Enter the reseller's phone number 2. Select tier 3. The referrer's phone number that is not in the company 4. Click invite button 	<ul style="list-style-type: none"> • Reseller's phone number • Tier • The referrer's phone number that is not in the company 	Display error message	Display error message	Successful
TC08	When admin invites a reseller by emptying the reseller's phone number field	<ol style="list-style-type: none"> 1. Empty reseller's phone number 2. Select tier 3. Enter the reseller's referrer phone number 4. Click invite button 	<ul style="list-style-type: none"> • Tier • Reseller's referrer phone number 	Display error message	Display error message	Successful

TC09	When admin invites a reseller by entering the reseller's phone number field other than that number	<ol style="list-style-type: none"> 1. Enter other than the number in the reseller's phone number 2. Select tier 3. Enter the reseller's referrer phone number 4. Click invite button 	<ul style="list-style-type: none"> • Reseller's phone number: abc!#\$ • Tier • Reseller's referrer phone number 	Display error message	Display error message	Successful
TC10	When admin invites a reseller by entering the reseller's referrer phone number field other than that number	<ol style="list-style-type: none"> 5. Enter the reseller's phone number 6. Select tier 7. Enter other than the number in the reseller's referrer phone number 5. Click invite button 	<ul style="list-style-type: none"> • Reseller's phone number • Tier • reseller's referrer: abc!#\$ 	Display error message	Display error message	Successful

7.2.2 Agent and Reseller Registration

Table 7-2: Test Cases of Agent and Reseller Registration Module

Login Reseller Account						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC11	When a reseller enters a valid phone number and password	<ol style="list-style-type: none"> 1. Enter a valid phone number 2. Enter valid password 3. Click login button 	<ul style="list-style-type: none"> • valid phone number • valid password 	Redirect to "Reseller dashboard"	Redirect to "Reseller dashboard"	Successful

TC12	When a reseller enters an invalid phone number and password	<ol style="list-style-type: none"> 1. Enter an invalid phone number 2. Enter invalid password 3. Click login button 	<ul style="list-style-type: none"> • invalid phone number • invalid password 	Display error message	Display error message	Successful
TC13	When a reseller enters a valid phone number and an invalid password	<ol style="list-style-type: none"> 1. Enter a valid phone number 2. Enter invalid password 3. Click login button 	<ul style="list-style-type: none"> • valid phone number • invalid password 	Display error message	Display error message	Successful
TC14	When a reseller enters a valid phone number and an empty password	<ol style="list-style-type: none"> 4. Enter a valid phone number 5. Empty password 6. Click login button 	<ul style="list-style-type: none"> • valid phone number 	Display error message	Display error message	Successful
TC15	When a reseller enters an empty phone number	<ol style="list-style-type: none"> 4. Empty phone number 5. Enter password 7. Click login button 	<ul style="list-style-type: none"> • password 	Display error message	Display error message	Successful
TC16	When a reseller enters that number in the phone number	<ol style="list-style-type: none"> 6. Enter other than that number in a phone number 7. Enter password 8. Click login button 	<ul style="list-style-type: none"> • Phone Number: abc!#\$ • password 	Display error message	Display error message	Successful

Register Reseller Account

TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
-------	-------------------	------------	-----------	-----------------	---------------	--------

TC17	When resellers register a new account through an invitation link	<ol style="list-style-type: none"> 1. Enter phone number 2. Enter password 3. Re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Phone number • Password • Re-enter password 	Display the organization's reseller agreement	Display the organization's reseller agreement	Successful
TC18	When a reseller registers a new account through an invitation link with not match password	<ol style="list-style-type: none"> 1. Enter phone number 2. Enter password 3. Re-enter password that does not match 4. Click register button 	<ul style="list-style-type: none"> • Phone number • Password • Not match password 	Display error message	Display error message	Successful
TC19	When a reseller registers a new account through an invitation link with an empty password	<ol style="list-style-type: none"> 1. Enter phone number 2. Empty password 3. Empty re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Phone number 	Display error message	Display error message	Successful
TC20	When a reseller registers a new account through an invitation link with an empty re-password	<ol style="list-style-type: none"> 1. Enter phone number 2. Enter password 3. Empty re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Phone number • Password 	Display error message	Display error message	Successful
TC21	When a reseller registers a new account through an invitation link with an empty password	<ol style="list-style-type: none"> 1. Enter phone number 2. Empty password 3. Enter re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Phone number • Re-enter password 	Display error message	Display error message	Successful

TC22	When a reseller registers a new account through an invitation link with an empty phone number	<ol style="list-style-type: none"> 1. Empty phone number 2. Enter password 3. Re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Password • Re-enter password 	Display error message	Display error message	Successful
TC23	When resellers register a new account through an invitation link enter a phone number other than the number	<ol style="list-style-type: none"> 1. Enter a phone number other than a number 2. Enter password 3. Re-enter password 4. Click register button 	<ul style="list-style-type: none"> • Phone number other than a number • Password • Re-enter password 	Display error message	Display error message	Successful
Respond to the Organization's Reseller Agreement						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC24	Admin agrees with the organization's reseller agreement	<ol style="list-style-type: none"> 1. Click on the agree button 		Redirect to the reseller dashboard page	Redirect to the reseller dashboard page	Successful
TC25	Admin disagrees with the organization's reseller agreement	<ol style="list-style-type: none"> 1. Click on the cancel button 		Redirect to the reseller register page	Redirect to the reseller register page	Successful

