

CHAPTER I

INTRODUCTION

The Background, Problem Statement, Final Project Objectives, Scope and Limitations, Final Project Methodology, and Final Project Outline are all explained in the Introduction.

1.1 Background

E-food is an online application designed for the buying and selling of various dishes, particularly Indonesian specialties. It is often referred to as a food ordering application. Each server can only be used by one restaurant, meaning that if another restaurant wishes to integrate the application into their establishment, they are required to purchase the entire application.

Indonesia's history includes a period of colonization by several countries for many centuries. One of the reasons for this colonization was the desire to obtain the abundant spices found in Indonesia. These spices were highly sought after due to their exceptional quality. Indonesian spices are considered among the finest in the world, and their incorporation into dishes enhances their flavors significantly. The exceptional taste resulting from the use of these world-class spices has contributed to the acceptance of Indonesian cuisine by people from other nations.

Indonesian traditional cuisine encompasses a wide range of dishes originating from various regions across the country. The recipes and cooking techniques used in these traditional culinary delights have been passed down from one generation to the next. Each region's traditional cuisine boasts unique

characteristics, flavors, and tastes. There is no definitive reference that can provide an exact count of the diverse range of traditional culinary delights found in Indonesia.

Numerous restaurants specialize in serving Indonesian specialties. Therefore, E-food is a website-based application that can be utilized by any restaurant. This application will offer three user access roles: restaurant owners, kitchen staff or customer service, and customers. The website will feature various functionalities to enhance user experience, such as a chat feature for customers to communicate with the kitchen, sorting options, and estimated preparation times for each item.

1.2 Problem Statement

Based on the background above, below are the problem stated:

1. How to make a website so customers can order food and drinks until it's finished?
2. How to make a website to bring up the estimated time of making orders from customers?
3. How to make a website so that customers and customer service can chat?
4. How to make a website so that you can sort food and beverage menus based on certain criteria?

1.3 Objectives

To make it simpler for customers to see the expected time for cooking food or drinks based on what they requested, this final project intends to develop a web-based application. Additionally, users can utilize the bubble sort method to

filter the food and beverage menu to see only the items that suit them. Customers can also make purchases on this website through the payment process. There will be a chat option on the customer service and customer pages so that people may ask each other questions about lines or anything else.

1.4 Scope and Limitations

1.4.1 Scope

This final project will focus on developing a website so that it can :

1. Show estimate time of making an order
2. Register and Login to system
3. Filter menu using bubble sort
4. Order Product until payment
5. Online chat between customer and customer service

1.4.2 Limitations

The limitation of this web based application are:

1. This web cannot using qris and bank virtual account for payment
2. The chat app will not use encrypt and decrypt
3. User cannot change the password

1.5 Project Methodology

Rapid application development (RAD) is an agile project management technique that is quite common in the creation of software applications. The advantage of this RAD method approach is the quick turnaround on projects,

which makes it a desirable choice for professionals working in hectic situations like software application development. This quick pace was made feasible by RAD's emphasis on cutting down on the planning stage and increasing application prototyping. The diagram of the RAD is shown in figure 1.1.

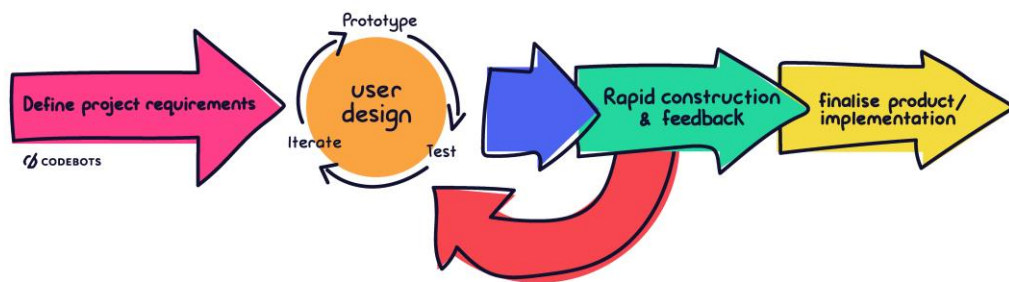


Figure 1. 1 Rapid Application Design (RAD) Diagram.

RAD have 4 main phase, here the phase :

1. Requirement Phase.

In comparison to other project management approaches, the planning phase is summarized; it is a crucial stage for the project's final success. In this phase, communication between the developer, client, and team members establishes the project's goals, development budget, timeframe, and expectations, as well as any existing or anticipated problems that must be resolved. and many other elements.

2. User Design.

Users can test every product prototype at every level to ensure that it meets their expectations, just like custom software development. Prototypes are created by developers, tested by clients, and then discussed among all parties to determine what works and what doesn't. With this approach, designers have the flexibility to make model adjustments until they arrive at a satisfying design. To ensure that nothing potentially slips through the cracks, software developers and clients both learn from experience.

3. Construction.

The prototypes and beta systems from phase two are transformed into functional models in step three. Developers can thus create a final working model much more quickly than they could if they used a conventional project management methodology.

4. Finalise.

Phase 4 This entails educating users, converting data, testing, and implementing the new system. All final modifications are made while the coder and customer are continuously checking for faults in the system.

1.6 Final Project Outline

There's 7 chapter in this final report :

1. Chapter I : Introduction

Describes the initial steps that must be taken before moving on to the creation of the prototype and finishing the application. The information and introduction to the author's ultimate endeavor are the goals of this first

chapter. includes a background section, a problem formulation section, final project objectives, a scope and limitation section, a final project methodology section, and a final project outline..

2. Chapter II : Literature Review

This second chapter will describe the theories and concepts the writers used to construct a web-based meal-ordering system. It consists of an algorithm, a greedy algorithm for estimating how long things will take, an algorithm for sorting data using bubbles, a library called SignalR for building chat features, and related jobs.

3. Chapter III : System Analysis

It uses examples and analysis to explain how the author's system is implemented in its components. Consists of the following: Software Hardware Requirements, Use Case Diagrams, Functional Analysis, Development Process Analysis, and System Overview.

4. Chapter IV : System Design

System Design or chapter four explains how the system works, system flow, and system interface design. Consists of User Interface Design sketches and Class diagrams.

5. Chapter V : System Implementation

System Implementation or chapter five explains how the process of implementing and developing a web-based food order is. Consists of User Interface and Application Details.

6. Chapter VI : System Testing

System Testing or chapter six explains how this web-based food order application runs in a test environment. Consists of Test Environment and Test summary.

7. Chapter VII : Conclusion and Future Works

The Conclusion and Future Works explain the summary of the Final Project result and possible future works for development of this project.

CHAPTER II

LITERATURE REVIEW

Collection of ideas and theories that helped the author develop final project. The Algorithm, Greedy Algorithm, Bubble Sort Algorithm, SignalR Library, Real Time Chat, Estimate Time, and Remark will all be explained in this chapter.

2.1 Algorithm

The algorithm is attributed to Abu Ja'far Muhammad Ibn Musa Al-Khuwarizmi, a well-known Arabic author. Westerners who read the word AL-Khuwarizme do so using algorithms. The book "titled Kitab Al Jabar Wal-Muqabala" by Abu Ja'far Muhammad Ibn Musa Al-Khuwarizmi contains a root word from "Algebra" (Algebra). The word algorithm becomes the word algorithm because it is frequently confused with arithmetic.[1]

An algorithm is defined as a sequence or a path used in computational or systematic problem-solving. An algorithm is frequently considered logic when choosing which program to write in programming activities. According to a different point of view, an algorithm is defined as a calculation, an operation, or a process that a computer is supposed to follow. In other words, a set of logical configurations that are used to solve a problem and are by a specific system are referred to as an algorithm. Below is an example of one of the well-known algorithms, namely the narrative algorithm :

Example : Graduation_student Algorithm, for instance.

The date is presented with the student's name and grade, which is a problem. The student will be deemed successful if their score is 60 or better. The student is classified as failing if their score is lower than 60.

The algorithm is as follows: read the name and grades of the student. If the value is more than or equal to 60, the statement succeeds; otherwise, it fails. Include your name and a brief description. [2]

2.2 Greedy Algorithm

A greedy method is a well-known approach to resolving various issues to optimize (minimize or maximize) particular objective functions. The greedy method is a controlled search strategy that chooses the next state to achieve the most significant improvement in the value of some measure, which may or may not be the objective function. Numerous contemporary heuristics or algorithms have recently been introduced in the literature, and multiple types of enhanced greedy algorithms have also been proposed. The foundation of many Meta-heuristics, including genetic algorithms and simulated annealing, is a greedy strategy.

A search tree, like the one in Fig 2.1, can be used to illustrate how the BG method is represented. A partial solution is represented by each node in the search tree, and the addition of a candidate choice to an existing partial solution is represented by a line connecting two nodes.

As a result, the leaf nodes at the end of the tree represent finished solutions.

The level 1 black circle in Fig 2.1 indicates an initial partial solution. The current partial solution at level 2 has four candidate options represented by four nodes. Each node's promise should be assessed to choose the best one. The second node with the largest benefit (the grey circle at level 2) is selected after using an evaluation function. The sub-problem and partial answer are then adjusted appropriately.

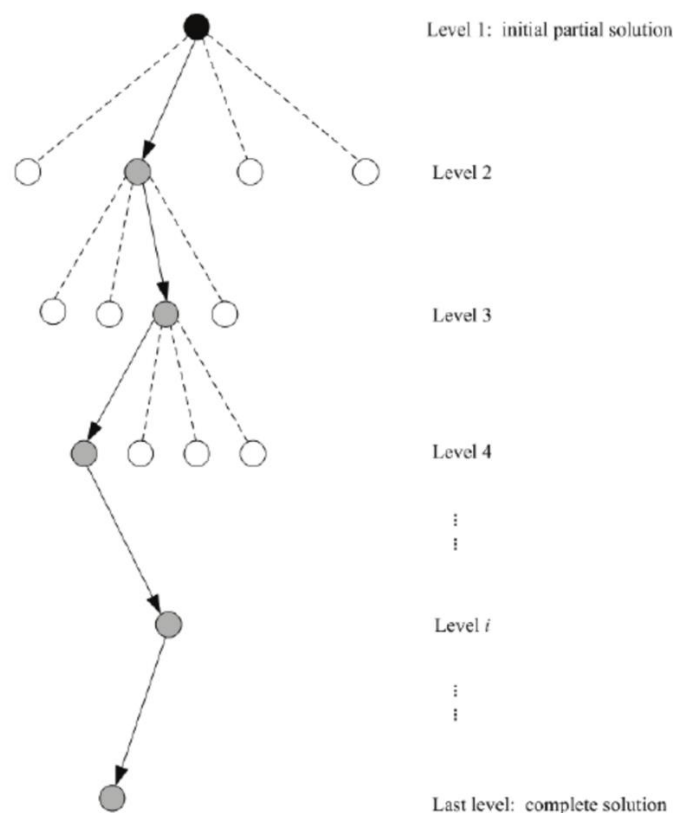


Figure 2. 1 Greedy Algorithm Tree

Efficiency and simplicity of implementation are two key characteristics of the greedy technique that contribute to its popularity. Even though it is straightforward, the BG method is highly effective and occasionally results in the best possible answer to an optimization problem. For issues like the fractional knapsack problem, the minimum spanning tree issue, and the activity selection

problem, BG An algorithm can arrive at the best option by making a sequence of greedy decisions. A trait is shared by all of these problems for which the BG method can produce an optimal solution: the optimal solution to the main problem includes optimal solutions to its subproblems.[3]

2.3 Bubble Sort Algorithm

One of these and the easiest sorting algorithms in terms of understanding and use is the bubble sort algorithm. This algorithm's concept is comparing each element array again and swapping them if the order is incorrect.

Until it no longer needs to be changed, the components will continue to be compared. Algorithms A class of comparison sorting algorithms covers this. Because operations between elements use comparisons, here's an example image of the bubble sort algorithm



Figure 2. 2 Bubble Sort Logic

Because it causes smaller or larger entries to "Bubble" from the top of the elements dataset to find their proper position, the bubble sort method gets its name. Bubble sort's strongest point is how straight forward it is.[4]

2.4 SignalR Library

Microsoft created the library function called SignalR to help developers create real-time Chat. The main applications of SignalR principles include dashboards used mostly in businesses, weather forecasts, and social media applications. It primarily consists of the Javascript and Asp.net components necessary for successful server-to-client communication. Push content operations are mainly employed when the client has to be updated regularly. Between the server and client in SignalR, a persistent connection is created. In contrast to the HTTP request-response model, SignalR keeps the connection open.

Real-time functionality is when a user is updated as soon as new information becomes available on the server. Most of the time, Real-time functionality was only partially incorporated in earlier programs, as evident in several. When data is available in the server buffer, the real-time capability enables the user to receive it immediately. When using a website's real-time functionality, the user gets new updates directly from the server without reloading the page.

The server end and client end has several levels, but in outline they have the same list as below :

1. Hub API
2. Persistent Connection API
3. In Transport Layer:

- Web socket module
- Server sent event module
- Forever frame module
- Long polling module.

[5]

2.5 Estimate Time

Informatics advancements have led to a change in technology utilization from basic to more advanced. information. For the business to later market and sell its products at competitive pricing with its rivals. Because, generally speaking, the company's primary objective is to turn a profit.

PT Hasil Raya Industries, a manufacturer, produce packaging bottles. Due to the large number of order bottles, customers also require a scheduled time to receive them. This and production management go hand in hand. As a result, the author develops a system to estimate when a product will be finished. This can help production management make decisions more quickly.

Processing sales data to see and obtain the appropriate time estimate. No matter when the customer orders will be finished, the industry can anticipate the time required to accomplish them by processing this sales data. in this case, the time estimation is made using the naive bayes algorithm, the formula can be seen below:[6]

$$P(H|X) = \frac{P(X|H).P(H)}{P(X)}$$

Formula 1 Naïve Bayes Formula

Whereas in the case that E-food working on, E-food use a greedy algorithm to calculate the estimated time. The formula can be seen below:

```

X = 0
IF Total Ongoing Order > Station Kitchen Quantity

    Y = Total Ongoing Order - Station Kitchen Quantity
    X = Item Production Estimate Time * Y
    (NB : The number of orders that have not been made multiplied by the estimated time for each item to be made)

Delivery Time = (Item Quantity / Number Of Cook) * Item Production Estimate Time + X

```

Formula 2 Estimate Time Using Linear Equation

In E-food use this calculation to estimate the cost of customer orders for food or beverages. The database extracts the value of the total outstanding order, station kitchen, and product estimation time. Additionally, the input determines the value for the quantity and number of chefs. Then the formula's output will be stored in the database as a DateTime. The DateTime will only be shown as an estimate of an hour in the views of the client area.

2.6 Related Work

2.6.1 Mie Gacoan

An Indonesian restaurant called Mie Gacoan serves meals. Mie Gacoan business was established in early 2016 in Malang and is a PT Pesta Pora Abadi subsidiary. In 2021, or five years later, Mie Gacoan will have 54 outlets in Indonesia. These shops are spread across Indonesia, and most branches are in East Java and Central Java. Currently, Mie Gacoan is creating a custom app for ordering food.

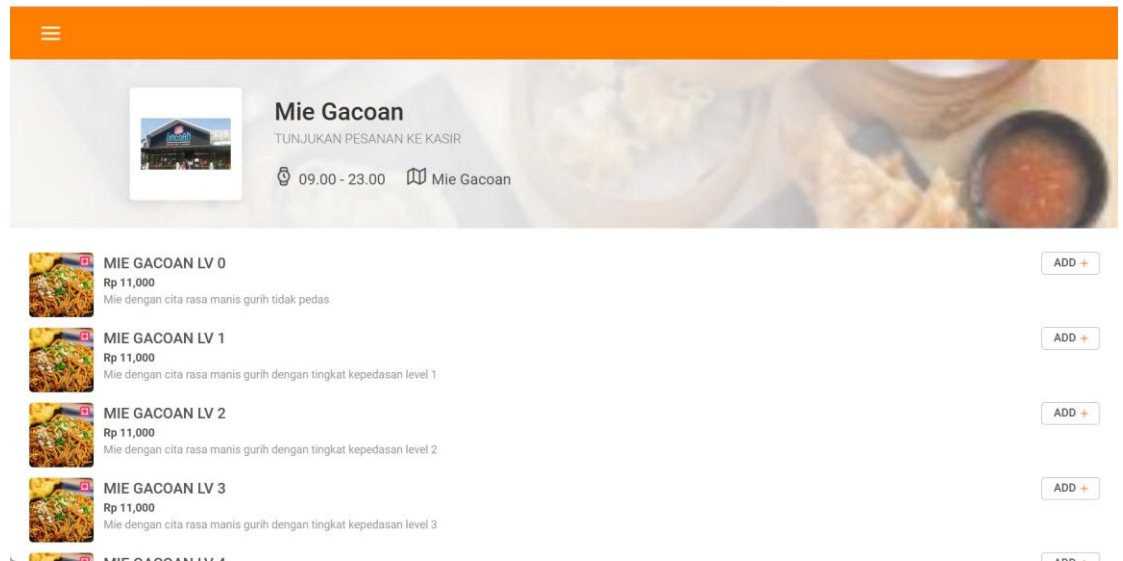


Figure 2. 3 Mie Gacoan Interface

2.7 Comparison Overview with Related Work

Table 2. 1 Comparison between E-Food with Mie Gacoan System

Feature	E-Food	Mie Gacoan System
Login and Register	YES	NO
Order Food and Beverage	YES	YES
Add to cart	YES	YES
Bubble sort	YES	NO
Payment Online	YES	NO
History Order	YES	NO
Estimate time	YES	NO
Chat	YES	NO

CHAPTER III

SYSTEM ANALYSIS

This chapter contains a description of the program features and an analysis of their behavior to meet the essential requirements of this project. This system analysis chapter describes how the application works based on the goals of the program. This chapter consists of a system overview, functional analysis, use case diagrams, use case descriptions, comparative work overviews, Swimlane diagrams, and software and hardware requirements.

3.1 System Overview

This project aims to create a food ordering website using a sorting algorithm for food recommendations according to the criteria. and use the Greedy Algorithm as well for estimating the finished time of the food. Made this website need **visual studio** as tools and **asp.net mvc** as a framework and for the database used **sql server management studio**.

3.2 Function Analysis

Table 3. 1 Table of Function Description

No	Name	Function Description
1	Register	Allow User to register their account
2	Login Pages	Allow User, Customer Service, And Owner sign in to the website
3	Forgot Password	Allow User to reset their password

4	Menu Page	Allow User to see the menu in restaurant
5	Category List	Allow User to see the list of category from each menu
6	Menu List	Allow the User to see and sort menu in the restaurant based on the price and estimated time of each menu also add to cart
7	Cart Page	Allow User to edit and delete some menu in Cart and see items that have been added to the cart
8	Account Page	Allow User to see information about the account
9	History Order Pages	Allow Customer Service and Owner to see what orders have been delivered
10	Current Order Pages	Allow Customer service and owner to see and updated the status for orders have recently entered and Allow Users to see what they order with Greedy Algorithm
11	Payment	Allow User to pay the total amount of ordered item
12	Payment Page	Allow Owner to change the payment method
13	Categories Page	Allow Owner to add and edit the categories of dishes
14	Menulist Page	Allow Owner to add and edit new menu of dishes
15	Reports Page	Allow owner to see income per day and per month
16	Chat Page	Allow Users and Customer Service to chat directly for asking the order
17	Analytic Page	Allow the Owner to see a chart of the number of orders per month eachyear

18	Logout	Allow User, Customer Service, and Owner to sign out from the website
----	--------	--

3.3 Use Case Diagram

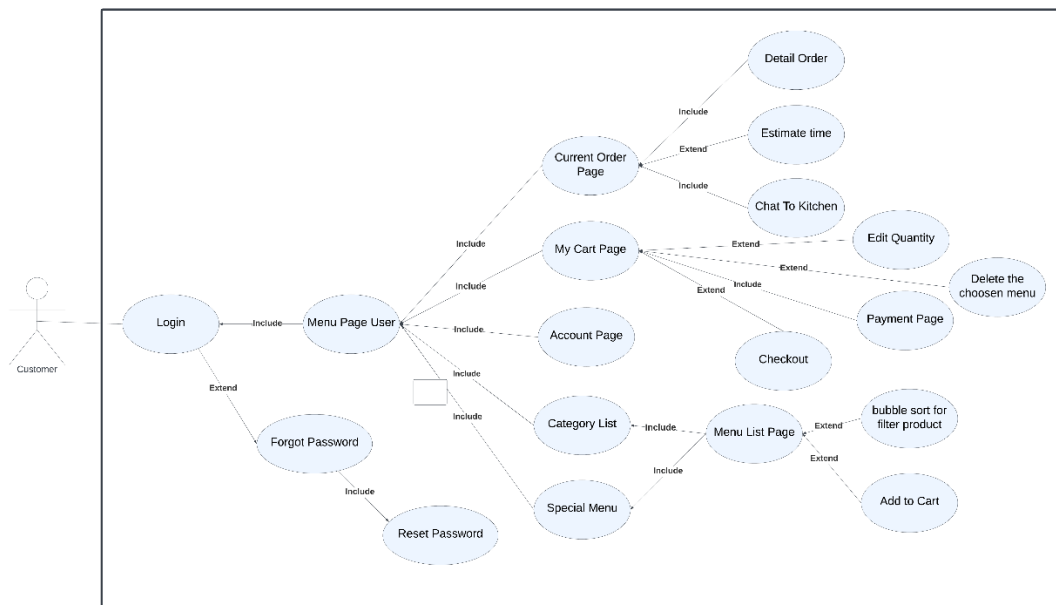


Figure 3. 1 Customer Use Case Diagram

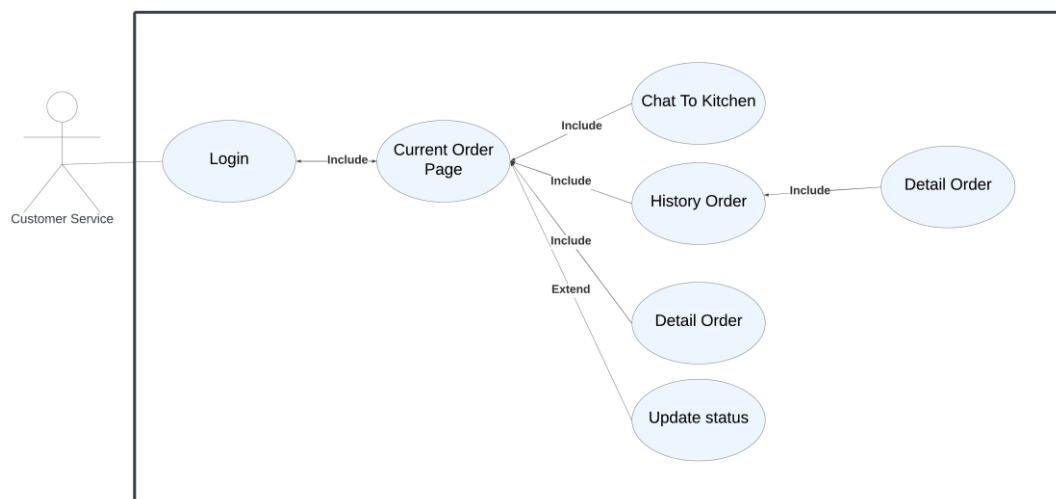


Figure 3. 2 Customer Service Use Case Diagram

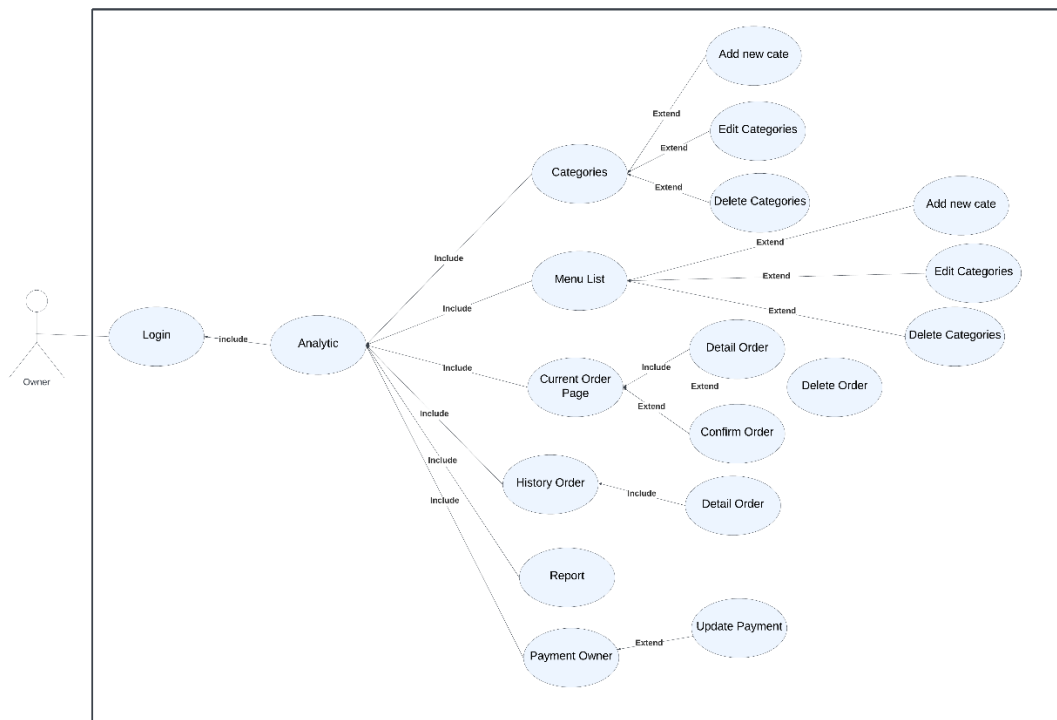


Figure 3. 3 Owner Use case Diagram

3.4 Use Case Narrative

Table 3. 2 Use Case Narrative for “Access Register Page”

Use Case Name	Access Register Page	
Use Case ID	UCID-01	
Priority	High	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	All User/Customer Must Have Account To Login	
Trigger	The User Click On Register Button	
Typical Course of Event	Action User	System Response

	Step 1: User will click create an account button	
		Step 2: The modal of register is displayed
	Step 3: Input Full name, Username, and password	
		Step 4: Save the data to the database system
Post Condition	Register Success	

Table 3. 3 Use Case Narrative for “Access Login Page”

Use Case Name	Access Login Page	
Use Case ID	UCID-02	
Priority	High	
Primary Business Actor	User, Customer Service, And Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	All User Must Have Account	
Trigger	The User Click On Login Button	
Typical Course of Event	Action User	System Response
	Step 1: Fill the username and	

	password to login	
		Step 2: System Approve if password is correct redirect to menu page, and return to login page if username or password false.
	Step 3: Re-Fill the textbox of Username and password to login again	
		Step 4: Re-approved the password and username if true and redirect to Menu page
Post Condition	The user has been redirect to the Menu page	

Table 3. 4 Use Case Narrative for “Access Forgot Password Page”

Use Case Name	Access Forgot Password Page
Use Case ID	UCID-03
Priority	Medium
Primary Business Actor	User
Primary System Actor	System

Other Participating Actor	None	
Precondition	The user is on the login page and have email	
Trigger	The User Click On Forgot password link and fill the email for reset password	
Typical Course of Event	Action User	System Response
	Step 1: The user access the login page	
		Step 2: The system will display login page
	Step 3: The user click forgot password	
		Step 4: The system will display forgot password page
	Step 5: The user fill their email in the textbox	
		Step 6: The system will send the email for reset password
	Step 7: User fill the new password and save it	
	Step 8: The password will updated in database	

Post Condition	The user has been redirect to the Login page
-----------------------	--

Table 3. 5 Use Case Narrative for “Access Menu Page”

Use Case Name	Access Menu Page	
Use Case ID	UCID-04	
Priority	Low	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer must login first to access page	
Trigger	After attempt login success	
Typical Course of Event	Action User	System Response
		Step 1: Show the special menu and category list
Post Condition	The user accesses the Menu page	

Table 3. 6 Use Case Narrative for “Access Menu List Page”

Use Case Name	Access Menu List Page
Use Case ID	UCID-05
Priority	Moderate
Primary Business Actor	User/Customer
Primary System Actor	System
Other Participating Actor	None

Precondition	User/Customer must Choose one of the category list in menu page	
Trigger	Click one category in menu page	
Typical Course of Event	Action User	System Response
	Step 1: User click one category in menu page	
		Step 2: System will bring user/customer to the menu list page
	Step 3: User click the filter product and filtered by any option which the filter using bubble sort algorithm	
		Step 4: The menu has been filtered based on the options selected by the user/customer
		Step 5: User click the add to cart button
		Step 6: Modal will displayed, and button yes clicked, item will added to my cart.
Post Condition	Add to cart succesfull and redirect to menu page	

Table 3. 7 Use Case Narrative for “Access Cart Page”

Use Case Name	Access Cart Page	
Use Case ID	UCID-06	
Priority	Moderate	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer has finished the choosing menu in menu list	
Trigger	Click the my cart in header menu	
Typical Course of Event	Action User	System Response
	Step 1: User click my cart menu in header	
		Step 2: The menu which already chosen by user will be displayed here
	Step 3: User/customer edit the quantity	
		Step 4: The quantity will be edited and the value will saved to temporary table in database

	Step 5: User click the button pay	
		Step 6: The modal will be displayed, and the user/customer will have to fill in the some data
Post Condition	data entry is complete, proceed to the user payment page	

Table 3. 8 Use Case Narrative for “Access Account Page”

Use Case Name	Access Account Page	
Use Case ID	UCID-07	
Priority	Low	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer already registered in database	
Trigger	Click the Account page in header menu	
Typical Course of Event	Action User	System Response
		Step 1: displays some personal data from the user
Post Condition	The user/Customer accessed the Account Page	

Table 3. 9 Use Case Narrative for “Access History Order Page”

Use Case Name	Access History Order Page	
Use Case ID	UCID-08	
Priority	Moderate	
Primary Business Actor	Owner and Customer Service	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer has finished the payment of their order	
Trigger	Click the history in header menu	
Typical Course of Event	Action User	System Response
		Step 1: Show only order who has delivered status
	Step 2: Owner or Customer Service click the detail button	
		Step 3: Details of each food or drink order from each user are displayed
Post Condition	Order finished, return to main page	

Table 3. 10 Use Case Narrative for “Access Payment Page”

Use Case Name	Access Payment Page	
Use Case ID	UCID-09	
Priority	Moderate	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer has finished the entry data for checkout	
Trigger	After entry the data for checkout, click the go to payment button	
Typical Course of Event	Action User	System Response
	Step 1: entry the data for checkout and click go to payment	
		Step 2: The qr code and some information about transaction purpose for payment will be displayed
	Step 3: User/Customer Click the upload approval photo and upload it	
		Step 4: The approval photo will be saved in database and will

		displayed in order detail owner
	Step 5: User/Customer click the cancel order	
		Step 6: the data in checkout table will be reset and the stock will reset too
Post Condition	Upload approval photo finished and auto redirect to current order for user	

Table 3. 11 Use Case Narrative for “Access Current Order user Page”

