



## **GROUP ASSIGNMENT**

**CT038-3-2-ODJ**

**OBJECT ORIENTED DEVELOPMENT WITH JAVA**

**APD2F2411CS**

**HAND OUT DATE: WEEK 7 DECEMBER 2024**

**HAND IN DATE: WEEK 13 FEBRUARY 2025**

**LECTURER NAME: MUHAMMAD HUZAIFAH BIN ISMAIL**

### **Workload Matrix**

<b>Group 15</b>	<b>Student Name</b>	<b>Student ID</b>	<b>Parts done</b>
1	Eshchanov Umidjon Umrbek Ugli	TP071568	Administrator
2	Choong Ti Shen	TP078540	Delivery Runner
3	Phang Sieow Jun	TP072541	Customer
4	Nye Wei Jun	TP066836	Vendor
5	Mohammed Ashraf Hasanain	TP078045	Manager

## Table of Contents

1.0 Design solution .....	5
1.1 Use Case Diagram.....	5
1.1.1 Use Casy Diagram System.....	5
1.1.2 Use Casy Diagram Administrator .....	6
1.1.3 Use Case Diagram Vendor .....	7
1.1.4 Use Case Diagram Customer .....	8
1.1.5 Use Case Diagram Runner.....	9
1.3 Class Diagram.....	10
1.3.1 Class Diagram Administrator:.....	10
1.3.2 Class Diagram Vendor .....	11
1.3.3.3 Class Diagram Runner .....	12
2.0 Screenshots of output of the program with appropriate explanations.....	13
2.1 Login Page .....	13
2.2 Administrator (Eshchanov Umidjon TP071568) .....	14
2.3 Manager (Mohammed Ashraf TP078045) .....	19
2.4 Vendor (Nye Wei Jun TP066836) .....	24
2.5 Runner (Choong Ti Shen TP078540) .....	31
2.6 Customer (Phang Sieow Jun TP072541) .....	46
2.6.1 Main menu .....	46
2.6.2 Payment Statement.....	47
2.6.3 Check Status Interface .....	48
2.6.4 Review/Complain Interface .....	49
2.6.5 Order history Interface .....	50
3.0 Description and justification of Object-oriented concepts incorporated into the solution .....	52
3.1 Administrator (Eshchanov Umidjon TP071568) .....	52
3.1.1 Concept 1 (Encapsulation):.....	52
3.1.2 Concept 2 (Inheritance): .....	53
3.1.3 Concept 3 (Polymorphism):.....	54
3.2 Manager (Mohammed Ashraf TP078045) .....	55
3.2.1 Concept 1 (Encapsulation):.....	55
3.2.2 Concept 2 (Encapsulation):.....	56

3.3 Vendor (Nye Wei Jun TP066836) .....	57
3.3.1 Concept 1 (Encapsulation).....	57
3.3.2 Concept 2 (Inheritance).....	59
3.4 Runner (Choong Ti Shen TP078540) .....	60
3.4.1 Encapsulation.....	60
3.4.2 Inheritance.....	62
3.4.3 Abstraction .....	63
3.4.4 Composition.....	64
3.5 Customer (TP072541).....	65
3.5.1 Encapsulation.....	65
3.5.2 Polymorphism.....	66
4.0 Additional feature.....	67
4.1 Administrator (Real-Time Receipt Generation):.....	67
4.2 Manager (Enhanced Functionality): .....	68
4.3 Vendor (Enhancing the Order Management System: Adding Cursor Selection for Order Acceptance).....	69
4.4 Runner.....	70
4.4.1 Total Earned Calculating in Dashboard .....	70
4.4.2 Enhanced Rating Message at View Customer Review .....	72
5.0 Limitations .....	73
6.0 Conclusion .....	73
7.0 References.....	74
8.0 Appendix .....	75
8.1 Administrator (Eshchanov Umidjon TP071568) .....	75
8.2 Manager (Mohammed Ashraf TP078045) .....	76
8.3 Vendor (Nye Wei Jun TP066836) .....	78
8.4 Runner (Choong Ti Shen TP078540) .....	79
8.5 Customer (TP072541).....	81
9.0 Presentation video links: .....	82
9.1 Administrator: .....	82
9.3 Vendor: Java Presentation (Vendor).mp4.....	82
9.4 Customer: Java customer by Phang Sieow Jun TP072541.mp4.....	82

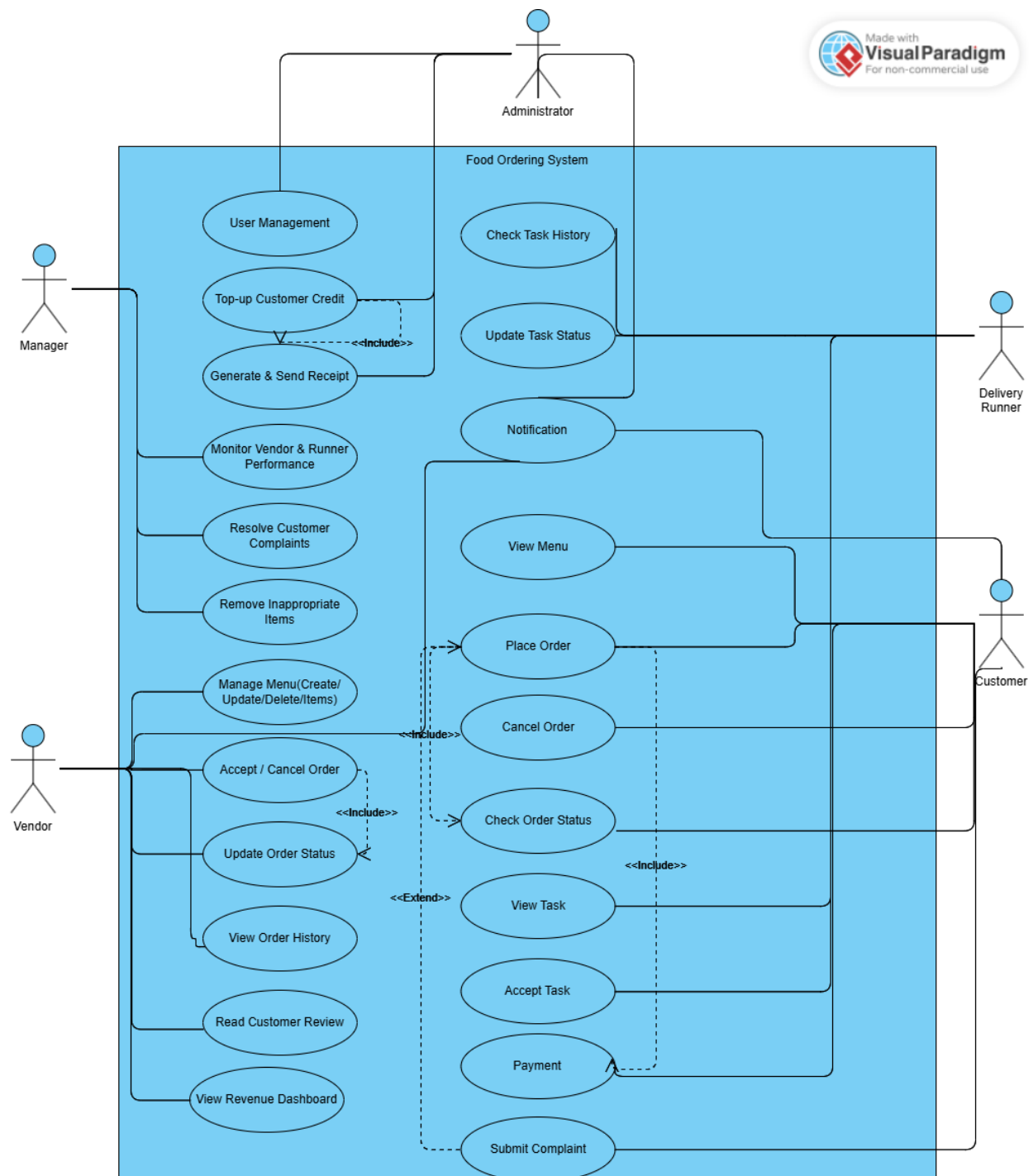
## 9.5 Runner

[https://drive.google.com/drive/folders/1oYMQC5UjOo1wi\\_Z0vH5J\\_zd2X3t7OP9s](https://drive.google.com/drive/folders/1oYMQC5UjOo1wi_Z0vH5J_zd2X3t7OP9s)..... 82

# 1.0 Design solution

## 1.1 Use Case Diagram

### 1.1.1 Use Case Diagram System



### 1.1.2 Use Case Diagram Administrator

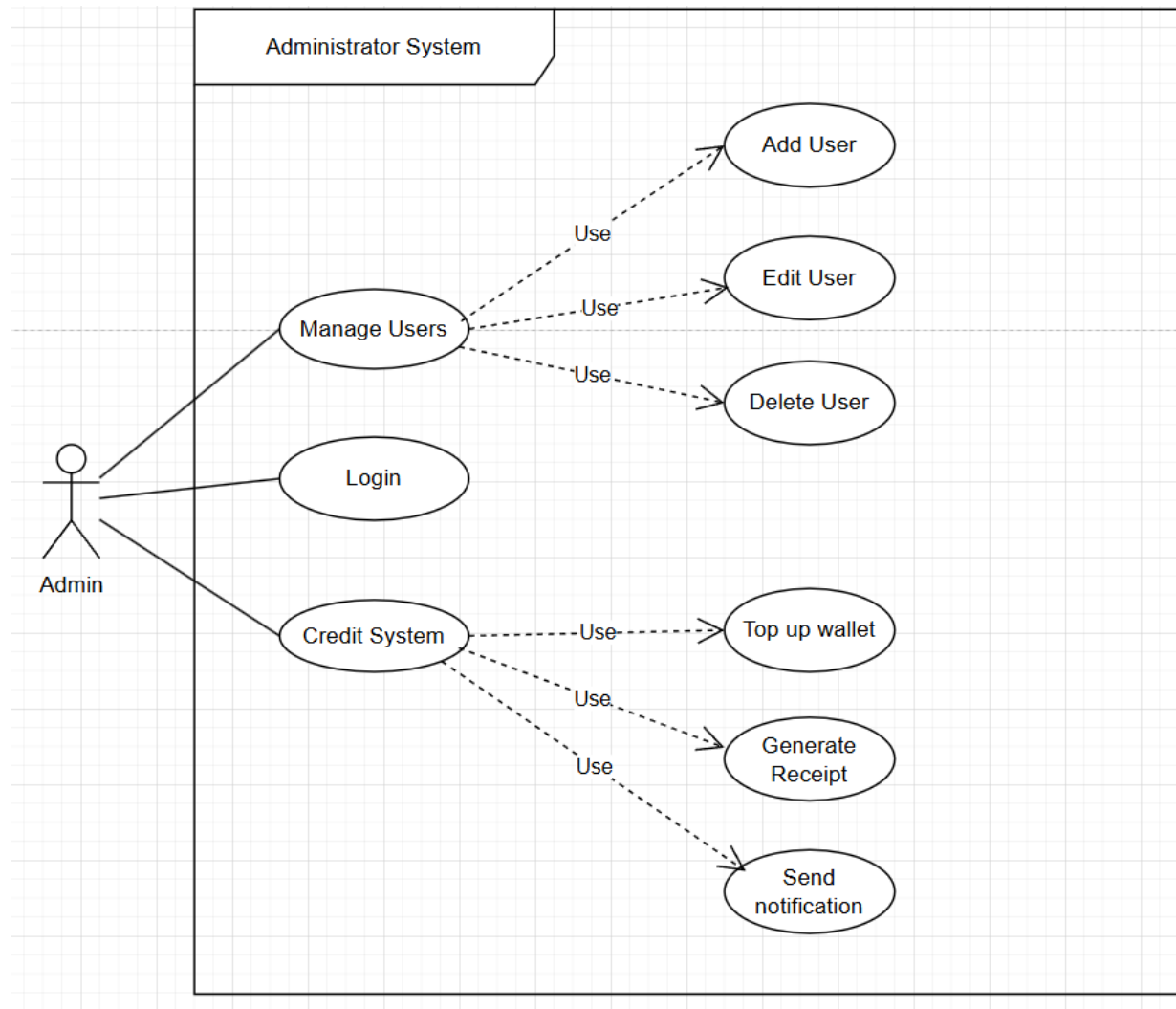


Figure 1.1.2.1 Administrator Use case Diagram

### 1.1.3 Use Case Diagram Vendor

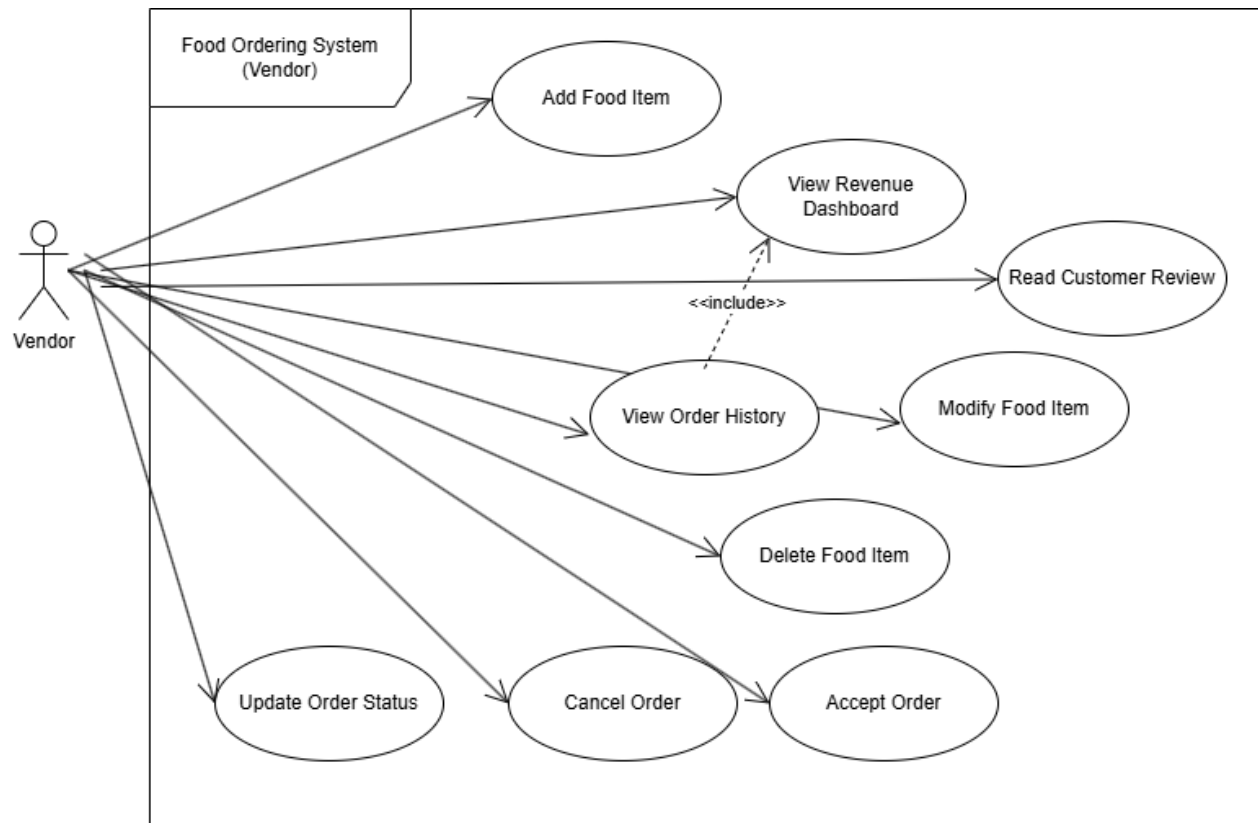
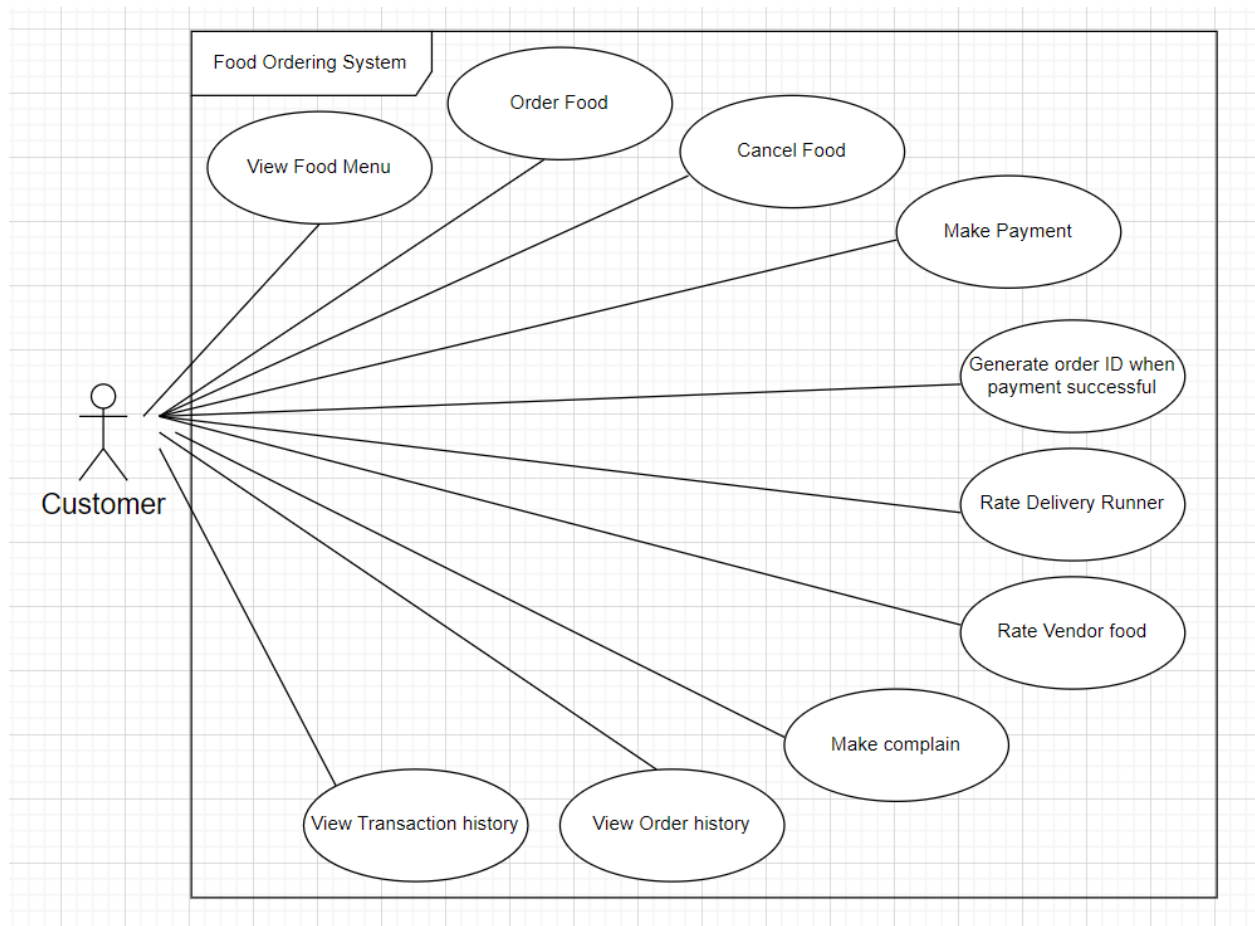


