CT038-3-OODJ

OBJECT ORIENTED DEVELOPMENT WITH JAVA

GROUP ASSIGNMENT

APD2F2206CE

Group Members:

Name	TP Number
Mohammad Fawzan Alim	TP064501
Nicholas Tan Peng Gen	TP061291

Lecturer Name: Tanveer Khaleel Shaikh

Submission Date: 2nd September 2022

Table of Contents

List of Figures

List of Tables

1.0 Introduction

We've created a system to handle registration, login, ordering, allow the customers to pay for the food that they've ordered and collect feedback from customer. APU Cafeteria Ordering System intends to create an interactive and intuitive ordering system that allows managers to handle the general operations, view feedback and manage the cafeteria through this application. This system allows APU Cafeteria to be more sophisticated in this digital world. This is because this system allows the user to ceaselessly pay for their meal, which reduces the time it takes to pay for the food. This is because cashless transaction is faster than transactions that are made with cash as the seller will have to return the balance to the customer if the customer pays the food using a large cash note. In addition to that, cashless transactions are also more hygienic as there wouldn't be the need to have physical contact between the food seller and the customer. The overall hygiene improvement of APU cafeteria area will certainly reduce the chances of diseases spreading.

The use of cashless transaction also gives the customers more flexibility as they do not have to be physically present to place an order. They can order online and collect the food later. They can take the food away or eat it at the cafeteria. A crucial feature of the cafeteria system is the feedback submission feature that all customers can use. This allows the customer to rate the food and their overall experience at the cafeteria. They can also write a comment to tell the manager on specific things that the cafeteria can improve on. The manager can take the feedback given by the customers and make the needed changes to the cafeteria. In addition to that, we've used object-oriented programming to enhance our application and make it more interactive and create advance features. We also used images in every window of the system to make it more visually pleasing to look at.

1.1 Assumption

For this assignment, we have assumed that the cafeteria system we will develop will be used exclusively on the APU campus. There are only two types of system users: managers and customers. In the cafeteria, there will be only one manager who will handle all issues. All cafeteria customers will be APU students and employees. Consequently, they will all have APU email addresses. We assumed all APU email addresses would end in @apu.edu.my, @mail.apu.edu.my or @staffmail.apu.edu.my.

There is no option to pay with cash in the cafeteria. The AP card is also not accepted. The only accepted forms of payment are debit and credit cards. We assume that customers will provide accurate card information and have sufficient funds on their cards to cover the cost of the food. We have also assumed that they will receive their food immediately after paying for it.

2.0 UML Diagrams

2.1 Use Case Diagram

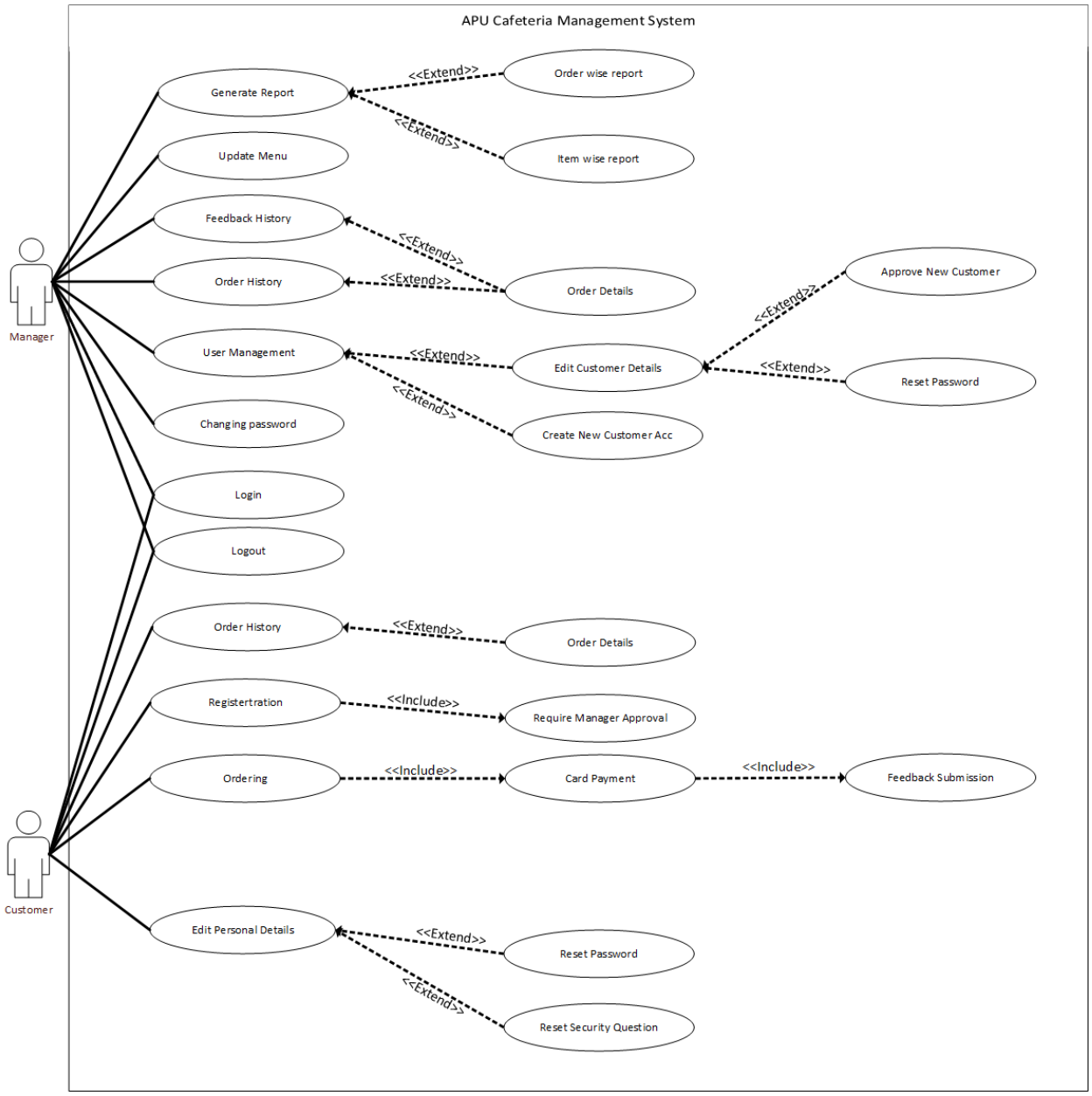


Figure 1: Use Case Diagram for APU Cafeteria Ordering System

2.2 Use Case Specification

Table 1: Use Case A1 – Login

Use Case ID	A1
Use Case Name	Login
Case Description	This use case allows manager and customers to login to their respective main menu using a unique username and a password.
Actors	Manager, Customer
Preconditions	1. Manager and Customers must enter their valid username followed by their password.
Main flow	1. The use case begins when the customer or manager starts the program 2. The system prompts the actor for their username and password 3. The user fills up the username and password fields with their respective username and password. 4. The system checks whether the username is valid or not. If the username is valid, the system proceeds to check whether the password belongs to the username or not.
Alternative flow	1. The actor enters the wrong username or password, and the system will display an error message. 2. The actor leaves the username and password fields empty, and the system will display an error message. 3. The actor doesn't remember the password, and the actor can reset their password by clicking on the "Reset Pass" option.

Table 2: Use Case A2 - Registration

Use Case ID	A2
Use Case Name	Registration
Case Description	This use case allows the customer to create their own unique username and password so that they can use it to log into the Cafeteria Ordering System.
Actors	Customer
Preconditions	1. Actors do not have an account
Main flow	1. The system prompts the actor for required information such as their username, first name, last name, password, email, telephone number, a security question so that they can recover their account and the answer to the security question. 2. The actor fills up all the fields and clicks on the register button 3. All the inputs are validated 4. The system will prompt a message that tells the actor that their registration is pending and that they need to wait for the approval of the manager. 5. The actor can logout and go back to the main login page while waiting for the manager to approve the registration
Alternative flow	1. The actor enters a username that has already been taken, and the system will display an error message. 2. The actor leaves any of the fields empty, and the system will display an error message. 3. The actor enters a username that is shorter than 5 letters, longer than 15 letters or doesn't start with alphabetical letters, and the system will display an error message.

	4. The actor provides a phone number that consists of anything other than numbers, then then the system will show an error message. 5. The actor enters non-APU email, then the system will show an error message. 6. The actor enters different input for the password and confirm password field, and the system will display an error message. 7. Actor already has an account with the given email but tries to register, the system will display an error message and the actor can logout to the login page and login with their account. Or if they have forgotten their password, they can reset it using Forgot Password page.

Table 3: Use Case A3 – Ordering

Use Case ID	A3
Use Case Name	Ordering
Case Description	This use case allows customers to order various foods of different categories from the menu
Actors	Customer
Preconditions	2. Customers must have an account and be logged into the Cafeteria Ordering System.
Main flow	1. The use case begins after the customer logs in. 2. The system displays a total of 23 different dishes of various categories along with their images and price to the customer. 3. The system prompts the customer to add dishes that they want to into the cart. The customer can also remove dishes from the cart if they changed their mind or made a mistake.

	4. The name, quantity, price, and total price of the selected dishes are displayed to the customer on the same page by the system. 5. The customer can procced to pay for their dishes.
Alternative flow	1. The actor doesn't feel like eating a specific dish that they added to the cart. They can remove it and the cart will reflect the change. 2. The actor changes their mind and wants to remove all of the food from the cart. The customer can remove all of the dishes from the cart with a single button press and the cart will be empty. 3. The actor tries to proceed and pay with an empty cart. The system will display an error message to the actor. 4. The actor does not want to order and logs out of the system 5. The actor checks their order history or goes to edit their personal details

Table 4: Use Case A4 – Payment

Use Case ID	A4
Use Case Name	Payment
Case Description	This use case allows customers to pay for their food using a valid credit or debit card.
Actors	Customer
Preconditions	1. The customer must have an account and be logged in 2. The customer must have at least one item in the cart.
Main flow	1. This use case begins when a customer has dishes in their cart and decides to pay

	2. The system displays the item, quantity of items, price of each item and the total price to be paid for the order. 3. The system prompts the user to pay for the food using their credit or debit card. 4. The actor enters their valid card information and pays for the food. 5. The system validates the payment information and proceed to the feedback page.
Alternative flow	1. The actor enters an invalid card number. The system will display an error message. 2. The actor leaves any of the required field black. The system will display an error message. 3. The actor enters the information of an already expired card. The system will display an error message. 4. The actor changes their mind and wants to add or remove dishes to the cart. They can go back to the menu page and update the cart.

Table 5: Use Case A5 – Feedback Submission

Use Case ID	A5
Use Case Name	Feedback Submission
Case Description	This use case allows customers to give feedback. The customer can rate the food and the overall satisfaction that they had in the cafeteria. They can also write a comment which will more accurately describe their opinion about the Cafeteria Ordering System.
Actors	Customer
Preconditions	1. Customer must have an account

	2. Customer must be logged in 3. Customer must have already ordered their dish and paid for it
Main flow	1. The use case begins when the user finishes paying for their meal. 2. The system displays 2 sliders and a text area to determine the level of satisfaction of the customer towards the food and the overall environment of the Cafeteria Ordering System. 3. Customer rate their level of satisfaction on a scale of 1 to 5, 1 being terrible and 5 being extremely satisfied with the Cafeteria Ordering System. 4. Customer rate how much they like their food on a scale of 1 to 5. 5. Customer then can write feedback which would be more detailed and personal when compared with rating. 6. The customer can then enjoy their food after submitting feedback.
Alternative flow	1. The actor can doesn't want to provide feedback can press skip to avoid giving feedback

Table 6: Use Case A6 – Csutomer Editing Personal Details

Use Case ID	A6
Use Case Name	Customer Editing Personal Details
Case Description	Customers can edit the details of their account. For example, if the customer wants to change their username, they can change it. They can also edit their telephone number if they aren't using them anymore.
Actors	Customer
Preconditions	1. Customer must have an account

	2. Customer must be logged in
Main flow	1. The use case begins by the system showing the customer the details of the current account. Such as username, first name, last name, email and the telephone number. 2. The customer can only edit username and telephone numbers by editing the related text fields that the system displayed to them. 3. The customer presses the save button and the system saves the information after validating which updates the account to have the latest details.
Alternative flow	1. The customer changes the username to an already existing username. The system will display an error message. 2. The customer leaves any of the text fields blank. The system will display an error message. 3. The customer provides a username with a length of less than 5 or more than 15, the system will display an error message. 4. The username does not start with a letter or has any invalid special character, the system will display an error message. 5. The customer wants to reset their password. The system redirects them to another page to make it more secure. The customer can then change their password in the more secured page. 6. The customer wants to reset their security question. The system redirects them to another page to make it more secure. The customer can change their security question in that page. 7. The customer wants to check their order history they can click on the appropriate button and proceed to the order history page.

Table 7: Use Case A7 – Customer Changing Password

Use Case ID	A7

Use Case Name	Customer Changing Password
Case Description	Customer can reset their password securely in a different page.
Actors	Customer
Preconditions	1. Customer must have an account 2. Customer must be logged in
Main flow	1. System prompts the actor to enter their new password and to confirm their password. 2. Actor enters their new password and confirms the password 3. System verifies whether the inputs match each other. If the input matches, the new password is saved in the file.
Alternative flow	1. The actor enters a different password in the "confirm password" field. The system will display an error message. 2. The actor leaves the field empty. The system will display an error message. 3. The actor changes their mind and cancels the procedure using cancel button.

Table 8: Use Case A8 – Customer Changing Security Question

Use Case ID	A8
Use Case Name	Customer Changing Security Question
Case Description	Customer can change their security question and provide answer to that question. This would be beneficial to the customer if they forget their password and need to log in to their account.
Actors	Customer

Preconditions	1. Customer must have an account 2. Customer must be logged in
Main flow	1. System displays the actor their current security question and the security answer. 2. Actor can change the security question and answer based on their preference 3. The System accepts the input and saves the security question and answer after validating.
Alternative flow	1. The actor leaves the field empty. The system will display an error message. 2. The actor changes their mind and cancels the procedure using cancel button.

Table 9: Use Case A9 – Customer Checking Order History

Use Case ID	A9
Use Case Name	Customer Checking Order History
Case Description	Customer can check all the orders they have done previously and check the details, including feedback, of each order.
Actors	Customer
Preconditions	1. Customer must have an account 2. Customer must be logged int 3. Customer must have ordered at least once before
Main flow	1. System displays all the previous orders in a table with order ID, date and time of order, total quantity and amount paid. 2. User can select an order from the table and proceed to check the details of the order

Alternative flow	1. The actor does not select any item when trying to check order details. The system will display an error message. 2. The actor changes their mind and goes back to previous page

Table 10: Use Case A10 – Customer Checking Order Details

Use Case ID	A10
Use Case Name	Customer Checking Order Details
Case Description	Customer can see the details of the order they have done previously. The details include order ID, date and time of order, feedbacks, items they have order and their quantity, price of each item and grand total price paid.
Actors	Customer
Preconditions	1. Actor must have an account 2. Actor must be logged in 3. Actor must have at least one previous order
Main flow	1. Actor must select and click on the Check button. 2. Actor can see the details of the order but cannot edit anything. 3. The details include the order ID, time and date of the order, their feedback and ratings, names and quantity of the items they have ordered, the price of each item and lastly the grand total price paid. 4. After checking the order details, they can go back to the previous page.
Alternative flow	None

Table 11: Use Case A11 – Logout

Use Case ID	A11
Use Case Name	Logout
Case Description	Customer and Manager can logout and return to the login page once they are done using the system for their work.
Actors	Customer and Manager
Preconditions	1. Actor must have an account 2. Actor must be logged in
Main flow	1. Actor must navigate to the "Logout" button and press it 2. The system will redirect them to the login page. 3. The actors will be logged out and their details and history will be hidden from other actors.
Alternative flow	None

Table 12: Use Case A12 – User Management

Use Case ID	A12
Use Case Name	User Management
Case Description	Manager can view the list of all users and their information in a table. They can then select a user and proceed to check their details and update them if necessary.
Actors	Manager
Preconditions	1. Manager must be logged in

Main flow	1. The system will display a table that has information about the customer which allows the manager to take a quick glance at the names. 2. The manager can search for the specific customer using the search functionality 3. The manager can select a user they want to check the details or edit their details and then click on Check button. 4. The system will show the next window with the details of the user
Alternative flow	1. The manager changes their mind and goes back to manager dashboard. 2. The manager does not select any user and clicks on Check button. The system will show an error message.

Table 13: Use Case A13 – Manage Customer

Use Case ID	A13
Use Case Name	Manage Customers
Case Description	Manager can view and alter the customer's registration information. The manager can also reject or accept the customer's registration.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The system will display a table that has information about the customer which allows the manager to take a quick glance at the names.

	2. The manager can search for the specific customer username and edit it. 3. The system will redirect the manager to a new page that contains the registration information of the customer. 4. The manager changes the status of the customer to active or rejected 5. The system saves the status of the customer. If the customer has an active status, it means that the account has been created.
Alternative flow	1. The manager changes the detail for the customer. This may happen if the customer accidently enters wrong detail during their registration. After altering with the customer's basic details, the system saves the information. 2. The manager resets the customer password for them. The system will prompt the manager for a password and to confirm the password. Once the manager keys in the password together with the "confirm password" field, the system verifies it to check whether both fields are equal or not. If the fields are equal, then the customer's password is successfully changed, and the system saves the information. 3. The manager wants to check their order history and clicks on the appropriate button. The system takes the manager to the order report page and puts the email of the customer in the search text field which filters out the orders made by that specific customer. 4. The manager wants to check the feedback history of that customer and the system will show the feedbacks from that customer just like order history. 5. The manager cancels the procedure by clicking on Back button.

Table 14: Use Case A14 – Creating Customer Account

Use Case ID	A14
Use Case Name	Creating Customer Account
Case Description	The manager has the ability to register new customers in the scenario where the user cannot register themselves.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The system will prompt the manager to enter the details of the customer such as their username, their first name, last name, password, email, telephone number, security question and the answer to the security question. 2. After the manager fills up the form, the system will verify the registration information that it receives. 3. The system creates a new account for the customer after validating.
Alternative flow	1. The manager enters username that is taken for the user. The system will display an error message. 2. The manager leaves any text field empty, and the system will display an error message. 3. The manager cancels the procedure, and the system goes back to User Management page. 4. The manager provides invalid phone number, and the system shows an error message. 5. The manager provides non-APU email, and the system shows an error message.

Table 15: Use Case A15 – Changing Manager Password

Use Case ID	A15
Use Case Name	Changing Manager Password
Case Description	The manager can change their own password. However, there is an added security feature where the manager can only change the password if the old password is known. This is to ensure that only the manager can change it.
Actors	Manager
Preconditions	1. The manager must be logged in
Main flow	1. The system prompts the manager for the old password, the new password and to confirm the new password. 2. After the manager fills up the form, the system will verify whether the old password matches the current manager's password and whether the new password matches with the field that confirms the new password. 3. After the system successfully verifies the passwords, the new password is saved and stored.
Alternative flow	1. The manager enters the old password wrongly. The system will display an error message to the manager. 2. The manager leaves any of the text fields empty. The system will display an error message to the manager. 3. The manager cancels the procedure by clicking on the Cancel button.

Table 16: Use Case A16 – Manager Checking Order History

Use Case ID	A16
Use Case Name	Manager Checking Order History
Case Description	The manager can view all order history in an organized manner. The manager can view all orders and purchases done by the customer in the Cafeteria Ordering System in a chronological manner. It also gives the manager insight on the customer that orders the food, the time and date that the customer ordered the food as well as the amount of money the customer spent on the food. The manager can use a search feature to look for specific customers.
Actors	Manager
Preconditions	1. Manager is logged in
Main flow	1. The system displays the ID of the order, the name of the customer, the time and date that the meal was ordered and the total amount of money that the customer paid for the order to the manager. 2. The system displays the data in a table in a chronological order so that it is easier for the manager to view the customers and the food that they bought. 3. The system allows the manager to search for a specific username and it only shows orders that were ordered by the specific username. 4. The manager can select an order from the table and proceed to check the details of that order.
Alternative flow	1. The manager changes their mind and goes back to the dashboard

	2. The manger does not select an order when trying to check the order details. The system will show an error message.

Table 17: Use Case A17 – Manager Checking Order Details

Use Case ID	A17
Use Case Name	Manager Views Order Details
Case Description	The manager is able to view the details of every order. The manager can view the details of the customer that made the order, the unique order ID that's given to each order to differentiate the orders, the dishes that were ordered, the identifier for the customer credit card for verification purposes, and the feedback given by the customer.
Actors	Manager
Preconditions	1. Manager is logged in 2. Customer has bought, paid and left feedback for the order.
Main flow	1. The system displays the ID of the order, the name of the customer, the customer's contact details, the last 4 digits of the credit card, the dishes that were ordered, the time and date that the dish was ordered, and the feedback given by the customer. 2. After viewing the information about the order, the manager can return to the dashboard or to check for more details on the user.
Alternative flow	1. The manager can edit the user's account settings by pressing the *Check User* button. The manager can deactivate the user's account if the manager wishes to do so. 2. The manager can check the order history of that user by clicking on *Order History* button. Which will redirect them to the Order History page and put the email of this user in the search box.

	Because of this the table will only show the orders made by this user 3. The manager can check the feedback history of this user by clicking on *Feedback History* button. This button will take the manager to the Feedback page and the email of this user in the search box.

Table 18: Use Case A18 – Manager Checking Feedback History

Use Case ID	A18
Use Case Name	Manager Checking Feedback History
Case Description	The manager can view the feedback that was submitted by the customers.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The system displays a table that contains the order ID, the name of the customer that made the order, the contact details of the customer, the time and date that the customer ordered the dish, and the rating with the written feedback that the customer gave. Only the orders with feedbacks show up in this table. 2. The manager can click on the individual order ID to view the feedback provided by the customer in greater detail. The manager can either click on the specific order ID in the table or use the search function and search for the customer's unique ID. 3. The manager can then proceed to check the order details along with feedback of that order.

Alternative flow	1. The manager changes their mind and goes back to the dashboard. 2. The manager does not select an order when trying to check the feedback of the order. The system will show an error message.

Table 19: Use Case A19 – Manager Generating Order Wise Report

Use Case ID	A19
Use Case Name	Manager Generating Order Wise Report
Case Description	The manager can generate an order wise report where all the orders are sorted in ascending order of the order ID. This report will include date and time of the order, total quantity and price paid for the order.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The system will display all the orders inside a table in ascending order along with the price paid, item quantity, date and time of the order. At the bottom of the table, there is a grand total. 2. The manager can filter out the table based on need using the filters provided. The system provides three filters: ID range, date range and amount range. The manager can use single or multiple filters at a time. This help the manager generate daily, weekly or monthly report.
Alternative flow	1. The manager can cancel the procedure and go back to the dashboard. 2. The manager can enable a filter and leave it empty. The system will display an error when the *Filter* button is clicked. 3. The manager can provide invalid input in the filters text fields.

Table 20: Use Case A20 – Manager Generating Item Wise Report

Use Case ID	A20
Use Case Name	Manager Generating Item Wise Report
Case Description	The manager can generate an item wise report where all the items are shown along with total quantity sold and amount of products sold.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The System will show a table with 23 rows as we have 23 items in the menu. Where first column is the item name, then quantity and lastly total price. 2. The manager can filter out the table based on need using the filter provided. The system provides only one filter. Which is date range. So, the manager can generate item wise report which would help them analyze the customer's need.
Alternative flow	1. The manager goes back using *Back* button. 2. The manager clicks on the *Filter* button after enabling the filter without putting any value. This will display an error message. 3. If the date filter does not have a valid input, it will display an error message.

Table 21: Use Case A21 – Updating Menu

Use Case ID	A21
Use Case Name	Manager Updates Menu
Case Description	The manager can update the menu by changing the availability of the dish to either available or unavailable or changing the price of the dish.
Actors	Manager
Preconditions	1. Manager must be logged in
Main flow	1. The system will display all the items in the menu, the price of each item and the availability of the item in a table to the manager. There are a total of 23 items in the menu, hence the table will display 23 rows of different dishes. 2. The manager can select any item from the table and change the price of the item by either increasing it or decreasing it. 3. The manager can also change the availability of the item by setting the availability of an item to not available. 4. The system will save the changes and update the price of the items and the availability of each item to the menu. 5. If a dish is not available, it can't be ordered from the menu.
Alternative flow	1. The manager goes back to the dashboard using *Back* button.

2.3 Class Diagram

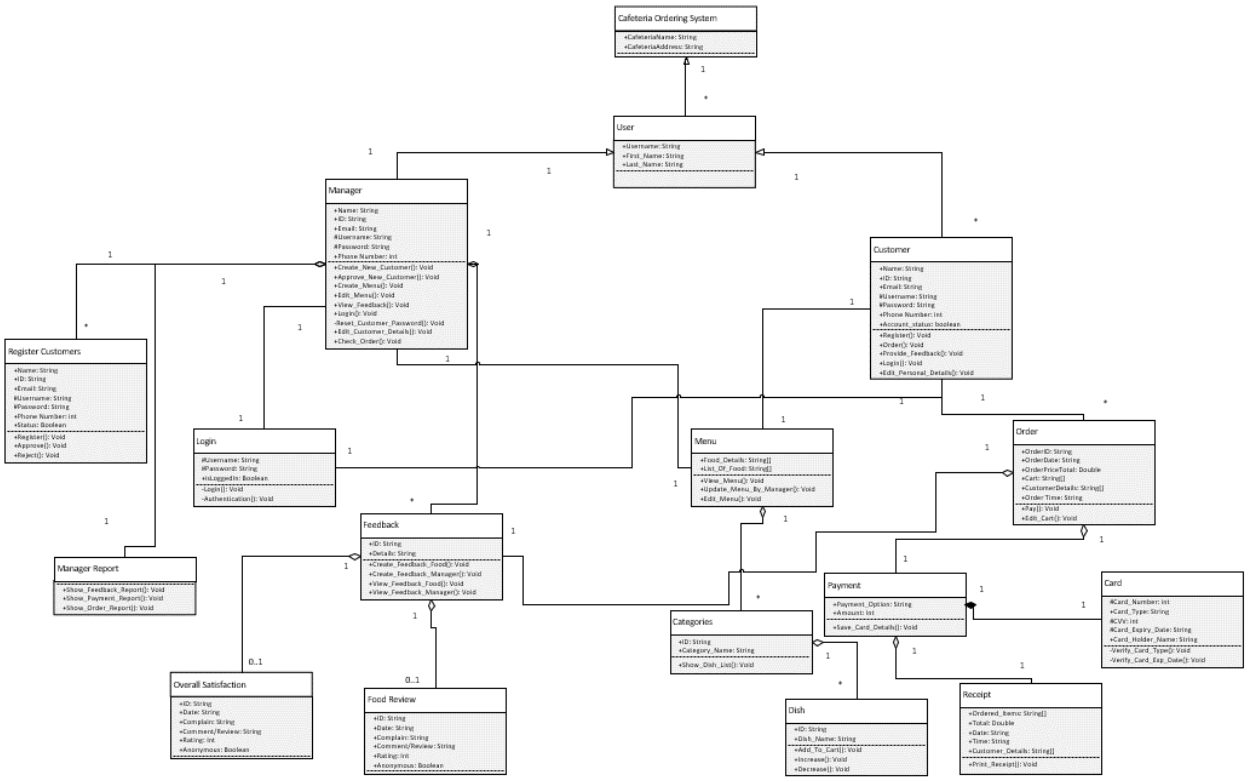


Figure 2: Full view of class diagram

This is the full view of the class diagram. There are several classes that are in the class diagram which are the cafeteria ordering system class, the manager class, the customer class, the registration class, the manager report class, the login class, the feedback class, the overall satisfaction class, the food review class, the menu class, the categories class, the order class, the payment class, the card class, the receipt class and the dish class. The relationship between classes is expressed by using aggregation, association, and composition. Association shows that a relationship exists between two classes. Aggregation shows that a class is a part of another class whereas composition is a relationship in which a class needs the other class to exist. The cardinality of the classes is also shown in the class diagram.

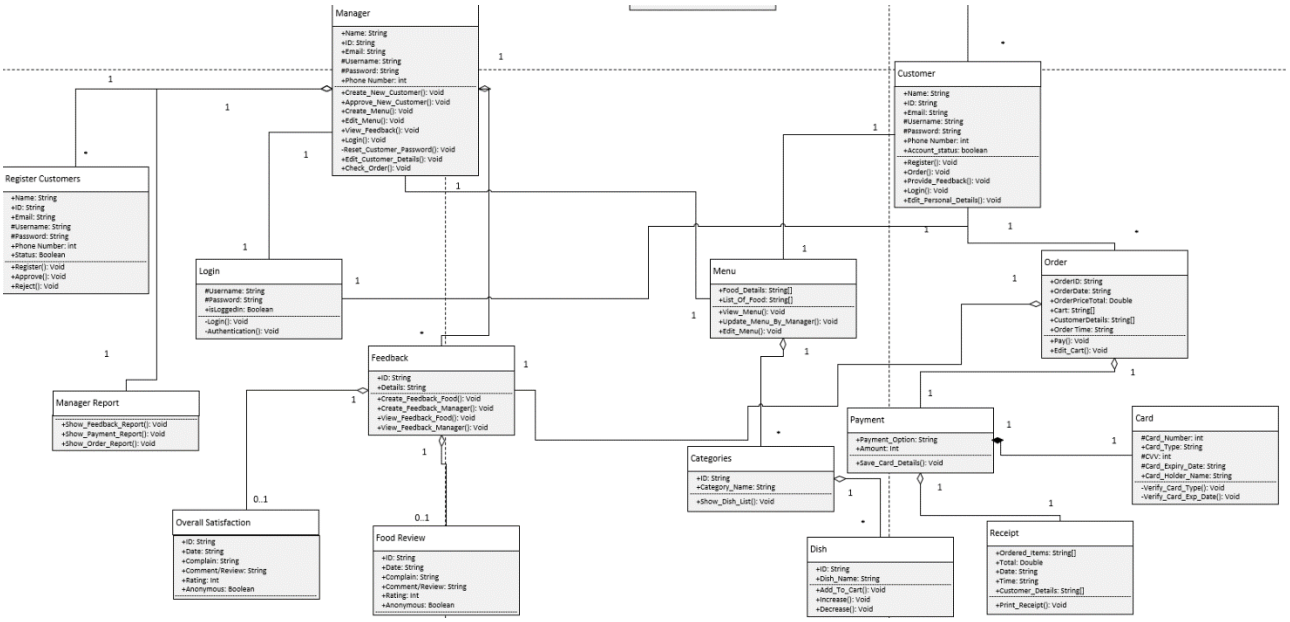


Figure 3: View of the connections between the manager class and the customer class

It can be seen from the figure above that the manager class and customer class have an association with the menu class. The cardinality of the manager class is 1 to 1 with the menu class. The cardinality of the customer class is also 1 to 1 with the menu class. The manager class and the customer class have an association with the login class. The cardinality of the manger class is 1 to 1 with the login class. The cardinality of the customer class is 1 to 1 with the login class.

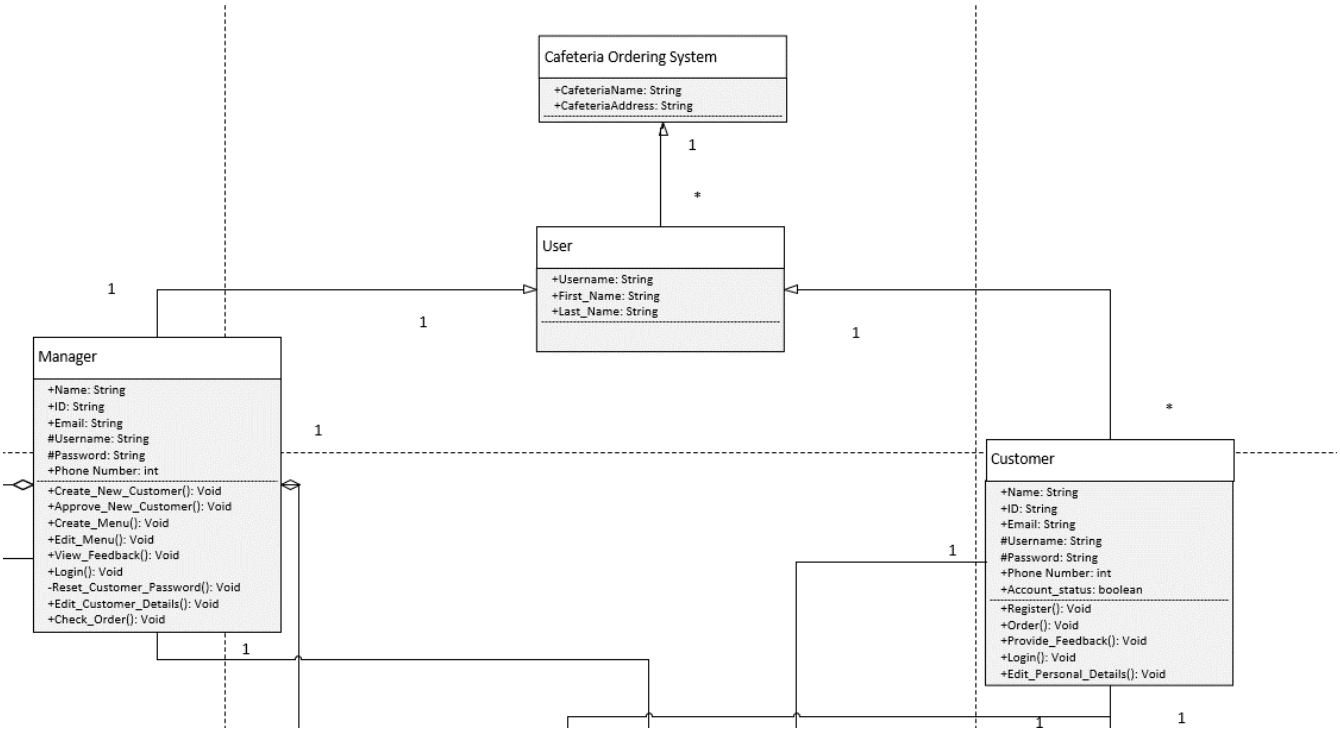


Figure 4: View of the connection between the user class and its subclass

These are the attributes and methods for the manger class, the customer class, the user class and the cafeteria ordering system class. The cardinality of the user class with the manager class is 1 to 1, this is because there is only 1 manager at a time. The cardinality of the user class with the customer class is 1 to many, this is because there is more than one customer. The cardinality of the cafeteria ordering system class with the user class is 1 to many, this is because there are many users.

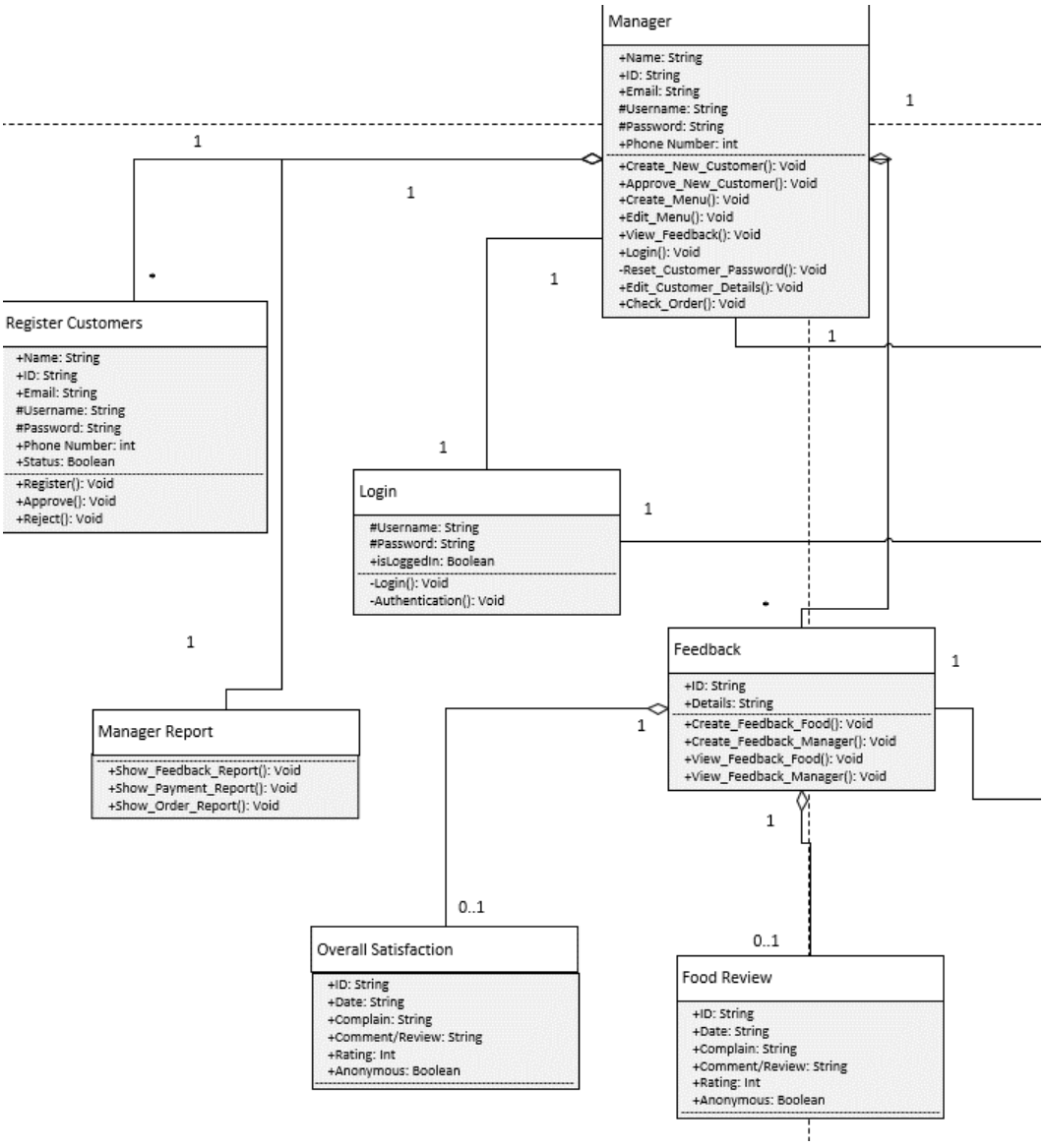


Figure 5: Classes that has a relationship with Manager class

These are the attributes and methods of the register customers class, the login class, the feedback class, the manager report class, overall satisfaction class, the food review class, and the feedback class. The cardinality of the manager class to the register customers class is 1 to many because one manager can create many users. The cardinality of the manager with the manager report is 1 to 1 because at one given point in time the manager can generate one report. The cardinality of the manager class with the feedback class is 1 to many this is because the manager can view several reports made by the customer. The cardinality of the feedback class with the food review class is 1 to 0 or 1, this is because the customer has a choice to not give feedback. The

cardinality of the feedback class with the overall satisfaction class is also 1 to 0 or 1, this is because the customer ha a choice to not give feedback.

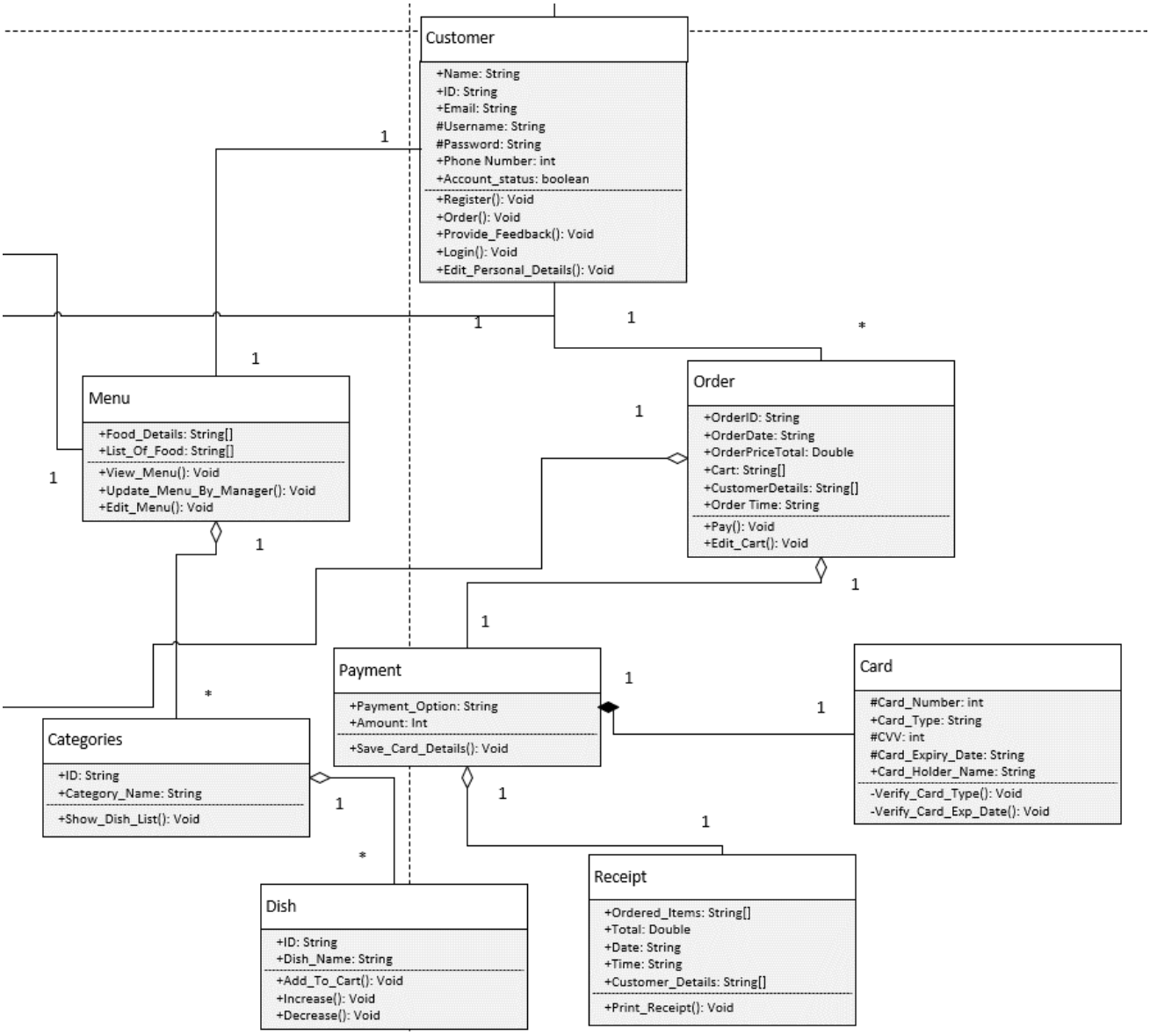


Figure 6: Classes that has a relationship with Customer Class

These are the attributes and methods of the menu class, the categories class, the payment class, the dish class, the receipt class, the card class and the order class. The cardinality of the customer class to the menu class is 1 to 1, this is because 1 customer will only utilize 1 menu. The cardinality of the customer class with the order class is 1 to many, this is because one customer can order more than one dish. The cardinality of the menu class with the categories class is 1 to

many, this is because there are several categories in a menu. For example, there is the meat category, the fish category, the rice category, the noodle category and other food categories. The cardinality of the categories class with the dish class is 1 to many. This is because on category can have many different dishes. The cardinality of the order class with the payment class is 1 to 1 as each order only needs to be paid once. The cardinality of the payment class is 1 to 1 with the card class. This is because each payment only requires one credit card to pay for it. The relationship between the payment class and the card class is composition because the card class cannot exist without the payment class. The cardinality of the payment class with the receipt class is 1 to 1. This is because after paying, the customer will receive 1 receipt.

