Declaration of Original Work for CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

We have honoured the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course	Lab	Group	Signature/Date
Choo Wei Boon	SC2002	FDAB	2	[illegible] 26/04/2024
Duong Hoang Vu Lam	SC2002	FDAB	2	[illegible] 26/04/2024
Goh Jie Rong, Sean	SC2002	FDAB	2	[illegible] 26/04/2024
Krystal Pek Ke Yun	SC2002	FDAB	2	Krystal 26/04/2024
Wong Kang Yi, Rhyan	SC2002	FDAB	2	[illegible] 26/04/2024

1. Design Considerations

1.1 Design Approach

The Fastfood Ordering and Management System (FOMS) is a Java Console Application that is designed in a way to ensure reusability, extensibility and maintainability so as to achieve loose coupling and high cohesion. We separated the application into different packages with their distinct roles, specifically `'branch'`, `'cart'`, `'item'` , `'order'`, `'payment'`, `'staff'`, `'admin'`, `'application'` and `'system'`. Within each package, it contains classes that carry out respective responsibilities.

1.2 Assumptions made

a. Only one user is able to use the System at one time
b. Removing a branch will mean that all staff, managers and items associated with it will also be removed (a warning will be shown when action is performed)

c. Only be saving data for Item, Branch and Staff, not for other data like orders where no pre-orders can be made

1.3. Design Principles

1.3.1 SOLID design principles

Single Responsibility Principle (SRP)

SRP states that a class should only have one reason to change, where each class only has one specific responsibility rather than being overloaded with multiple tasks. We applied SRP by splitting the system into different classes to handle distinct functionalities related to the system's responsibilities.

Focusing on the '`customerui`' package, we have classes like '`BranchSelectionPage`' responsible for facilitating selection of a specific branch, and others like '`OrderingPage`' , '`MenuPage`' and more . Each of these classes is focused on a single aspect of the customer's interaction with the system, ensuring clarity of responsibility and minimising the risk of unintended coupling between different functionalities.

In addition to this, the use of SRP can be seen in the use of management classes such as '`ItemManagement`' and '`ManagerManagement`’. Each of these classes focus on a single task, with ‘ItemManagement’ having the responsibility of managing items and ‘`ManagerManagement`’ focusing on managing managers, with both classes having get, add and remove functions. ‘`ManagerPage`’ interacts with both of them by delegating the task of managing items and managers to them, where it does not contain any logic or implementation details related to those tasks, hence adhering to SRP.

This design makes it easier to modify individual components without impacting other parts of the system, hence helping in achieving our aim of low coupling and high cohesion with it being maintainable.

Open/Closed Principle (OCP)

OCP states that a module should be open for extension but closed for modification, where we should be able to add new functionality without modifying the existing source code. In our system, we adhered to this principle by making use of both abstraction and inheritance concepts.
For instance, an abstract '`User`' class is created with basic set and get functions for protected attributes like their userId and password where the concept of encapsulation is present, with '`Admin`' and '`Staff`' class extending from the abstract class. The '`Admin`' and '`Staff`' subclasses include more functions that are unique for their role. By doing so, this allows for extension because new user types can be added to

the system by creating additional subclasses without modifying existing code if there were going to be more roles or positions available in the future, hence showing how we applied OCP in our application design and ensuring that our system is extensible and maintainable.

Liskov Substitution Principle (LSP)

LSP states that subclasses should behave in such a way that they do not violate the expected behaviour of the superclass, where application should continue to run when objects of superclass are replaced with objects of its subclasses. In our project, we applied this principle when storing user accounts within a list, where it made use of user hierarchy and the concept of polymorphism.
All staff user accounts, which includes '`Manager`', '`Admin`', and '`Staff`' subclasses, are stored in an ArrayList '`AccountList`' where all instances are upcasted to the abstract superclass '`User`'. This shows polymorphism of user objects. When accessing individual instances from the '`AccountList`', we downcast that instance so that it can access specific subclass functionalities, where even after downcasting, the program continues to run smoothly. This shows how each subclass expects no more and provides no less than superclass, thus maintaining compatibility and consistency throughout the application.