Figure 1.1.3.1 Vendor Use Case Diagram

### 1.1.4 Use Case Diagram Customer





### 1.1.5 Use Case Diagram Runner

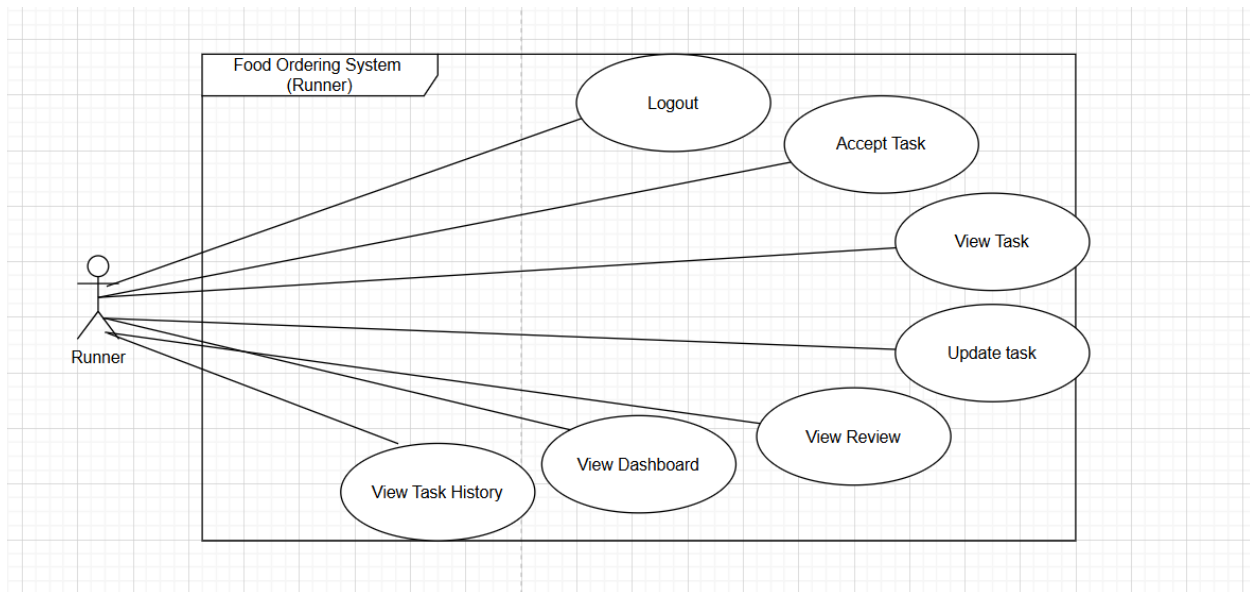


Figure 1.1.5: Use Case Diagram for Runner

## 1.3 Class Diagram

### 1.3.1 Class Diagram Administrator:

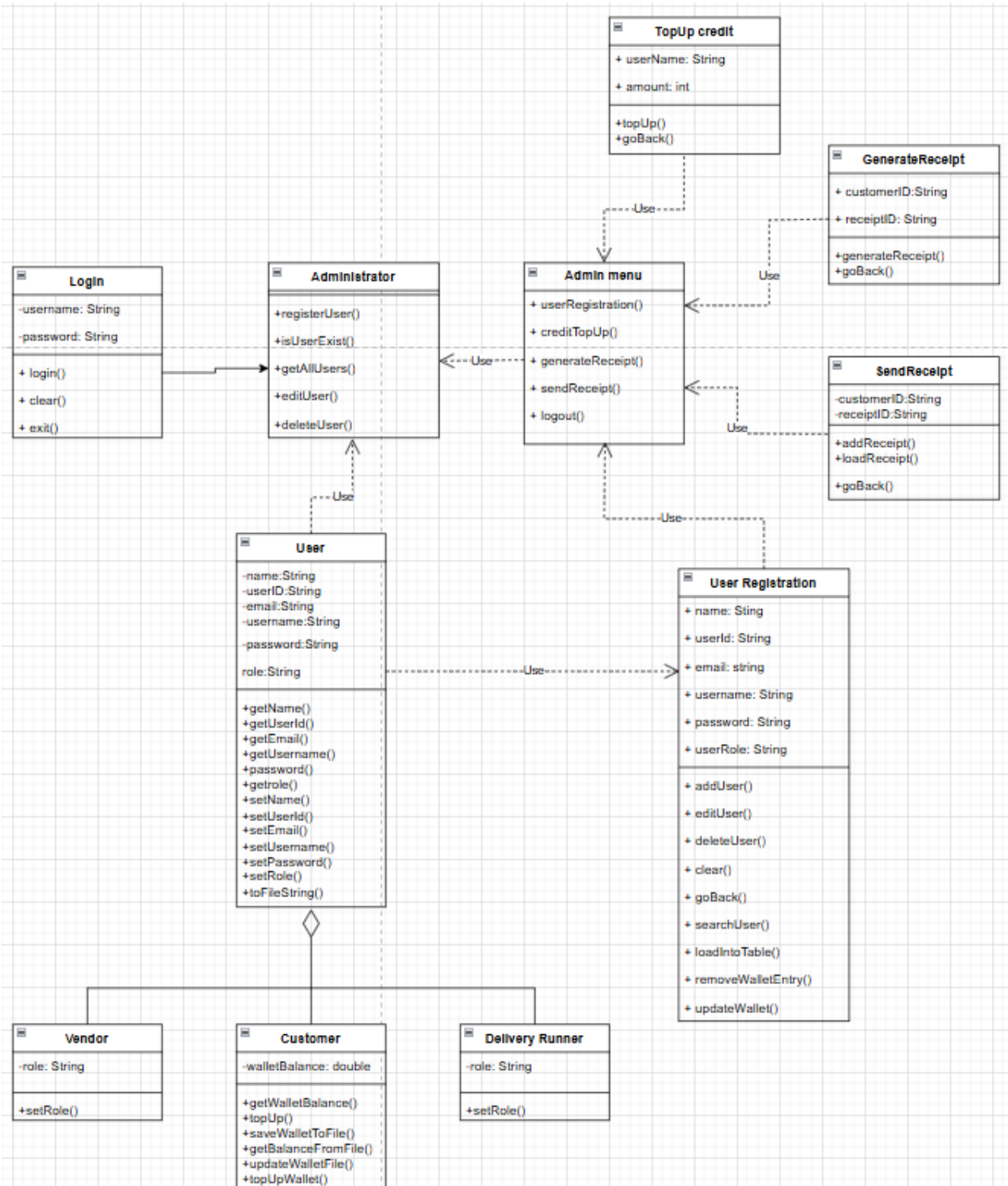


Figure 1.3.1.1 Administrator Class Diagram

### 1.3.2 Class Diagram Vendor

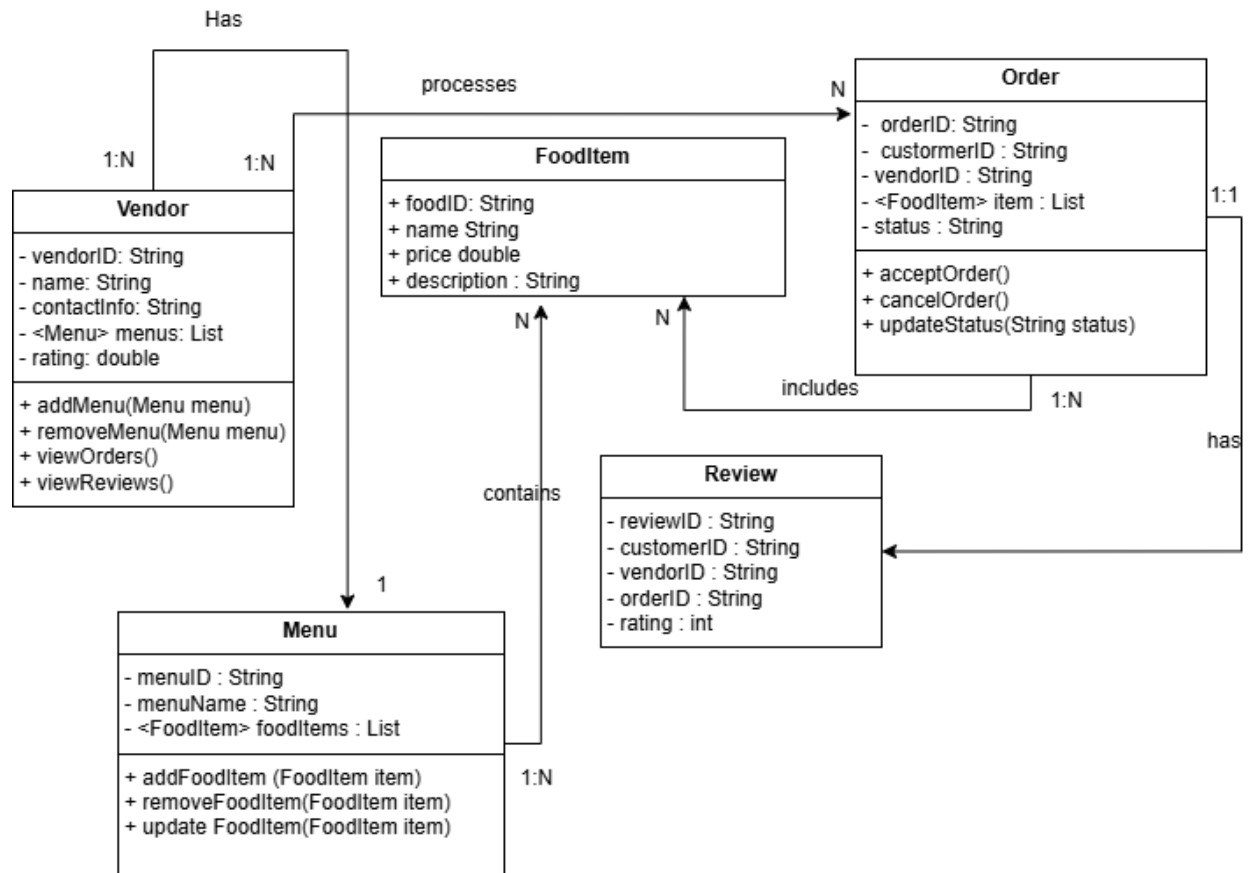


Figure 1.3.2.1 Vendor Class Diagram

### 1.3.3.3 Class Diagram Runner

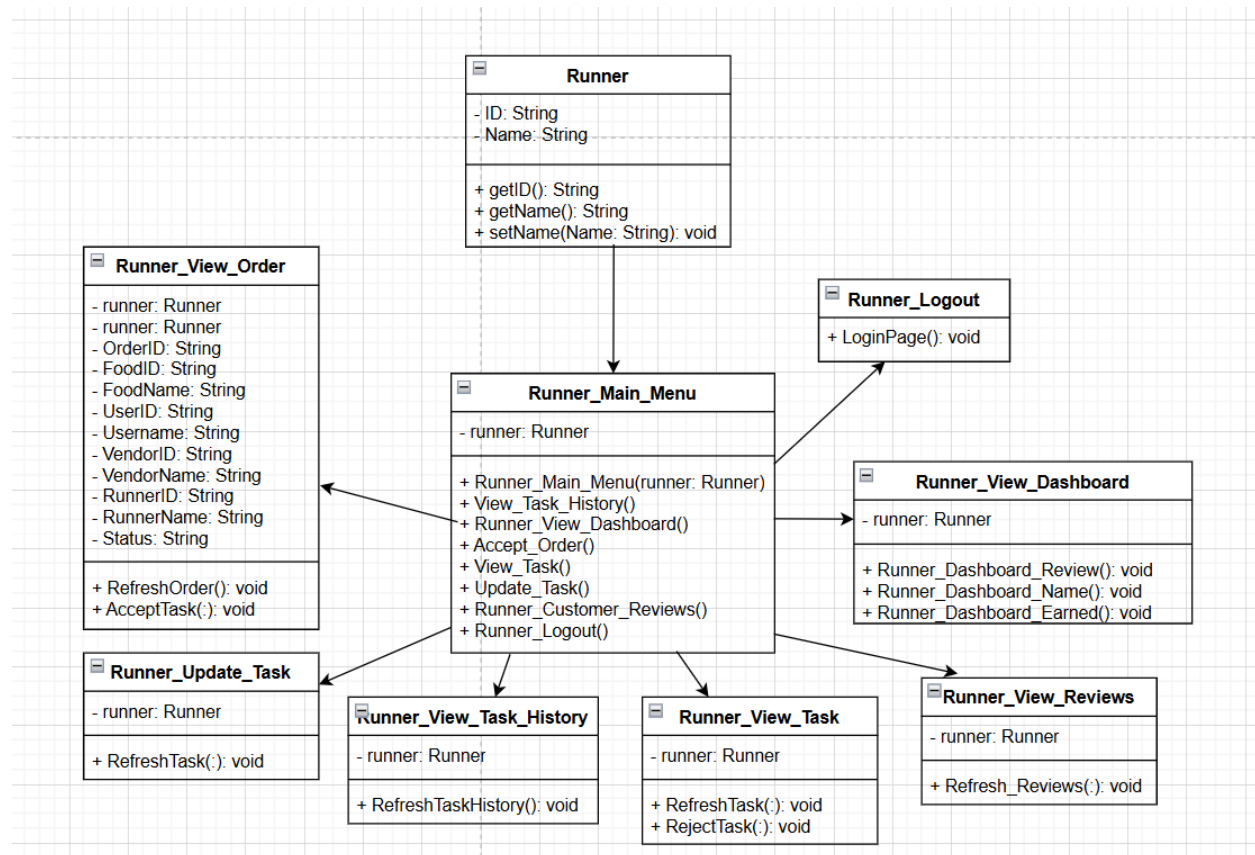


Figure 1.3.3.3 Class Diagram Runner

## 2.0 Screenshots of output of the program with appropriate explanations

### 2.1 Login Page

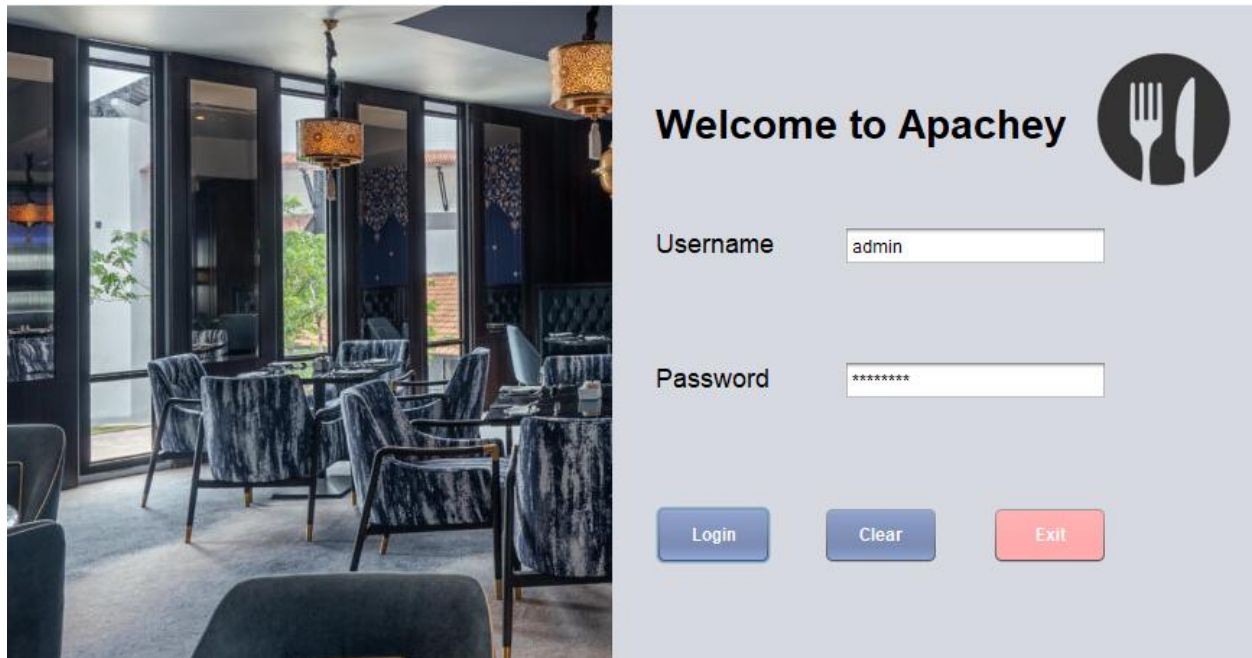


Figure 2.1.1 Login page

The LoginPage functions as the login access point between users and the system through which they must provide their username and password for entrance. Users can access the application through the LoginPage interface by using buttons including a Login confirmation and a Clearing option and an Exit command which will terminate the application. The interface presents a combination of welcome message, system logo and background image during its display. After starting up the program uses the main method to display the login screen. To ensure stronger authentication, the system requires a password to be at least eight characters long. In addition, all usernames and passwords are changed to lowercase during the login process to avoid case-sensitive inconsistencies. After a successful login, the system takes users to their individual dashboards and independently checks the credentials of the Admin, Manager, Vendor, and Runner. The user is notified by a pop-up error message if they enter wrong credentials. By improving usability and security, these changes guarantee a seamless login process.

## 2.2 Administrator (Eshchanov Umidjon TP071568)

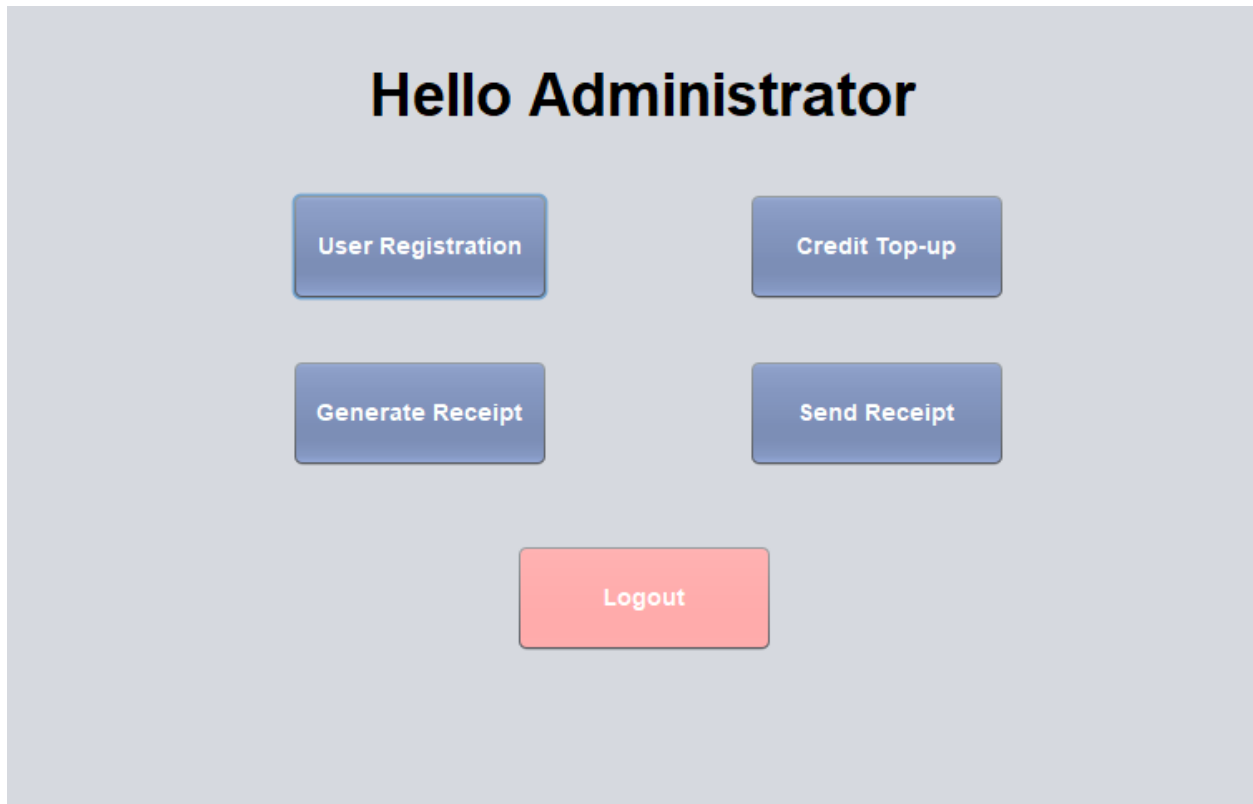


Figure 2.2.1 Administrator Menu

The primary interface for managing users and transactions is the Admin Menu. The administrator can access several features using this interface, including User Registration, Credit Top-up, Generate Receipt, and Send Receipt. The administrator can effectively manage user accounts and money activities by clicking on each item in the menu, which provides access to a specific feature. All the main features of the system are easily accessible and can be navigated thanks to this structured user interface. The program user may log into the admin menu by entering administrator details, username: admin, password: admin123.

## User Registration

Name

User ID

Email

Username

Password

User Role

Vendor

Vendor

Customer

Runner

Add

Clear

Back

User Search by Name

Name	User ID	Email	Username	Password	User Role
umid	1	umid@gmail.com	umid	12345678	Vendor
mak	2	mak@gmail.com	mak	87654321	Customer
tishen	3	tishen@gmail.com	tishen	00000000	Runner
weili	4	weili@gmail.com	weili	1234567890	Customer

Figure 2.2.2 User registration menu

The administrator can add, update, and remove users in real time through the User Registration interface. The system applies to a number of validations checks when a new user is registered. Together with the username, the User ID must be a unique integer. Emails must be formatted correctly, ending in '@gmail.com', and passwords must have at least eight characters. To maintain data integrity, the system shows the relevant error message if any of these requirements are not fulfilled. Following a successful registration, a user's information is instantly shown in the user table, which is updated in real time. Additionally, the administrator can easily modify user details by selecting a user from the table, which immediately fills in the text boxes. To provide consistency across the data files, once a user is deleted, their information is erased from the system. Once the deleted user was a client, wallets.txt is also cleared of their wallet information.

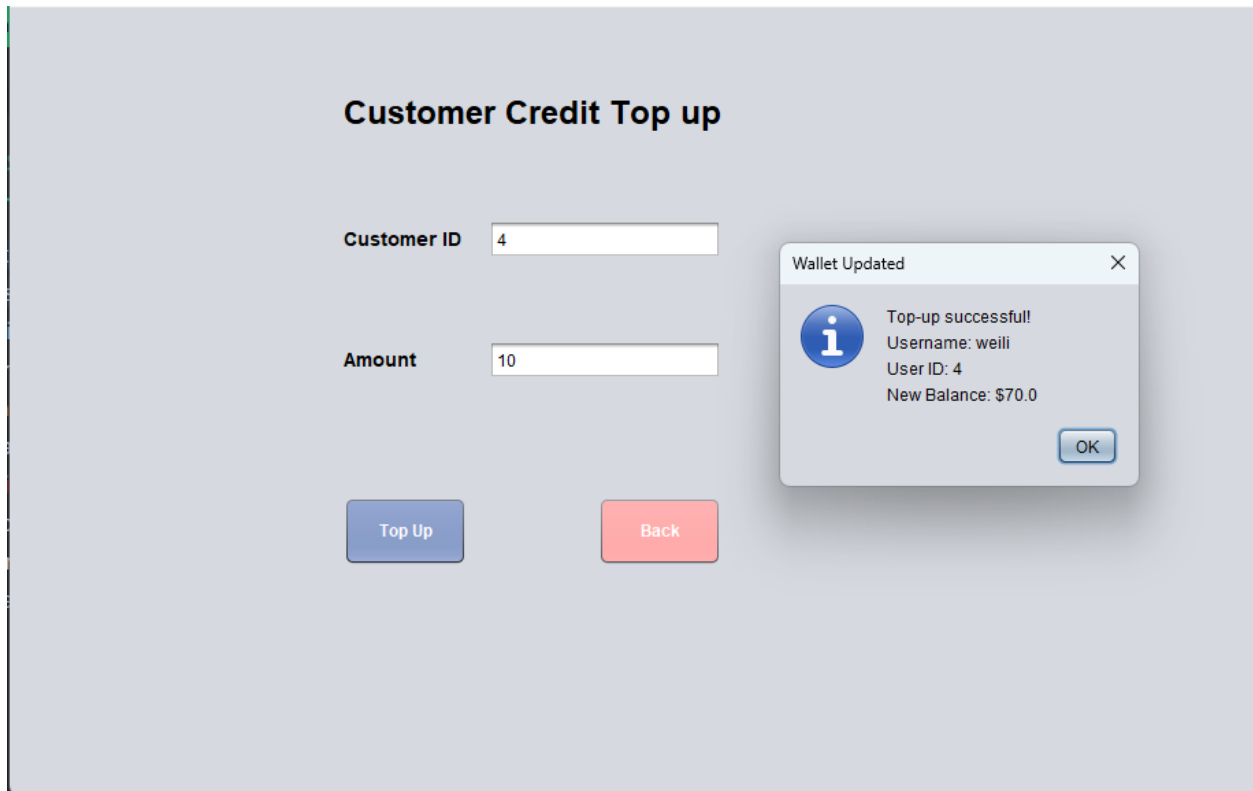


Figure 2.2.3 Top up customer menu

The administrator can add money to a customer's wallet balance by using the Credit Top-up tool. First, the system verifies that the user is present in the system and that the entered User ID is an integer. An error notice appears if the User ID entered is invalid or nonexistent. The wallet balance is updated in wallets.txt in accordance with the top-up amount entered and a valid User ID. A confirmation message with the user ID, username, and updated wallet balance is shown following a successful top-up. The wallets table also shows the updated wallet balance, guaranteeing accurate real-time recording of financial activities.



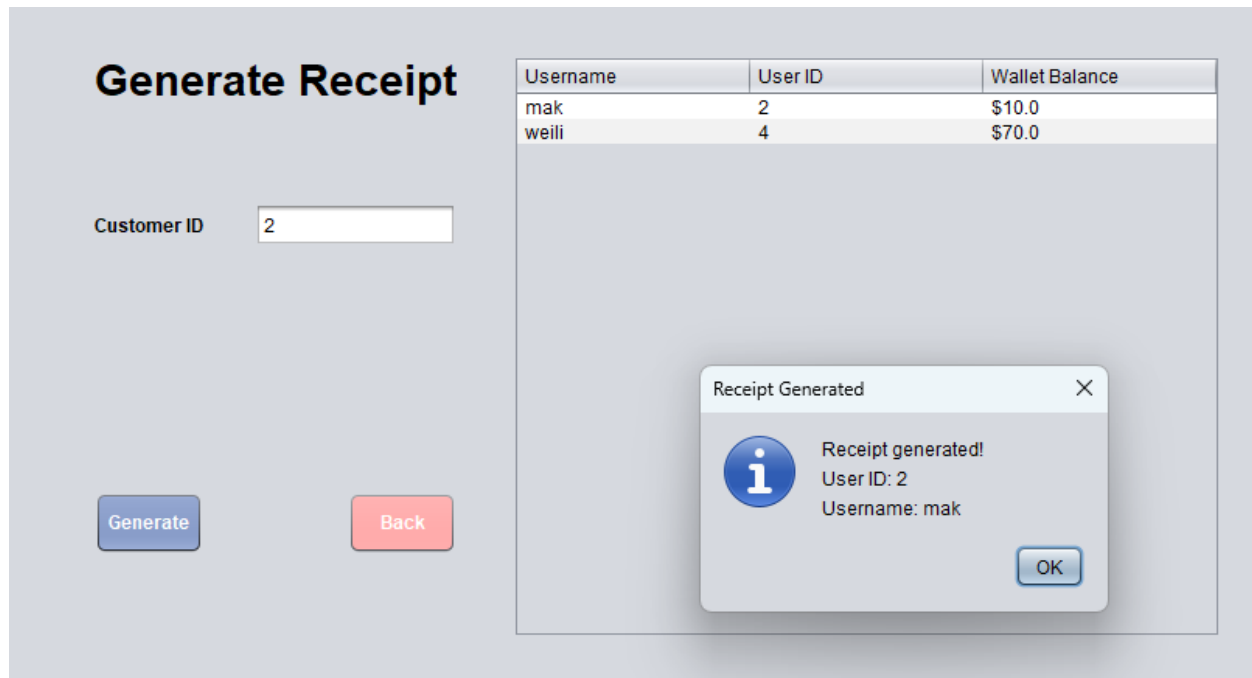


Figure 2.2.4 Generate receipt menu

The administrator can generate a receipt for a customer's most recent transaction using Generate Receipt functionality. The administrator must input a legitimate User ID in order to issue a receipt. The User ID must be an integer and contained in wallets.txt, according to the system's verification. An error message is shown if an invalid ID is entered. After verification, the system creates a structured receipt with the modified wallet balance, username, date and time, and receipt ID. In order to guarantee that every transaction has a documented history, this data is subsequently saved in receipts.txt. In order to enable the administrator to keep track of every receipt in the system, the created receipt ID is also added to the receipt table.