3.0 Sample Source Code

3.1 Header files

```
import java.io.*;
import javax.swing.JOptionPane;
import java.awt.event.KeyEvent;
```

There are three header files as shown above, the first header file is java.io*. This header file provides for system input and output through data streams. The asterisk imports all the classes that fall under java.io. For example, java.io.Console is under java.io, and the purpose of it is to print and take user input.

The next header is javax.swing.JOptionPane. The JOptionPane is a class that has built in methods that allows it to create pop up box to inform the users about something.

The next header is the java.awt.event.KeyEvent. The KeyEvent is a class that is used to determine whether a keystroke has occurred on the keyboard. For example, it can determine whether the enter key on the keyboard is pressed or not.

```
import javax.swing.JLabel;
```

In addition to that, the header javax.swing.JLabel, is also used. The JLabel class displays a label which is an area that can be used for a short string text or an image. A label does not react to input from the user as it only displays things to them. The label's position can be specified.

```
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
```

The javax.swing.table.DefaultTableCellRenderer is also a header that is used. The class DefaultTableCellRenderer is used to render and display individual cells in a JTable.

The javax.swing.table.DefaultTableModel is another header that is ued. The class Default Tablemodel is using a vector of vectors to store the cell value of objects in the table.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
```

The java.io.BufferedReader is a header that is used. The BuffferedReader is a class that reads text from a character-input stream from a text file. It buffers the characters to provide more efficient reading of characters, arrays, and lines.

The java.io.file is also another header that is used. The File is a class that represents the file and directory pathnames.

The java.io.IOException is also another header that is used. The IOException is a class that signals that an input output exception has occurred. This is used in the try-catch block.

```
import java.io.FileOutputStream;
```

The java.io.FileOutputStream is also another header that is used. The FileOutputStream is a class that is an output stream for writing data to a file.

```
import javax.swing.RowFilter;
```

The javax.swing.RowFillter is also another header that is used. The RowFilter is a class that is used to filter out entries from the table so that they aren't shown in the view.

```
import javax.swing.table.TableRowSorter;
```

The javax.swing.table.TableRowSorter is also another header that is used. The TableRowSorter is a class that used to implement the RowSorter which provides sorting and filtering to the JTable.

```
import java.text.SimpleDateFormat;
```

The java.text.SimpleDateFormat is also another header that is used. The SimpleDateFormat is a class that is used to format and parse date in in a locale sensitive manner. It allows date to be formatted into text and it parses text to date.

```
import java.util.Date;
```

The java.util.Date is a header that is used. The Date is a class that represents a specific instant in time, with a precision down to the milliseconds.

```
import java.time.format.DateTimeFormatter;
```

The java.time.format.DateTimeFormatter is also another header that is used. The DateTimeFormatter is a class that formats printing and parsing of date-time objects.

```
import java.time.LocalDateTime;
```

The java.time.LocalDateTime is a header that is used. The LocalDateTime is an immutable class that represents a date time in the ISO-8601 calender system.

```
import java.io.FileWriter;
```

The java.io.FileWriter is a header that is used. The FileWriter is a class that is used to write text to character files using a default buffer size.

```
import java.util.Arrays;
```

The java.util.Arrays is also another header that is used. The Arrays is a class that is used to manipulate arrays with its various methods.

3.2 Variables

```
int id = 1;
String username = jTextField3.getText();
```

Some of the data type of variables that are used in the program are integers, and strings. In the example above the variable with the name "id" of data type integer is created and set to the value of 1. Besides that, another variable with the name of "username" of data type string is created and set to the value of the text field that is located in the Jform.

```
char[] pass = jPasswordField2.getPassword();
```

Besides that, a variable with the name of "pass" of data type character array is created and it is set to the password field that is located in the JForm. The password field keeps the text that is inputted by the user as a character, therefore it is read as a character array and will be converted into a string later on.

```
double unitPrice = Double.parseDouble(itemalprice.getText().substring(2));
```

In addition to that, another variable with the name of "unitPrice" of data type double is declared and set to the value of the number in a Jlabel. A double is different form an integer as a double has decimal places whereas an integer doesn't.

```
boolean closeRegister = false;
```

An attribute with the name of "close register" of data type Boolean is declared and set to false. A Boolean data type can either be true or false only.

3.3 Control Structure

3.3.1 If

```
if(username.length() == 0)
{
    JOptionPane.showMessageDialog(this, "Username cannot be empty", "Username Error", JOptionPane.ERROR_MESSAGE);
    return;
}
```

This is part of a validation that is used, and it uses a control structure. The validation uses an if block to verify whether the length of the username that is entered by the user is equal to 0. If the username length is 0, which means the user left the text field empty, a pop-up message will indicate that there is an error, and that the username of the user can't be empty.

3.3.2 If Else

```
if (this.table.getRowCount() == 0)
{
    JOptionPane.showMessageDialog(this, "Cart is empty", "Order Error", JOptionPane.ERROR_MESSAGE);
}
else
{
    this.setVisible(false);

    Payment pf = new Payment(this.user, this.table);
    pf.setLocation(this.getLocation());
    pf.setResizable(false);
    pf.setVisible(true);
}
```

This is another validation that is used when the customer is ordering food, it uses the if else block control structure. The if block determines whether the number of rows in the table is equal to 0, if the number of tables is equal to 0, then there would be a pop-up box that displays an error message to the user, telling them that their cart is empty.

The else block occurs when the condition that the if block is checking is false. The else block hides the order page and creates an object which is the payment page and makes the object visible. The else block will run only when the number of rows in the table is not 0.

3.3.3 Nested If Else

```
if (file.exists())
{
    BufferedReader data;
    data = new BufferedReader(new FileReader(file));
    String line;
    data.readLine();
    while((line = data.readLine()) != null)
    {

        if(username.compareTo(line.split("\t")[1]) == 0)
        {
            JOptionPane.showMessageDialog(this, "Username already taken", "Username Error", JOptionPane.ERROR_MESSAGE);
            data.close();
            return;
        }

    }

    data.close();
}
```

If we ignore the while loop in the figure above, we can see that there is an if-else block inside another if-else block. The second if-else is the nested if-else in this scenario. The first if-else block is checking whether the file exists or not. If the file exists it will go inside and reach the nested if-else block. But if the files do not exist, then the program will skip over the nested if-else block. As for the second if-else block, it is comparing the *username* with the second element of every line. If they are the same, then the program is going inside the nested if-else block. In order to execute the code inside the nested if-else block, both the condition of the if-else block need to be true.

3.4 Looping structure

3.4.1 While loop

```
while((line = data.readLine()) != null)
{
    if(this.id.compareTo(line.split("\t")[0]) == 0)
    {
        this.user = line;
        break;
    }


}
```

A while looping structure is used in this code snippet. The while loop will run until its condition is false. In this code snippet, the while loop will read until the end of file. The readLine method reads the file line by line. It returns null when it reaches the end of file. We are assigning the line that we read inside the line variable and use it in each loop. When the line variable value is equal to null, the loop stops. Inside the loop, the line variable is split into an array. The if block checks whether the id of the order is equal with the value of the first element in the line array. If it is equal, then the user is attribute is set to the value of line.

3.4.2 For Loop

```
for (int i = 0; i < username.length(); i++)
{
    char b = username.charAt(i);
    if (!((b >= 'a' && b <= 'z') || (b >= 'A' && b <= 'Z') || (b >= '0' && b <= '9') || b == '-' || b == '_' || b == '.'))
    {
        JOptionPane.showMessageDialog(this, "Username can only contain letters, numbers, dot, dash and underscore", "Username Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
}
```

This code snippet is used to validate the username that is entered by the user in a text field. The validation is assisted by a for loop. The for-loop loops for the length of the username. For example, if the username has 7 letters, the for-loop will loop for 7 times. The code snippet then will take the index of each letter of the username and compare it with letters that we allow for the username. For example, the if block checks whether the character index is an alphabet of small or capital letters, or a dot, or a dash or an underscore. If the character index doesn't equal to those, then a pop up box will display an error message that tells the user that the username can only contain letters, numbers, dot, dash and underscore.

4.0 OOP concepts

4.1 Class

```
public class Manager extends User{


    private char[] oldPass;
    private char[] pass;
    private char[] passConfirm;



    public char[] getNewPass(){
    //get method
        return pass;
    }
     public void setNewPass(char[] oldPassword, char[] password, char[] passwordConfirm){
        oldPass = oldPassword;
        pass = password;
        passConfirm = passwordConfirm;
          //Pass length 8-20
        if(oldPass.length == 0)
        {
            JOptionPane.showMessageDialog(null, "Must provide old pass", "Password Error",
            return;
        }
```

A class is a blueprint which will be used to create objects. It represents the set of properties or methods that the object has. In the code snippet above, there is a class with the name “Manager”. The class has several attributes which are all private, these include *oldPass*, pass, and *passConfirm*. In addition to that, there are two visible methods in this class which are the *getNewPass* method and *setNewPass* method. When an object is created from this class, the object will have all the methods and attributes of this class.

4.2 Object

```
Login lf = new Login();
lf.setVisible(true);
lf.setLocation(320, 100);
lf.setResizable(false);
```

In the code snippet above, lf is the object created from the Login class. The object will have all the attributes, and methods of the Login class. The setVisible, setLocation, and setResizable method belongs to the Login class, but since lf is the object that was created from the Login class, lf can use those methods.

4.3 Encapsulation

Encapsulation is a way to protect data from being accessed by other classes.

```
private char[] oldPass;
private char[] pass;
private char[] passConfirm;
```

These are the private attributes that are declared in the Manager class. Private attributes cannot be accessed by other classes and can only be accessed by the Manager class, since the attributes is declared in it.

```
public char[] getNewPass(){
    return pass;
}
```

This is the get method that is in the Manager class. It is a public method which means it can be called by other classes. The purpose of this method is to return the value of the private attribute. By setting the method as a public method, it allows the other classes to get the value of a private attribute of another class.

```
public void setNewPass(char[] oldPassword, char[] password, char[] passwordConfirm){
    oldPass = oldPassword;
    pass = password;
    passConfirm = passwordConfirm;
      //Pass length 8-20
    if(oldPass.length == 0)
    {
        JOptionPane.showMessageDialog(null, "Must provide old pass", "Password Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    try
    {

        BufferedReader data;
        data = new BufferedReader(new FileReader("files/managers.txt"));
        StringBuilder inputBuffer = new StringBuilder();
        String line;
        line = data.readLine(); // Read the header row
        inputBuffer.append(line).append("\n");




        line = data.readLine();
        data.close();
```

This is a set method, that is in the Manager class. It is a public method which means that the other classes can call it. The purpose of this method is to set the private attribute of the Manager to a specific value. There is also a validation to determine whether the data entered by the user is

valid or not. Therefore, this public set method allows the users to access and change the value of the Manager class private attribute.

4.4 Constructor

A constructor is a special method that is used to initialize objects. With a constructor, the values of the attributes can directly be set when the object is created.

```
public class Customer extends User{
    int id = 1;
    String fname;
    String lname;
    char[] pass;
    char[] passConfirm;
    String email;
    String number;
    String securityQ;
    String securityAns;
    boolean closeRegister = false;
```

These are the attributes in the Customer Class that is a child class from the User class.

```
Customer(String username1, String fname1, String lname1, char[] pass1, char[] passConfirm1, String email1, String number1, String securityQ1, String securityAns1){
    username = username1;
    fname = fname1;
    lname = lname1;
    pass = pass1;
    passConfirm = passConfirm1;
    email = email1;
    number = number1;
    securityQ = securityQ1;
    securityAns = securityAns1;
}
```

This is the constructor method that is in the Customer Class. A constructor method is a method with the same name as the class itself. The values of the attributes are set to the values of the parameters of the constructor. Therefore, when a Customer object is created, the attributes will already have a value that is set to it.

4.5 Get-Set Method

```
private char[] oldPass;
private char[] pass;
private char[] passConfirm;
```

These are the attributes of the Manager class.

```
public char[] getNewPass(){
    return pass;
}
```

The getNewPass method gets the value of the pass attribute from the Manager class. Since the getNewPass method is a public method, other classes can call it.

```
public void setNewPass(char[] oldPassword, char[] password, char[] passwordConfirm){
    oldPass = oldPassword;
    pass = password;
    passConfirm = passwordConfirm;
      //Pass length 8-20
    if(oldPass.length == 0)
    {
        JOptionPane.showMessageDialog(null, "Must provide old pass", "Password Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    try
    {

        BufferedReader data;
        data = new BufferedReader(new FileReader("files/managers.txt"));
        StringBuilder inputBuffer = new StringBuilder();
        String line;
        line = data.readLine(); // Read the header row
        inputBuffer.append(line).append("\n");




        line = data.readLine();
        data.close();
```

The setNewPass method is a public method which means it can be called from other class. The purpose of this method is to set the value of the attribute to the value of its parameters. After setting the value of the attribute of an object that's created from the Manager class, the value of the attribute can be retrieved by using the get method. In this scenario, the get method is the getNewPass method.

4.6 Inheritance

```
public class Manager extends User{


    private char[] oldPass;
    private char[] pass;
    private char[] passConfirm;
```

```
public class User {
    public String username = "manager";
    public String firstName;
   public boolean flag = false;

    public boolean getFlag(){
        return flag;
    }
}
```

The Manager class inherits attributes and methods from the User class. This is because of the extends keyword, the Manager class is extended from the User class. Therefore, the Manager class is the subclass of the User class. The User class has attributes and methods such as username, firstName, flag, and getFlag. These attributes and methods are inherited to the Manager. This means that the manager also has a string username attribute value with the value of "manager".

4.7 Polymorphism

4.7.1 Overloading

```
public class User {
    public String username = "manager";
    public String firstName;
   public boolean flag = false;

    public boolean getFlag(){
        return flag;
    }
}
```

This is the User class, it is the superclass of the Customer class. The method that is inherited by the Customer class is the *getFlag* method.

```
public class Customer extends User{
    int id = 1;

    String fname;
    String lname;
    char[] pass;
    char[] passConfirm;
    String email;
    String number;
    String securityQ;
    String securityAns;
    boolean closeRegister = false;
```

The Customer class is the subclass of the User class and in addition to the attributes and methods that it inherits from the User class, it also has unique attributes and methods of its own. It has an attribute named “closeRegister” that is of data type boolean and the value is set to false.

```
public boolean getFlag(){
    return closeRegister;
}
```

The Customer class has a method that has the same name as the method that it inherited from its superclass, but it returns the value of a different attribute, which is the value of the closeRegister attribute. It returns the closeRegister attribute instead of the flag attribute because of an OOP concept known as overloading.

4.7.2 Overriding

```
public class User {
    public String username = "manager";
    public String firstName;
   public boolean flag = false;

    public boolean getFlag(){
        return flag;
    }
}
```

This is the superclass User and it has several attributes which includes username. The username is a public string attribute that is initialized to “manager”.

```
public class Customer extends User{
    int id = 1;

    String fname;
    String lname;
    char[] pass;
    char[] passConfirm;
    String email;
    String number;
    String securityQ;
    String securityAns;
    boolean closeRegister = false;

    Customer(String username1, String fname1, String lname1, char[] pass1, char[] passConfirm1, String email1, String number1, String securityQ1, String securityAns1){
        username = username1;
        fname = fname1;
        lname = lname1;
        pass = pass1;
        passConfirm = passConfirm1;
        email = email1;
        number = number1;
        securityQ = securityQ1;
        securityAns = securityAns1;
    }
```

This is the subclass Customer, it inherits the attributes and methods from the superclass, User. Therefore, the Customer will also have the username attribute. However, the username is

initialized to the value of the parameter in the constructor. Therefore, the initial username attribute that is in the Customer class is overridden by the value that is passed to the constructor.

4.8 Exception Handling

```
try
{

    BufferedReader data;
    data = new BufferedReader(new FileReader("files/managers.txt"));
    StringBuilder inputBuffer = new StringBuilder();
    String line;
    line = data.readLine(); // Read the header row
    inputBuffer.append(line).append("\n");



    line = data.readLine();
    data.close();
    if(String.valueOf(oldPass).compareTo(line.split("\t")[1]) != 0)
    {
        JOptionPane.showMessageDialog(null, "Incorrect Old Password", "Password Error", JOptionPane.ERROR_MESSAGE);
        return;

    }

}
catch(IOException e)
{

    e.printStackTrace();
    flag = false;
}
```

Errors can occur during the execution of Java code. The try statements defines a block of code that might have an error when it is executed. It tries to run that block of code. If it fails to run at any point, the *catch* block continues from there based on the type of Exception or error the *try* block throws. In the example above, the try statement opens a file and compares the value of the old password with the password that is stored in a text file. However, if there was an error that occurs while opening the text file, the catch block will be executed. In this, case the flag attribute will be set to false.

4.9 File Concepts

4.9.1 Write Data Code

```
try {
    File folder = new File("files");
    if(!folder.exists())
        folder.mkdir();


    File file = new File("files/users.txt");

    if (file.createNewFile()){
        FileWriter fw;
        fw = new FileWriter("files/users.txt", true);

        fw.write("ID\tUsername\tFirst Name\tLast Name\tEmail\tPassword\tNumber\tStatus\tSecurity Q\tSecurity Ans\n");
        fw.close();
    }
}
catch (IOException e) {
    e.printStackTrace();
}
```

First part of the code snippet above is used to check if the folder and file exists or not. If the folder does not exist, it creates the folder. Then tries to create the file using *createNewFile* method, which returns true if it can create the file and returns false if the file already exists. If it successful in the creation of the file, it is writing into it using *write* method. The whole block of the code is inside a *try-catch* block as a standard procedure with a *IOException* event.

4.9.2 Read Data Code

```
try
{
    File file = new File("files/users.txt");
    if (file.exists())
    {
        BufferedReader data;
        data = new BufferedReader(new FileReader(file));
        String line;
        data.readLine(); // Read the header row
        while((line = data.readLine()) != null)
        {
            String id = line.split("\t")[0];
            String username = line.split("\t")[1];
            String fname = line.split("\t")[2];
            String lname = line.split("\t")[3];
            String email = line.split("\t")[4];
            String status = line.split("\t")[7];

            table.insertRow(table.getRowCount(), new Object[] {id, username, fname, lname, email, status});

        }
        data.close();
    }

}
catch(IOException e)
{
    e.printStackTrace();
}
```

In the code snippet, it is checking if the file exists or not. If it exists, then it reads it line by line using *readLine* method. This method returns *null* when it reaches end of file. Until then it reads the file using a while loop. Inside the while, it does the necessary processes. In this case, it is splitting line into array and assigning the value into the variables. After reading the file, at the end of while loop, it closes the file. The whole block is then put inside a *try-catch* block with *IOException* handling.

4.9.3 Search Data Code

```
try
{

    BufferedReader data;
    data = new BufferedReader(new FileReader("files/users.txt"));
    String line;
    data.readLine(); // Read the header row
    while((line = data.readLine()) != null)
    {

        if(this.id.compareTo(line.split("\t")[0]) != 0 && username.compareTo(line.split("\t")[1]) == 0)
        {
            JOptionPane.showMessageDialog(this, "Username already taken", "Username Error", JOptionPane.ERROR_MESSAGE);
            data.close();
            return;
        }

    }
    data.close();
}
catch(IOException e)
{

    e.printStackTrace();
}
```

The code snippet above is an example of searching through a file. At first, the file was opened for reading. In this scenario, it was sure that the file already exists because we have created it. Therefore no need to check if the file exists or not. Then we are reading the file line by line using *readLine* method. In each line, we are looking for the *id* and *username* provided by the user for a match. If a match is found, then the program shows an error message and returns after closing the file. On the other hand, if it does not find any match and reaches the end of file, then *readLine* will return *null*. Which will make the program to break out of the loop and close the file that we were reading. We used this code snippet for validating username.

4.9.4 Update Data Code

There is no built-in method for updating file data. So we have to use a combination of reading and writing to update the file.

```
try
{
    BufferedReader data;
    data = new BufferedReader(new FileReader("files/users.txt"));
    StringBuilder inputBuffer = new StringBuilder();
    String line;
    line = data.readLine(); // Read the header row
    inputBuffer.append(line).append("\n");
    while((line = data.readLine()) != null)
    {
        if(this.id.compareTo(line.split("\t")[0]) == 0)
        {
            this.user = line;
            line = line.split("\t")[0] + "\t"+ line.split("\t")[1] + "\t"+ line.split("\t")[2] + "\t" + line.spl
        }
        inputBuffer.append(line).append("\n");
    }
    data.close();

    FileOutputStream fileOut;
    fileOut = new FileOutputStream("files/users.txt");
    fileOut.write(inputBuffer.toString().getBytes());
    fileOut.close();

    JOptionPane.showMessageDialog(this, "Password has been reset successfully", "Password Update", JOptionPane.
}
catch(IOException e)
{
    e.printStackTrace();
}
```

In the first section of the code snippet above, it opens the file for reading. Then reads the whole file line by line using *readLine* method. If it finds the line it was looking for, it modifies it. After the program is done with a line, then it immediately gets written into a *Stringbuilder* named *inputBuffer.* After the cursor reaches the end of the file, *readLine* method returns *null* and it gets out of the while loop. After reading the whole file, it closes the file and opens again for writing using *FileOutputStream*. *FileOutputStream* is a bit different from *FileWriter* that has been used before to write. *FileOutputStream* writes into file byte by byte while *FileWriter* writes character by character. It overwrites the existing data and writes all the data from *inputBuffer* where we stored data of the file after modifying. For writing, the data was converted to primitive data type using *getBytes* method. Once the program is done writing into the file, it cloes the file. This whole chunk of the code is wrapped inside a *try-catch* block with *IOException* handling.

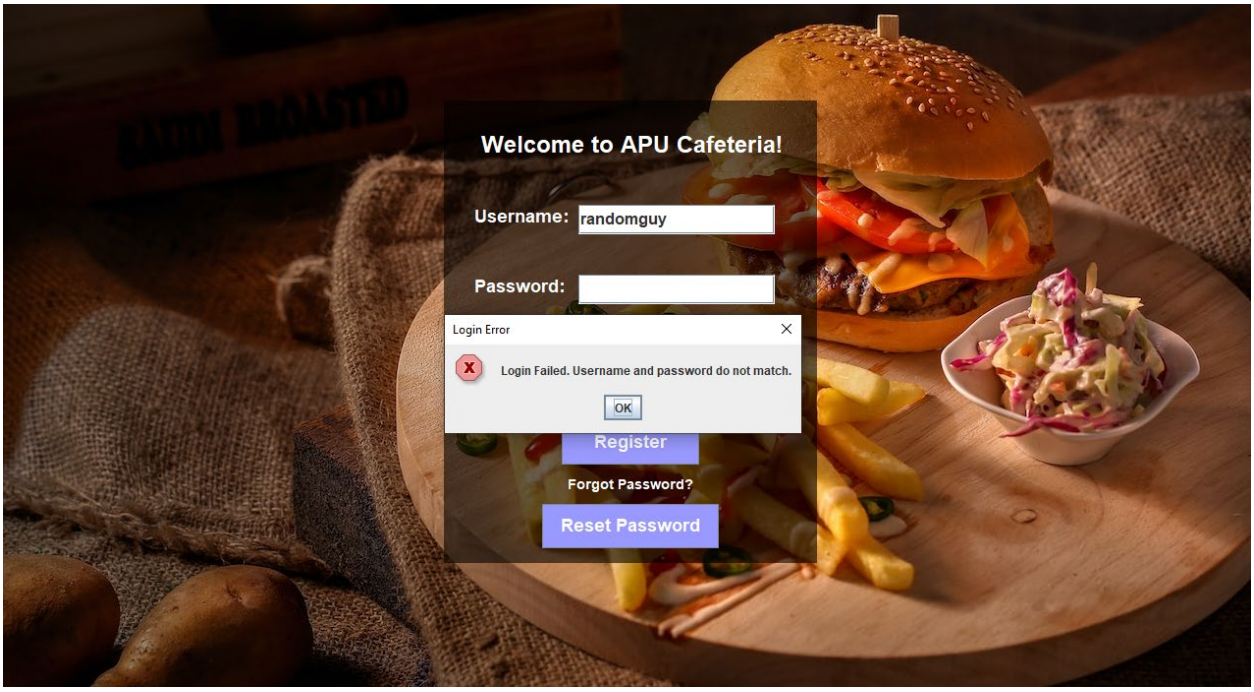
4.9.5 Display Data Code

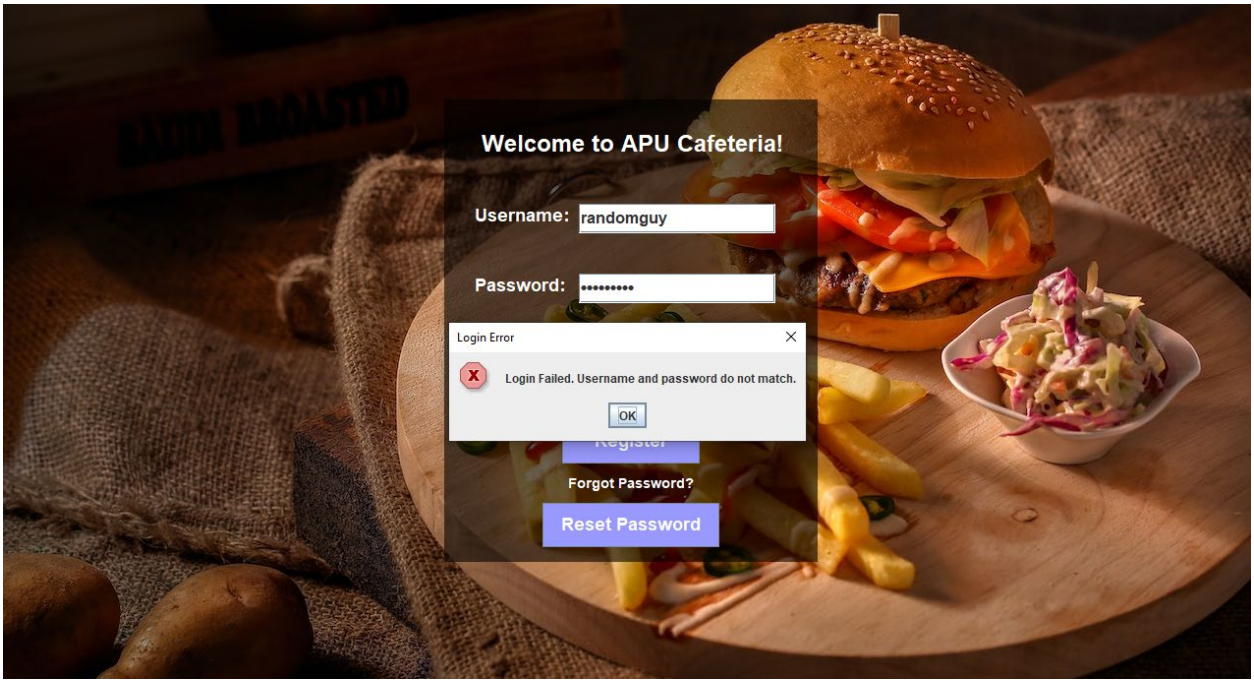
```
try
{
    File file = new File("files/orders.txt");
    if (file.exists())
    {
        BufferedReader data;
        data = new BufferedReader(new FileReader(file));
        String line;
        data.readLine(); // Read the header row
        while((line = data.readLine()) != null)
        {
            if(userId.compareTo(line.split("\t")[3]) == 0)
            {
                String id = line.split("\t")[0];
                String date = line.split("\t")[1];
                String time = line.split("\t")[2];
                double total = 0;