7.2.3 Agent and Reseller Tier Structure Module

Table 7-3: Test Cases of Agent and Reseller Tier Structure Module

View Reseller Diagram						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC26	Admin navigates to the reseller network page	1. Click on the reseller network inside the sidebar		Display reseller diagram	Display reseller diagram	Successful
Set Tier Naming						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC27	When admin updates tier naming	1. Enter tier naming 2. Click on the save button	<ul style="list-style-type: none"> Tier naming 	Display new tier naming	Display new tier naming	Successful
TC28	When admin updates tier naming with empty	1. Empty tier naming 2. Click on the save button		Display error message	Display error message	Successful

7.2.4 Commission Management Module

Table 7-4: Test Cases of Commission Management Module

Allocate Commission Rate						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC29	Admin update the rate of Direct Commission	<ol style="list-style-type: none"> 1. Enter new Direct Commission rates 2. Click on the save button 	<ul style="list-style-type: none"> • Direct Commission rates 	Display new Direct Commission rates	Display new Direct Commission rates	Successful
TC30	Admin update the rate of Direct Commission with empty	<ol style="list-style-type: none"> 1. Empty Direct Commission rates 2. Click on the save button 		Display error message	Display error message	Successful
TC31	Admin enable the Indirect Commission	<ol style="list-style-type: none"> 1. Tick on Indirect Commission 2. Click on the save button 		Display Indirect Commission enabled	Display Indirect Commission enabled	Successful
TC232	Admin disable the Indirect Commission	<ol style="list-style-type: none"> 1. Untick on the Indirect Commission 2. Click on the save button 		Display Indirect Commission disabled	Display Indirect Commission disabled	Successful
TC33	Admin update the rate of Leadership Commission	<ol style="list-style-type: none"> 1. Enter new Leadership Commission rates 2. Click on the save button 	<ul style="list-style-type: none"> • Leadership Commission rates 	Display new Leadership Commission rates	Display new Leadership Commission rates	Successful

TC34	Admin update the rate of Leadership Commission with empty	1. Empty Leadership Commission rates 2. Click on the save button		Display error message	Display error message	Successful
TC35	Admin update the rate of Pool Commission	1. Enter new Pool Commission rates 2. Click on the save button	<ul style="list-style-type: none"> Pool Commission rates 	Display new Pool Commission rates	Display new Pool Commission rates	Successful
TC35	Admin update the rate of Pool Commission with empty	1. Empty Pool Commission rates 2. Click on the save button		Display error message	Display error message	Successful
Withdraw Commission						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC36	The reseller withdraws the commission with a valid amount	1. Enter an amount for withdraw 2. Click on the withdraw button	<ul style="list-style-type: none"> Withdraw amount 	Display misused amount of commission wallet and added amount of withdraw wallet	Display misused amount of commission wallet and added amount of withdraw wallet	Successful
TC37	The reseller withdraws commission with an amount that is larger	1. Enter an amount that is larger than the commission wallet	<ul style="list-style-type: none"> Amount that is larger than the commission wallet 	Display error message	Display error message	Successful

	than the commission wallet	2. Click on the withdraw button				
TC38	The reseller withdraws the commission without entering an amount	1. Empty amount for withdraw 2. Click on the withdraw button		Display error message	Display error message	Successful
View Commission Transaction History						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC39	Admin navigates to view commission transaction history	1. Click on the dashboard inside the sidebar		Display commission transaction history	Display commission transaction history	Successful
Upload Withdraw Transaction Slip Number						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC40	Admin uploads the withdrawal transaction number	1. Click on the play button on the specific withdrawal request 2. Enter the transaction number 3. Click on the upload button	<ul style="list-style-type: none"> Withdraw transaction number 	Display the transaction number that is uploaded	Display the transaction number that is uploaded	Successful

TC41	Admin uploads the withdraw transaction number with an empty	<ol style="list-style-type: none"> 1. Click on the play button on the specific withdrawal request 2. Empty the transaction number 3. Click on the upload button 		Display error message	Display error message	Successful
Calculate Commission						
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result	Status
TC42	Calculate direct commission	<ol style="list-style-type: none"> 1. Check out the invoice through the POS system 	<ul style="list-style-type: none"> • Direct Commissions (First Tier: 25% of the commission value) • Product price RM100 • Customer's referrer: Yong Jie (First-tier) 	Distributed Direct Commissions of RM25 to Yong Jie	Distributed Direct Commissions of RM25 to Yong Jie	Successful
TC43	Calculate direct commission	<ol style="list-style-type: none"> 1. Check out the invoice through the POS system 	<ul style="list-style-type: none"> • Direct Commissions (Second Tier: 18% of the commission value) • Product price RM100 • Customer's referrer: Yong Jie (Second tier) 	Distributed Direct Commissions of RM18 to Yong Jie	Distributed Direct Commissions of RM18 to Yong Jie	Successful
TC44	Calculate direct commission	<ol style="list-style-type: none"> 1. Check out the invoice through the POS system 	<ul style="list-style-type: none"> • Direct Commissions (Third Tier: 12% of the commission value) • Product price RM100 	Distributed Direct Commissions of RM12 to Yong Jie	Distributed Direct Commissions of RM12 to Yong Jie	Successful