Use Case Name	Access Current Order List User Page	
Use Case ID	UCID-10	
Priority	High	
Primary Business Actor	User/Customer	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer has finished the upload of approval picture	
Trigger	After upload approval photo, it will directly to this page, or user can click from header menu	
Typical Course of Event	Action User	System Response
	Step 1: User upload approval picture	

		Step 2: History of the order will displayed
	Step 3: User/Customer click the detail button	
		Step 4: Will open the orderdetail page, Details of each food or drink order from each user are displayed with estimates time feature using Greedy Algorithm
	Step 5: User/Customer click the chat button	
		Step 6: it will directly move to chat page of user/customer
	Post Condition	the user/customer will click the chat button to ask whether the order has been approved or not. or maybe they ask whether their order has been completed or not

Table 3. 12 Use Case Narrative for “Access Chat Page”

Use Case Name	Access Chat Page
Use Case ID	UCID-11

Priority	High	
Primary Business Actor	User/Customer and Customer Service	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer goes to order details in current order page. And for customer service if there is already an incoming message from the customer	
Trigger	User/Customer service and Customer service click the button chat	
Typical Course of Event	Action User	System Response
	Step 1: User click the button chat	
		Step 2: The chat and history of the chat will displayed
	Step 3: User/Customer or Customer Service type the message and sended	
		Step 4: The new chat will be appear in the display chat real time
Post Condition	User/Customer and Customer Service Accessed the chat page	

Table 3. 13 Use Case Narrative for “Current Order Owner And Customer Service”

Use Case Name	Access Current Order List Owner and Customer Service Page	
Use Case ID	UCID-12	
Priority	High	
Primary Business Actor	Owner and Customer Service	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	User/Customer has finished payment and upload of approval picture	
Trigger	Owner and Customer Service can click Order/Recent Order List in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner or Customer Service Click the option in header menu	
		Step 2: recent order will displayed
	Step 3: Owner or Customer Service click the detail button	
		Step 4: Will open the orderdetail page, Details of

		each food or drink order from each user are displayed.
	Step 5: Owner click the Confirm Order button	
	Step 6: Customer Service click the Order on the way button and Order Delivered button	
		Step 7: it will directly change the status of the order and button confirm per item will displayed for Owner.
	Step 8: Owner click confirm order per item	
		Step 9: Data will be saved in database, and the Greedy Algorithm will be work
	Step 10: Owner click the cancel order	

		Step 11: it will cancelled the order, and all value will reset.
Post Condition	Owner and Customer Service can access the recent order page	

Table 3. 14 Use Case Narrative for “Payment Owner”

Use Case Name	Access Payment Owner Page	
Use Case ID	UCID-13	
Priority	Moderate	
Primary Business Actor	Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	Owner want to edit the method of payment transaction	
Trigger	Owner Click the Payment page in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner click the payment menu in header	
		Step 2: Current qr code and payment method will be displayed

	Step 3: Owner fill the textbox for updating the new payment transcation	
		Step 4: new data will be saved in database, and will be displayed by new data
Post Condition	Owner successful update the payment transaction and has access the payment owner page	

Table 3. 15 Use Case Narrative for “Category list Owner”

Use Case Name	Access Categories Page Owner	
Use Case ID	UCID-14	
Priority	Moderate	
Primary Business Actor	Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	Owner want to edit or add new category	
Trigger	Owner Click the Categories page in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner click the Categories in header	

		Step 2: Current Category list will be displayed here
	Step 3: Owner click add new categories and fill the new value inside the modal	
		Step 4: new data will be saved in database, and the new data will be displayed in the table
	Step 5: Owner click edit and fill the new value	
		Step 6: new edited data will be saved in database and will be displayed inside the table
	Step 7: owner click the delete button	
		Step 8: the category who has been deleted will be disappear from table and the data in

		database will be deleted too
Post Condition	Owner successful update the category list and has access the category list page	

Table 3. 16 Use Case Narrative for “Menulist Owner”

Use Case Name	Access Menulist Page Owner	
Use Case ID	UCID-15	
Priority	Moderate	
Primary Business Actor	Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	Owner want to edit or add new category	
Trigger	Owner Click the Categories page in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner click the Menulist in header	
		Step 2: Current Menu list will be displayed here
	Step 3: Owner click add new Menu and fill the new value inside the modal	

		Step 4: new data will be saved in database, and the new data will be displayed in the table
	Step 5: Owner click edit in price column and fill the new value	
		Step 6: new edited data will be saved in database and will be displayed inside the table
	Step 7: Owner click edit in stock column and fill the new value	
		Step 8: new edited data will be saved in database and will be displayed inside the table
	Step 9: owner click the delete button	
		Step 10: the category who has been deleted will be disappear from

		table and the data in database will be deleted too
Post Condition	Owner successful update the menu list and has access the menu list page	

Table 3. 17 Use Case Narrative for “Report Page”

Use Case Name	Access Report Page Owner	
Use Case ID	UCID-16	
Priority	Moderate	
Primary Business Actor	Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	Owner want to see how many income	
Trigger	Owner Click the Report page in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner click the Report in header	
		Step 2: Total income per day and per month will displayed
	Step 3: Owner search the income using date range	

		Step 4: the data will be searched by the date
Post Condition	Owner has access in report page	

Table 3. 18 Use Case Narrative for “Analytic Page”

Use Case Name	Access Analytic Page Owner	
Use Case ID	UCID-17	
Priority	Moderate	
Primary Business Actor	Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	Owner want to see total order by chart	
Trigger	Owner Click the Analytic page in header menu	
Typical Course of Event	Action User	System Response
	Step 1: Owner click the Analytic in header	
		Step 2: Total order per month will displayed by chart
Post Condition	Owner has access in Analytic page	

Table 3. 19 Use Case Narrative for “Logout”

Use Case Name	Access Logout
----------------------	---------------

Use Case ID	UCID-18	
Priority	High	
Primary Business Actor	User, Customer Service, And Owner	
Primary System Actor	System	
Other Participating Actor	None	
Precondition	All user must finished all activity and want to logout	
Trigger	All user Click the Logout in header menu	
Typical Course of Event	Action User	System Response
	Step 1: user click the Logout in header	
		Step 2: user will directly send to login page
Post Condition	Owner has access in Analytic page	

3.5 Swim Lane Diagram

3.5.1 Swim Lane Diagram for Login And Register Page

3.5.1.1 Login Page

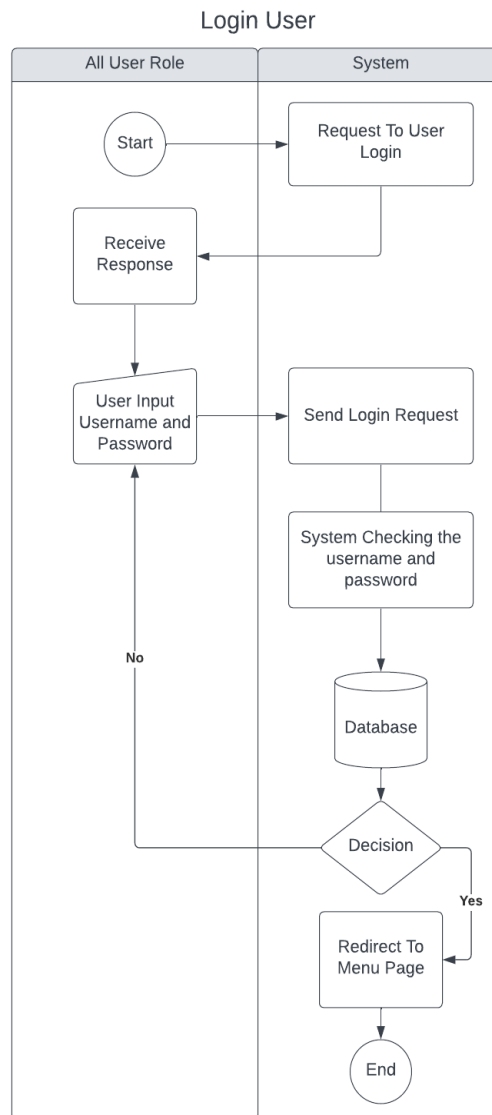
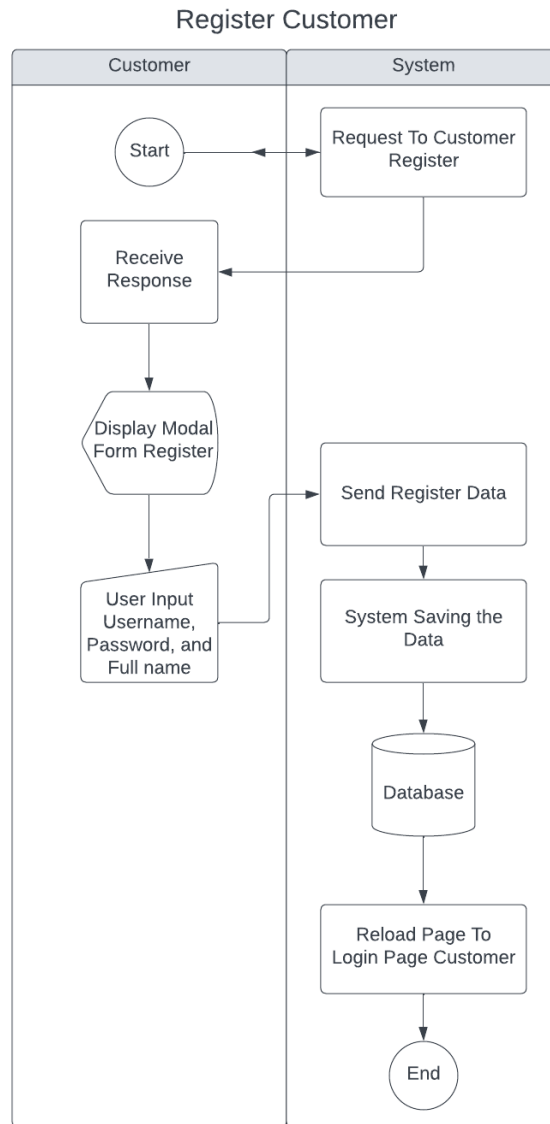