                for (int i = 0; i < (line.split("\t").length - 9)/4; i++)
                {
                    total += Double.parseDouble(line.split("\t")[9+i*4]);
                }
                table.insertRow(table.getRowCount(), new Object[] {id, date, time, String.format("%.2f", total)});
            }
        }
        data.close();
    }
}
catch(IOException e)
{
    e.printStackTrace();
}
```

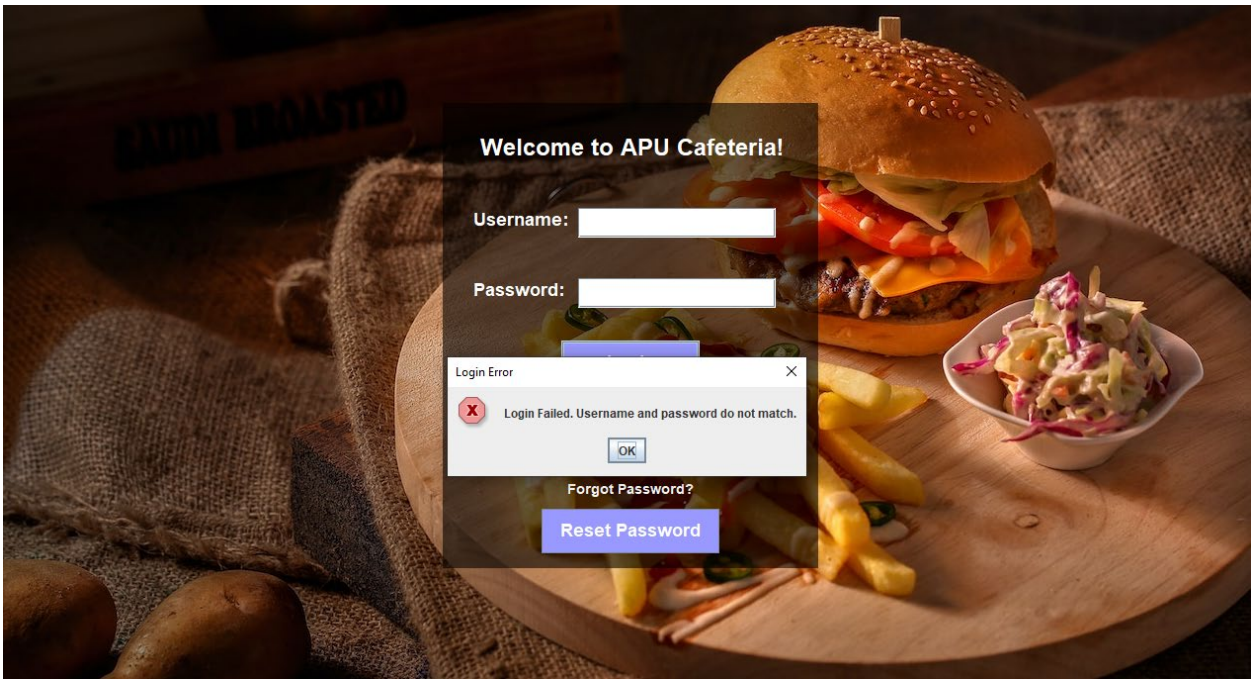
Display data code is almost the same as read data code. In the figure above, the program is reading the file line by line using a while loop and *readLine* method. This method reads the file until it reaches end of file, then it returns *null*. Then the line is split and assigned into different variables. The only difference from reading the file is that this section shows or displays the data. For example, the code here shows the data into a table. After it is done, it closes the file. The whole section of this code is wrapped inside a *try-catch* block with *IOException* handling.

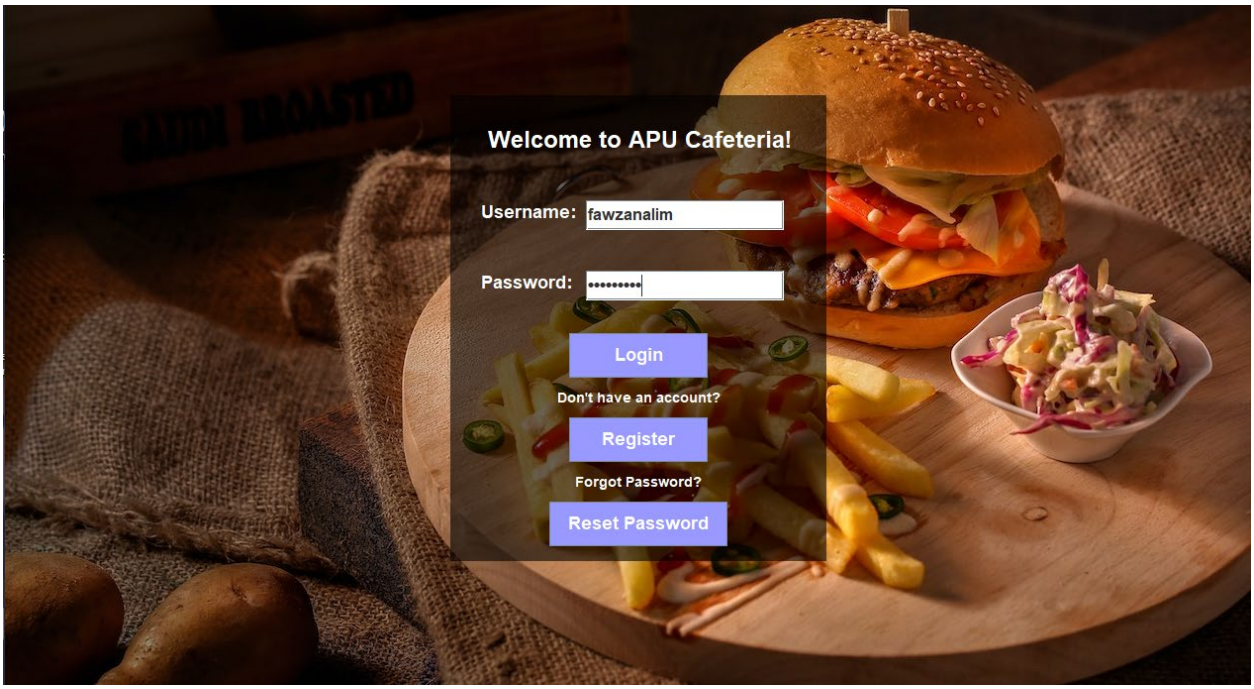
5.0 Sample Output Screen

The screen size for each window is 1280 pixels wide and 700 pixels tall.

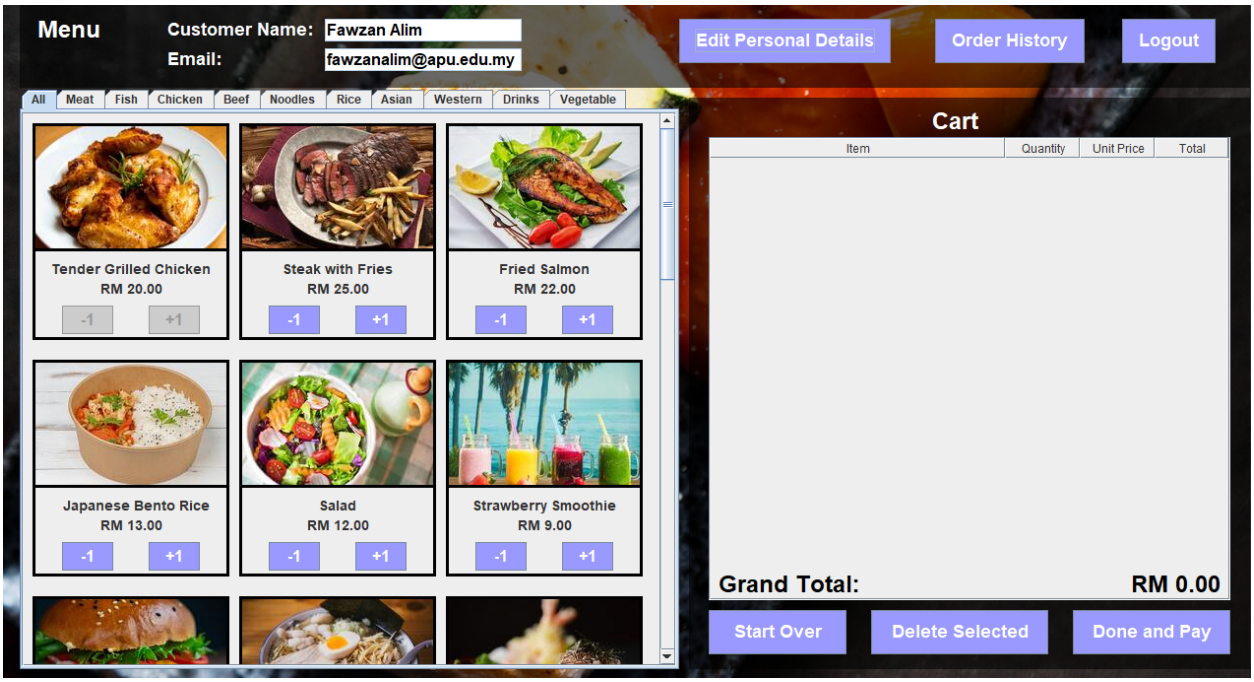


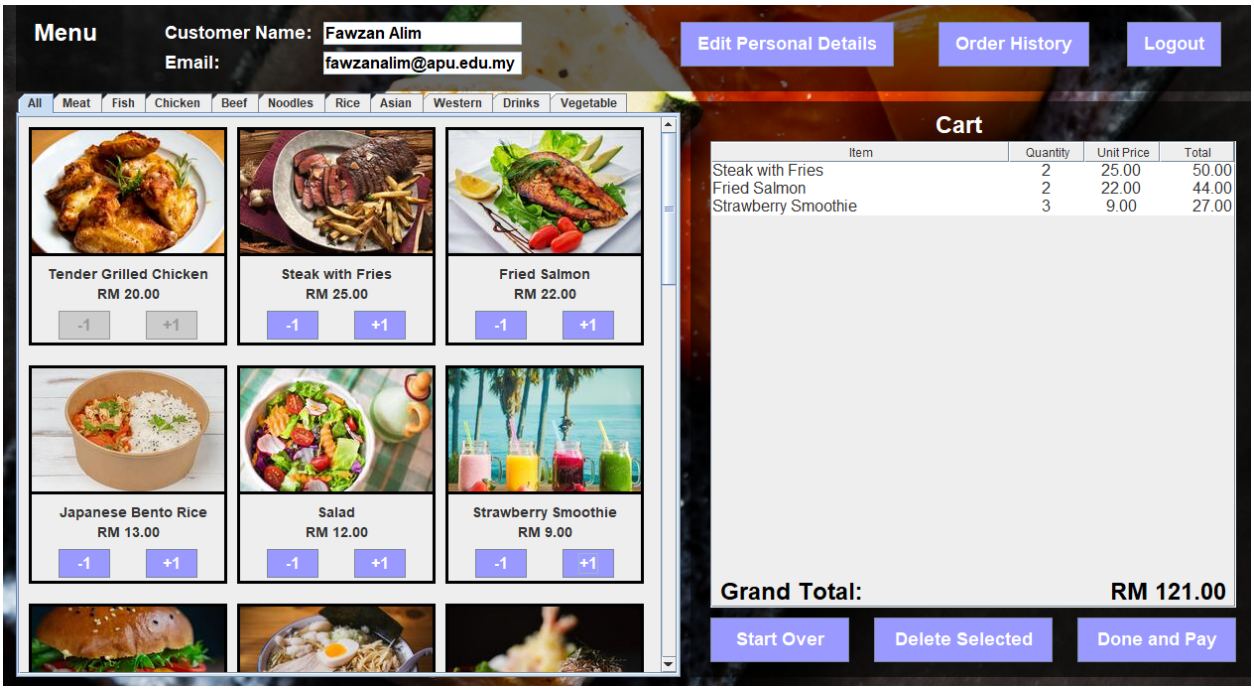






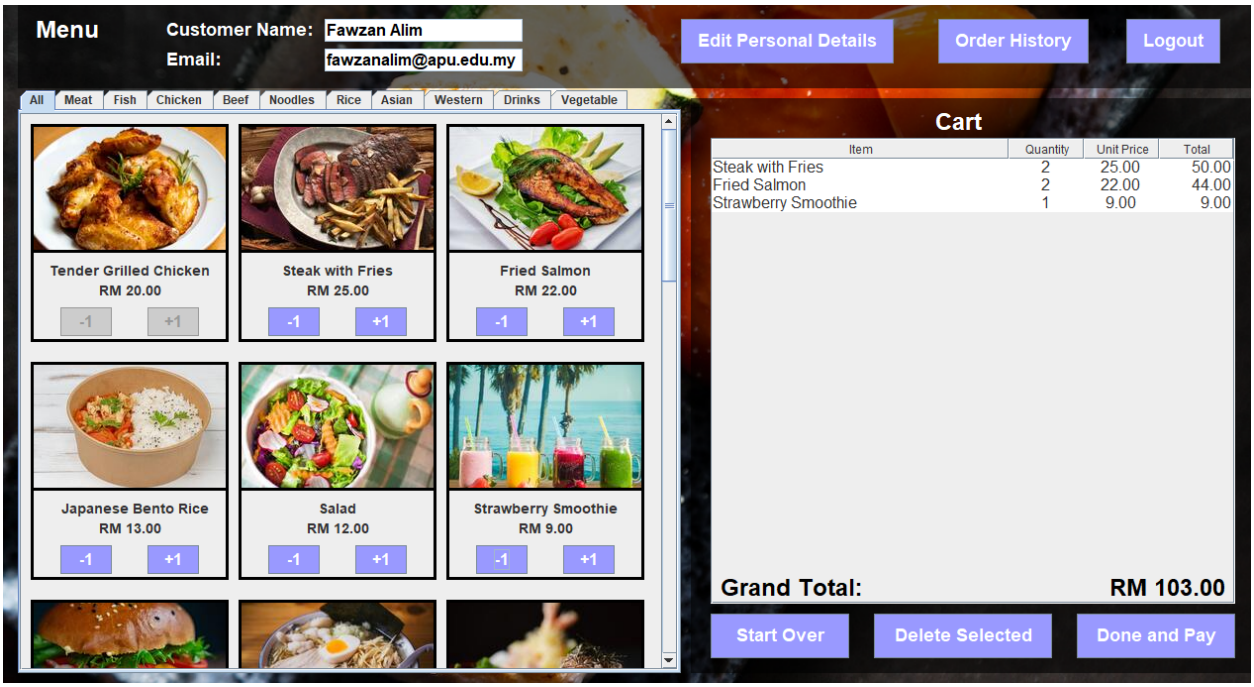
The images above show the login page, which is the first page that both managers and customers see when they first start the system. There are built in validations to prevent users from logging in without inputting a correct username and password. Leaving any of the text fields blank or inputting wrong username and password will lead to a pop up box giving the user an error message.



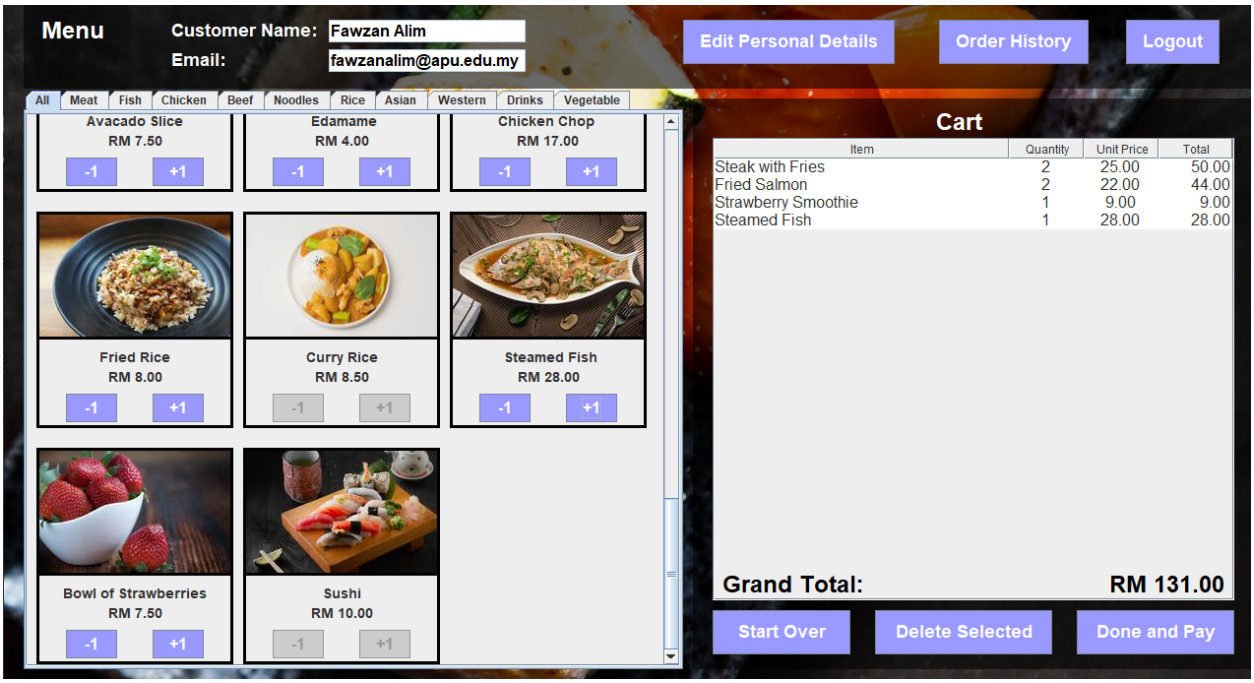


After successfully logging in, the customer will be redirected to the menu page. The images above are the menu in which the customer can order dishes from. Visual elements such as images are used to make it more visually pleasing and attractive to look at. The customer's detail is shown at the top of the page to make it more personalized, and it also ensure that the customer knows that they are using their own account and not someone else's account. The customer can click on the

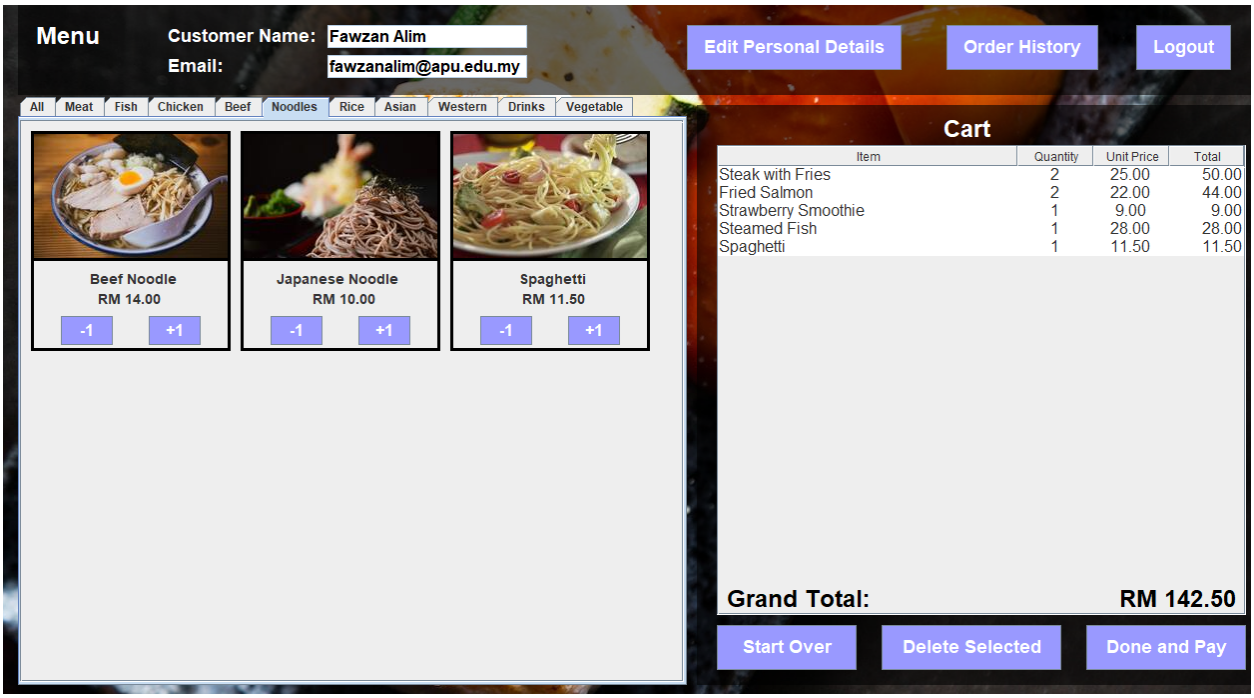
"+1" button that is below the image to add the dish to a cart that is at the right side of the screen. Crucial information about the dishes is shown to the customer in the cart such as the item name, the quantity of the item that is in the cart, the price of the item per unit and the total cost as well as the grand total that the customer has to pay. The grand total is in bold to that customers are aware of the amount of money that they need to pay.



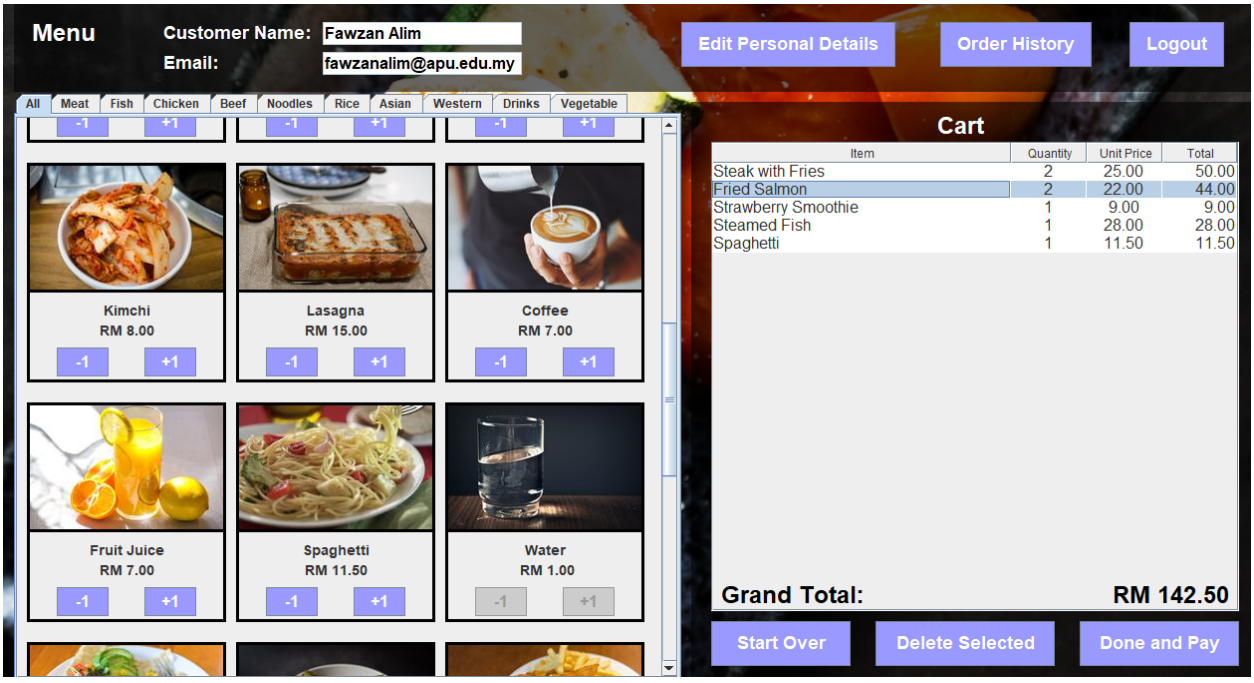
The customer can subtract the dish from the cart by pressing on the "-1" button. This would reduce the total cost of that the customer need to pay.



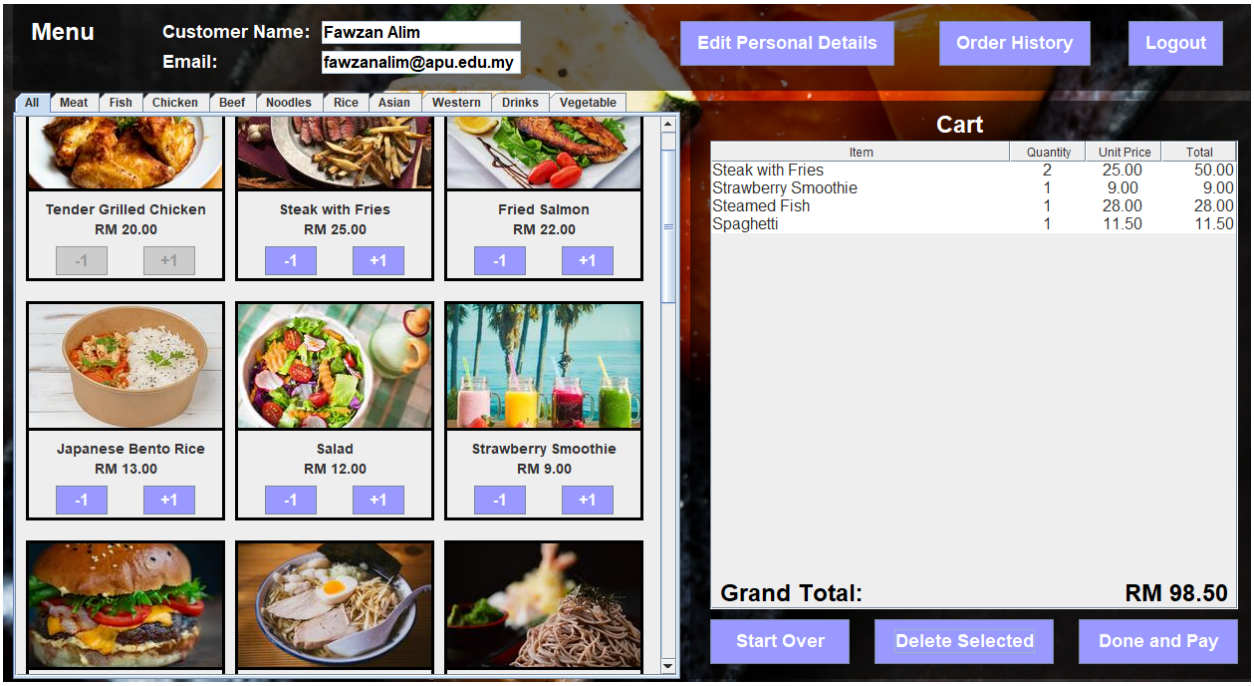
There are a total of 23 unique dishes with stunning images, therefore the customer can use the scroll bar to view the rest of the dishes below. Some of the button appears grey, this is a feature that the manager can use which is to make dishes unavailable or available. If the dishes are available, the customer can add it to the cart, however if the dishes are unavailable then the customer cannot add it to the cart.



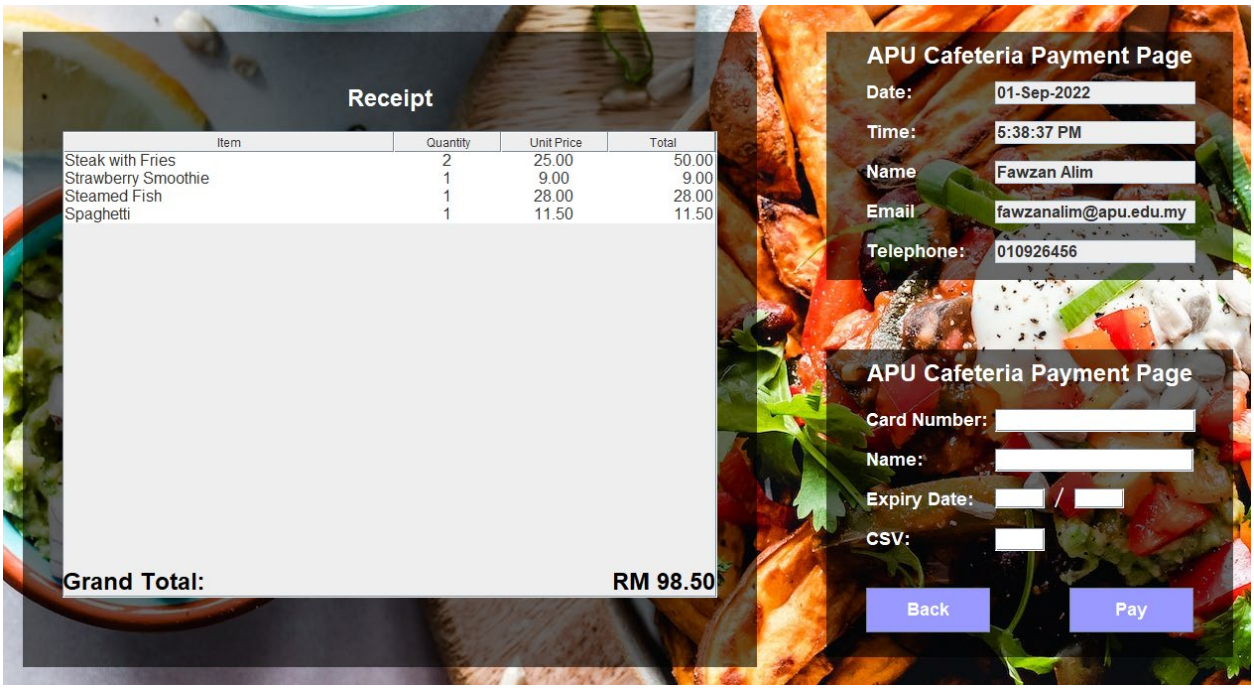
There are several tabs to the menu which divides the dishes into categories. For example, if the customer wants to eat noodles, the customer can quickly filter out all the non-noodles by clicking on the noodles tab on the top of the menu.



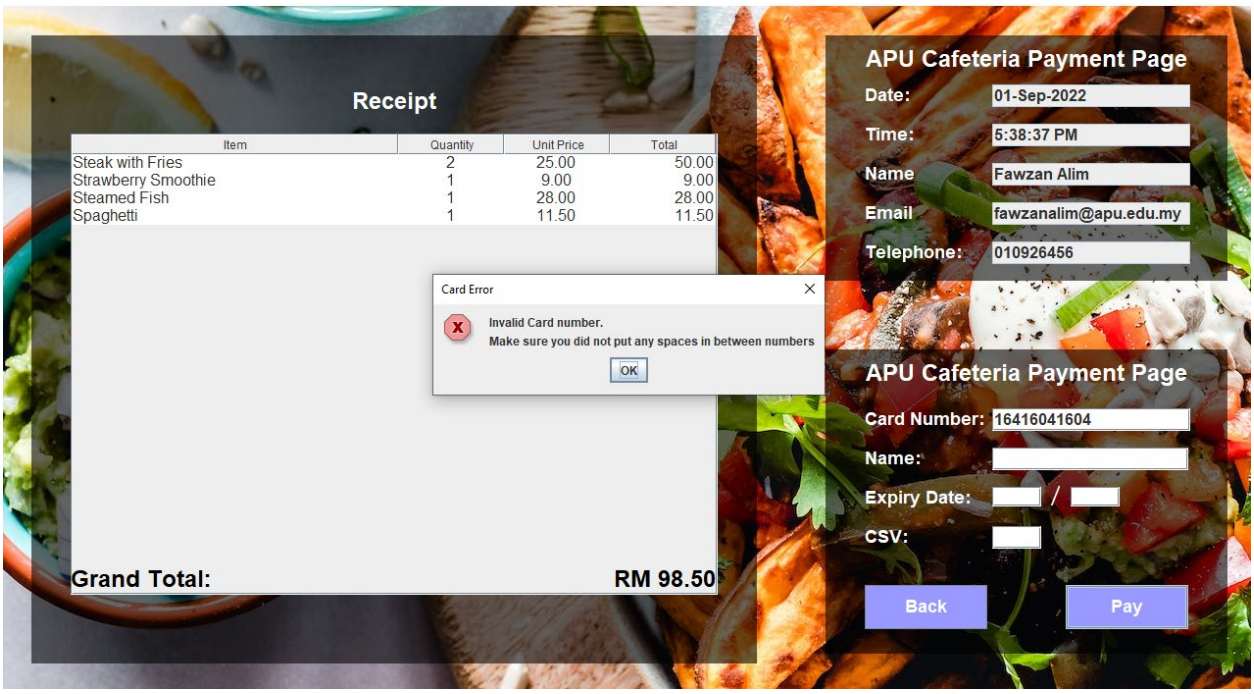
The customer can select dishes from the cart and perform unique actions such as to delete the item from the cart. If the customer presses the “Start Over” button, the cart is emptied.



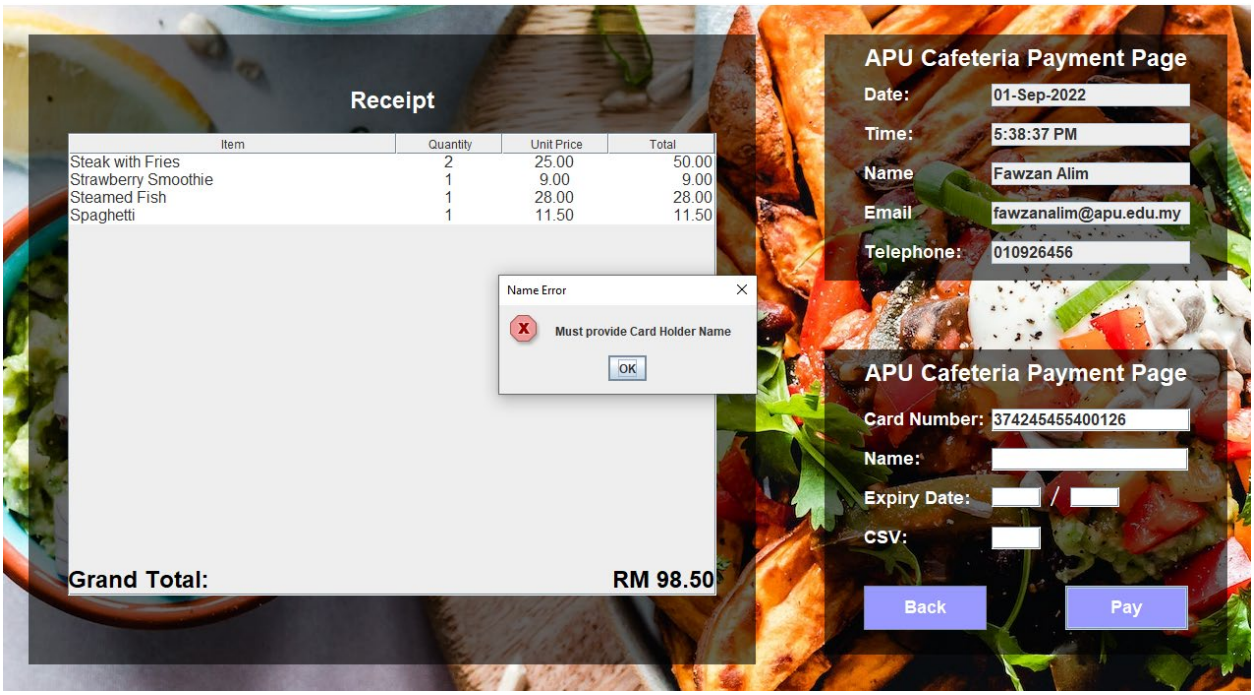
This is an image after the fried salmon is removed using the “Delete Selected” button.



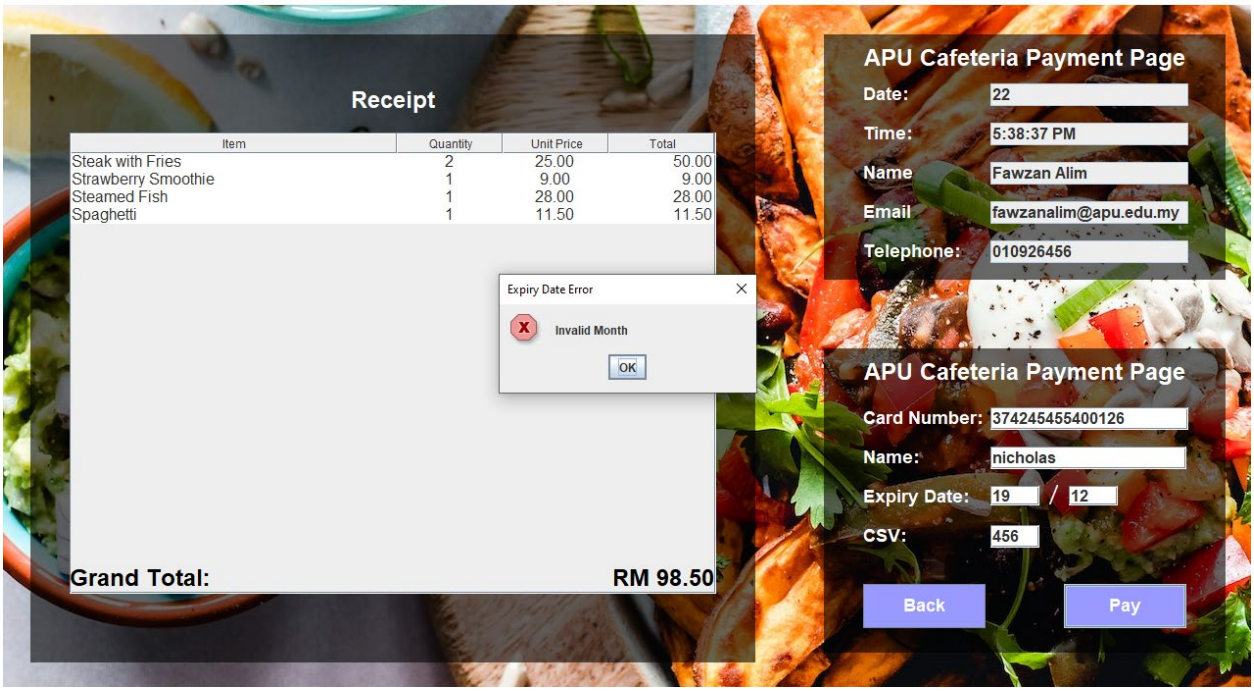
The customer can proceed to pay after they are happy with what they have in the cart. They will be redirected to the payment page which can be seen in the image shown above. Here, a receipt is shown to the customer, highlighting the important details to them, such as the names of the items that they have in the cart, the quantity of the item in the cart, the price per unit, the total cost for the item and the grand total for all the items. This will limit the confusion that the customer may experience. The grand total is bolded to direct the customer attention to it. The details of the customer are shown in the top right, such as the name and contact details of the customer as well as the time and date that the customer decides to pay. The bottom right side is where the customer will enter their credit card details and pay for their meal. There are built in validations to prevent customers from entering invalid credit card numbers.