			<ul style="list-style-type: none"> • Customer's referrer: Yong Jie (Third tier) 			
TC45	Calculate indirect commission	1. Check out the invoice through the POS system	<ul style="list-style-type: none"> • Indirect Commissions are enabled • Direct Commissions <ul style="list-style-type: none"> - Silver Tier: 18% of the commission value - Gold Tier: 25% of the commission value • Product price RM100 • Customer's referrer: Yong Jie (Second tier) • Yong Jie's referrer: Dimon (First-tier) 	Distributed Indirect Commissions RM7 to Dimon	Distributed Indirect Commissions RM7 to Dimon	Successful
TC46	Calculate indirect commission	1. Check out the invoice through the POS system	<ul style="list-style-type: none"> • Indirect Commissions are enabled • Direct Commissions <ul style="list-style-type: none"> - Silver Tier: 18% of the commission value - Gold Tier: 25% of the commission value • Product price RM100 • Customer's referrer: Yong Jie (First-tier) 	No Indirect Commissions have been distributed	No Indirect Commissions have been distributed	Successful

TC47	Calculate leadership commission	1. Check out the invoice through the POS system	<ul style="list-style-type: none"> • Leadership Commissions <ul style="list-style-type: none"> - 1st Level (5%) - 2nd Level (3%) - 3rd Level (2%) - Tier type: Gold • Product price RM100 • Customer's referrer: <ul style="list-style-type: none"> - 1st Level (Yong Jie) [Gold] - 2nd Level (Dimon) [Gold] - 3rd Level (Max) [Gold] - 	Distributed Leadership Commissions: <ul style="list-style-type: none"> - Yong Jie: RM5 - Dimon: RM3 - Max: RM2 	Distributed Leadership Commissions: <ul style="list-style-type: none"> - Yong Jie: RM5 - Dimon: RM3 - Max: RM2 	Successful
TC48	Calculate pool commission	1. Admin click on the distribute pool commission button	<ul style="list-style-type: none"> • Pool Commissions <ul style="list-style-type: none"> - 1st Level (50%) - 2nd Level (30%) - 3rd Level (30%) - portion of total profit: 5% • RM100,000 profit for this month • 1st Level Resellers: <ul style="list-style-type: none"> - Reseller A - Reseller B - Reseller C - Reseller D - Reseller E 	Distributed Pool Commissions: <ul style="list-style-type: none"> - Reseller A: RM 5,555.55 - Reseller B: RM 5,555.55 - Reseller C: RM 5,555.55 - Reseller D: RM 5,555.55 - Reseller E: RM 5,555.55 - Reseller F: RM 5,555.55 	Distributed Pool Commissions: <ul style="list-style-type: none"> - Reseller A: RM 5,555.55 - Reseller B: RM 5,555.55 - Reseller C: RM 5,555.55 - Reseller D: RM 5,555.55 - Reseller E: RM 5,555.55 - Reseller F: RM 5,555.55 	Successful

			<ul style="list-style-type: none"> - Reseller F - Reseller G - Reseller H - Reseller I • 2nd Level Resellers: <ul style="list-style-type: none"> - Reseller J - Reseller K - Reseller L • 3rd Level Resellers: <ul style="list-style-type: none"> - Reseller N 	<ul style="list-style-type: none"> - Reseller G: RM 5,555.55 - Reseller H: RM 5,555.55 - Reseller I: RM 5,555.55 - Reseller J: RM10,000 - Reseller K: RM10,000 - Reseller L: RM10,000 - Reseller N: RM30,000 	<ul style="list-style-type: none"> - Reseller G: RM 5,555.55 - Reseller H: RM 5,555.55 - Reseller I: RM 5,555.55 - Reseller J: RM10,000 - Reseller K: RM10,000 - Reseller L: RM10,000 - Reseller N: RM30,000 	
--	--	--	--	---	---	--

7.2.5 Sales, Analytics, and Reporting Module

Table 7-5: Test Cases of Sales, Analytics, and Reporting Module

View Sales Analysis					
TC ID	Test Case Summary	Test Steps	Test Data	Expected Result	Actual Result

TC49	Admin navigates to view sales analysis	1. Click on the dashboard inside the sidebar		Display sales analysis	Display sales analysis
TC50	The reseller navigates to view the sales analysis	2. Click on the dashboard inside the sidebar		Display sales analysis	Display sales analysis

7.3 System Usability Testing

System Usability Testing (SUS) is being used in this project to assess the system. A standardised questionnaire called SUS can be used to assess a system's usability. It is a simple-to-deploy, reliable, and popular method for assessing a system's effectiveness, efficiency, and user satisfaction.