Figure 3. 4 Swim Lane Login

3.5.1.2 Register Page

**Figure 3. 5 Swim Lane Register**

3.5.2 Swim Lane Diagram for Forgot Password

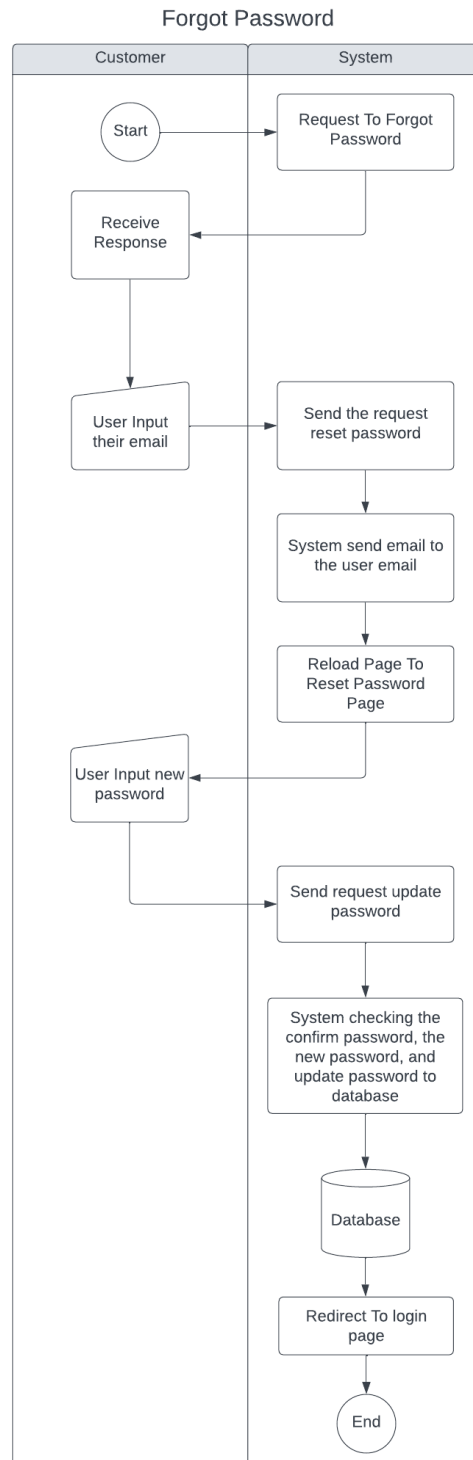


Figure 3. 6 Swim lane Forgot Password

3.5.3 Swim Lane Diagram for Menu Page

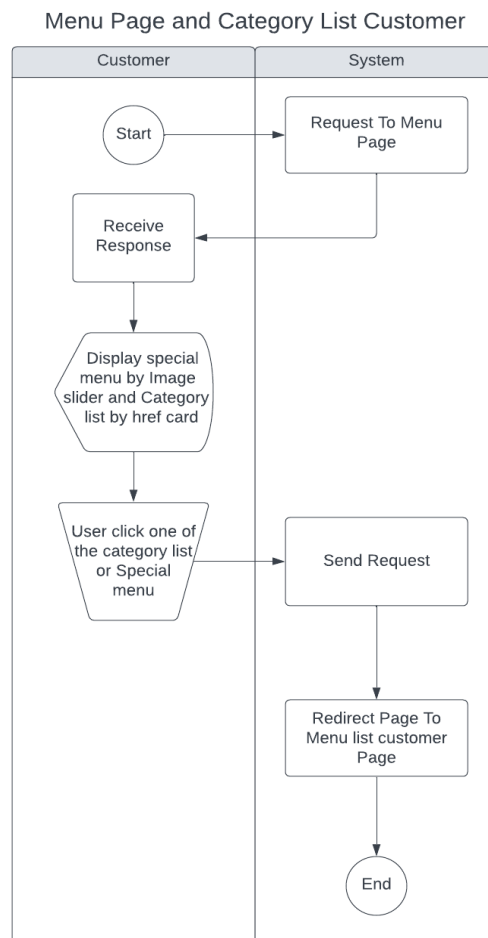


Figure 3. 7 Swim lane menu page

3.5.4 Swim Lane Diagram for Menu List Page

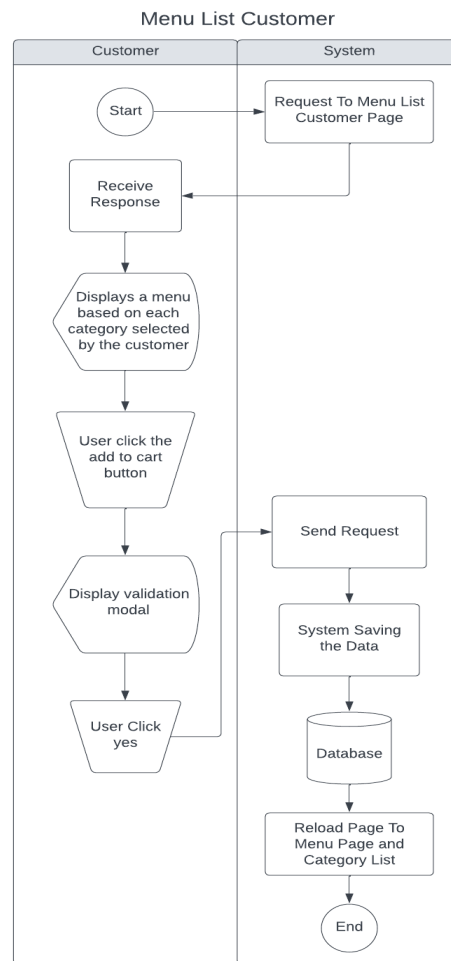


Figure 3. 8 Swim Lane menu list page

3.5.5 Swim Lane Diagram for Cart Page

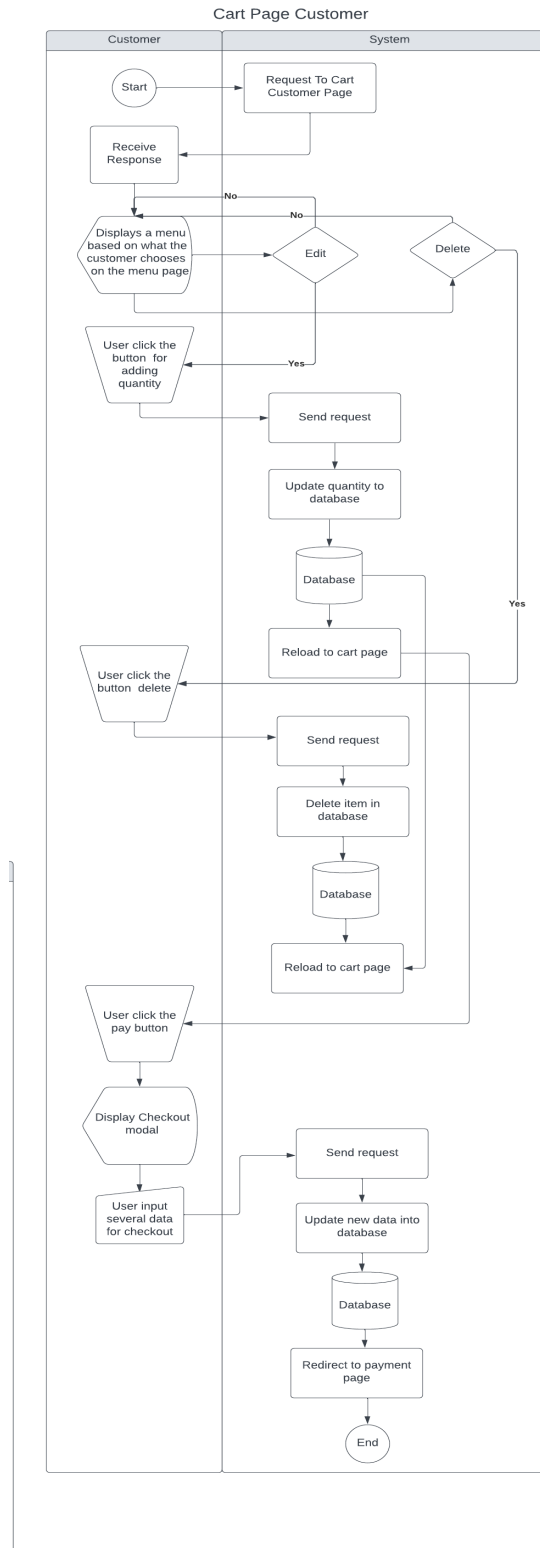


Figure 3. 9 Swim lane Cart page

3.5.6 Swim Lane Diagram for Account Page

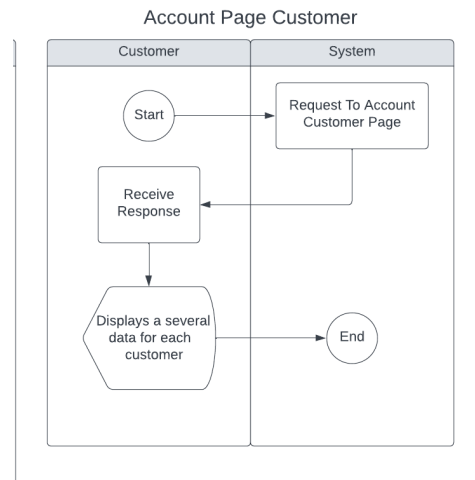


Figure 3. 10 Swim lane Account page

3.5.7 Swim Lane Diagram for History Order Page

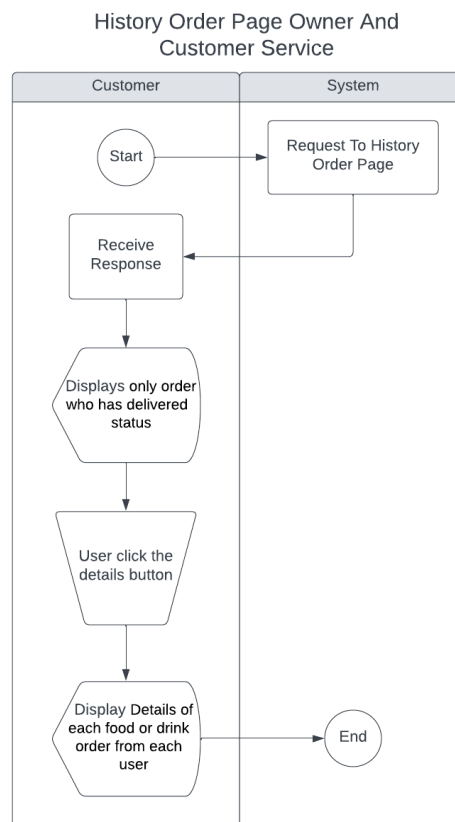


Figure 3. 11 Swim lane History Order

3.5.8 Swim Lane Diagram for Current Order Page

3.5.8.1 Current Order Customer

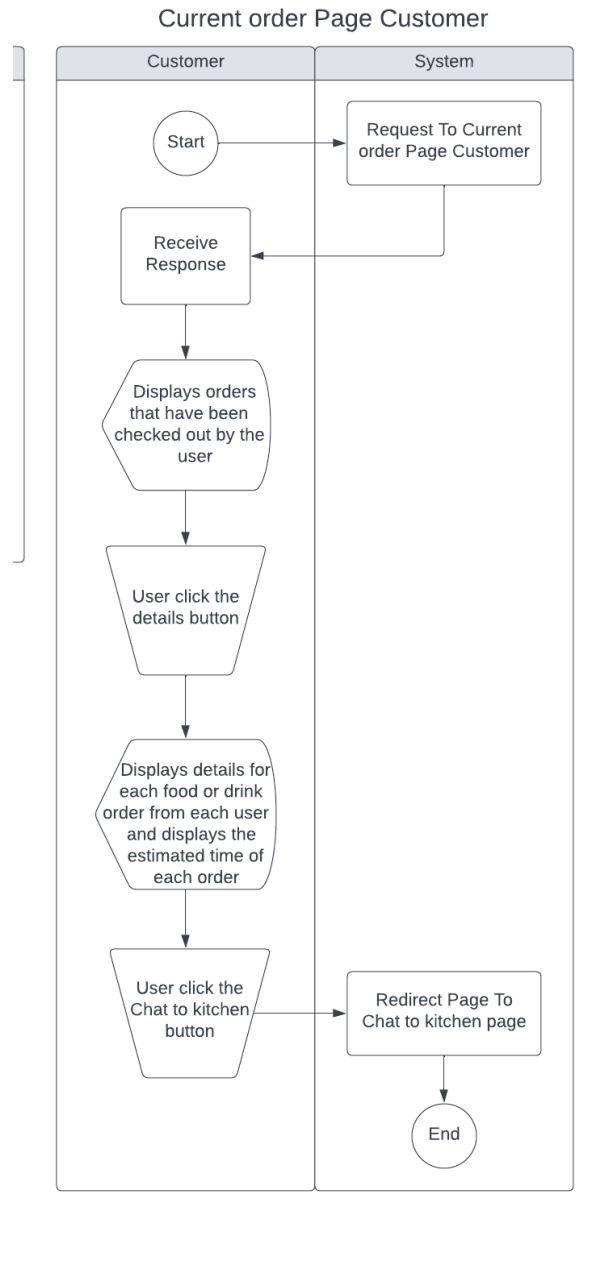


Figure 3. 12 Current Order customer

3.5.8.2 Current Order Owner and Customer Service

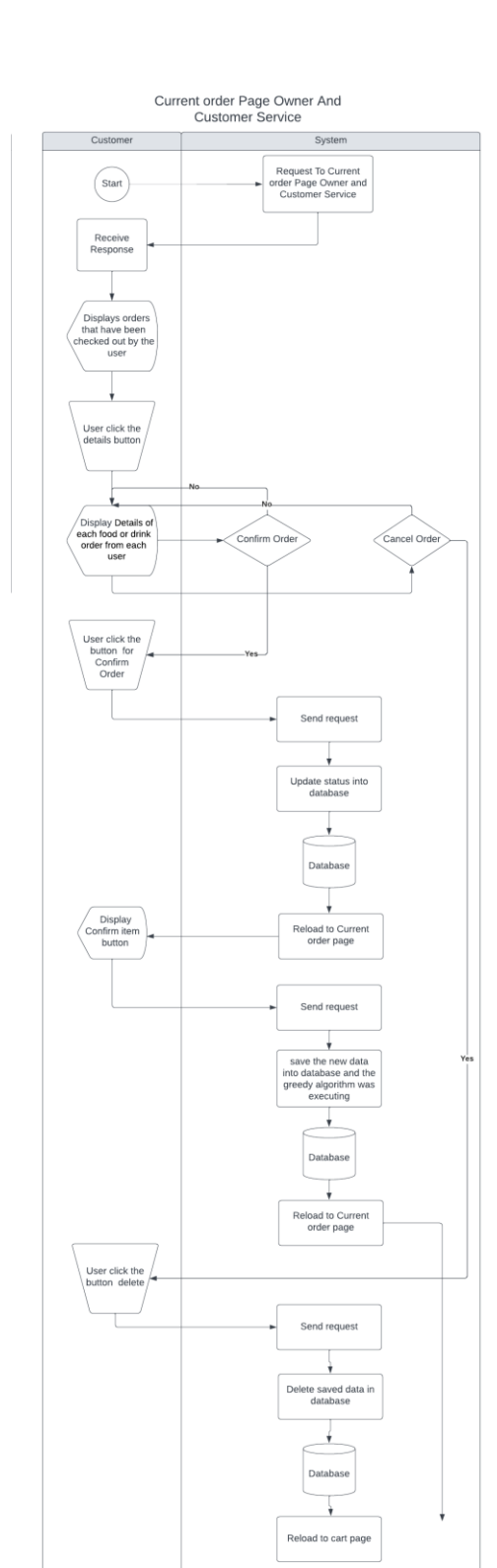


Figure 3. 13 Swim Lane Current Order Owner and Customer Service

3.5.9 Swim Lane Diagram for Payment Page

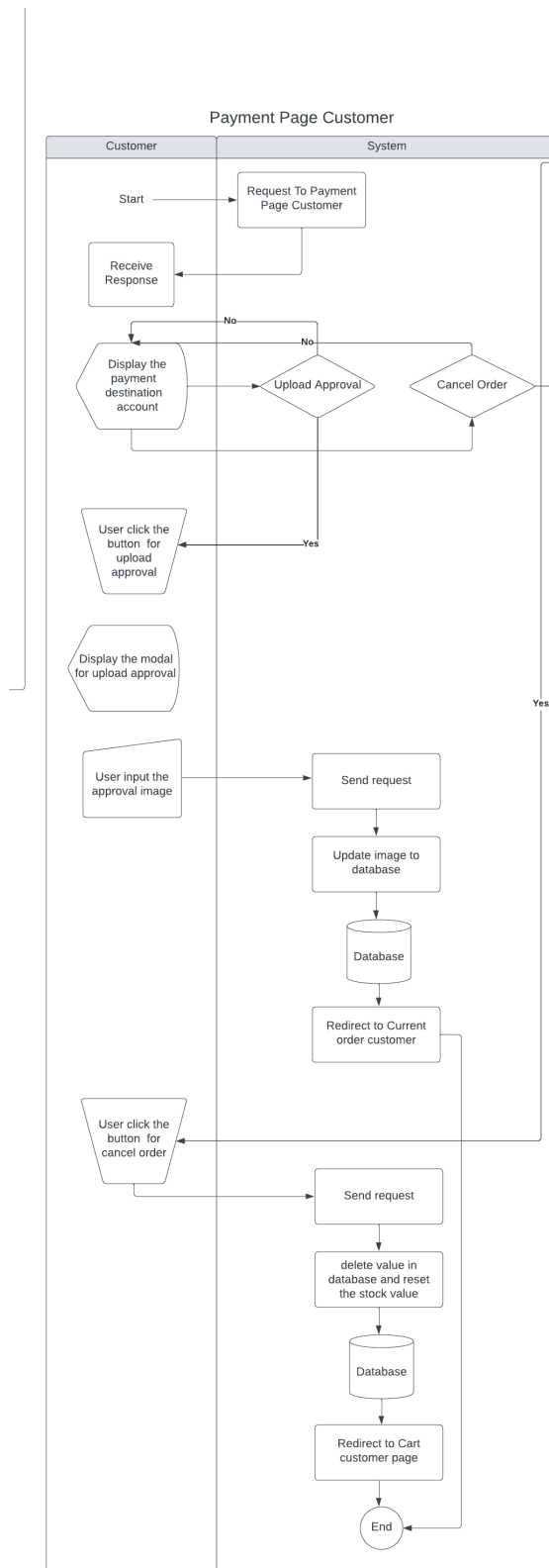


Figure 3. 14 Swim Lane Payment Page

3.5.10 Swim Lane Diagram for Payment Owner Page

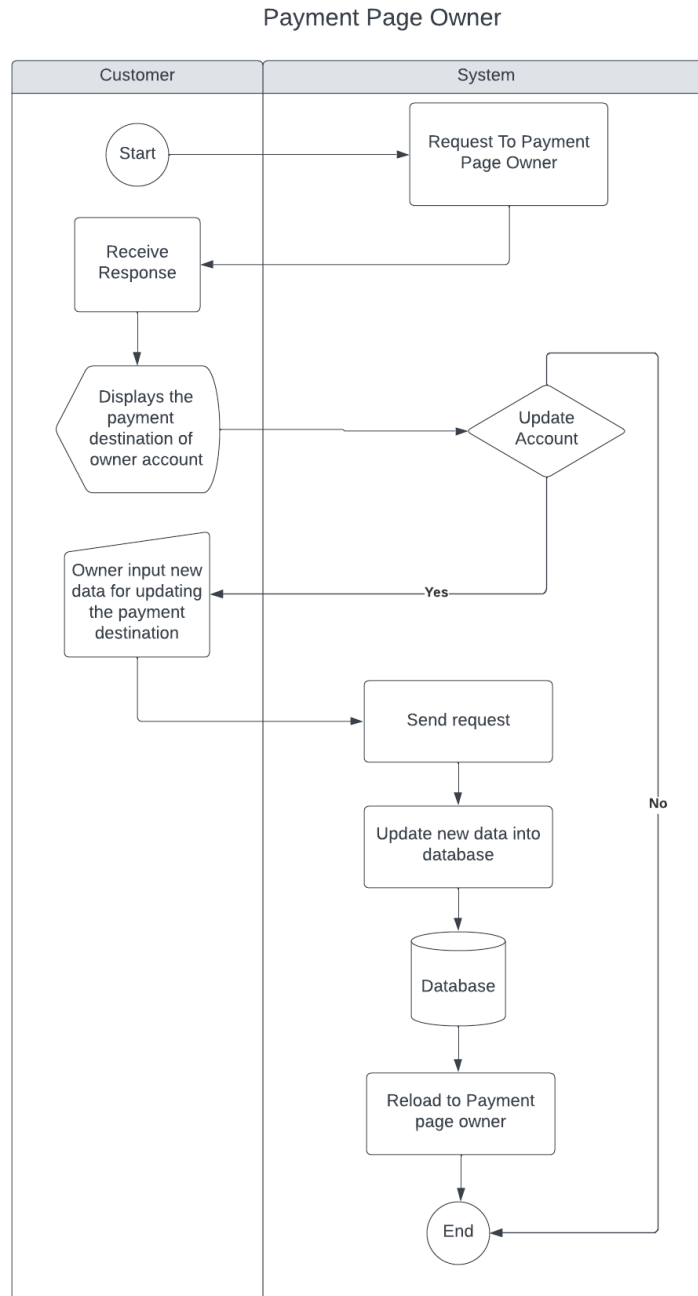


Figure 3. 15 Payment Owner Page

3.5.11 Swim Lane Diagram for Categories Page

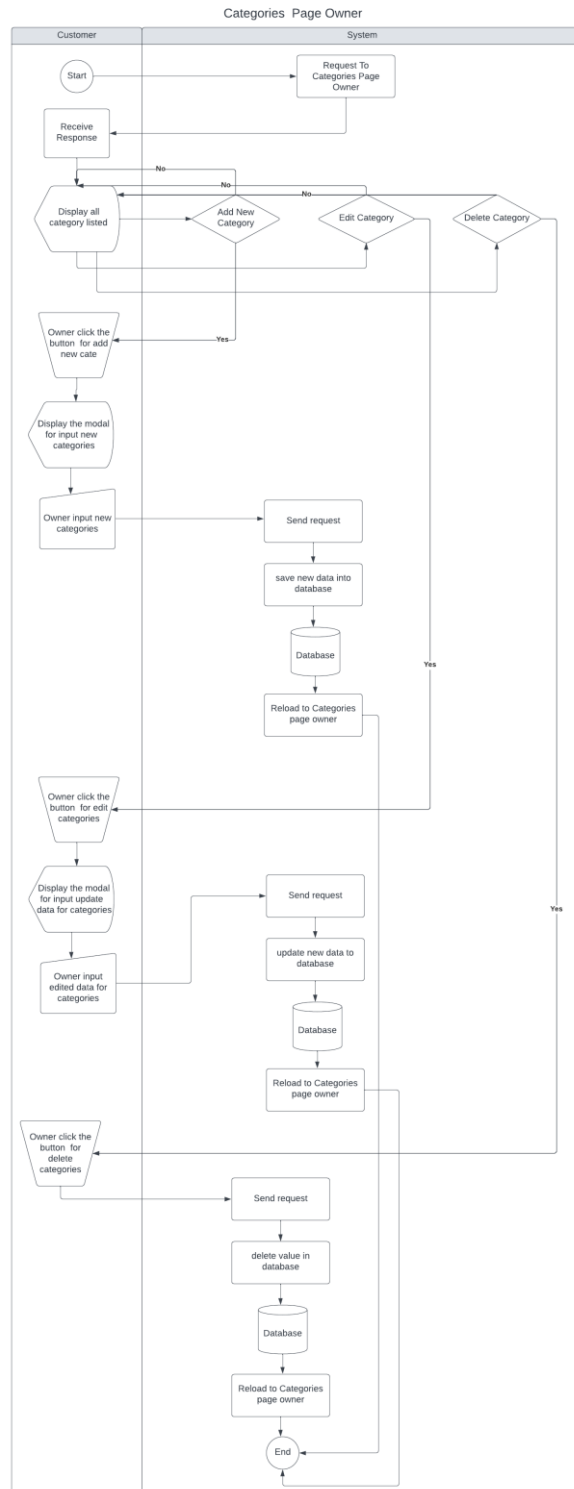


Figure 3. 16 Swim lane categories page

3.5.12 Swim Lane Diagram for Menulist Owner Page

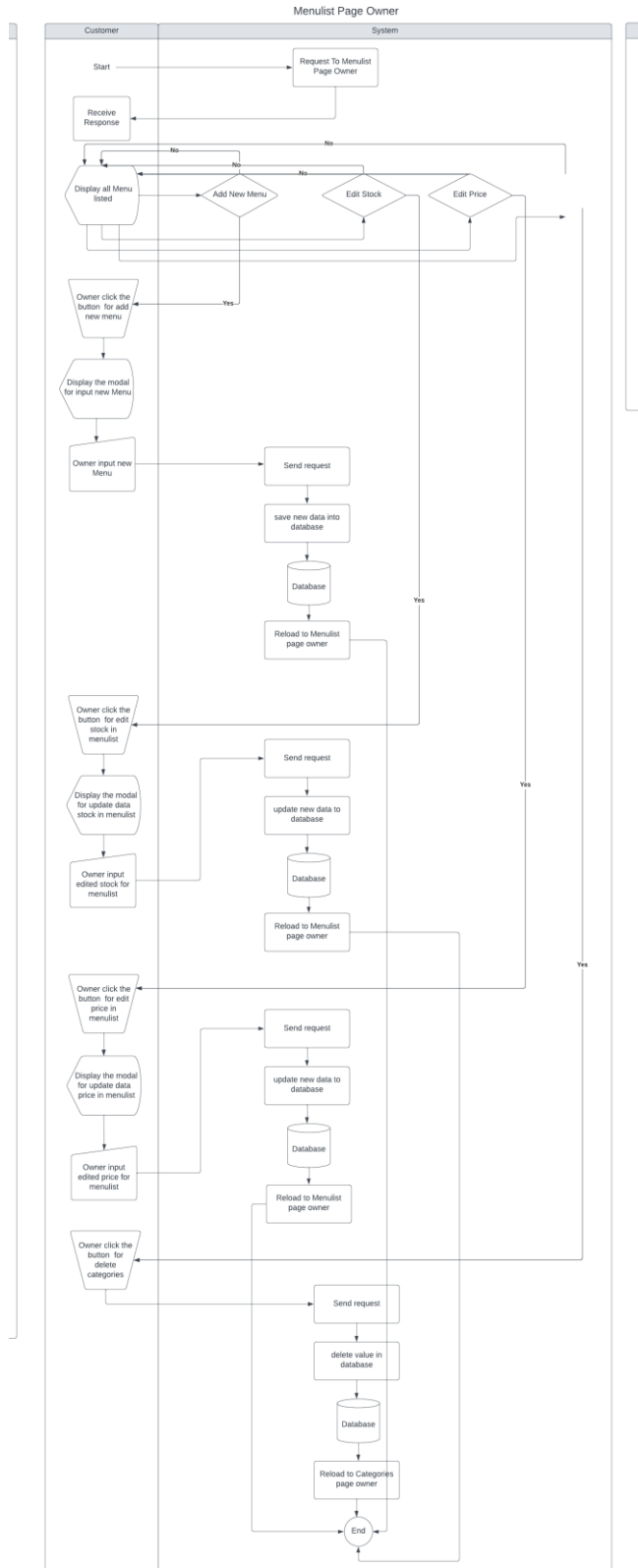
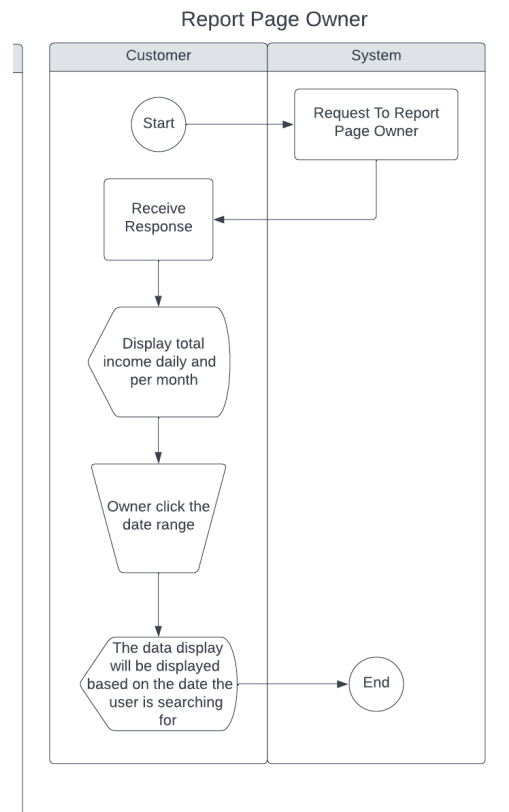


Figure 3. 17 Swim Lane menulist owner**3.5.13 Swim Lane Diagram for Report Page****Figure 3. 18 Swim lane report page**

3.5.14 Swim Lane Diagram for Chat Page

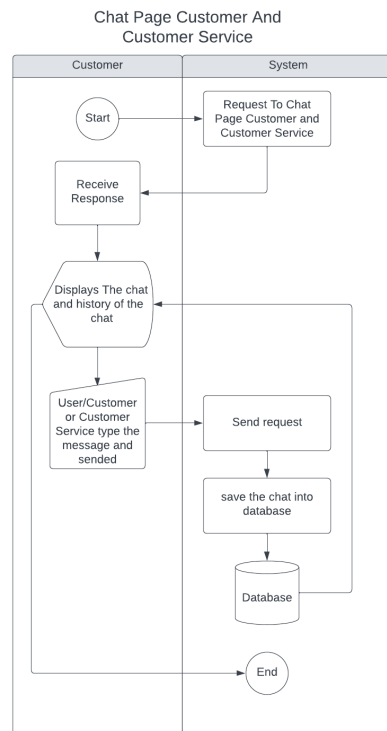


Figure 3. 19 Swim lane Chat page

3.5.15 Swim Lane Diagram for Analytic Page

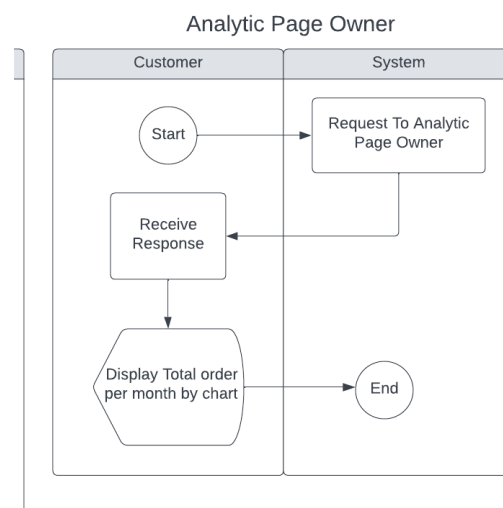


Figure 3. 20 Swim lane Analytic page

3.6 Hardware and Software Requirement

The Hardware and Software needed to develop E-Food website below :

3.6.1 Hardware Requirement

The Hardware requirements:

1. Laptop or PC for Develop the system
2. Laptop or PC for Testing the system

3.6.2 Software Requirement

The required software for this web application development is shown on Table below:

Table 3. 20 Hardware Requirement

Software	Function
Windows 10	As the main laptop's operating system
Visual Studio 2019	As the tools for developing the website
Asp.net web application (.NET Framework)	As the framework
Search engine	As to access the system.
C#	As the programming language
SQL Server Management	As the database

CHAPTER IV

SYSTEM DESIGN

System design is the process defining modules, architecture, components, and interfaces and its data of a system to fulfill the specified requirements. as well as data, based on given requirements. It is the process of identifying, creating, and designing the app that meets the final project's objectives and expectations.

4.1 User Interface Design

User interface design is the process of designing or developing to create interfaces for software or computerized systems with a focus on appearance or style. To make visitors turn to buyers, they need to feel well facilitated between themselves and the website or app by having a good user interface design.

4.1.1 Login & Register

The first thing you see when you access the website is a selection of buttons for users to log in according to their roles. The role will be brought to the following page, which is the login page specific to the chosen account role when one of these buttons is clicked.

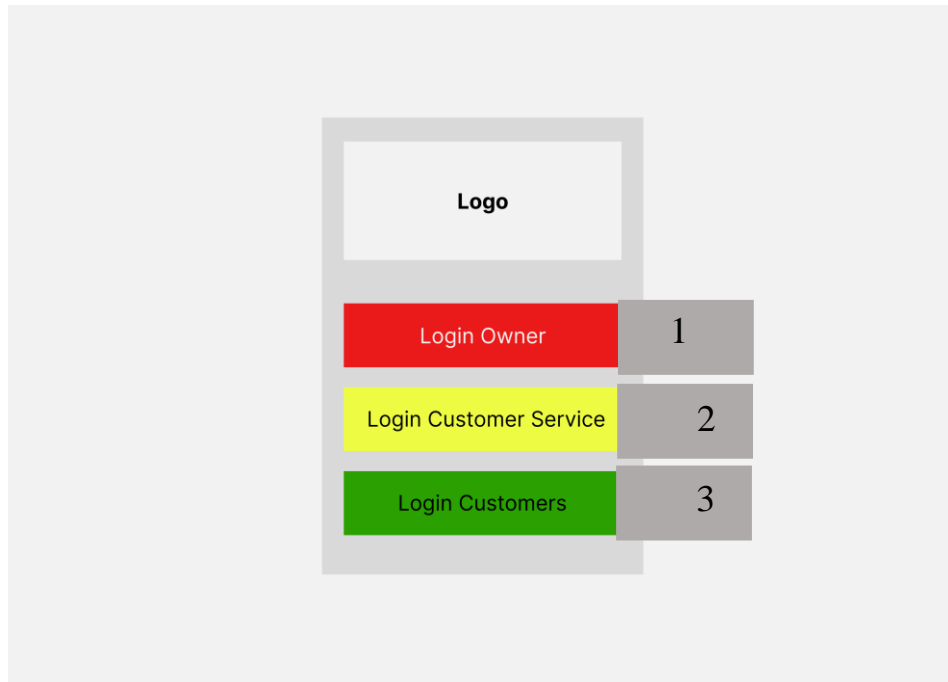


Figure 4. 1 Implementation first page.

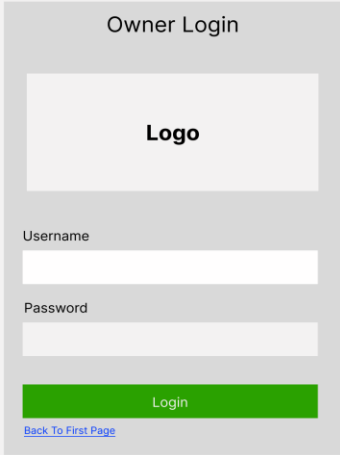
Table 4. 1 Label Description from Figure 4.1

No.	Label Description
1	Login Owner
2	Login Customer Service
3	Login Customers

4.1.1.1 Login Owner & Login Customer Service

The owner and customer service logins are only available to restaurant employees. Because of this, regular customers cannot access their accounts, and the developer does not offer registration options. The username and password are required to log in here, as demonstrated below.

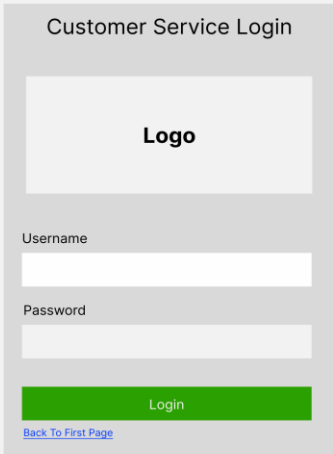
1



The image shows a web form titled "Owner Login". At the top, there is a "Logo" placeholder. Below the logo are two input fields: "Username" and "Password". A green "Login" button is positioned below the password field. At the bottom of the form, there is a blue link labeled "Back To First Page".

Figure 4. 2 Implementation Login Owner.

2



The image shows a web form titled "Customer Service Login". At the top, there is a "Logo" placeholder. Below the logo are two input fields: "Username" and "Password". A green "Login" button is positioned below the password field. At the bottom of the form, there is a blue link labeled "Back To First Page".

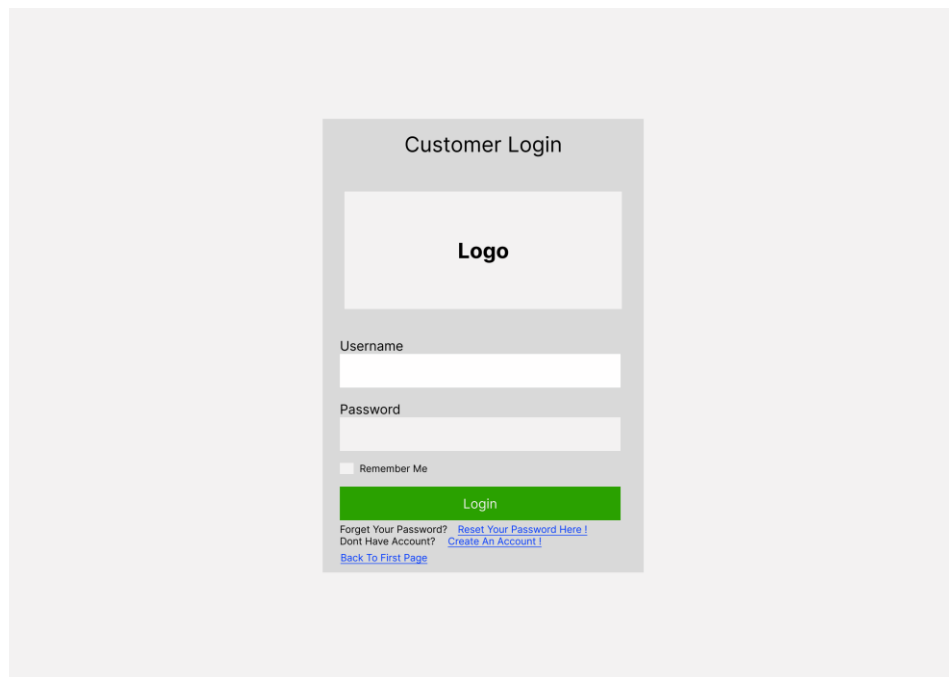
Figure 4. 3 Implementation Login Customer Service.

Table 4. 2 Label Description from Login & Register

No	Label Description
1	Login Owner Figure
2	Login Customer Service Figure

4.1.1.2 Login Customers

Only the username and password are required for the customer log-in, much as the Owner and Customer Service logins. The registration feature in the customer log-in, which is necessary for new users and customers from restaurants, is the only distinction between the customer service, owner, and customer logins.



Customer Login

Logo

Username

Password

Remember Me

Login

Forgot Your Password? [Reset Your Password Here !](#)
Don't Have Account? [Create An Account!](#)
[Back To First Page](#)

Figure 4. 4 Implementation Login Customer.

4.1.1.3 Register Customers

They only need to enter their full name, username, and password in the form below to register as a new customer. Dates and roles are automatically filled in based on the day of registration.

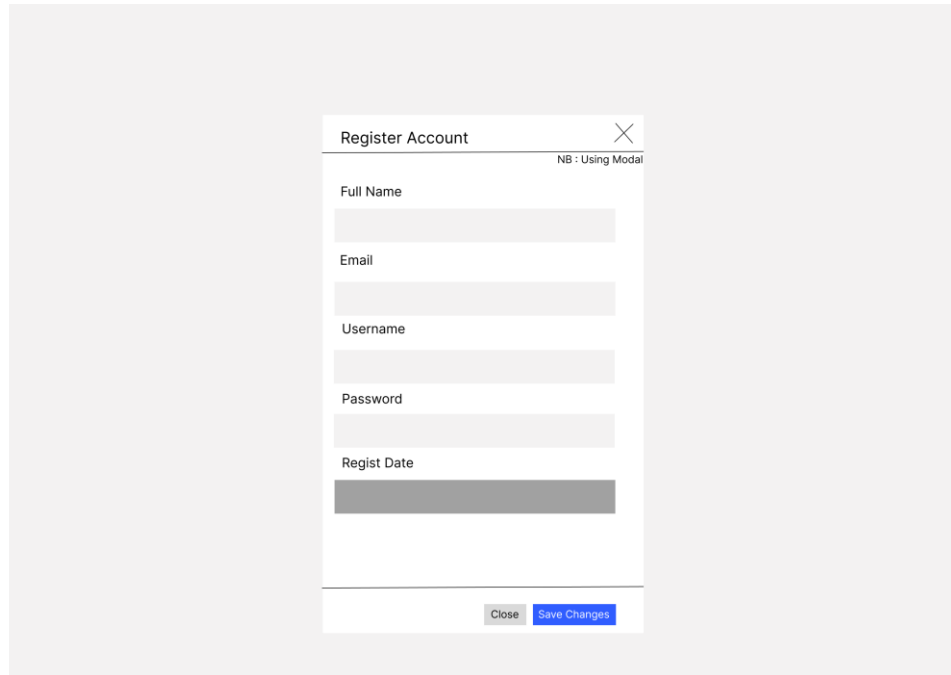
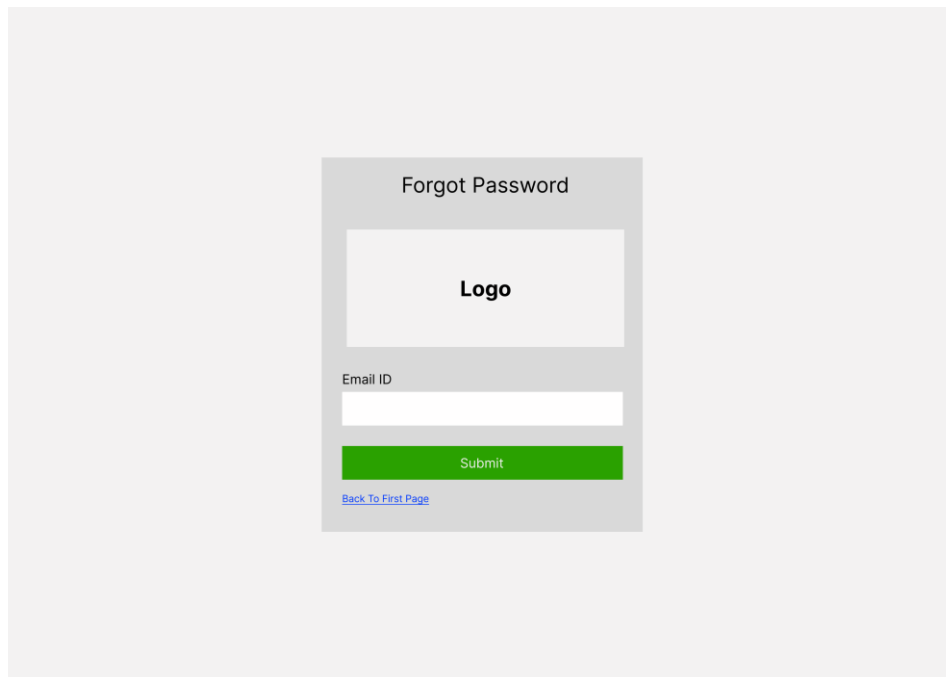
A screenshot of a web application's 'Register Account' modal. The modal has a title bar with 'Register Account' and a close button (X). Below the title bar, there is a small text 'NB : Using Modal'. The form contains five input fields: 'Full Name', 'Email', 'Username', 'Password', and 'Register Date'. The 'Register Date' field is filled with a greyed-out date. At the bottom of the modal, there are two buttons: 'Close' and 'Save Changes'.

Figure 4. 5 Implementation Register Customer.

4.1.2 Forgot Password

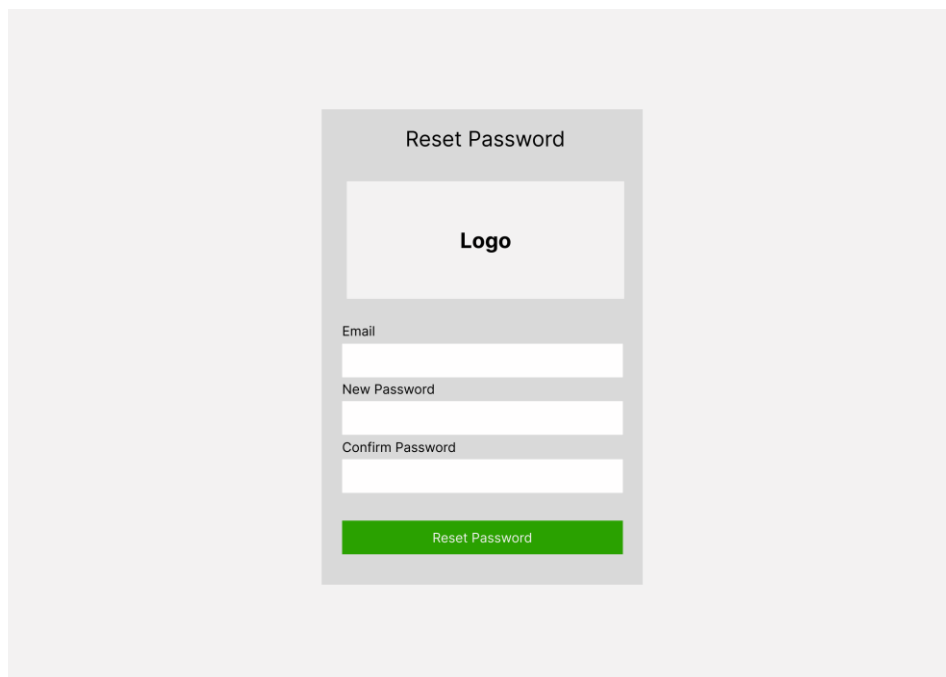
Here for user reset password. User can input their email that have been registered in this system before. After user input this textbox, the email will sended into the email they've been input before.



The image shows a 'Forgot Password' form. At the top, the title 'Forgot Password' is centered. Below the title is a placeholder for a 'Logo'. Underneath the logo is a text input field labeled 'Email ID'. Below the input field is a green button labeled 'Submit'. At the bottom of the form, there is a blue link labeled 'Back To First Page'.

Figure 4. 6 Implementation Forgot Password.

After the email being sended, user need to press the link inside that email, because it will directly move in here.



The image shows a 'Reset Password' form. At the top, the title 'Reset Password' is centered. Below the title is a placeholder for a 'Logo'. Underneath the logo are three text input fields: 'Email', 'New Password', and 'Confirm Password'. Below the input fields is a green button labeled 'Reset Password'.

Figure 4. 7 Implementation Reset Password.

4.1.3 Menu Page & Category List

The next page, the Customer page menu, is reached when a customer logs in using his account. The consumer has a choice on this page between picking the category from the special menu displayed, as indicated by the number 1 on the special menu, or choosing individual food items from the category list, as indicated by the number 2 on the category list.

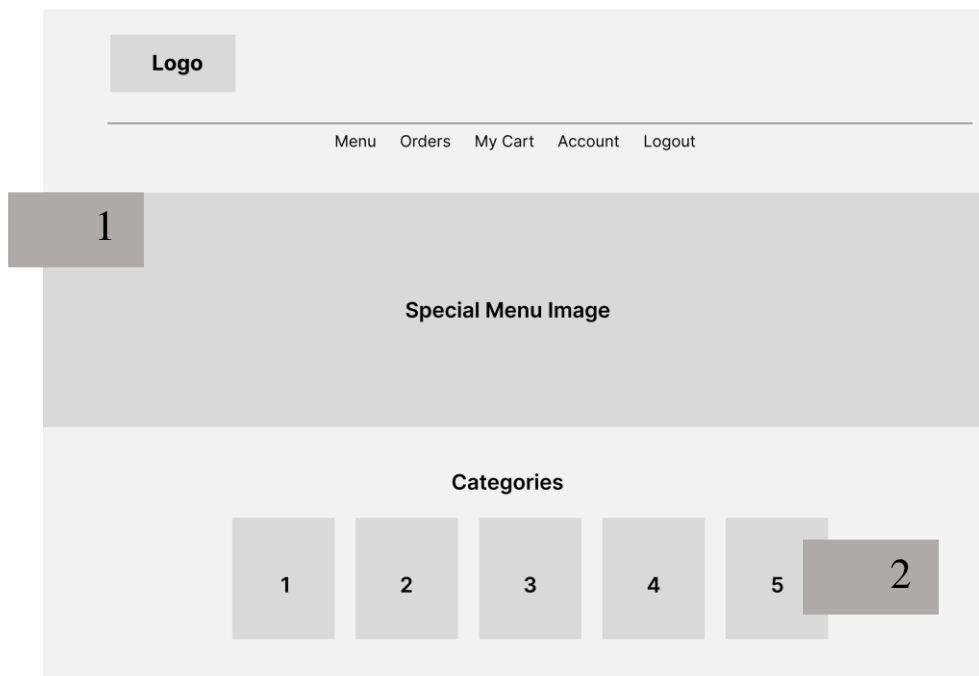


Figure 4. 8 Implementation Menu Page.

Table 4. 3 Label Description from Figure 4.6

No	Label Description
1	Special Menu
2	Categories List

4.1.4 Menu List

Customers are taken to the Menu list page after choosing a category or a specific menu category. On this page, the menu is organized according to their chosen category. Additionally, there is an add-to-cart button in the action column on this page, also bubble sort feature which helps put the customer's preferred food in the cart.

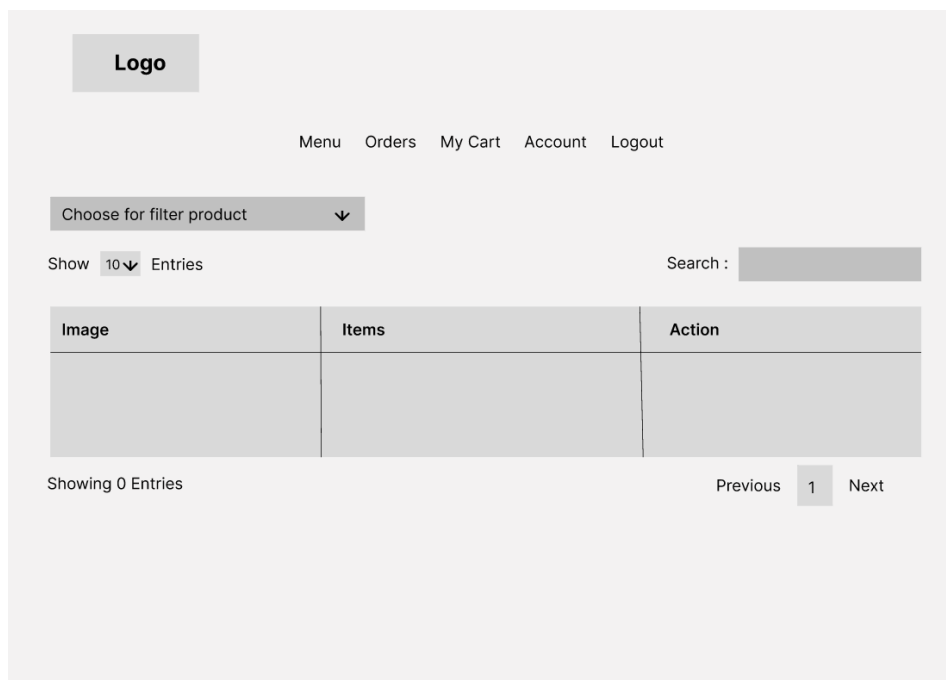


Figure 4. 9 Implementation Menu List Page.

4.1.5 Cart Page

The customer will check the items in their basket through the cart page once they have completed choosing their meals and beverages. They can change the quantities of each meal and drink on this page, as well as remove any previously chosen items. Additionally, there is a "Pay" button that leads them to the payment page.