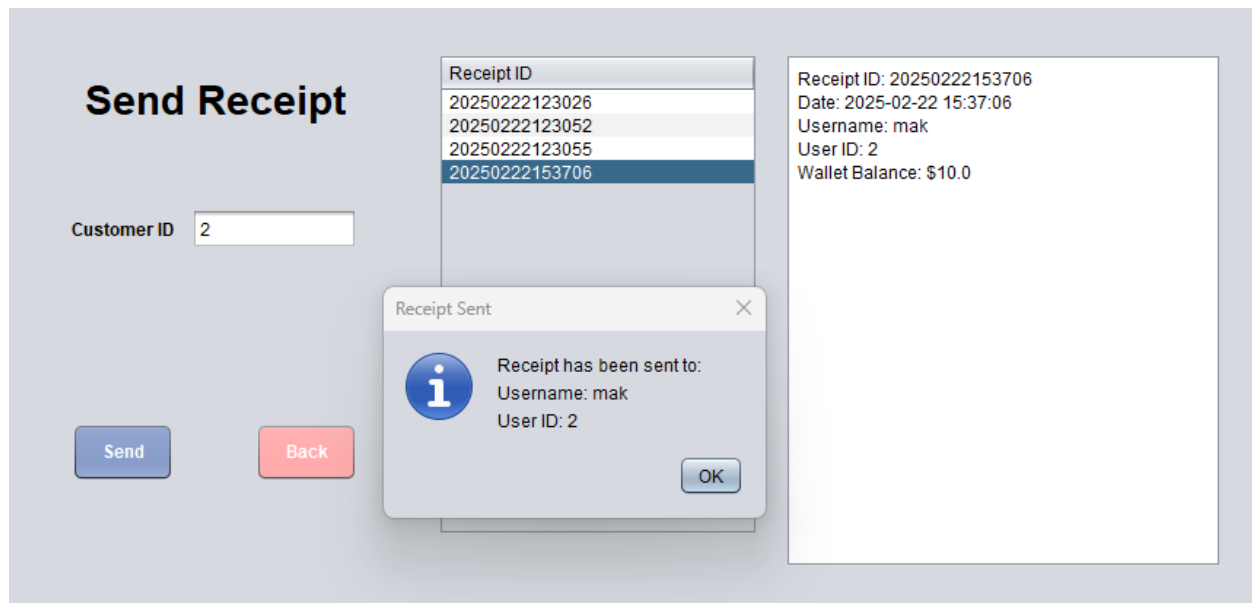


Figure 2.2.5 Send receipt to customer menu

The administrator can inform clients about their transactions by using the Send Receipt tool. Only the Receipt IDs of each produced receipt are shown on the receipt table. An overview of the transaction is provided via the receipt information section, which displays the whole receipt details when the administrator selects a receipt. The administrator must input a User ID prior to delivering a receipt. The entered User ID is compared to the one linked to the selected receipt by the system. Incorrect receipts cannot be sent since an error message is shown if the User ID does not match. A confirmation message informing the client that the receipt has been successfully sent to them shows after the right User ID has been entered. This function guarantees that clients receive precise transaction information and keeps an accurate record of all financial transactions.

## 2.3 Manager (Mohammed Ashraf TP078045)

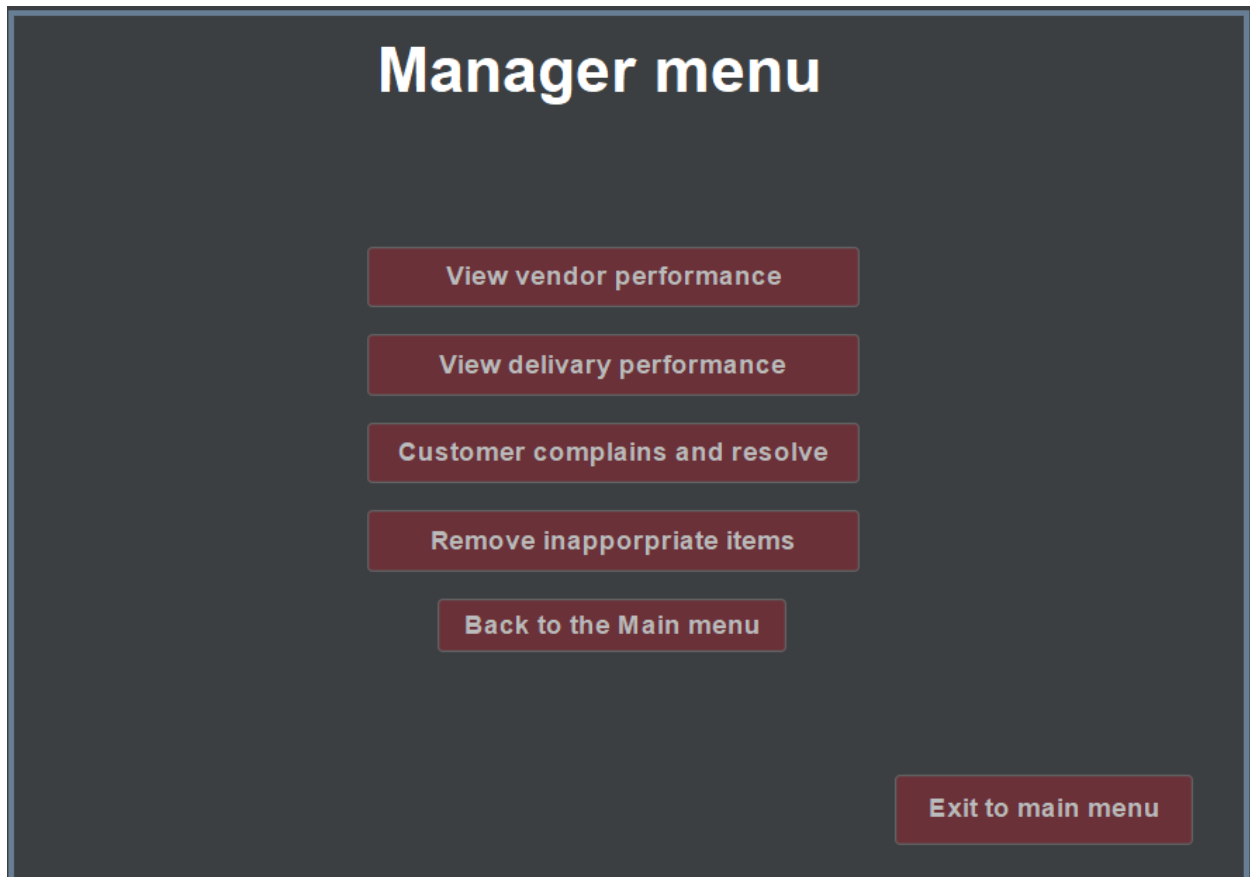


Figure 2.3.1 Manager menu

Managers use the centrally located Manager Main Interface to handle vendor administration as well as complaints management and food items oversight. The platform incorporates a neat display which presents well-arranged buttons that facilitate smooth navigation. Task management stays efficient because every section of the system is easily reachable to users. Social interfaces display a professionally elegant design which uses contemporary elements. The system improves workflow by creating an organized layout that maximizes system usability. The system maintains current data in real-time so managers can access the most recent information. The system components enable simple expansion when developers need to update the program infrastructure.

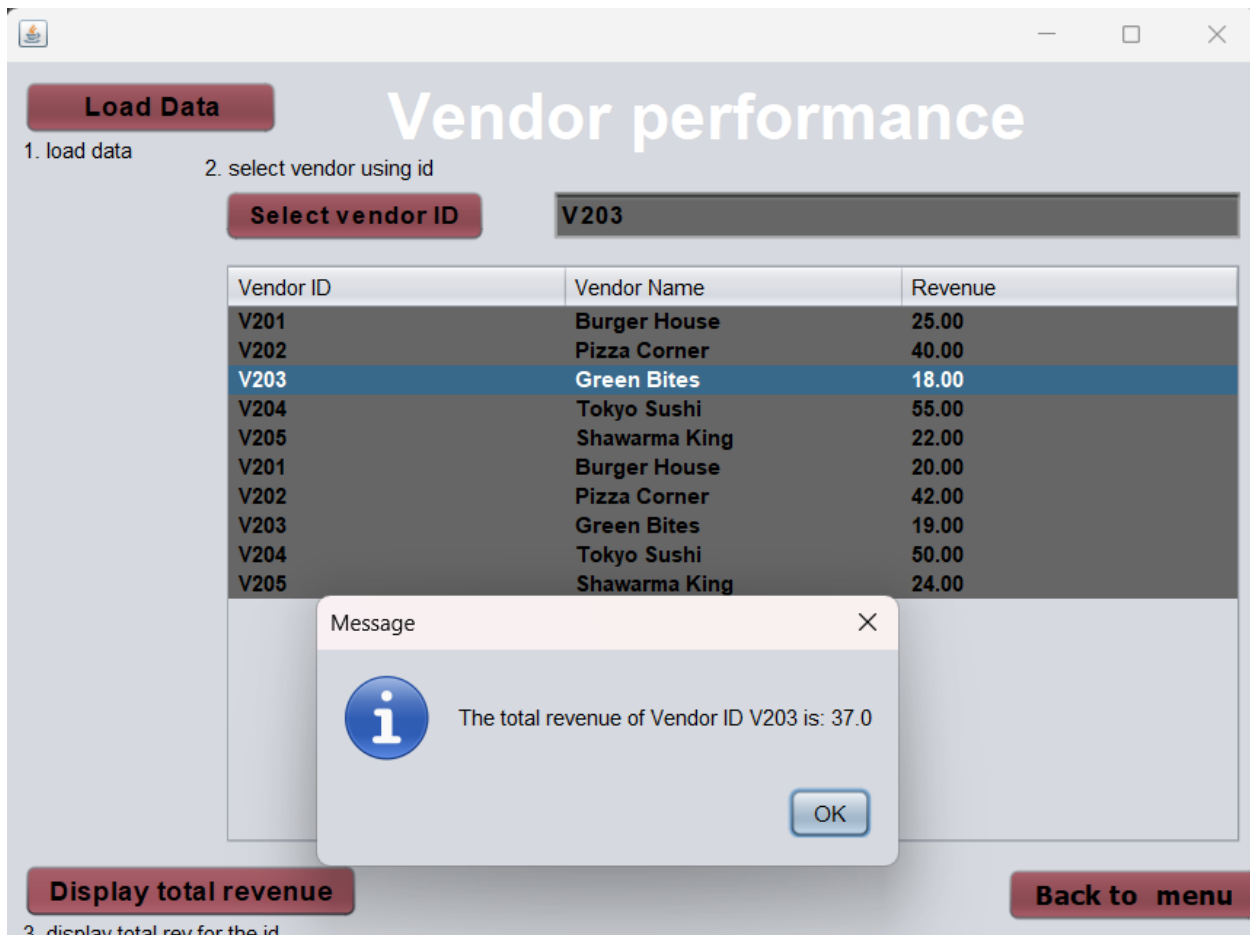


Figure 2.3.2 monitor vendor performance

The Vendor Performance Interface serves as a tool which delivers information about vendor reliability as well as operational efficiency. A dynamic table within the system shows important performance metrics and ratings as well as delivery speed combined with review data. Managers have an uncomplicated way to filter and sort vendor performance data while conducting assessments. The platform shows its data through a systematic user interface which makes it easier to understand information. Decision-quality increases through this interface which provides an organized professional dashboard presentation. The application draws its vendor information from the Review\_Vendor.txt file inside its database. Users can fetch financial data

about a vendor by selecting their vendor ID from the system which enables the presentation of their total revenue amount.

serch using runner id

**Search** R9

Runner name	Runner ID	Order ID	Customer Revi...	Runner Rate
John	R9	4	2	★ ★

please load data first

**Load Delivery data** **Back to menu**

Figure 2.3.3 delivery performance monitoring

Through the Delivery Preferences Interface managers possess the ability to adjust delivery configuration preferences. The system offers users to control delivery timing and options through select boxes and text entry fields. The system offers functional buttons to provide users with easy access to system reset functions and admin changes. The configuration process becomes smooth when there is a neat and organized design structure. Through its design approach the delivery management system increases accuracy which leads to better operational efficiency. The interface fetches delivery runner performance data which exists in the Review\_Runner.txt

document. Using a "Search" command with a runner ID leads the interface to show the rating belonging to the selected runner for performance assessment by managers.

Customer name	Customer ID	Order ID	Runner ID	Compliant
Aisha	C1003	3	R6	pizza was burnt
Ahmed	C1004	4	R9	fries were soggy
Ahmed	C1005	5	R12	sandwich had no sa...

Figure 2.3.4 resolve customer complaint

A dedicated interface named Customer Complaint Interface enables rapid and efficient complaint handling process. Complaints appear in an organized table structure which enables simple review of customer problems. The manager operates through functional buttons which let them provide responses for complaint resolution. The user interface system provides both efficient documentation features and capable issue management capabilities. This interface design improves how customers experience and handle complaint management activities. The system retrieves complaints in complain.txt format to permit managers for choosing complaints through customer names. A user selection of customer ID will display matching name information and a

"Resolve" button lets the manager transform complaint status to "Resolved" while eliminating the entry from the display table.

**Inappropriate Item remove**

2. search using food id

search Food ID

Food ID	Food Name	Price	Vendor ID	Vendor Name
F5	Popcorn Chick...	4.99	V2	Popcorn Chic...
F6	Mashed Potat...	2.79	V2	Mashed Potat...
F7	Turkey Sub	6.99	V3	Turkey Sub
F8	Italian BMT	7.49	V3	Italian BMT

1. Load data

Load Data

3. by selecting delete it will delete

Delete inappropriate

Back to menu

Figure 2.3.5 inappropriate item remove

Through the Inappropriate Item Interface managers achieve the ability to inspect flagged food items before removing them from the system. A table displays inappropriate items with details for managers to make better decisions. Buttons located near the table allow users to accomplish their tasks with speed and efficiency before taking a final step. The system design provides an intuitive structure which helps users navigate items without errors during management procedures. Quality control depends on confirmations which stop accidental removal of items. The system requires input from the database followed by manager selection of items through

their foodID values. The press of "Delete" enables the system to remove the selected row both from the table display and the Order.txt data storage for optimized removal operations.

## 2.4 Vendor (Nye Wei Jun TP066836)

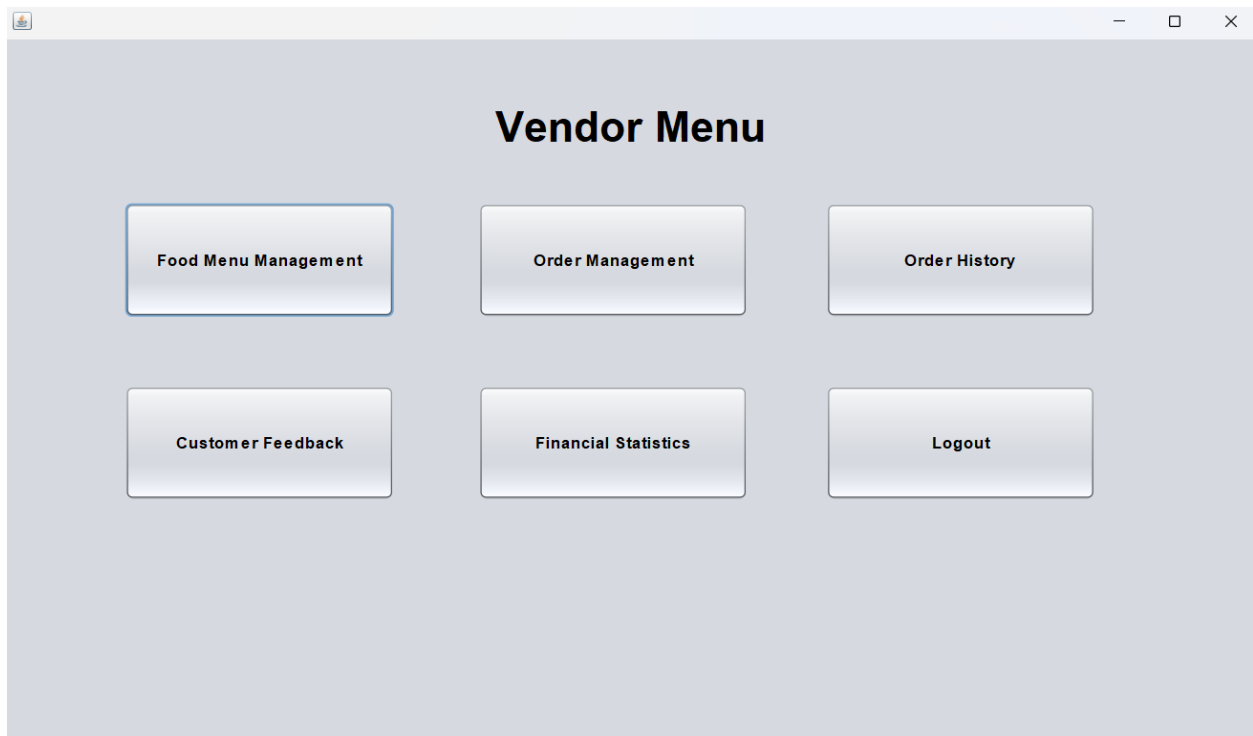


Figure 2.4.1 Vendor Menu

Figure 2.4.1 is the main menu page of vendor role. There were distributed by 6 buttons, which are "Food Menu Management", "Order Management", "Order History", "Customer Feedback", "Revenue Dashboard" and the last button "Logout". Once users access anyone of the buttons, they still could be back to the Vendor Menu page.



## Food Menu Management

Vendor ID

Vendor Name:

Food ID

Food Name:

Price:

Description:

Add

Modify

Delete Food

Clear

Back

## Food Menu

Vendor ID	Vendor Name	Food ID	Food Name	Price	Description
a4	weijun	b4	Duck rice	30	jdkmf
a4	weijun	b4	Chicken rice	30	no rice
a4	weijun	b6	Nasi Kuning	88	Very very spicy
a4	weijun	b6	Nasi Kuning	88	Very very spicy

Figure 2.4.2 Food Menu Management Page

Figure 2.4.2 is the page of food menu management for vendor, it allows vendors to add their own food and drinks easily. And the latest information of the food data will be updated to the food menu table. Hence, this function would be easier for users to check what has been updated on their own menu. Users only need to key in the food data in every text field then press the “Add” button then may see the added food on updated on the menu table. To delete or remove the food item, user just need to select the food item rows then click the “Delete” button the remove the food item rows. Also, to modify the food item data, users still have to select the food item rows they attempt to edit. Then, key in again the latest food item data.

# Order Management Menu

Order ID	Food ID	Food Name	Quantity	Food_Price	Customer Na...	Customer ID	Vendor	Remarks	Runner ID	Runner	Delivery_Price	Status	Cancel By	Date
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	2022-02-05
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	Completed	null	2023-02-05
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Rejected	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Rejected	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	null
O002	F002	Pizza1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null	null

Accept Order

Reject Order

Complete Order

Back

Figure 2.4.3 Order Management Page

Figure 2.4.3 is the page of Order Management Menu. Basically, when the customer side place an order from this vendor, then by right the the order request will be appears on this table by showing all the information: “Order ID”, “Food ID”, “Food Name”, “Quantity”, “Food Price”, “Customer Name”, “Customer ID”, “Vendor”, “Remarks”, “Runner name”, “Delivery Price”, “Status”, “Cancel By” and “Date”. Once vendor sees the order, vendor can select either to “Accept order” or “Reject Order”. Once the vendor accepts the order and completed preparing the food, then vendor may press “Complete Order” to remind runner.

# Order Management Menu

Order ID	Food ID	Food Name	Quantity	Food_Price	Customer Na...	Customer ID	Vendor	Remarks	Runner ID	Runner	Delivery_Price	Status	Cancel By	Date
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Completed	Completed	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Accepted	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Rejected	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	2022-02-07
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	2022-02-06
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Rejected	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Pending	null	2022-02-05
O002	F002	Pizza	1	25.00	Jane Doe	C002	Vendor B	Extra Cheese	R002	Runner Y	5.00	Pending	null	null

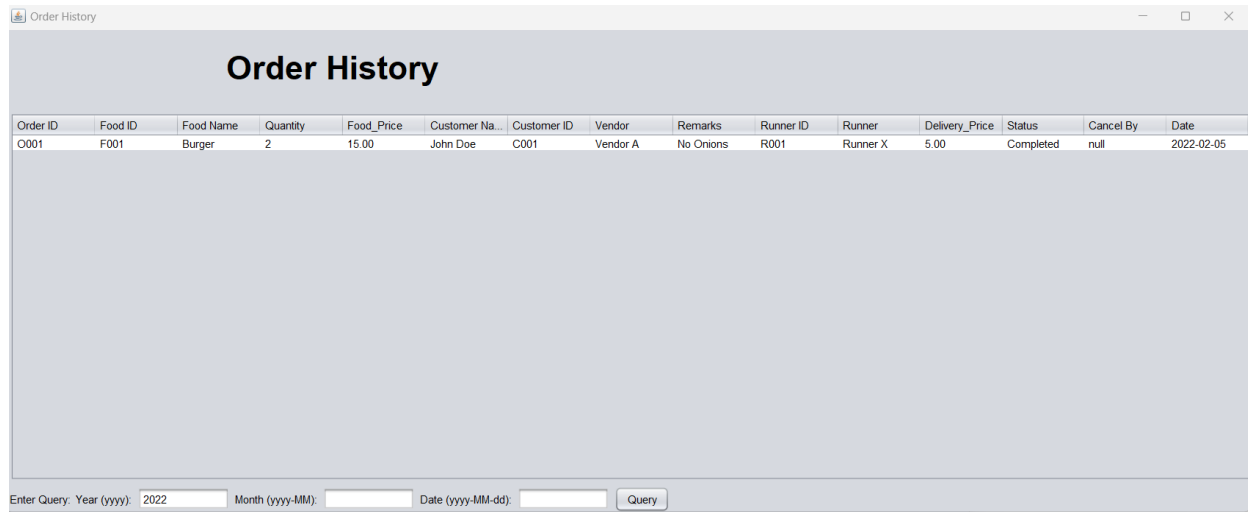
Accept Order

Reject Order

Complete Order

Figure 2.4.4 Order Management Testing

As Figure 2.4.4 shown, I selected the 4<sup>th</sup> row which is still pending. Once I click “Accept Order”, there is a window that pops out to announce the vendor. Or else, the vendor also can reject the order.



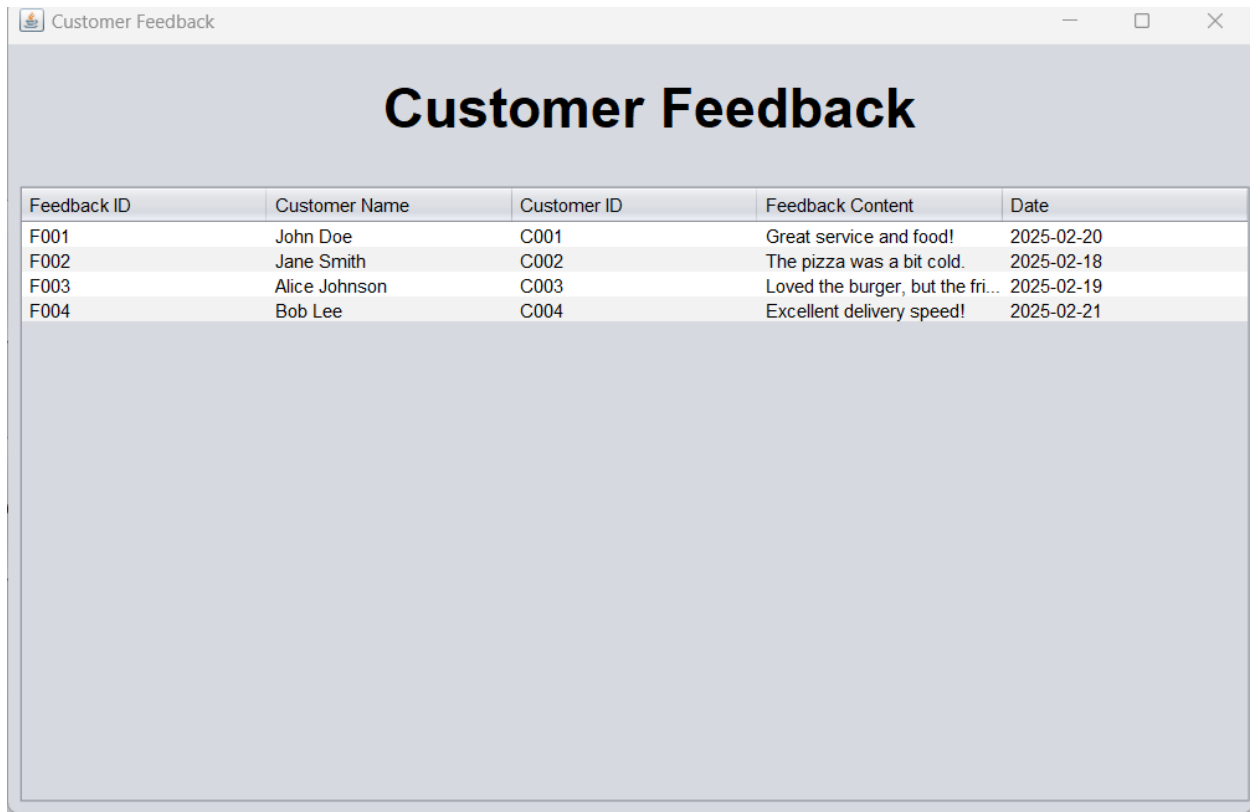
The screenshot shows a web application window titled "Order History". The main content area contains a table with 14 columns: Order ID, Food ID, Food Name, Quantity, Food\_Price, Customer Na..., Customer ID, Vendor, Remarks, Runner ID, Runner, Delivery\_Price, Status, Cancel By, and Date. The first row of data shows Order ID O001, Food ID F001, Food Name Burger, Quantity 2, Food\_Price 15.00, Customer Na... John Doe, Customer ID C001, Vendor Vendor A, Remarks No Onions, Runner ID R001, Runner Runner X, Delivery\_Price 5.00, Status Completed, Cancel By null, and Date 2022-02-05. Below the table is a search filter section with the text "Enter Query: Year (yyyy):" followed by a text input containing "2022", "Month (yyyy-MM):" followed by an empty text input, "Date (yyyy-MM-dd):" followed by an empty text input, and a "Query" button.