The system usability test invited a total of five participants to conduct which are two experts from Enlliance Management Sdn Bhd and 3 normal people. There are two primary sections of testing which are the system admin side and system reseller side. The testing methods for doing usability testing are mainly in-person testing with participants and online testing using online Meetings. The developer tested the system's usability with the participant in Figure 7-1.

Before the exam, the host will provide a brief overview of the system's history. Then will next read each test scenario to the participants, asking them to perform the tasks that correspond to each scenario. A form will be given to the participants once they have finished all the activities.



Figure 7-1: System Usability Test

7.3.1 Test Scenario

Table 7-6 contains the test scenarios used to conduct the system usability test for the Admin Side. Table 7-7 contains the test scenarios used to conduct the system usability test for the Reseller Side.

Table 7-6: Usability Testing Scenario (Admin Side)

No.	Test Scenario Title	Description
1.	View Sales Analyse	Scenario: <ul style="list-style-type: none"> • You want to view the overview of the company in the application. Task: <ul style="list-style-type: none"> • Perform view company sales analysis action.
2.	Invite New Reseller	Scenario: <ul style="list-style-type: none"> • You want to invite a new gold-tier reseller to join the company without a referrer in the application. Task: <ul style="list-style-type: none"> • Perform invite new reseller action.
3.	Change Reseller Status	Scenario: <ul style="list-style-type: none"> • You want to disable a reseller in the application. Task: <ul style="list-style-type: none"> • Perform change reseller status action.
4.	Distribute Pool Commission	Scenario: <ul style="list-style-type: none"> • You want to distribute pool commission to the reseller. Task: <ul style="list-style-type: none"> • Perform distribute pool commission action.
5.	Update Withdrawal Transaction Number	Scenario: <ul style="list-style-type: none"> • You want to update the withdrawal transaction number that you have transferred. Task:

		<ul style="list-style-type: none"> • Perform update withdrawal transaction number action.
6.	View Reseller Network	<p>Scenario:</p> <ul style="list-style-type: none"> • You want to view the overview network of all resellers. <p>Task:</p> <ul style="list-style-type: none"> • Perform view reseller network action.
7.	Update Tier naming and reseller agreement	<p>Scenario:</p> <ul style="list-style-type: none"> • You want to change the tier naming and agreement of the system. <p>Task:</p> <ul style="list-style-type: none"> • Perform update tier setting action.
8.	Update Commission Setting	<p>Scenario:</p> <ul style="list-style-type: none"> • You want to change the rate of commission in the system. <p>Task:</p> <ul style="list-style-type: none"> • Perform update commission setting action.

Table 7-7: Usability Testing Scenario (Reseller Side)

No.	Test Scenario Title	Description
1.	Register an account	<p>Scenario:</p> <ul style="list-style-type: none"> • You want to register an account in the application. <p>Task:</p> <ul style="list-style-type: none"> • Perform register a reseller account action.
2.	Login the account	<p>Scenario:</p> <ul style="list-style-type: none"> • You want to access the application. <p>Task:</p> <ul style="list-style-type: none"> • Perform login reseller account action.

3.	Logout from the application	Scenario: <ul style="list-style-type: none"> You want to logout from the application. Task: <ul style="list-style-type: none"> Perform logout reseller account action.
4.	Withdraw commission from the application	Scenario: <ul style="list-style-type: none"> You want to withdraw RM10 from the application. Task: <ul style="list-style-type: none"> Perform commission withdrawal action.
5.	Check the commission withdrawal record	Scenario: <ul style="list-style-type: none"> You want to check the previous commission withdrawal record. Task: <ul style="list-style-type: none"> Perform view commission withdrawal record action.
6.	View sales report	Scenario: <ul style="list-style-type: none"> You want to check the sales report. Task: <ul style="list-style-type: none"> Perform view sales report action.

7.3.2 System Usability Test Result

The four primary methods to generate the SUS score were given by Sauro (2011). The SUS score may be generated using the data provided in [Appendix B](#). The four actions are as follows:

- i. Deduct 1 point from the score for questions with odd numbers.
- ii. Deduct by 5 points from the score for questions with even numbers.
- iii. Multiply by 2.5 after adding together all of the question final scores.
- iv. The final result of the calculation is the product's SUS usability score.

Table 7-8 displays the mapping of the SUS score with the adjective rating by Bangor, Kortum, and Miller (2009). Tables 7.15 and 7.16 will see the application of the adjective rating in this project.

Table 7-8: SUS Score Interpretation (Bangor, et al., 2009)

Adjective Rating	SUS Score
Worst Imaginable	12.5
Awful	20.3
Poor	35.7
OK	50.9
Good	71.4
Excellent	85.5
Best Imaginable	90.9

Table 7-9: SUS Score of Admin Side Application

Participants	Usability Score per Questions										Total	Percentage
	1	2	3	4	5	6	7	8	9	10		
Kong Khai Jien	2	3	2	3	2	2	3	3	3	2	25	62.5
Dimon Kee Yong Kit	3	3	3	3	3	3	3	3	3	2	29	72.5
Lim Fang Nie	4	4	4	3	4	4	4	4	4	3	38	95
Kok Shao Huai	4	4	3	4	4	4	4	4	4	4	39	97.5
Chew Jia Jie	4	4	4	4	4	4	4	4	4	4	40	100
Average SUS Score											85.5	
Grade												A
Adjective Rating												Excellent