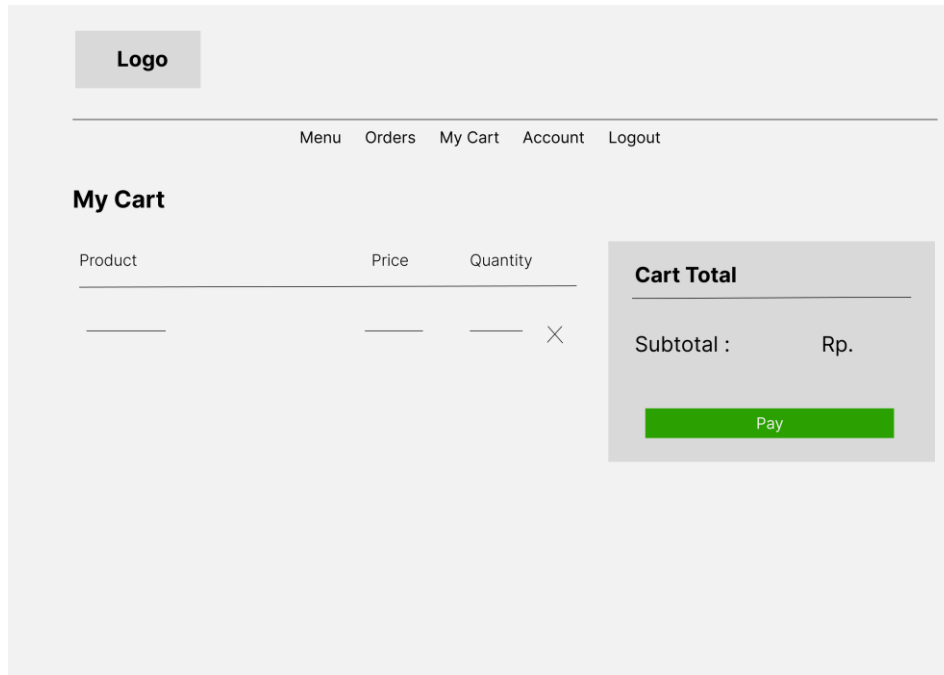


Figure 4. 10 Implementation Cart Page.

4.1.6 Account Page

On this page there is information from each customer about their full name, role, and username.

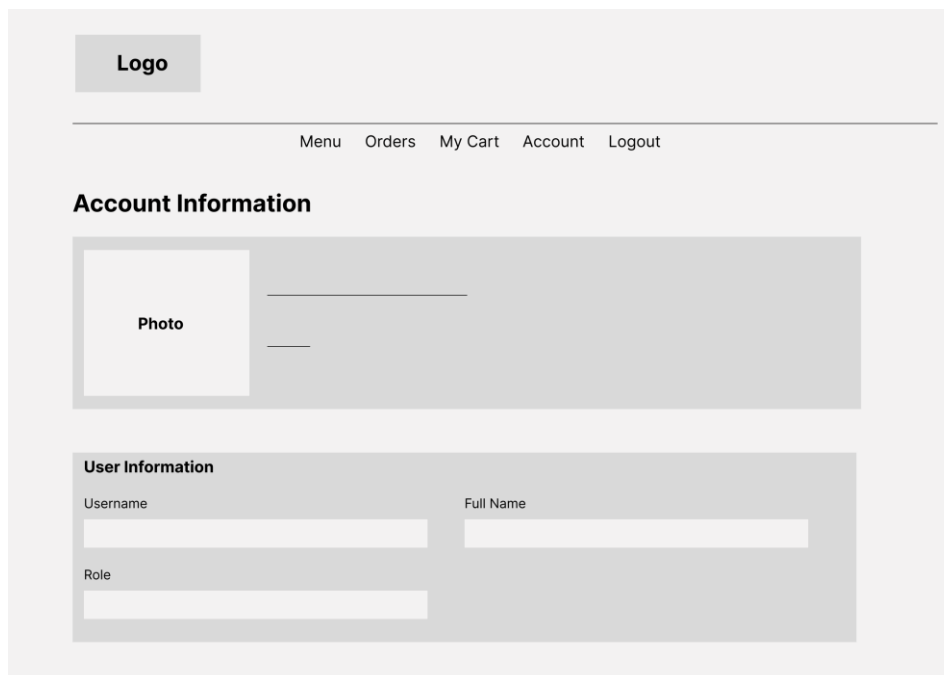


Figure 4. 11 Implementation Account Page.

4.1.7 History Order Pages

The History Order page is shown below. It's useful to recognize that each user has finished purchasing any food or drink because the order history pages for the two jobs on this website, Owner and Customer Service, look similar.

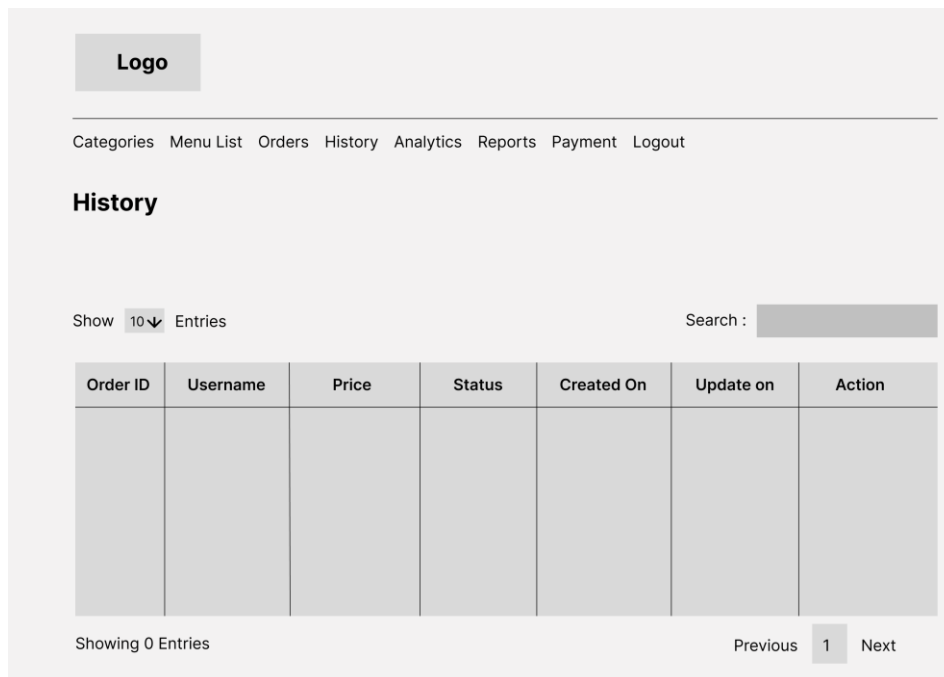


Figure 4. 12 Implementation History Page.

4.1.8 Current Order Pages

4.1.8.1 Current Order Owner & Current Order Customer Service

The Owner's current order table will reflect the most recent information once customers have finished paying for their food. Figure 5.1 shows a sketch of the page. After the Owner confirms that the payment is legitimate, the Owner must update the Status to PAID. The data will then be sent to Customer Service, who will keep the customers informed of the situation until the order has been delivered.

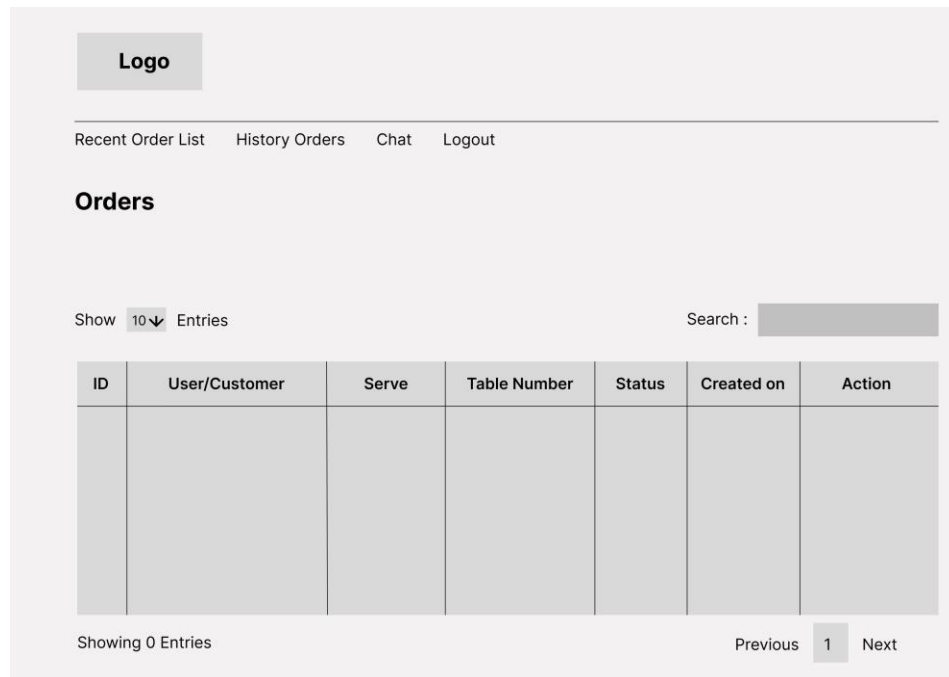


Figure 4. 13 Implementation Current Order Page.

4.1.8.2 *Current Order Customers*

The current order of the customers is similar. Customers will be redirected to this website as soon as they have completed their food payment. Only the Owner or Customer Service can update the statuses in the table. This section makes a distinction between the customer, owner, and customer service points of view. There is a Greedy Algorithm-based queuing feature in the Current Order Customer area.

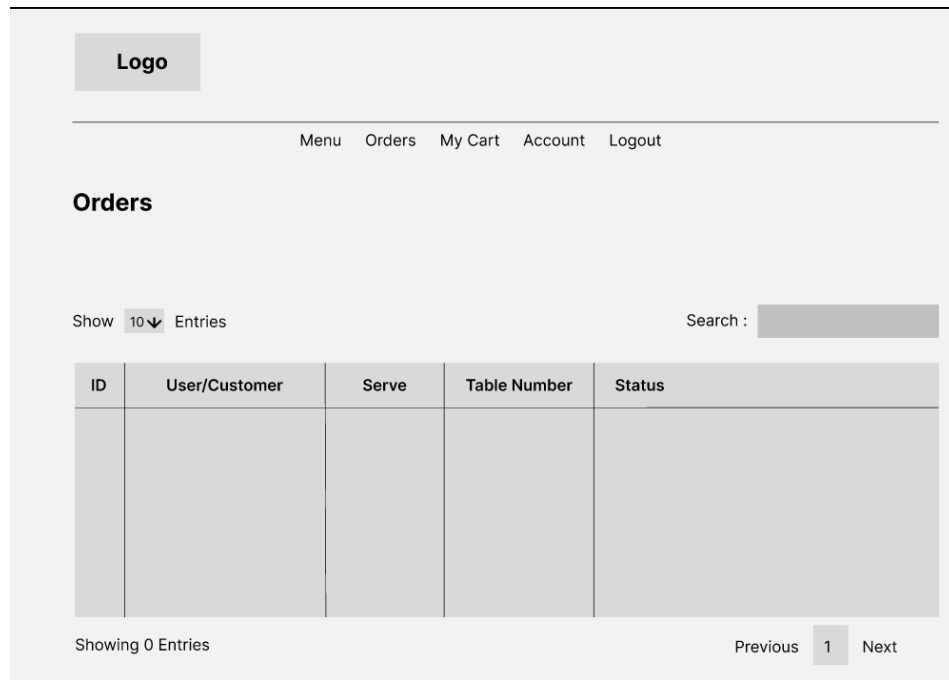


Figure 4. 14 Implementation Current Order Customers Page.

4.1.9 Payment

When Customers finish selecting food and drinks or editing the quantity on the Cart page, they will proceed to the payment page, on this page there is some info about qr from the payment method the Customers have chosen. And there is also a button to submit proof of transfer which is used for approval from the Owner.

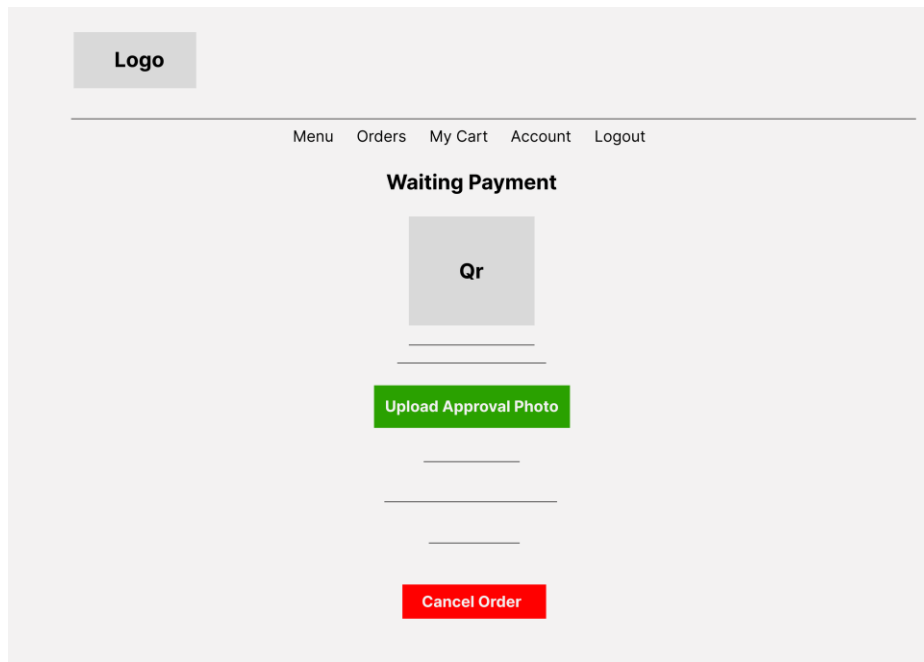


Figure 4. 15 Implementation Payment Page.

4.1.10 Payment Pages

This page is dedicated to Owners. On this page is the place for the Owner to update the payment destination account and display information from the Owner's account data.

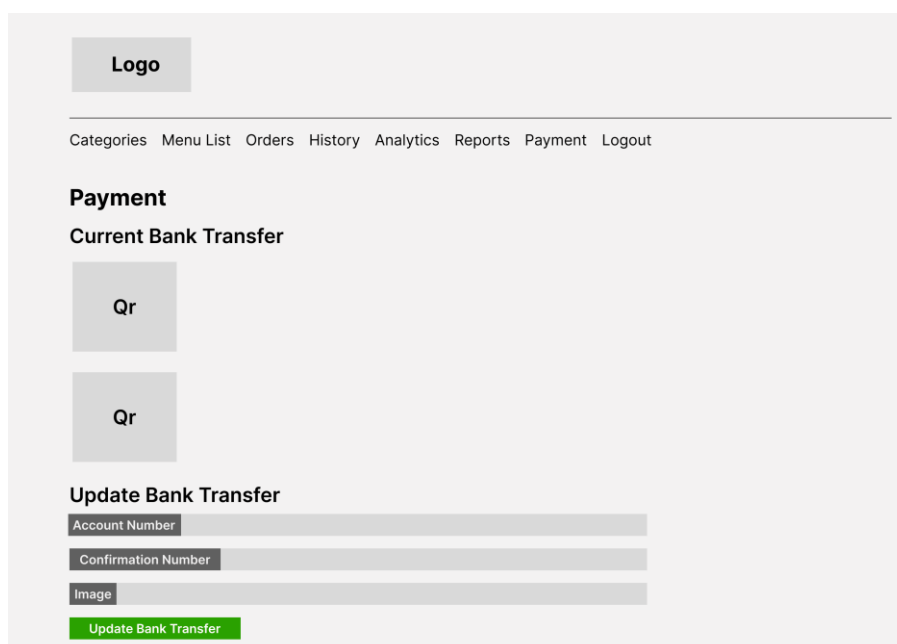


Figure 4. 16 Implementation Payment Owner Page.

4.1.11 Categories Page

This page is dedicated to Owners. On this page there is a list of menu categories available at the restaurant. Of course the Owner can update and add to the data.

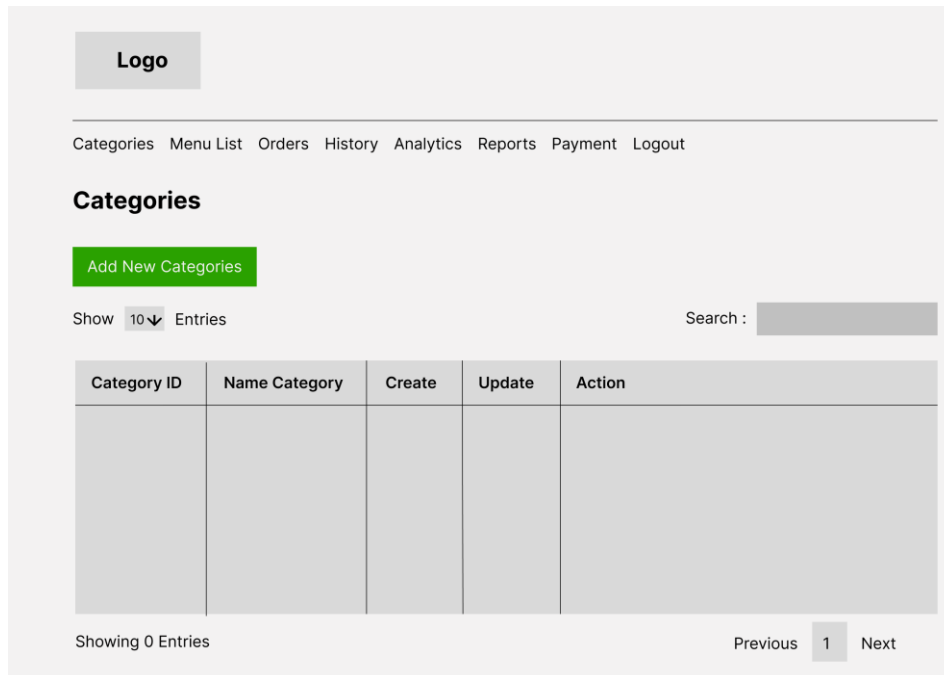


Figure 4. 17 Implementation Categories Page Owner.

4.1.12 Menulist Page

This page is dedicated to Owners. On this page there is a list of food and drink menus available at the restaurant. Of course the Owner can update and add to the data.

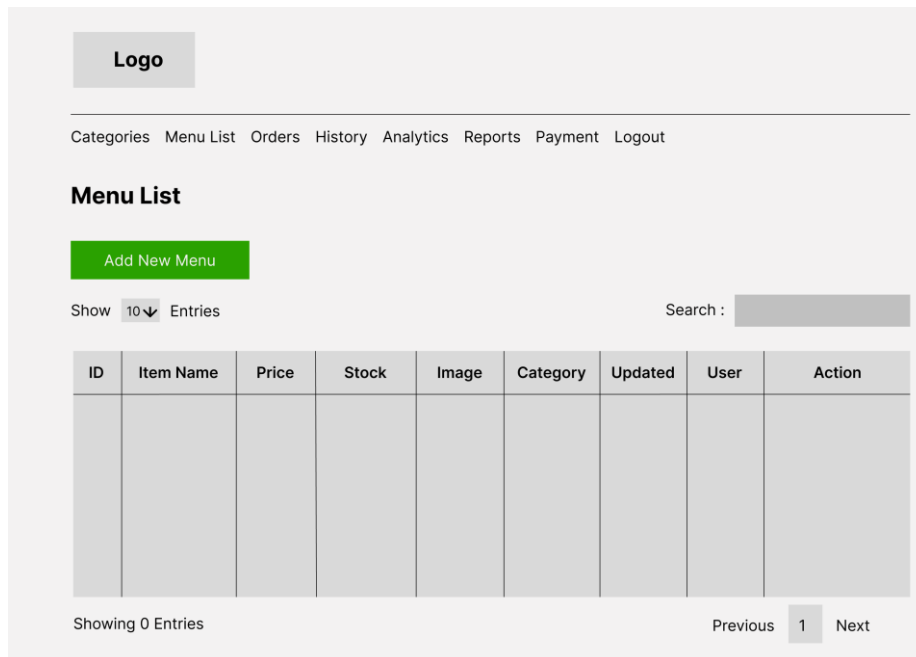


Figure 4. 18 Implementation Categories Page Owner.

4.1.13 Reports Page

This page is dedicated to Owners. The contents of this page are income data starting from per day or per month and there is a date range which is used to filter income per date that the Owner chooses.

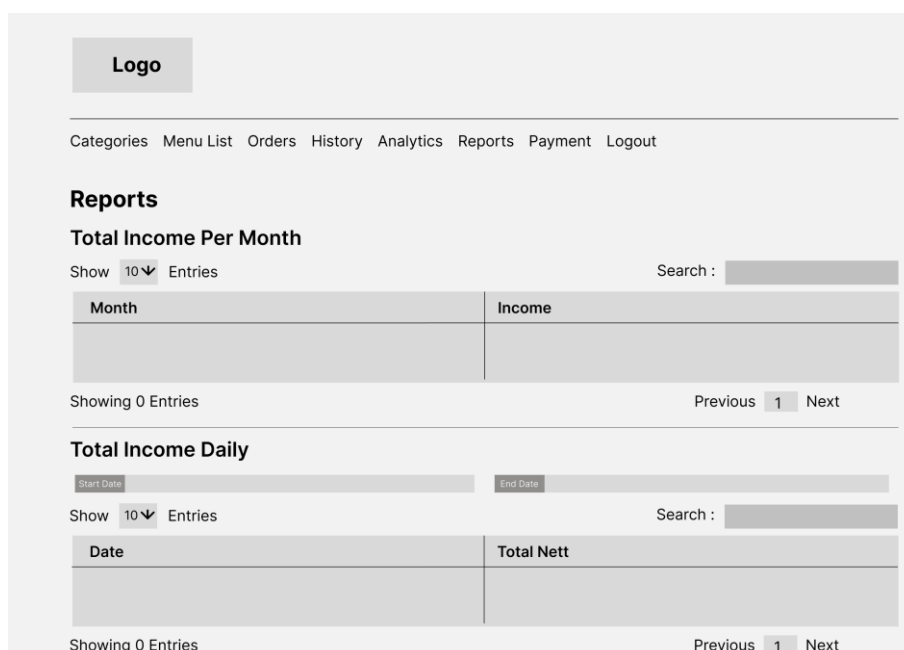


Figure 4. 19 Implementation Reports Page Owner.

4.1.14 Chat Pages

Below is one of the main features of this Javanese Restaurant website. Customers can freely communicate with the kitchen for anything related to their order queue.

4.1.14.1 Chat Customer Service

To answer questions about queues of orders from customers, the developer also prepared a chat view for Customer Service. Customer Service simply selects the name of the customer on the left. So they can communicate with each other well.

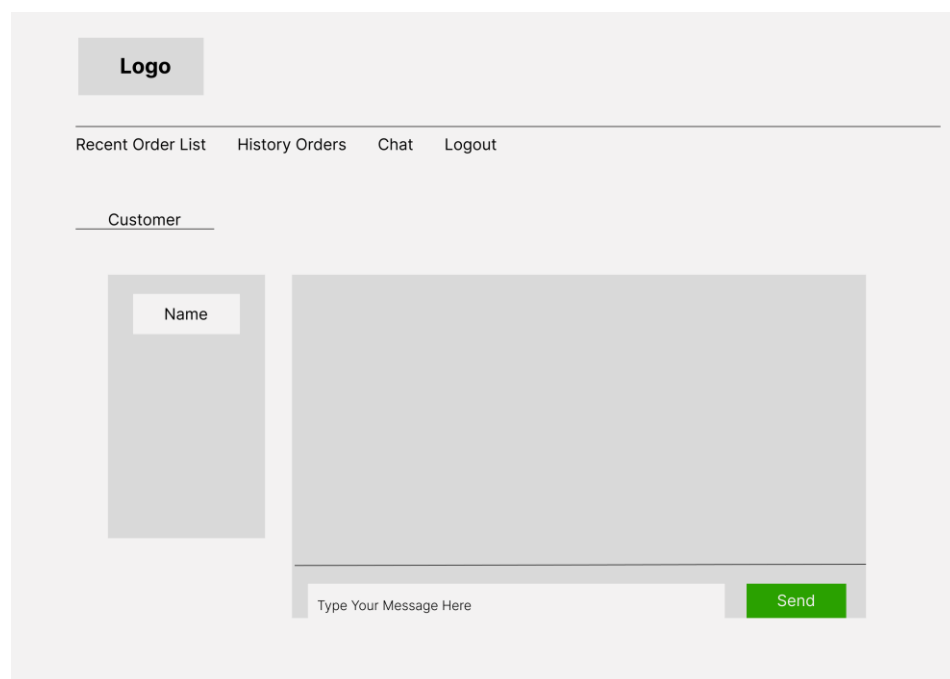


Figure 4. 20 mImplementation Chat Customer Service.

4.1.14.2 Chat Customers

Just like the Chat feature found in Customer Service, To ask questions about the order queue, the developer also prepared a chat view

for Customers. One thing that differs from Customer Service chat, chat from Customers can only be directed to the "Kitchen".

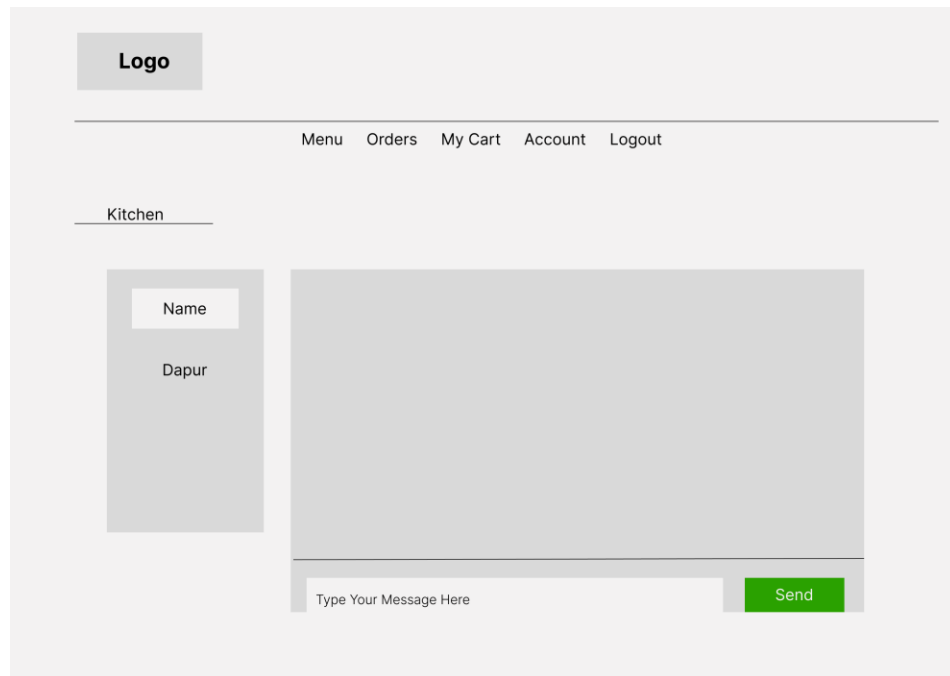


Figure 4. 21 Implementation Chat Customers.

4.1.15 Analytics Page

This Analytics section contains a chart that displays the number of orders per month. It can be downloaded in png, jpeg and pdf formats.



Figure 4. 22 Implementation Analytics Page.

4.2 Class Diagram

Classes that will be implemented in the application.

CHAPTER V

SYSTEM IMPLEMENTATION

This chapter will explain how the website will be built, ensuring that the information system is used and operational and meets quality standards. and in this chapter we will focus on the crud section as an intermediary for storing data to the database and also we will discuss how the sorting, queuing and estimated time algorithms for cooking work on this website.

5.1 User Interface

There are fourteen interfaces in this system: Login and register, Menu Page & Category List, Menu List, Cart Page, Account Page, History Order Pages, Current Order Pages, Payment Customer, Payment Owner, Category Page, Menulist Page, Report Page, Chat Page, and Analytics Page. The design concept is to make the user easy to access the wanted features because it is simple and understandable.

5.1.1 Login & Register

On the main menu page, users can select a button to enter as a customer, and to enter as Owner and Customer Service using only one account because the Owner and Customer Service functions are administrators of the restaurant. To log in as a user, if you don't have an account, you can register first. This page has a function to bring you in as your role, and the Owner can add food or drinks and approve Customers buying goods, and finally users can search for products and buy products. The front page is shown in Figure 5.1.

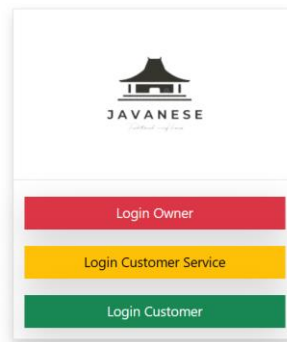
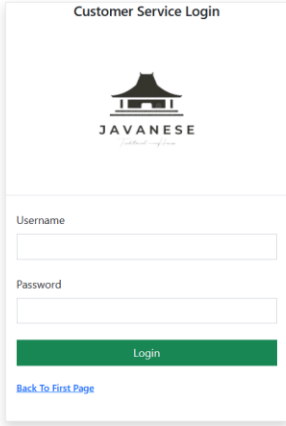


Figure 5. 1 Implementation first page.

5.1.1.1 Login Owner & Login Customer Service

On the Login Owner and Customer Service page, you can login by account Owner and Customer Service, for that two you cannot register an account because for Owner and Customer Service account just have one account. The Login Owner page is shown by Figure 5.2 and for Customer Service is shown by Figure 5.3.

Figure 5. 2 Implementation of Owner login page.

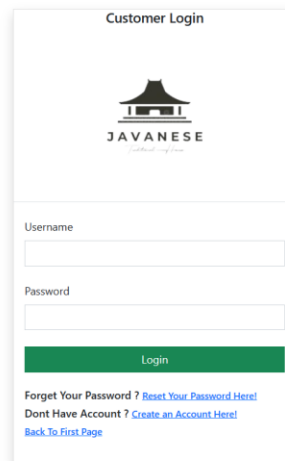


The screenshot shows a login form titled "Customer Service Login". At the top center is the JAVANESE logo, which consists of a stylized house icon above the word "JAVANESE" and the tagline "Javanese Restaurant". Below the logo are two input fields: "Username" and "Password". A green "Login" button is positioned below the password field. At the bottom left, there is a blue link labeled "Back To First Page".

Figure 5. 3 Implementation of Customer Service login page.

5.1.1.2 Login Customers

For the Customer account you can register first and after that you can login using your account and after login you will bring it to the main page of Customer page. The Login User page is shown by Figure 5.4.



Customer Login

JAVANESE
JAVANESE

Username

Password

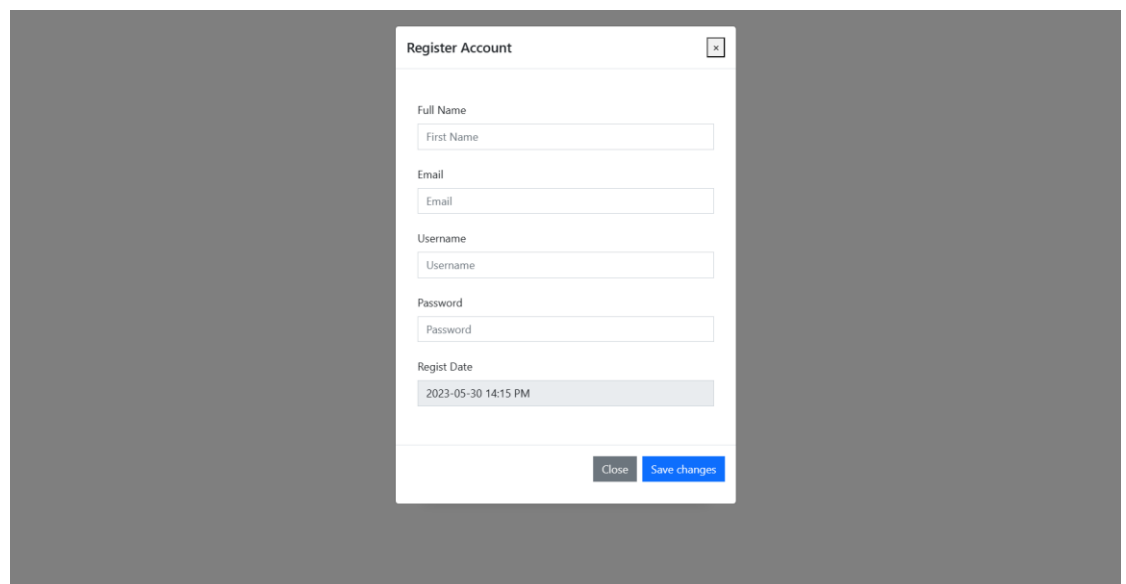
Login

[Forget Your Password ? Reset Your Password Here!](#)
[Dont Have Account ? Create an Account Here!](#)
[Back To First Page](#)

Figure 5. 4 Implementation of Customer login page.