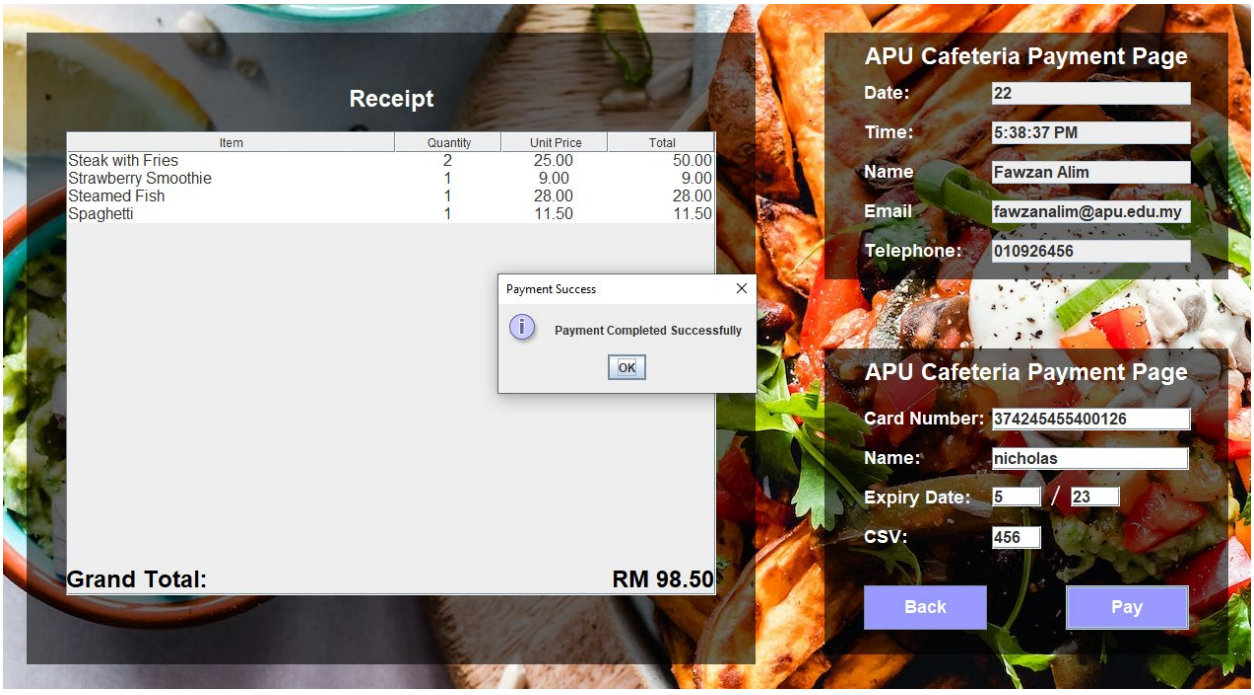
Fake credit card numbers are not accepted and the system will display an error.



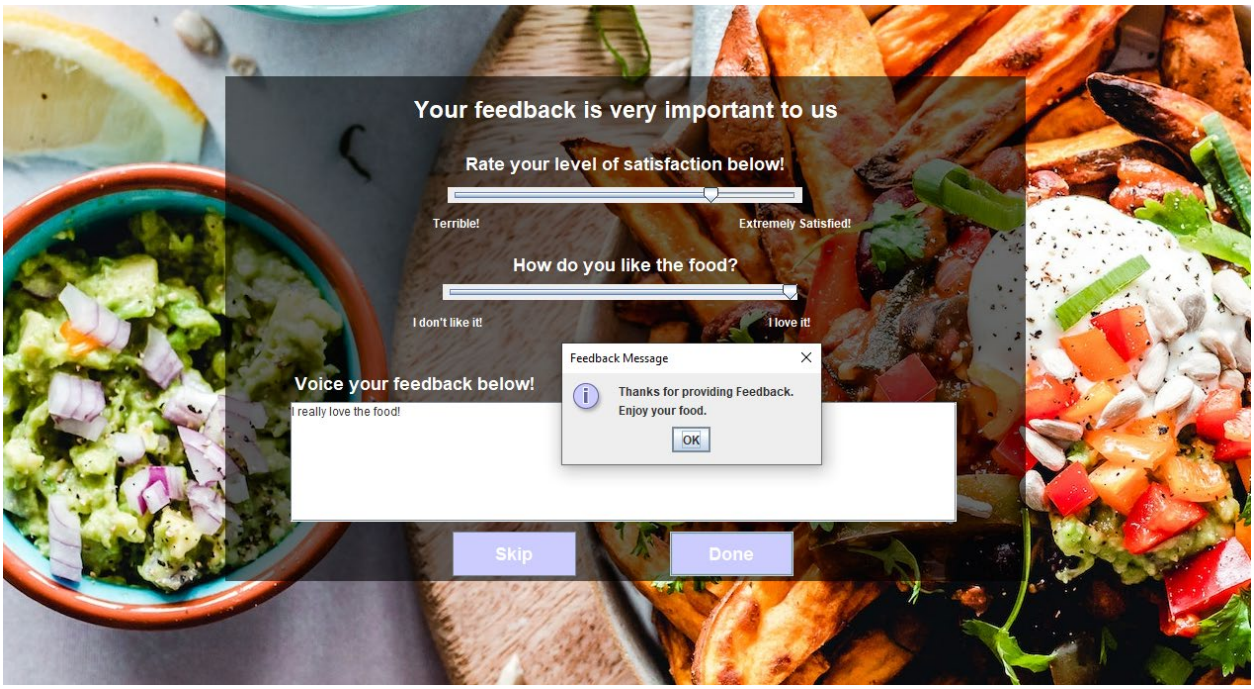
None of the text fields should be empty. If any of the text fields are empty, then the system will display an error message.



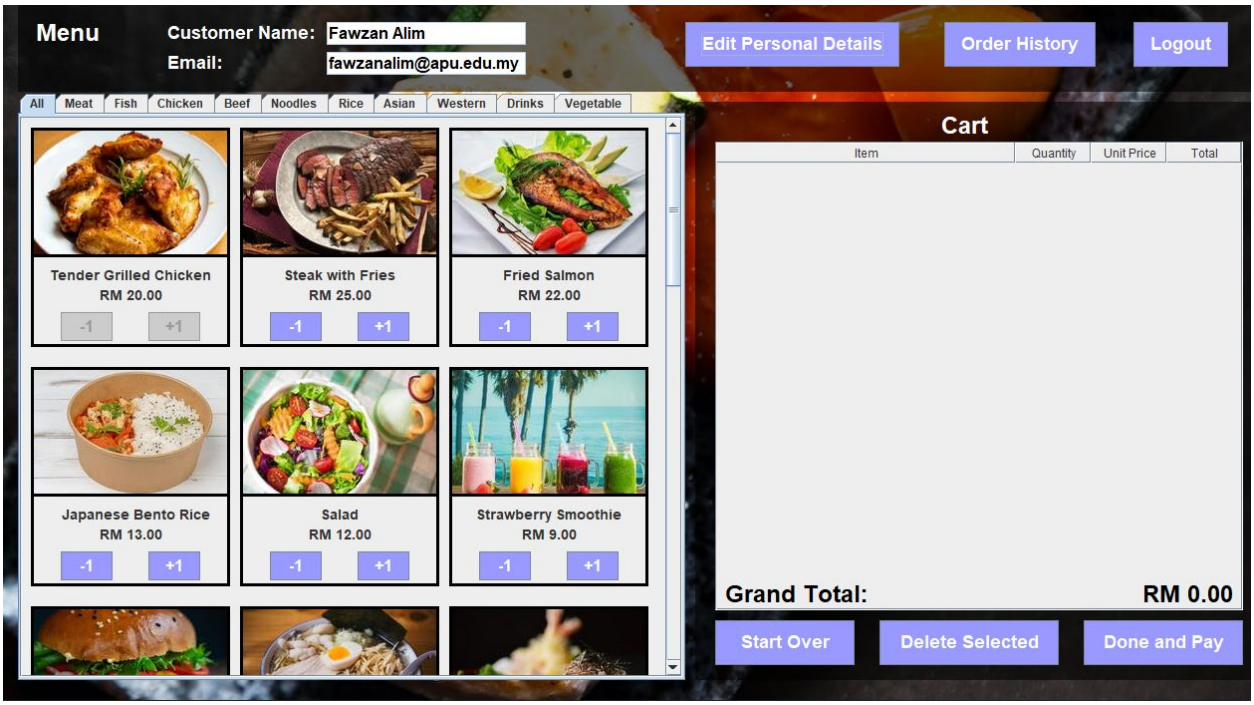
There is also a built-in validation that prevents users from entering a date that has already passed.



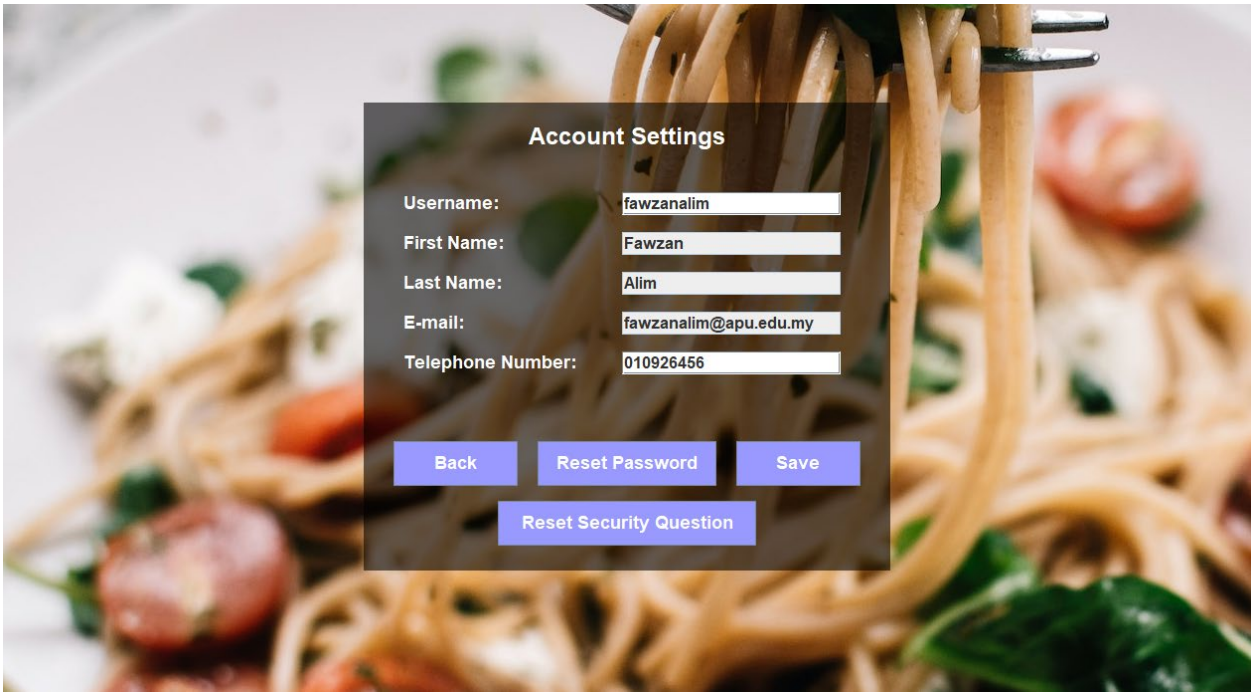
If the user successfully enters all the information correctly, the payment is completed, and the user will be redirected to the feedback form.



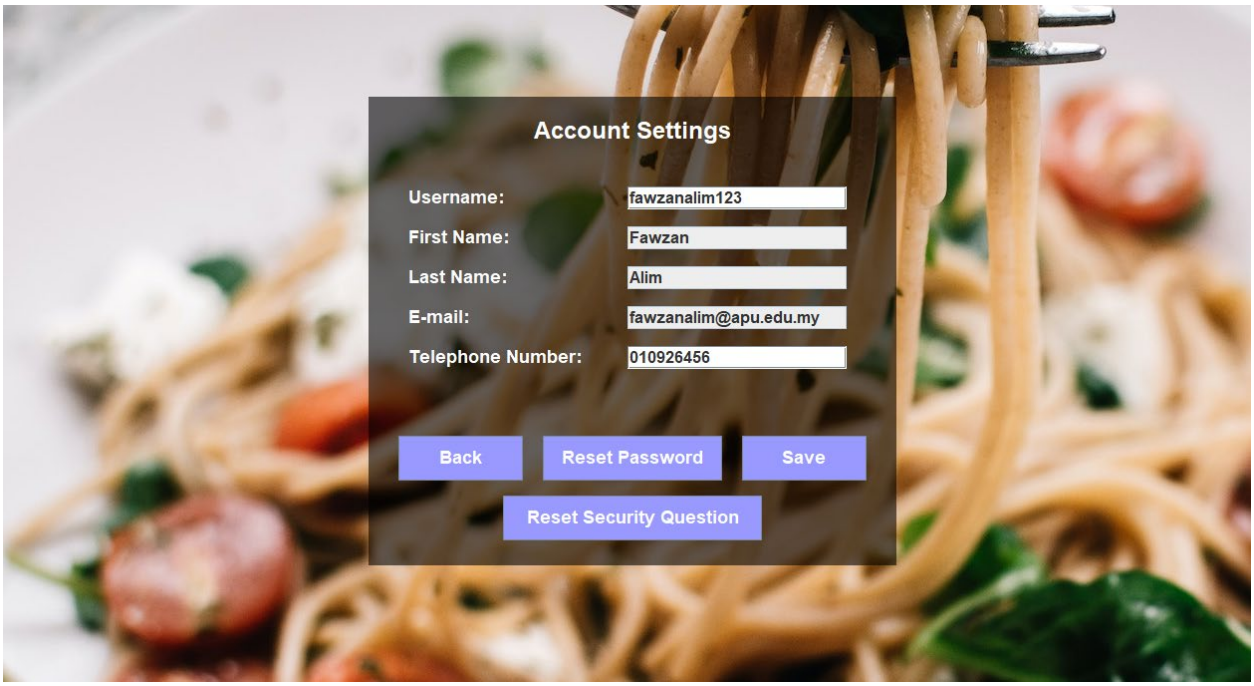
This is the feedback form that the customer will see. The customer can rate their overall satisfaction at the cafeteria, the food taste and give detailed written feedback.



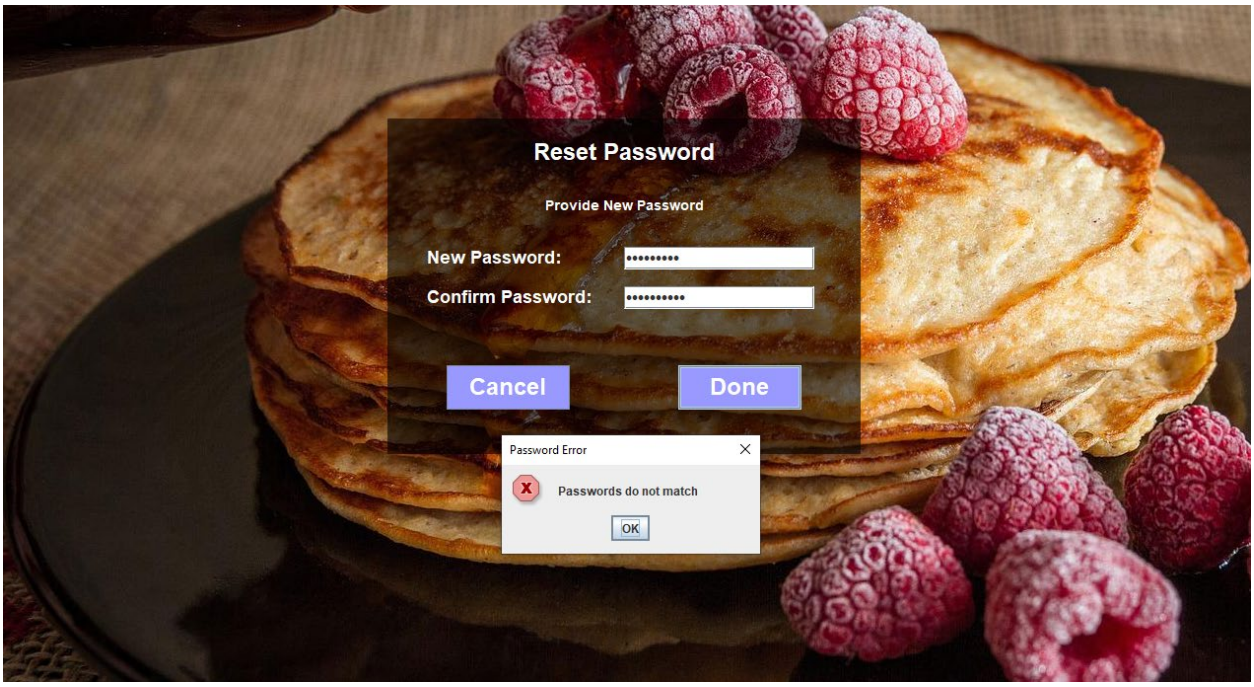
Returning back to the menu, the customers can also edit their personal details by clicking on the button located on the top right.



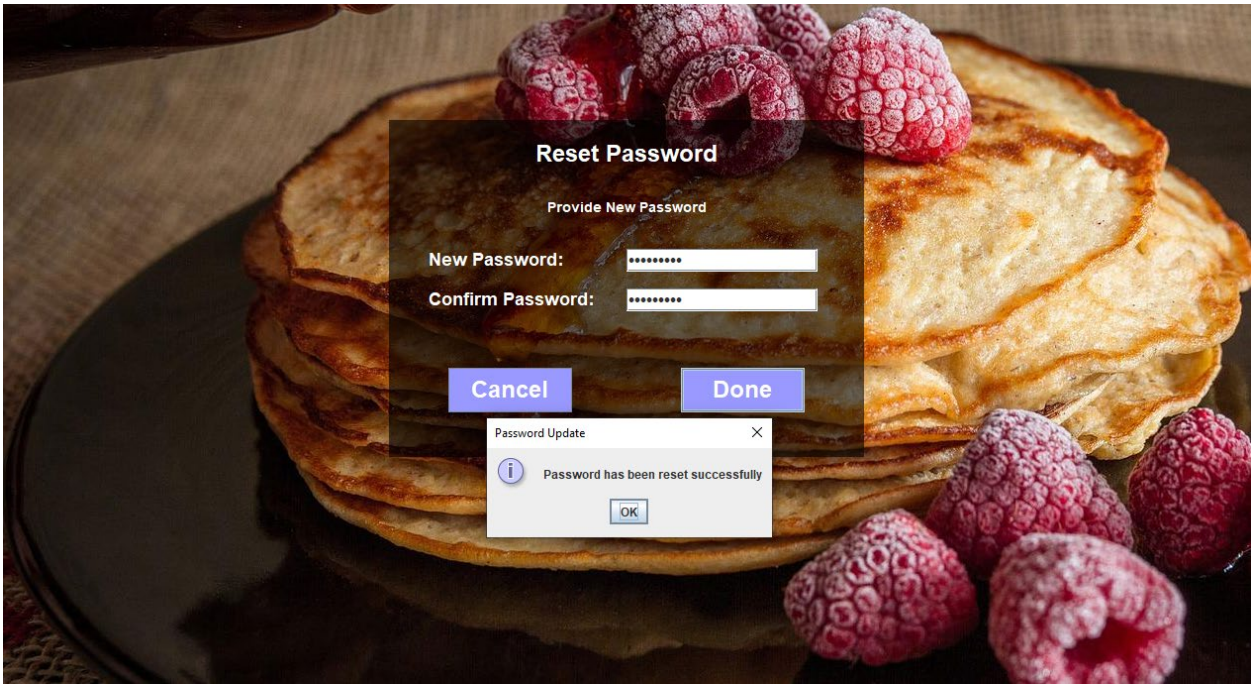
They can edit their username and telephone number.



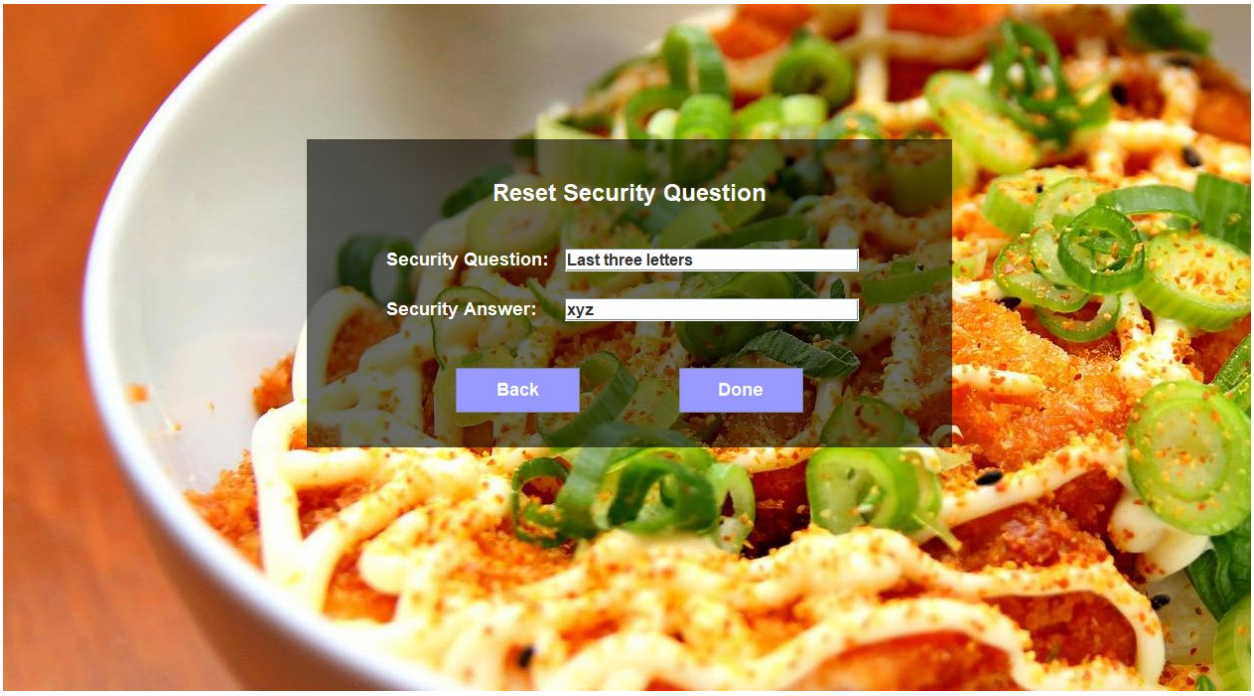
This is an example on how the customer can edit their username. Their name and email are not editable because names do not change and the email is an APU email, therefore it doesn't need to be changed. The customer can also change their password by clicking on the "Reset Password" button.



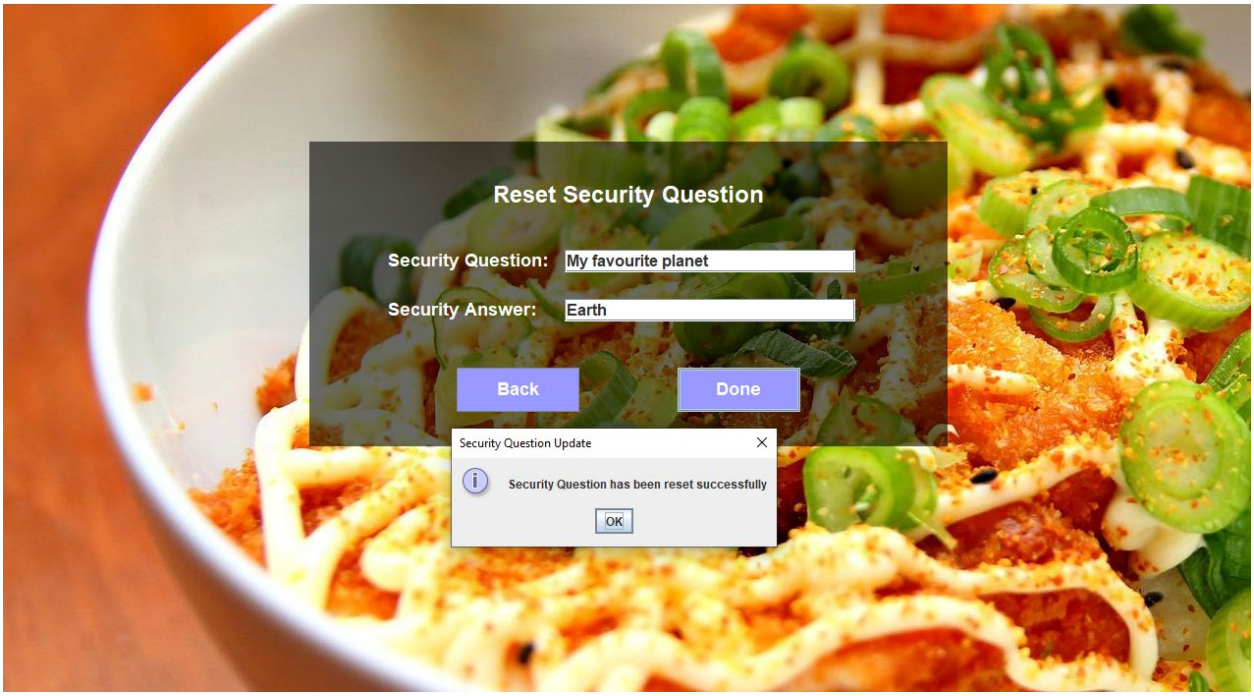
This is the page where the customer can reset their password. They are required to type in the new password, and to confirm the password. If the two are different, the system will display an error message.



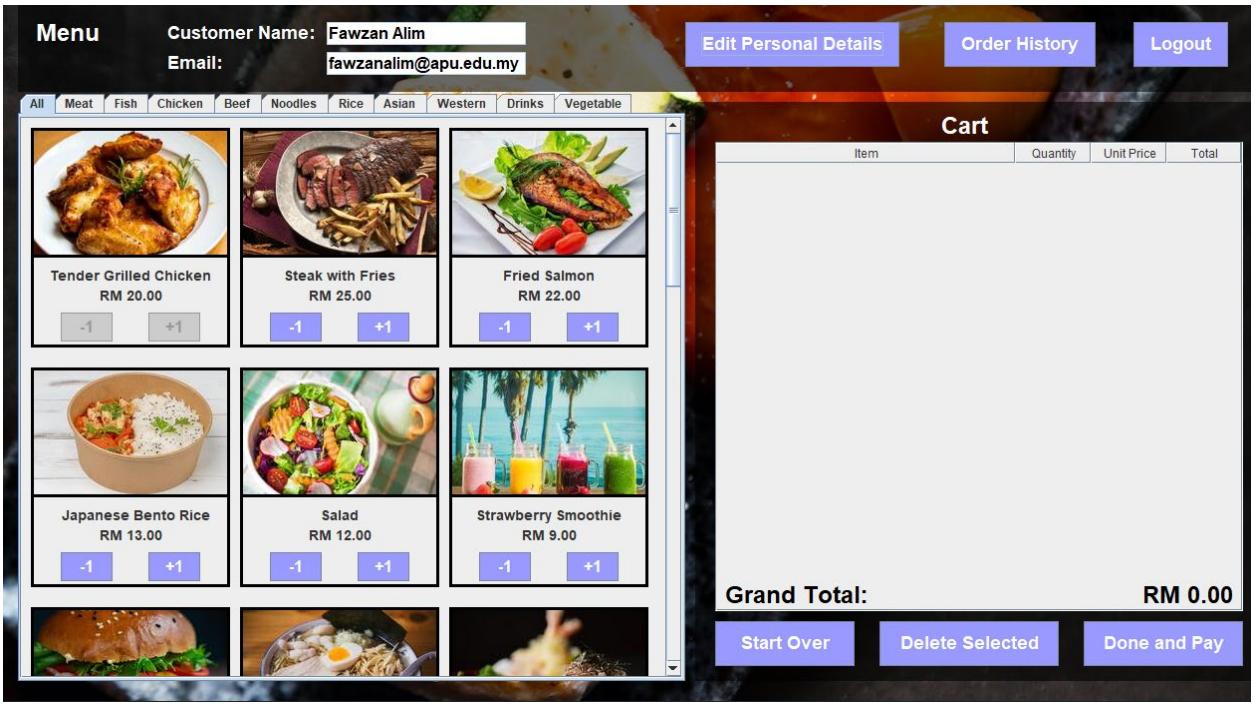
If the text in the password field and the confirm password field are same then the password will successfully be changed.



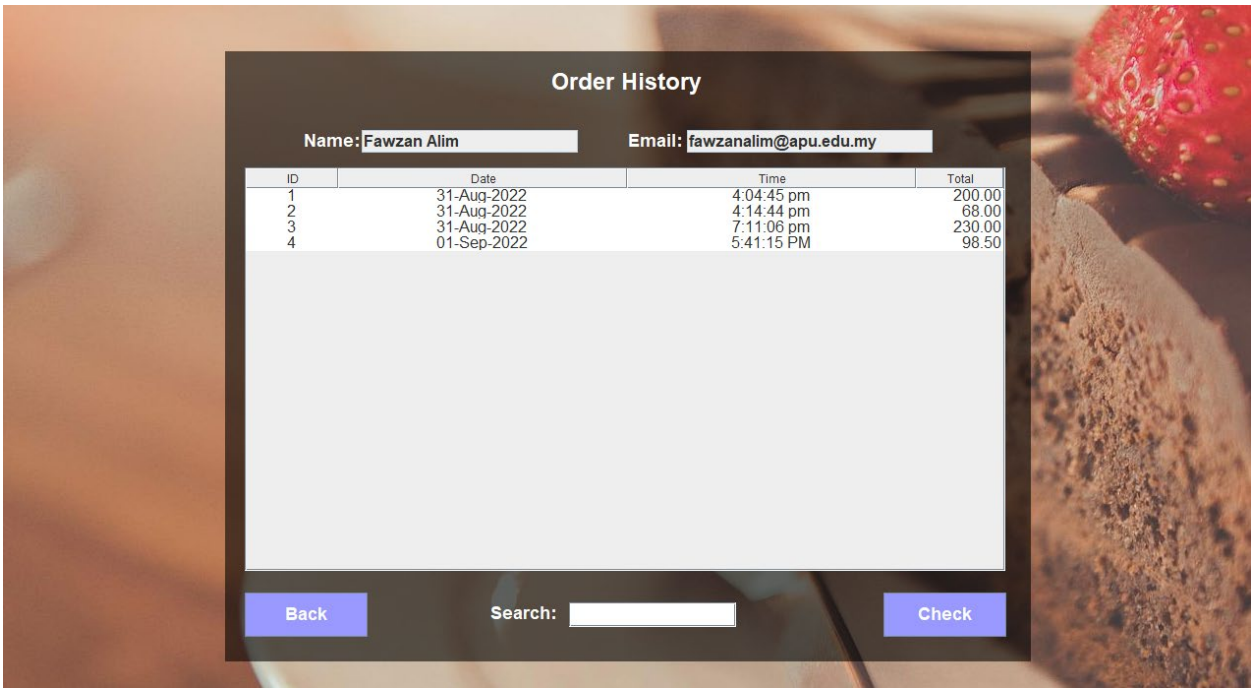
In the account settings, the customer also has the option to reset their security question. The security question is used to reset the password if the customer forgets their password.



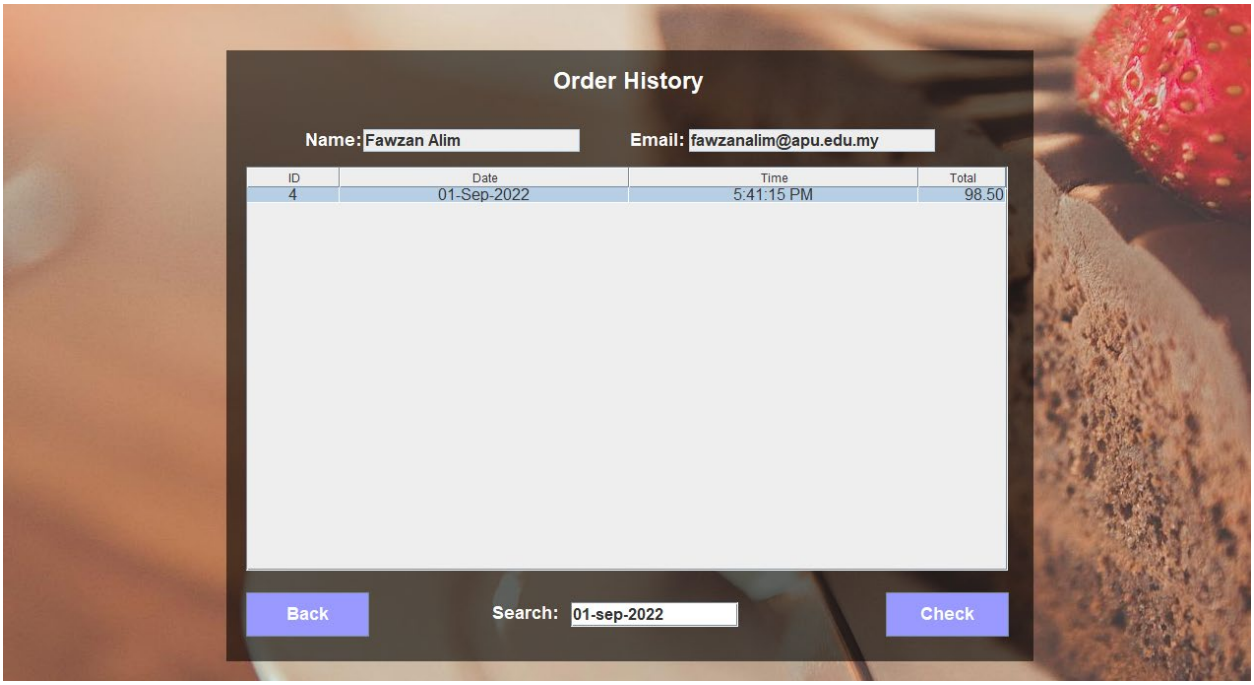
The image above shows the process of changing the security question and answer.



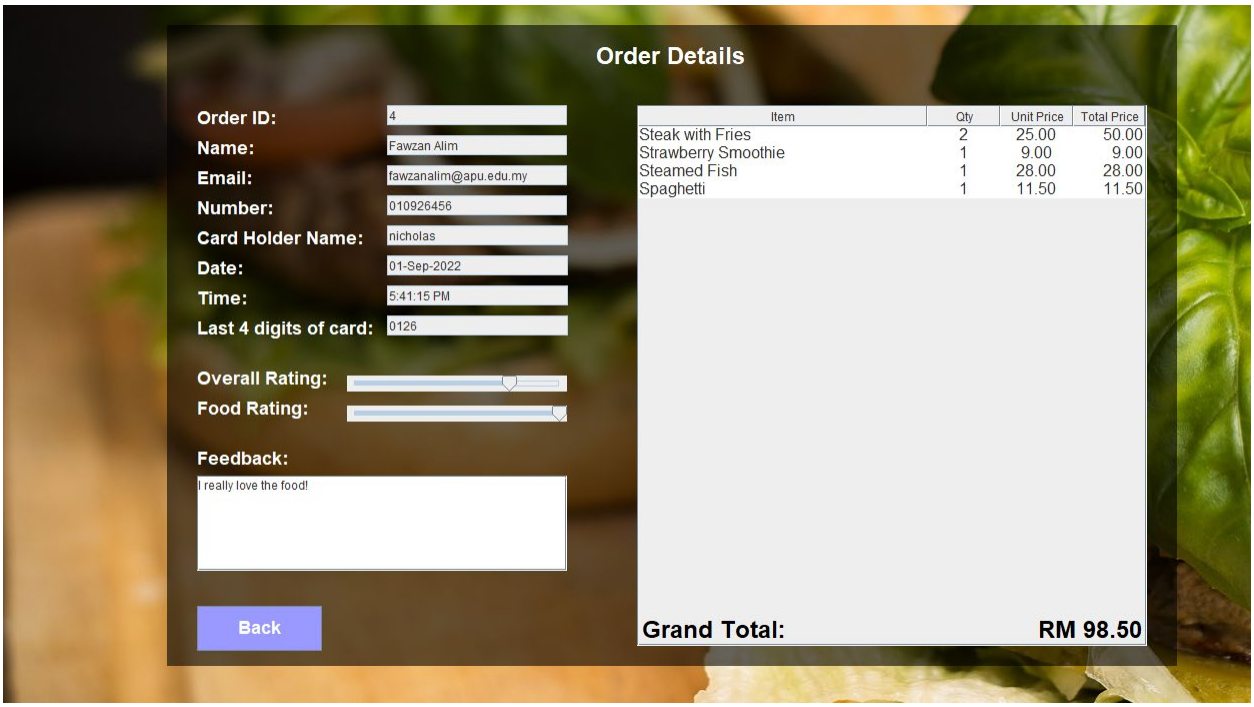
Returning back to the menu, the customer also has the option to view their order history or to logout by pressing the buttons that are located on the top right.



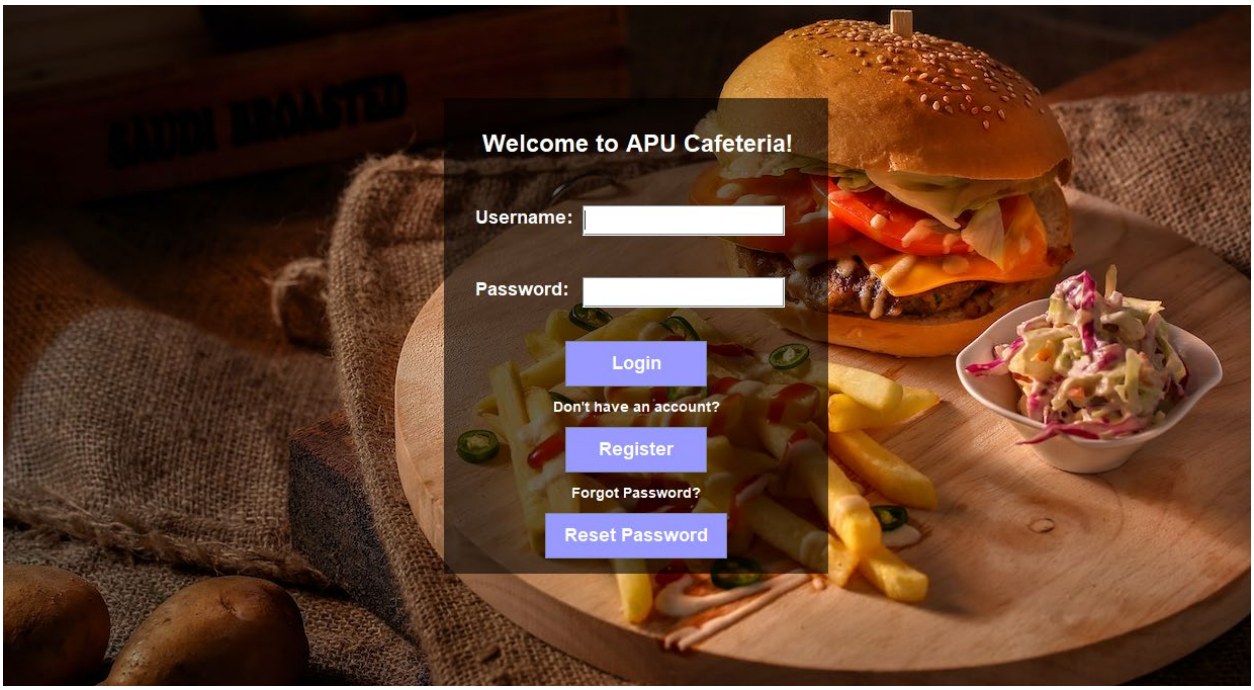
This is the page which shows the order history of an individual. The customer can only see the previous order that the customer made and not anyone else. It shows crucial information such as the date, the time and the total amount the customer spent on the dishes.



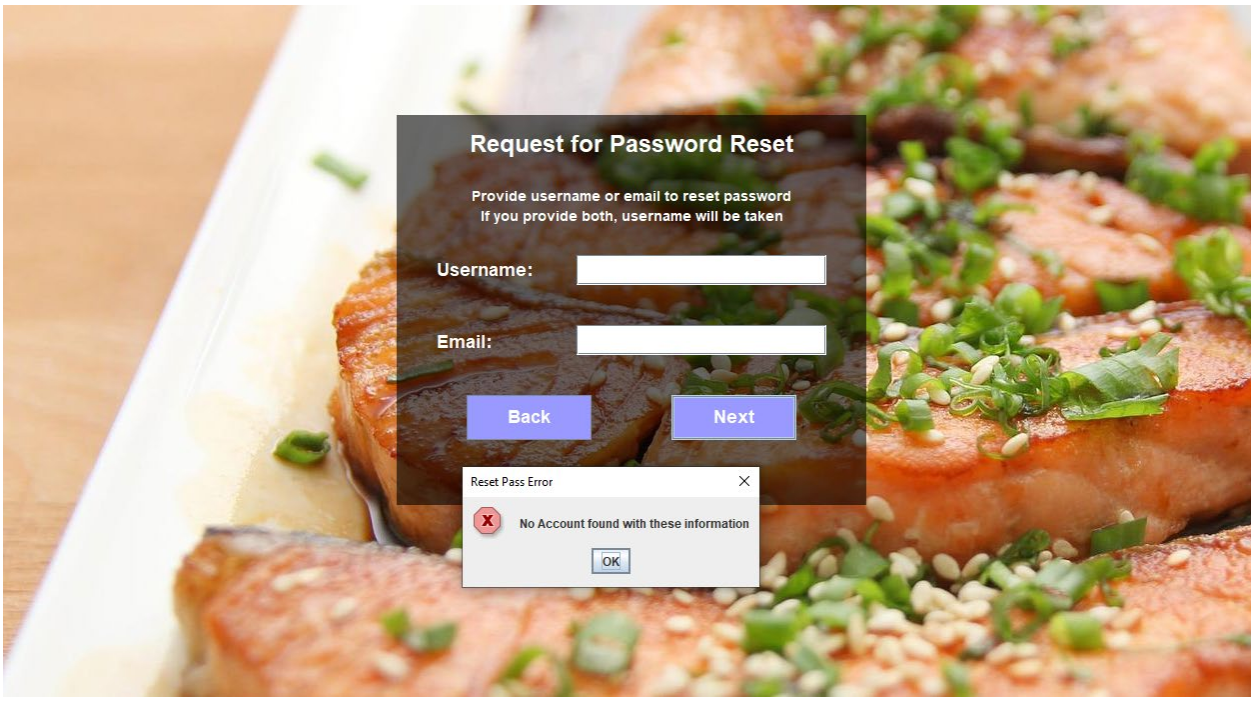
There is a search function at the bottom, it allows users to search for a specific thing, such as date. In the example above, the customer searched for all the orders made on 1st September 2022 and a result is returned.



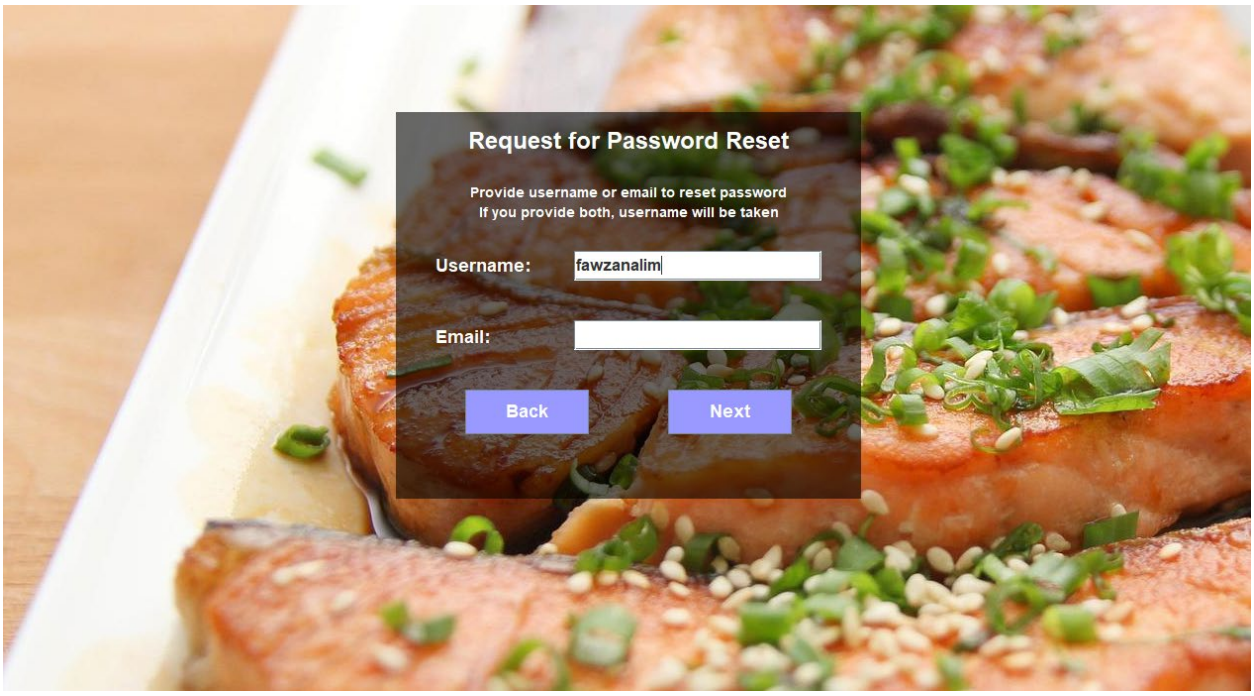
The customer can learn more about it by pressing the check button. The customer can view the customer's detail, the feedback given and the receipt of the meals that they ordered.



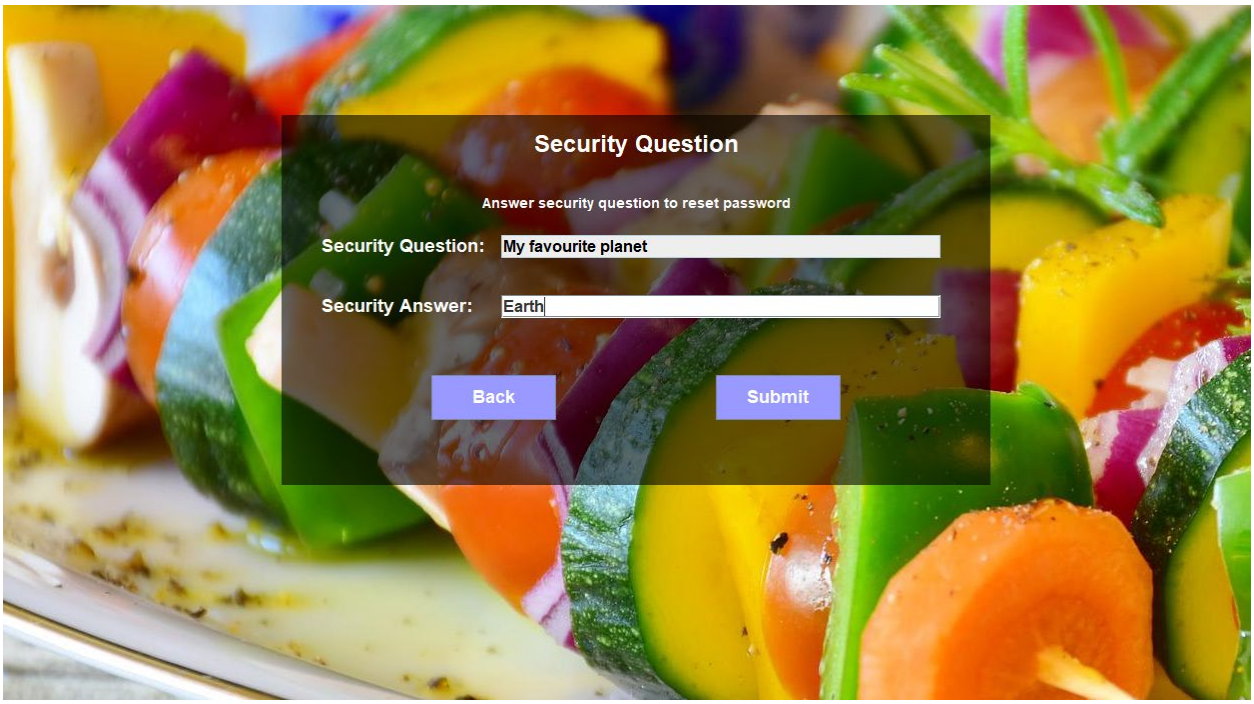
Returning back to the login page, there is a button where the users can reset their password if they forget their own password.



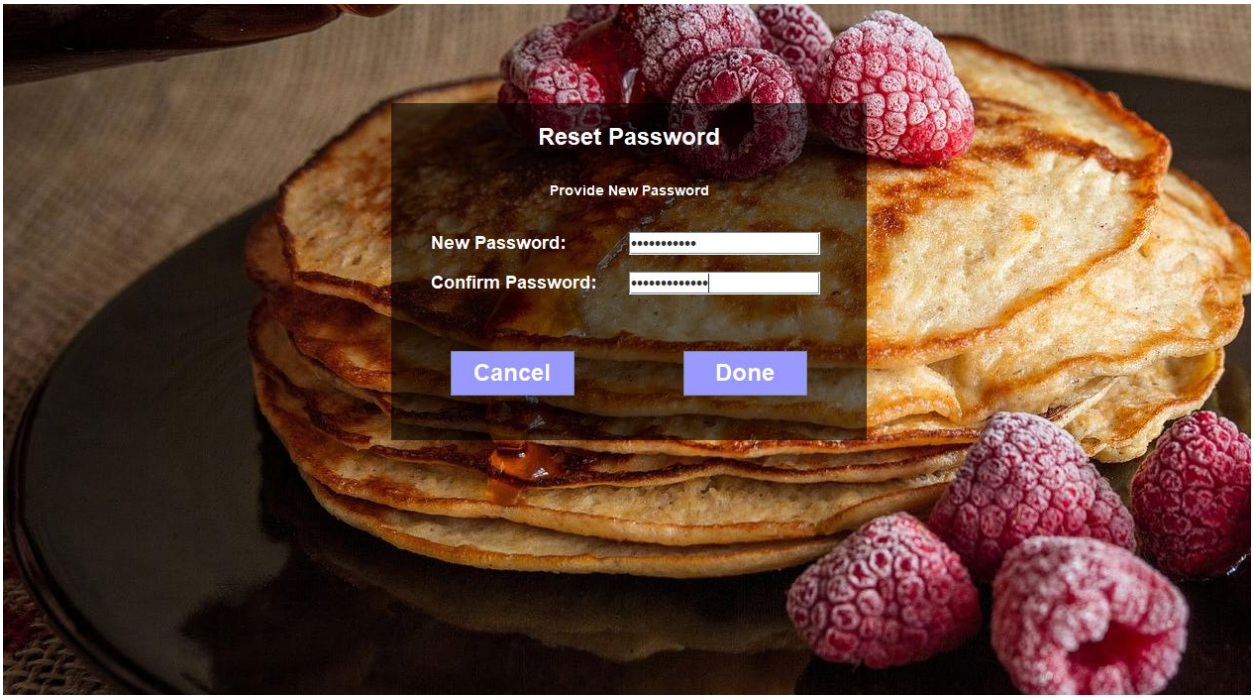
The user can request for their password reset by entering either their username or their email. Leaving the fields blank will result in the system sending an error message.

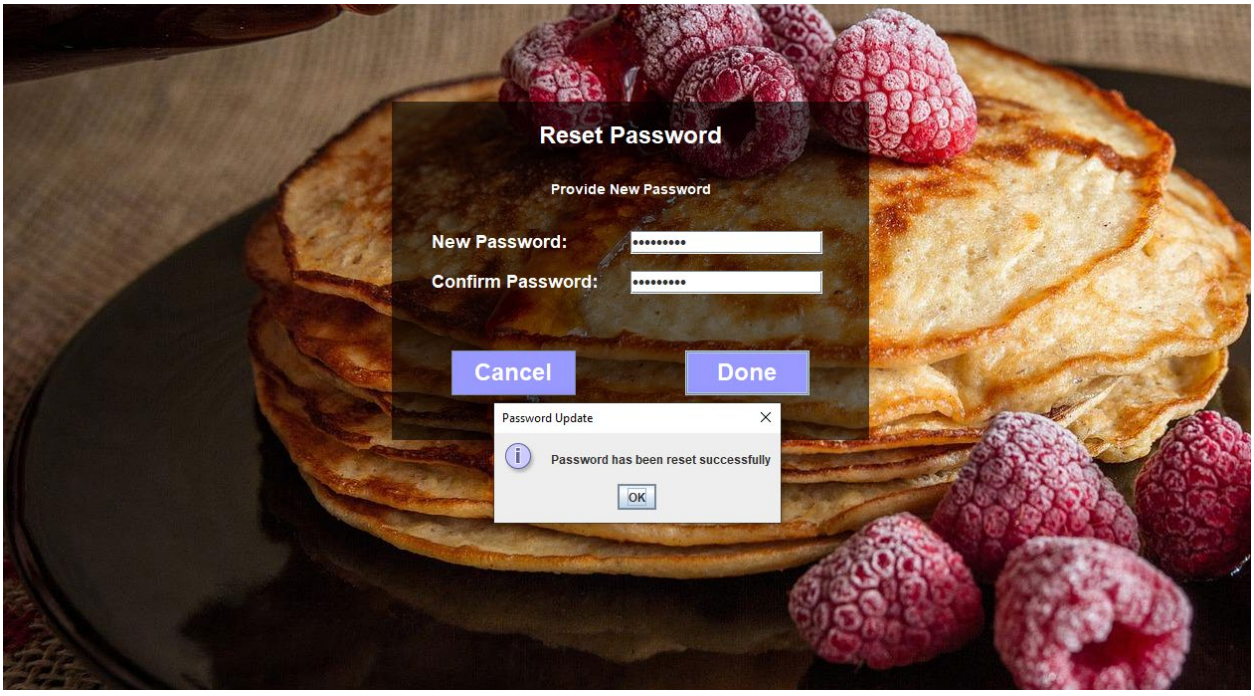


By typing either the username or the email of the account of the user that forgotten the password. The user will receive a security question, which is a question that they've set in case they needed to reset their password.

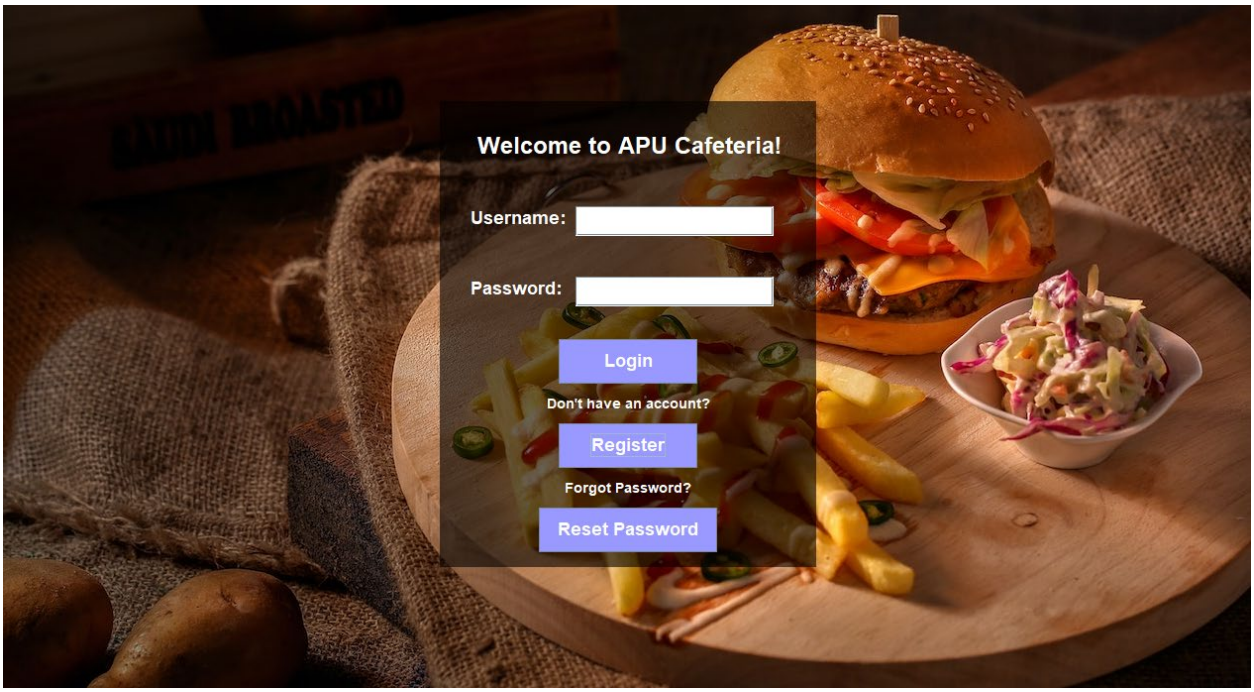


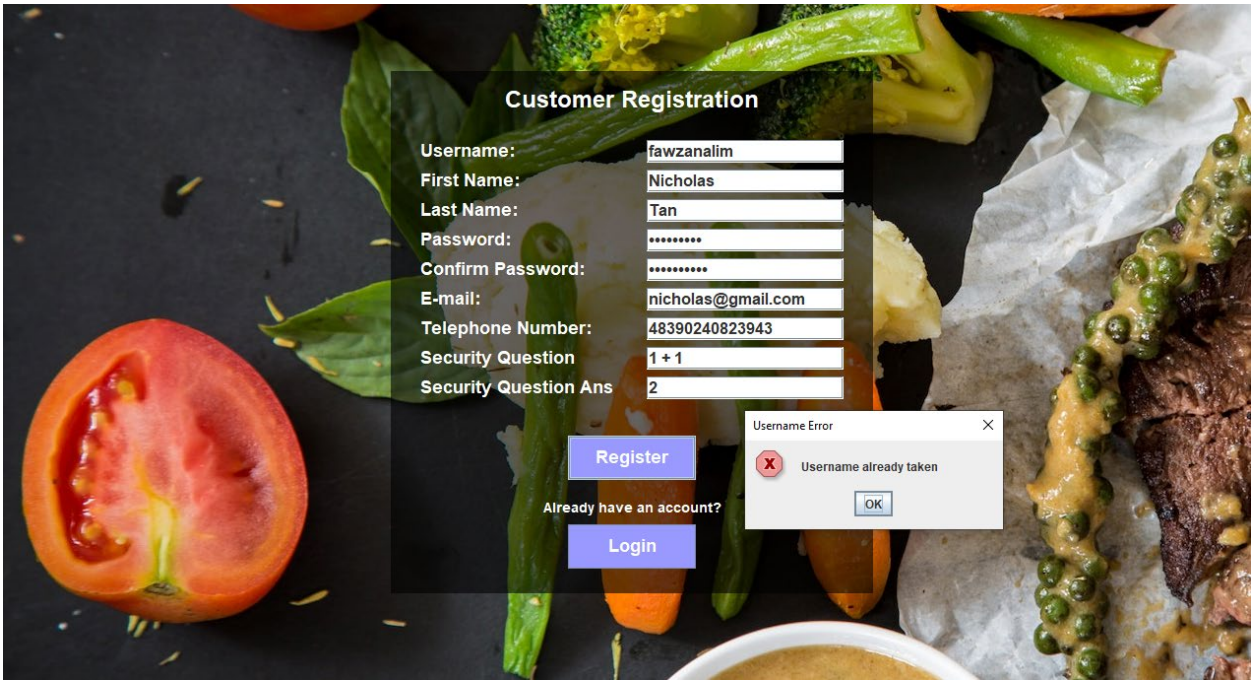
If the user enters the correct answer to the security question, the user may then proceed to reset their password.



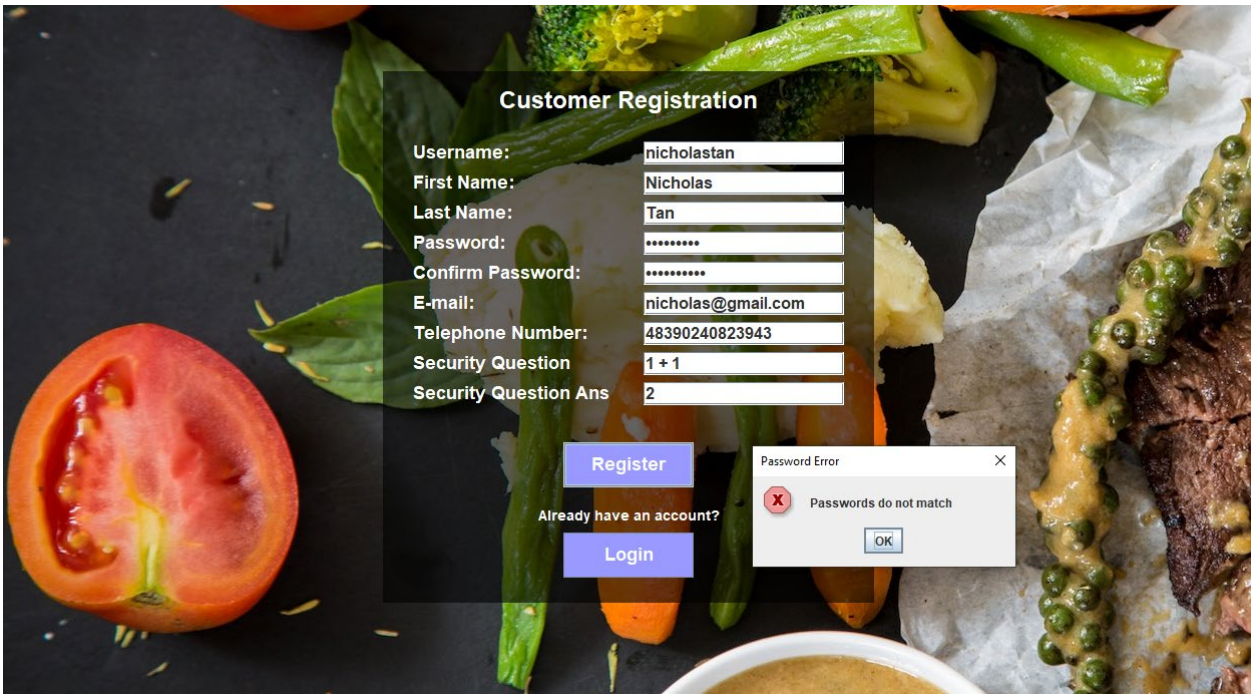


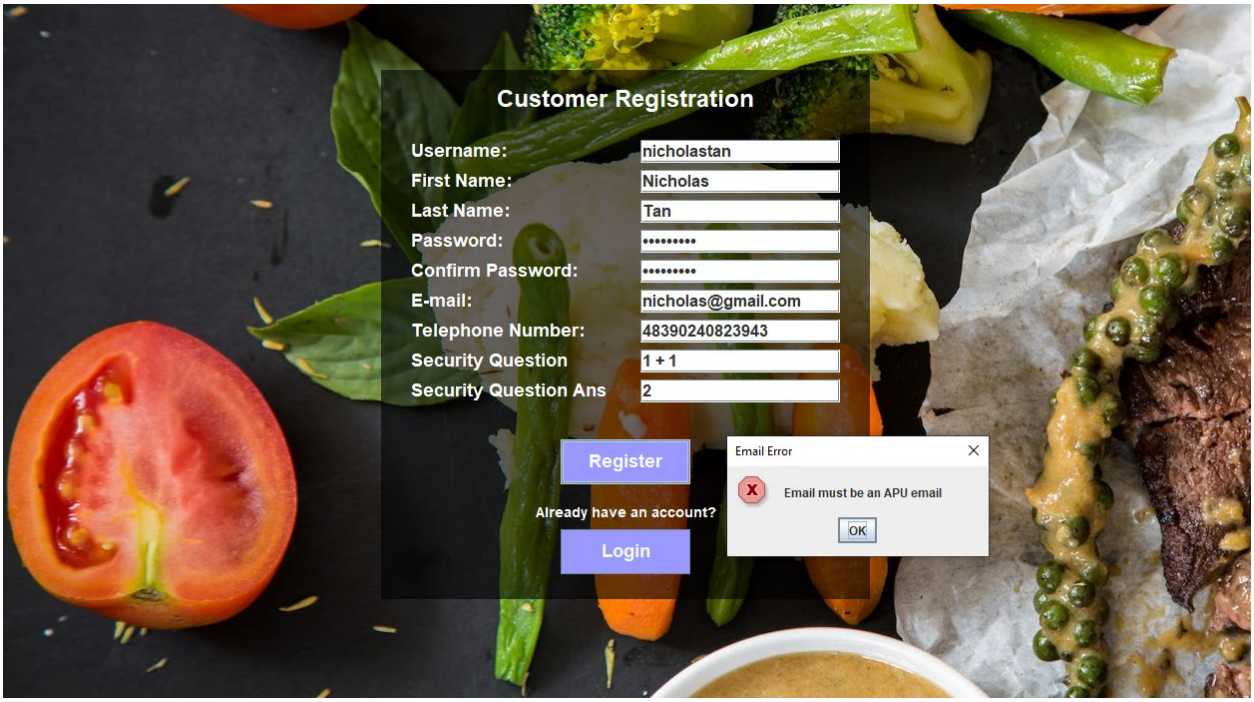