Table 7-10: SUS Score of Reseller Side Application

Participants	Usability Score per Questions										Total	Percentage
	1	2	3	4	5	6	7	8	9	10		
Kong Khai Jien	3	4	2	2	3	4	2	3	2	3	28	70
Dimon Kee Yong Kit	4	2	4	3	3	2	3	2	3	3	29	72.5
Lim Fang Nie	4	4	4	3	4	4	4	4	3	3	37	92.5
Kok Shao Huai	3	4	4	4	4	4	4	3	4	3	37	92.5
Chew Jia Jie	4	4	4	4	4	4	4	4	4	4	40	100
Average SUS Score											85.5	
Grade												A
Adjective Rating												Excellent

CHAPTER 8

CONCLUSION AND DISCUSSION

8.1 Conclusions

In conclusion, this project took a total of six months to complete. The main purpose of the project is to use the system to manage a large number of resellers in an organization. In order to obtain accurate user requirements, a stakeholder consultation technique was used, and the valuable data was then formulated for the project's objective and requirement specification. Following the system development, to ensure the consistency, security, and stability of web applications, this project uses development inside the current Gather Deal environment. In addition, connecting the system with the existing AWS services is also used to ensure that the project runs smoothly.

After the system is developed, unit testing and system usability testing, are carried out to ensure that the system can run stably in the working environment. The average SUS score for the application was 85.5%. Furthermore, the three objectives shown in the list below were successfully accomplished:

- 1. To develop a real-time web reseller application to monitor the organization reseller's activities.**

By using the Agent and Reseller System, the admin can monitor the organization's reseller in real-time. In addition, web applications use the React UI library to ensure the uniformity of the interface, so that users can handle the system smoothly, thus abandoning the traditional process of using paper to record.

- 2. To facilitate the process of commissions and tiers of the resellers in the organization.**

By using the Agent and Reseller System, the calculation of the commission distribution will be auto generated by the system. In

addition, reseller records with the network will be recorded in the system which may advance the process of administering.

3. To deploy web applications using cloud services.

This project makes use of EC2, Elastic Beanstalk, RDS, and S3 services from Amazon. These services guarantee that users stay within budget while using the reliable project's web application online.

8.2 Limitations and Future Enhancement

8.2.1 Limitations

Time restrictions are the major issues that are frequently faced while creating a comprehensive and complex Agent and Reseller system. Concerns about usability centre on making sure the program is simple to use, capable of giving agents and resellers a smooth experience, and intuitive. On the other side, the project's complexity and the requirement for adaptations may lengthen the development cycle, thus delaying the deployment of the system and impeding its capacity to satisfy urgent business requirements. It is essential to deal with these issues by offering an effective solution that meets user expectations and streamlines the development process to achieve the business objectives.

Developing a complete and complex Agent and Reseller system is a time-consuming endeavour that can span several months to a year or more. This is because agent and reseller systems can be specified and customized for each very niche situation. A customized solution is required since each organization may have different needs, business processes, and workflows. To create a system that seamlessly integrates with the organization's activities, the development team must have a complete understanding of the unique demands of the organization, its agents, and its resellers. Additionally, to guarantee that the system operates perfectly and complies with all relevant security and regulatory requirements, thorough testing and quality assurance are crucial. Despite the time commitment, the finished system is strong and effective, enabling agents and resellers to carry out their duties with ease and enhancing overall business performance.

8.2.2 Future Enhancement

An efficient solution to the problems would be to use an iterative process. Firstly, deploying a stripped-down version of the system at first enables speedier deployment, addressing urgent business requirements while continuing to improve and add functionality in later releases. This strategy enables a more flexible and user-centred development process, guarantees the efficacy of the system, and effectively addresses usability issues, all within a realistic timeline.

A pragmatic solution to address the challenges in developing a complete and complex Agent and Reseller system, iteratively updating and improving the system based on user feedback and changing requirements. Essential features may be supplied fast by using an agile and incremental development strategy, and early user feedback can direct future enhancements. This approach not only reduces time-to-market but also guarantees that the system adapts over time to users' demands, resulting in a more sophisticated and user-centric solution. Additionally, this strategy reduces the dangers of creating a comprehensive system all at once, promoting ongoing improvement and encouraging increased participation from agents and resellers in customizing the system to suit their requirements.