Order ID	Food ID	Food Name	Quantity	Food_Price	Customer Na...	Customer ID	Vendor	Remarks	Runner ID	Runner	Delivery_Price	Status	Cancel By	Date
O001	F001	Burger	2	15.00	John Doe	C001	Vendor A	No Onions	R001	Runner X	5.00	Completed	null	2022-02-05

Enter Query: Year (yyyy): 2022 Month (yyyy-MM): Date (yyyy-MM-dd): Query

Figure 2.4.5 Ordered History Page

Figure 2.4.5 is an Order History page. It allows vendors to view all past completed orders. It provides details such as order ID, food details, customer information, vendor name, delivery price, status, and date. Vendors can also filter past orders based on date queries at the bottom side by either by year, month, or day. For example, shown as Figure 2.4.5, I want to track the past order only at the year of “2022”, then it will only show me the past order at the year. This allows vendors to easier to track their order history when someday they need it. The system loads order history from Orders.txt, which keeps every order client places. The order history shows just completed orders.



Feedback ID	Customer Name	Customer ID	Feedback Content	Date
F001	John Doe	C001	Great service and food!	2025-02-20
F002	Jane Smith	C002	The pizza was a bit cold.	2025-02-18
F003	Alice Johnson	C003	Loved the burger, but the fri...	2025-02-19
F004	Bob Lee	C004	Excellent delivery speed!	2025-02-21

Figure 2.4.6 View Customer Feedback Page

Figure 2.4.6 is the page to view customer feedback for the past completed orders. It allows vendors to examine the rating and comment on the food, thereby vendor may improve their food quality, comply with customer's standards and service experience. The "Customer Feedback" page was created by JTable component and read the customer feedback data that stored at "Feedback.txt" and shows the real-time customer feedback on the table.

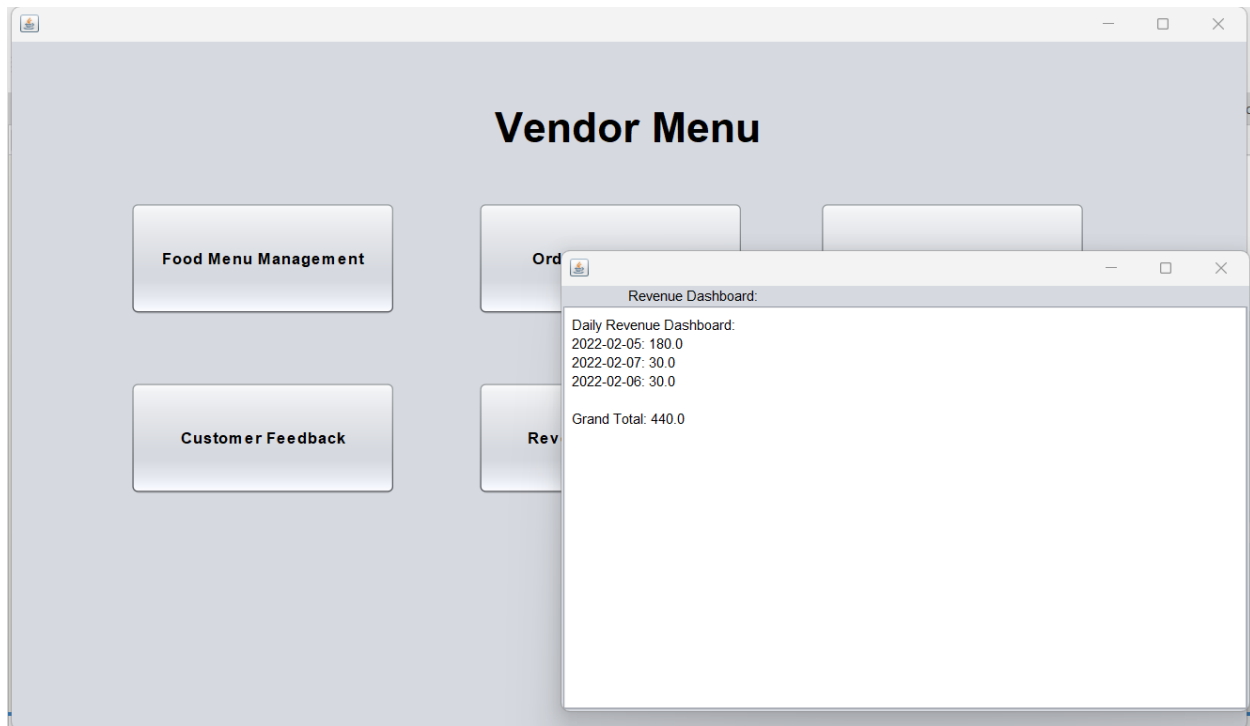


Figure 2.4.7 Revenue Dashboard

Figure 2.4.7 is the page of revenue dashboard of vendor; it allows vendors to track the everyday sales income. This page basically is calculated and shows the total sales income for every day and as well as all income from the orders, this could be helpful for vendors to have a better understanding of the local sales situation and their sales achievements growth. The principle of the dashboard revenue is basically from the orders data that stored at "orders.txt", it consists of all orders detailing data. Hence, we make the revenue dashboard to read the "orders.txt" file and filter the past completed (only the orders status at "completed") orders to calculate the revenue income according to date by date.

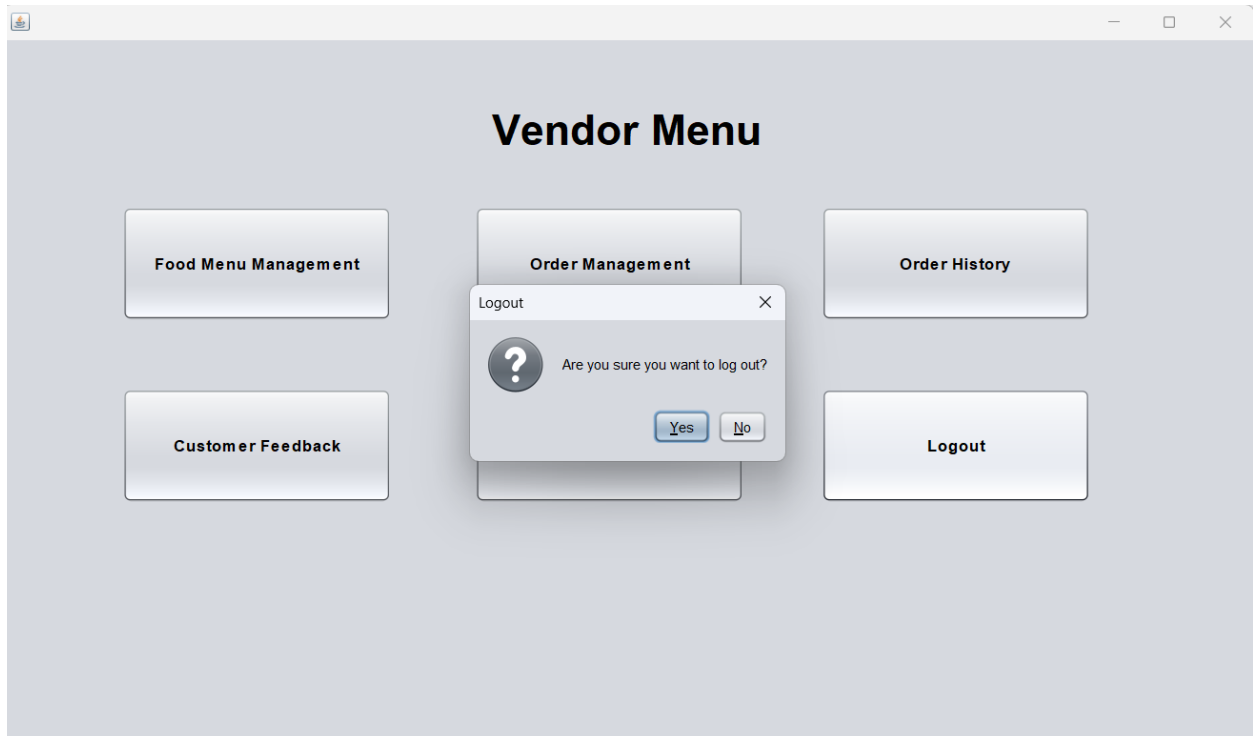


Figure 2.4.8 Log out Page

Figure 2.4.8 is a log out button to bring user left from vendor menu. It will lead to the user back to the user login interface. When users press on the “logout” button, it should pop out a window to ensure with user to logout from this page.

## 2.5 Runner (Choong Ti Shen TP078540)

### 2.5.1 Runner Login

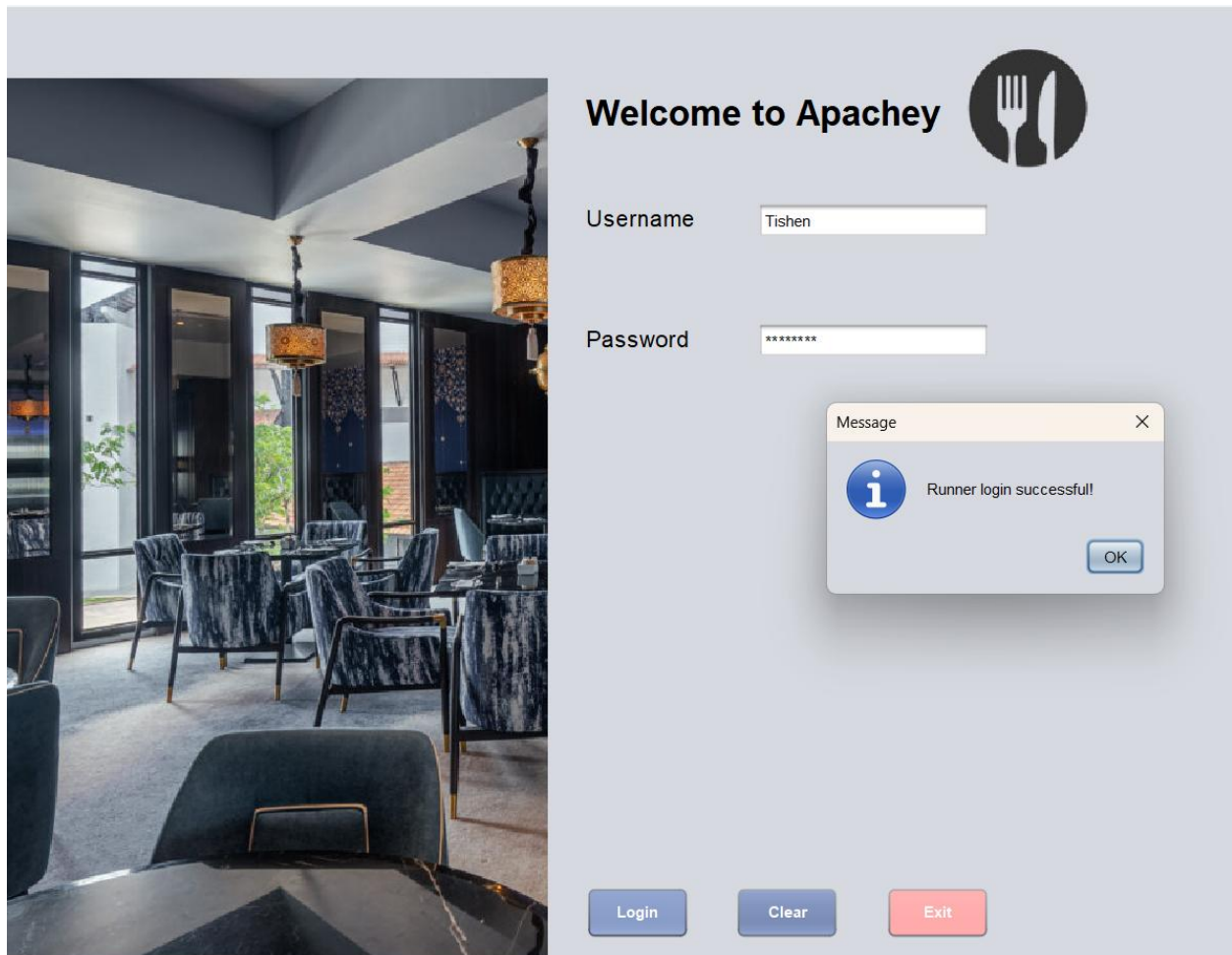


Figure 2.5.1.1 Login Page

When the user Login, if the Username and Password matches the role of Runner, then user is login as runner. The system will then send user to Runner\_Main\_Menu. The username and UserID are stored and encapsulated at Runner with the parameter of Runner\_Main\_Menu(runner).

### 2.5.2 Runner Main Menu

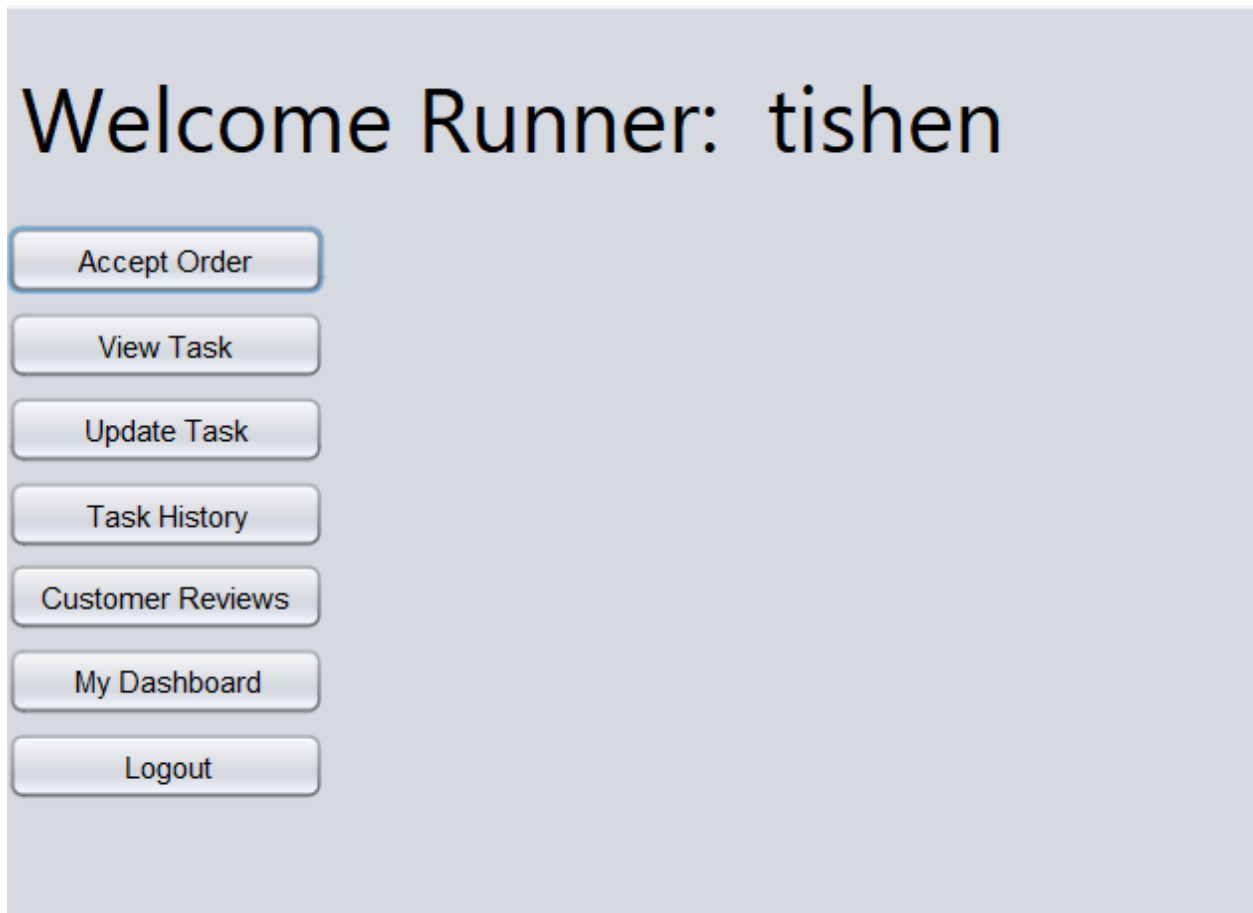
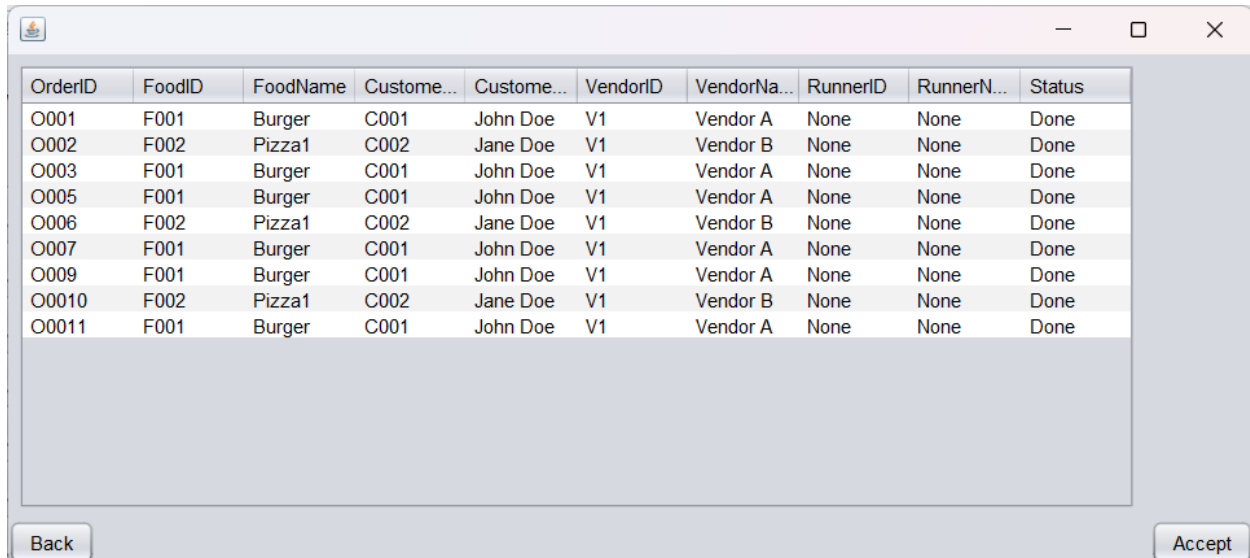


Figure 2.5.2.1: Runner Main Menu

At main menu, it will show: "Welcome Runner:" and the name is according to name used login page. This is done by data encapsulating. The username and userID are parsed to main menu and that username is shown in JLabel by doing `runner.getName()`. There are 7 selections at main menu page, which are Accept Order, View Task, Update Task, Task History, Customer Reviews, My Dashboard and Logout.



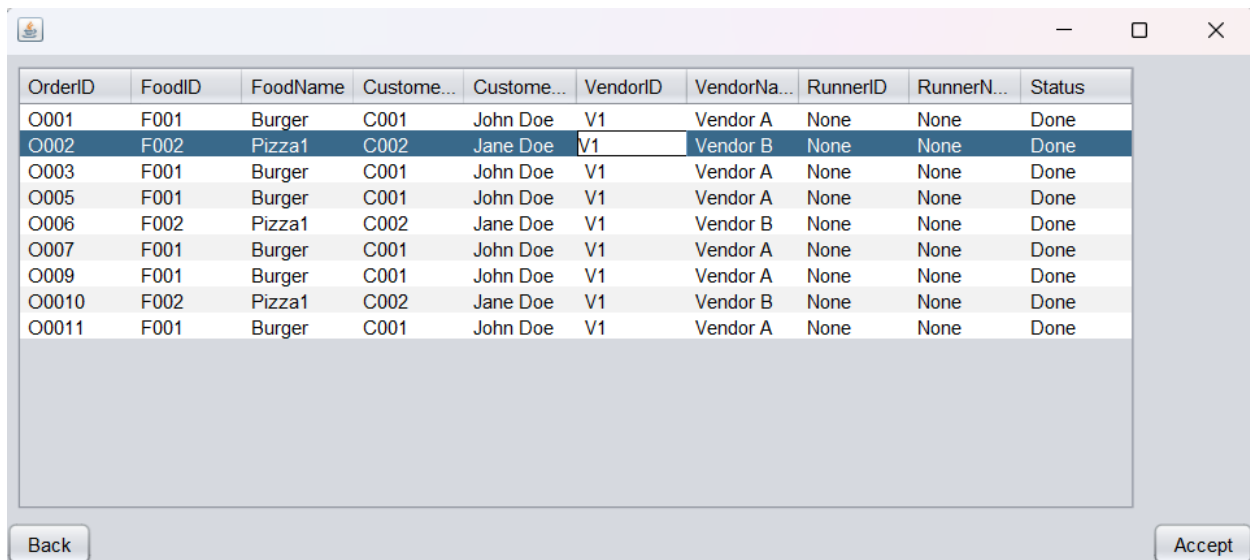
### 2.5.3 Accepting Task



OrderID	FoodID	FoodName	Custome...	Custome...	VendorID	VendorNa...	RunnerID	RunnerN...	Status
O001	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O002	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O003	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O005	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O006	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O007	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O009	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O0010	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O0011	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done

Figure 2.5.3.1: Customer's order to Accept

By clicking Accept Order Button at main menu, the system will send runner to a page where runner can choose to Accept which Order. Those order printed in the table are foods already done prepared by Vendor with status “Done”.



OrderID	FoodID	FoodName	Custome...	Custome...	VendorID	VendorNa...	RunnerID	RunnerN...	Status
O001	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O002	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O003	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O005	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O006	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O007	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O009	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done
O0010	F002	Pizza1	C002	Jane Doe	V1	Vendor B	None	None	Done
O0011	F001	Burger	C001	John Doe	V1	Vendor A	None	None	Done

Figure 2.5.3.2: Accepting Task

Runner can choose which Order to accept as task by selecting the row and click the button Accept. For example, this runner tishen accept OrderID O002 as task.

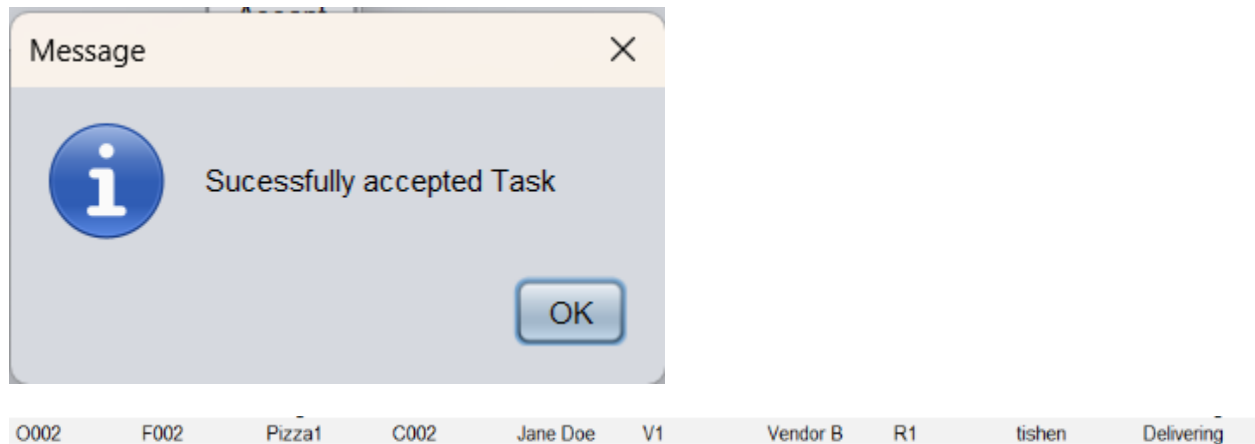


Figure 2.5.3.3: Successful Accepting Task

After accepted the task, the RunnerID and RunnerName will be written to the current runner id and name by using `runner.getName` and `runner.getID`. The status is also changed to Delivering instead of done. The task accepted will appear at View Task function at main menu. Now runner can use that Back Button to return to Main Menu.