5.1.1.3 Register Customers

Below is a form for new customer registration, they only need to fill in their full name, username and password. The roles and dates are automatically filled in according to the day they registered. The front page is shown in Figure 5.5.



Register Account

Full Name
First Name

Email
Email

Username
Username

Password
Password

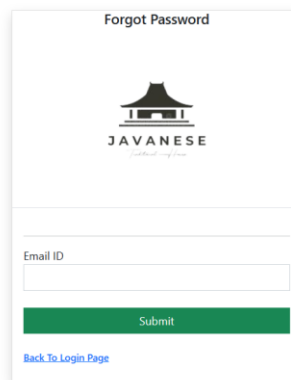
Register Date
2023-05-30 14:15 PM

Close Save changes

Figure 5. 5 Implementation of Register Customer page.

5.1.2 *Forgot Password*

When a customer logs in using his account, but they forgot with the password. On this page the customer can change their password. Customer need to input their email. Filled email is the email used to register for this platform. Forgot Password Page is shown in Figure 5.6.



The screenshot shows a web page titled "Forgot Password". At the top center is the logo for "JAVANESE", which features a stylized house icon above the text "JAVANESE" and the tagline "Cultural of Java" below it. Below the logo is a horizontal line. Underneath the line is a text input field labeled "Email ID". Below the input field is a green rectangular button with the text "Submit" in white. At the bottom left of the page, there is a blue text link that says "Back To Login Page".

Figure 5. 6 Implementation of Forgot Password page.

After the customer input their email, the email for reset password will sended in their email. By clicking the link in their email, on this page the customer can reset the password. Customer need to input new password and cofirmation password. If the confirmation password and the new password is different, it will be failed, and need to filled again. Reset Password Page is shown in Figure 5.7.

Reset Password

JAVANESE

Email

New Password

Confirm Password

Reset Password

Figure 5. 7 Implementation of Reset Password page.

5.1.3 Menu Page & Category List

When a customer logs in using his account, he will be taken to the next page, namely the Customer page menu. On this page the customer is given a choice, between going to a category from the special menu that is displayed, or selecting food items one by one using the category list. Home Page Menu Page is shown in Figure 5.8.

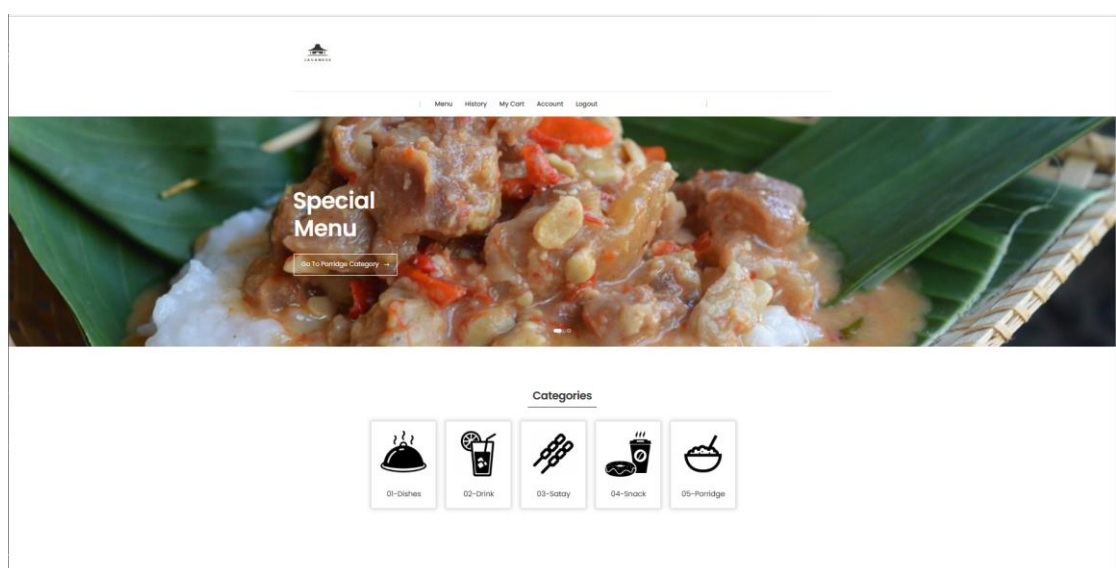


Figure 5. 8 Implementation of Menu & Category page.

5.1.4 Menu List

When a customer selects a category, they will be directed to the Menu list page, here there is an arrangement of menus based on the category selected by the customer. And on this page there is also an add to cart button in the action column, also bubble sort feature which is useful for adding the food selected by the customer to the basket which will be continued to the My Cart page. The Menu List home page is shown in Figure 5.9.

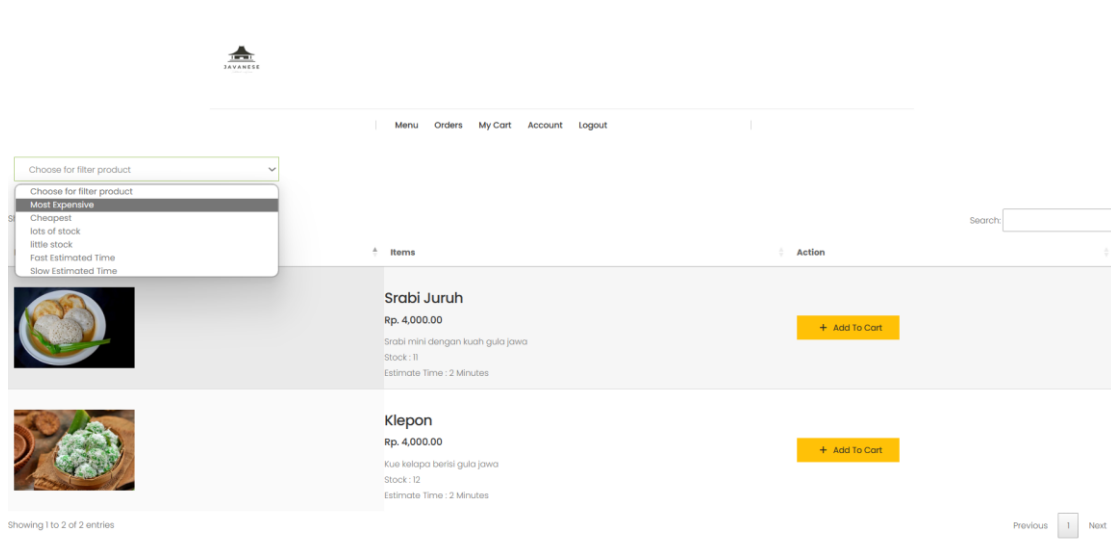


Figure 5. 9 Implementation of Menu List page.

5.1.5 Cart Page

This page is a continuation when the User has finished entering the food and drinks they want to buy. On this page, users can add, subtract, and delete orders that they put in the basket. And can immediately fill in the data for

payment when it's finished editing. The Cart Page home page is shown in Figure 5.10.

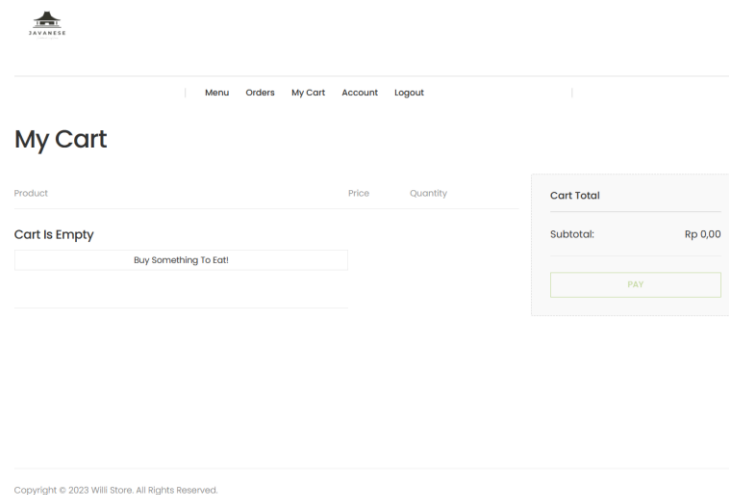


Figure 5. 10 Implementation of Cart page.

5.1.6 Account Page

On the following account page, there is some information such as the full name, role, and username of the customer according to the session that is logged in using their respective accounts. The Account Page home page is shown in Figure 5.11.

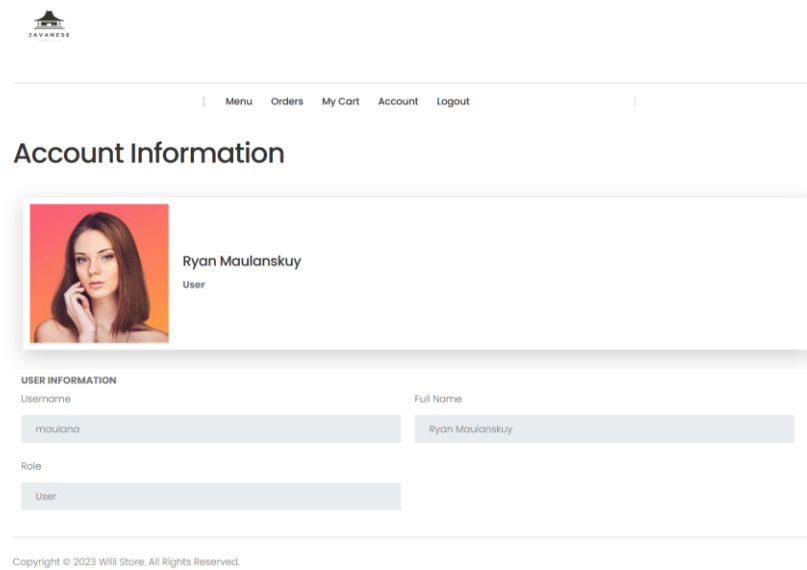


Figure 5. 11 Implementation of Account page.

5.1.7 History Order Pages

Below is the History Order page. Of the 2 roles on this website, Owner and Customer Service, they both have an order history page and the appearance is made similar, it's useful to know that each user has finished buying any food or drink. As can be seen below, there is a status column that shows which stage the order has reached. And those who edit the status are only the Owner and Customer Service. The History Order Page home page is shown in Figure 5.12.

Categories Menu List Orders History Analytics Reports Payment Logout

History

Show 10 entries

Order_ID	Username	Price	Status	Created_on	Updated_on	Action
54	hakim	Rp. 24,000.00	OK / Delivered	13/03/2023 09:05:08	13/03/2023 09:05:08	Detail
57	hakim	Rp. 34,000.00	OK / Delivered	14/03/2023 08:47:50	14/03/2023 08:47:50	Detail
58	moutono	Rp. 30,000.00	OK / Delivered	14/03/2023 09:53:37	14/03/2023 09:53:37	Detail
59	hakim	Rp. 15,000.00	OK / Delivered	14/03/2023 10:09:48	14/03/2023 10:09:48	Detail
64	hakim	Rp. 34,000.00	OK / Delivered	14/03/2023 16:00:09	14/03/2023 16:00:09	Detail
65	hakim	Rp. 10,000.00	OK / Delivered	15/03/2023 08:49:32	15/03/2023 08:49:32	Detail
66	moutono	Rp. 15,000.00	OK / Delivered	15/03/2023 09:50:40	15/03/2023 09:50:40	Detail
67	hakim	Rp. 20,000.00	OK / Delivered	15/03/2023 09:51:36	15/03/2023 09:51:36	Detail
68	hakim	Rp. 24,000.00	OK / Delivered	16/03/2023 09:18:05	16/03/2023 09:18:05	Detail
69	hakim	Rp. 30,000.00	OK / Delivered	16/03/2023 09:22:37	16/03/2023 09:22:37	Detail

Showing 1 to 10 of 15 entries Previous 1 2 Next

Copyright © 2023 W8 Store. All Rights Reserved

Figure 5. 12 Implementation of History Order page.

5.1.8 Current Order Pages

5.1.8.1 Current Order Owner & Current Order Customer Service

When Customers finish paying for food, the latest data will appear in the Owner's current order table. The front page of Current Order Owner and Customer Service is shown in Figure 5.13. then the Owner checks whether the payment is valid, when it is valid, the Owner must update the Status to PAID. After that, the data will be forwarded to Customer Service, CS will always update the status until the order has been delivered to the Customers.

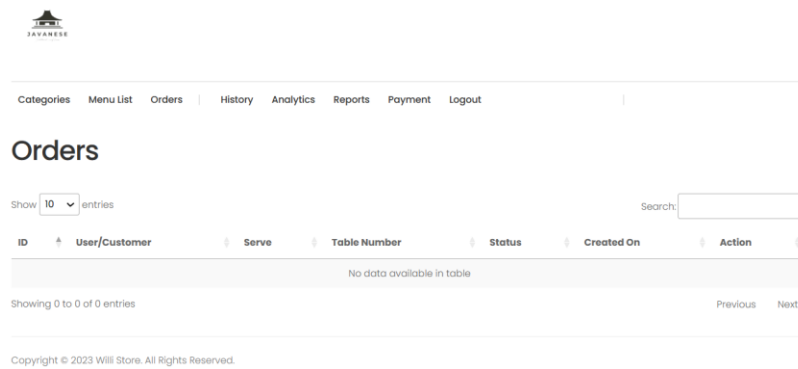
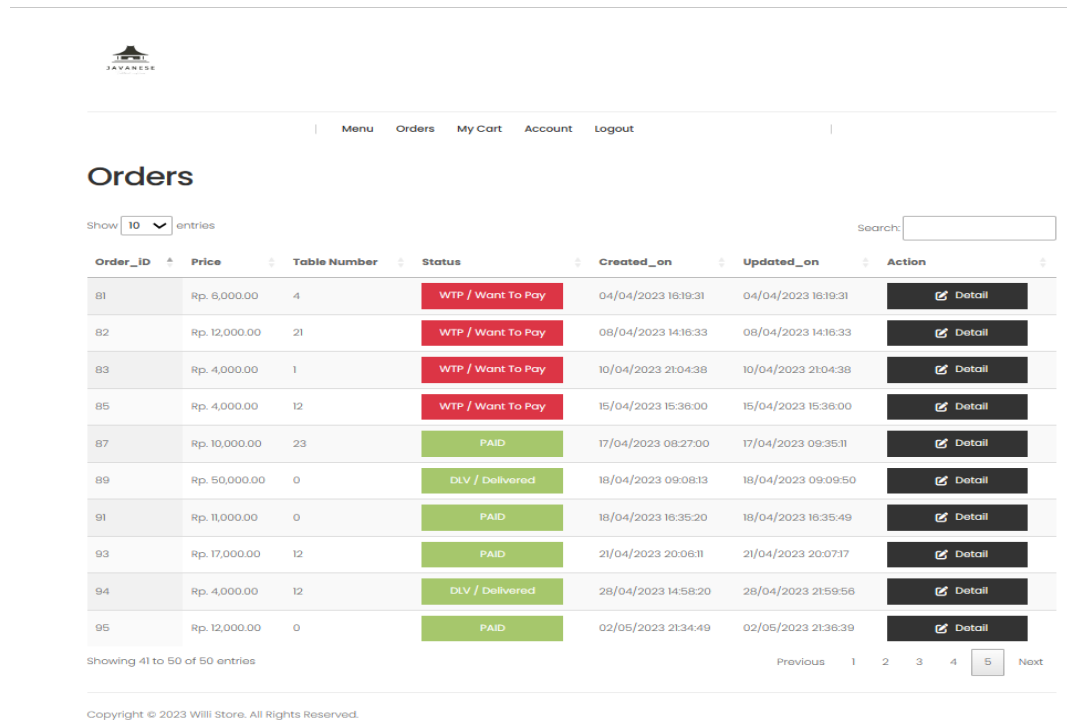


Figure 5. 13 Implementation of Current Order Owner page.

5.1.8.2 *Current Order Customers*

Likewise with Customers' Current Order. When customers finish paying for food, they will be immediately directed to this page. The table contains statuses, which can only be updated by the Owner or Customer Service. It is in this section that distinguishes between Customer, Owner & Customer Service views. In the Current Order Customer section, they have a queuing feature using the Greedy Algorithm algorithm. The Current Order Customer home page is shown in Figure 5.14.



Menu Orders My Cart Account Logout

Orders

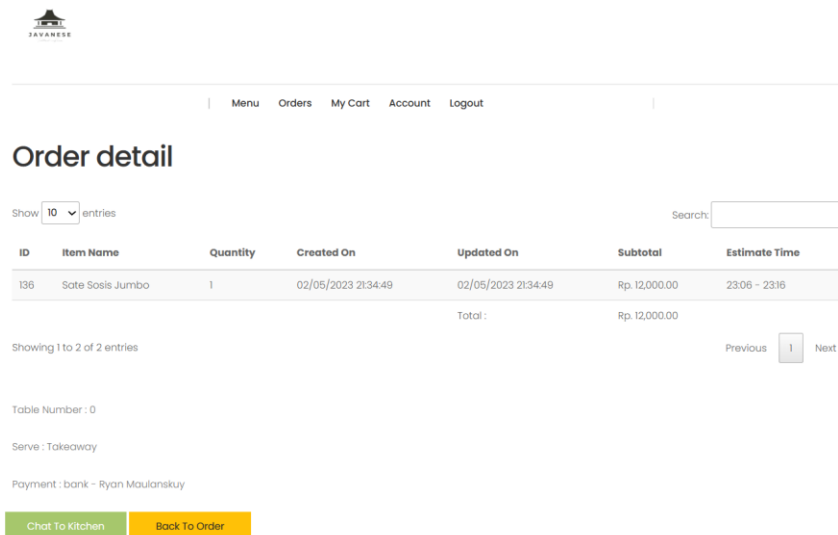
Show entries Search:

Order_ID	Price	Table Number	Status	Created_on	Updated_on	Action
81	Rp. 6,000.00	4	WTP / Want To Pay	04/04/2023 16:19:31	04/04/2023 16:19:31	Detail
82	Rp. 12,000.00	21	WTP / Want To Pay	08/04/2023 14:16:33	08/04/2023 14:16:33	Detail
83	Rp. 4,000.00	1	WTP / Want To Pay	10/04/2023 21:04:38	10/04/2023 21:04:38	Detail
85	Rp. 4,000.00	12	WTP / Want To Pay	15/04/2023 15:36:00	15/04/2023 15:36:00	Detail
87	Rp. 10,000.00	23	PAID	17/04/2023 08:27:00	17/04/2023 09:35:11	Detail
89	Rp. 50,000.00	0	DLV / Delivered	18/04/2023 09:08:13	18/04/2023 09:09:50	Detail
91	Rp. 11,000.00	0	PAID	18/04/2023 16:35:20	18/04/2023 16:35:49	Detail
93	Rp. 17,000.00	12	PAID	21/04/2023 20:06:11	21/04/2023 20:07:17	Detail
94	Rp. 4,000.00	12	DLV / Delivered	28/04/2023 14:58:20	28/04/2023 21:59:56	Detail
95	Rp. 12,000.00	0	PAID	02/05/2023 21:34:49	02/05/2023 21:36:39	Detail

Showing 41 to 50 of 50 entries Previous 1 2 3 4 **5** Next

Copyright © 2023 Willi Store. All Rights Reserved.

Figure 5. 14 Implementation Current Order Customers Page.



Menu Orders My Cart Account Logout

Order detail

Show entries Search:

ID	Item Name	Quantity	Created On	Updated On	Subtotal	Estimate Time
136	Sate Sosis Jumbo	1	02/05/2023 21:34:49	02/05/2023 21:34:49	Rp. 12,000.00	23:06 - 23:16
Total :					Rp. 12,000.00	

Showing 1 to 2 of 2 entries Previous **1** Next

Table Number : 0

Serve : Takeaway

Payment : bank - Ryan Maulanskiy

[Chat To Kitchen](#) [Back To Order](#)

Figure 5. 15 Implementation Estimation Time in orderdetail

5.1.9 Payment

When customers finish selecting food and drinks or editing the quantity on the Cart page, they will proceed to the payment page, on this page there is some info about the account the payment is made to, such as the qr code of the payment method the Customer has chosen. And there is also a button to submit proof of transfer which is used for approval of proof of transfer from the Owner. Payment Customer is shown in Figure 5.16.

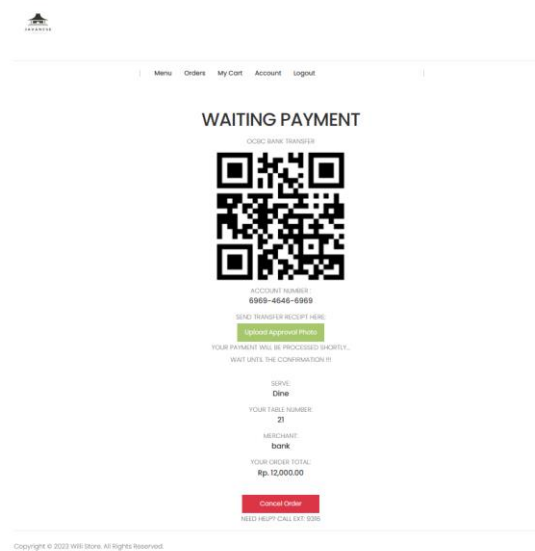


Figure 5. 16 Implementation Payment Customers Page.

5.1.10 Payment Pages

This Payment page is specifically for the Owner only. In here there is information about the destination account of the restaurant for payment. Here the Owner can update the Qr code and the destination account. The Payment Page page is shown in Figure 5.17.

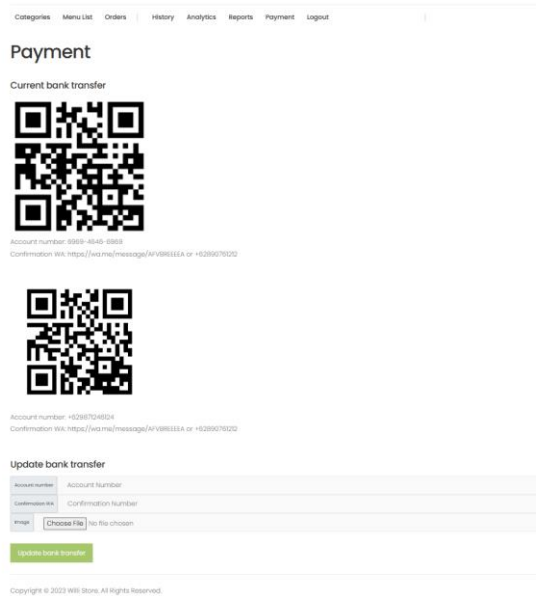


Figure 5. 17 Implementation Payment Page.

5.1.11 Categories Page

This page is dedicated to Owners as well. On this page there is a list of menu categories available at the restaurant. Of course the Owner can update and add to the data. The Categories page is shown in Figure 5.18.

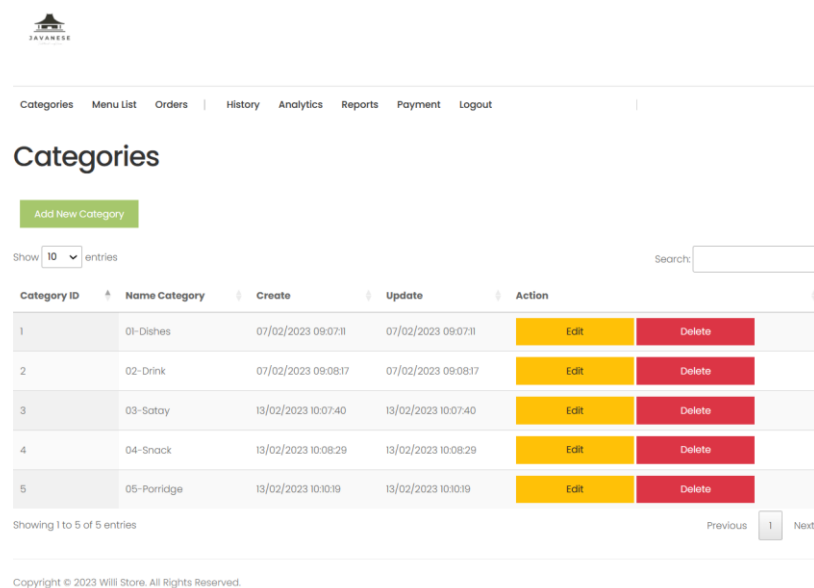


Figure 5. 18 Implementation Categories Page.

5.1.12 Menulist Page

This page is dedicated to Owners as well. On this page there is a list of food and drink menus available at the restaurant. Of course the Owner can update and add to the data. Writing page is shown in Figure 5.19.

The screenshot shows a web application interface for managing a menu. At the top, there is a navigation bar with links for Categories, Menu List, Orders, History, Analytics, Reports, Payment, and Logout. Below the navigation bar, the title 'Menulist' is displayed. A green button labeled 'Add New Menu' is positioned above a table. The table has a search bar and a dropdown menu set to '10 entries'. The table columns are: ID, Item Name, Price, Stock, Image, Category, Updated, and User. There are 8 rows of menu items, each with an 'Edit' button. The items are:

ID	Item Name	Price	Stock	Image	Category	Updated	User
6	Bakmi Godek Jawa	Rp. 20.000,00	8	bakmi.png	1	16/02/2023 08:18:34	1
7	Wedang Jaje	Rp. 7.000,00	15	jaje.png	2	16/02/2023 08:18:45	1
8	Wedang Ukuh	Rp. 8.000,00	16	ukuh.png	2	13/02/2023 10:58:00	1
9	Sate Usus	Rp. 5.000,00	14	usuk.png	3	13/02/2023 10:58:03	1
10	Sate Isas Jumbo	Rp. 12.000,00	15	isapung	3	13/02/2023 10:58:07	1
11	Sate Juring	Rp. 4.000,00	13	sate.png	4	13/02/2023 10:22:24	1
12	Klepon	Rp. 4.000,00	14	klepon.png	4	16/02/2023 08:18:57	1
13	Bubur Tumpang	Rp. 8.000,00	16	tumpang.png	5	13/02/2023 10:44:37	1

At the bottom of the table, it says 'Showing 1 to 8 of 8 entries'. There are 'Previous' and 'Next' buttons. A copyright notice at the bottom reads 'Copyright © 2023 Wili Store. All rights reserved.'

Figure 5. 19 Implementation Menulist Page.

5.1.13 Reports Page

This page is dedicated to Owners. The contents of this page are income data starting per day or per month and there is a date range that is used to filter income per date that the Owner chooses. Report page is shown in Figure 5.20.

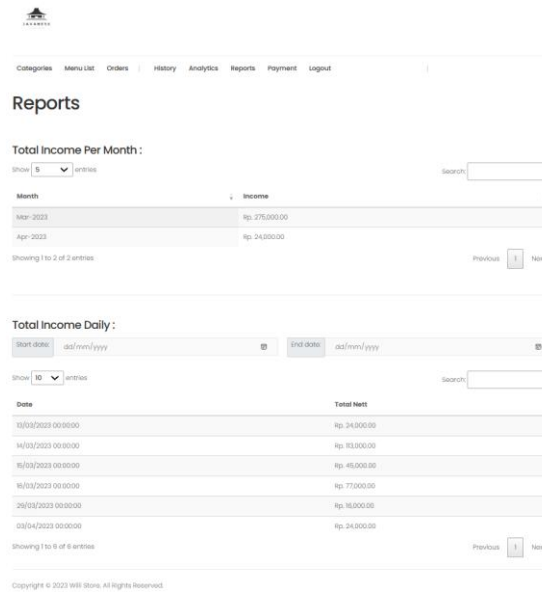


Figure 5. 20 Implementation Reports Page.

5.1.14 Chat Pages

5.1.14.1 Chat Customer Service

To answer questions about queues of orders from customers, the developer also prepared a chat view for Customer Service. Customer Service simply selects the name of the customer on the left. So they can communicate with each other well. Customer Service Chat Page by Figure 5.21.

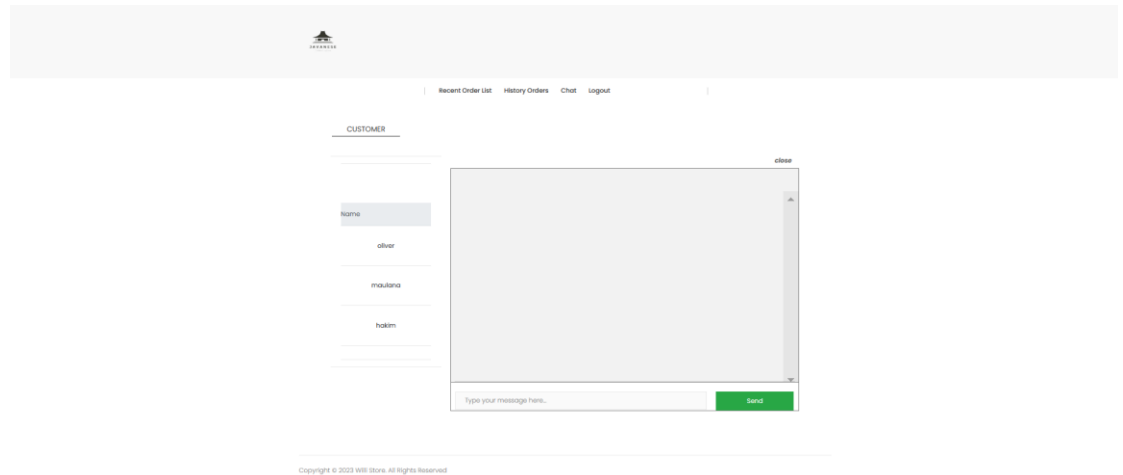


Figure 5. 21 Implementation Chat Customer Service Page.

5.1.14.2 Chat Customers

Meanwhile, for chat from the customer, the different thing is, the developer has prepared a special purpose for the chat only for customer service. The point is to ask about their food queue to Customer Service. Customer Chat page by Figure 5.22.

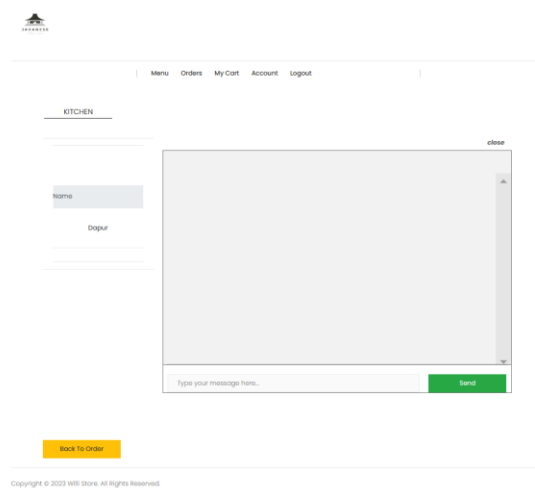


Figure 5. 22 Implementation Chat Customer Page.