The user successfully changes their password when the new password field and the confirm password field match each other.



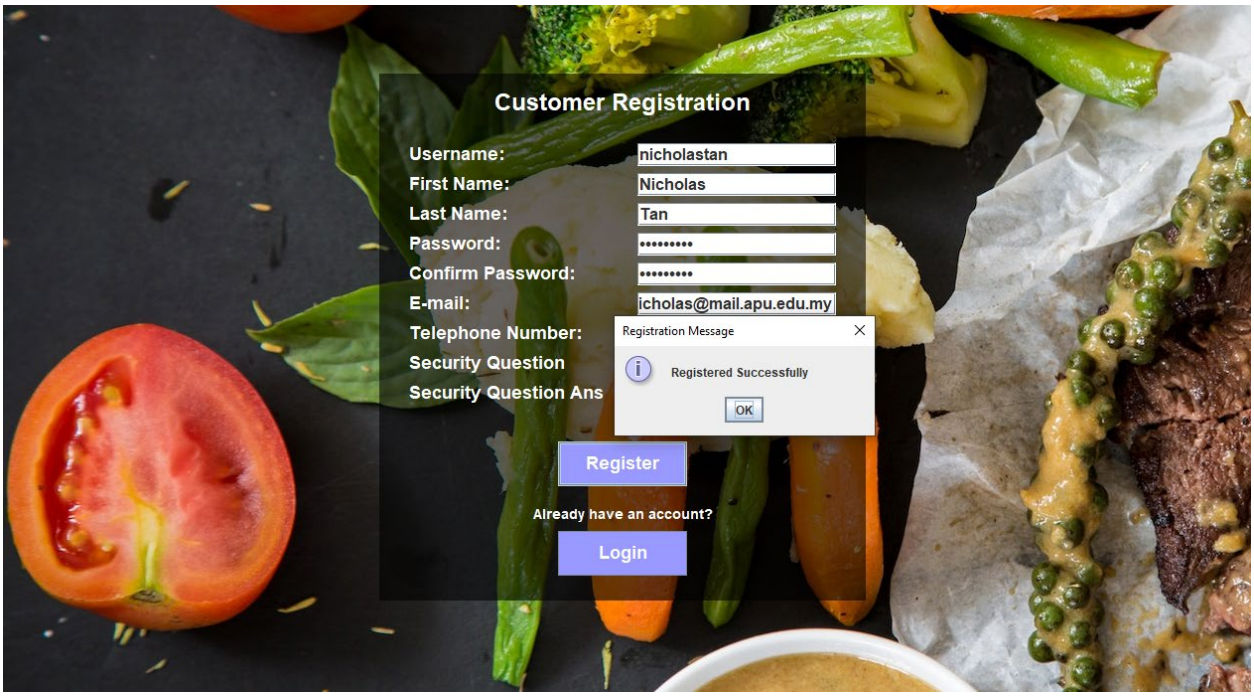


In addition to that, new customers can register to create an account from the login page.

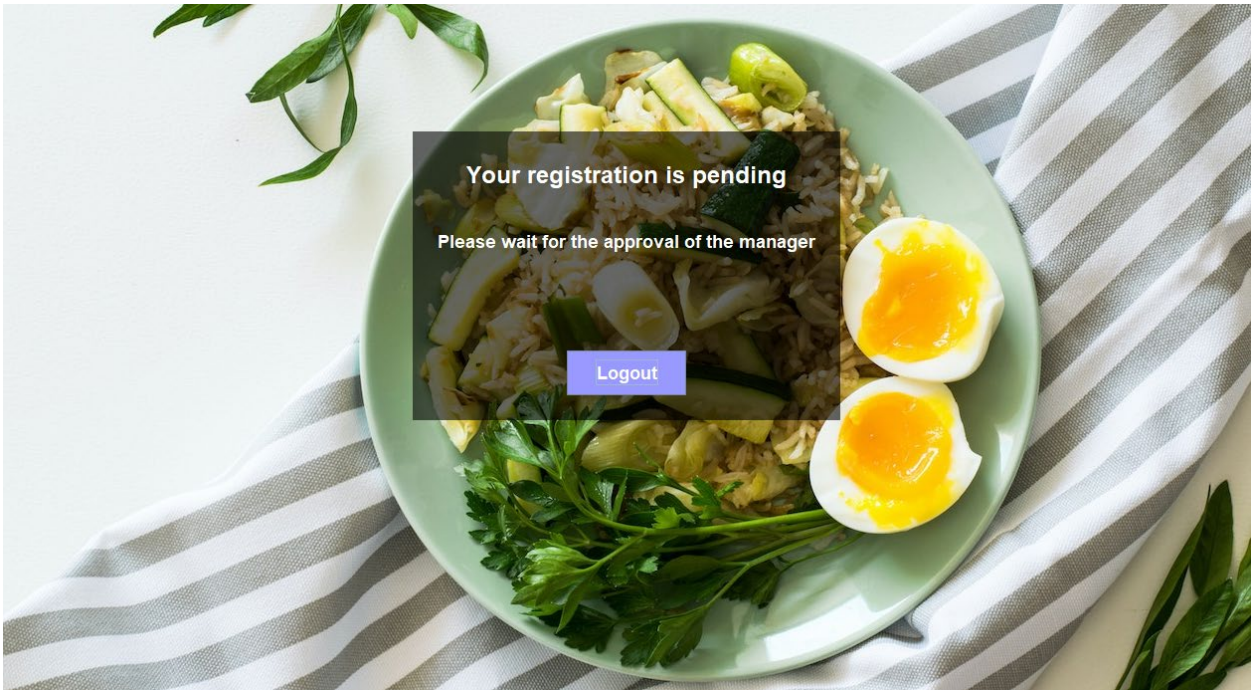




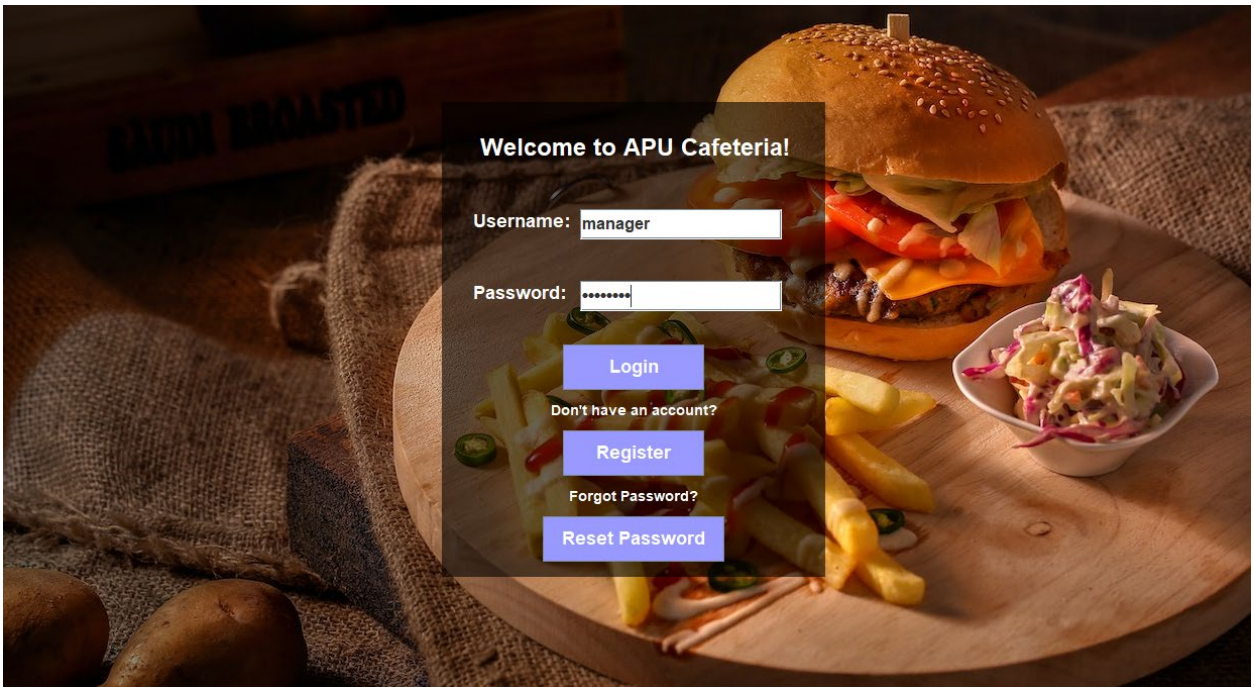
There are built in validations to prevent users from entering arbitrary values to the text fields.



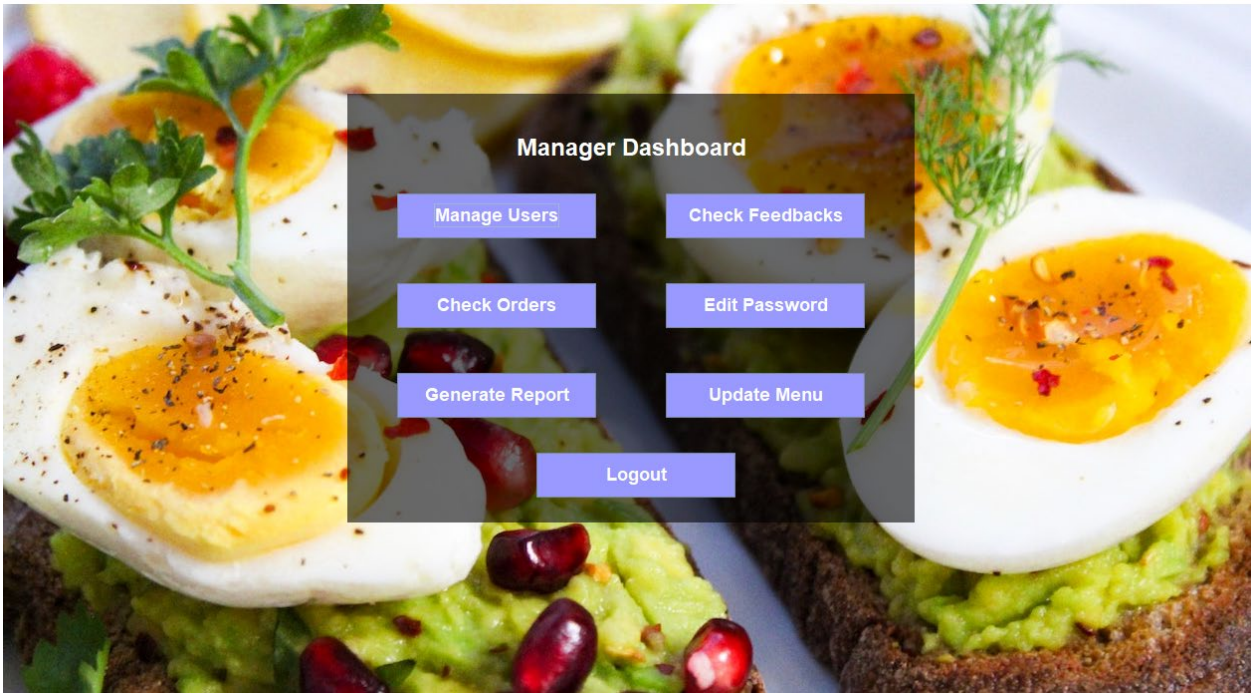
Once the user enters all the fields correctly, they will be successfully registered.



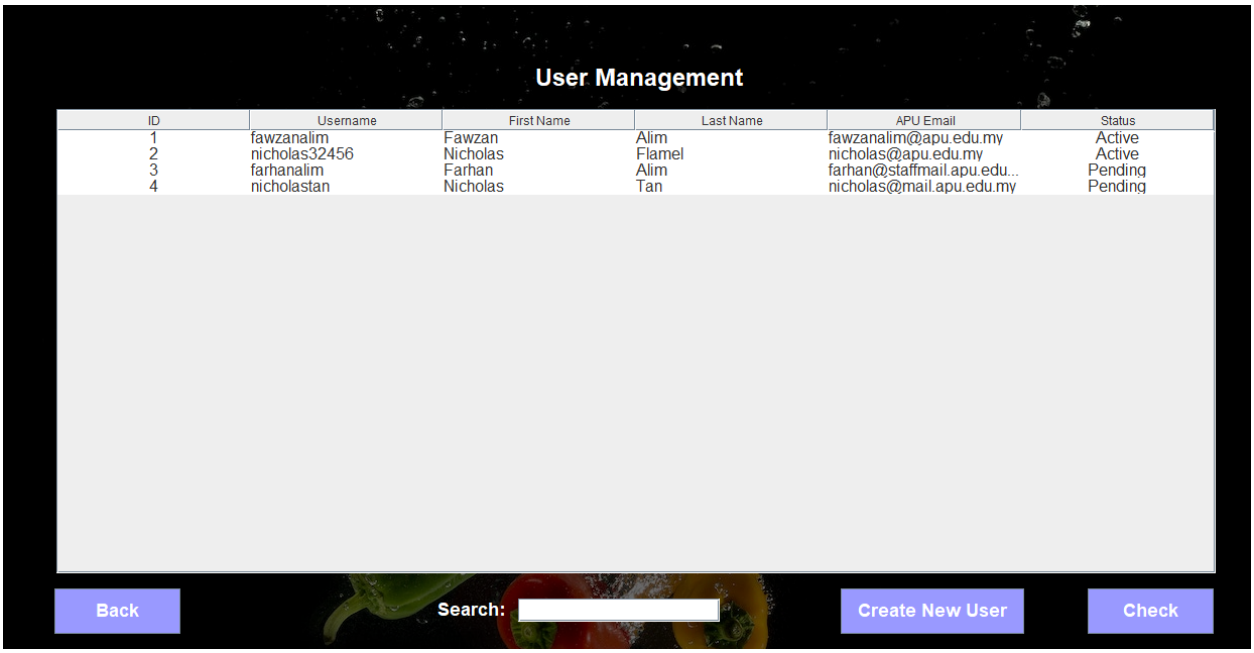
However, they can't use their account yet, new customer's account will have their status set to pending as they would need the manager to accept their registration.



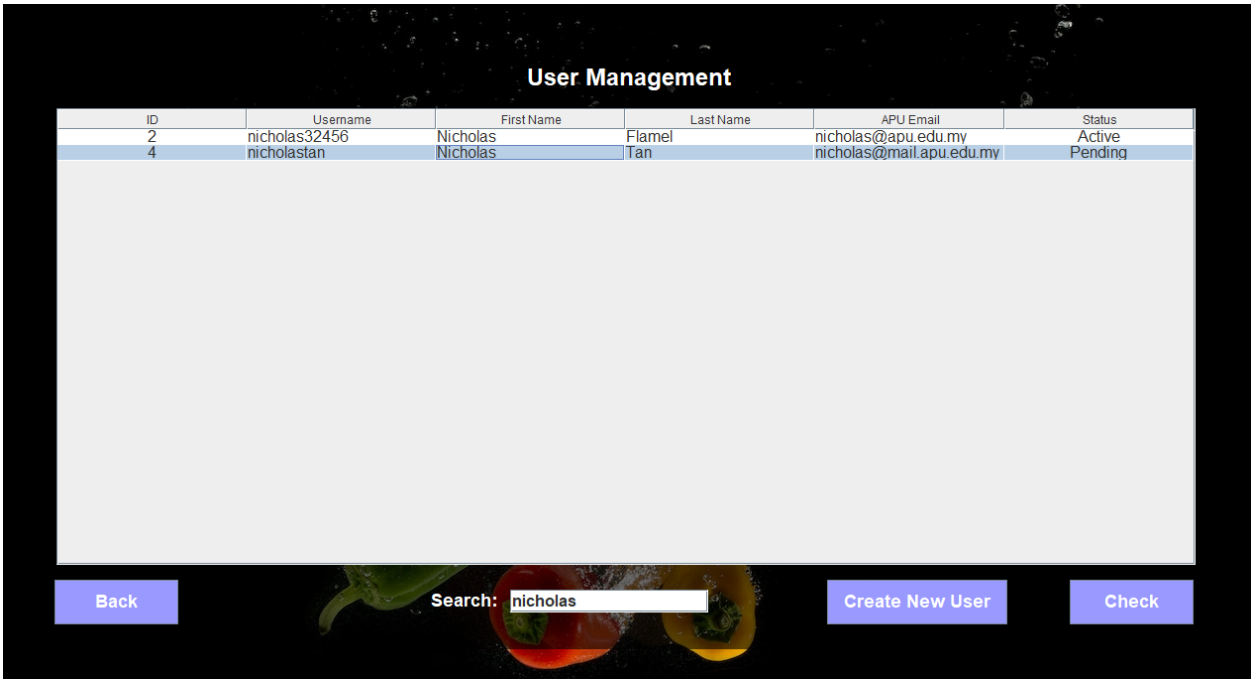
This is the login page but with the manager logging in this time instead of a customer.



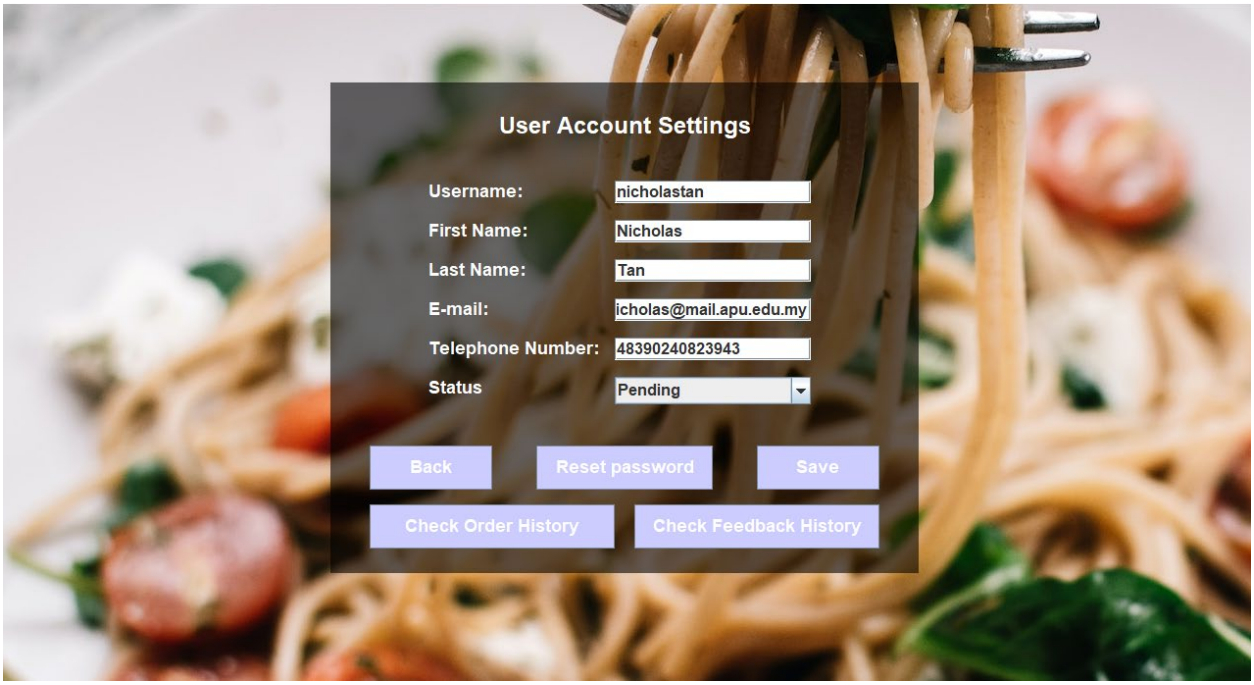
The manager has a sophisticated dashboard that allows the manager to perform various tasks.



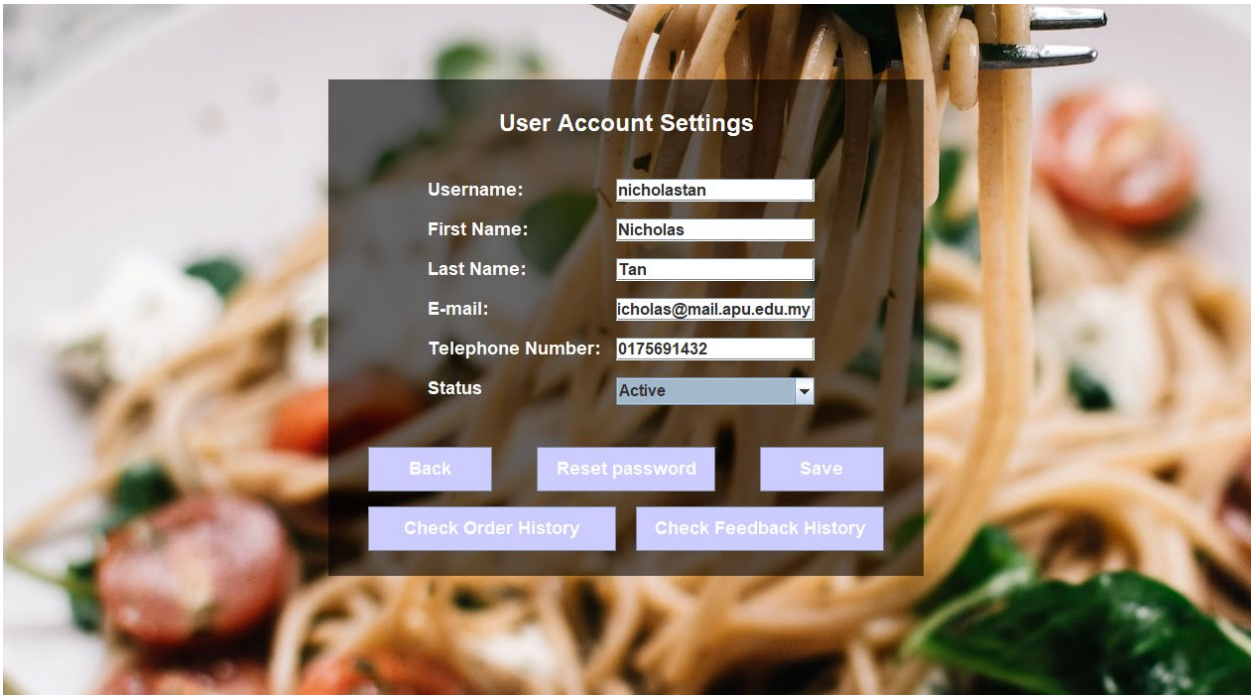
The manger can view all users and manage them by clicking on the “Manage Users” button on the dash board.



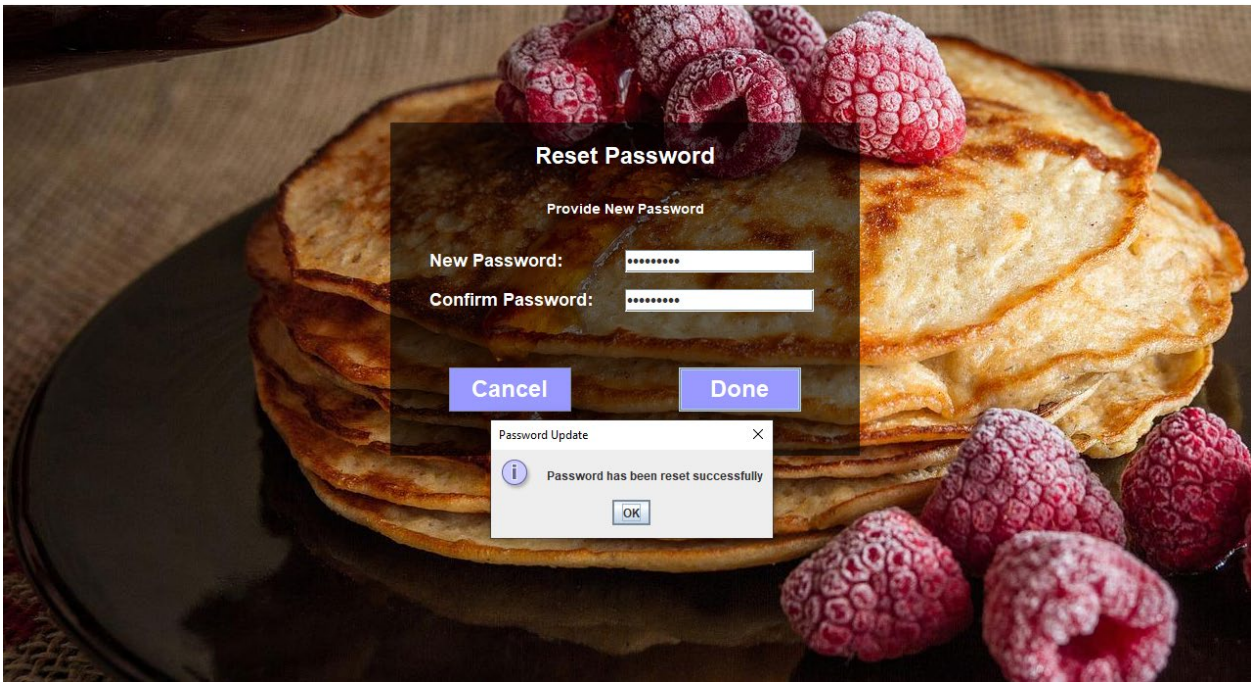
There is a search bar for the manager to get the details of a specific user.



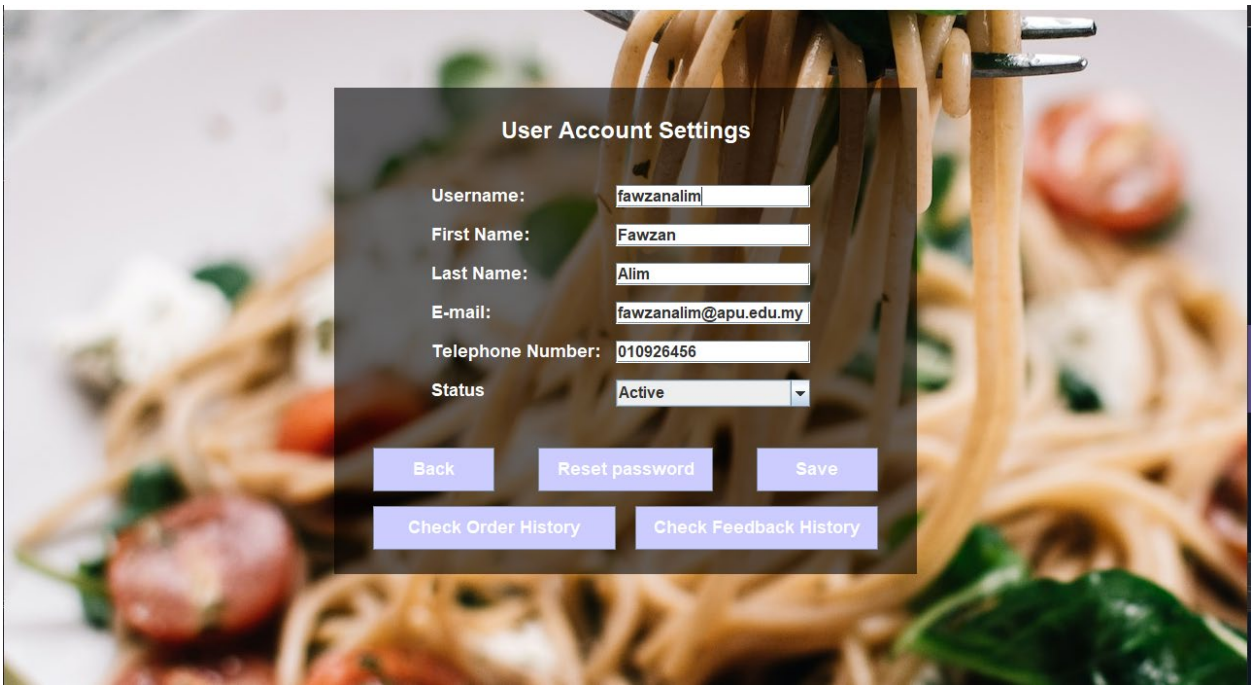
The manager can view the personal information of the customer in this page.



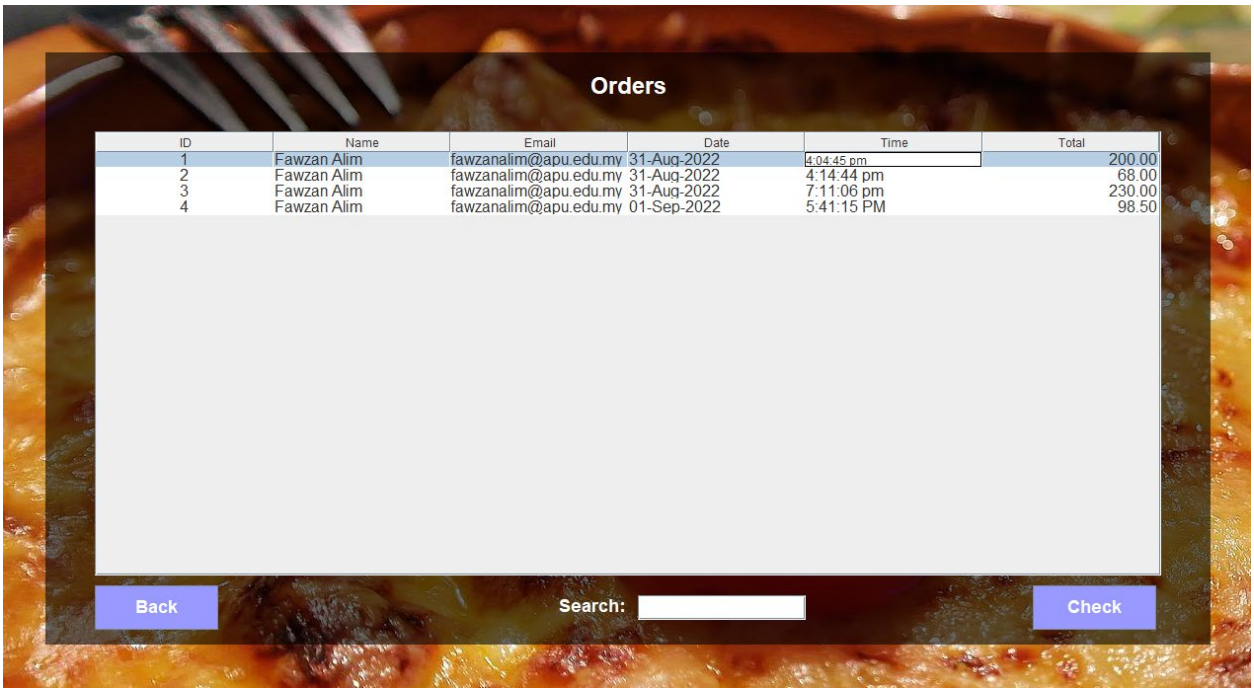
The manager can change the status of the customer's account from pending to active, this would activate the customer's account.



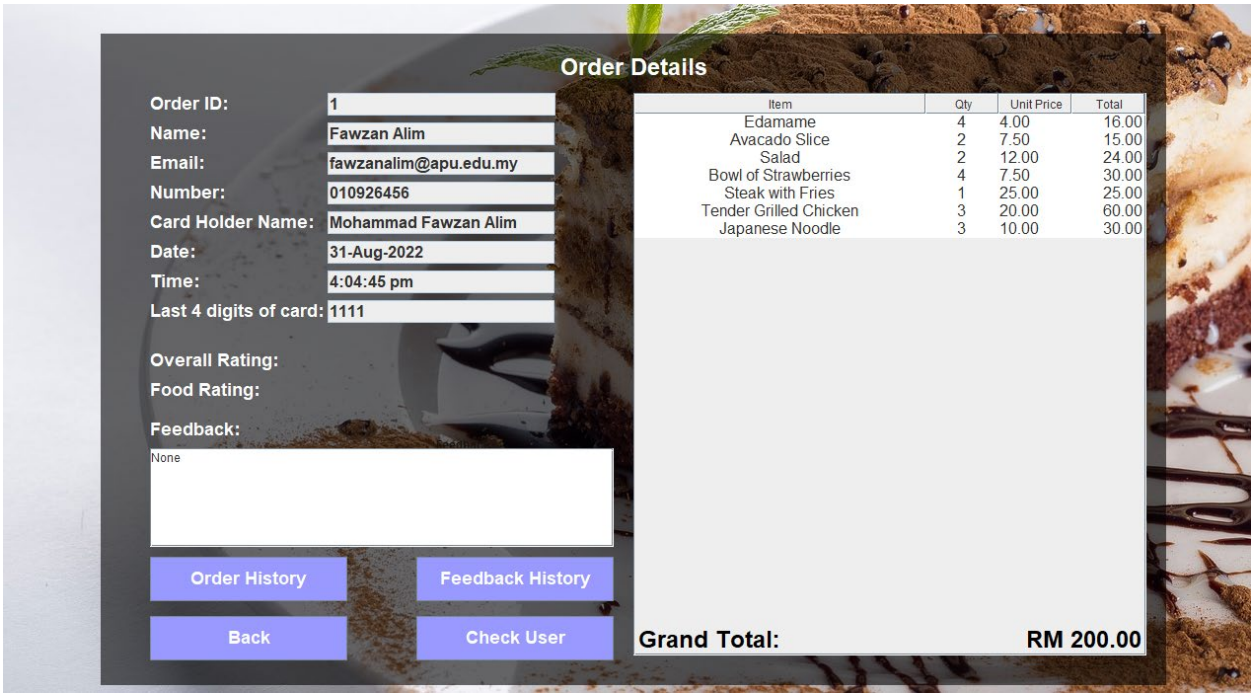
The manager can also reset the customer's password for them if the customer needs help.



In the account detail page, there is an option for the manager to check order history of that user.

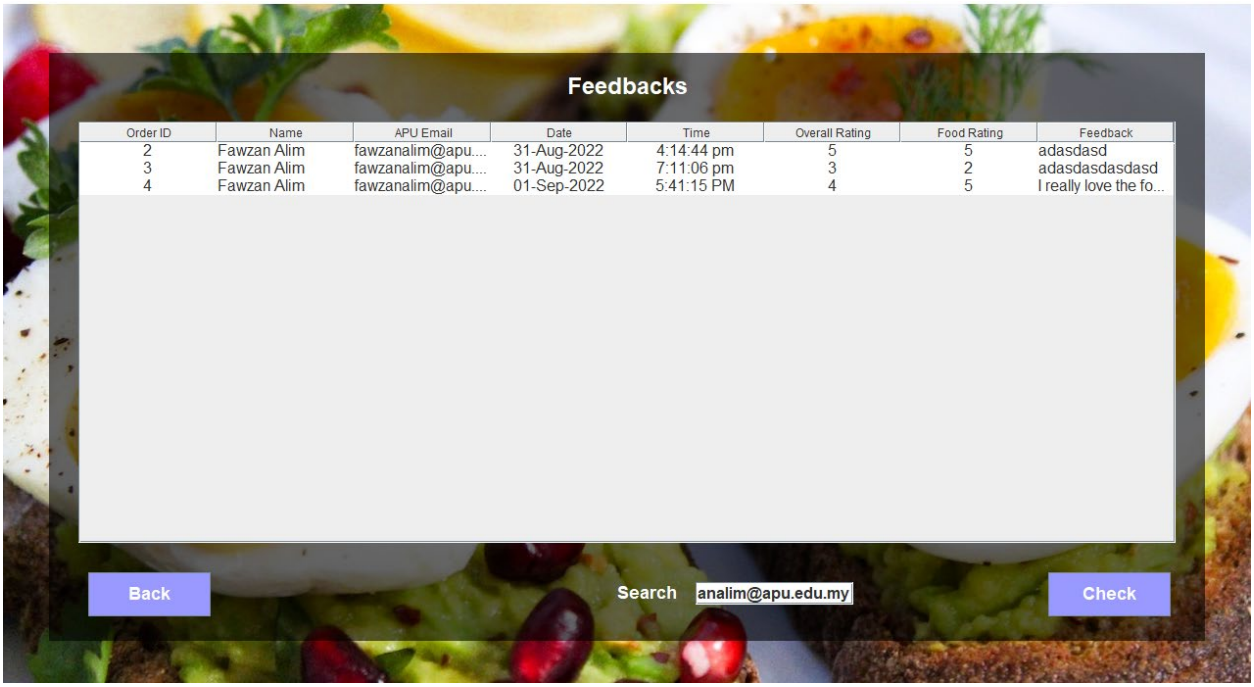


This shows the specific order history of that individual.

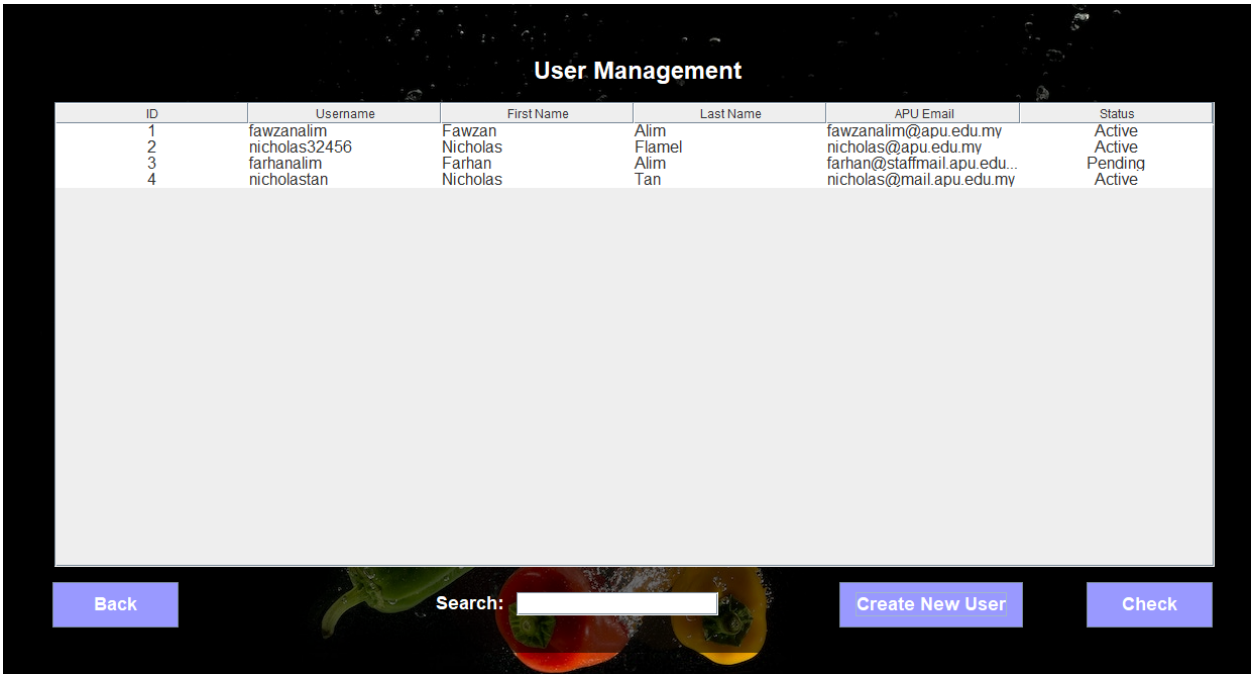


The manager can select the customer and check on them to view even greater detail about the order which includes the feedback given and the details of the order as well as the personal

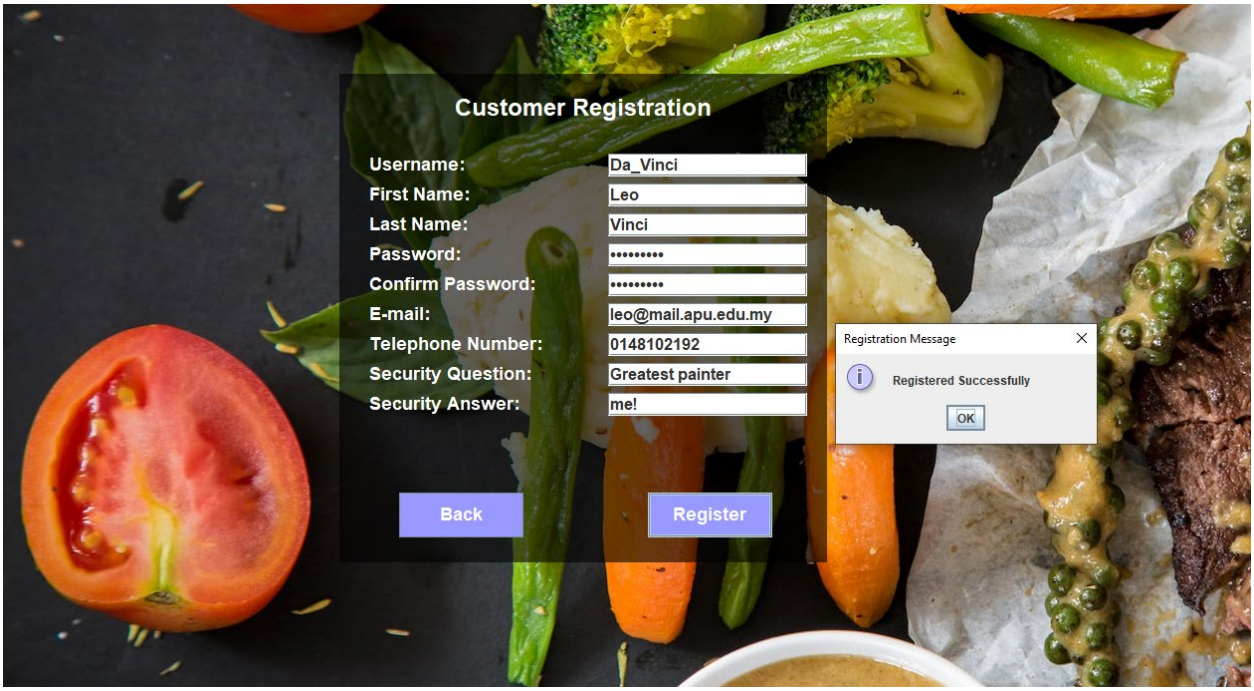
information of the customer. In this scenario, the user skips the feedback.



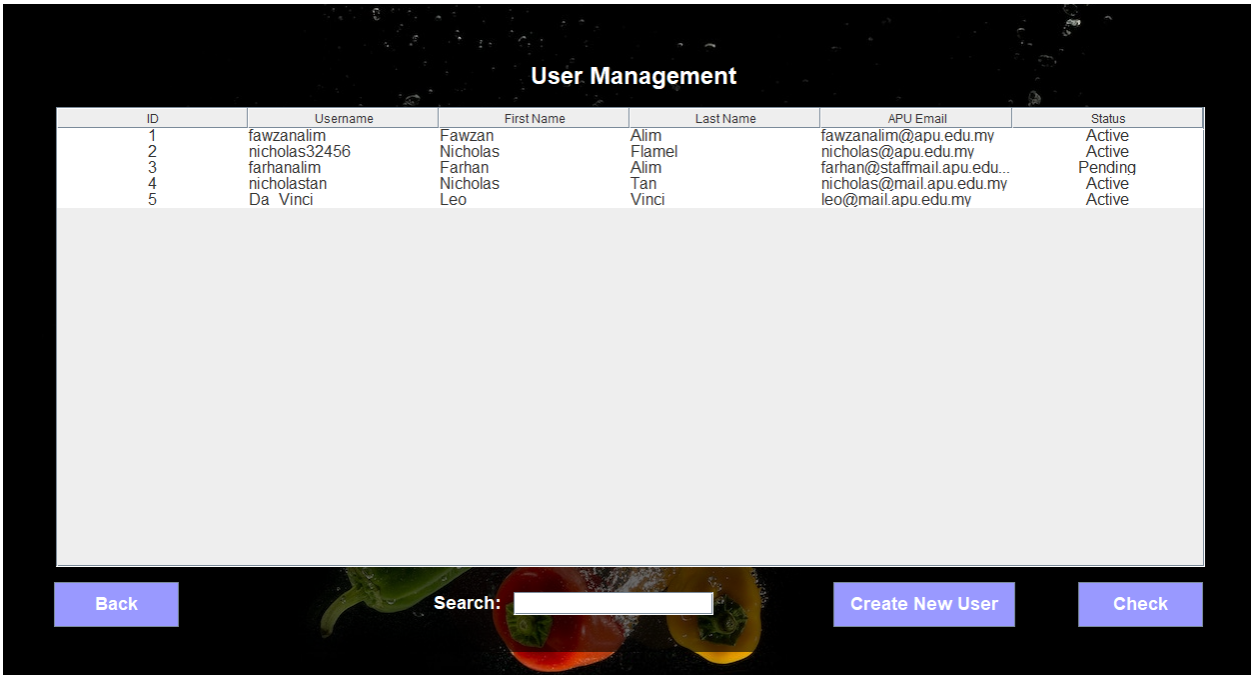
The manager can then view the feedback history of the customer, this shows all the feedback given by that individual.



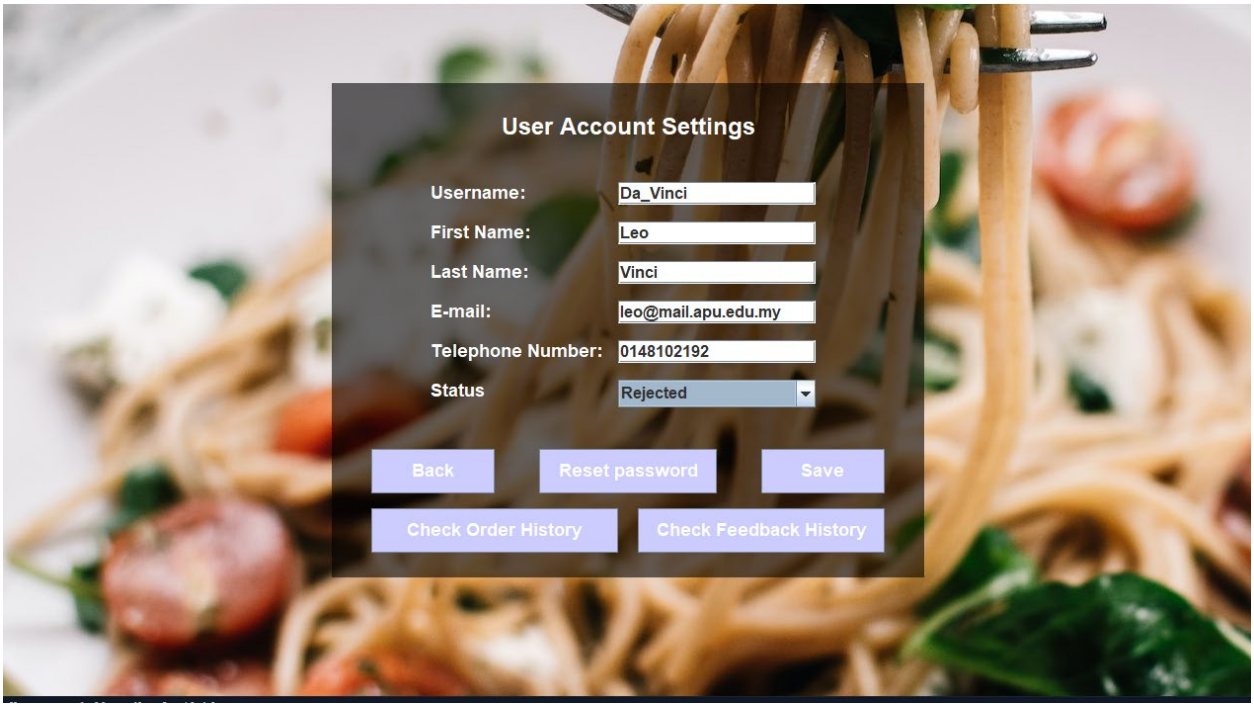
The manager can select and check the customers detail in this page. The manager can also register new customers for them.



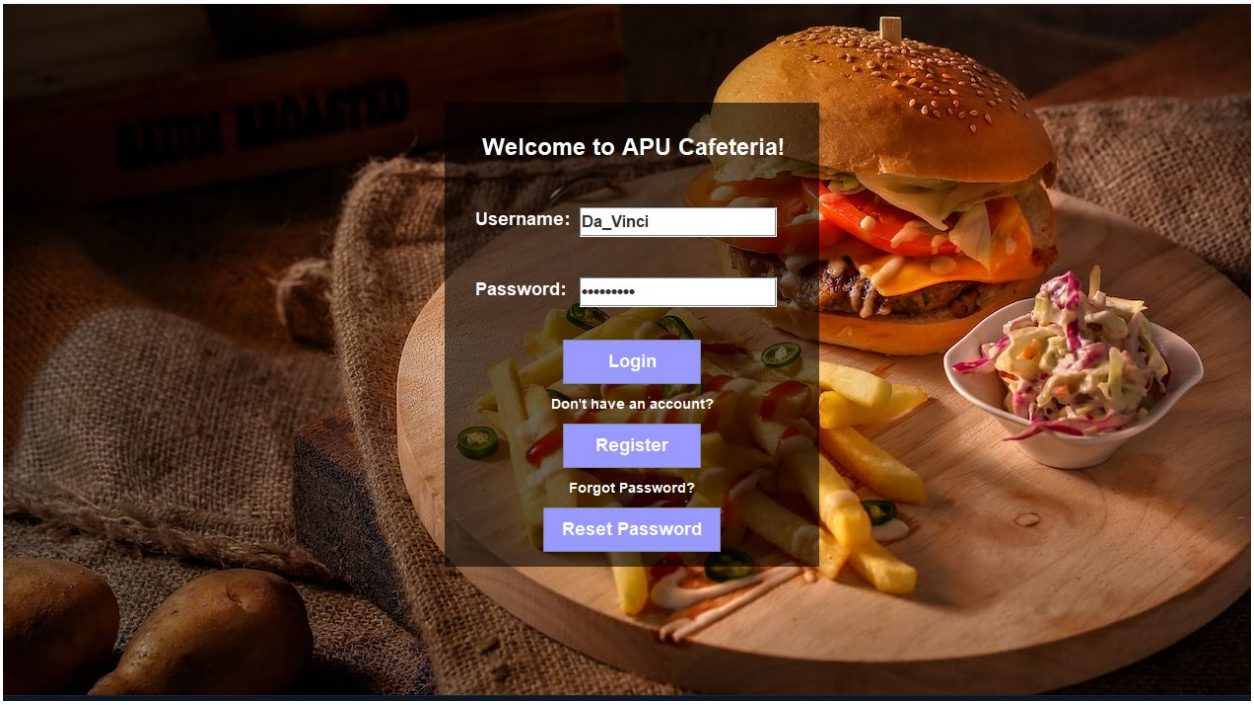
After entering the information of the customer, the customer's account is successfully created.



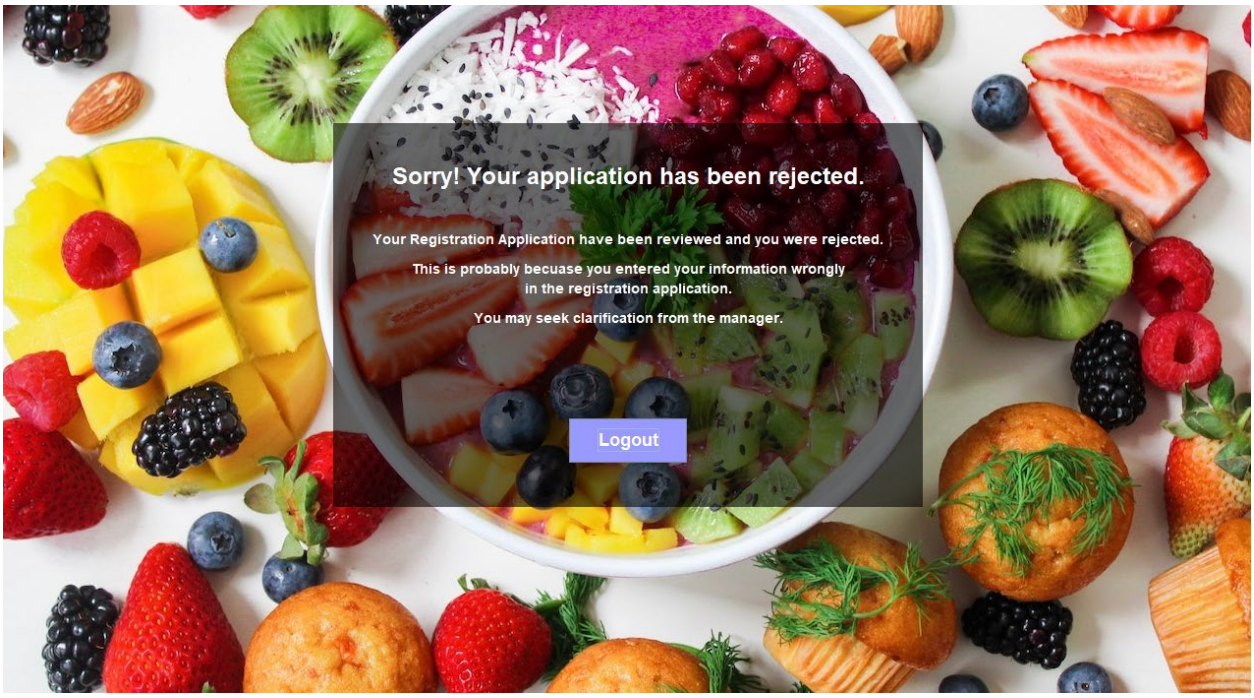
There is an additional feature where if a manager creates the account for the user, the status of the user is automatically set to active instead of pending.



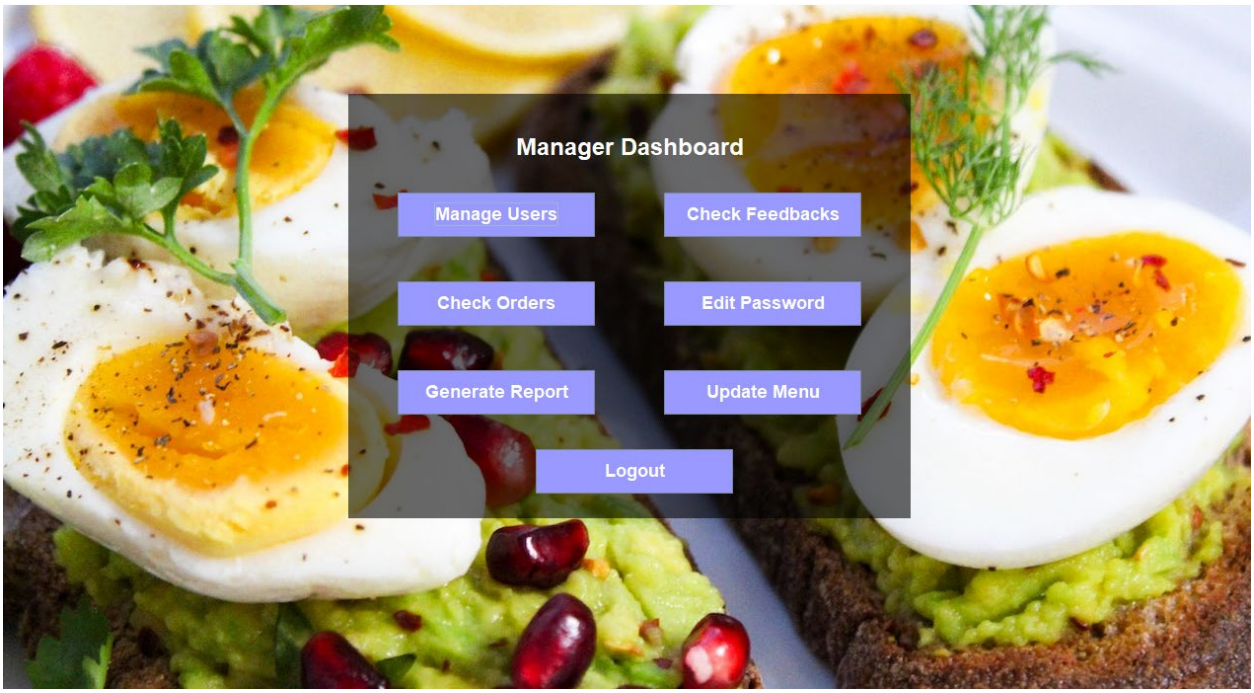
The manager can change the status of the active account into rejected.



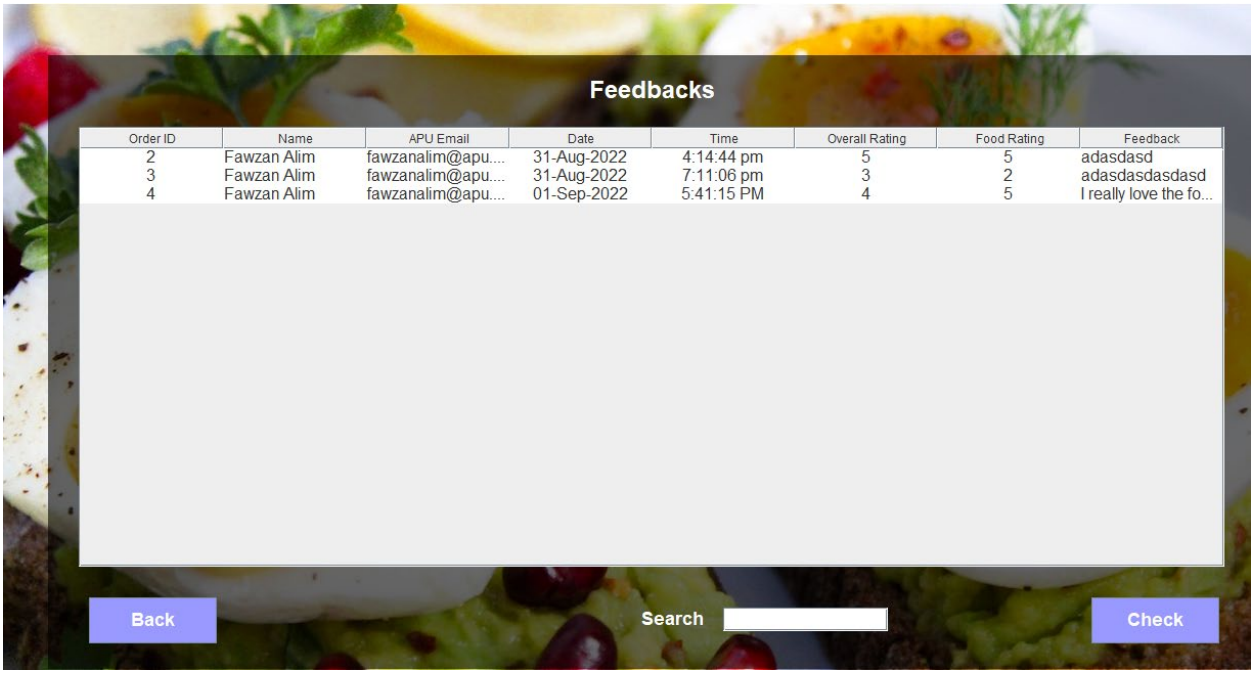
If the customer enters their credentials and tries to login, they will not be able to.



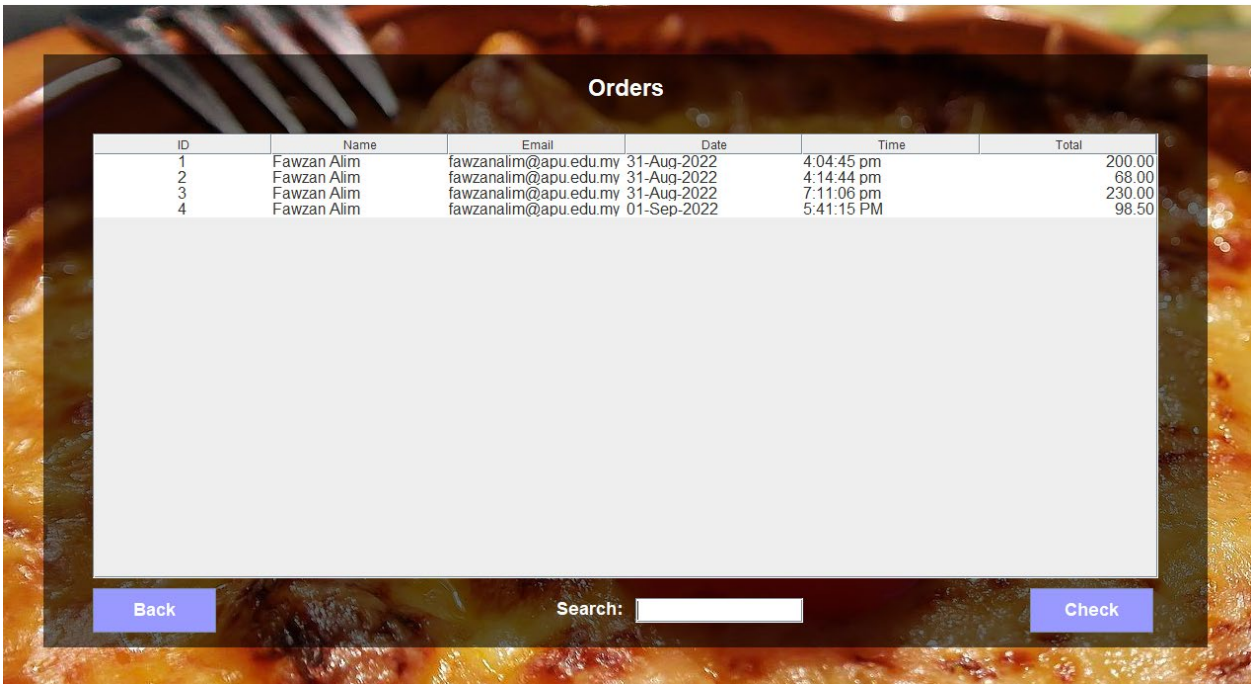
Customer's account with the rejected status will see this page, they cannot do anything but logout.



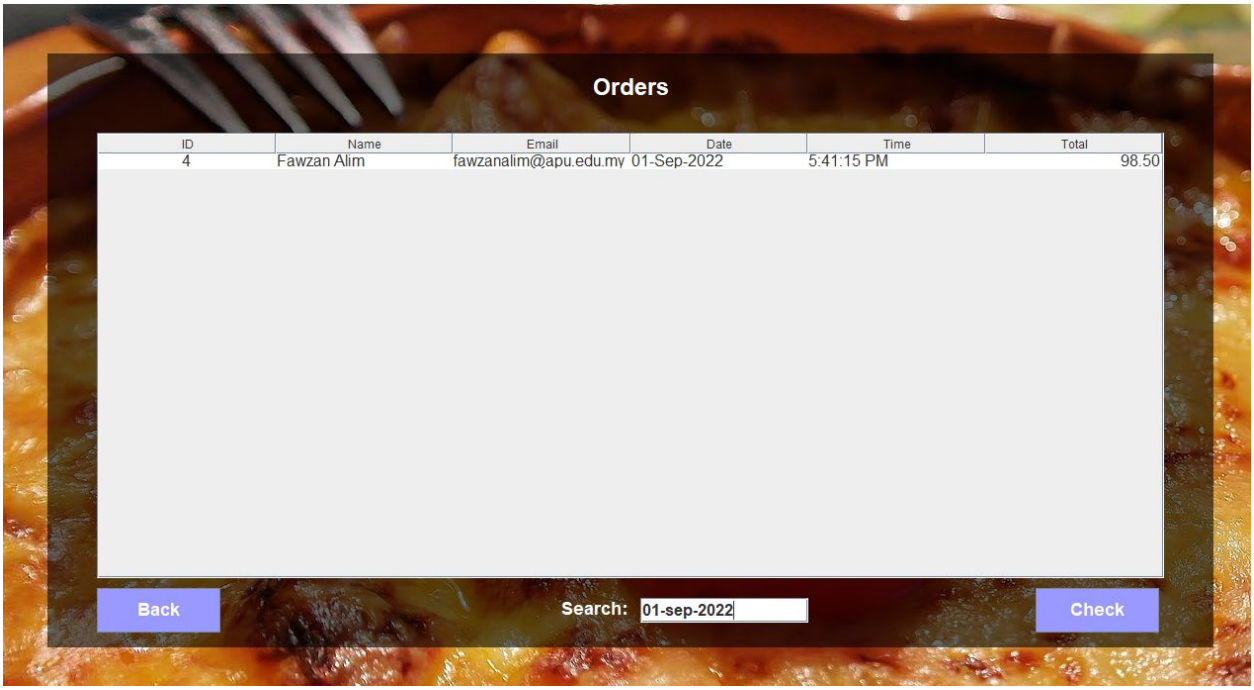
The manager can also check the feedback from the dashboard.



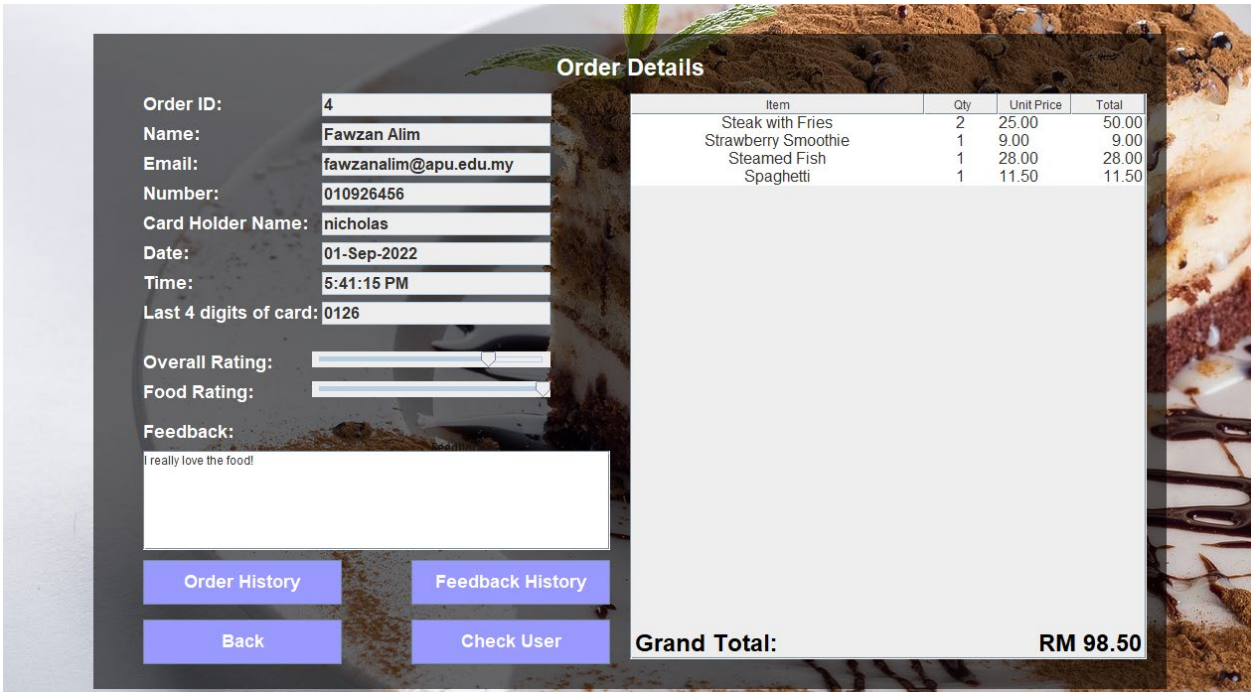
The user can view the general feedback given by the customer in this table.



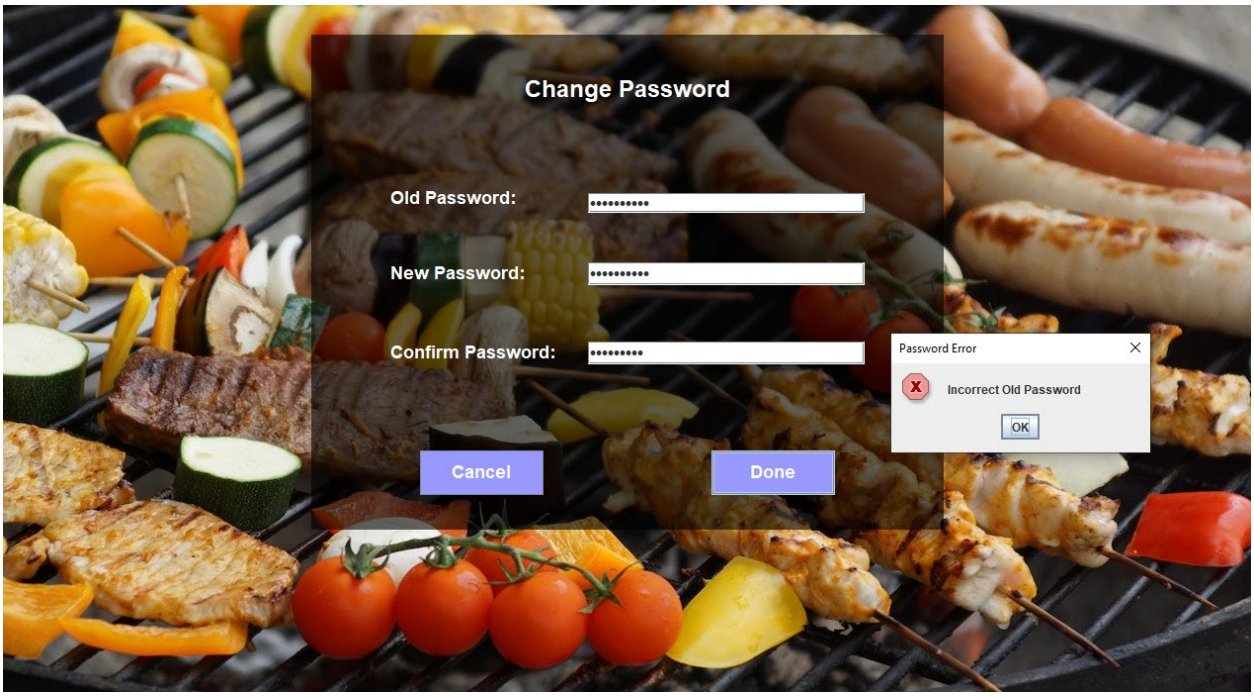
By pressing on the check button, the manager can quickly switch between the viewing the feedbacks given by the customers to viewing the orders made by the customers.



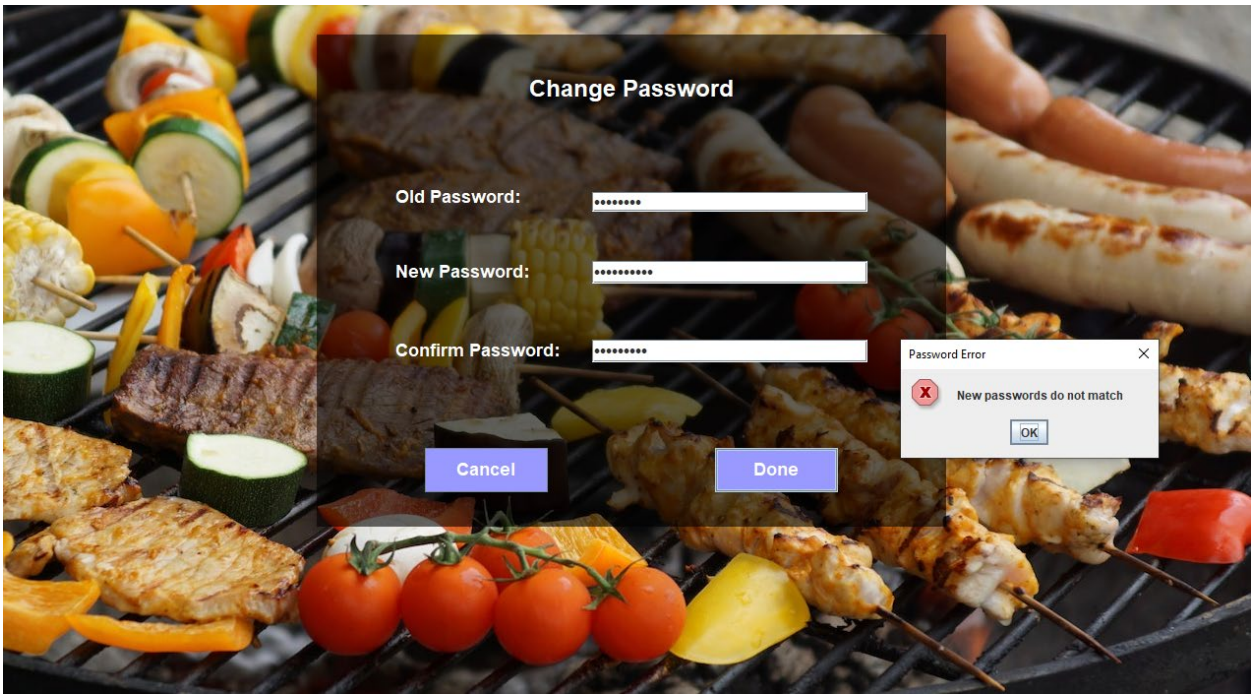
There is also a search functionality that allows the manager to specify the date or the name, or the email of the customer that they want to check on.

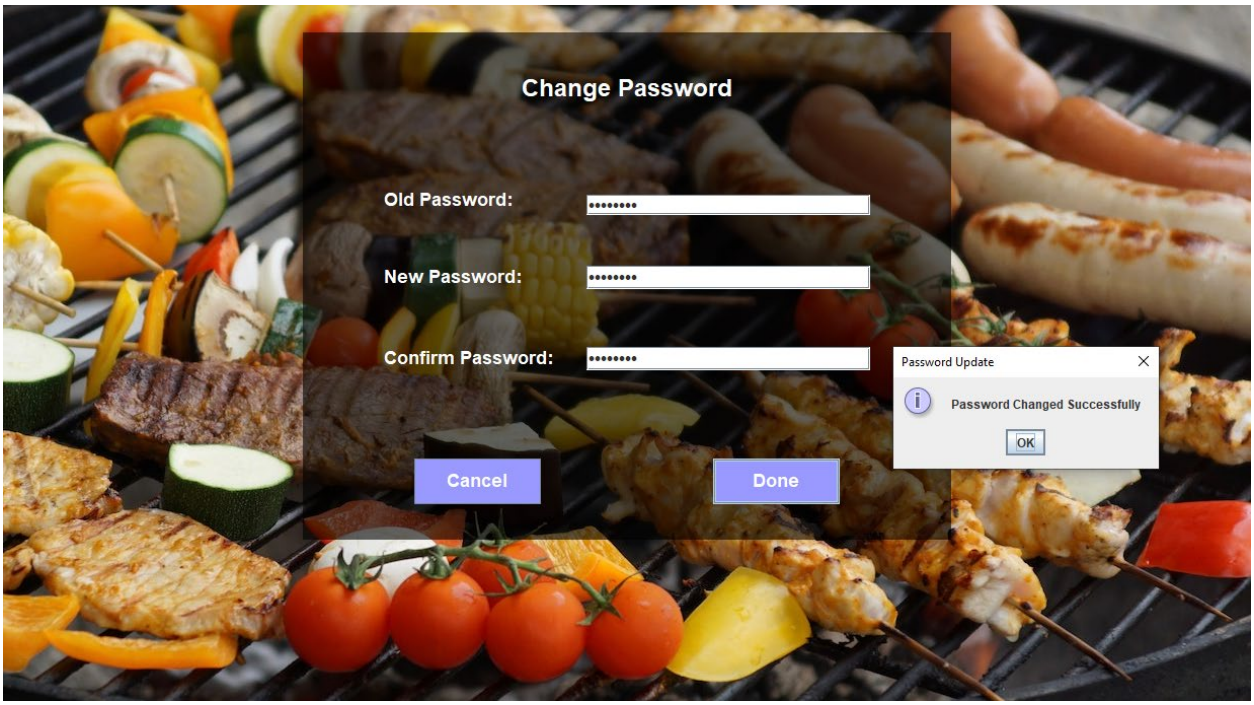


This is the page where the manager can view the order details. The manager can view the things that were ordered by the customer, the feedback given and the customer's details.

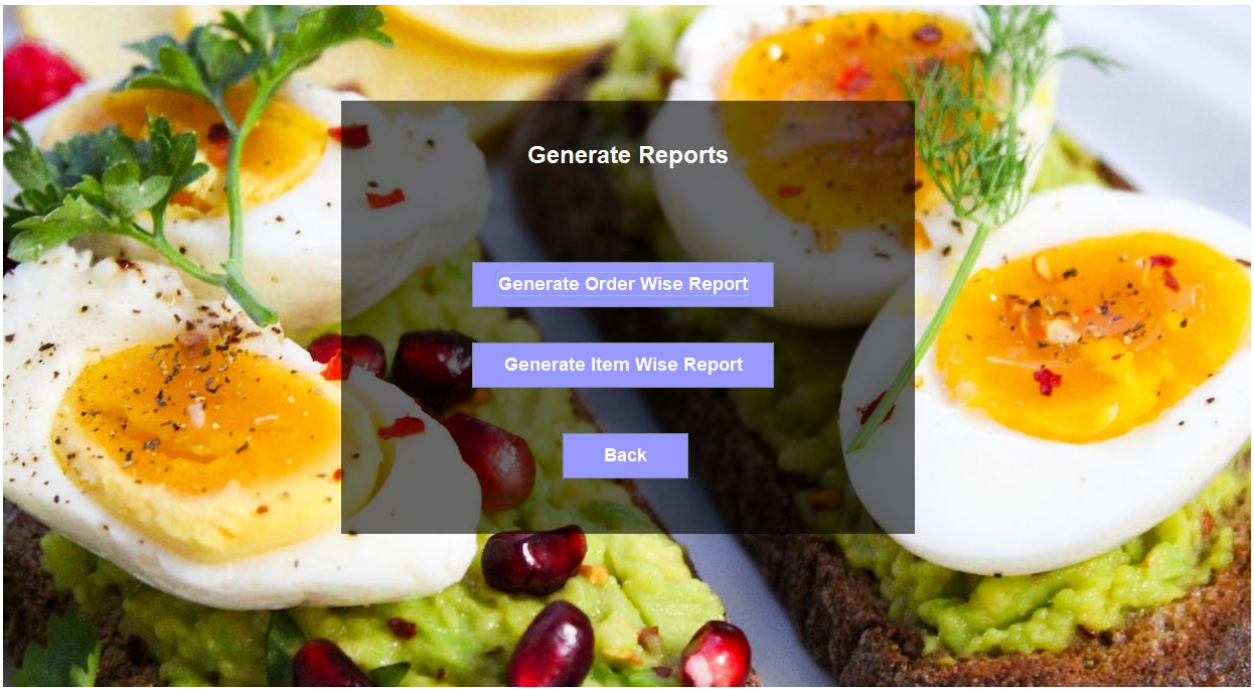


In addition to that, the manager can also change password. To add a security feature, the old password needs to be known before the password can be changed.

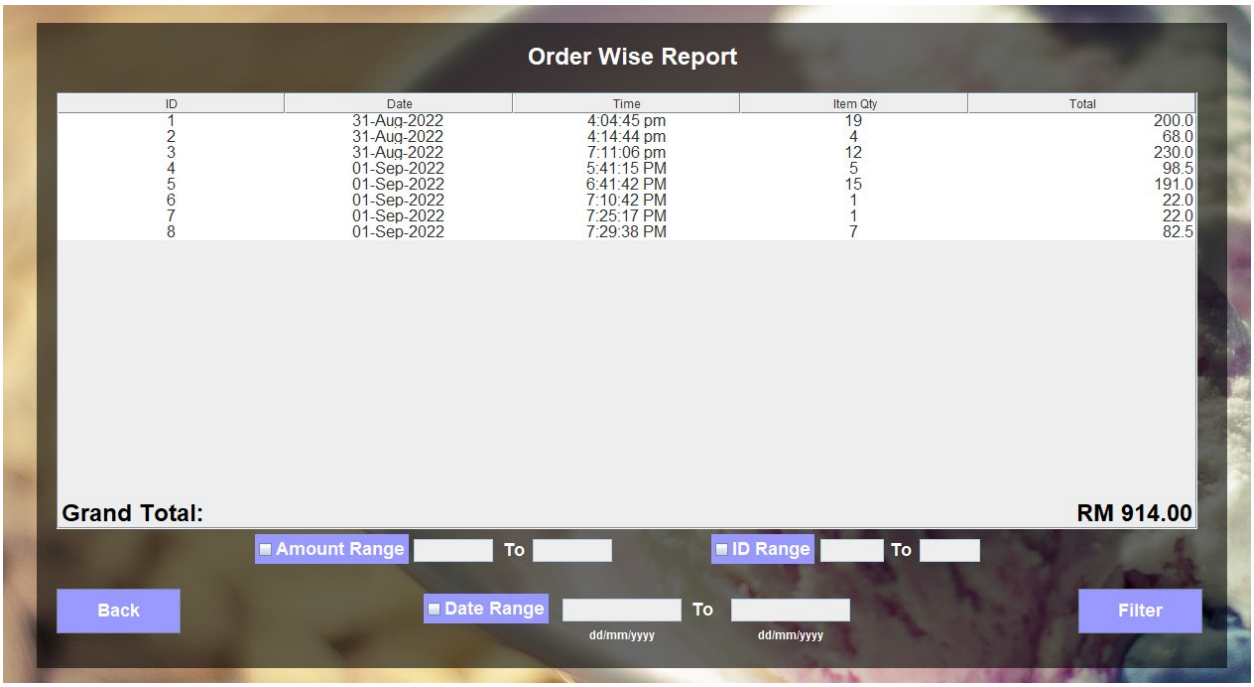




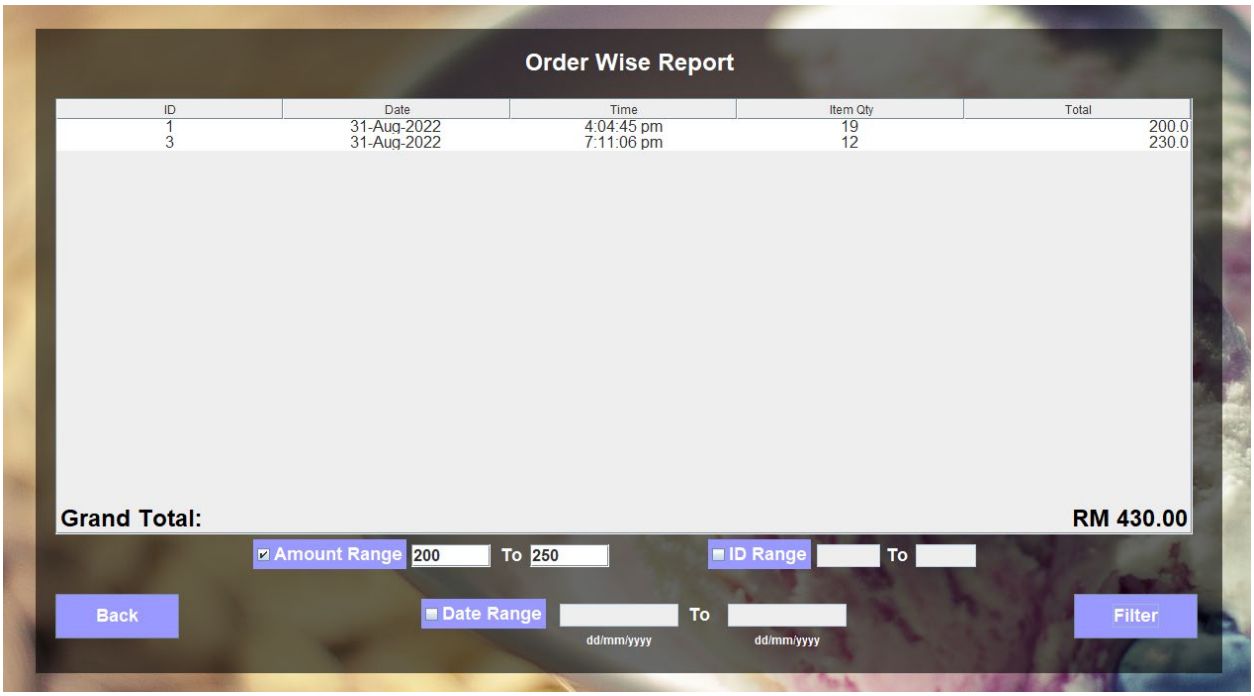