#### 2.5.4 Back Button

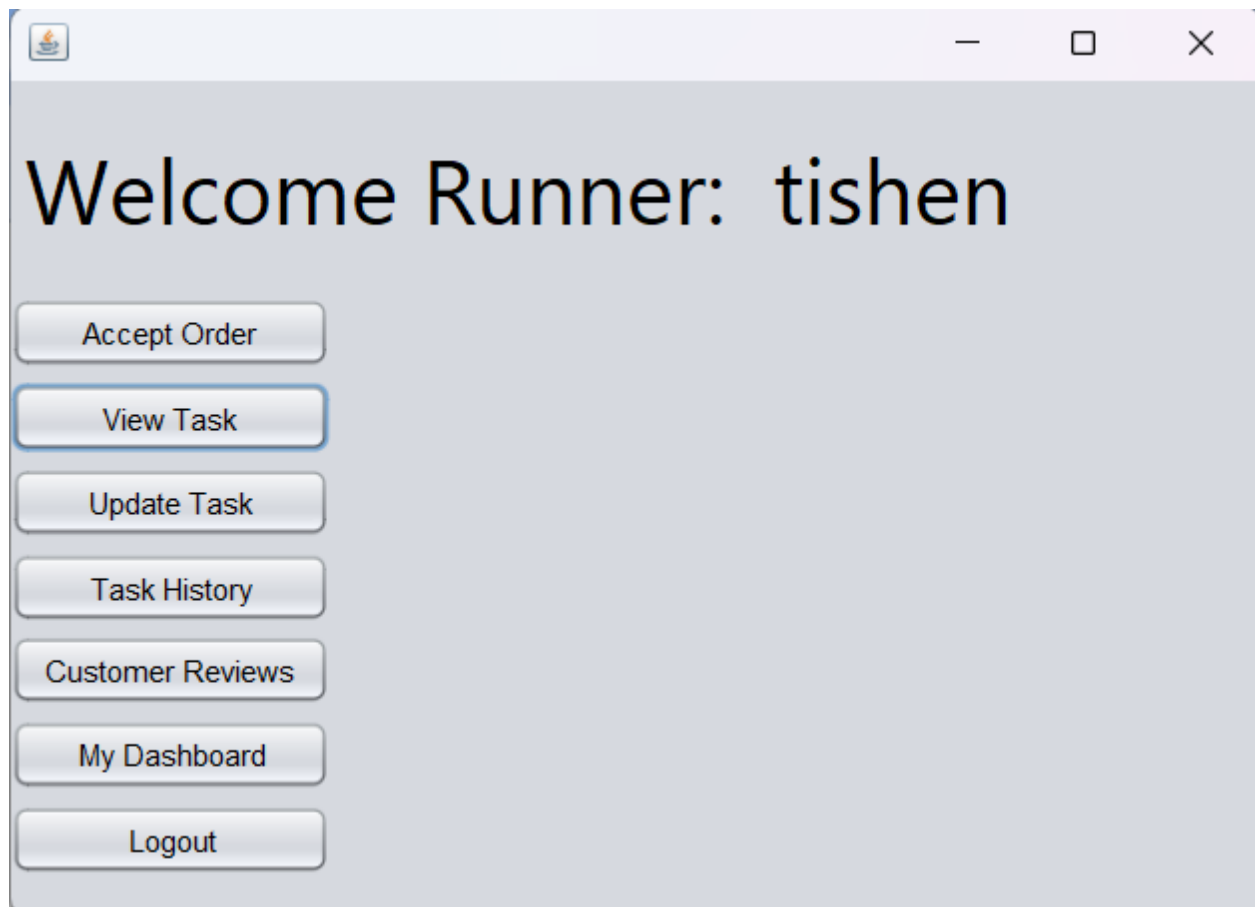
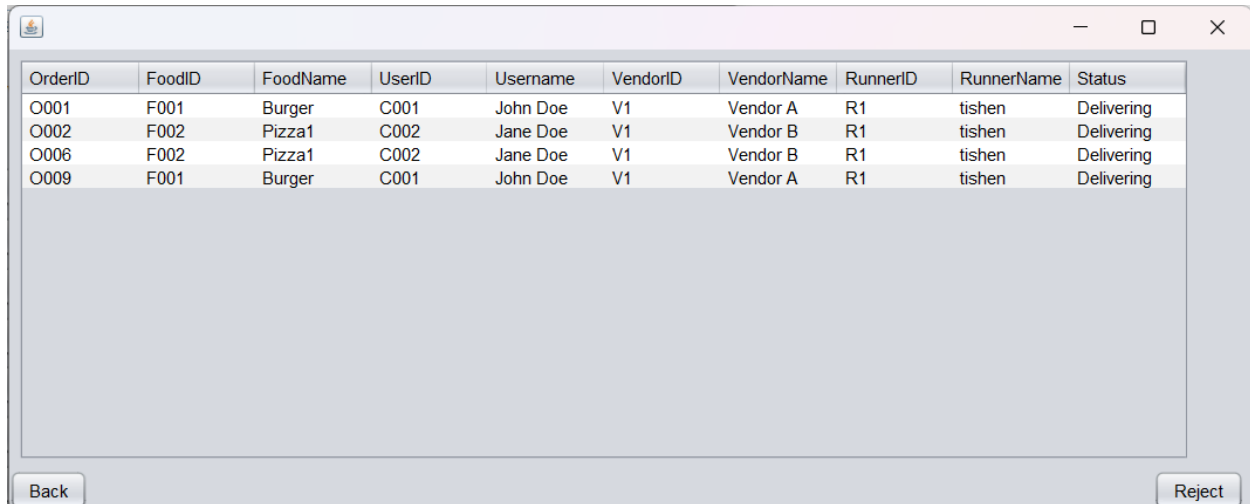


Figure 2.5.4.1 Back Button function

When the runner click on Back Button, the system will send them back to Main Menu. The Back Buttom is implemented in each page of the other functions too.

### 2.5.5 Viewing Task



OrderID	FoodID	FoodName	UserID	Username	VendorID	VendorName	RunnerID	RunnerName	Status
O001	F001	Burger	C001	John Doe	V1	Vendor A	R1	tishen	Delivering
O002	F002	Pizza1	C002	Jane Doe	V1	Vendor B	R1	tishen	Delivering
O006	F002	Pizza1	C002	Jane Doe	V1	Vendor B	R1	tishen	Delivering
O009	F001	Burger	C001	John Doe	V1	Vendor A	R1	tishen	Delivering

Figure 2.5.5.1: Viewing Task

When the user click on Button View Task at main menu, it will send them to a page with task they accepted. For example, runner tishen accepted 4 tasks, then the table will shows the detail of those 4 tasks which include OrderID, FoodID, FoodName to prevent runner taking wrong food from vendor. VendorID, VendorName are also printed so that runner can track their shop. UserID and UserName and customer details so that runner know who is the customer ordering the food and where to send. Runner can choose to reject the task too by selecting the row and click reject button.

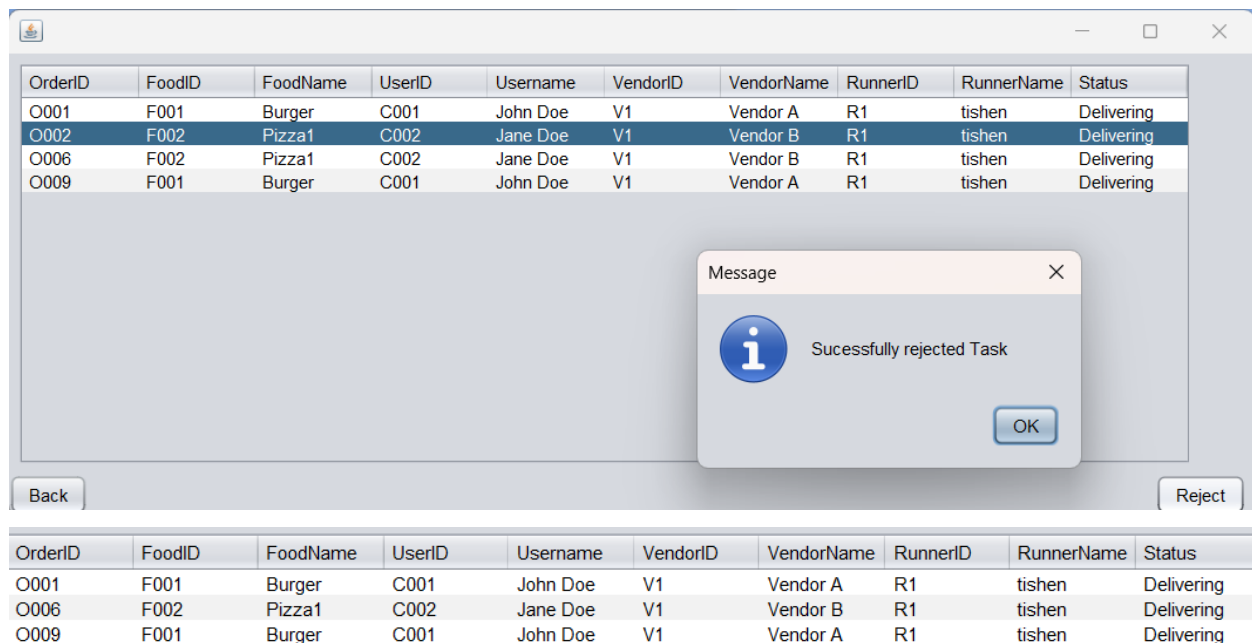


Figure 2.5.5.2 Successfully Rejecting Task

After the task successfully rejected, system will show that message 'Successfully rejected Task'. After that, the row of data of order rejected are also removed.

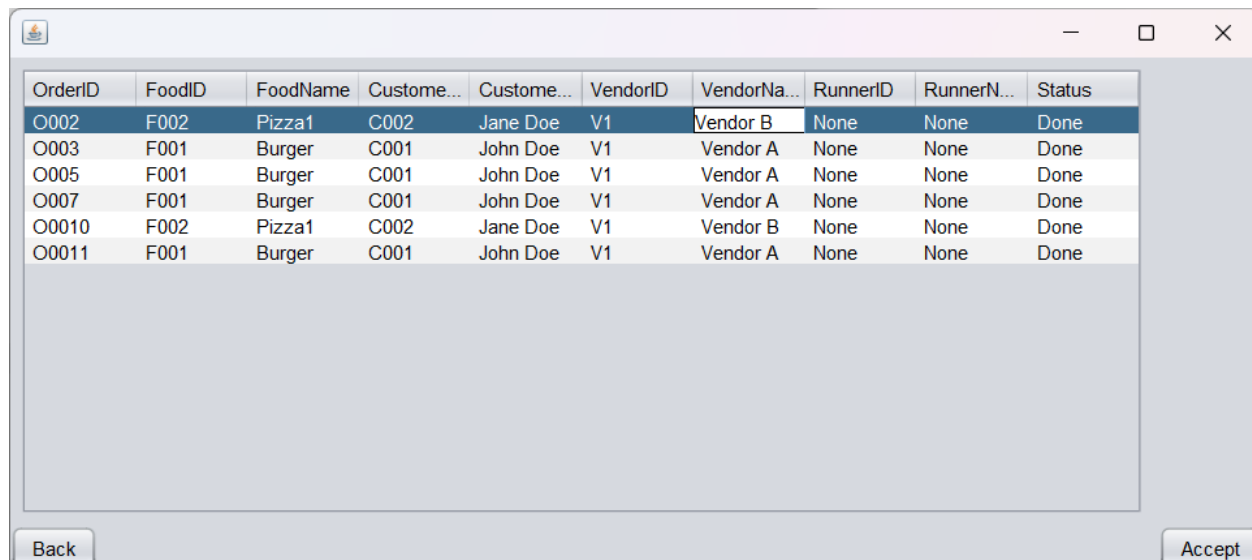


Figure 2.5.5.3: Output of Order Rejected at Order list.

After rejecting a task, that task will appear back at the order list for another runner to accept that task. Changing RunnerID and RunnerName back to None and status back to Done.

## 2.5.6 Updating Task

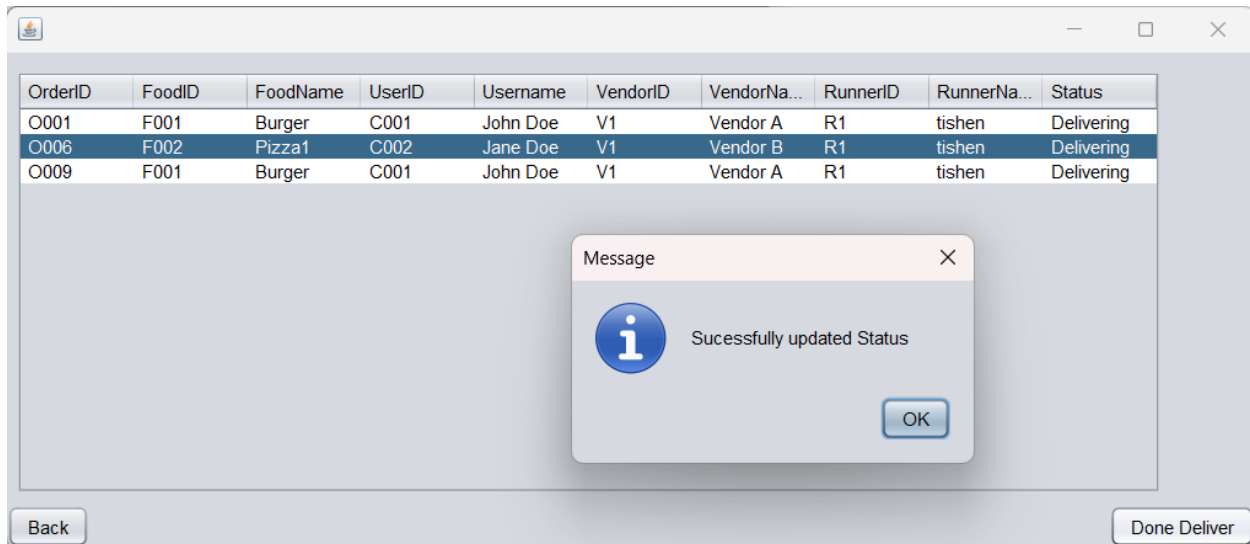


Figure 2.5.6.1: Sucessfully updating Task

After runner successfully send that Order to Customer, they can click on Update Task button at main menu to update the task status. Same as the viewing task table do, but this time it appears done button. Runner just click on the data on the task list and press Done Deliver button. Then the status will be sucessfully updated and the work of Runner is currently over.

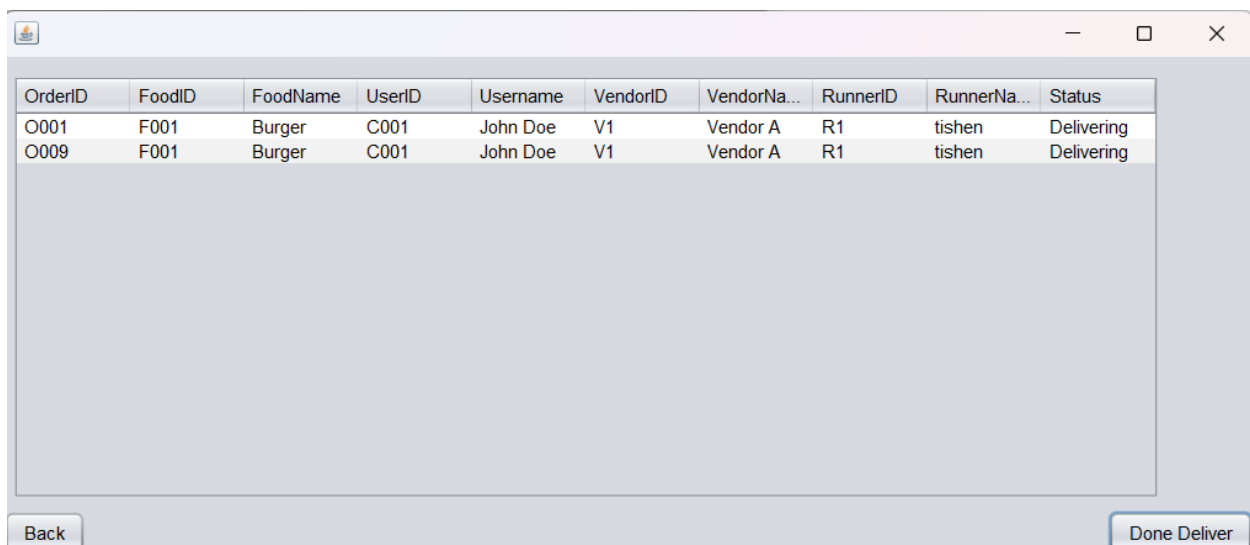


Figure 2.5.6.2: Table after updating Task Status.

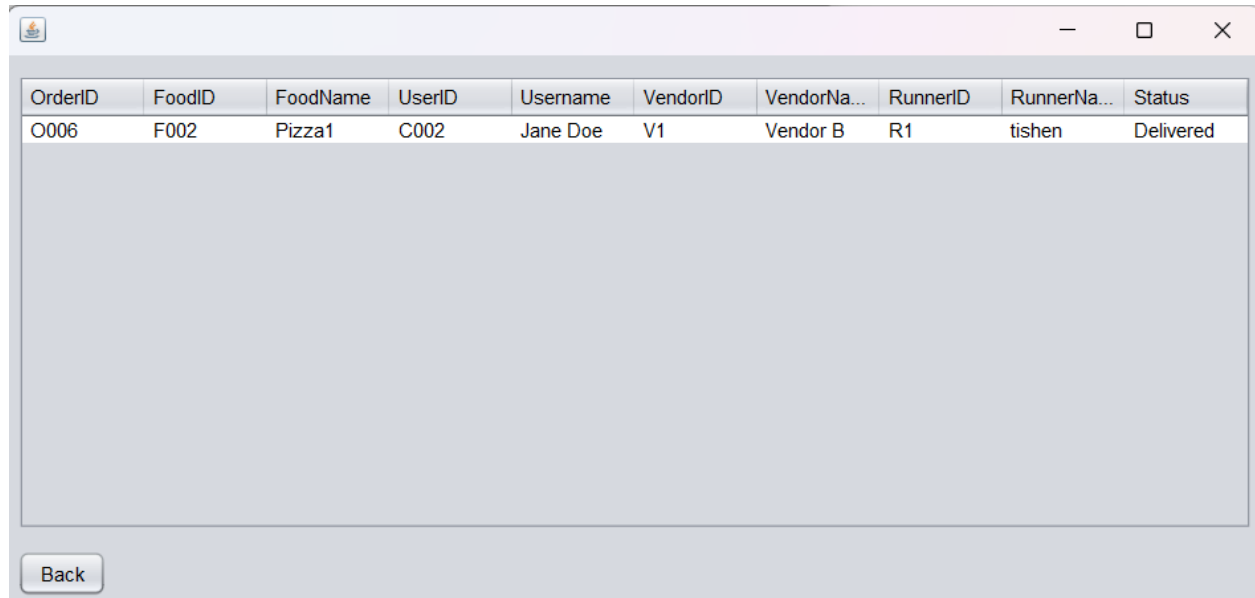
After runner successfully done the Task, that Task will disappear on the table. It can be view by that runner at the function view Task History.

C001;John Doe;-5.00;2022-02-05
R1; <u>tishen</u> ;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1; <u>tishen</u> ;5.00;2023-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1; <u>tishen</u> ;5.00;2022-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1; <u>tishen</u> ;5.00;2022-02-05
R1; <u>tishen</u> ;-5.00;2022-02-05
R1; <u>tishen</u> ;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1; <u>tishen</u> ;5.00;2023-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1; <u>tishen</u> ;5.00;2023-02-05

Figure 2.5.6.3: Updating Transactions of the whole Order.

Since the order is now done, that money is no longer holded by the system. The money that paid by Customer now transfered to Vendor and Runner with their name as recipients of add amount, and Customer with their name of reduct amount.

### 2.5.7 View Task History



OrderID	FoodID	FoodName	UserID	Username	VendorID	VendorNa...	RunnerID	RunnerNa...	Status
O006	F002	Pizza1	C002	Jane Doe	V1	Vendor B	R1	tishen	Delivered

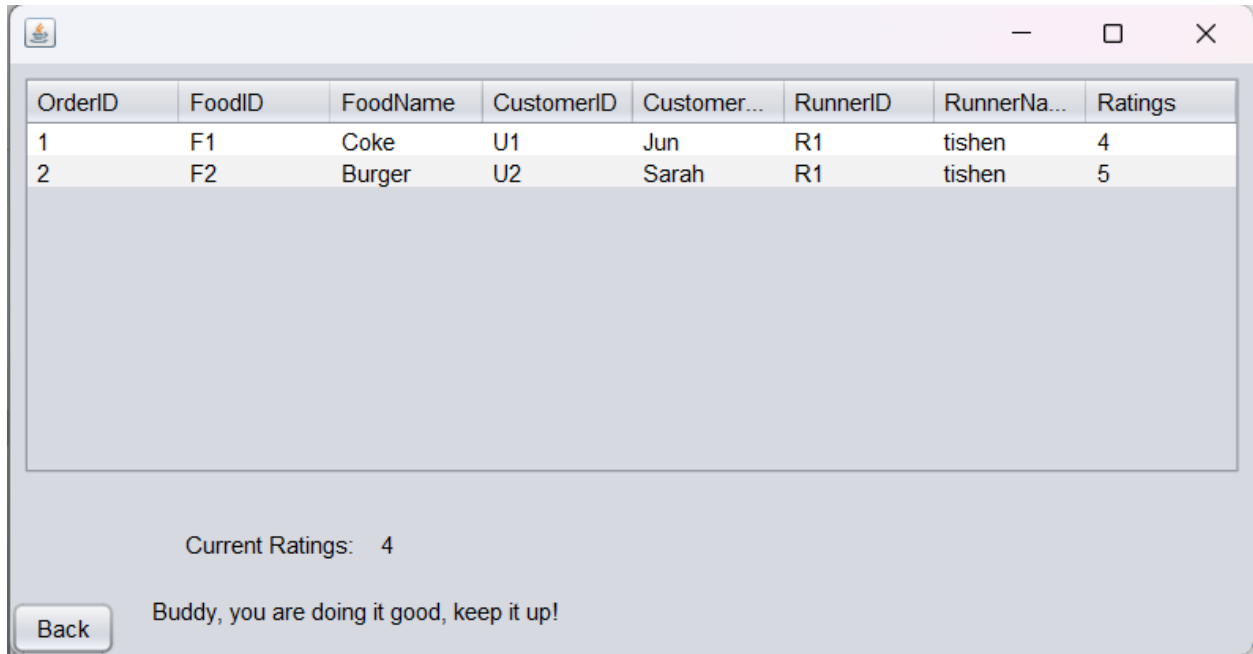
Back

Figure 2.5.7.1: View Task History

Runner can view their completed task at here. It includes the status delivered which is changed from delivering after runner updated that Task Status. This makes manager and customer able to trackdown that order status too.



### 2.5.8 View Customer Reviews



OrderID	FoodID	FoodName	CustomerID	Customer...	RunnerID	RunnerNa...	Ratings
1	F1	Coke	U1	Jun	R1	tishen	4
2	F2	Burger	U2	Sarah	R1	tishen	5

Current Ratings: 4

Buddy, you are doing it good, keep it up!

Back

Figure 2.5.8.1: View Customer Reviews

At the main menu page, runner can view their customer reviews by clicking Customer Reviews Button. The system will show all previous Ratings made by customer, rating from 1-5 stars where 1 is lowest and 5 is highest. The system will calculate their average ratings and update at the line current ratings: too. For each average rating level, system prints message according to it to guide how runner should do to improve their rating.

OrderID	Foodid	food name	User ID	Username	Runner Id	Runner	Review
1	F1	Coke	U1	Jun	R1	tishen	4
2	F2	Burger	U2	Sarah	R1	tishen	5
3	F3	Pizza	U3	David	R6	Sam	3
4	F4	Fries	U4	Emily	R9	John	2
5	F5	Sandwich	U5	Michael	R12	Jacob	5
6	F5	Sandwich	U5	Michael	R12	Jacob	2

Figure 2.5.8.2: Page of Customer Review Records

Since there are many reviews made by customer for different runner, the system sorts out the runner name according by using

```
Review_RunnerID.equalsIgnoreCase(runner.getID) &&  
Review_RunnerName.equalsIgnoreCase(runner.getName)
```

So that runner view their reviews only.