REFERENCES

- Anderson, J., 2019. *I Am a Supplier. What Are the Benefits of Using a Reseller Arrangement?*. [Online] Available at: <https://legalvision.com.au/reseller-arrangement-benefits/> [Accessed 20 July 2023].
- Anon., n.d. *Performio vs CaptivateIQ vs Everstage Comparison*. [Online] Available at: <https://www.getapp.com/hr-employee-management-software/a/performio/compare/captivateiq-vs-everstage/> [Accessed 11 April 2024].
- Bangor, A., Kortum, P. & Miller, J., 2009. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *Journal of Usability Studies*, 4(3), pp. 114-123.
- Berni, A. & Borgianni, Y., 2021. FROM THE DEFINITION OF USER EXPERIENCE TO A FRAMEWORK TO CLASSIFY ITS APPLICATIONS IN DESIGN. *Proceedings of the Design Society*, 27 7, Volume 1, pp. 1627-1636.
- Business Transformation Career Advice Database DevOps Observability, 2022. *What Is DevOps?*. [Online] Available at: <https://orangematter.solarwinds.com/2022/03/21/what-is-devops/> [Accessed 12 July 2023].
- Carter, R., 2023. *What does reseller mean? A 101 guide on how the reselling model works..* [Online] Available at: <https://ecommerce-platforms.com/glossary/reseller> [Accessed 17 July 2023].
- Forbes Business Development Council, 2019. *11 Smart Ways To Structure Your Sales Team's Commission*. [Online] Available at: <https://www.forbes.com/sites/forbesbusinessdevelopmentcouncil/2019/07/11/11-smart-ways-to-structure-your-sales-teams-commission/?sh=7ce4da4f58bc> [Accessed 20 July 2023].
- Gupta, B., Mittal, P. & Mufti, T., 2021. A Review on Amazon Web Service (AWS), Microsoft Azure & Google Cloud Platform (GCP) Services. *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and*

Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India.

Hasani, R. A., Muhammad, Y. R., Sukmasetya, P. & Yudatama, U., 2023. Design and evaluating information system for admission student in University Muhammadiyah of Magelang using rapid application development approach. *3RD BOROBUDUR INTERNATIONAL SYMPOSIUM ON SCIENCE AND TECHNOLOGY 2021*, 2706(1).

Hayes, A., 2023. *Indirect Sales: What it is, How it Works*. [Online] Available at: <https://www.investopedia.com/terms/i/indirect-sales.asp> [Accessed 20 July 2023].

kissflow, 2023. *What is Rapid Application Development (RAD)? An Ultimate Guide for 2023*. [Online] Available at: <https://kissflow.com/application-development/rad/rapid-application-development/> [Accessed 18 July 2023].

Li, W., Zhou, Y., Luo, S. & Dong, Y., 2022. Design Factors to Improve the Consistency and Sustainable User Experience of Responsive Interface Design. *Sustainability*, 14(15).

Maurya, S. et al., 2021. A Study on Cloud Computing: A Review.

Mayank, G. & Raju, S., 2021. DevOps: A Historical Review and Future Works. *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 366-371.

Mohammad, G. S. et al., 2019. Waterative Model: an Integration of the Waterfall and Iterative Software Development Paradigms. *Database Systems Journal*, 10(1), pp. 75-81.

Mukherjee, S., 2019. Benefits of AWS in Modern Cloud.

Pérez, J. F., Wang, W. & Casale, G., 2015. Towards a DevOps Approach for Software Quality Engineering. *WOSP '15: Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*, pp. 5-10.

Podmurnyi, S., 2022. *Business App Development Cost in 2022: Factors & Estimation*. [Online]

Available at: <https://www.visartech.com/blog/how-much-app-development->

cost/

[Accessed 29 July 2023].

Rachma, N. & Muhlas, I., 2022. Comparison Of Waterfall And Prototyping Models In Research And Development (R&D) Methods For Android-Based Learning Application Design. *Jurnal Inovatif : Inovasi Teknologi Informasi dan Informatika*, 5(1).

Widestadh, C., 2021. *App Maintenance Cost Can Be Three Times Higher than Development Cost*. [Online]

Available at: <https://www.techstep.io/articles/app-maintenance-cost-can-be-three-times-higher-than-development-cost>

[Accessed 27 July 2023].

APPENDICES

Appendix A: Stakeholder Consultations Discussion

The system will consist of two interfaces: one for resellers and another for administrators.

Reseller interface

- Existing resellers will need to log in using their phone number and password.
- New resellers will require an invitation link from administrators to register, during which the system will display the reseller agreement.
- Resellers will have two distinct wallets:
 - **Commission Wallet:** Accumulates all earned commissions from reseller activities.
 - **Withdraw Wallet:** Facilitates transfers from the commission wallet. The company performs monthly cash outs from the withdraw wallet to the reseller's bank.
- Resellers will be able to access and review their sales records through the interface.

Administrators interface

- Provide a comprehensive overview of sales to view the overall health of the company's reseller.
- A complete list of all registered resellers within the system.
- A list of requests made by resellers for commission withdrawals.
- Upload transaction slips as part of the cash-out withdrawal request process.
- A visual network diagram illustrating the interconnections among all resellers.
- Specific naming to each tier of resellers within the network.
- Adjust commission rates for various types, including Direct, Indirect, Leadership, and Pool commissions.

How Commission works

All the commission percentage is based on the commissionable value (CV) of the item

Direct Commissions

- Rewarding with varying commission rates according to their tier.

Example:

- Bronze Tier: 12% of the commission value
 - Silver Tier: 18% of the commission value
 - Gold Tier: 25% of the commission value
- ✓ Reseller A, who is in the Bronze Tier, sells a product worth RM 10,000. Reseller A would earn a direct commission of RM 1,200 (12% of RM 10,000) for this sale.
 - ✓ Reseller B, who is in the Silver Tier, sells a product worth RM 10,000. Reseller B would earn a direct commission of RM 1,800 (18% of RM 10,000) for this sale.
 - ✓ Reseller C, who is in the Gold Tier, sells a product worth RM 10,000. Reseller C would earn a direct commission of RM 2,500 (25% of RM 10,000) for this sale.