There are validations that prevents the manager's passwords from simply being changed. Only if the old password matches with the current password that the manager used to log in, and the new password is same with the text entered in the confirm password field, then the manager's password will be changed.



In addition to that, the manager can also view two different types of reports and this page can be accessed from the manager dashboard.

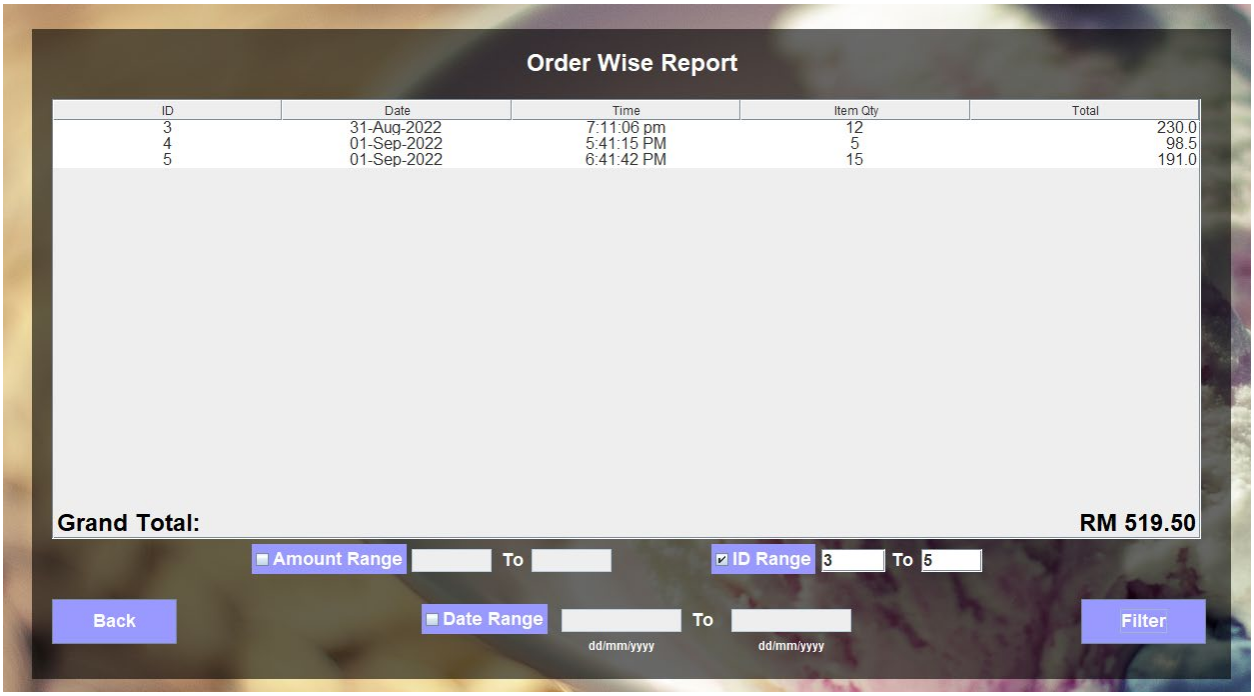


In the order wise report, the manager can see the list of all the orders in ascending order of the order ID. The table also contains the date, time, item quantity and the total amount of each order. At the bottom of the table, it also shows the grand total of the orders. There are three filters manager can use to filter out the orders, which need to be enabled using appropriate checkboxes.

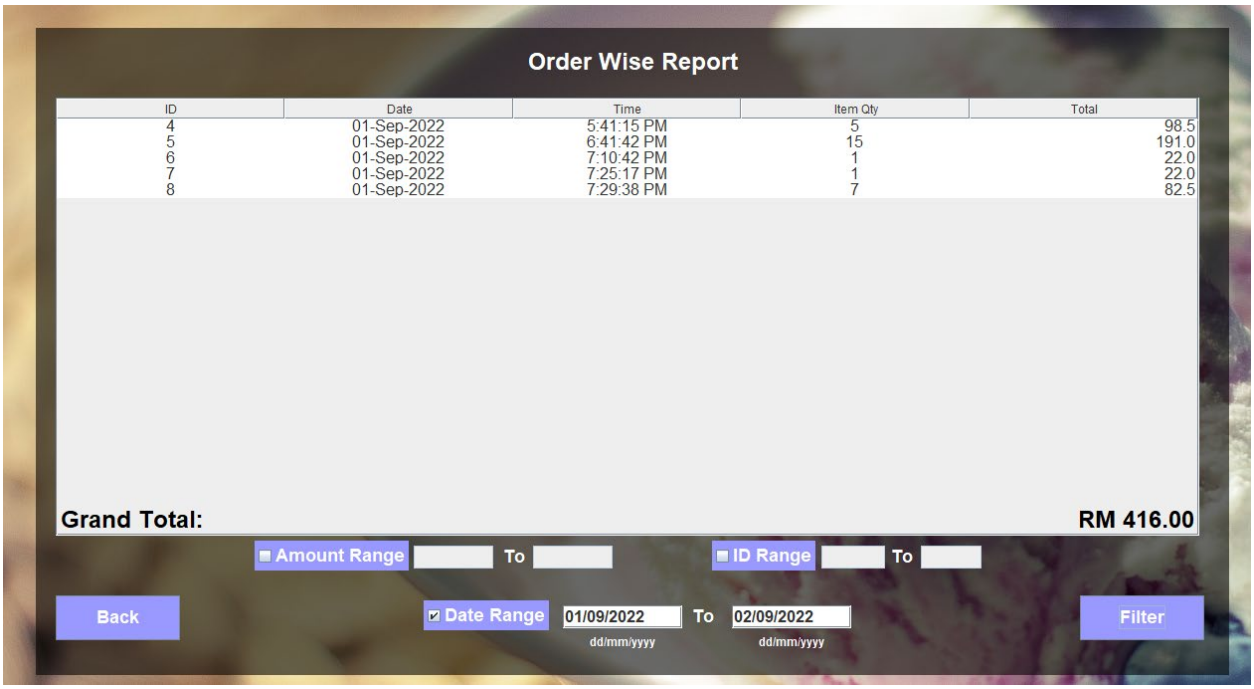


Here we can see amount filter working. It is filtering out the orders with the total amount

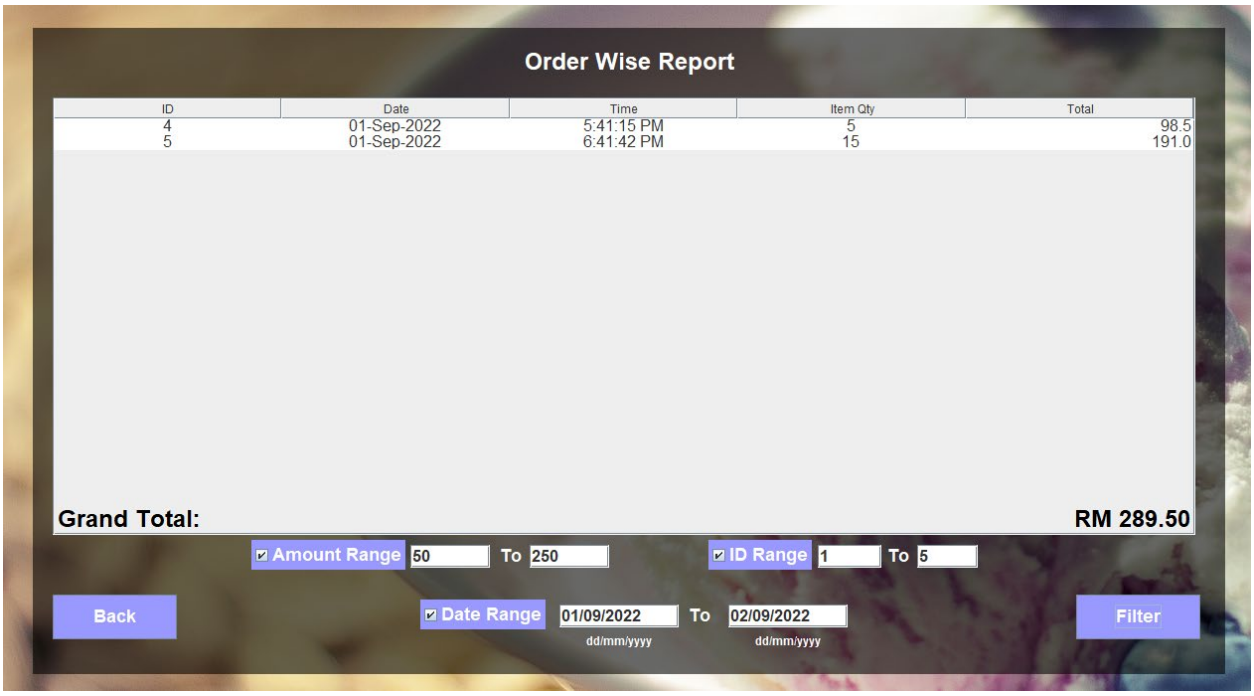
between 200 and 250. In order to filer by amount, manager need to provide starting amount and ending amount then click on filter button. This filter is very useful for analyzing order pattern of the customer.



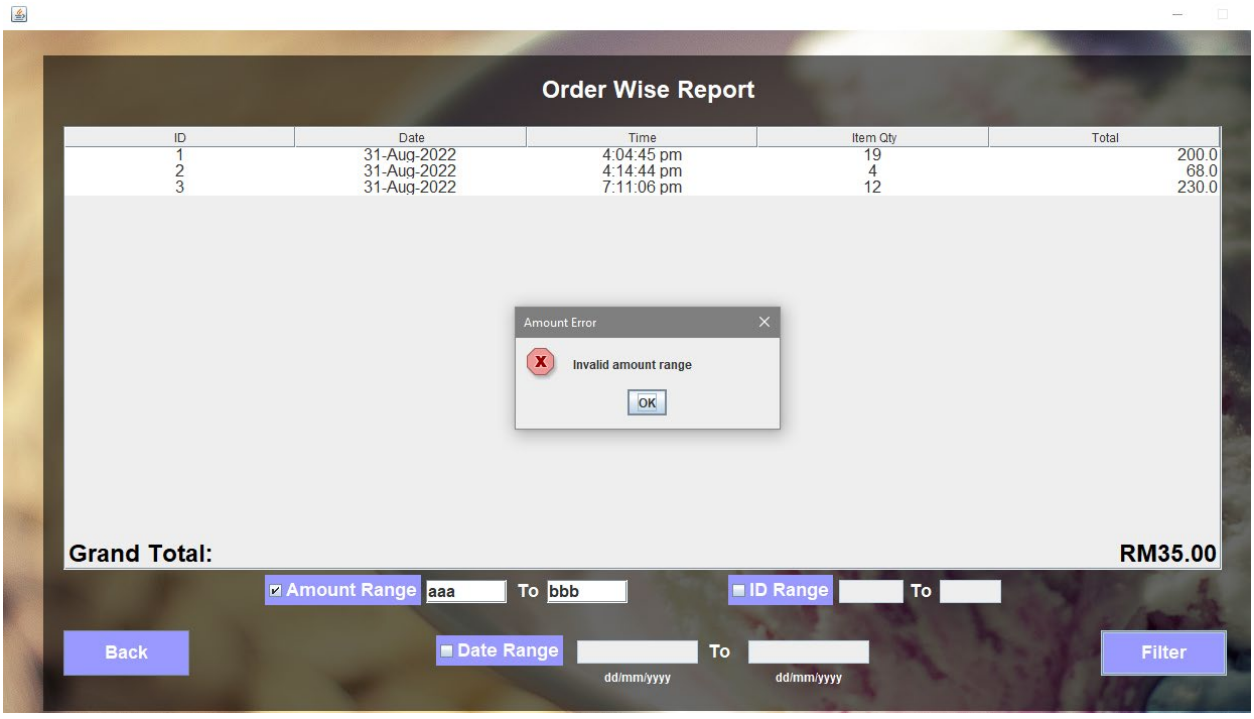
In the figure above, the table is filtered using ID. It is showing all the orders with the id between 3 and 5. In order to use the filter, manager need to enable it using the check box and provide proper input. After that, they just need to click on the filter button.



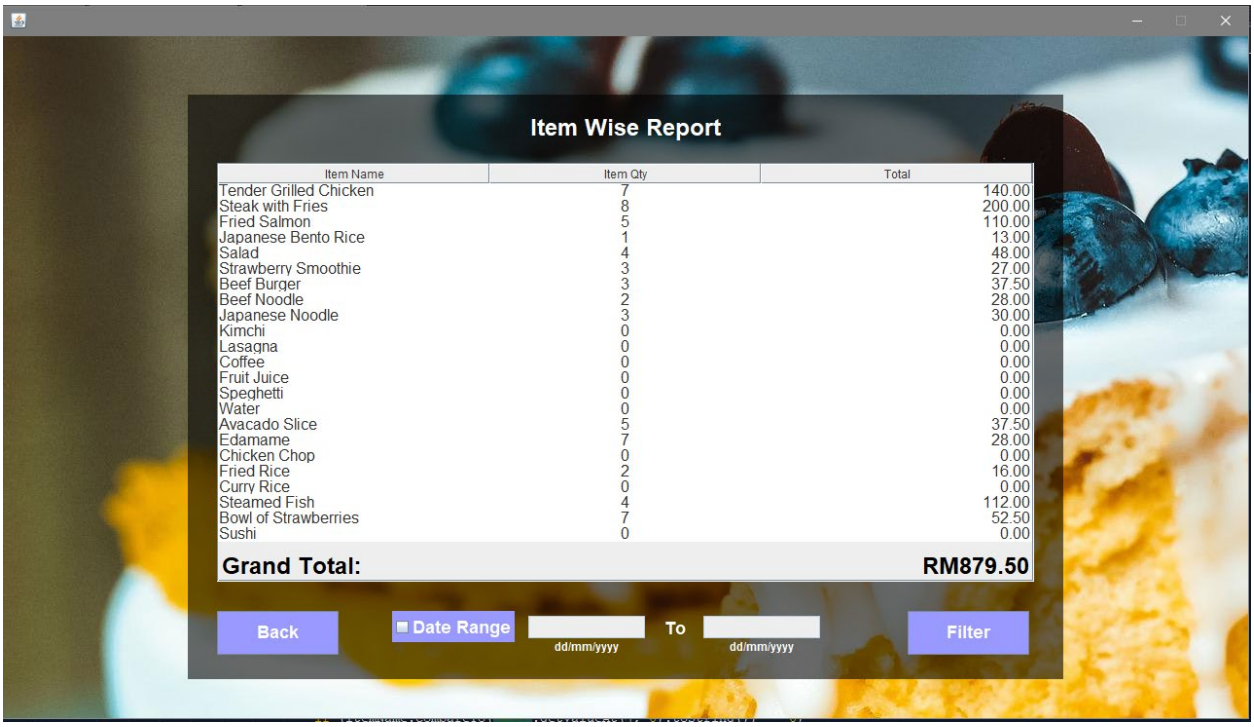
Last filter, which is filter by date is used here. The table is filtered to show the orders between the date 1st September and 2nd September. In order to use the filter, user need to enable the filter using the checkbox. Then provide proper input as instructed below the text field. If invalid input is provided, it will show error message. After that, manager simply need to click on filter button to filter out the table. This date filter is very useful to generate a daily, weekly or monthly report.



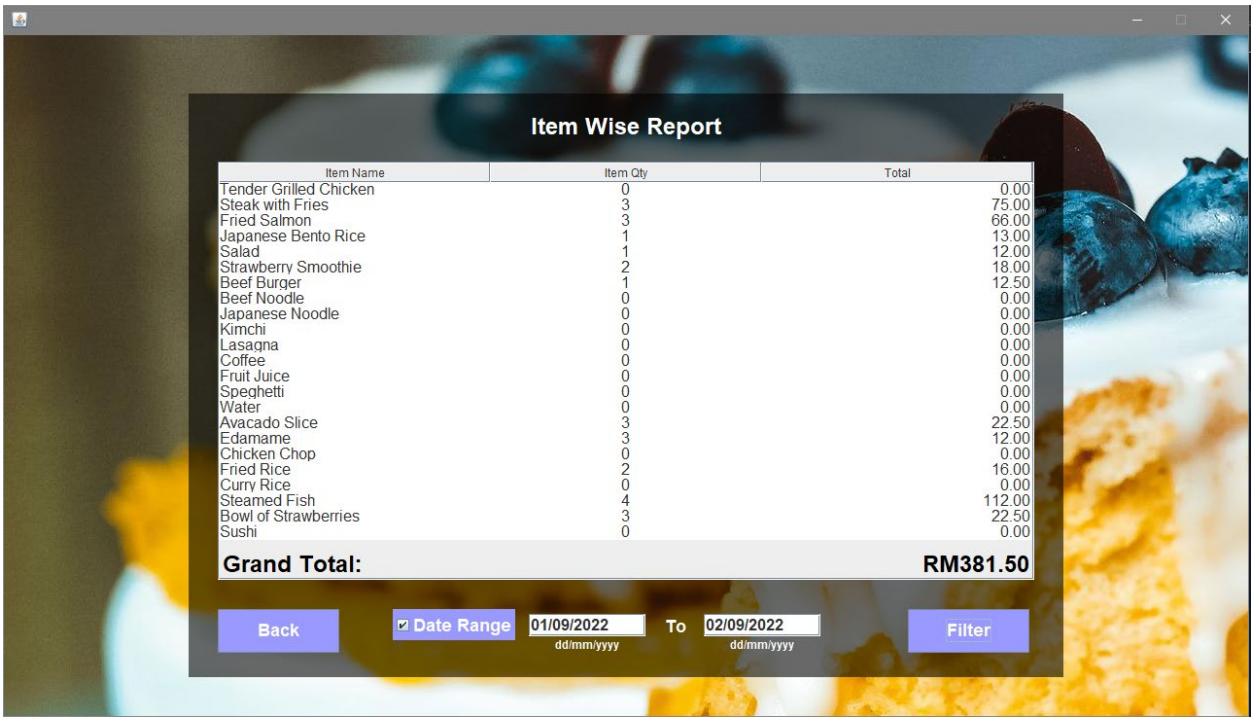
Here we can see all three filters working at the same time.



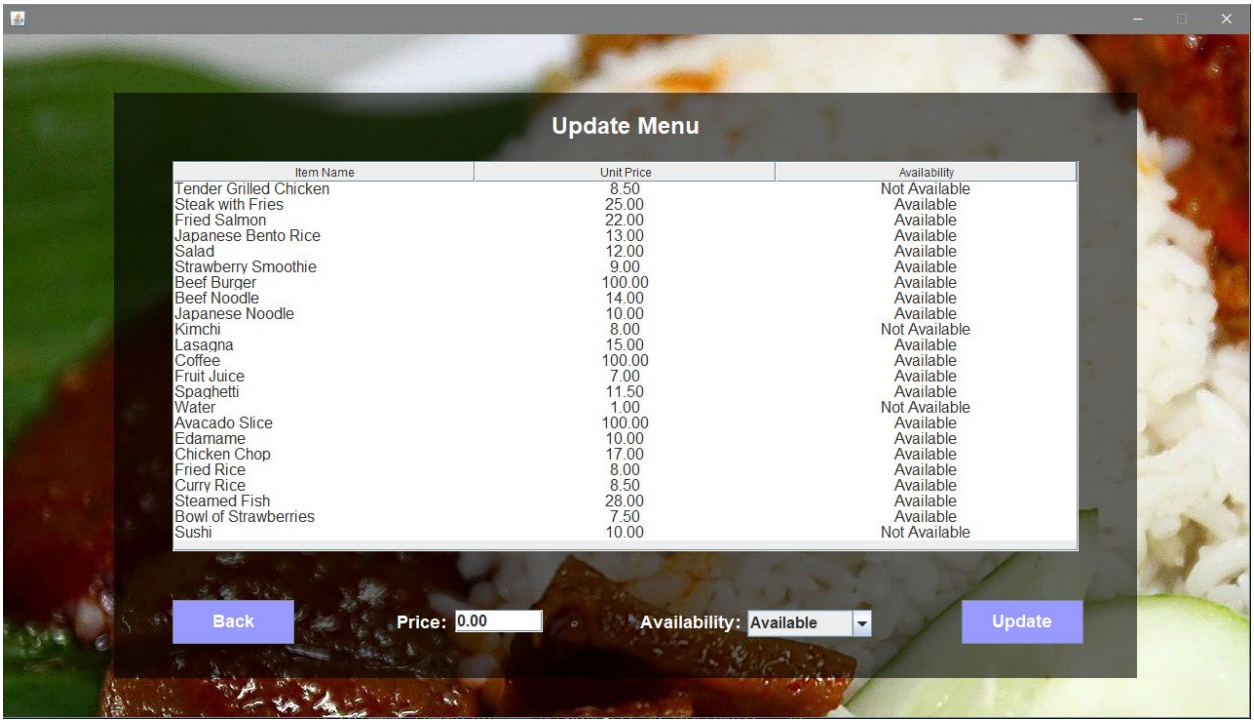
All the inputs of the filters goes through validation check and shows proper error message if the input is invalid. Here we can see that the amount should have been a number but the manager provided a string, therefore it is giving an error.



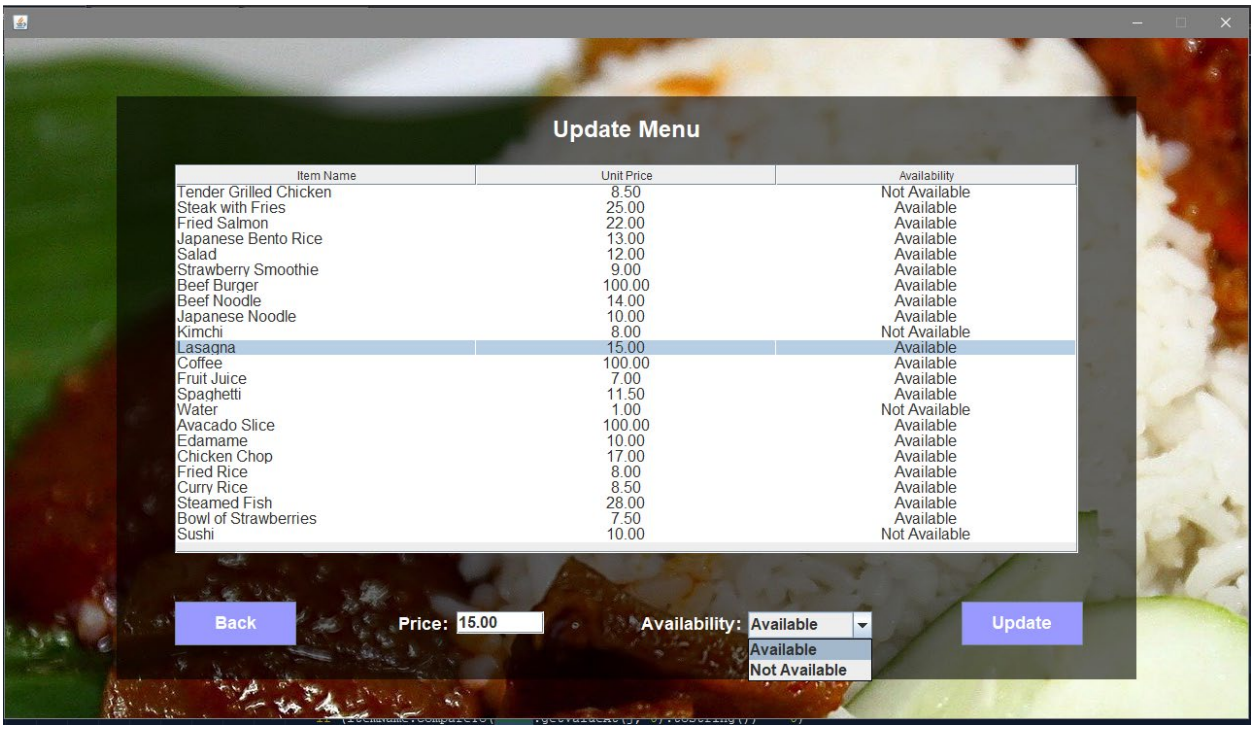
Item wise report is similar to order wise report. Instead of showing all the orders, it shows all the items, along with quantity and amount sold. At the end of the table, it shows the grand total. This will be very useful for the manager to analyze the products and take business decisions based on the user demands.



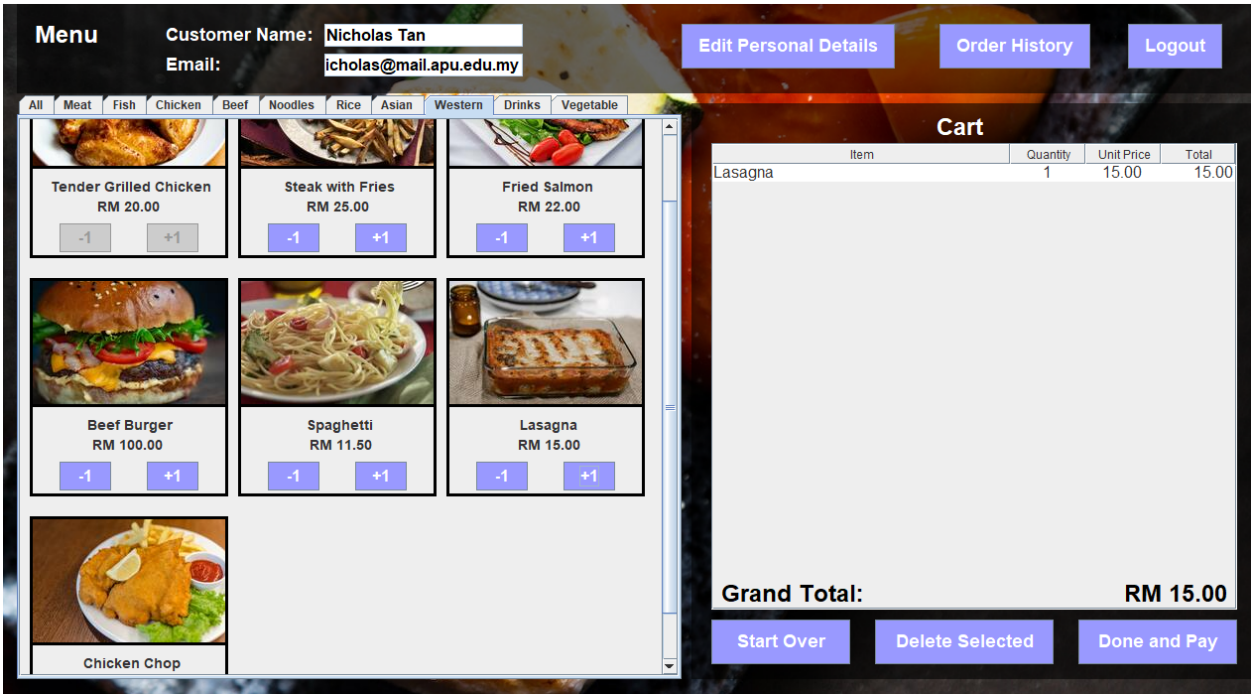
Unlike order wise report, there is only one filter for this report. Manager can filter out the report using the date range. This way, the manager can generate an item wise report for a specific day or month which will help them analyze their products properly within a given date range. Just as before, the date filter needs to be enabled before use. It goes through multiple validations and shows appropriate error message when an invalid input is provided.



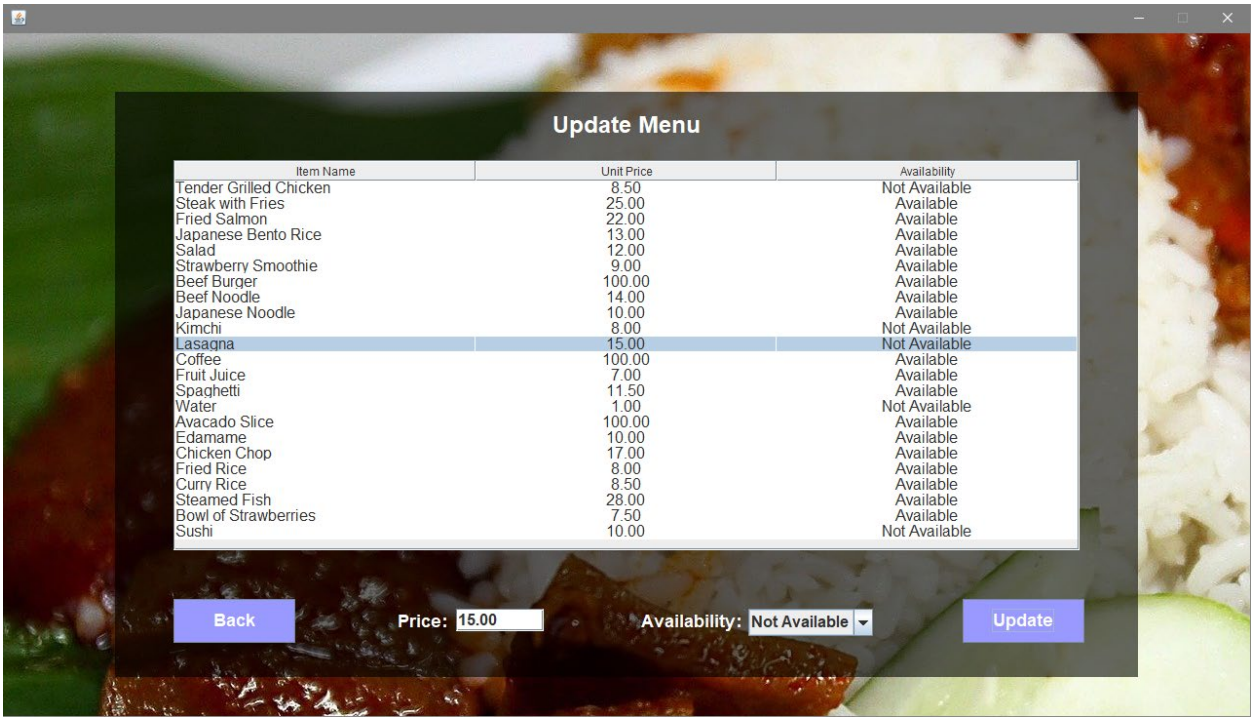
One of the requirements of the program was to let the manager update menu. This is that function. Manager can access this window from the manager dashboard. Here, all the items with their price and availability in stock can be seen



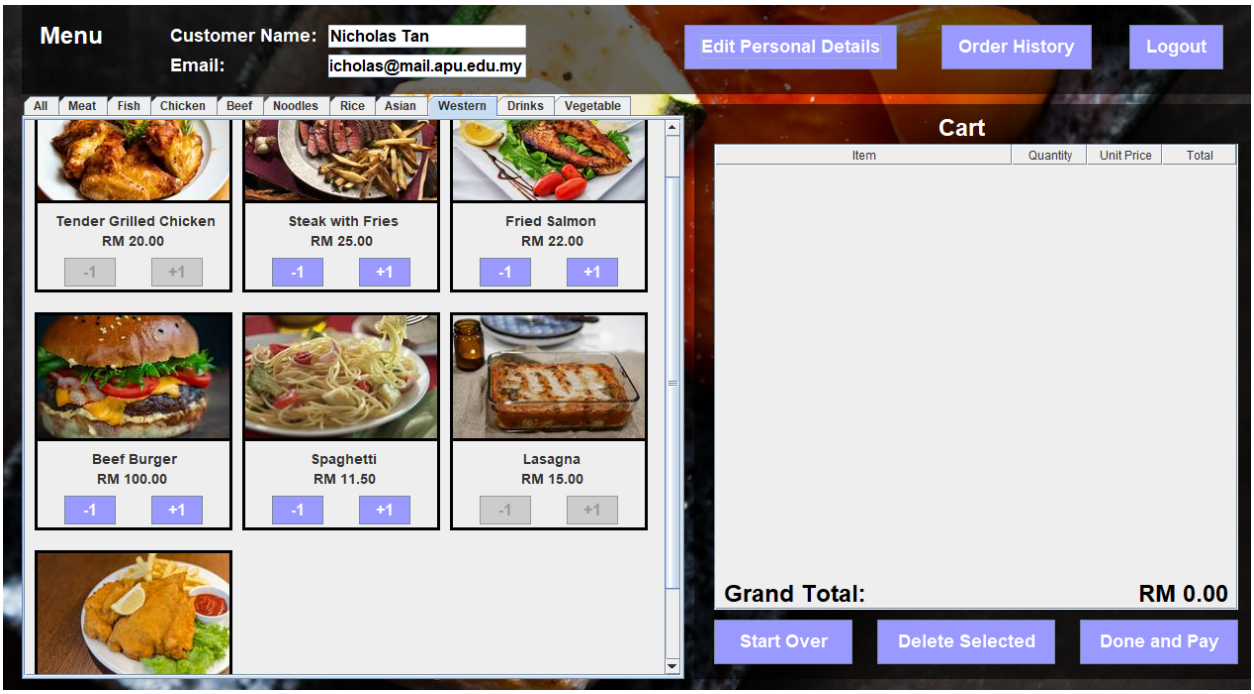
Manager can select an item and update its price as well as availability status using the input fields at the bottom. When the user selects a row, the price and available drop-down status reads the values from that row. Manager can then modify the values and click on update button to update the table. At the same time it will also update in the file which customer menu reads from.



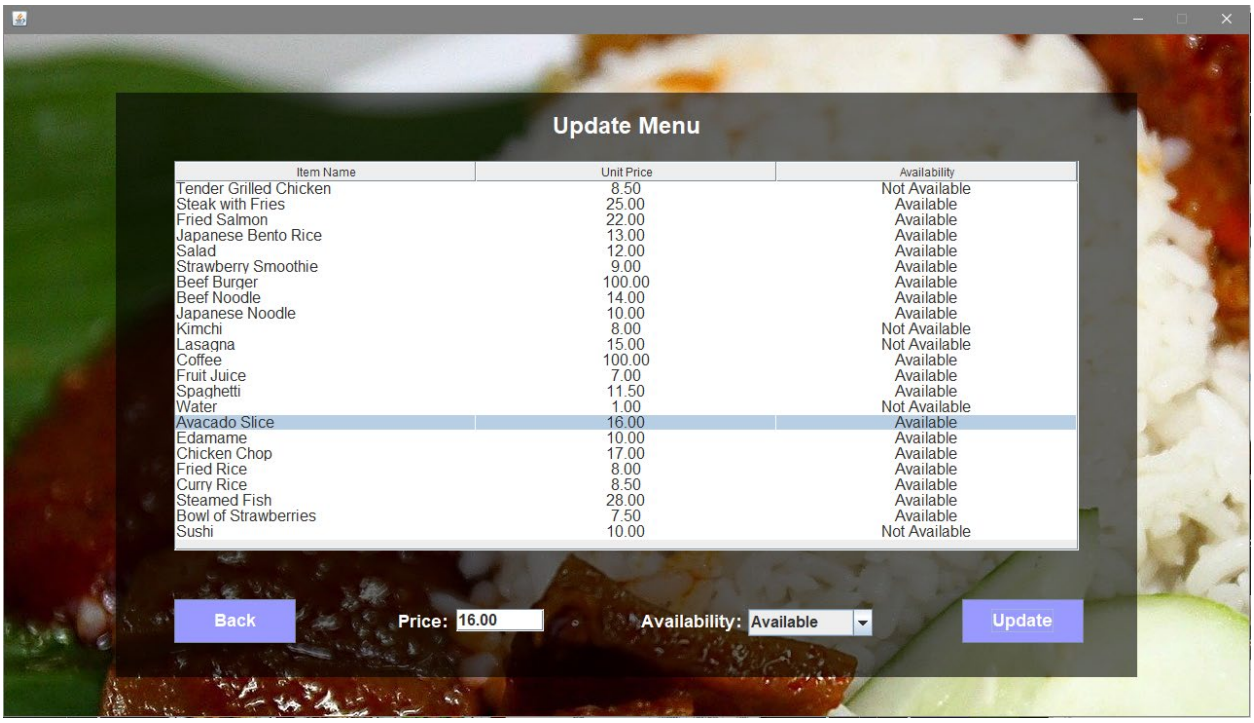
For example, in here, Lasagna is available. We can tell that by looking at the buttons as they are purple.



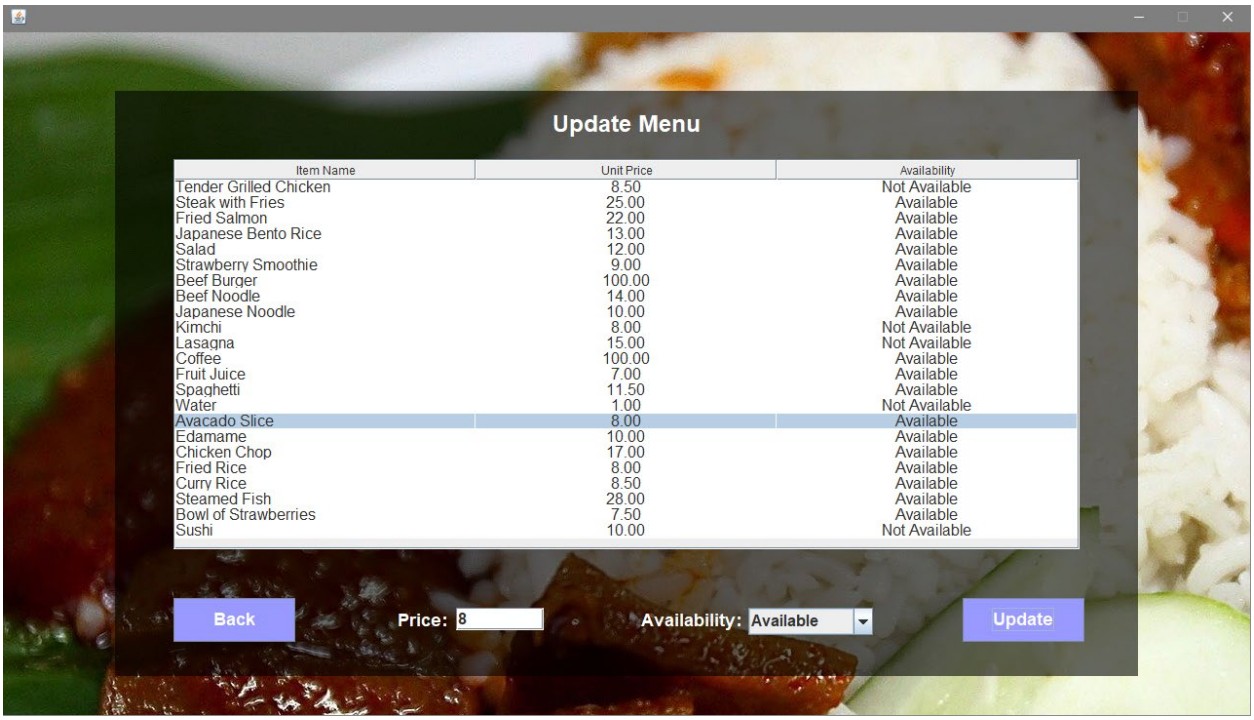
Later the availability status of Lasagna was changed to Not Available.



Because of that, buttons to add or remove Lasagna from cart are disabled.



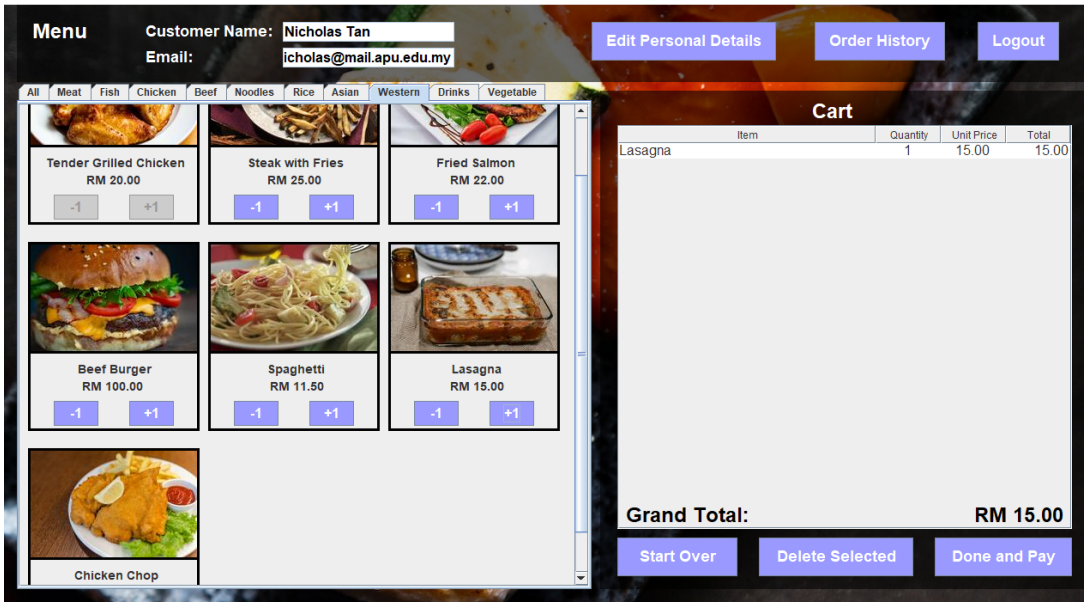
Similarly, for the price, Avocado Slice was selected as an example. The price of the item was 16.00.



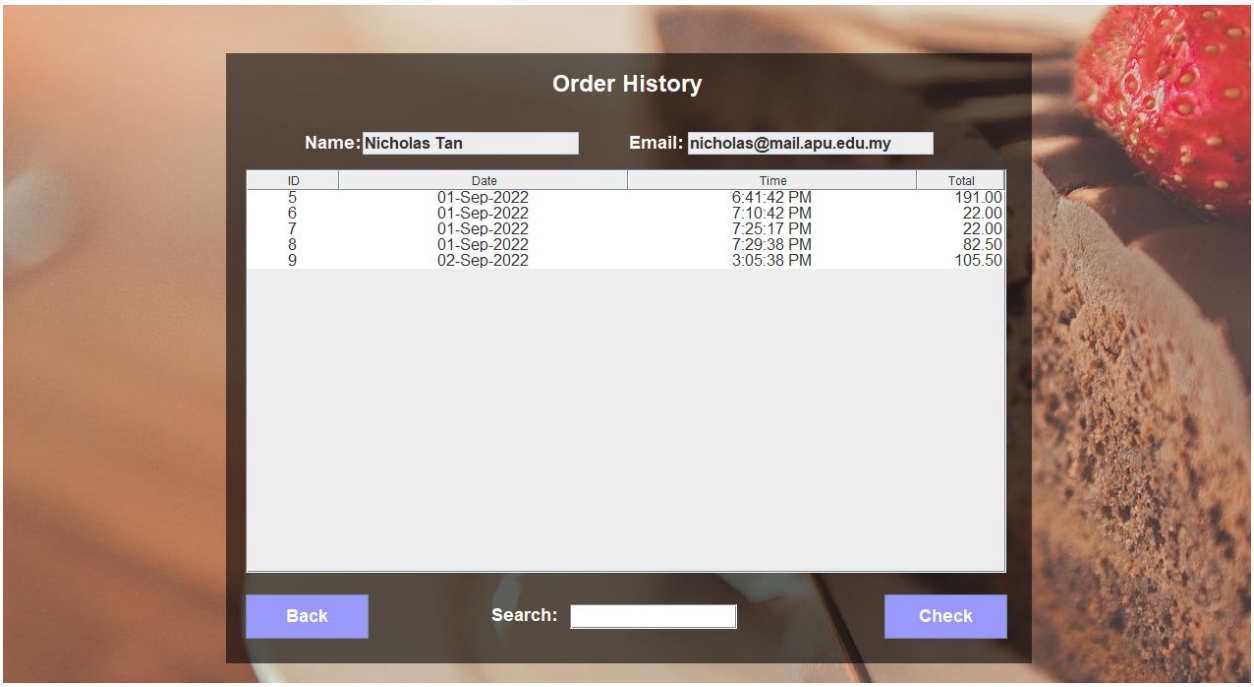