### 2.5.9 Runner Dashboard

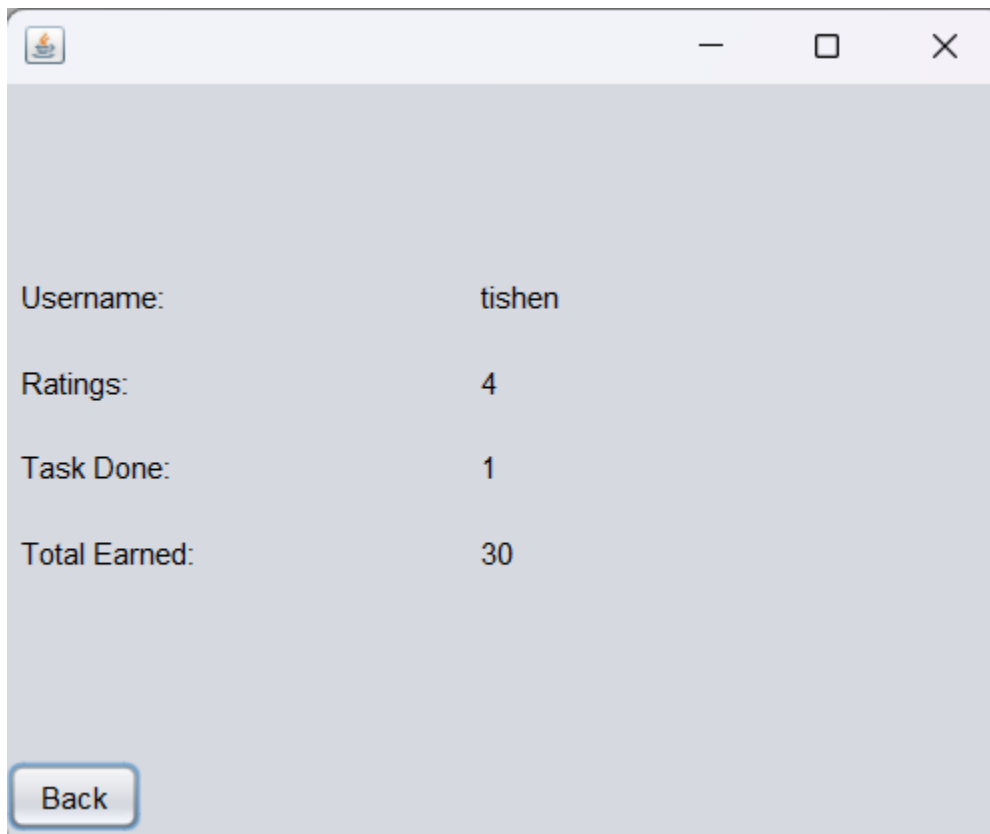


Figure 2.5.9.1: Runner Dashboard

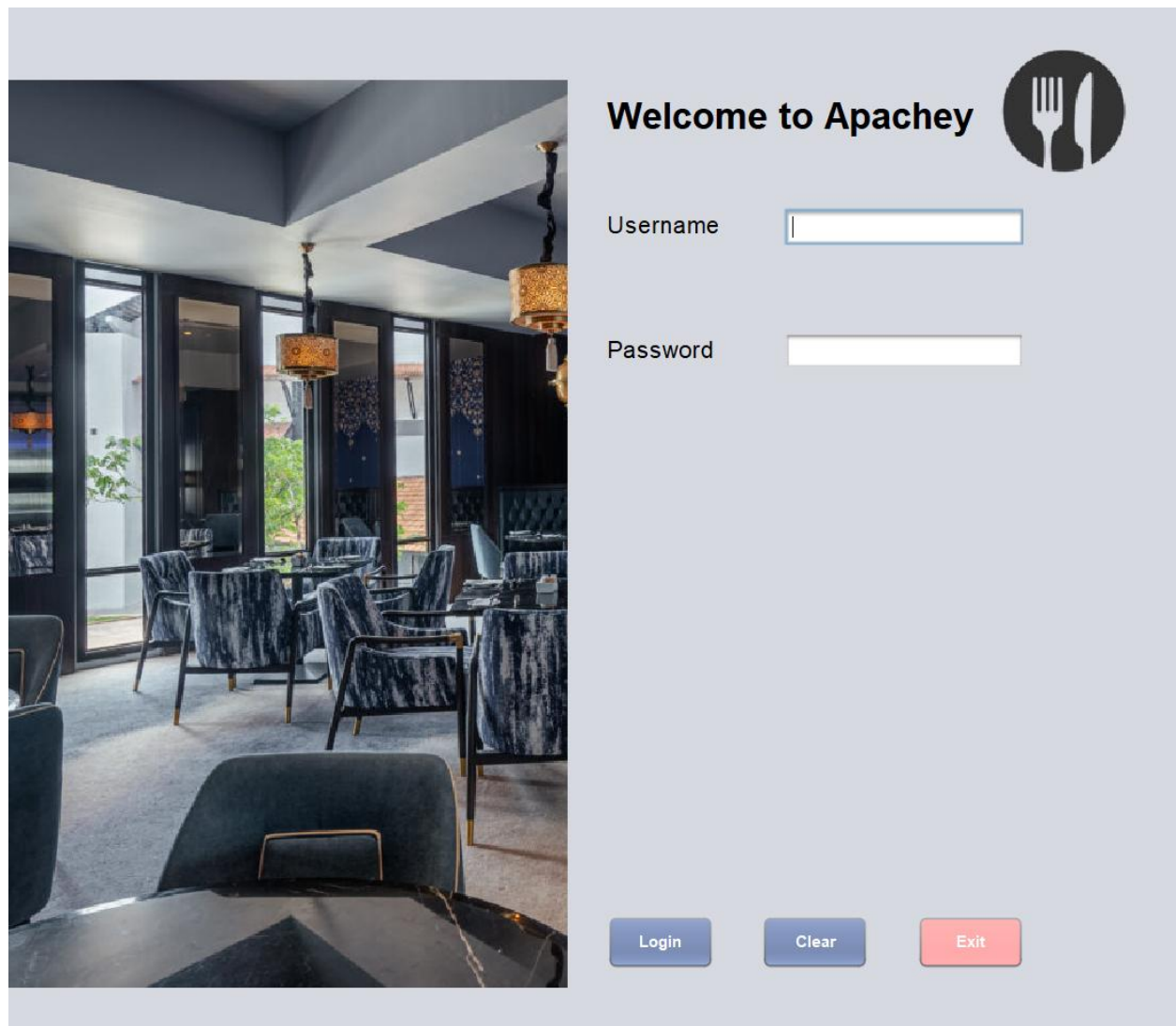
When runner view Runner Dashboard, system will prints username of runner, average ratings stars which have the same function from view customer records, total tasks done and total money earned.

```
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
R1;tishen;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
```

Figure 2.5.9.2: Transaction History Calculator

To test out the function calculate total earned by runner, I added positive and negative values. The system sorts by runner name and add all the amount in positive and negative values.

### 2.5.10 Logout



**Welcome to Apachey**

Username

Password

[Login](#) [Clear](#) [Exit](#)

Figure 2.5.10: Runner Logout

By clicking on Logout Button at main menu, the system sent the Runner back to login page. The whole session for Runner functions are done.

## 2.6 Customer (Phang Sieow Jun TP072541)

### 2.6.1 Main menu

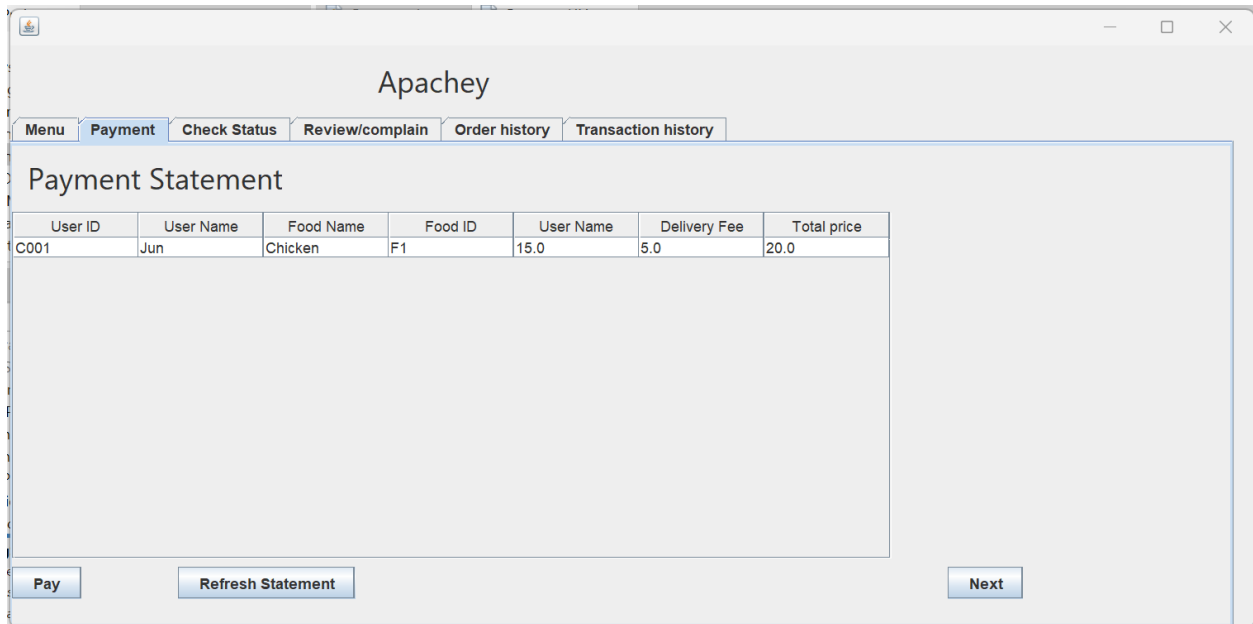
The screenshot shows a web application titled "Apachey". At the top, there is a horizontal tabbed menu with six tabs: "Menu", "Payment", "Check Status", "Review/complain", "Order history", and "Transaction history". The "Menu" tab is currently selected. Below the tabs, the interface is divided into two main sections. On the left, under the heading "Food list", there is a table with three columns: "Food ID", "Food N...", and "Review". The table contains four rows of data: F1 (Chicken, 15), F2 (Burger, 10), F3 (Pizza, 20), and F4 (Sushi, 25). Below this table is a text input field labeled "Enter the Food ID" and two buttons, "Add" and "Cancel". On the right, under the heading "Order list", there is a table with four columns: "Food ID", "Food Name", "Total price", and "Delivery price". This table is currently empty. Below the "Order list" table are two buttons: "Renew Order list" and "Next".

Food ID	Food N...	Review
F1	Chicken	15
F2	Burger	10
F3	Pizza	20
F4	Sushi	25

Food ID	Food Name	Total price	Delivery price
---------	-----------	-------------	----------------

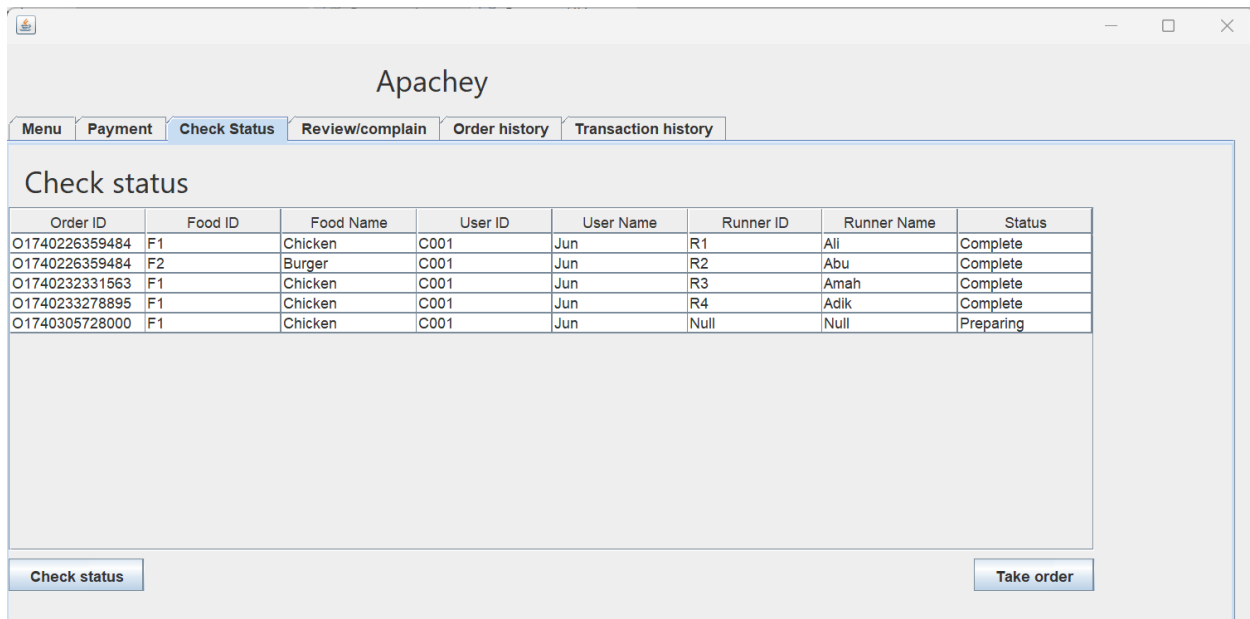
A food ordering system provides access to customers through customer user interface, which is their main interactive platform. Through the tabbed menu at the top, users can access the menu, payment, check status, view/complaint, order history and transaction history. The menu tab in the provided picture shows a structured platform on which users can view all available foods, place orders and manage their choices at the same time. The food list on the left panel shows the food items read by the Food\_Menu.txt file, along with the food ID and food name columns and comment scores. Users need to write their food ID into the text field, and then press the "Add" button, and the order will appear in the order list displayed on the right. To cancel the input, the user can access the Cancel button. Users can get the latest order list by clicking the "Update Order List" button. The convenient interface enables users to order smoothly, and they can effectively manage menu selection and place orders through user-friendly interaction.

## 2.6.2 Payment Statement



The CustomerUI Payment Statement page displays the order information and a summary of the payment to the customer. The table on this tab shows important transaction information, including user ID, username, food name, food ID, freight and total price. The system retrieves the displayed data from its database and text file to ensure the correct tracking of customer orders. Click "Pay" and the user can start the payment process for the selected order, thus completing the transaction. Users can click the "Refresh Statement" button to realize the instant screen update of the order. Users can go to the next section through the "Next" button, which will optimize the ordering process from beginning to end. This interface provides a structured platform for customers to view payment information, thus realizing an easy checkout process.

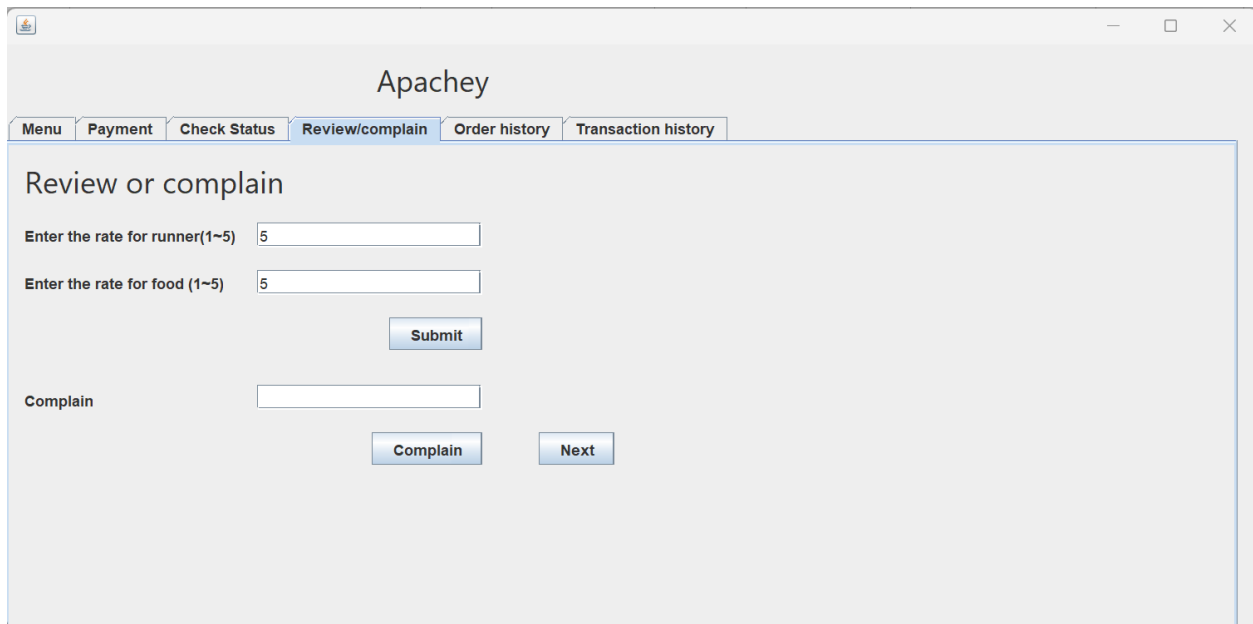
### 2.6.3 Check Status Interface



By checking the status in CustomerUI, users can directly check the progress of their food order, and it will go through different stages. The table shows all the basic information of the order, including order id, food id, food name, user id and name, and operator id, name and status. The status label determines the progress of the order between "completion" and "preparation" and all other processing stages. Transparency appears in the interface because it reveals which delivery person (operator) handles each delivery task. Users can view the updated order information by clicking the Check Status button, which also provides them with an Accept Order button to perform status-related processing operations. The system improves customer satisfaction, because users can know their purchase status and shipment progress at any time.



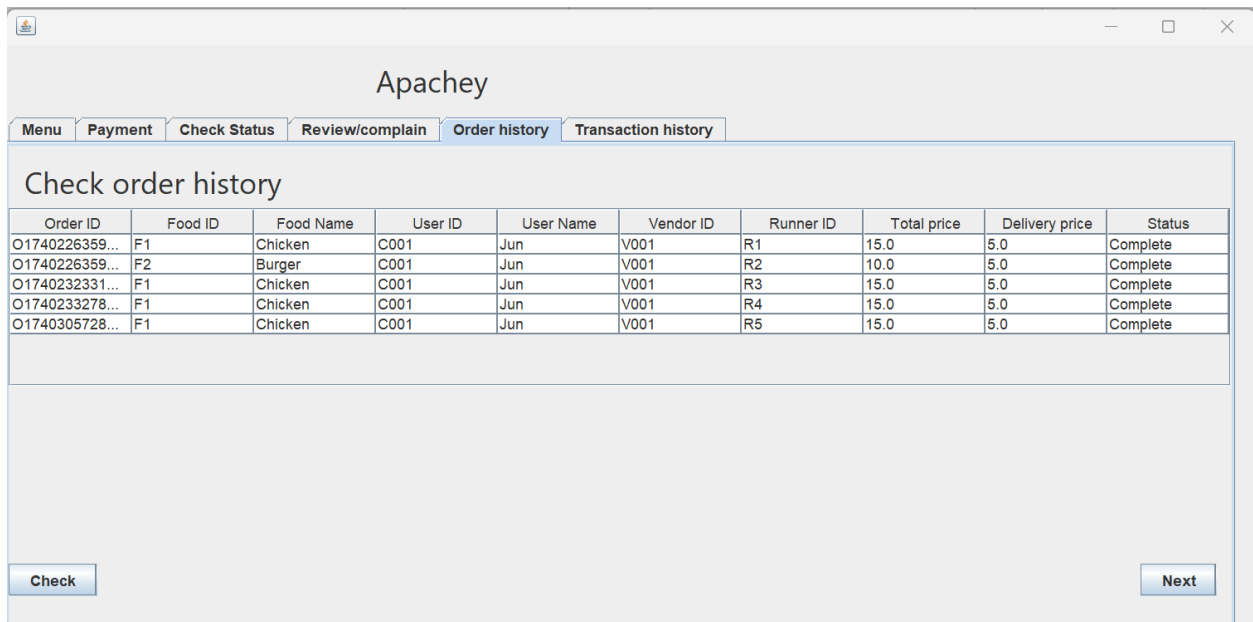
## 2.6.4 Review/Complain Interface



The screenshot shows a web application window titled "Apachey". At the top, there is a navigation bar with five tabs: "Menu", "Payment", "Check Status", "Review/complain" (which is currently selected), "Order history", and "Transaction history". Below the navigation bar, the main content area is titled "Review or complain". It contains two rating sections. The first section is labeled "Enter the rate for runner(1~5)" and has a text input field containing the number "5". The second section is labeled "Enter the rate for food (1~5)" and also has a text input field containing the number "5". Below these two sections is a blue "Submit" button. Further down, there is a section labeled "Complain" with a text input field. Below this field are two buttons: a blue "Complain" button and a blue "Next" button.

Through the comments or complaints section of CustomerUI, users can share their opinions on food orders and experience of delivery service. In this interface, customers use two scoring fields, from 1 to 5, to evaluate the food delivery personnel and their food delivery products. The rating that full users must enter is submitted when they press the submit button. Users can submit a complaint through the text field before using the Complaint button to submit a report. The system helps users feel satisfied, because it allows them to send feedback and complaints to support higher quality services. Click the "Next" button, and the user can enter the subsequent interface of the system.

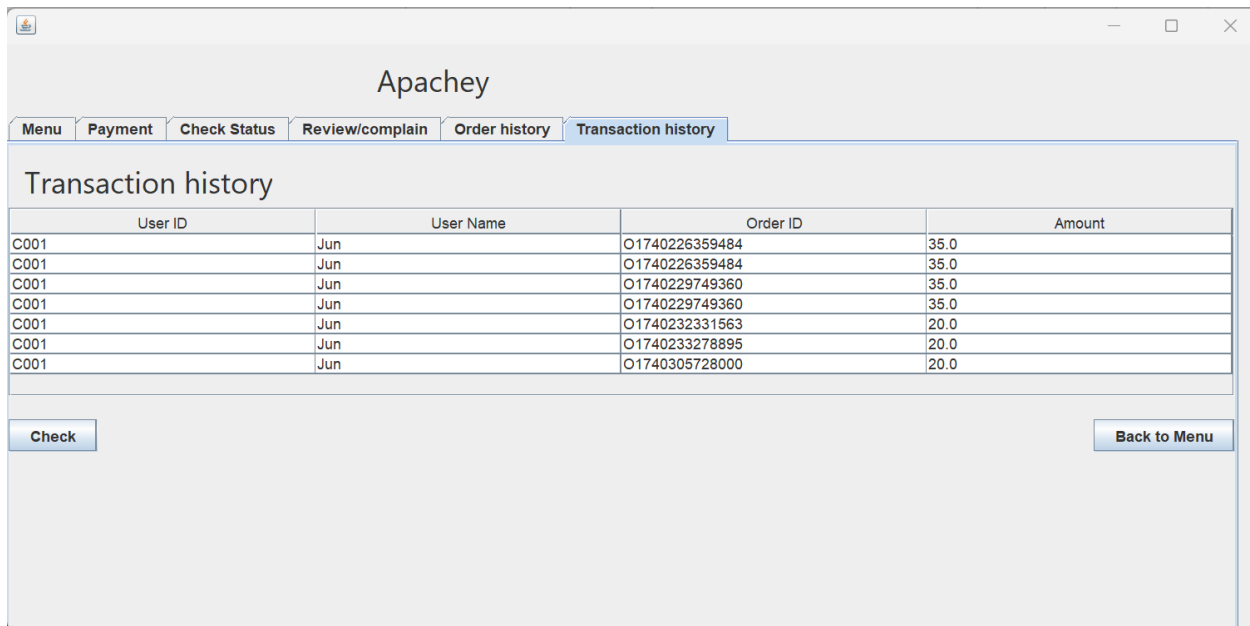
## 2.6.5 Order history Interface



Order ID	Food ID	Food Name	User ID	User Name	Vendor ID	Runner ID	Total price	Delivery price	Status
O1740226359...	F1	Chicken	C001	Jun	V001	R1	15.0	5.0	Complete
O1740226359...	F2	Burger	C001	Jun	V001	R2	10.0	5.0	Complete
O1740232331...	F1	Chicken	C001	Jun	V001	R3	15.0	5.0	Complete
O1740233278...	F1	Chicken	C001	Jun	V001	R4	15.0	5.0	Complete
O1740305728...	F1	Chicken	C001	Jun	V001	R5	15.0	5.0	Complete

Through the order history section of CustomerUI, users can monitor their historical orders through organized tabular display. The information listed in the table includes order ID, food ID, food name, user ID, username, supplier ID, runner ID, total price, delivery price and order status. Through this interface, customers can clearly monitor their past orders, verify their details and check the delivery status. Users can view the latest updates by pressing the "Check" button to update their history. Users can enter the following system parts by pressing the "Next" button. Through the order history function, customers can fully understand and understand all their past transactions, and it is easier to view them.