5.1.15 Analytics Page

This Analytics section contains a chart that displays the number of orders per month. It can be downloaded in png, jpeg and pdf formats. The Analytics page is shown in Figure 5.23.

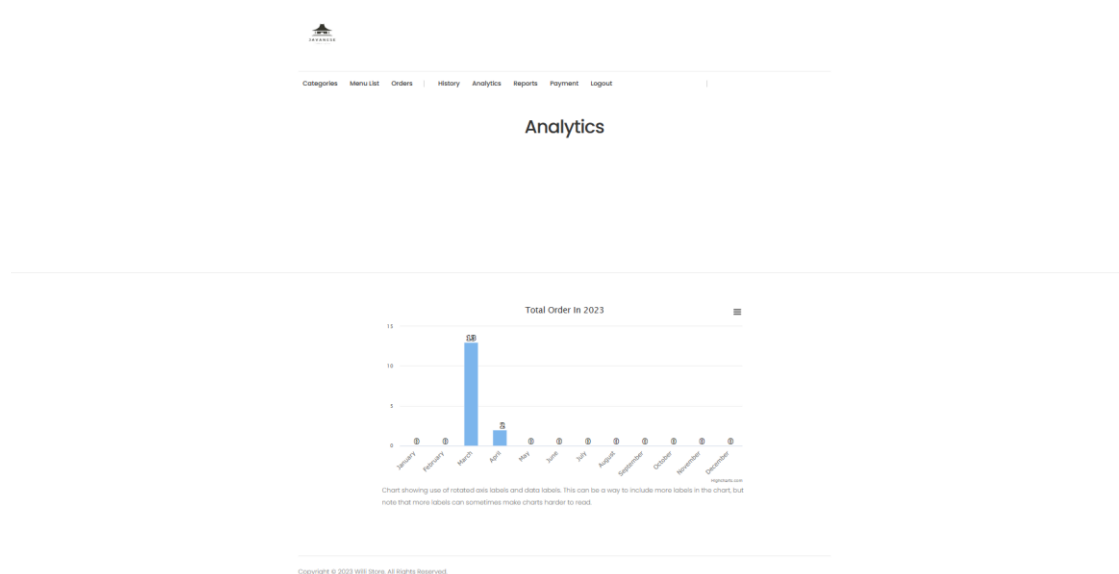


Figure 5. 23 Implementation Analytics Page.

5.2 Application Details

5.2.1 Login & Register

On the login and register page I use the MD5 encryption method. Is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used to authenticate the original message.

The first, create a new configuration for the account login and register. The following is a configuration image for the configuration made in the web config from asp.net mvc.

```

<connectionStrings>
  <add name="Finpro" providerName="System.Data.SqlClient" connectionString="Data Source=LAPTOP-D1571PGP\SQLEXPRESS;Initial Catalog=Final_Project;Integrated Security=True" />
  <add name="Final_ProjectEntities" connectionString="metadata=res://*/Models.Model.csdl|res://*/Models.Model.ssd|res://*/Models.Model.msl;provider=System.Data.SqlClient;
  provider connection string=&quot;data source=LAPTOP-D1571PGP\SQLEXPRESS;initial catalog=Final_Project;integrated security=True;multipleactiveresultsets=True;App=EntityFramework&quot;;
  providerName=System.Data.EntityClient" />
  <add name="Final_ProjectEntities_Owner" connectionString="metadata=res://*/Models.Model1.csdl|res://*/Models.Model1.ssd|res://*/Models.Model1.msl;provider=System.Data.SqlClient;
  provider connection string=&quot;data source=LAPTOP-D1571PGP\SQLEXPRESS;initial catalog=Final_Project;integrated security=True;multipleactiveresultsets=True;application name=EntityFramework&quot;;
  providerName=System.Data.EntityClient" />
  <add name="Final_ProjectEntities1_Dapur" connectionString="metadata=res://*/Models.Model2.csdl|res://*/Models.Model2.ssd|res://*/Models.Model2.msl;provider=System.Data.SqlClient;
  provider connection string=&quot;data source=LAPTOP-D1571PGP\SQLEXPRESS;initial catalog=Final_Project;integrated security=True;multipleactiveresultsets=True;application name=EntityFramework&quot;;
  providerName=System.Data.EntityClient" />
  <add name="Final_ProjectEntitiesChat" connectionString="metadata=res://*/Models.ModelChat.csdl|res://*/Models.ModelChat.ssd|res://*/Models.ModelChat.msl;provider=System.Data.SqlClient;
  provider connection string=&quot;data source=LAPTOP-D1571PGP\SQLEXPRESS;initial catalog=Final_Project;integrated security=True;multipleactiveresultsets=True;App=EntityFramework&quot;;
  providerName=System.Data.EntityClient" />
</connectionStrings>

```

Figure 5. 24 Implementation connecting to the database

Here is a picture of MD5 encryption.

```

4 references
public class encrypt
{
    //internal object us_pwd;
    4 references
    public static string GetMD5(string str)
    {
        MD5 md5 = new MD5CryptoServiceProvider();
        byte[] fromData = Encoding.UTF8.GetBytes(str);
        byte[] targetData = md5.ComputeHash(fromData);
        string byte2String = null;

        for (int i = 0; i < targetData.Length; i++)
        {
            byte2String += targetData[i].ToString("x2");
        }

        return byte2String;
    }
}

```

Figure 5. 25 Implementation of encrypt password function

5.2.1.1 Login Owner & Login Customer Service

When the user account wants to log in, the Razor page (cshtml) as the front end will make a connection between the Views and the Controller, which matches the user name that has the right to enter, and the account owner must fill in the username and password. After that click login. If the account is correct, then the login is successful.

```

<form method="post" action="/Login/LoginDapur" enctype="multipart/form-data">
  @Html.AntiForgeryToken()
  @Html.ValidationSummary(true)
  <div class="mb-4">
    <label for="username" class="form-label">Username</label>
    @Html.EditorFor(model => model.Username, new { htmlAttributes = new { @class = "form-control", @data_val = "true", @required = "required" } })
    @Html.ValidationMessageFor(model => model.Username)
  </div>
  <div class="mb-4">
    <label for="password" class="form-label">Password</label>
    @Html.EditorFor(model => model.Password, new { htmlAttributes = new { @class = "form-control", @data_val = "true", @type = "password", @required = "required" } })
    @Html.ValidationMessageFor(model => model.Password)
  </div>
  <div class="d-grid">
    <button type="submit" class="btn btn-success">Login</button><br />
  </div>
  <div class="d-grid">
    <h6>
      <a href="/Login/lamanawal">
        <span class="small">
          Back To First Page
        </span>
      </a>
    </h6>
  </div>
</form>

```

Figure 5. 26 Implementation input login owner and customer service

After the user account clicks login, the system will take the resulting data that we input and will pass it on to the controller. But before clicking login, the user will be given 3 choices of roles to login, Owner, Customer Service, and Customer. Users must know where they are logged in. For example for Owner and Customer Service, they can only log in using their respective roles, except for Owner, they can log in using a Customer Service account. Because the Owner and Customer Service are specifically for internal restaurants. The data that has been posted will be checked based on the email and password input, if the username and password are correct then your login has been successful and will be taken to the Owner dashboard page.


```

0 references
public ActionResult LoginOwner()
{
    if (Session["Username"] != null)
    {
        Session["Username"].ToString();
        return RedirectToAction("Analytics", "HomeOwner");
    }
    else
    {
        return View();
    }
}

[HttpPost]
0 references
public ActionResult LoginOwner(OwnerTBL credentials)
{
    string encrypt1 = encrypt.GetMD5(credentials.Password);
    bool userExist = entities_Owner.OwnerTBLs.Any(x => x.Username == credentials.Username && x.Password == encrypt1);
    OwnerTBL u = entities_Owner.OwnerTBLs.FirstOrDefault(x => x.Username == credentials.Username && x.Password == encrypt1);

    if (userExist)
    {
        Session["Username"] = u.Username.ToString();
        FormsAuthentication.SetAuthCookie(u.Username, false);

        return RedirectToAction("Analytics", "HomeOwner");
    }
    ModelState.AddModelError("", "udah ada");
    if (Session["Username"] == null)
    {
        return RedirectToAction("LoginOwner", "Login");
    }
    return View();
}

```

Figure 5. 27 Implementation from login owner function

5.2.1.2 Login Customers

Just like Owner and Customer Service logins, user logins also use md5 to encrypt passwords. And on the Razor Page (cshtml), the system will check whether the username and password are the same as those in the database. What differentiates the controller from Owner and Customer is the database used for the role. When the account is logged in and correct, it will be taken to the Customer dashboard page.

```

0 references
public ActionResult Login()
{
    if (Session["Username"] != null)
    {
        Session["Username"].ToString();
        return RedirectToAction("Index", "Home");
    }
    else
    {
        return View();
    }
}

[HttpPost]
0 references
public ActionResult Login(UserTBL credentials)
{
    string encrypt1 = encrypt.GetMD5(credentials.Password);
    bool userExist = entity.UserTBLS.Any(x => x.Username == credentials.Username && x.Password == encrypt1);
    UserTBL u = entity.UserTBLS.FirstOrDefault(x => x.Username == credentials.Username && x.Password == encrypt1);

    if (userExist)
    {
        Session["Username"] = u.Username.ToString();
        Session["Fullname"] = u.Fullname.ToString();
        Session["ID"] = u.ID;

        FormsAuthentication.SetAuthCookie(u.Username, false);

        return RedirectToAction("Index", "Home");
    }
    ModelState.AddModelError("", "udah ada");
    if (Session["Username"] == null)
    {
        return RedirectToAction("Login", "Login");
    }
    return View();
}

```

Figure 5. 28 Implementation from login customer function

5.2.1.3 Register Customers

When there are new customers who want to log in but don't have an account yet, they must first register their account. It's the same as logging in, here the Customer inputs Username, Password, Fullname, Role, and Register Date. For roles and datetime, they are auto-filled in value. When finished, they will be directed to the login page of the Customer.

```

<form method="post" action="/Login/Register" enctype="multipart/form-data">
  @Html.ValidationSummary()
  @Html.AntiForgeryToken()
  <div class="modal-body">
    <div class="card-body">
      <div class="mb-4">
        <label for="namapanjang" class="form-label">Full Name</label>
        @Html.TextBoxFor(m => m.Fullname, new { @class = "form-control", @placeholder = "First Name", @required = "required" })
        @Html.ValidationMessageFor(m => m.Fullname, "", new { @class = "badge badge-danger" })
      </div>
      <div class="mb-4">
        <label for="username" class="form-label">Username</label>
        @Html.TextBoxFor(m => m.Username, new { @class = "form-control", @placeholder = "Username", @required = "required" })
        @Html.ValidationMessageFor(m => m.Username, "", new { @class = "badge badge-danger" })
      </div>
      <div class="mb-4">
        <label for="password" class="form-label">Password</label>
        @Html.PasswordFor(m => m.Password, new { @class = "form-control", @placeholder = "Password", @required = "required" })
        @Html.ValidationMessageFor(m => m.Password, "", new { @class = "badge badge-danger" })
      </div>
      <div class="mb-4">
        <label hidden for="role" class="form-label">Role</label>
        @Html.TextBoxFor(m => m.GroupID, new { @class = "form-control", @placeholder = "Pembeli", @readonly = "readonly", Value = "3", @type = "hidden" })
        @Html.ValidationMessageFor(m => m.GroupID, "", new { @class = "badge badge-danger" })
      </div>
      <div class="mb-4">
        <label for="Regis" class="form-label">Regist Date</label>
        @Html.TextBoxFor(m => m.Created_at, new { @class = "form-control", @placeholder = "Tanggal Regis", @readonly = "readonly", Value = DateTime.Now.ToString("yyyy-MM-dd HH:mm tt") })
        @Html.ValidationMessageFor(m => m.Created_at, "", new { @class = "badge badge-danger" })
      </div>
    </div>
  </div>
  <div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Close</button>
    <button type="submit" class="btn btn-primary">Save changes</button>
  </div>
</form>

```

Figure 5. 29 Implementation input register customer

And for the controller part, in this register method they input or enter the values of the Username, Password, Fullname, Role and Register Date into the database. Unlike the login, the method is checking the username and password in the database.

```

LoginController.cs | UserModel.cs | Chat.cshtml | HomeController.cs | OwnerModel.cs | Reports.cshtml | Analytics.cs
Final Project | Final_Project.Controllers.LoginController | Register

28 public ActionResult lamanawal()
29 {
30     return View();
31 }
32
33 [HttpPost]
34 [ValidateAntiForgeryToken]
35 public ActionResult Register(UserTBL_user)
36 {
37     if (ModelState.IsValid)
38     {
39         var check = entity.UserTBLs.FirstOrDefault(s => s.Username == _user.Username);
40         if (check == null)
41         {
42             _user.Password = encrypt.GetMD5(_user.Password);
43             entity.Configuration.ValidateOnSaveEnabled = false;
44             entity.UserTBLs.Add(_user);
45             entity.SaveChanges();
46             return RedirectToAction("Login", "Login");
47         }
48         else
49         {
50             ViewBag.error = "Username already exists";
51             return RedirectToAction("Login", "Login");
52         }
53     }
54     return View();
55 }
56

```

Figure 5. 30 Implementation of register customer function

5.2.2 *Forgot Password*

The image below is function for forgot password. Using entity framework for connect to the database. Overall in this function is used for call the send email function. Here it can see in the controller they call SendVerificationLinkEmail function. That is for calling the function of send email.

```
[HttpPost]
0 references
public ActionResult ForgotPassword(string EmailID)
{
    //Verify Email ID
    //Generate Reset password link
    //Send Email
    string message = "";

    using (Final_ProjectEntities1 dc = new Final_ProjectEntities1())
    {
        var account = dc.UserTBLS.Where(a => a.Email == EmailID).FirstOrDefault();
        if (account != null)
        {
            //Send email for reset password
            string resetCode = Guid.NewGuid().ToString();
            SendVerificationLinkEmail(account.Email, resetCode, "ResetPassword");
            account.ResetPasswordCode = resetCode;
            //This line I have added here to avoid confirm password not match issue , as we had added a confirm password property
            //in our model class in part 1
            dc.Configuration.ValidateOnSaveEnabled = false;
            dc.SaveChanges();
            message = "Reset Password Link has been Sent to Your Email Account.";
        }
        else
        {
            message = "Account Email Not Found";
        }
    }
    ViewBag.Message = message;
    return View();
}
```

Figure 5. 31 Implementation of forgot password function

The image below is function for SendEmailVerification. This function will be called when the user has input their email on the forgot password page. Then after input, the function will run and send an email using the smtp client method. And the link from the password reset page will be displayed with the words listed in the body variable.

```

[NonAction]
1 reference
public void SendVerificationLinkEmail(string emailID, string activationCode, string emailFor = "VerifyAccount")
{
    var verifyUrl = "/Login/" + emailFor;
    var link = Request.Url.AbsoluteUri.Replace(Request.Url.PathAndQuery, verifyUrl);
    var fromEmail = new MailAddress("muhammad.hardiyanto@student.president.ac.id", "E-Food Owner");
    var toEmail = new MailAddress(emailID);
    var fromEmailPassword = "20020111"; // Replace with actual password
    string subject = "";
    string body = "";
    if (emailFor == "ResetPassword")
    {
        subject = "Reset Password";
        body = "Hi," + "<br/>" + "We got request for reset your account password. Please click on the below link to reset your password" +
            "<br/><br/><a href=" + link + ">Reset Password link</a>";
    }
    SmtplibClient smtp = new SmtplibClient();
    {
        smtp.Host = "smtp.gmail.com";
        smtp.Port = 587;
        smtp.EnableSsl = true;
        NetworkCredential nc = new NetworkCredential("muhammad.hardiyanto@student.president.ac.id", "20020111");
        smtp.UseDefaultCredentials = true;
        smtp.Credentials = nc;
        nc = new NetworkCredential(fromEmail.Address, fromEmailPassword);
    }
}
using (var message = new MailMessage(fromEmail, toEmail)
{
    Subject = subject,
    Body = body,
    IsBodyHtml = true
})
smtp.Send(message);
}

```

Figure 5. 32 Implementation of send email function

The image below is a views of input email for send verification reset password. After this data being submitted, the function of send email function in Figure 5.32 will executed.

```

@using (Html.BeginForm())
{
    <div class="form-horizontal">
        <hr />
        <div class="text-success">
            @ViewBag.Message
        </div>
        <div class="form-group">
            <label class="control-label col-md-2">Email ID</label>
            @Html.TextBox("EmailID", "", new { @class = "form-control" })
        </div>
        <br />
        <div class="form-group">
            <div class="d-grid">
                <input type="submit" value="Submit" class="btn btn-success" />
            </div>
        </div>
    </div>
}

```

Figure 5. 33 Implementation of input email verification

The image below is a function of reset password or updated a new password. Using entity framework for updating the password in database, After user input new password, This function will be running. In here the function can read if confirm password is different with the new password.

```
[HttpPost]
0 references
public ActionResult ResetPassword(string email, string newPassword, string confirmPassword)
{
    // Temukan pengguna berdasarkan email
    var user = entity.UserTBls.FirstOrDefault(u => u.Email == email);

    if (user != null)
    {
        // Pastikan new password dan confirm password cocok
        if (newPassword == confirmPassword)
        {
            // Perbarui password pengguna
            user.Password = encrypt.GetMD5(newPassword);
            // Simpan perubahan ke database
            entity.SaveChanges();

            // Tampilkan pesan sukses
            ViewBag.Message = "Password has been changed and updated.";

            return RedirectToAction("Login", "Login");
        }
        else
        {
            // Tampilkan pesan error jika new password dan confirm password tidak cocok
            ViewBag.Error = "New password and confirm password do not match.";
            return View();
        }
    }
    else
    {
        // Tampilkan pesan error jika email tidak ditemukan
        ViewBag.Error = "Email not registered.";
        return View();
    }
}
```

Figure 5. 34 Implementation of reset password function

The image below is a views of input new password and confirmation password for reset the password. After this data being submitted, the function of reset password in Figure 5.34 will be executed.

```

<div class="card-body">
  @if (!string.IsNullOrEmpty(ViewBag.Error))
  {
    <div class="alert alert-danger text-white fw-bold">@ViewBag.Error</div>
  }

  @if (!string.IsNullOrEmpty(ViewBag.Message))
  {
    <div class="alert alert-success text-white fw-bold">@ViewBag.Message</div>
  }

  @using (Html.BeginForm())
  {
    <div class="form-group">
      @Html.Label("Email")
      @Html.TextBox("email", "", new { @class = "form-control" })
    </div>

    <div class="form-group">
      @Html.Label("New Password")
      @Html.Password("newPassword", "", new { @class = "form-control" })
    </div>

    <div class="form-group">
      @Html.Label("Confirm Password")
      @Html.Password("confirmPassword", "", new { @class = "form-control" })
    </div>
    <br />
    <Center>
      <div class="d-grid">
        <button type="submit" class="btn btn-success">Reset Password</button>
      </div>
    </Center>
  }
}

```

Figure 5. 35 Implementation of input new password

5.2.3 Menu Page & Category List

On the page menu and Category List, the featured menu will be displayed and the various categories contained in the restaurant will also be displayed. For example code from the featured menu below. The result will be in the form of an image slider.

```

<div class="intro-slider-container">
  <div class="owl-carousel owl-simple owl-light owl-nav-inside" data-toggle="owl" data-owl-options='{ "nav": false}'>
    <div class="intro-slide" style="background-image: url('/Gambar/Bubur Tumpang Koyor.jpeg');">
      <div class="container intro-content">
        <h1 class="intro-title">Special <br> Menu</h1><!-- End .intro-title -->
        <a href="#" class="btn btn-primary">
          <span>Go To Porridge Category</span>
          <i class="icon-long-arrow-right"></i>
        </a>
      </div><!-- End .container intro-content -->
    </div><!-- End .intro-slide -->
    <div class="intro-slide" style="background-image: url('/Gambar/3656752147.jpg');">
      <div class="container intro-content">
        <h3 class="intro-subtitle">Dishes</h3><!-- End .h3 intro-subtitle -->
        <h1 class="intro-title">Find The Dish <br> You Want</h1><!-- End .intro-title -->
        <a href="#" class="btn btn-primary">
          <span>Go To Dishes Category</span>
          <i class="icon-long-arrow-right"></i>
        </a>
      </div><!-- End .container intro-content -->
    </div><!-- End .intro-slide -->
  </div><!-- End .owl-carousel owl-simple -->
  <span class="slider-loader text-white"></span><!-- End .slider-loader -->
</div><!-- End .intro-slider-container -->

```

Figure 5. 36 Implementation of show special menu

For Razor Page (cshtml) the list of menu categories is below, foreach is used to loop data that has been retrieved via the controller. Then also install the href so that the card from the category can pass data from the category_id to open a new page according to the category_id.

```

<div class="container">
  <div class="row justify-content-center mb-3">
    <div class="col-9">
      <center>
        <foreach (var data in Model)>
          <a href="@Url.Action("daftarmenu", "Home", new {id = data.category_id})">
            <div class="card-box">
              <div class="card-thumbnail">
                
              </div><br />
              <h3 class="product-title" style="text-align:center">@data.name_category</h3>
            </div>
          </a>
        </foreach>
      </center>
    </div>
  </div>
</div>

```

Figure 5. 37 Implementation of show category list

In the Controller, I use the Connection String for the database configuration. And use the select query method to filter the data.

The following is the function of add to cart. Still using the Connection string method for database configuration. The only thing that distinguishes between functions that display data and input data is the query. And there is some additional code for input as below.

```

0 references
public ActionResult cartlogic(FormCollection form)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();
        string quantity = form["quantity"];
        string iduser = form["iduser"];
        string itemid = form["itemid"];
        //var balik = "/Home/daftarmenu/";

        SqlConnection sqlconn = new SqlConnection(connectionString);
        string query = "insert into Cart_Item (ID, item_id, quantity, created_on, updated_on) values (@ID, @item_id, @quantity, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP)";
        SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
        sqlconn.Open();
        {
            sqlcomm.Parameters.AddWithValue("@ID", iduser);
            sqlcomm.Parameters.AddWithValue("@item_id", itemid);
            sqlcomm.Parameters.AddWithValue("@quantity", quantity);

            sqlcomm.ExecuteNonQuery();
            sqlconn.Close();
        }

        TempData["messagee"] = "Success";
        myConnection.Close();
        return Redirect("Index");
    }
    else
    {
        return RedirectToAction("Login", "Login");
    }
}

```

Figure 5. 40 Implementation of add to cart function

For the menu list function, this is a function to display data from the database, exactly the same method as select in the Category list. Using Connectionstring and sql queries.

```

0 references
public ActionResult daftarmenu(int id)
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = "select FORMAT( price, 'N') AS RP, item_id, name_item, description, price, stock, image_filename, category_id, estimate_minute from Items where category_id = {id}";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);
        List<UserModel> uc = new List<UserModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                UserModel uc10 = new UserModel();
                uc10.item_id = Convert.ToInt32(dr["item_id"]);
                uc10.estimate_minute = Convert.ToInt32(dr["estimate_minute"]);
                uc10.description = Convert.ToString(dr["description"]);
                uc10.name_item = Convert.ToString(dr["name_item"]);
                uc10.price = Convert.ToInt32(dr["price"]);
                uc10.stock = Convert.ToInt32(dr["stock"]);
                uc10.image_filename = Convert.ToString(dr["image_filename"]);
                uc10.RP = Convert.ToString(dr["RP"]);

                uc.Add(uc10);
            }
        }
        sqlconn.Close();
        return View(uc);
    }
    else
    {
        return RedirectToAction("Login", "Login");
    }
}

```

Figure 5. 41 Implementation of menu list function

5.2.5 Cart Page

The Cart page is used to store orders from customers before they checkout. Below is the code in the Views section which is used to display a list of foods, the total price of the food, and the number of orders that the customer ordered. And in this code, the customer can also edit the quantity and delete the order in the Cart.

```

@foreach (var data1 in ViewBag.Uci)
{
    if (data1.quantity == 0)
    {
        <tr>
        <td>
            <h3>Cart Is Empty</h3>
            <a href="/home/index" class="btn btn-outline-dark-2 btn-block mb-3"><span>Buy Something To Eat!</span></a>
        </td>
        </tr>
    }
    else
    {
        @foreach (var data in Model)
        {
            var jumlah = data.quantity * data.price;
            <tr>
            <td class="product-col">
                <div class="product">
                    <h3 class="product-title">
                        @data.name_item
                        <input type="hidden" name="cartid[@i]" value="@data.cart_id" />
                    </h3><!-- End .product-title -->
                </div><!-- End .product -->
            </td>
            <td class="price-col">
                Rp. <span class="hargabarang">@jumlah</span>
                <input type="hidden" class="harga" onchange="sum();" value="@data.price" />
                <input type="hidden" class="hasilharga" value="@jumlah" name="subtotal[@i]" />
            </td>
            <td class="quantity-col">
                <div class="form-outline">
                    <input type="number" name="qty" class="form-control quantity" value="@data.quantity" min="1" max="@data.stock" step="1" data-decimals="0" required>
                </div>
            </td>
            <td class="remove-col">
                <a class="open-exampleModaldelete btn-remove" data-cartid="@data.item_id" data-cartname="@data.name_item" data-toggle="modal" href="#exampleModaldelete">
                    <i class="icon-close"></i>
                </a>
            </td>
            </tr>
        }
    }
}

```

Figure 5. 42 Implementation of show data in cart page

Below is a Function to display data in the Cart page, the method is the same as the other controllers.

```

public ActionResult Cart()
{
    if (Session["Username"] != null)
    {
        string ID = Session["ID"].ToString();
        string Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = $"select FORMAT(price, 'N') AS RP, name_item, description, price, stock, category_id, " +
            $"Cart_Item.quantity, Cart_Item.ID, Cart_Item.cart_id, Items.item_id from Items JOIN Cart_Item on Items.item_id = Cart_Item.item_id where Cart_Item.ID = @ID";
        sqlconn.Open();

        using (SqlCommand cmd2 = new SqlCommand(sqlquery, sqlconn))
        {
            cmd2.Parameters.AddWithValue("@ID", ID);
            SqlDataAdapter sda = new SqlDataAdapter(cmd2);
            DataTable ds = new DataTable();
            sda.Fill(ds);
            List<UserModel> uc = new List<UserModel>();
            List<UserModel> uc1 = new List<UserModel>();
            {
                foreach (DataRow dr in ds.Rows)
                {
                    UserModel uc10 = new UserModel();
                    uc10.item_id = Convert.ToInt32(dr["item_id"]);
                    uc10.ID = Convert.ToInt32(dr["ID"]);
                    uc10.cart_id = Convert.ToInt32(dr["cart_id"]);
                    uc10.description = Convert.ToString(dr["description"]);
                    uc10.name_item = Convert.ToString(dr["name_item"]);
                    uc10.price = Convert.ToInt32(dr["price"]);
                    uc10.stock = Convert.ToInt32(dr["stock"]);
                    uc10.quantity = Convert.ToInt32(dr["quantity"]);
                    uc10.RP = Convert.ToString(dr["RP"]);
                    uc10.category_id = Convert.ToInt32(dr["category_id"]);

                    uc.Add(uc10);
                }
            }
        }
    }
}

```

Figure 5. 43 Implementation cart page function

```

String sqlquery1 = $"SELECT COUNT(quantity) AS quantity FROM Cart_Item WHERE ID = @ID";
using (SqlCommand cmd3 = new SqlCommand(sqlquery1, sqlconn))
{
    cmd3.Parameters.AddWithValue("@ID", ID);
    SqlDataAdapter sda1 = new SqlDataAdapter(cmd3);
    DataTable ds1 = new DataTable();
    sda1.Fill(ds1);

    {
        foreach (DataRow dr1 in ds1.Rows)
        {
            UserModel uc11 = new UserModel();
            uc11.quantity = Convert.ToInt32(dr1["quantity"]);

            uc1.Add(uc11);
        }
    }

    ViewBag.uc1 = uc1;
    sqlconn.Close();
    return View(uc);
}
else
{
    return RedirectToAction("Login", "Login");
}
}

```

Figure 5. 44 Implementation of cart page function (continue)

The function below is a function to edit the quantity of customer orders. Here I use Connectionstring as usual, update query, and use index to edit each row of Customers orders.

```

public ActionResult editcart(FormCollection form, List<string> cartid, List<string> qty, List<string> subtotal)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();

        SqlConnection sqlconn = new SqlConnection(connectionString);
        string varString = "";
        for (int i = 0; i < cartid.Count(); i++)
        {
            varString += $"update Cart_Item set subtotal = @subtotal{i}, quantity = @quantity{i}, updated_on = CURRENT_TIMESTAMP where cart_id = @cart_id{i}";
        }
        SqlCommand sqlcomm = new SqlCommand(varString, sqlconn);
        sqlconn.Open();
        {
            for (int i = 0; i < cartid.Count(); i++)
            {
                sqlcomm.Parameters.AddWithValue("@subtotal" + i, subtotal[i]);
                sqlcomm.Parameters.AddWithValue("@quantity" + i, qty[i]);
                sqlcomm.Parameters.AddWithValue("@cart_id" + i, cartid[i]);
            }

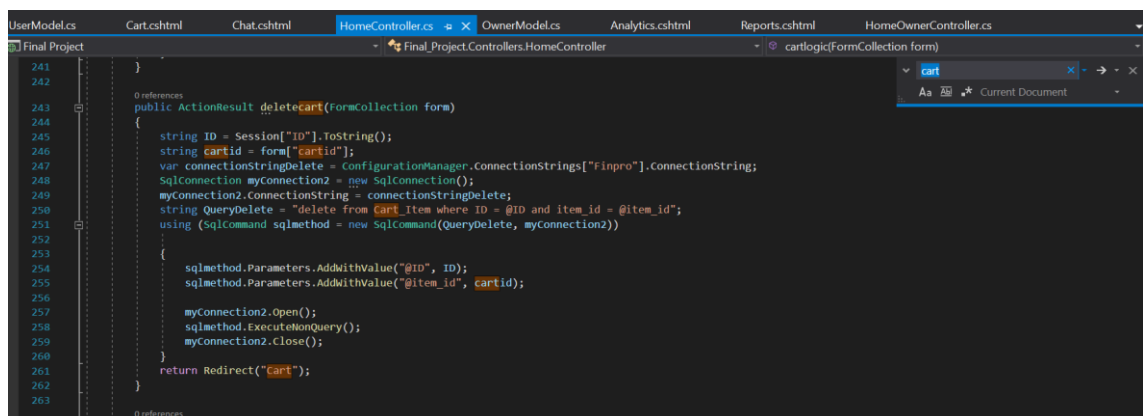
            sqlcomm.ExecuteNonQuery();
            sqlconn.Close();
        }

        myConnection.Close();
        return Redirect("Cart");
    }
    else
    {
        return RedirectToAction("Login", "Login");
    }
}

```

Figure 5. 45 Implementation of edit cart function

This function is used to delete Customers orders.



```

UserModel.cs Cart.cshtml Chat.cshtml HomeController.cs OwnerModel.cs Analytics.cshtml Reports.cshtml HomeOwnerController.cs
Final Project Final_Project.Controllers.HomeController cartlogic/FormCollection form
241 }
242 }
243 0 references
244 public ActionResult deletecart(FormCollection form)
245 {
246     string ID = Session["ID"].ToString();
247     string cartid = form["cartid"];
248     var connectionStringDelete = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
249     SqlConnection myConnection2 = new SqlConnection();
250     myConnection2.ConnectionString = connectionStringDelete;
251     string QueryDelete = "delete from Cart_Item where ID = @ID and item_id = @item_id";
252     using (SqlCommand sqlmethod = new SqlCommand(QueryDelete, myConnection2))
253     {
254         sqlmethod.Parameters.AddWithValue("@ID", ID);
255         sqlmethod.Parameters.AddWithValue("@item_id", cartid);
256     }
257     myConnection2.Open();
258     sqlmethod.ExecuteNonQuery();
259     myConnection2.Close();
260 }
261 return Redirect("Cart");
262 }
263 0 references

```

Figure 5. 46 Implementation of delete item in cart function

5.2.6 Account Page

On this page, simply use session to display the full name, role, and username. Does not require any function from the Controller.

```

<h2>Account Information</h2>
<br />
<div class="card shadow-lg mx-4 card-profile-bottom">
  <div class="card-body p-3">
    <div class="row gx-4">
      <div class="col-auto">
        <div class="avatar avatar-xl position-relative">
          
        </div>
      </div>
      <div class="col-auto my-auto">
        <div class="h-100">
          <h5 class="mb-1">
            @Session["Fullname"].ToString()
          </h5>
          <p class="mb-0 font-weight-bold text-sm">
            User
          </p>
        </div>
      </div>
    </div>
  </div>
</div>
<br />
<div class="card">
  <div class="card-body">
    <p class="text-uppercase text-sm"><b>User Information</b></p>
    <div class="row">
      <div class="col-md-6">
        <div class="form-group">
          <label for="example-text-input" class="form-control-label">Username</label>
          <input class="form-control" type="text" value="@Session["Username"].ToString()" readonly>
        </div>
      </div>
      <div class="col-md-6">
        <div class="form-group">
          <label for="example-text-input" class="form-control-label">Full Name</label>
          <input class="form-control" type="email" value="@Session["Fullname"].ToString()" readonly>
        </div>
      </div>
      <div class="col-md-6">
        <div class="form-group">
          <label for="example-text-input" class="form-control-label">Role</label>
          <input class="form-control" type="text" value="User" readonly>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figure 5. 47 Implementation of show account information

5.2.7 History Order Pages

The following is the Code from the History Order page for the Owner and Customer Service. Using the Connectionstring and Select Query methods. Then on the Razor Page (html) display it using the Datatable plugin.