Which was later changed to 8.00 by providing the new price in the price text field and clicking on the button. This change is also happening in the *menu-items.txt* file.

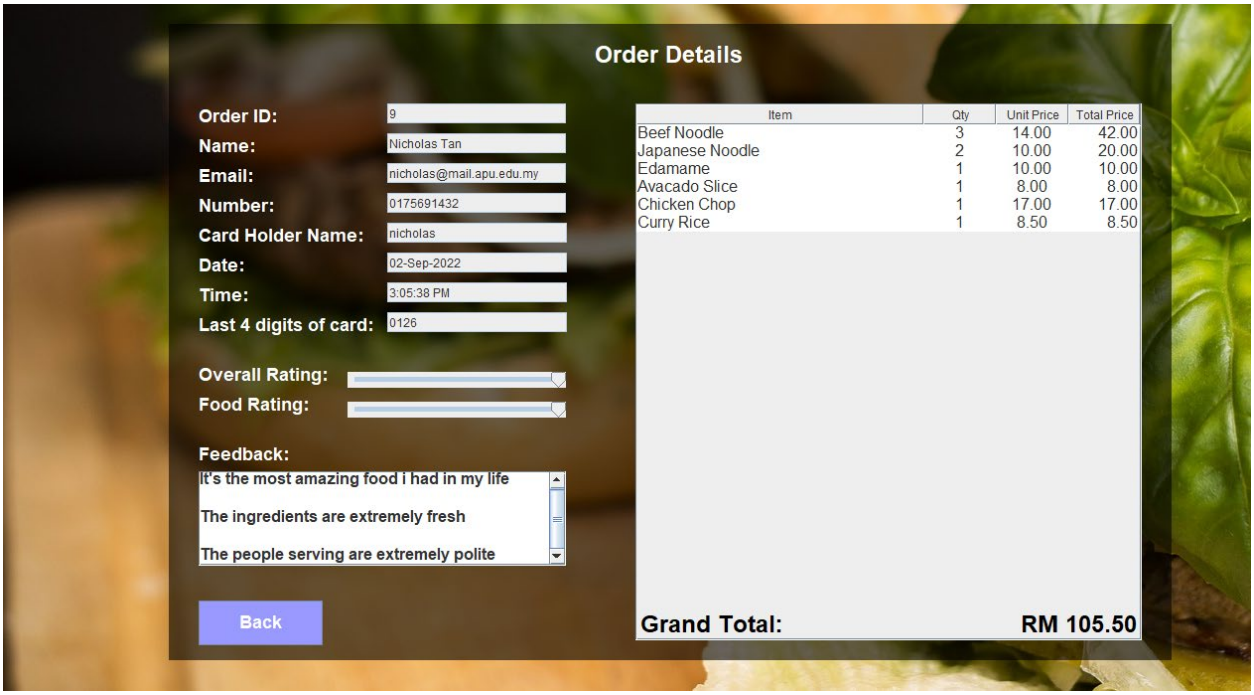
6.0 Special Features

The sample output of the entire program has been described above. Still there are some features that are worth mentioning separately. The visual appeal and user-friendliness of the interactive customer menu is the first and most essential feature. Instead of a text-based menu, we've decided to use an image-based menu. This will attract more customers and allow them to view the image of the item they are ordering. By clicking the button on each item, users can easily add new items and remove existing ones. Additionally, the items are categorized so that customers can order based on their eating habits. As soon as they add an item, it is updated in the cart where they can remove it if needed. The customers can also start over if they change their mind.

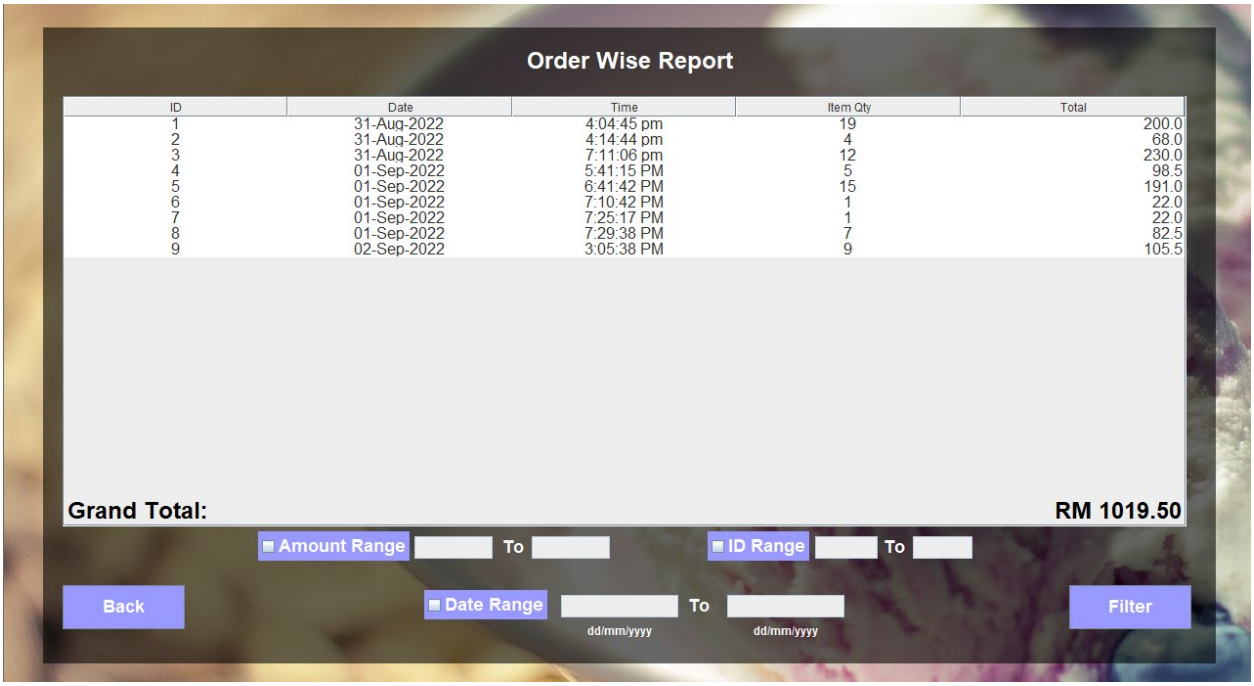


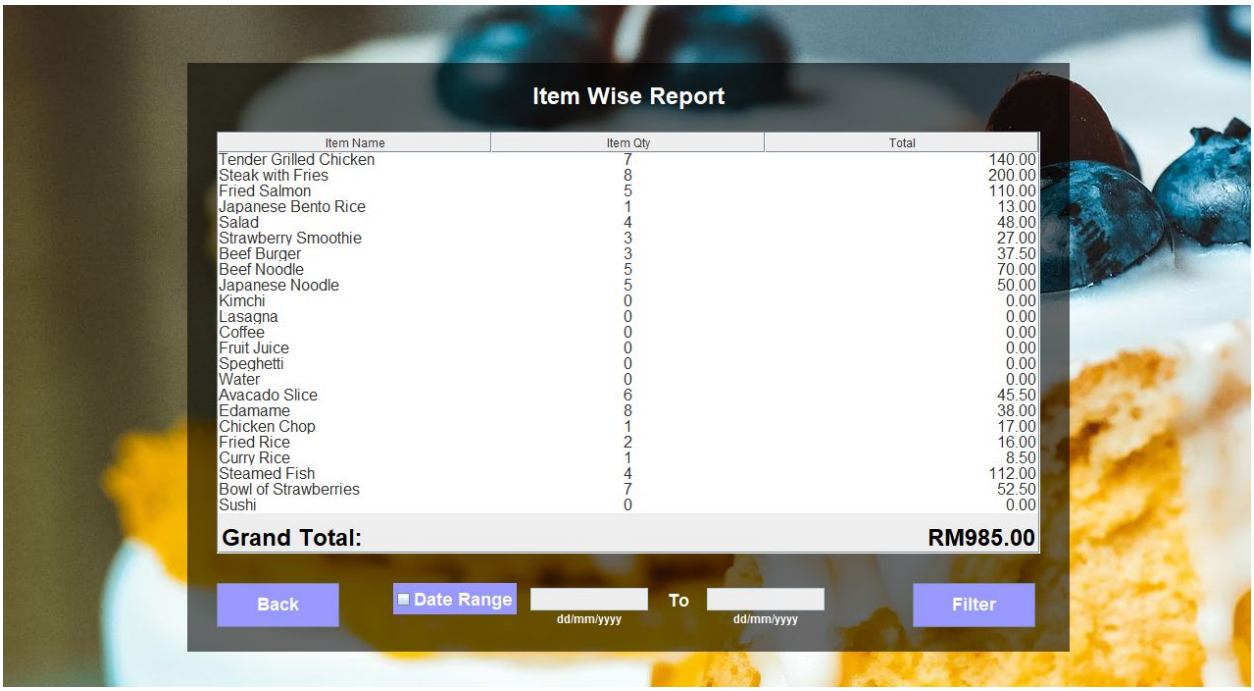
The system allows the customer to view their order history. The order history page is accessible via the customer menu. On the order history page, the customer can view all of their previous orders, including the date, time, quantity, and total amount. Customers can review the details of any order by selecting it and clicking the Check button. This will display every detail regarding the order, including customer feedback and the last four digits of the credit or debit card used to pay for the order.


Order History
Name: Nicholas Tan
Email: nicholas@mail.apu.edu.my
ID Date Time Total
5 01-Sep-2022 6:41:42 PM 191.00
6 01-Sep-2022 7:10:42 PM 22.00
7 01-Sep-2022 7:25:17 PM 22.00
8 01-Sep-2022 7:29:38 PM 82.50
9 02-Sep-2022 3:05:38 PM 105.50
Back
Search:
Check

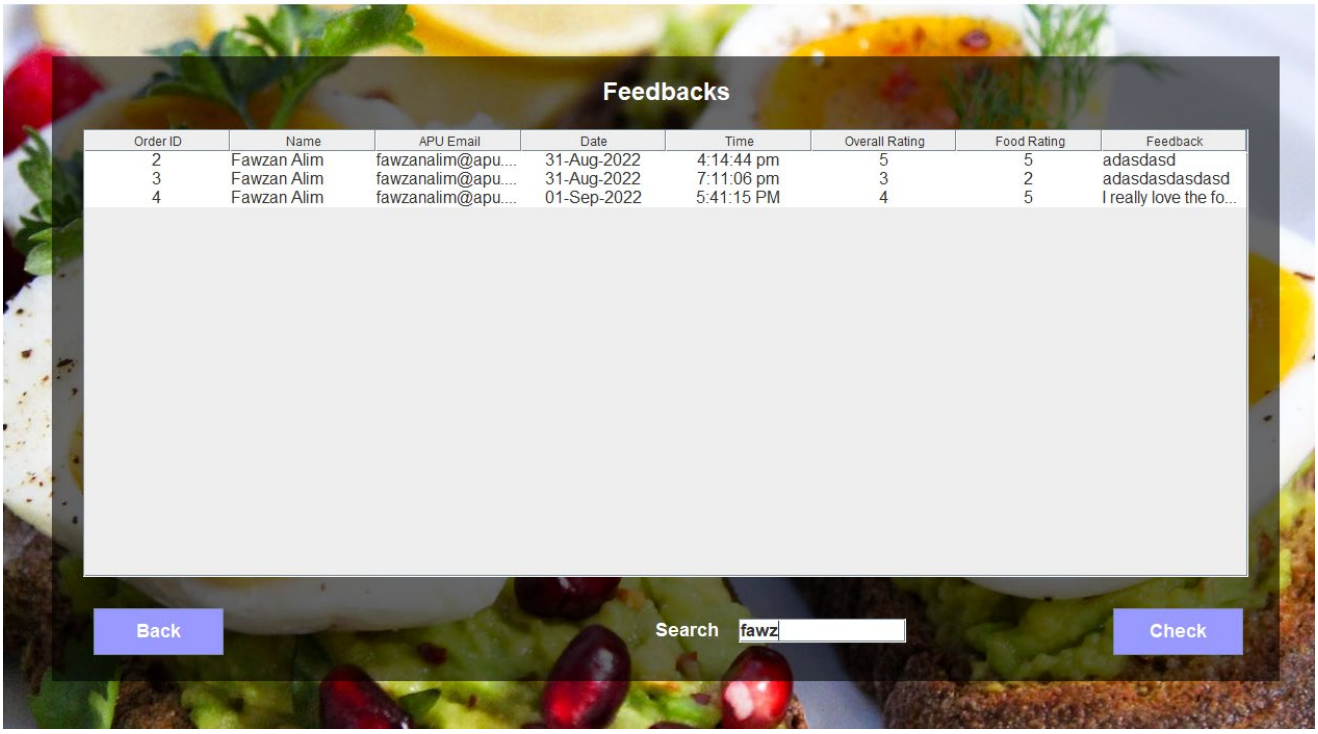

Order Details
Order ID: 9
Name: Nicholas Tan
Email: nicholas@mail.apu.edu.my
Number: 0175691432
Card Holder Name: nicholas
Date: 02-Sep-2022
Time: 3:05:38 PM
Last 4 digits of card: 0126
Overall Rating:
Food Rating:
Feedback:
It's the most amazing food i had in my life
The ingredients are extremely fresh
The people serving are extremely polite
Back
Item Qty Unit Price Total Price
Beef Noodle 3 14.00 42.00
Japanese Noodle 2 10.00 20.00
Edamame 1 10.00 10.00
Avacado Slice 1 8.00 8.00
Chicken Chop 1 17.00 17.00
Curry Rice 1 8.50 8.50
Grand Total: RM 105.50

There are filters available to help managers filter out their reports. The manager will be able to obtain a report of orders completed between specified dates, enabling them to generate a daily, weekly, or monthly report. Using the amount range filter, they can also generate and analyze reports for orders with a high or low total amount. The item wise report is also useful for analyzing the items sold during a given time period.

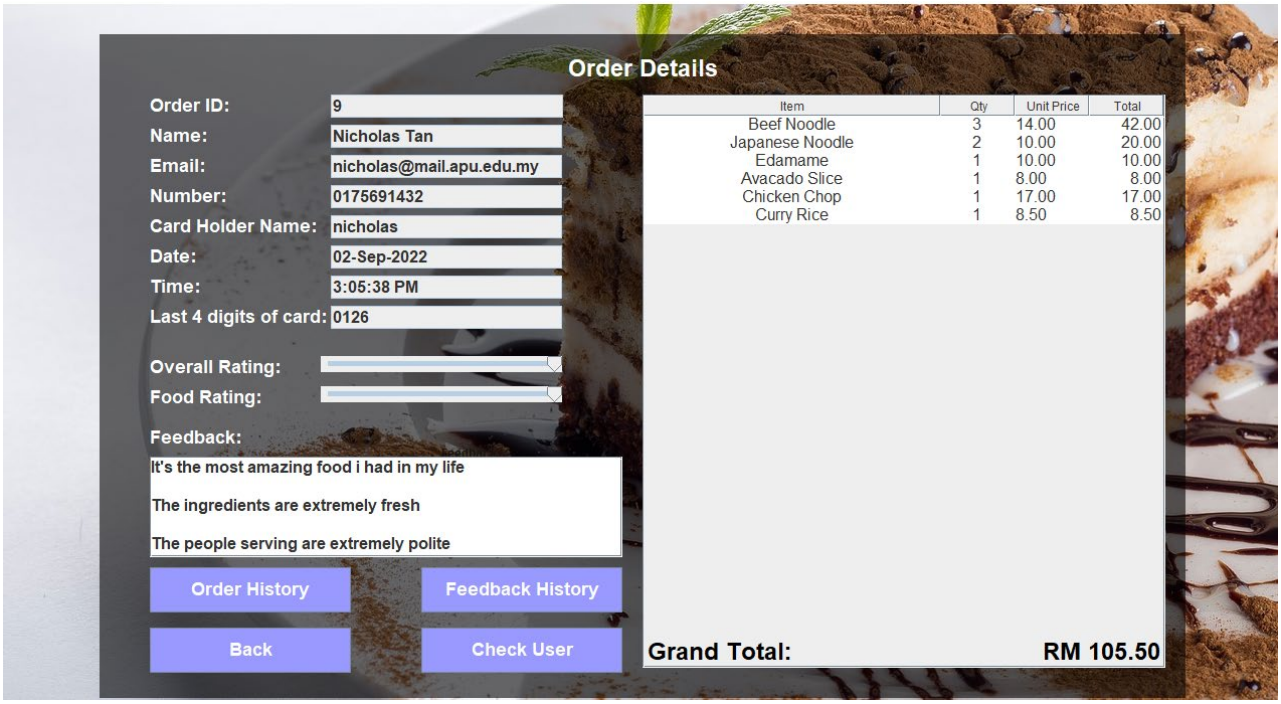




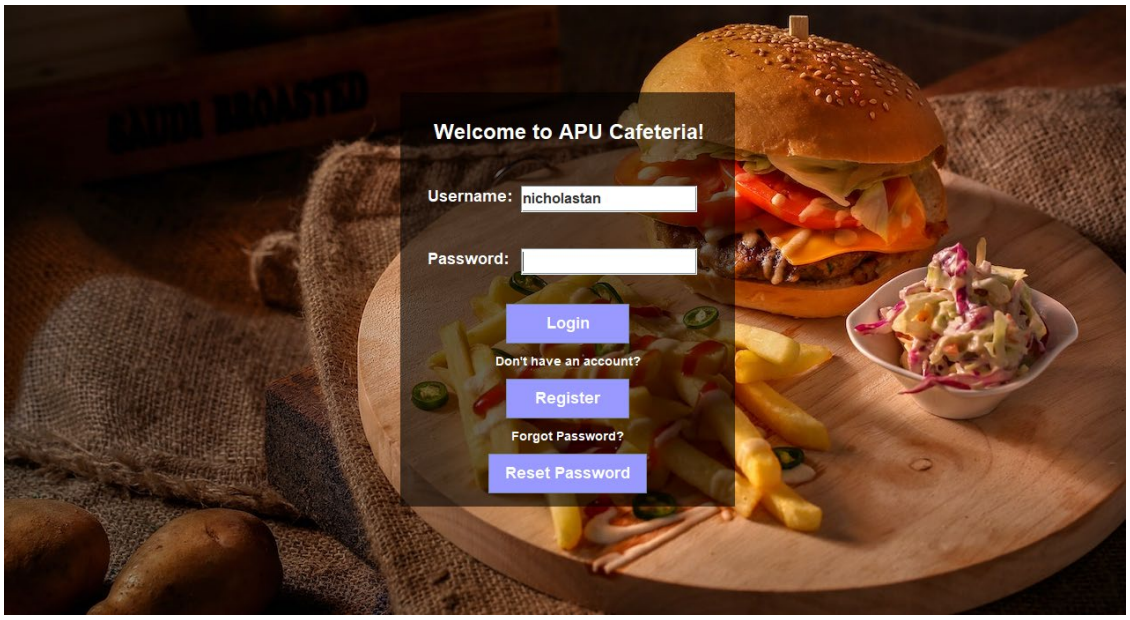
The manager section contains numerous tables. Every day, thousands of students will use the cafeteria for its intended purpose. Scrolling through the entire list of users, orders or feedbacks is a lengthy process. To make it easier for the manager to find something in the table, we have implemented the search functionality on nearly all tables. The search box automatically updates the table without the need to click a button.



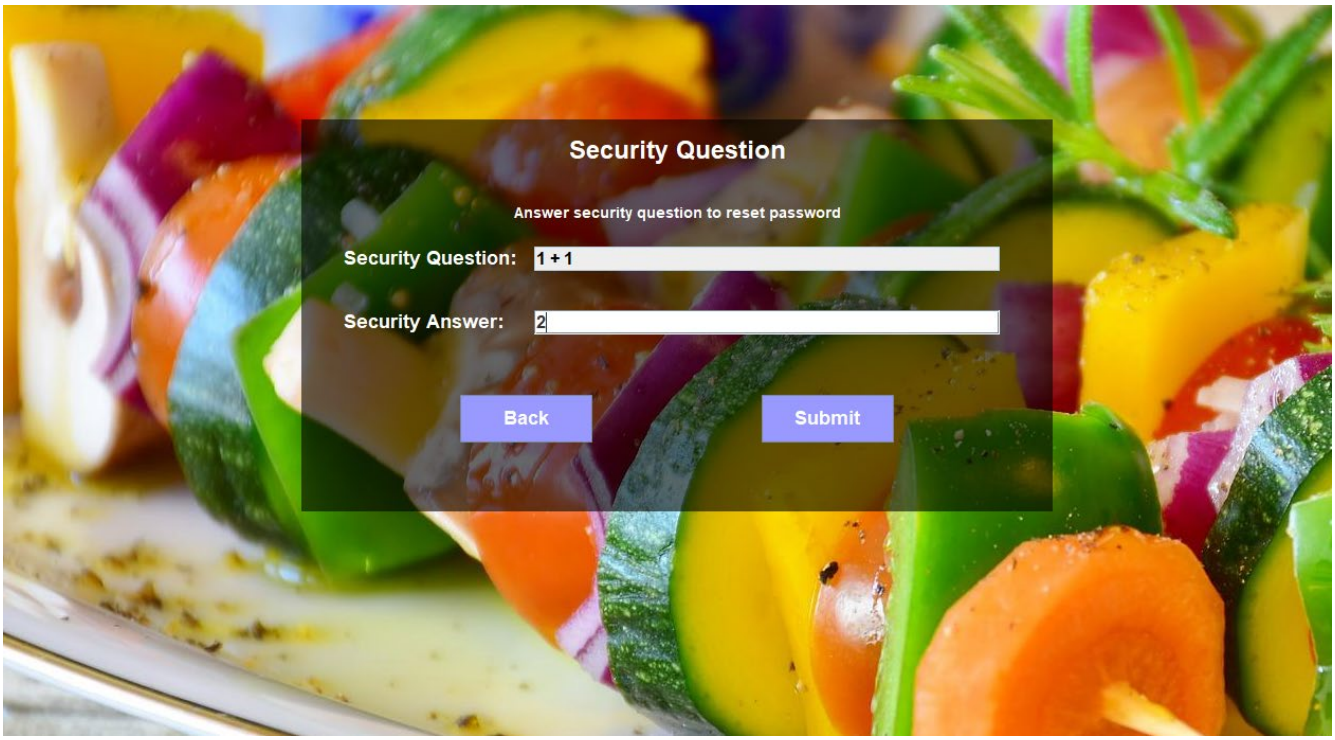
Besides that, manager can check the order history and feedback history of a customer from the details page of that specific customer. The order history button will take the manager to the overall order history page and put the email of the It will do the same for feedback history of the customer by taking the manager to feedback history.customer in the search box.



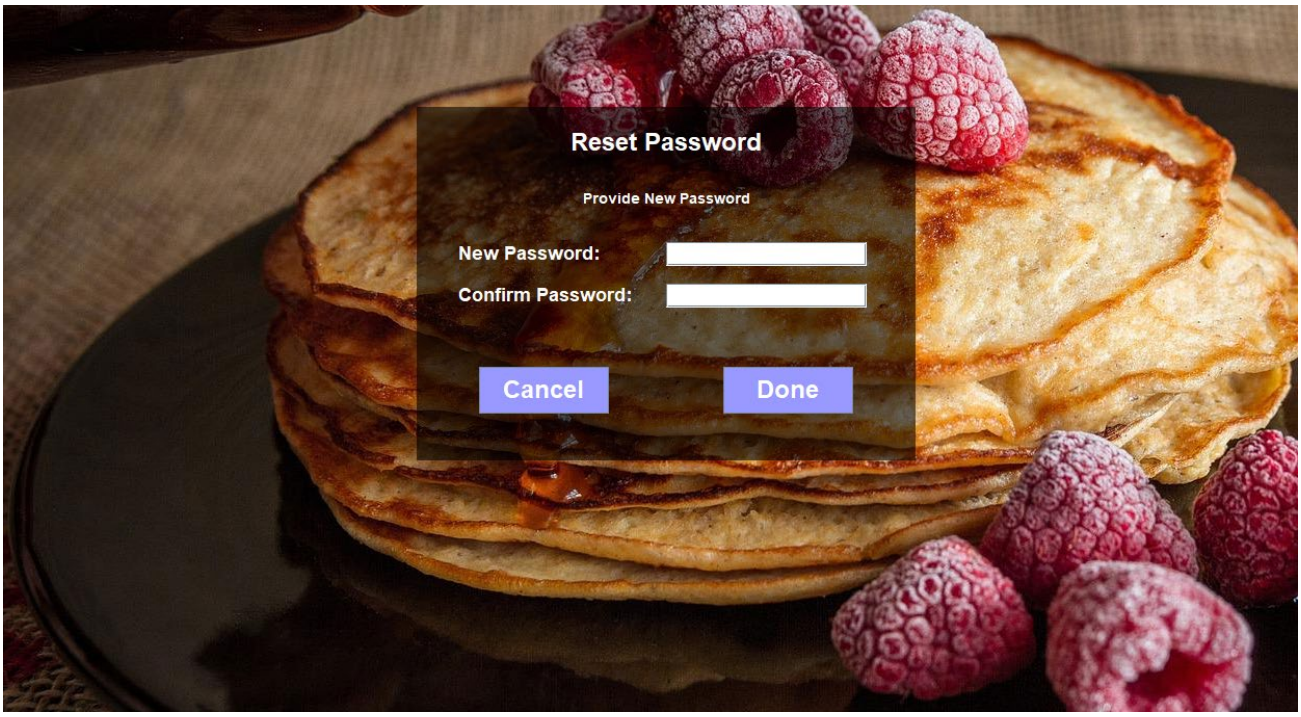
In addition to that, to make the program more user friendly for the customer, the text fields have a unique feature in which if the customer presses the enter button, the cursor jumps to the next text field. This would reduce the number of times the user has to use the mouse to navigate through the interface. It makes it more intuitive for the customer. By pressing the enter, the cursor in the username will be redirected to the password in the example below.