## 2.6.1 Transaction history Interface



User ID	User Name	Order ID	Amount
C001	Jun	O1740226359484	35.0
C001	Jun	O1740226359484	35.0
C001	Jun	O1740229749360	35.0
C001	Jun	O1740229749360	35.0
C001	Jun	O1740232331563	20.0
C001	Jun	O1740233278895	20.0
C001	Jun	O1740305728000	20.0

The CustomerUI transaction history section shows customers a complete collection of their past purchases. The table shows four important fields, including user ID and user name, as well as order ID and the amount spent per transaction. Through this function, customers can check their food order payments and fees, which also allows them to keep detailed transaction records. The Check button in CustomerUI updates the transaction history display to show the current transaction. Users can navigate through the whole system through the "Back to Menu" button and return to the main menu interface. The system interface improves the user experience by clearly displaying the purchase cost.

## 3.0 Description and justification of Object-oriented concepts incorporated into the solution

### 3.1 Administrator (Eshchanov Umidjon TP071568)

#### 3.1.1 Concept 1 (Encapsulation):

```
11  /*
12  *
13  * public class User implements Serializable{
14  *     private String name;
15  *     private String userId;
16  *     private String email;
17  *     private String username;
18  *     private String password;
19  *     private String role;
20  *
21  *     // Constructor
22  *     public User(String name, String userId, String email, String username, String password, String role) {
23  *         this.name = name;
24  *         this.userId = userId;
25  *         this.email = email;
26  *         this.username = username;
27  *         this.password = password;
28  *         this.role = role;
29  *     }
30  *
31  *     // Getters
32  *     public String getName() { return name; }
33  *     public String getUserId() { return userId; }
34  *     public String getEmail() { return email; }
35  *     public String getUsername() { return username; }
36  *     public String getPassword() { return password; }
37  *     public String getRole() { return role; }
38  *
39  *     // Setters
40  *     public void setName(String name) { this.name = name; }
41  *     public void setUserId(String userId) { this.userId = userId; }
42  *     public void setEmail(String email) { this.email = email; }
43  *     public void setUsername(String username) { this.username = username; }
44  *     public void setPassword(String password) { this.password = password; }
45  *     public void setRole(String role) { this.role = role; }
46  *
47  *     // Convert user details to a string format for file storage
48  *     public String toFileString() {
49  *         return name + "," + userId + "," + email + "," + username + "," + password + "," + role;
50  *     }
51  *
52  *     // Convert a stored line back into a User object
53  *     public static User fromFileString(String data) {
54  *         String[] fields = data.split(",");
55  *         if (fields.length == 6) {
56  *             return new User(fields[0], fields[1], fields[2], fields[3], fields[4], fields[5]);
57  *         }
58  *         return null;
59  *     }
60  * }
```

Figure 3.1.1.1 User.java class

This project uses encapsulation to safeguard user information and guarantee restricted access to class properties. In order to avoid direct alteration from outside the class, the User class contains private attributes like name, userId, email, username, password, and role. Rather, these data are updated and retrieved using getter and setter methods. This guarantees that validation requirements are followed before data is altered and stops unwanted changes. For instance, the system makes sure that the email address ends with @gmail.com and verifies that the username and User ID are unique when a new user is registered. An error message is shown if certain requirements are not fulfilled, preventing the storage of erroneous data. The AdminBackend class, which manages file operations like reading and writing to users.txt, also uses encapsulation. By doing this, UI classes are unable to directly alter data files, improving concern separation and preserving data integrity.

### 3.1.2 Concept 2 (Inheritance):

```
10
11  /**
12   *
13   * @author Umid
14   */
15  public class Customer extends User {
16      private double walletBalance;
17      private static final String WALLET_FILE = "wallets.txt";
18
19      public Customer(String name, String userId, String email, String username, String password, double walletBalance) {
20          super(name, userId, email, username, password, "Customer");
21          this.walletBalance = walletBalance;
22      }
23
24      // Get balance
25      public double getWalletBalance() { return walletBalance; }
26
```

Figure 3.1.2.1 Customer.java class

The application uses inheritance to create a hierarchical structure among various user categories and remove repetitive code. The foundation class is the User class, which has common characteristics like username, password, email, user ID, and name. User is extended by the Customer, Vendor, and DeliveryRunner classes, which inherit these properties and permit particular changes for each role. The Vendor and DeliveryRunner classes are just extensions of User, while the Customer class adds a new attribute called walletBalance that allows wallet-related transactions. The software may describe common actions in one place thanks to this

structure, which also gives role-specific attributes and methods flexibility. The program is easier to maintain and expand in the future because it avoids code duplication and adheres to the notion of code reusability by using inheritance.

### 3.1.3 Concept 3 (Polymorphism):

```
26
27 // Top up balance
28 public void topUp(double amount) {
29     if (amount > 0) {
30         this.walletBalance += amount;
31         updateWalletFile();
32     }
33 }
34
35 // Save customer wallet to wallets.txt (called during registration)
36 public void saveWalletToFile() {
37     try (BufferedWriter writer = new BufferedWriter(new FileWriter(WALLET_FILE, true))) {
38         writer.write(getUsername() + "," + getUserId() + "," + walletBalance);
39         writer.newLine();
40     } catch (IOException e) {
41         System.out.println("Error writing to wallet file: " + e.getMessage());
42     }
43 }
44
45 // Read balance from file
46 public static double getBalanceFromFile(String username) {
47     try (BufferedReader reader = new BufferedReader(new FileReader(WALLET_FILE))) {
48         String line;
49         while ((line = reader.readLine()) != null) {
50             String[] data = line.split(",");
51             if (data.length == 3 && data[0].equals(username)) {
52                 return Double.parseDouble(data[2]);
53             }
54         }
55     } catch (IOException e) {
56         System.out.println("Error reading wallet file: " + e.getMessage());
57     }
58     return -1; // Return -1 if user not found
59 }
60 }
```

Figure 3.1.3.1 Customer.java class

Method overriding in the Customer class, where the topUp() method modifies the customer's wallet balance, serves as an example of polymorphism. The Customer class adds functionality by permitting changes to the wallet balance, while the User class offers generic properties. The topUp() method makes sure that the balance is updated and kept in wallets.txt whenever an administrator performs a top-up. This demonstrates polymorphism by enabling the same User

object to act differently depending on whether it is a DeliveryRunner, Vendor, or Customer. In order to maintain data consistency across several storage files, the updateWalletUsername() method further modifies the username in wallets.txt if a customer's username is changed. The system can dynamically modify its behavior according to the type of user being handled by utilizing method overriding.

## 3.2 Manager (Mohammed Ashraf TP078045)

### 3.2.1 Concept 1 (Encapsulation):

```
3  public class Vendor {  
4      private String vendorID;  
5      private String vendorName;  
6      private String revenue;  
7      // Constructor  
8      public Vendor(String vendorID, String vendorName, String revenue) {  
9          this.vendorID = vendorID;  
10         this.vendorName = vendorName;  
11         this.revenue = revenue;  
12     }  
13     // Getters  
14     public String getVendorID() {  
15         return vendorID;  
16     }  
17     public String getVendorName() {  
18         return vendorName;  
19     }  
20     public String getRevenue() {  
21         return revenue;  
22     }  
23     // Setters  
24     public void setVendorID(String vendorID) {  
25         this.vendorID = vendorID;  
26     }  
27     public void setVendorName(String vendorName) {  
28         this.vendorName = vendorName;  
29     }  
30     public void setRevenue(String revenue) {
```

The Vendor class sets its data elements as private attributes but provides public methods for secure data control through setters and getters. The abstraction mechanism controls data processing operations by disclosing vendor-specific functions plus it obscures implementation aspects from users. Through separation of concerns both the Vendor class conducts data management but the ViewVendor class maintains user interactions along with GUI behavioral operations. Other system components remain unaffected when expanding vendor-related features

because of this design structure. These OOP concepts lead to a system design that provides both advanced structure and efficient scalability and operational performance.

### 3.2.2 Concept 2 (Encapsulation):

```
1 package Manager;
2
3 public class RatinRunnerStarts {
4
5     public static String getStars(String rating) {
6         try {
7             int stars = Integer.parseInt(rating.trim());
8             if (stars >= 1 && stars <= 5) {
9                 return "★".repeat(stars); // Repeat '★' based on rating
10            }
11        } catch (NumberFormatException e) {
12        }
13        return "";
14    }
15 }
```

The program implements data organization through appropriate classes while maintaining method segregation according to principle of encapsulation. Through the RatinRunnerStarts class the application contains a special method which converts numerical scores into star symbols while maintaining separate data processing functions. The viewDelivaryPref class serves as an encapsulation unit for GUI features along with the related logic which blocks external systems from reaching internal data structures. The program achieves modularity through methods that perform unique tasks independently such as loadData() and searchRunner(). Complete error protection mechanisms ensure data reliability by enabling smooth program functionality.



## 3.3 Vendor (Nye Wei Jun TP066836)

### 3.3.1 Concept 1 (Encapsulation)

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new VendorMenu().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton CustomerFeedbackbtn;
private javax.swing.JButton FinancialStatisticsbtn;
private javax.swing.JButton FoodMenuManagementbtn;
private javax.swing.JButton Logoutbtn;
private javax.swing.JButton OrderHistorybtn;
private javax.swing.JButton OrderManagementbtn;
private javax.swing.JLabel VendorMenu;
private javax.swing.JPanel jPanel1;
// End of variables declaration
}
```

Figure 3.3.1.1 Encapsulation

The VendorMenu class uses encapsulation that keeping UI components private and offering public getter methods by which declared as private, UI buttons (Order Management Button, Customer Feedback Button), it allows to prevent direct access. Alos for Public getter methods like (getOrder Management Btn(), getOrder History Btn()) are offered rather than direct access. These getter techniques allow VendorMenu Controller to access UI components, hence guaranteeing modularity and regulated access.

```

public class OrderManagement extends javax.swing.JFrame {

    private DefaultTableModel tableModel;
    private final String FILE_PATH = "orders.txt"; // File path for storing orders

    public OrderManagement() {

        String[] columns = {"Order ID", "Food ID", "Food Name", "Quantity", "Food_Pr",
                           "Customer Name", "Customer ID", "Vendor", "Remarks",
                           "Runner ID", "Runner", "Delivery_Price", "Status", "Cancel B

        tableModel = new DefaultTableModel(columns, 0);
        initComponents();
        loadOrdersFromFile(); // Load data on startup

        // ✔ Initialize the table model BEFORE loading orders

        OrderManagementTable.setModel(tableModel); // ✔ Assign table model to JTable
        // ✔ Now it's safe to load orders
    }

```

Figure 3.3.1.2 Capsulation 2.0

In the “OrderManagement” class, my code have encapsulated tabModel and FILE\_PATH as variables and they are in private situation, which means cannot be accessed by outside so that the file path would not be modified. “loadOrders()” and “saveOrders()” provide the controlled access to, but the outside code can only operate the orders data by these methods and would not be able to access FILE\_PATH directly.

### 3.3.2 Concept 2 (Inheritance)

```
public class OrderManagement extends javax.swing.JFrame {

    private DefaultTableModel tableModel;
    private final String FILE_PATH = "orders.txt"; // File path for storing orders

    public OrderManagement() {

        String[] columns = {"Order ID", "Food ID", "Food Name", "Quantity", "Food_Pi",
                           "Customer Name", "Customer ID", "Vendor", "Remarks",
                           "Runner ID", "Runner", "Delivery_Price", "Status", "Cancel I

        tableModel = new DefaultTableModel(columns, 0);
        initComponents();
        loadOrdersFromFile(); // Load data on startup

        // ✔ Initialize the table model BEFORE loading orders
    }
}
```

Figure 3.3.2.1 Inheritance

Here using the “extends” syntax to express the “OrderManagement” inherited from JFrame. Javax.swing.JFrame is a Java Swing GUI framework in window class. Hence, “OrderManagement” inherited from JFrame indicated that it automatically obtains all the JFrame methods. For example, “setTitle()” to setting the size of the title, “setSize()” to setting the size of window, setVisible(true) to show the window. “OrderManagement” can also override methods from JFrame to customize behavior if needed.

## 3.4 Runner (Choong Ti Shen TP078540)

### 3.4.1 Encapsulation

```
package Runner;

/**
 *
 * @author Acer
 */
public class Runner {
    private String id;
    private String name;

    public Runner(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Figure 3.4.1.1: Encapsulation 1

The Runner class sets its data elements as private attributes but provides public methods for secure data control through setters and getters. The abstraction mechanism controls data processing operations by disclosing vendor-specific functions plus it obscures implementation aspects from users. Through separation of concerns both the Runner class conducts data management but the Runner\_View\_Order class maintains user interactions along with GUI behavioral operations. Other system components remain unaffected when expanding runner-related features because of this design structure. These OOP concepts lead to a system design that provides both advanced structure and efficient scalability and operational performance.

```

public class Runner_View_Task extends javax.swing.JFrame {
    private Runner runner;
    private String task_OrderID, task_FoodID, task_FoodName, task_UserID, task_Us
public class Runner_View_Dashboard extends javax.swing.JFrame {
    private Runner runner;
    private String Review_RunnerID, Review_RunnerName, Review_Review, Money_Runner

```

Figure 3.4.1.2: Encapsulation 2

Encapsulations also appear in all my classes with "private" attribute.

```

// Variables declaration - do not modify
private javax.swing.JButton AcceptTask;
private javax.swing.JButton Back;
private javax.swing.JTable OrderTable;
private javax.swing.JScrollPane jScrollPane1;
// End of variables declaration

```

Figure 3.4.1.3: Encapsulation 3

The Runner\_View\_Order class uses encapsulation that keeping UI components private and offering public getter methods by which declared as private, UI button (AcceptTask button), it allows to prevent direct access. This getter technique allow Runner\_View\_Order Controller to access UI components, hence guaranteeing modularity and regulated access.

### 3.4.2 Inheritance

```
public class Runner_Update_Task extends javax.swing.JFrame {  
    private Runner runner;  
    private String task_OrderID, task_FoodID, task_FoodName, task_UserID, task_  
  
    /**  
     * Creates new form Runner_Update_Task  
     */  
    public Runner_Update_Task(Runner runner) {  
        this.runner = runner;  
        initComponents();  
        RefreshTask();  
    }  
}
```

Figure 3.4.2.1: Inheritance 1

Here using the “extends” syntax to express the “Runner\_Update\_Task” inherited from JFrame. Javax.swing.JFrame is a Java Swing GUI framework in window class. Hence, “Runner\_Update\_Task” inherited from JFrame indicated that it automatically obtains all the JFrame methods. For example,

```
public Runner_Main_Menu(Runner runner) {  
    this.runner = runner;  
    initComponents();  
    jLabel12.setText(runner.getName());  
}
```

SetText to print the text on JLabel.

“Runner\_Update\_Task” can also override methods from JFrame to customize behavior if needed.

### 3.4.3 Abstraction

```
public Runner_View_Task_History(Runner runner) {  
    this.runner = runner;  
    initComponents();  
    RefreshTaskHistory();  
}
```

Figure 3.4.3.1: Abstraction 1

Abstraction is demonstrated through the RefreshTaskHistory() method, which hides the complexity of reading and filtering data from Order.txt while updating the table. It encapsulates file handling, data extraction, and UI updates, ensuring that other parts of the program only need to call this method without worrying about its internal workings.

```
public Runner_View_Task_History(Runner runner) {  
    this.runner = runner;  
    initComponents();  
    RefreshTaskHistory();  
}  
  
private void RefreshTaskHistory() {  
    try {  
        DefaultTableModel model = (DefaultTableModel) TaskTable.getModel();  
        model.setRowCount(0); // Clear table before adding new data  
        String filename = "Order.txt";  
        FileReader fr = new FileReader(filename);  
        BufferedReader br = new BufferedReader(fr);  
        String read;  
        while ((read = br.readLine()) != null) {  
            String data[] = read.split(";"); // Assuming data is separated by ';'   
            if (data.length >= 14) {  
                // ...  
            }  
        }  
    }  
}
```

Code above shows function of RefreshTaskHistory().

```
if (task_Status.equalsIgnoreCase("Delivered") && task_RunnerName.equalsIgnoreCase(runner.getName())) {  
    model.addRow(new Object[]{task_OrderID, task_FoodID, task_FoodName, task_UserID, task_Username, task_`  
}
```

Figure 3.4.3.2: Abstraction 2

Besides, this also shows abstraction method. The Runner class abstracts runner details, allowing runner.getName() to filter relevant records.

### 3.4.4 Composition

```
FileReader fr = new FileReader(filename);
BufferedReader br = new BufferedReader(fr);
ArrayList<String> lines = new ArrayList<>();
String line;
while ((line = br.readLine()) != null) {
    lines.add(line);
}
br.close();
```

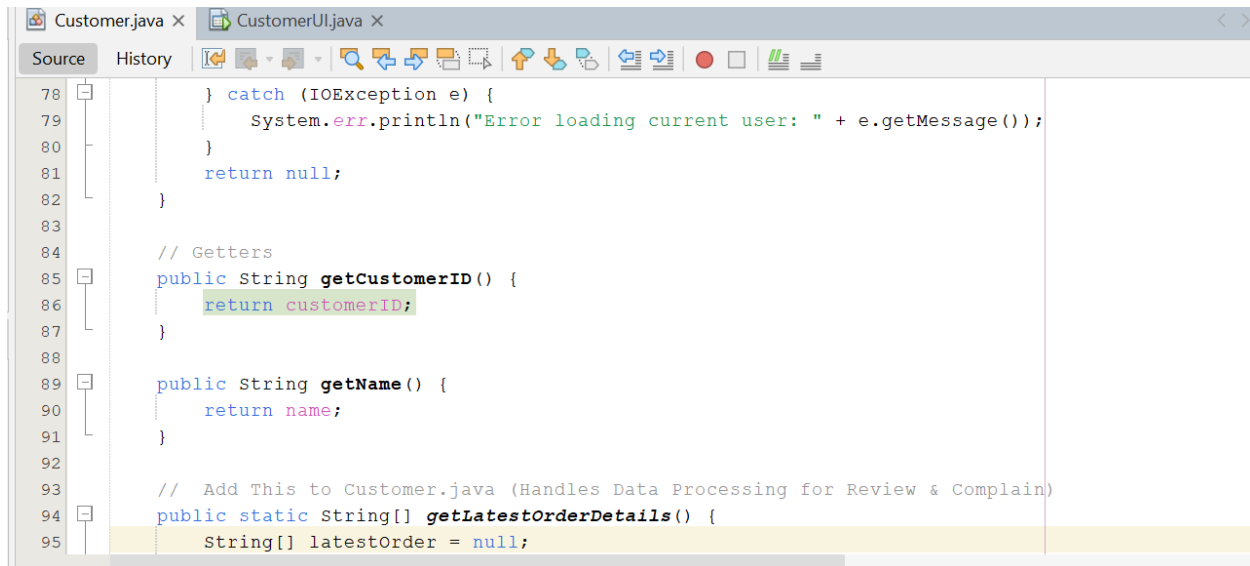
Figure 3.4.4.1: Composition 1

In Object-Oriented Programming (OOP), composition is a design principle where a class is composed of one or more objects of other classes. It allows you to build complex objects by combining simpler ones, promoting code reuse and modularity. Composition is shown through the use of `FileReader`, `BufferedReader`, and `ArrayList`, where each object works together to achieve efficient file reading and storage. `BufferedReader` wraps `FileReader` to improve reading performance, demonstrating a has-a relationship. Similarly, `ArrayList<String>` is used to store file content dynamically for later use. This composition enhances encapsulation, reusability, and maintainability by keeping file handling and data storage separate, making the code more modular and efficient.



## 3.5 Customer (TP072541)

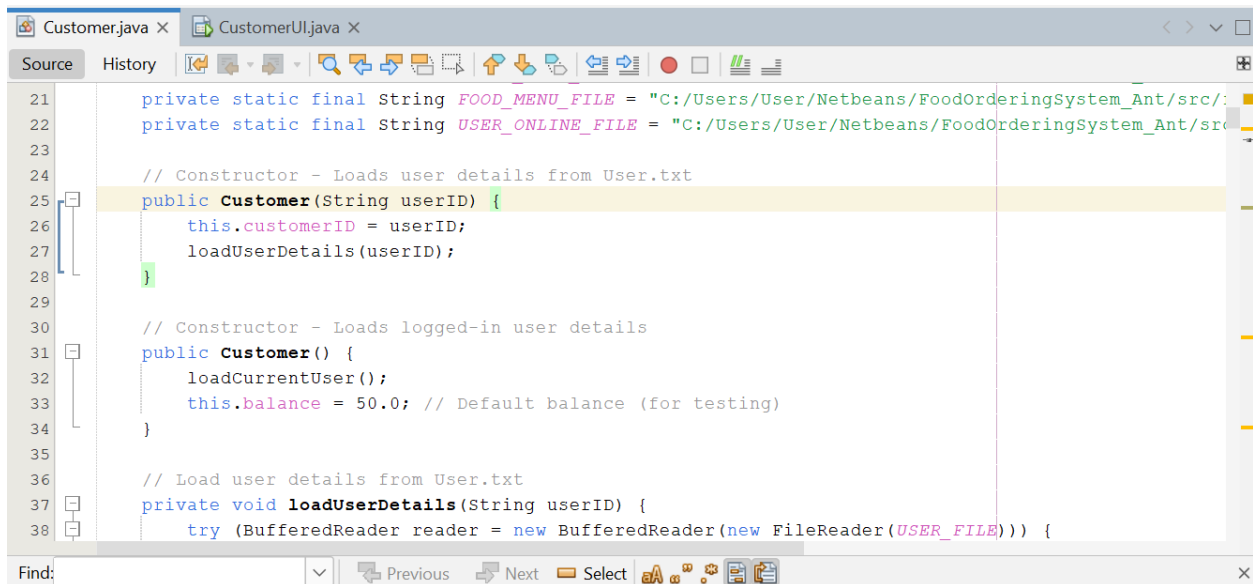
### 3.5.1 Encapsulation



```
Customer.java x CustomerUI.java x
Source History
78 } catch (IOException e) {
79     System.err.println("Error loading current user: " + e.getMessage());
80 }
81 return null;
82 }
83
84 // Getters
85 public String getCustomerID() {
86     return customerID;
87 }
88
89 public String getName() {
90     return name;
91 }
92
93 // Add This to Customer.java (Handles Data Processing for Review & Complain)
94 public static String[] getLatestOrderDetails() {
95     String[] latestOrder = null;
```

The security and integrity of data in object-oriented programming (OOP) depend on encapsulation because it largely restricts the access to object attributes. The Customer class is encapsulated by the private declarations of the customerID and name variables supported by the getCustomerID () and getName() getter methods. This method restricts the external modification of class attributes, thus making the authorized method the only entity that can access and change user information. The program realizes data integrity, error reduction and maintainability improvement through encapsulation, because it hides the internal data representation and makes it inaccessible to external systems.

### 3.5.2 Polymorphism



```
21 private static final String FOOD_MENU_FILE = "C:/Users/User/Netbeans/FoodOrderingSystem_Ant/src/";
22 private static final String USER_ONLINE_FILE = "C:/Users/User/Netbeans/FoodOrderingSystem_Ant/src/";
23
24 // Constructor - Loads user details from User.txt
25 public Customer(String userID) {
26     this.customerID = userID;
27     loadUserDetails(userID);
28 }
29
30 // Constructor - Loads logged-in user details
31 public Customer() {
32     loadCurrentUser();
33     this.balance = 50.0; // Default balance (for testing)
34 }
35
36 // Load user details from User.txt
37 private void loadUserDetails(String userID) {
38     try (BufferedReader reader = new BufferedReader(new FileReader(USER_FILE))) {
```

In object-oriented programming (OOP), polymorphism represents a key feature, which enables methods or constructors to use the same name but make different responses according to the received input parameters. The constructor overload appears in the Customer class because it implements compile-time polymorphism. As part of the constructor overload, this class provides a constructor that accepts userID and a default constructor that automatically gets the active login user. This design provides adaptive object creation and supports file-based detailed information retrieval and active user loading from specific customers. Using polymorphism in this code base improves the scalability, maintainability and reusability of the system, which makes future changes have the least impact on existing functions.