Indirect Commissions

- Resellers receive a percentage of the sales commission value, and part of this commission is distributed to the upper-level resellers based on their tiers.

Example:

- ✓ The Bronze reseller will receive a direct commission of 12% of RM 1,000, which is RM 120 and the remaining 13% (25% - 12%) will be passed to the upper-level resellers.
 - If the upper-level reseller is a Gold reseller, it will take all 13% of the indirect commission.
 - If the upper-level reseller is a Silver reseller, it will take only 6% (18%-12%) of the indirect commission.

- If the upper-level reseller is a Bronze reseller, it will not receive any indirect commission.
- ✓ The Silver reseller will receive a direct commission of 18% of RM 1,000, which is RM 180 and the remaining 7% (25% - 18%) will be passed to the upper-level resellers.
 - If the upper-level reseller is a Gold reseller, it will take all the 7% of the indirect commission.
 - If the upper-level reseller is a Silver reseller, it will not receive any indirect commission.
 - If the upper-level reseller is a Bronze reseller, it will not receive any indirect commission.
- ✓ The Gold reseller will receive a direct commission of 25% of RM 1,000, which is RM 250, and no remaining will be passed to the upper-level resellers.

Leadership Commissions

- 1st Level (5%): You earn a 5% commission on all the sales made directly by your personally recruited resellers.
- 2nd Level (3%): You also earn a 3% commission on sales made by resellers recruited by your 1st level Gold resellers.
- 3rd Level (2%): You earn an additional 2% commission on sales made by resellers recruited by your 2nd level Gold resellers.

Example

- ✓ Let's say you are a Gold reseller, and you've recruited two resellers: Silver Reseller A and Gold Reseller B.
- ✓ 1st Level (5%): You earn a 5% commission on all the sales made directly by Reseller A and Reseller B. So, if Reseller A makes RM4,000 in sales and Reseller B makes RM8,000, you earn 5% of both, which is RM200 (5% of RM4,000) from Reseller A and RM400 (5% of RM8,000) from Reseller B.
- ✓ Now, Reseller B recruits Gold Reseller C and Bronze Reseller D:

- ✓ 2nd Level (3%): You also earn a 3% commission on all the sales made by Reseller C and Reseller D. If Reseller C makes RM3,200 in sales and Reseller D makes RM6,000, you earn 3% of both, which is RM96 (3% of RM3,200) from Reseller C and RM180 (3% of RM6,000) from Reseller D.
- ✓ Now, Reseller C recruits Gold Reseller E, and Silver F:
- ✓ 3rd Level (2%): You earn an additional 2% commission on all the sales made by Reseller E and Reseller F. If Reseller E makes RM2,400 in sales and Reseller F makes RM4,800, you earn 2% of both, which is RM48 (2% of RM2,400) from Reseller E and RM96 (2% of RM4,800) from Reseller F.
- ✓ In total, for this example, your leadership commission earnings in Malaysian Ringgit (RM) would be $RM200 + RM400 + RM96 + RM180 + RM48 + RM96 = RM1,020$.
- ✓ So, your total leadership commission earnings in RM would be RM1,020 for this example.