Besides that, the users can set and reset their security question. The security question is a question that the user manually inputs. The security question is paired with a security answer which the user will also set themselves. The security question is used when the user forgets their password but still wants to login into the system, by answering the security question correctly, the user can reset their password. This is a security feature that allows the user to recover their account if they forget their password, but it also prevents strangers from being able to access their account. Since the user can set their own question and password, the user can set the question to something personal to them, which would make it incredibly hard for other users to hack into the account. Thus, we implemented a crucial security feature that gives flexibility to the user in the sense that they do not completely lose their account if they forget their own password.



The customer can set their own security question with their own answer.



If they answer the security question correctly, they can reset their password.

7.0 Conclusion

Objective of this assignment was to develop an APU cafeteria management system using Java OOP. It was challenging to work on initially, but as we progressed, we became more and more comfortable using OOP concepts. The program meets all the requirements of the assignment such as login, signup, ordering, payment, feedbacks, updating menu, storing data in text file, generating reports and so on. On top of that, the program also has some special features which include visually pleasing menu, customer checking their order history, manager filtering report, resetting forgotten customer password using security question and many more.

During the development of the system, we had to overcome a number of obstacles. One of them is updating the file data. Initially, we considered copying the data into a new file and renaming it to replace the current one. That did not unfold as expected. Therefore, we simply read the entire file into a variable, modified it, and then overwrote file. Netbeans is an extremely user-friendly GUI development tool for beginners. However, it does not permit users to modify the code underlying the GUI elements. We wanted to create an array of buttons that would allow us to repeat same thing for all the buttons using a loop. But we were unable to do it. We had to place each button separately, design it, and then add event action for each. It was very time consuming.

Even though it meets all the requirements, no system is perfect. There is still room for improvements. Such as our current system does not allow manager to add new item to the menu. This is happening because we cannot run a loop for images as Netbeans makes it complicated to do so. Another improvement to do so would be to let the manager export the reports as an excel file. Which would help them save it and organize it properly. Overall it is a professional looking system. We are proud to make it and personally would use it if we were in that position.

8.0 References

Programmiz. (n.d.). *Java Ternary Operator*. https://www.programiz.com/java-programming/ternary-operator

Manohar, V. (2022). *New Java 7 Feature: String in Switch Support*. https://dzone.com/articles/new-java-7-feature-string

W3schools. (n.d.). *Java Exceptions- Try…Catch.* https://www.w3schools.com/java/java_try_catch.asp

tutorialspoint. (n.d.). *Java – Constructors*. https://www.tutorialspoint.com/java/java_constructors.htm

GeeksforGeeks. (2022). *Overriding in Java*. https://www.geeksforgeeks.org/overriding-in-java/

GeeksforGeeks. (2022). *Encapsulation in Java.* https://www.geeksforgeeks.org/encapsulation-in-java/

GeeksforGeeks. (2021). *Polymorphism in Java*. https://www.geeksforgeeks.org/polymorphism-in-java/

Oracle. (n.d.). *Class JOptionPane*. https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html

Oracle. (n.d.). *Class DefaultTableCellRenderer.* https://docs.oracle.com/javase/7/docs/api/javax/swing/table/DefaultTableCellRenderer.html

Oracle. (n.d.). *Class JOptionPane*. https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html

Oracle. (n.d.). *Class DefaultTableModel*. https://docs.oracle.com/javase/7/docs/api/javax/swing/table/DefaultTableModel.html

Oracle. (n.d.). *Class BufferedReader*. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/BufferedReader.html

Oracle. (n.d.). *Class File*. https://docs.oracle.com/javase/7/docs/api/java/io/File.html#separator

GeeksforGeeks. (2022). *Classes and Objects in Java*. https://www.geeksforgeeks.org/classes-objects-java/

W3schools. (n.d.). *Java Inheritance*. https://www.w3schools.com/java/java_inheritance.asp

Visual Paradigm. (n.d). *UML Class Diagram Tutorial*. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/