```

0 references
public ActionResult History()
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = "select FORMAT( total_cost, 'M') AS RP, order_id, ID, serve, no_table, Username, merchant, order_status, created_on, updated_on from Order_log where order_status = 'DLV'";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);
        List<OwnerModel> uc = new List<OwnerModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                OwnerModel uc10 = new OwnerModel();
                uc10.order_id = Convert.ToInt32(dr["order_id"]);
                uc10.ID = Convert.ToInt32(dr["ID"]);
                uc10.serve = Convert.ToString(dr["serve"]);
                uc10.no_table = Convert.ToInt32(dr["no_table"]);
                uc10.RP = Convert.ToString(dr["RP"]);
                uc10.Username = Convert.ToString(dr["Username"]);
                uc10.merchant = Convert.ToString(dr["merchant"]);
                uc10.order_status = Convert.ToString(dr["order_status"]);
                uc10.created_on = Convert.ToDateTime(dr["created_on"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);
                uc.Add(uc10);
            }
        }
        sqlconn.Close();
        return View(uc);
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 48 Implementation of history order function

5.2.8 Current Order Pages

5.2.8.1 Current Order Owner & Current Order Customer Service

Using the same method with other controllers, this Current Order is for the Owner. This function is to display order data on the same day.

```

0 references
public ActionResult Order()
{
    if (Session["Username"] != null)
    {
        var products = GetProducts();
        var priorityProducts = products.OrderBy(p => p.updated_on);
        var model = priorityProducts;

        return View(model);
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

1 reference
private List<OwnerModel> GetProducts()
{
    List<OwnerModel> persons = new List<OwnerModel>();
    String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
    using (SqlConnection connection = new SqlConnection(Mainconn))
    {
        connection.Open();
        string query = "SELECT * FROM Order_log WHERE cast(created_on as DATE) = cast(GETDATE() as DATE)";
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            OwnerModel person = new OwnerModel();
            person.order_id = (int)reader["order_id"];
            person.no_table = (int)reader["no_table"];
            person.Username = (string)reader["Username"];
            person.serve = (string)reader["serve"];
            person.order_status = (string)reader["order_status"];
            person.created_on = (DateTime)reader["created_on"];
            person.updated_on = (DateTime)reader["updated_on"];

            persons.Add(person);
        }
    }
    return persons;
}

```

Figure 5. 49 Implementation of current order page owner function

Before entering the formula in the controller, the owner must confirm each item from the customer's order. They have to input some values like the number of chefs. Then submit the data, and the controller for calculating the estimated time will run like the picture after this.

```

<div class="modal-body">
  <div class="col-lg-8 col-12 mt-3">
    <div class="input-group">
      <div class="input-group-append">
        <span class="input-group-text bg-gray-200">Order Time</span>
      </div>
      <input class="form-control" type="text" value="@now.ToString("yyyy-MM-dd HH:mm:ss")" name="time" placeholder="Order Time" readonly />
    </div>
    <input hidden value="" id="cateid" name="cateid" />
    <input hidden value="" id="orderid" name="orderid" />
    <input hidden value="" id="itemids" name="orderitemid" />
    <input hidden value="" id="status" name="status" />
    <input hidden value="" id="estimasi" name="estimasi" />
    <div class="col-lg-8 col-12 mt-3">
      <div class="input-group">
        <div class="input-group-append">
          <span class="input-group-text bg-gray-200">Quantity</span>
        </div>
        <input class="form-control" type="text" value="" id="qty" name="qty" placeholder="Quantity" readonly />
      </div>
      <div class="col-lg-8 col-12 mt-3">
        <div class="input-group">
          <div class="input-group-append">
            <span class="input-group-text bg-gray-200">Number Of Cooks</span>
          </div>
          <input class="form-control" type="text" name="chef" placeholder="Chef" value="2" required />
        </div>
        <div class="col-lg-8 col-12 mt-3">
          <div class="input-group">
            <div class="input-group-append">
              <span class="input-group-text bg-gray-200">Table Number</span>
            </div>
            <input class="form-control" type="text" id="tabnum" value="" name="tabnumber" placeholder="Table Number" readonly />
          </div>
        </div>
      </div>
    </div>
  </div>
  <div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
    <input type="submit" class="btn btn-warning" value="Save">
  </div>
</div>

```

Figure 5. 50 Implementation of confirm item input for estimation time calculation

Below is a controller for calculating the time estimation formula, a derivative of the greedy algorithm. After the user orders food and the order goes to the owner. The owner will confirm the order, then confirm each item from the user's order to make a calculation of the estimated running time.


```

int stationqty = 0;
string method2 = @"select station_quantity from Categories where category_id = {cateid}";
SqlDataAdapter damethod1 = new SqlDataAdapter(method2, Mainconn);
DataTable dtmethod1 = new DataTable();
damethod1.Fill(dtmethod1);
foreach (DataRow dr1 in dtmethod1.Rows)
{
    stationqty = (int)dr1["station_quantity"];
}

int ongoingcount = 0;
string method3 = @"select count (*) as ongoingcount from Order_Item_Log left join Items on Order_Item_Log.item_id = Items.item_id left join Categories on Items.category_id = Categories.category_id
left join Order_Log on Order_Item_Log.order_id = Order_Log.order_id where Categories.category_id = " + cateid.ToString() + " and Order_Log.order_status <> 'OTW' and Order_Log.order_status <> 'DLV'";
SqlDataAdapter damethod2 = new SqlDataAdapter(method3, Mainconn);
DataTable dtmethod2 = new DataTable();
damethod2.Fill(dtmethod2);
foreach (DataRow dr1 in dtmethod2.Rows)
{
    ongoingcount = (int)dr1["ongoingcount"];
}

DateTime waktuPengiriman;
if (!DateTime.TryParseExact(time, "dd/MM/yyyy HH:mm:ss", null, System.Globalization.DateTimeStyles.None, out waktuPengiriman))
{
    //Jika waktu yang dimasukkan tidak valid, maka gunakan waktu saat ini
    waktuPengiriman = DateTime.Now;
    waktuPengiriman = DateTime.Now;
    //waktuPengiriman = Convert.ToDateTime(form["paydate"]);
}

int x = 0;
if (ongoingcount > stationqty)
{
    int diff = ongoingcount - stationqty;
    x = estimasi * diff; //Jumlah pesanan yang belum dibuat dikali estimasi dari waktu pembuatan masing masing item
}

waktuPengiriman = waktuPengiriman.AddMinutes((((double)qty / Chef) * estimasi + x);

```

Figure 5. 51 Implementation of calculation estimate time function

As for Customer service, it is still the same as Owner's, but the difference here is that the function for Customer service is only for orders whose status is already PAID and OTW.

```

@inherits
public ActionResult History()
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = "select FORMAT( total_cost, 'N') AS RP, order_id, ID, serve, no_table, Username, merchant, order_status, created_on, updated_on from Order_Log where order_status = 'DLV'";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);

        List<DapurModel> uc = new List<DapurModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                DapurModel uc10 = new DapurModel();
                uc10.order_id = Convert.ToInt32(dr["order_id"]);
                uc10.ID = Convert.ToInt32(dr["ID"]);
                uc10.serve = Convert.ToString(dr["serve"]);
                uc10.no_table = Convert.ToInt32(dr["no_table"]);
                uc10.RP = Convert.ToString(dr["RP"]);
                uc10.Username = Convert.ToString(dr["Username"]);
                uc10.merchant = Convert.ToString(dr["merchant"]);
                uc10.order_status = Convert.ToString(dr["order_status"]);
                uc10.created_on = Convert.ToDateTime(dr["created_on"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);
                uc.Add(uc10);
            }
        }
        sqlconn.Close();
        return View(uc);
    }
    else
    {
        return RedirectToAction("LoginDapur", "Login");
    }
}

```

Figure 5. 52 Implementation of current order page customer service function

5.2.8.2 Current Order Customers

```

public ActionResult History()
{
    if (Session["Username"] != null)
    {
        string ID = Session["ID"].ToString();
        string Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        string sqlquery = $"select FORMAT( total_cost, 'N') AS RP, order_id, ID, serve, no_table, Username, merchant, order_status, created_on, updated_on from Order_log where ID = @ID";
        sqlconn.Open();

        using (SqlCommand cmd2 = new SqlCommand(sqlquery, sqlconn))
        {
            cmd2.Parameters.AddWithValue("@ID", ID);
            SqlDataAdapter sda = new SqlDataAdapter(cmd2);
            DataTable ds = new DataTable();
            sda.Fill(ds);
            List<UserModel> uc = new List<UserModel>();
            foreach (DataRow dr in ds.Rows)
            {
                UserModel uc10 = new UserModel();
                uc10.order_id = Convert.ToInt32(dr["order_id"]);
                uc10.ID = Convert.ToInt32(dr["ID"]);
                uc10.serve = Convert.ToString(dr["serve"]);
                uc10.no_table = Convert.ToInt32(dr["no_table"]);
                uc10.RP = Convert.ToString(dr["RP"]);
                uc10.Username = Convert.ToString(dr["Username"]);
                uc10.merchant = Convert.ToString(dr["merchant"]);
                uc10.order_status = Convert.ToString(dr["order_status"]);
                uc10.created_on = Convert.ToDateTime(dr["created_on"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);

                uc.Add(uc10);
            }
            sqlconn.Close();
            return View(uc);
        }
    }
    else
    {
        return RedirectToAction("Login", "Login");
    }
}

```

Figure 5. 53 Impementation of current order customer function

5.2.9 Payment

For the payment page, it contains the destination account for the restaurant and a function for uploading proof of transfer. The qr image is displayed according to the payment method chosen by the customer.

```

@Foreach (var data in Model)
{
    if (data.merchant == "bank")
    {
        <h2 class="section-header">WAITING PAYMENT</h2>
        <p>OCBC BANK TRANSFER</p>
        <center>
        
        </center>
        <p>ACCOUNT NUMBER : <h6>@data.account_number</h6></p>
        <p>SEND TRANSFER RECEIPT HERE:</p>
        <button class="btn btn-primary btn-sm" data-toggle="modal" data-target="#upfoto">
        Upload Approval Photo
        </button>
        <br />
        <p>YOUR PAYMENT WILL BE PROCESSED SHORTLY...</p>
        <p>WAIT UNTIL THE CONFIRMATION !!!</p>
        <br />
        <p>SERVE: <h6>@data.serve</h6></p>
        <p>YOUR TABLE NUMBER: <h6>@data.no_table</h6></p>
        <p>MERCHANT: <h6>@data.merchant</h6></p>
        <p> YOUR ORDER TOTAL: <h6>Rp. @data.RP</h6></p>
    }
}

```

Figure 5. 54 Implementation of show payment method

The Controller section is still the same as before. Can be seen below using Connstring with Select query.

```

0 references
public ActionResult cartpay()
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();

        int Method = 0;
        string method = "SELECT * FROM Order_log";
        SqlDataAdapter damethod = new SqlDataAdapter(method, myConnection);
        DataTable dtmethod = new DataTable();
        damethod.Fill(dtmethod);
        foreach (DataRow dr in dtmethod.Rows)
        {
            Method = (int)dr["order_id"];
        }

        string ID = Session["ID"].ToString();
        string sqlquery = $"select order_id, ID, FORMAT(total_cost, 'N') AS RP, serve, no_table, Username, " +
            $"Payment_method.account_number, Payment_method.phone_num, Payment_method.img_file, Payment_method.merchant " +
            $"From Order_log JOIN Payment_method on Order_log.merchant = Payment_method.merchant where ID = @ID and order_id = " + Method + """;

        using (SqlCommand cmd2 = new SqlCommand(sqlquery, myConnection))
        {
            cmd2.Parameters.AddWithValue("@ID", ID);
            SqlDataAdapter sda = new SqlDataAdapter(cmd2);
            DataTable ds = new DataTable();
            sda.Fill(ds);
            List<UserModel> uc = new List<UserModel>();
            List<UserModel> uc1 = new List<UserModel>();
        }
    }
}

```

Figure 5. 55 Implementation of payment function

Here for the input function of proof of transfer. Using update queries and Connectionstring.

```

0 references
public ActionResult Uploadgambar(IEnumerable<HttpPostedFileBase> imageupload)
{
    if (Session["Username"] != null)
    {
        List<UserModel> jc = new List<UserModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();

        string ID = Session["ID"].ToString();
        int Method = 0;
        string method = "SELECT * FROM Order_log";
        SqlDataAdapter damethod = new SqlDataAdapter(method, myConnection);
        DataTable dtmethod = new DataTable();
        damethod.Fill(dtmethod);
        foreach (DataRow dr in dtmethod.Rows)
        {
            Method = (int)dr["order_id"];
        }

        foreach (var ImageSave in imageupload)
        {
            string imagemolded = Path.GetFileName(ImageSave.FileName);
            string filePathmolded = Path.Combine(Server.MapPath("/Bukti_transfer/"), imagemolded);
            ImageSave.SaveAs(filePathmolded);
            Stream strm = ImageSave.InputStream;
            var targetFile = filePathmolded;

            ReduceImageSize(0.5, strm, targetFile);

            string Query2 = "UPDATE Order_log set bukti_tf = @bukti_tf, updated_on = CURRENT_TIMESTAMP where ID = " + ID + " and order_id = " + Method + """;
            using (SqlCommand sqlpict = new SqlCommand(Query2, myConnection))
            {
                sqlpict.Parameters.AddWithValue("@bukti_tf", imagemolded);
                sqlpict.ExecuteNonQuery();
            }
        }

        TempData["message"] = "succes";
        myConnection.Close();
        return Redirect("History");
    }
}

```

Figure 5. 56 Implementation of upload approval image function

5.2.10 Payment Pages

For the Owner's Payment page. Used to display Owner / Restaurant account information. There is also a transfer method update feature.

```

<div class="container">
  <h5>Current bank transfer</h5>
  <Foreach (var data in Model)
  {
    
    <p>Account number: @data.account_number</p>
    <p>Confirmation WA: @data.phone_num</p><br />
  }
</div>
<br />
<div class="container">
  <h5>Update bank transfer</h5>

  <form method="post" action="/HomeOwner/editbanktransfer" enctype="multipart/form-data">
    <div class="input-group">
      <label class="input-group-text">Account number</label>
      <input class="form-control" type="text" name="accountNumber" placeholder="Account Number" required />
    </div>
    <div class="input-group">
      <label class="input-group-text">Confirmation WA</label>
      <input class="form-control" type="text" name="phoneNum" placeholder="Confirmation Number" required />
    </div>
    <div class="input-group">
      <label class="input-group-text">Image</label>
      <input class="form-control" type="file" name="imageupload" required />
    </div>
    <br />
    <input type="submit" Class="btn btn-primary" value="Update bank transfer" />
  </form>
</div>
<br />

```

Figure 5. 57 Implementation of showing payment method and input for update payment

To create a Payment page we must first create a controller function and in the controller we will select a database query like usually and we will fetch data from the database to display to cshtml and we will get the data via models.

```

0 references
public ActionResult Payment()
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = "select * from Payment_method";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);
        List<OwnerModel> uc = new List<OwnerModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                OwnerModel uc10 = new OwnerModel();
                uc10.account_number = Convert.ToString(dr["account_number"]);
                uc10.payment_id = Convert.ToInt32(dr["payment_id"]);
                uc10.phone_num = Convert.ToString(dr["phone_num"]);
                uc10.img_file = Convert.ToString(dr["img_file"]);
                uc10.merchant = Convert.ToString(dr["merchant"]);
                uc10.created_on = Convert.ToDateTime(dr["created_on"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);

                uc.Add(uc10);
            }
        }
        sqlconn.Close();
        return View(uc);
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 58 Implementation of show payment method function

For the code below, it is used to edit the account number and qr code of the transaction destination. Used when there is a change from the owner.

```

0 references
public ActionResult editbanktransfer(FormCollection form, IEnumerable<HttpPostedFileBase> imageupload)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection(connectionString);
        myConnection.Open();
        string accountNumber = form["accountNumber"];
        string phoneNum = form["phoneNum"];

        foreach (var ImageSave in imageupload)
        {
            string imagemolded = Path.GetFileName(ImageSave.FileName);
            string filePathMolded = Path.Combine(Server.MapPath("~/Qr_Bank/"), imagemolded);
            ImageSave.SaveAs(filePathMolded);

            string mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(mainconn);
            string query = "update BankTransfer set account_number = @account_number, phone_num = @phone_num, img_file = @img_file, created_on = CURRENT_TIMESTAMP, updated_on = CURRENT_TIMESTAMP";
            SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
            sqlconn.Open();
            {
                sqlcomm.Parameters.AddWithValue("@account_number", accountNumber);
                sqlcomm.Parameters.AddWithValue("@phone_num", phoneNum);
                sqlcomm.Parameters.AddWithValue("@img_file", imagemolded);

                sqlcomm.ExecuteNonQuery();
                sqlconn.Close();
            }
        }

        TempData["messageee"] = "Success";
        myConnection.Close();
        return RedirectToAction("Payment");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 59 Implementation of update new payment function

5.2.11 Categories Page

The menu category is an element that is quite important for dividing the types of food and drinks. To create the page, as usual, start by pulling data through the controller using the Connection string and select query to display the data. Then for the Razor Page (cshtml) you only need to display data using a foreach loop and the Datable plugin.

```

0 references
public ActionResult Categories()
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = "select * from Categories";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);
        List<OwnerModel> uc = new List<OwnerModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                OwnerModel uc10 = new OwnerModel();
                uc10.name_category = Convert.ToString(dr["name_category"]);
                uc10.category_id = Convert.ToInt32(dr["category_id"]);
                uc10.user_id = Convert.ToInt32(dr["user_id"]);
                uc10.img_category = Convert.ToString(dr["img_category"]);
                uc10.created_on = Convert.ToDateTime(dr["created_on"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);

                uc.Add(uc10);
            }
        }
        sqlconn.Close();
        return View(uc);
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 60 Implementation of show categories function

The code below is useful for adding a list of categories that will be displayed on the page and on the Customer page. By using Insert queries.

```

[HttpPost]
0 references
public ActionResult addcategories(FormCollection form, IEnumerable<HttpPostedFileBase> imageupload)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection(connectionString);
        myConnection.ConnectionString = connectionString;
        myConnection.Open();

        string category = form["category"];
        string userid = form["userid"];
        foreach (var ImageSave in imageupload)
        {
            string imagemolded = Path.GetFileName(ImageSave.FileName);
            string filePathmolded = Path.Combine(Server.MapPath("~/Logo_dishes and drink/"), imagemolded);
            ImageSave.SaveAs(filePathmolded);

            string mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(mainconn);
            string query = "insert into Categories (name_category, created_on, updated_on, user_id, img_category) values (@name_category, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, @user_id, @img_category)";
            SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
            sqlconn.Open();
            {
                sqlcomm.Parameters.AddWithValue("@name_category", category);
                sqlcomm.Parameters.AddWithValue("@user_id", userid);
                sqlcomm.Parameters.AddWithValue("@img_category", imagemolded);

                sqlcomm.ExecuteNonQuery();
                sqlconn.Close();
            }
        }
        TempData["message"] = "Success";
        myConnection.Close();
        return Redirect("Categories");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 61 Implementation of input new categories function

As for this one code, it is used to edit the category name or image of the category using an update query.

```

[HttpPost]
0 references
public ActionResult editcategories(FormCollection form, IEnumerable<HttpPostedFileBase> imageupload)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection(connectionString);
        myConnection.ConnectionString = connectionString;
        myConnection.Open();

        string cateid = form["cateid"];
        string category = form["category"];
        foreach (var ImageSave in imageupload)
        {
            string imagemolded = Path.GetFileName(ImageSave.FileName);
            string filePathmolded = Path.Combine(Server.MapPath("~/Logo_dishes and drink/"), imagemolded);
            ImageSave.SaveAs(filePathmolded);

            string mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(mainconn);
            string query = "update Categories set name_category = @name_category, img_category = @img_category, updated_on = @updated_on where category_id = '' + cateid + ''";
            SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
            sqlconn.Open();
            {
                sqlcomm.Parameters.AddWithValue("@name_category", category);
                sqlcomm.Parameters.AddWithValue("@img_category", imagemolded);

                sqlcomm.ExecuteNonQuery();
                sqlconn.Close();
            }
        }
        TempData["message"] = "Success";
        myConnection.Close();
        return Redirect("Categories");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 62 Implementation of edit categories function

5.2.12 Menulist Page

The menu list is an element that is important enough to display what menu is sold at the restaurant. To create the page, as usual, start by pulling data through the controller using the Connection string and select query to display the

data. Then for the Razor Page (cshtml) you only need to display data using a foreach loop and the Datatable plugin.

```

0 references
public ActionResult MenuList()
{
    if (Session["Username"] != null)
    {
        String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn = new SqlConnection(Mainconn);
        String sqlquery = " select FORMAT( price, 'N') AS RP, item_id, name_item, description, price, stock, image_filename, estimate_minute, " +
            "Categories.name_category, Categories.category_id, Items.user_id, Items.updated_on from Items join Categories on Items.category_id = Categories.category_id";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
        sqlconn.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);
        List<OwnerModel> uc = new List<OwnerModel>();
        {
            foreach (DataRow dr in ds.Rows)
            {
                OwnerModel uc10 = new OwnerModel();
                uc10.item_id = Convert.ToInt32(dr["item_id"]);
                uc10.price = Convert.ToInt32(dr["price"]);
                uc10.estimate_minute = Convert.ToInt32(dr["estimate_minute"]);
                uc10.category_id = Convert.ToInt32(dr["category_id"]);
                uc10.description = Convert.ToString(dr["description"]);
                uc10.name_item = Convert.ToString(dr["name_item"]);
                uc10.name_category = Convert.ToString(dr["name_category"]);
                uc10.stock = Convert.ToInt32(dr["stock"]);
                uc10.image_filename = Convert.ToString(dr["image_filename"]);
                uc10.RP = Convert.ToString(dr["RP"]);
                uc10.updated_on = Convert.ToDateTime(dr["updated_on"]);

                uc.Add(uc10);
            }
        }
    }
}

```

Figure 5. 63 Implementation of show menulist function

The code below is useful for adding a Menu list that will be displayed on that page and on the Customer page. By using Insert queries.

```

[HttpPost]
0 references
public ActionResult addmenu(FormCollection form, IEnumerable<HttpPostedFileBase> imageupload)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionstring = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myconnection = new SqlConnection();
        myConnection.connectionString = connectionstring;
        myConnection.Open();
        string menuname = form["menuname"];
        string desc = form["desc"];
        string price = form["price"];
        string stock = form["stock"];
        string category = form["category"];
        string time = form["time"];

        foreach (var ImageSave in imageupload)
        {
            string imagemolded = Path.GetFileName(ImageSave.FileName);
            string filepathmolded = Path.Combine(Server.MapPath("/Items_Images/"), imagemolded);
            ImageSave.SaveAs(filepathmolded);

            string mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
            SqlConnection sqlconn = new SqlConnection(mainconn);
            string query = "insert into Items(name_item, description, price, stock, image_filename, category_id, created_on, updated_on, estimate_minute) " +
                "values(@name_item, @description, @price, @stock, @image_filename, @category_id, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, @estimate_minute)";
            SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
            sqlconn.Open();
            {
                sqlcomm.Parameters.AddWithValue("@name_item", menuname);
                sqlcomm.Parameters.AddWithValue("@description", desc);
                sqlcomm.Parameters.AddWithValue("@price", price);
                sqlcomm.Parameters.AddWithValue("@stock", stock);
                sqlcomm.Parameters.AddWithValue("@image_filename", imagemolded);
                sqlcomm.Parameters.AddWithValue("@category_id", category);
                sqlcomm.Parameters.AddWithValue("@estimate_minute", time);

                sqlcomm.ExecuteNonQuery();
                sqlconn.Close();
            }
        }

        TempData["message"] = "Success";
        myConnection.Close();
        return Redirect("~/MenuList");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}

```

Figure 5. 64 Implementation of add new menu function

As for this one code, it is used to edit stock and prices using an update query.

```
[HttpPost]
0 references
public ActionResult editstock(FormCollection form)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();
        string itemid = form["itemid"];
        string stock = form["stock"];

        SqlConnection sqlconn = new SqlConnection(connectionString);
        string query = "update Items set stock = @stock, updated_on = CURRENT_TIMESTAMP where item_id = '" + itemid + "'";
        SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
        sqlconn.Open();
        {
            sqlcomm.Parameters.AddWithValue("@stock", stock);

            sqlcomm.ExecuteNonQuery();
            sqlconn.Close();
        }

        TempData["messagee"] = "Success";
        myConnection.Close();
        return Redirect("MenuList");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}
```

Figure 5. 65 Implementation of edit stock function

```
[HttpPost]
0 references
public ActionResult editprice(FormCollection form)
{
    if (Session["Username"] != null)
    {
        List<OwnerModel> jc = new List<OwnerModel>();
        var connectionString = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection myConnection = new SqlConnection();
        myConnection.ConnectionString = connectionString;
        myConnection.Open();
        string itemid = form["itemid"];
        string price = form["price"];

        SqlConnection sqlconn = new SqlConnection(connectionString);
        string query = "update Items set price = @price, updated_on = CURRENT_TIMESTAMP where item_id = '" + itemid + "'";
        SqlCommand sqlcomm = new SqlCommand(query, sqlconn);
        sqlconn.Open();
        {
            sqlcomm.Parameters.AddWithValue("@price", price);

            sqlcomm.ExecuteNonQuery();
            sqlconn.Close();
        }

        TempData["messageee"] = "Success";
        myConnection.Close();
        return Redirect("MenuList");
    }
    else
    {
        return RedirectToAction("LoginOwner", "Login");
    }
}
```

Figure 5. 66 Implementation of edit price function

Below is bubble sort which is one of the features that uses the sorting algorithm on this menu list page.

```

<div class="col-3">
  <select id="my-select" class="form-control" aria-label=".form-select-lg example">
    <option selected disabled>Choose for filter product</option>
    <option value="option1">Most Expensive</option>
    <option value="option2">Cheapest</option>
    <option value="option3">lots of stock</option>
    <option value="option4">little stock</option>
    <option value="option5">Fast Estimated Time</option>
    <option value="option6">Slow Estimated Time</option>
  </select>
</div>
<br />

```

Figure 5. 67 Implementation of select dropdown value

```

$(document).ready(function () {
  // ketika nilai elemen select berubah
  $("#my-select").change(function () {
    // ambil nilai dari elemen select
    var selectedValue = $(this).val();

    // periksa nilai opsi yang dipilih
    if (selectedValue === "option1") {
      var $wrap = $(''.outer');
      $wrap.find(''.child').sort(function (b, a) {
        return + a.getAttribute('data-percentage') -
          + b.getAttribute('data-percentage');
      })
      .appendTo($wrap);
      down.innerHTML = "Elements sorted";
      // lakukan sesuatu untuk opsi 1
      console.log("Ops 1 dipilih");
    }
    else if (selectedValue === "option2") {
      var $wrap = $(''.outer');
      $wrap.find(''.child).sort(function (a, b) {
        return + a.getAttribute('data-percentage') -
          + b.getAttribute('data-percentage');
      })
      .appendTo($wrap);
      down.innerHTML = "Elements sorted";
      // lakukan sesuatu untuk opsi 2
      console.log("Ops 2 dipilih");
    }
  });
});

```

Figure 5. 68 Implementation of bubble sort method

5.2.13 Reports Page

For this page, before entering the Controller, a calculation is needed to calculate the income per day and per month. As the example below is a view that has just been created so that it can display calculation results using the count query.

	date	net_income
1	2023-03-13	24000
2	2023-03-14	113000
3	2023-03-15	45000
4	2023-03-16	77000
5	2023-03-29	16000
6	2023-04-03	24000

Figure 5. 69 List of income data in database

After completing calculating income per month and per day, continue to create functions in the controller. Using the previous method that has also been used frequently.