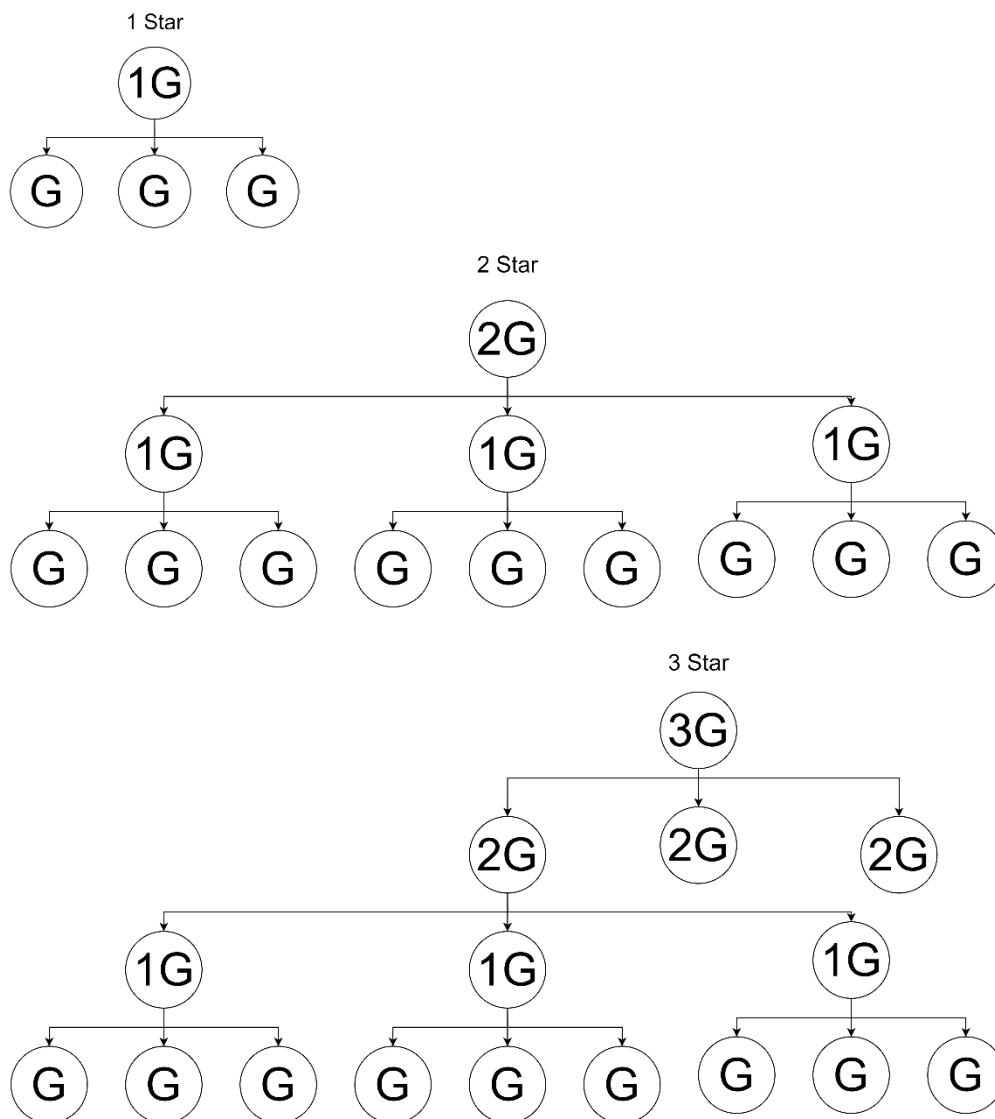
Pool Commissions

1 Star Gold -- 3 different Gold

2 Star Gold -- 3 x Different 1 Star Gold

3 Star Gold -- 3 x Different 2 Star Gold

- The company allocates a portion of its total profit (e.g., 5%) to a commission pool.
- 1 Star Gold resellers receive 50% of the pool's commission.
- 2 Star Gold resellers receive 30% of the pool's commission.
- 3 Star Gold resellers receive 30% of the pool's commission.
- Within each star level, the commission from the pool is distributed equally among all resellers at that level.
- All of the stars will require a minimum amount of sales in order to participate in this pool commission



Example:

- ✓ Company Profit: Let's say the company's total profit for a specific period is RM100,000.
- ✓ Pool Allocation: The company allocates 5% of this profit to the commission pool: 5% of RM100,000 = RM5,000.
- ✓ 1 Star Resellers: There are two 1 Star Gold resellers: Reseller A and Reseller B.
 - Reseller A's share: 50% of the pool, so 50% of RM5,000 = RM2,500.
 - Reseller B's share: 50% of the pool, so 50% of RM5,000 = RM2,500.
 - Each 1 Star Gold reseller receives RM2,500 from the pool.

- ✓ 2 Star Resellers: There are three 2 Star Gold resellers: Reseller C, Reseller D, and Reseller E.
 - Each 2 Star Gold reseller's share: 30% of the pool, so 30% of RM5,000 = RM1,500.
 - Each 2 Star Gold reseller receives RM1,500 from the pool.
- ✓ 3 Star Resellers: There are four 3 Star Gold resellers: Reseller F, Reseller G, Reseller H, and Reseller I.
 - Each 3 Star Gold reseller's share: 30% of the pool, so 30% of RM5,000 = RM1,500.
 - Each 3 Star Gold reseller receives RM1,500 from the pool.

Appendix B: System Usability Test Question

1. I think that I would like to use this product frequently. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

2. I found the product unnecessarily complex. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

3. I thought the product was easy to use. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

4. I think that I would need the support of a technical person to be able to use this product. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

5. I found the various functions in the product were well integrated. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

6. I thought there was too much inconsistency in this product. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

7. I imagine that most people would learn to use this product very quickly. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

8. I found the product very awkward to use. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

9. I felt very confident using the product. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

10. I needed to learn a lot of things before I could get going with this product. *
1 2 3 4 5
Strongly Disagree <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> Strongly Agree

Results

SUS(Admin Side)

Name	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Kong Khai Jien	3	2	3	2	3	3	4	2	4	3
Dimon Kee Yong Kit	4	2	4	2	4	2	4	2	4	3
Lim Fang Nie	5	1	5	2	5	1	5	1	5	2
Kok Shao Huai	5	1	4	1	5	1	5	1	5	1
Chew Jia Jie	5	1	5	1	5	1	5	1	5	1

SUS(Reseller Side)

Name	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Kong Khai Jien	4	1	3	3	4	1	3	2	3	2
Dimon Kee Yong Kit	5	3	5	2	4	3	4	3	4	2
Lim Fang Nie	5	1	5	2	5	1	5	1	4	2
Kok Shao Huai	4	1	5	1	5	1	5	2	5	2
Chew Jia Jie	5	1	5	1	5	1	5	1	5	1