## 4.0 Additional feature

### 4.1 Administrator (Real-Time Receipt Generation):

```
233 private void generateReceipt(String username, String userId, double balance) {
234     String receiptFile = "receipts.txt";
235     String timestamp = new SimpleDateFormat("yyyyMMddHHmmss").format(new Date());
236     String dateTime = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());
237
238     String receiptContent =
239         "Receipt ID: " + timestamp + "\n" +
240         "Date: " + dateTime + "\n" +
241         "Username: " + username + "\n" +
242         "User ID: " + userId + "\n" +
243         "Wallet Balance: $" + balance + "\n" +
244         "\n";
245
246     try (BufferedWriter writer = new BufferedWriter(new FileWriter(receiptFile, true))) {
247         writer.write(receiptContent);
248     } catch (IOException e) {
249         System.out.println("Error writing receipt file: " + e.getMessage());
250     }
251 }
```

Figure 4.1.1 GenerateReceipt.java JFrame

Real-time receipt generation is implemented by the system to guarantee that every transaction is accurately documented with a timestamp and unique identification. The precise date and time of the transaction are recorded by the system and stored in the format yyyy-MM-dd HH:mm:ss when an administrator creates a receipt. Based on the current system clock, this guarantees that each receipt has a distinct Receipt ID. Through dynamic receipt generation while processing, the system removes problems like missing or duplicating records. Transaction tracking becomes more effective when the receipt details, such as the User ID, Username, Wallet Balance, and Timestamp, are instantly saved in receipts.txt. Furthermore, administrators may quickly access and transmit receipts because the receipt ID is shown on the receipt table right away. This real-time method improves the system's financial records' correctness and dependability.

## 4.2 Manager (Enhanced Functionality):

```
6      try {
7          int stars = Integer.parseInt(rating.trim());
8          if (stars >= 1 && stars <= 5) {
9              return "★".repeat(stars); // Repeat '★' based on rating
10         }
11     }
```

Figure 4.2.1 getting the start for delivery rate

The code performs additional feature implementations to boost functionality as well as user experience. Staff notices enable try-catch function to prevent system crashes because of invalid inputs thus guaranteeing system stability. The getStars() function processes numerical ratings into star visual elements which enhances the appearance. The application includes a search feature which enables users to locate Runner ID entries thus improving their query efficiency. The DefaultTableModel automatically updates in real-time when users search for or load data in the system. The system features an easy-to-use interface through its Graphical User Interface navigation design which lets users perform operations without complications.

```
187 private void TotalRevenueBtnActionPerformed(java.awt.event.ActionEvent evt) {
188     // TODO add your handling code here:
189     String searchVendorID = jTextField1.getText().trim();
190     DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
191
192     double totalRevenue = 0;
193     boolean found = false;
194
195     for (int i = 0; i < model.getRowCount(); i++) {
196         String vendorID = model.getValueAt(i, 0).toString();
197         if (vendorID.equals(searchVendorID)) {
198             totalRevenue += Double.parseDouble(model.getValueAt(i, 2).toString());
199             found = true;
200         }
201     }
202
203     if (found) {
204         JOptionPane.showMessageDialog(this, "The total revenue of Vendor ID " + searchVendorID + " is: " + totalRevenue);
205     } else {
206         JOptionPane.showMessageDialog(this, "Vendor ID not found.");
207     }
208 }
```

Figure 4.2.2 GenerateReceipt.java JFrame

Users can calculate total revenue for chosen vendors by entering their vendor ID using the additional feature in this code. The feature allows system managers to evaluate vendor earnings through the Vendor Performance interface. The TotalRevenueBtnActionPerformed method operates on this feature from lines 209 through 230. The method uses the user-input Vendor ID from the text field while going through each row of the JTable to calculate the revenue totals associated with that ID. The program shows the total revenue in a dialog box upon discovering the ID using JOptionPane.showMessageDialog(). The program displays an error notification when the entered ID does not match any records. The system feature provides managers with efficient vendor performance analysis by automatically filtering data thus enhancing both the interface and user experience.

### 4.3 Vendor (Enhancing the Order Management System: Adding Cursor Selection for Order Acceptance)

```
32  /** Update selected order status and save changes back to file */
33  private void updateOrderStatus(String newStatus) {
34      int selectedRow = OrderManagementTable.getSelectedRow();
35      if (selectedRow == -1) {
36          JOptionPane.showMessageDialog(this, "Please select an order!", "Warning", JOptionPane.WARNING_MESSAGE);
37          return;
38      }
39
40      // Update JTable data
41      tableModel.setValueAt(newStatus, selectedRow, 12); // "Status" column is index 12
42
43      // Save the updated table back to the file
44      saveOrdersToFile();
45
46      JOptionPane.showMessageDialog(this, "Order updated to: " + newStatus);
47  }
48
49  /** Save orders from JTable back to text file */
50  private void saveOrdersToFile() {
51      try (BufferedWriter bw = new BufferedWriter(new FileWriter(FILE_PATH))) {
52          for (int i = 0; i < tableModel.getRowCount(); i++) {
53              StringBuilder row = new StringBuilder();
54              for (int j = 0; j < tableModel.getColumnCount(); j++) {
55                  row.append(tableModel.getValueAt(i, j));
56                  if (j < tableModel.getColumnCount() - 1) row.append(";"); // Use semicolon separator
57              }
58              bw.write(row.toString());
59              bw.newLine();
60          }
61      }
62  }
```

Figure 4.3 Enhancing the Order Management System

To allow vendors to quickly alter order statuses by choosing an order from a JTable and using one button click status modifications. Using the cursor, vendors may choose orders; thereafter, the order statuses can be changed immediately in the JTable. This function guarantees quick, accurate, and user-friendly order management by letting suppliers choose an order and edit it with a click, hence simplifying order processing. Since suppliers don't have to manually input order data and

updates quickly saved to "orders.txt could improve user experience," so this is rather crucial. It might also stop inadvertent updates that only the chosen order rows item changes.

## 4.4 Runner

### 4.4.1 Total Earned Calculating in Dashboard

```
public void Runner_Dashboard_Earned() {
    try {
        String filename = "Transaction_History.txt";
        FileReader fr = new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        int Runner_Money_Count = 0;
        String read;
        while ((read = br.readLine()) != null) {
            String data[] = read.split(";"); // Assuming data is separated by ';'
            if (data.length >= 4) {
                Money_RunnerID = data[0]; // RunnerID
                Money_RunnerName = data[1]; // RunnerName
                Money_RunnerAmount = data[2]; //amount

                if (Money_RunnerName.equalsIgnoreCase(runner.getName())) {
                    //int Money_RunnerAmountINT = Integer.parseInt(Money_RunnerAmount);
                    double Money_RunnerAmountINT = Double.parseDouble(Money_RunnerAmount);
                    Runner_Money_Count += Money_RunnerAmountINT;
                    Dashboard_Earned.setText(String.valueOf(Runner_Money_Count));
                }
            }
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    }
}
```

Figure 4.4.1.1: Total Earned Calculator Function

To allow runner track down how much money they earned, we implements a function for them to calculate the amount of total money earned. The system read from the file that saves transaction history, Transaction\_History.txt. If the runner name match at the column that saves runner name, the double amount of that row will saved to the total money calculator, Runner\_Money\_Count. After lopping through the whole file, that value of total money calculator is sent to jLabel with named Dashboard\_Earned.

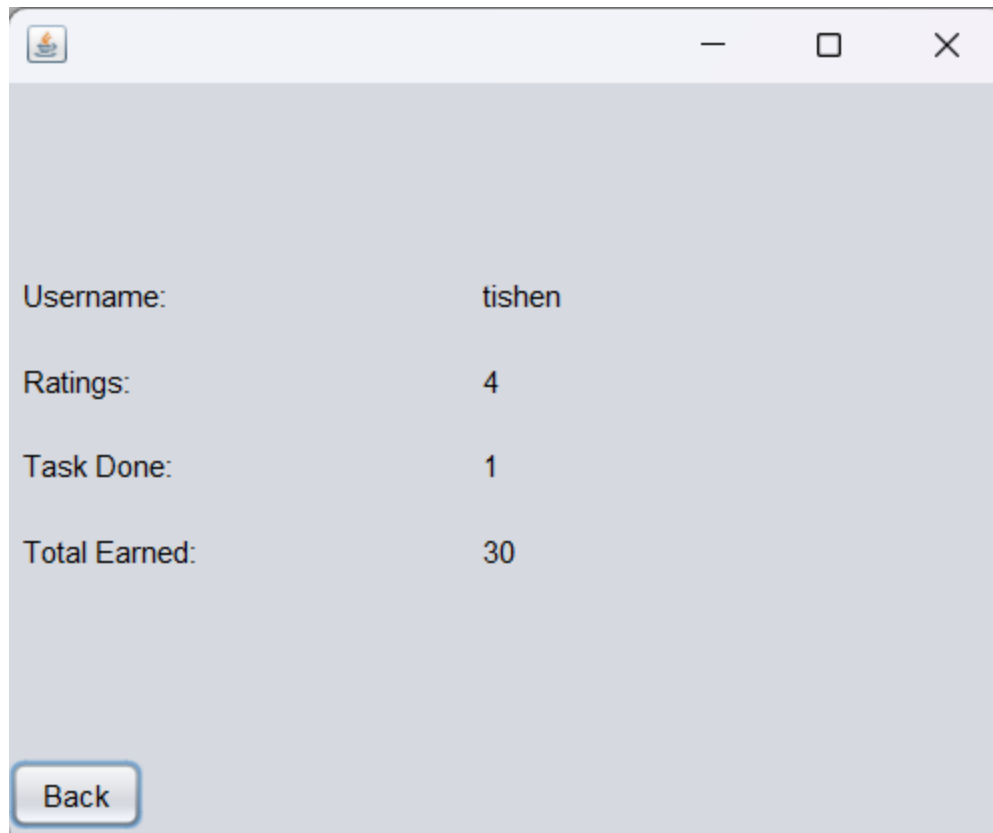


Figure 4.4.1.2: Sample output of the code

This is the output of the code. That total amount calculated is 30 at is showed at section total earned.

## 4.4.2 Enhanced Rating Message at View Customer Review

```
        RatingsCalc += RatingsLevel;
        RatingLevel = RatingsCalc/Loop;
        RatingsCount.setText(String.valueOf(RatingLevel));
        Loop ++;
    }

}

if (RatingLevel == 0){
    Runner_Performance.setText("Done be lazy, let's start by picking a task.");
}
else if (RatingLevel <= 2){
    Runner_Performance.setText("Come on, lets improve by being punctual, communicative, and professional.");
}
else if (RatingLevel <= 4){
    Runner_Performance.setText("Buddy, you are doing it good, keep it up!");
}
else if (RatingLevel == 5){
    Runner_Performance.setText("Fantastic work! Your 5-star rating shows your dedication and reliability.");
}
else{
    Runner_Performance.setText("System Error...");
}
```

Figure 4.4.2.1: Enhanced Rating Message

For the ratinglevel calculated at Average Rating Count at View Customer Review, I added some message to print out for specific level count. For example, the message “Buddy, you are doing it good, keep it up!” is sent to Runner\_Performance jLabel. For 0, it means that runner didnt completed any task yet, hence we show message “Done be lazy, let’s start by picking a task.”. By doing this, it guides that runner on how to improve their rating levels.

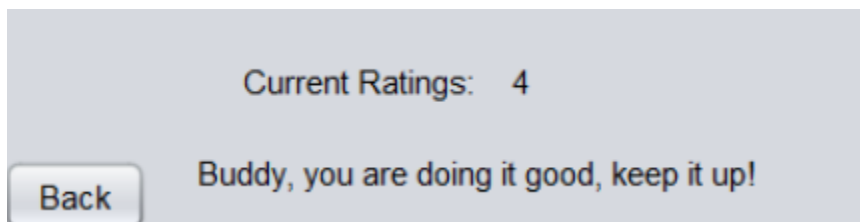


Figure 4.4.2.2: Sample Output of the code

This is the sample output of the code. At this case, runner have 4 ratinglevel. Therefore, the message “Buddy, you are doing it good, keep it up!” for its case is printed out at jLabel Runner\_Performance.



## 5.0 Limitations

The system delivers all functional specifications, but the implementation includes certain restrictions which could enhance performance. The current text file data storage causes scalability problems that lead to slower data retrieval with increasing user numbers. A database system implementation would deliver improved data accessibility together with enhanced efficiency. The real-time receipt generation process lacks automated capabilities for delivering receipts by email or SMS to customers since administrator intervention is necessary for notification. Managers and administrators must work without access to an integrated reporting feature to perform analytics on transactions, user activity and financial performance. The system operates as a desktop application that gives users access to it to run on a single platform only. The addition of web-based or mobile accessibility features would enhance system usage as it would establish remote management capabilities for operations.

## 6.0 Conclusion

The Courier Service Management System completes all specified requirements through its user management and transaction processing and delivery handling platform. The system adopts main principles from Object-Oriented Programming (OOP) including encapsulation together with inheritance together with polymorphism to deliver a modular and maintainable structure. Real-time receipt generation along with wallet management systems and user validation functionalities in the system create a faster and safer financial transaction environment.

Multiple team members participated in the development process of separate system segments. The Administrator module represents the platform for user administration together with credit addition features receipt printing along with transaction keeping. Through the Customer module users can access menus and order and track there. Projected items while undertaking transaction control. Through the Vendor module users gain access to product management systems together with order status updates and sales tracking features. With the Delivery Runner module task management functions allow both runners to take delivery assignments and conclude their tasks. The Manager module functions as an oversight of the complete system through vendor monitoring and delivery runner performance assessment and customer dispute resolution.

## 7.0 References

Muhammad, S. (2024, November 22). Master Java OOP Concepts: Key Concepts explained | Suraif Muhammad | Medium. *Medium*. <https://medium.com/@suraif16/object-oriented-programming-9e4627abf1be>

Simplilearn. (2023, February 23). *What are Java classes and objects and how do you implement them?* Simplilearn.com. <https://www.simplilearn.com/tutorials/java-tutorial/java-classes-and-objects>

Johari, A. (2024, August 23). *Netbeans Tutorial: What is NetBeans IDE and how to get started?* Edureka. <https://www.edureka.co/blog/netbeans-tutorial/>

Bangari, R. (2024, June 27). *Inheritance in java*. Geekster Article. <https://www.geekster.in/articles/java-inheritance/>

Behler, M. (2020, December 9). *How to work with files in Java*. Marco Behler GmbH. <https://www.marcobehler.com/guides/java-files>

## 8.0 Appendix

### 8.1 Administrator (Eshchanov Umidjon TP071568)

```
1  umid,1,umid@gmail.com,umid,12345678,Vendor
2  mak,2,mak@gmail.com,mak,87654321,Customer
3  tishen,3,tishen@gmail.com,tishen,00000000,Runner
4  weili,4,weili@gmail.com,weili,1234567890,Customer
5
```

Figure 8.1.1 users.txt

```
1  mak,2,10.0
2  weili,4,70.0
3  jun,5,0.0
4
```

Figure 8.1.2 wallets.txt

```
1  Receipt ID: 20250222123026
2  Date: 2025-02-22 12:30:26
3  Username: weili
4  User ID: 4
5  Wallet Balance: $0.0
6
7  Receipt ID: 20250222123052
8  Date: 2025-02-22 12:30:52
9  Username: mak
10 User ID: 2
11 Wallet Balance: $10.0
12
13 Receipt ID: 20250222123055
14 Date: 2025-02-22 12:30:55
15 Username: weili
16 User ID: 4
17 Wallet Balance: $30.0
18
19 Receipt ID: 20250222153706
20 Date: 2025-02-22 15:37:06
21 Username: mak
22 User ID: 2
23 Wallet Balance: $10.0
24
25
```

Figure 8.1.3 receipts.txt

## 8.2 Manager (Mohammed Ashraf TP078045)

```

1 Vendor ID, Vendor, Food ID, Food Name, Price, Description
2 V2, KFC, F5, Popcorn Chicken, 4.99, Bite-sized crispy chicken pieces
3 V2, KFC, F6, Mashed Potatoes, 2.79, Mashed potatoes with signature gravy
4 V3, Subway, F7, Turkey Sub, 6.99, Turkey breast sandwich with fresh veggies
5 V3, Subway, F8, Italian BMT, 7.49, Pepperoni, salami, and ham with cheese

```

Figure 8.2.1 FoodMenu.txt

```

1 OrderID, Foodid, food name, User ID, Username, Runner Id, Runner, Review
2 1, F1, Coke, U1, Jun, R3, Tishen, 4
3 2, F2, Burger, U2, Sarah, R8, Ronald, 5
4 3, F3, Pizza, U3, David, R6, Sam, 3
5 4, F4, Fries, U4, Emily, R9, John, 2
6 5, F5, Sandwich, U5, Michael, R12, Jacob, 5
7 6, F5, Sandwich, U5, Michael, R12, Jacob, 2
8

```

Figure 8.2.2 Review\_Runner.txt

```

Source History
1 101, Vendor A, 500, 15000, 45000
2 102, Vendor B, 600, 18000, 54000
3 103, Vendor C, 700, 21000, 63000
4 104, Vendor F, 700, 21000, 67000
5

```

Figure 8.2.3 Review\_Vendor.txt

1	1	2	3	4	5	6	7	8	9	10	11	12
2	1001,	F001,	Chicken Burger,	U101,	Ahmed,	V201,	Burger House,	R301,	Ali,	25.00,	5.00,	Deliv
3	1002,	F002,	Margherita Pizza,	U102,	Sara,	V202,	Pizza Corner,	R302,	Hassan,	40.00,	6.00,	
4	1003,	F003,	Caesar Salad,	U103,	Fatima,	V203,	Green Bites,	R303,	Omar,	18.00,	4.00,	Cance
5	1004,	F004,	Sushi Platter,	U104,	John,	V204,	Tokyo Sushi,	R304,	Bilal,	55.00,	7.00,	Deliv
6	1005,	F005,	Beef Shawarma,	U105,	Zain,	V205,	Shawarma King,	R305,	Karim,	22.00,	5.00,	Can
7	1006,	F006,	Veggie Burger,	U106,	Omar,	V201,	Burger House,	R306,	Faisal,	20.00,	5.00,	Del
8	1007,	F007,	Pepperoni Pizza,	U107,	Aisha,	V202,	Pizza Corner,	R307,	Khalid,	42.00,	6.00,	
9	1008,	F008,	Greek Salad,	U108,	Noor,	V203,	Green Bites,	R308,	Sami,	19.00,	4.00,	Delivere
0	1009,	F009,	California Roll,	U109,	Bilal,	V204,	Tokyo Sushi,	R309,	Hassan,	50.00,	7.00,	D
1	1010,	F010,	Chicken Shawarma,	U110,	Yasmin,	V205,	Shawarma King,	R310,	Omar,	24.00,	5.00,	

Figure 8.2.4 Order.txt

### 8.3 Vendor (Nye Wei Jun TP066836)

Vendor ID	Vendor	Food ID	Food Name	Price	Description
a4	weijun	b4	Duck Rice	30	less spicy
a4	weijun	b5	Chicken rice	30	no rice
a4	weijun	b6	Nasi Kuning	88	Very very spicy
a4	weijun	b7	spaghetti with chicken chop	68	black pepper chicken sauce

Figure 8.3.1 FoodMenu.txt

Order ID	Food ID	Food Name	Quantity	Food Price	Customer Name	Customer ID	Vendor	Remarks	Runner ID	Delivery Price	Status	Cancel by	Date
z1	b4	Duck Rice	1	30	sieowjun	Y1	tishen	None	RU9		5 pending	none	1/4/2023
z2	b5	Chicken rice	1	30	sieowjun	Y1	tishen	None	RU10		5 completed	none	9/5/2022
z3	b6	Nasi Kuning	2	88	sieowjun	Y1	tishen	None	RU11		5 completed	none	10/5/2022
z4	b7	spaghetti with chicken chop	1	68	sieowjun	Y1	tishen	None	RU12		5 completed	none	11/5/2022

Figure 8.3.2 Order.txt

```
F001;John Doe;C001;Great service and food!;2025-02-20
F002;Jane Smith;C002;The pizza was a bit cold.;2025-02-18
F003;Alice Johnson;C003;Loved the burger, but the fries were soggy.;2025-02-19
F004;Bob Lee;C004;Excellent delivery speed!;2025-02-21
```

Figure 8.3.3 Feedback.txt

## 8.4 Runner (Choong Ti Shen TP078540)

```
O001;F001;Burger;C001;John Doe;V1;Vendor A;R1;tishen;15.00;5.00;Delivering;null;2022-02-05
O002;F002;Pizza1;C002;Jane Doe;V1;Vendor B;None;None;25.00;5.00;Done;null;2023-02-05
O003;F001;Burger;C001;John Doe;V1;Vendor A;R1;tishen;15.00;5.00;Delivered;null;2022-02-05
O004;F002;Pizza1;C002;Jane Doe;V1;Vendor B;None;None;25.00;5.00;Delivered;null;2023-02-05
O005;F001;Burger;C001;John Doe;V1;Vendor A;None;None;15.00;5.00;Done;null;2022-02-05
O006;F002;Pizza1;C002;Jane Doe;V1;Vendor B;R1;tishen;25.00;5.00;Delivered;null;2023-02-05
O007;F001;Burger;C001;John Doe;V1;Vendor A;None;None;15.00;5.00;Done;null;2022-02-05
O008;F002;Pizza1;C002;Jane Doe;V1;Vendor B;None;None;25.00;5.00;Delivered;null;2023-02-05
O009;F001;Burger;C001;John Doe;V1;Vendor A;R1;tishen;15.00;5.00;Delivering;null;2022-02-05
O0010;F002;Pizza1;C002;Jane Doe;V1;Vendor B;None;None;25.00;5.00;Done;null;2023-02-05
O0011;F001;Burger;C001;John Doe;V1;Vendor A;None;None;15.00;5.00;Done;null;2022-02-05
O0012;F002;Pizza1;C002;Jane Doe;V1;Vendor B;None;None;25.00;5.00;Delivered;null;2023-02-05
```

Figure 8.4.1 Order.txt

```
OrderID;Foodid;food name;User ID;Username;Runner Id;Runner;Review
1;F1;Coke;U1;Jun;R1;tishen;4
2;F2;Burger;U2;Sarah;R1;tishen;5
3;F3;Pizza;U3;David;R6;Sam;3
4;F4;Fries;U4;Emily;R9;John;2
5;F5;Sandwich;U5;Michael;R12;Jacob;5
6;F5;Sandwich;U5;Michael;R12;Jacob;2
```

Figure 8.4.2 Review\_Runner.txt

```
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
R1;tishen;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
C002;Jane Doe;-25.00;2023-02-05
V1;Vendor B;25.00;2023-02-05
C002;Jane Doe;-5.00;2023-02-05
R1;tishen;5.00;2023-02-05
C001;John Doe;-15.00;2022-02-05
V1;Vendor A;15.00;2022-02-05
C001;John Doe;-5.00;2022-02-05
R1;tishen;5.00;2022-02-05
```

Figure 8.4.3: Transaction\_History.txt



## 8.5 Customer (TP072541)

01740226359484,F1,Chicken,C001,Jun,V001,WJUN,R1,Ali,15.0,5.0,Complete  
01740226359484,F2,Burger,C001,Jun,V001,WJUN,R2,Abu,10.0,5.0,Complete  
01740232331563,F1,Chicken,C001,Jun,V001,WJUN,R3,Amah,15.0,5.0,Complete  
01740233278895,F1,Chicken,C001,Jun,V001,WJUN,R4,Adik,15.0,5.0,Complete  
01740305728000,F1,Chicken,C001,Jun,V001,WJUN,R5,Aduk,15.0,5.0,Complete

F1,Chicken,C001,Jun,V001,WJUN,15.0,5.0

## 9.0 Presentation video links:

9.1 Administrator: [JavaPresentation.mp4](#)

9.3 Vendor: [Java Presentation \(Vendor\).mp4](#)

9.4 Customer: [Java customer by Phang Sieow Jun TP072541.mp4](#)

9.5 Runner

[https://drive.google.com/drive/folders/1oYMQC5UjOo1wi\\_Z0vH5J\\_zd2X3t7OP9s](https://drive.google.com/drive/folders/1oYMQC5UjOo1wi_Z0vH5J_zd2X3t7OP9s)