```

String Mainconn = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
SqlConnection sqlconn = new SqlConnection(Mainconn);
String sqlquery = "select date, FORMAT(net_income, 'N') AS RP from Order_Income_Day";
SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn);
sqlconn.Open();
SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
DataTable ds = new DataTable();
sda.Fill(ds);

List<OwnerModel> uc = new List<OwnerModel>();
List<OwnerModel> uc1 = new List<OwnerModel>();
{
    foreach (DataRow dr in ds.Rows)
    {
        OwnerModel uc10 = new OwnerModel();
        uc10.RP = Convert.ToString(dr["RP"]);
        uc10.date = Convert.ToString(dr["date"]);

        uc.Add(uc10);
    }
}

String sqlquery1 = $"select month, FORMAT(net_income, 'N') AS RP from Order_Income_Month";
using (SqlCommand cmd3 = new SqlCommand(sqlquery1, sqlconn))
{
    SqlDataAdapter sda1 = new SqlDataAdapter(cmd3);
    DataTable ds1 = new DataTable();
    sda1.Fill(ds1);

    {
        foreach (DataRow dr1 in ds1.Rows)
        {
            OwnerModel uc11 = new OwnerModel();
            uc11.RP = Convert.ToString(dr1["RP"]);
            uc11.month = Convert.ToString(dr1["month"]);

            uc1.Add(uc11);
        }
    }
}
ViewBag.uc1 = uc1;
sqlconn.Close();
return View(uc);

```

Figure 5. 70 Implementation of report page function

Continue to create views, here the display method is the same using the foreach and datatable plugins. The only difference is between the plugins installed on other pages, on this report page there is a calendar range filter connected to the datatable plugin as shown below.

```

<script>
$(document).ready(function () {
$.fn.dataTable.ext.search.push(function (settings, data, dataIndex) {
var startDate = $('#start-date').val();
var endDate = $('#end-date').val();
var date = moment(data[0], "DD/MM/YYYY"); // Column 0 is the date column

// If the start date and end date inputs are empty, return true to include all rows
if (startDate === '' && endDate === '') {
return true;
}

// If either the start date or end date input is empty, or if there is other search criteria entered, don't apply the date range filter
if (startDate === '' || endDate === '' || settings.oPreviousSearch.sSearch !== '') {
return true;
}

startDate = moment(startDate, "YYYY-MM-DD");
endDate = moment(endDate, "YYYY-MM-DD");

// Check if the row's date value is between the start date and end date
return (date >= startDate && date <= endDate);
});

// Initialize DataTable
var table = $('#example99').DataTable({
scrollX: true,
fixedColumns: {
leftColumns: 1,
},
ordering: false,
paging: true,
});

// Call the filter function when the start date or end date inputs are changed
$('#start-date, #end-date').on('change', function () {
table.draw();
});
});
</script>

```

Figure 5. 71 Implementation of date range inside datatable plugin

5.2.14 Chat Pages

Enter the chat page. To create this page, the first step is to install the SignalR library on asp.net mvc. Install-Package Microsoft.AspNet.SignalR. Type the command in the NuGet Package manager console. Then the library will be automatically installed.

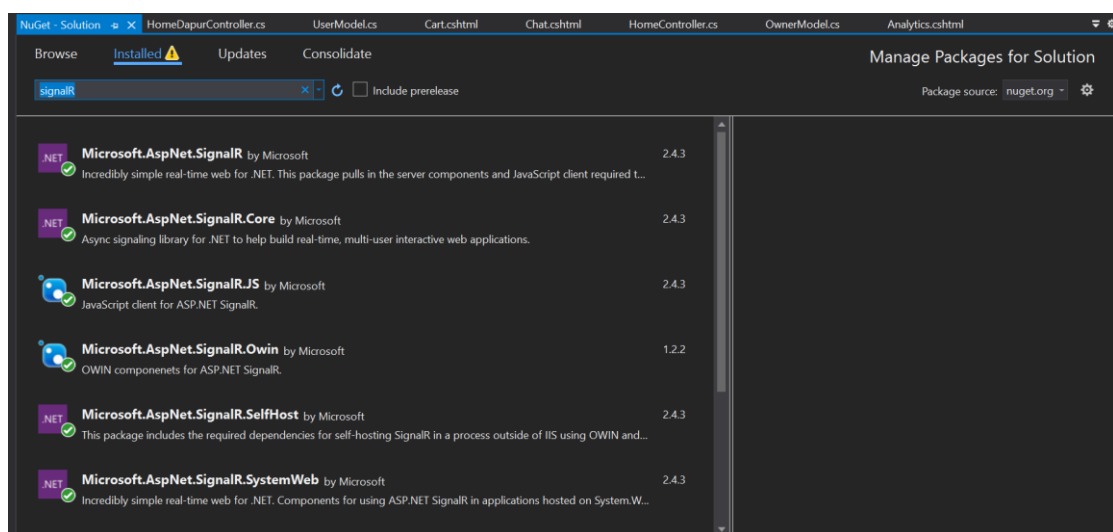


Figure 5. 72 List of library for making chat feature

After finished the installation of the library, In the SignalR Chat project folder, create a Hubs folder. A *hub* is a class that serves as a high-level pipeline that handles client-server communication. The ChatHub class inherits from the SignalR Hub class. The Hub class manages connections, groups, and messaging.

The SendMessage method can be called by a connected client to send a message to all clients. JavaScript client code that calls the method is shown later in the tutorial. SignalR code is asynchronous to provide maximum scalability. In the Hubs folder, create the ChatHub class with the following code:

```
public class ChatHub : Hub
{
    private Final_ProjectEntitiesChat db = new Final_ProjectEntitiesChat();
    0 references
    public void Send(string name, string message, string receiver)
    {
        Clients.All.addNewMessageToPage(name, message, receiver);

        var Date = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
        db.Database.ExecuteSqlCommand("insert into Online_Chat (Message, Sender, Receiver, Created_Date) values ('" + message + "','" + name + "','" + receiver + "','" + Date + "')");
        db.SaveChanges();
    }
}
```

Figure 5. 73 Implementation of ChatHub function

After that, create a new class and rename it to Startup.cs. Then follow the code like this:

```
using Microsoft.Owin;
using Owin;
using System;
using System.Threading.Tasks;

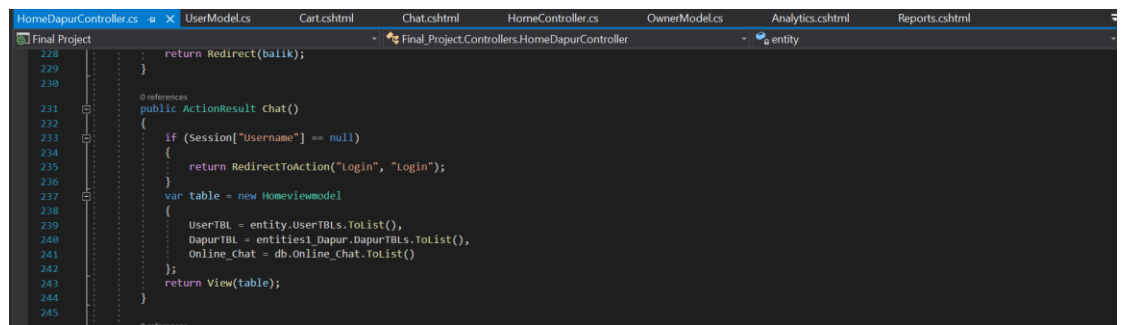
[assembly: OwinStartup("ProductionConfiguration", typeof(Final_Project.Startup))]

namespace Final_Project
{
    {
        1 reference
        public class Startup
        {
            {
                0 references
                public void Configuration(IAppBuilder app)
                {
                    app.MapSignalR();
                }
            }
        }
    }
}
```

Figure 5. 74 Implementation of startup function

5.2.14.1 Chat Customer Service

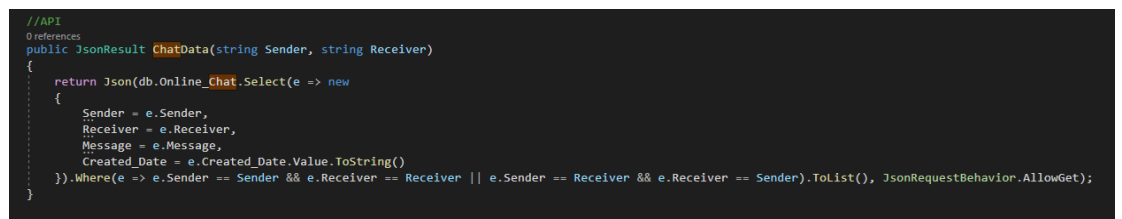
After finishing preparing everything needed for the signalR library, continue to enter the Controller. Here the chat function uses the entity framework method for configuration to the database. Then declare the necessary tables to display data such as Customer names and Kitchen Names.



```

228         return Redirect(balk);
229     }
230
231     public ActionResult Chat()
232     {
233         if (Session["Username"] == null)
234         {
235             return RedirectToAction("Login", "Login");
236         }
237         var table = new HomeViewModel
238         {
239             UserTBL = entity.UserTBLs.ToList(),
240             DapurTBL = entities1.Dapur.DapurTBLs.ToList(),
241             Online_Chat = db.Online_Chat.ToList()
242         };
243         return View(table);
244     }
245
  
```

Figure 5. 75 Implementation of chat customer service function



```

//API
0 references
public JsonResult ChatData(string Sender, string Receiver)
{
    return Json(db.Online_Chat.Select(e => new
    {
        Sender = e.Sender,
        Receiver = e.Receiver,
        Message = e.Message,
        Created_Date = e.Created_Date.Value.ToString()
    }).Where(e => e.Sender == Sender && e.Receiver == Receiver || e.Sender == Receiver && e.Receiver == Sender).ToList(), JsonRequestBehavior.AllowGet);
}
  
```

Figure 5. 76 Implementation of chatdata customer service function

In Views, we first create a view for the chat, along with the id and name which will later be passed to ChatHub.

```

<div class="card-header p-0 position-relative mb-n4 mx-3 z-index-2">
  <div class="row bg-gradient-primary shadow-primary border-radius-lg pt-2 pb-2">
    <div class="col-sm-12">
      <div class="row align-items-center">
        <div class="col-2 col-sm-1">
          <img src="" alt="bruce" class="profilepic2 avatar-sm position-relative profilepic_image" id="Photos" hidden>
        </div>
        <div class="col-8 col-sm-10">
          <div class="h-100">
            <h5 id="Penerima" class="mb-1 font-weight-bolder text-white">
            </h5>
          </div>
        </div>
        <div class="col-2 col-sm-1">
          <div class="avatar avatar-sm position-relative" id="remove" style="cursor:pointer">
            <i class="material-icons opacity-10">close</i>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="card" style="border:1px solid grey;background-image: url('../Img/ChatBG.jpg'); background-size:450px; filter: brightness(95%)">
  <div class="card-body pt-0 mt-5" id="chat" style="height:430px; overflow-y:scroll;">
    <input type="hidden" id="Sender" />
    <input type="hidden" id="Receiver" />
    <p id="discussion"></p>
  </div>
</div>
<div class="card-footer p-0 position-relative mt-n4 mx-3 z-index-2">
  <div class="row bg-white border-radius-lg pt-2" style="border:1px solid grey">
    <div class="col-sm-9">
      <div class="input-group input-group-dynamic">
        <input type="text" class="form-control" id="message" name="message" placeholder="Type your message here..." required>
      </div>
    </div>
    <div class="col-sm-1 text-end">
      <div class="input-group-btn input-group-dynamic">
        <input class="btn btn-success" type="submit" id="sendMessage" value="Send">
      </div>
    </div>
  </div>
</div>
</div>

```

Figure 5. 77 Implementation of show chat in views

In the script section that must be added Chathub, which we have made a class from the start.

```

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="~/Scripts/jquery.signalR-2.4.3.min.js"></script>
<script src="~/signalr/hubs"></script>

```

Figure 5. 78 Implementation of declare the library

The code below is used to display chats that have been sent along with the time they were sent, they take the value from the ChatData function which comes from the Home controller.

```

$.ajax({
  url: ".../Home/ChatData?Sender=" + Username + "&Receiver=" + Session["Username"].ToString(),
  type: "GET",
  contentType: "application/json;charset=utf-8",
  dataType: "json",
  success: function (data) {
    $.each(data, function (key, value) {
      if (value.Sender == Session["Username"].ToString()) {
        $('#discussion').append('<div class="card mb-2 form-label chat" style="text-align:right; width:auto; padding-right:10px; padding-top:10px"><label>'
          + value.Message + ' </label><span class="form-label text-end" style="font-size:11px"> + value.Created_Date + ' </span></div>');
      } else {
        $('#discussion').append('<div class="card mb-2 form-label chat" style="text-align:left; width:auto; padding-left:10px; padding-top:10px; background-color:#cffff0"><label>'
          + value.Message + ' </label><span class="form-label" style="font-size:11px"> + value.Created_Date + ' </span></div>');
      }
    });
  },
  error: function (textStatus, status) {
    console.log(textStatus);
    console.log(status);
  }
});

```

Figure 5. 79 Implementation of show the chat function

Then, the script below is used to connect to ChatHub. There is a console to validate whether the connection is connected or not.

```
$(function () {
    // Reference the auto-generated proxy for the hub.
    var chatHub = $.connection.chatHub;

    // Create a function that the hub can call back to display messages.
    chatHub.client.addNewMessageToPage = function (name, message) {
        if (name == "Session[Username].ToString()") {
            // Add the message to the page.
            $('#discussion').append('<div class="card mb-2 form-label .chat" style="text-align:right; width:auto; padding-right:10px; padding-top:10px"><label>'
                + htmlEncode(message) + ' </label><span class="form-label text-end" style="font-size:11px"> + new Date().toDateString() + ' </span></div>');
        }
        else {
            // Add the message to the page.
            $('#discussion').append('<div class="card mb-2 form-label .chat" style="text-align:left; width:auto; padding-left:10px; padding-top:10px; background-color:#ccff00"><label>'
                + htmlEncode(message) + ' </label><span class="form-label" style="font-size:11px"> + new Date().toDateString() + ' </span></div>');
        }
    };

    // Get the user name and store it to prepend to messages.
    $('#sender').val("Session[Username].ToString()");

    // Set initial focus to message input box.
    $('#message').focus();

    // Start the connection.
    $.connection.hub.start().done(function () {
        console.log("SignalR connection started");

        $('#sendMessage').click(function () {
            // Call the send method on the hub.
            chatHub.server.send($('#sender').val(), $('#message').val(), $('#receiver').val());

            // Clear text box and reset focus for next comment.
            $('#message').val('').focus();
        });
    }).fail(function () {
        console.error("SignalR connection failed");
    });
});
```

Figure 5. 80 Implementation of connect the chathub function

5.2.14.2 Chat Customers

Just like the customer service chat. Here the chat function uses the entity framework method also for configuration to the database. Then declare the necessary tables to display data data.

```
UserModel.cs Cart.cshtml Chat.cshtml HomeController.cs OwnerModel.cs Analytics.cshtml Reports.cshtml HomeOwnerController.cs
Final Project Final Project.Controllers.HomeController ChatData(string Sender, string Receiver)
764 }
765 }
766 }
767 0 references
768 public ActionResult Chat()
769 {
770     if (Session["Username"] == null)
771     {
772         return RedirectToAction("Login", "Login");
773     }
774     var table = new Homeviewmodel
775     {
776         UserTBL = entity.UserTBLs.ToList(),
777         DapurTBL = entities1.Dapur.DapurTBLs.ToList(),
778         Online_Chat = db.Online_Chat.ToList()
779     };
780     return View(table);
781 }
```

Figure 5. 81 Implementation of chat customer function

```
//API
References
public JsonResult ChatData(string Sender, string Receiver)
{
    return Json(db.Online_Chat.Select(e => new
    {
        Sender = e.Sender,
        Receiver = e.Receiver,
        Message = e.Message,
        Created_Date = e.Created_Date.Value.ToString()
    })).Where(e => e.Sender == Sender && e.Receiver == Receiver || e.Sender == Receiver && e.Receiver == Sender).ToList(), JsonRequestBehavior.AllowGet);
}
```

Figure 5. 82 Implementation of chatdata customer function

Just like chat in Customer Service. In Views, we first create a view for the chat, along with the id and name which will later be passed to ChatHub.

```
<div class="card-header p-0 position-relative mb-n4 mx-3 z-index-2">
  <div class="row bg-gradient-primary shadow-primary border-radius-lg pt-2 pb-2">
    <div class="col-sm-12">
      <div class="row align-items-center">
        <div class="col-2 col-sm-1">
          <img src="" alt="bruce" class="profilepic2 avatar-sm position-relative profilepic_image" id="Photos" hidden>
        </div>
        <div class="col-8 col-sm-10">
          <div class="h-100">
            <h5 id="Penerima" class="mb-1 font-weight-bolder text-white">
            </h5>
          </div>
        </div>
        <div class="col-2 col-sm-1">
          <div class="avatar avatar-sm position-relative" id="remove" style="cursor:pointer">
            <i class="material-icons opacity-10">close</i>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
<div class="card" style="border:1px solid grey;background-image: url('../Img/ChatBG.jpg'); background-size:450px; filter: brightness(95%)">
  <div class="card-body pt-0 mt-5" id="chat" style="height:430px; overflow-y:scroll;">
    <input type="hidden" id="Sender" />
    <input type="hidden" id="Receiver" />
    <p id="discussion"></p>
  </div>
</div>
<div class="card-footer p-0 position-relative mt-n4 mx-3 z-index-2">
  <div class="row bg-white border-radius-lg pt-2" style="border:1px solid grey">
    <div class="col-sm-9">
      <div class="input-group input-group-dynamic">
        <input type="text" class="form-control" id="message" name="message" placeholder="Type your message here..." required>
      </div>
    </div>
    <div class="col-sm-1 text-end">
      <div class="input-group-btn input-group-dynamic">
        <input class="btn btn-success" type="submit" id="sendmessage" value="Send">
      </div>
    </div>
  </div>
</div>
</div>
```

Figure 5. 83 Implementation of show chat in views

In the script section that must be added Chathub, which we have made a class from the start.

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script src="~/Scripts/jquery.signalR-2.4.3.min.js"></script>
<script src="~/signalr/hubs"></script>
```

Figure 5. 84 Implementation of declare the library

More or less, to display the chat that was just sent has the same code, so go straight to this stage. The script below is used to connect to ChatHub. The only difference between customer service and regular customers is that in the display section, type the first message to send.

```

$(function () {
    // Reference the auto-generated proxy for the hub.
    var chatHub = $.connection.chatHub;

    // Create a function that the hub can call back to display messages.
    chatHub.client.addNewMessageToPage = function (sender, message) {
        if (sender === $('#Receiver').val() || sender === $('#Sender').val()) {
            // Add the message to the page.
            var chatClass = 'chat-receiver';
            if (sender === $('#Sender').val()) {
                chatClass = 'chat-sender';
            }
            $('#discussion').append('<div class="chat ' + chatClass + '"><strong>' + htmlEncode(sender)
                + '</strong>' + htmlEncode(message) + '</p>');
        }
    };

    // Get the user name and store it to prepend to messages.
    $('#Sender').val($('#Session["Username"]').ToString());

    // Set initial focus to message input box.
    $('#message').focus();

    // Start the connection.
    $.connection.hub.start().done(function () {
        console.log("SignalR connection started");

        $('#sendMessage').click(function () {
            // Call the Send method on the hub.
            chatHub.server.send($('#Sender').val(), $('#message').val(), $('#Receiver').val());

            // Clear text box and reset focus for next comment.
            $('#message').val('').focus();
        });
    }).fail(function () {
        console.error("SignalR connection failed");
    });
});

```

Figure 5. 85 Implementation of connect the chathub function

5.2.15 Analytics Page

The Analytics page is a page that contains a chart to display the amount of sales data per month. Starting from the Controller, as usual we first get data via query, then we declare the columns of each month one by one.

```

public JsonResult About2023()
{
    try
    {
        string[] About2023 = new string[12];

        String Mainconn1 = ConfigurationManager.ConnectionStrings["Finpro"].ConnectionString;
        SqlConnection sqlconn1 = new SqlConnection(Mainconn1);
        String sqlquery = "select * from Order_Total_Month_Pivot where tahun = '2023'";
        SqlCommand sqlcomm = new SqlCommand(sqlquery, sqlconn1);
        sqlconn1.Open();
        SqlDataAdapter sda = new SqlDataAdapter(sqlcomm);
        DataTable ds = new DataTable();
        sda.Fill(ds);

        foreach (DataRow dr in ds.Rows)
        {
            if (dr["January"] == DBNull.Value)
            {
                About2023[0] = "0";
            }
            else
            {
                About2023[0] = ds.Rows[0]["January"].ToString();
            }

            if (dr["February"] == DBNull.Value)
            {
                About2023[1] = "0";
            }
            else
            {
                About2023[1] = ds.Rows[0]["February"].ToString();
            }

            if (dr["March"] == DBNull.Value)
            {
                About2023[2] = "0";
            }
        }
    }
}

```

Figure 5. 86 Implementation of analytic function

```

if (dr["March"] == DBNull.Value)
{
    About2023[2] = "0";
}
else
{
    About2023[2] = ds.Rows[0]["March"].ToString();
}

if (dr["April"] == DBNull.Value)
{
    About2023[3] = "0";
}
else
{
    About2023[3] = ds.Rows[0]["April"].ToString();
}

if (dr["May"] == DBNull.Value)
{
    About2023[4] = "0";
}
else
{
    About2023[4] = ds.Rows[0]["May"].ToString();
}

if (dr["June"] == DBNull.Value)
{
    About2023[5] = "0";
}
else
{
    About2023[5] = ds.Rows[0]["June"].ToString();
}

```

Figure 5. 87 Implementation of analytic function (continue)

Continue to Views, here we first create the div according to the id of the Chart function contained in the following script section. Call the function from About2023 via Ajax. And declare the variable of that month one by one.

```

<div class="card mb-4">
  <div class="card-body px-5 pt-5 pb-5">
    <figure class="highcharts-figure">
      <div id="container"></div>
      <p class="highcharts-description">
        Chart showing use of rotated axis labels and data labels. This can be a
        way to include more labels in the chart, but note that more labels can
        sometimes make charts harder to read.
      </p>
    </figure>
  </div>
</div>

```

Figure 5. 88 Implementation of declare id of chart in views

```

@section scripts
<script src="~/Scripts/jquery-3.4.1.js"></script>
<script src="~/Scripts/jquery-3.4.1.min.js"></script>
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
<script src="https://code.highcharts.com/modules/export-data.js"></script>
<script src="https://code.highcharts.com/modules/accessibility.js"></script>
</script>

// Build the chart

$(document).ready(function () {

$.ajax({
  type: "POST",
  url: 'About2023',
  data: JSON.stringify({}),
  contentType: "application/json;charset=utf-8",
  dataType: "json",
  success: function (json) {
    debugger
    var values = json.About2023;
    var January = parseInt(values[0]);
    var February = parseInt(values[1]);
    var March = parseInt(values[2]);
    var April = parseInt(values[3]);
    var May = parseInt(values[4]);
    var June = parseInt(values[5]);
    var July = parseInt(values[6]);
    var August = parseInt(values[7]);
    var September = parseInt(values[8]);
    var October = parseInt(values[9]);
    var November = parseInt(values[10]);
    var December = parseInt(values[11]);

    Highcharts.chart('container', {
      chart: {
        type: 'column'
      },
      title: {
        text: 'Total Order In 2023'
      },
    });
  }
});
}

```

Figure 5. 89 Implementation of declare chart.js in views

```

      style: {
        fontSize: '13px',
        fontFamily: 'Verdana, sans-serif'
      }
    },
    yAxis: {
      min: 0,
      title: {
        text: ''
      }
    },
    legend: {
      enabled: false
    },
    tooltip: {
      pointFormat: 'Total Orders: <b>{point.y}</b>'
    },
    series: [{
      name: 'Total Order',
      data: [
        ['January', January],
        ['February', February],
        ['March', March],
        ['April', April],
        ['May', May],
        ['June', June],
        ['July', July],
        ['August', August],
        ['September', September],
        ['October', October],
        ['November', November],
        ['December', December]
      ],
      dataLabels: {
        enabled: true,
        rotation: 0,
        color: '#FFFFFF',
        align: 'right',
        format: '{point.y}', // one decimal
        y: 5, // 10 pixels down from the top
      }
    }
  ]
}

```

Figure 5. 90 Implementation of declare chart.js in views (continue)

CHAPTER VI

SYSTEM TESTING

6.1 Testing Environment

The testing environment

Hardware:

- Lenovo Ideapad Laptop
- Windows 11

Software:

- Visual studio 2019
- Google chrome

Testing Scenario :

6.1.1 Login & Register

Table 6. 1 Testing scenario of Login & Register

No	Scenario	Expected Result	Result
1	Customers, Customer Service, And Owner can Login	Load the page and directly bring to the each their main page	As expected (shows in Figure 5.1.1)
2	Customers, Customer Service, And Owner input wrong Id & pass	Page, username, and password to reset. so the user must retype	As expected

3	Register Customers	Can register their account. And the data will saved to databases	

6.1.2 *Forgot Password*

Table 6. 2 Testing scenario of Forgot Password

No	Scenario	Expected Result	Result
1	Display forgot password input	Customers can see the textbox email and input their email	As expected (shows in Figure 5.1.2)
2	Customer can request the reset password	Load the page and the verification of reset password email will be sended into their email	As expected
3	Customer click the link of reset password	Directly move to reset password page	As Expected
4	Customer input new password and confirm password	Data inside database will be updated and directly move to the login page	As Expected

5	Customer input the confirm password not same with the new password	Notification of wrong input password is appear and user need to retype the confirm password same as new password	
---	--	--	--

6.1.3 Menu Page & Category List

Table 6. 3 Testing scenario of Menu Page & Category List

No	Scenario	Expected Result	Result
1	Display special menu	Customers can see the special menu in this page	As expected (shows in Figure 5.1.2)
2	Display Category List	Customers can see the category of each menu and can click the button of category to directly bring to Menu list page	As expected

6.1.4 Menu List

Table 6. 4 Testing scenario of Menu List

No	Scenario	Expected Result	Result
1	Display Menu of each category which customers choose	Customers can see the menu in this page	As expected (shows in Figure 5.1.3)

2	Add to cart	Customers can add the menu which they choose to cart	As expected
3	Search the menu with bubble sort feature	Customers can search which menu they want	

6.1.5 *Cart Page*

Table 6. 5 Testing scenario of Cart Page

No	Scenario	Expected Result	Result
1	Display the menu which already chosen by customers	Customers can see what are the menu has been chosen if click the pay it will directly open the modal to fill the information about payment method and table number and then directly send to payment page	As expected (shows in Figure 5.1.4)
2	Edit quantity	Customers can edit quantity minimum 1 dan maximum is based on total in stock	As expected

3	Product Empty	Load page cart empty and button to Menu Page	
---	---------------	--	--

6.1.6 Account Page

Table 6. 6 Testing scenario of Account Page

No	Scenario	Expected Result	Result
1	Display the information about Customers account	Customers can see their own information	As expected (shows in Figure 5.1.5)

6.1.7 History Order Pages

Table 6. 7 Testing scenario of History Order Pages

No	Scenario	Expected Result	Result
1	Display the order who has status been delivered	Customer service and owner can see the delivered order and if click the details button it will directly send to details menu	As expected (shows in Figure 5.1.6)
2	Search the menu with datatable search feature	Customer service and Owner can search which menu they want	As expected

6.1.8 Current Order Pages

Table 6. 8 Testing scenario of Current Order Pages

No	Scenario	Expected Result	Result
1	Display the order who has been ordered this day	Customer service and owner can see the new order this day. For customer they can see what they order in that time and if click the details button it will directly send to details menu	As expected (shows in Figure 5.1.7.1)
2	Display the detail order of each customer	Customer service and owner can see the detail order of each customer. For customer they can see detail of their order item and show the feature of estimation time	As expected (shows in Figure 5.1.7.2)
3	Search the menu with datatable search feature	Customer service and Owner can search which menu they want	As expected

6.1.9 Payment

Table 6. 9 Testing Scenario of Payment

No	Scenario	Expected Result	Result
1	Displays the payment destination account	Customer dan see the account number or qr code, then directly can do the payment	As expected (shows in Figure 5.1.8)

2	Upload the approval photo of payment	Customer can upload the approval photo of their payment to Owner	As expected
3	Cancel Order	Customer can cancel their order and then the stock will reset	

6.1.10 Payment Pages

Table 6. 10 Testing scenario of Payment Pages

No	Scenario	Expected Result	Result
1	Displays the payment destination of owner account	Owner can see their payment qr code and account number	As expected (shows in Figure 5.1.9)
2	Update payment method	Owner can updated their payment method	As expected

6.1.11 Categories Page

Table 6. 11 Testing scenario of Categories Page

No	Scenario	Expected Result	Result
1	Display all category listed	Owner can see their category listed in the restaurant	As expected (shows in Figure 5.1.10)
2	Add new category	Owner can add new category listed	As expected

3	Edit category	Owner can edit their category listed	
4	Delete category	Owner can delete their category listed	

6.1.12 Menulist Page

Table 6. 12 Testing scenario of Menulist Page

No	Scenario	Expected Result	Result
1	Display all Menu listed	Owner can see their Menu listed in the restaurant	As expected (shows in Figure 5.1.11)
2	Add new Menu	Owner can add new menu listed	As expected
3	Edit price and stock	The owner can update the price and stock of each menu listed	
4	Delete menu	Owner can delete their menu listed	

6.1.13 Reports Page

Table 6. 13 Testing scenario of Reports Page

No	Scenario	Expected Result	Result
1	Display total income daily and per month	Owner can see their income nett per day	As expected (shows in

		and per month	Figure 5.1.12)
2	Search the income with datatable search feature	Owner can search which value they want To search	As expected
3	Search income by date range	Owner can search which day they want to search by date range feature	

6.1.14 Chat Pages

Table 6. 14 Testing scenario of Chat Pages

No	Scenario	Expected Result	Result
1	Display the chat	Customer service and customer can see their message	As expected (shows in Figure 5.1.13)
2	Sending the chat	Customer can send their message to customer service. And customer service can send their message to customer	As expected

6.1.15 Analytics Page

Table 6. 15 Testing scenario of Analytics Page

No	Scenario	Expected Result	Result
----	----------	-----------------	--------

1	Display the total order	Owner can see the total orders per month displayed on the chart by year	As expected (shows in Figure 5.1.14)
2	Download the chart	Owner can download the chart data with pdf format, jpg format, and many more format	As expected

6.2 Testing Summary

The conclusion is that this application has been tested, and the results are what is expected, such as the user can already order the food and beverage , the owner will confirm the order, The customer service updated the status. And customer can see the feature using several algorithm in the website.

CHAPTER VII

CONCLUSION AND FUTURE WORKS

7.1 Conclusion

The project development has reached its completion in this chapter. In conclusion, the applications submitted generally meet the requirements. Here are the important features, estimating the time needed to make each food menu, bubble sorting, and chat function to allow customers and customer service to communicate, greedy algorithm theory. The linear equation used to predict this time based on customer trust can help customers understand when food menus will be provided. Then chat option between client and kitchen or customer service. Customers will find it easier to sort food, thanks to bubble sort..

7.2 Future Works

The application can be accessed on a cellphone, but some display issues remain. In particular, owners and customer service must access the website using a laptop or PC because there is a lot of data. Future applications could take advantage of AI or machine intelligence and incorporate Shopee pay APIs or those from other funds or e-wallets to facilitate customer payments. Then it is simpler for the user to learn the status order status if an automatic status system is used, such as automatic status updates once the status has been displayed.