Interface Segregation Principle (ISP)

ISP states that a class should not be forced to make use of interfaces that it does not require, hence encouraging the use of specific interfaces that focus on less functions for classes. This is done by breaking down large interfaces into smaller, more specific interfaces. Doing so helps reduce dependency issues and also helps in ensuring that there is low coupling to ensure the system's maintainability.

Dependency Inversion Principle (DIP)

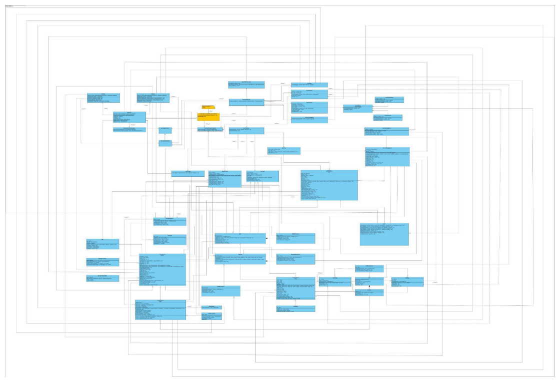
DIP states that high level modules should not depend on low level modules, instead both should depend on abstractions. Abstractions should also not depend on details. This encourages loose coupling and allows the system to be more flexible and maintainable.

1.3.2 Integration of SOLID principle and Object-Oriented concepts

These principles often work together while applying the fundamental Object-Oriented concepts: Abstraction, Encapsulation, Polymorphism and Inheritance as we have explained above. This integration helps ensure the reusability, extensibility and maintainability of our system. By making use of these principles and concepts, it allows our system to run smoothly and also ensure that the system achieves the aim of loose coupling and high cohesion.

2. UML Class Diagram

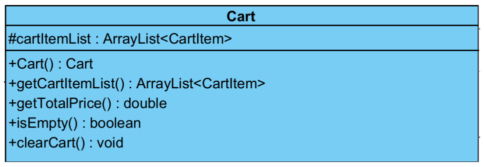
2.1 Building of UML Class Diagram:



Overview of UML Class Diagram

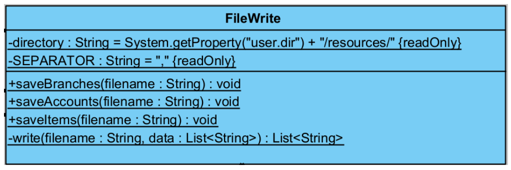
First develop the code (before creating the UML diagram), create the classes (the boxes) and lastly, define the relationship between the classes (the connection). We chose to follow the process described above as we found it to be more flexible and productive, cutting out the need to do several iterations of UML diagrams.

2.2 Creation of Classes in UML Diagram:


Cart
#cartItemList : ArrayList<CartItem>
+Cart() : Cart
+getCartItemList() : ArrayList<CartItem>
+getTotalPrice() : double
+isEmpty() : boolean
+clearCart() : void

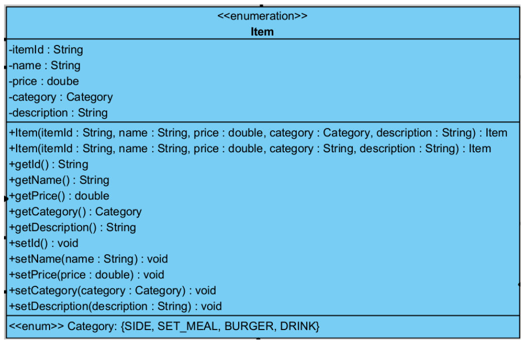
Example of Cart Class

As illustrated in the example, Cart class has the attribute "`cartItemList`" with the access modifier protected and it is of the data type ArrayList<CartItem>. Cart class also has the methods `Cart()`, `getCartItemList()`, `getTotalPrice()`, `isEmpty()`, `clearCart()`. In this class, all the methods are public and they do not have any parameters.


FileWrite
-directory : String = System.getProperty("user.dir") + "/resources/" {readOnly}
-SEPARATOR : String = "," {readOnly}
+saveBranches(filename : String) : void
+saveAccounts(filename : String) : void
+saveItems(filename : String) : void
-write(filename : String, data : List<String>) : List<String>

Example of FileWrite Class

FileWrite is another example of a class used in the UML diagram. From the diagram above, it can be noted that the attributes and methods are all underlined. This is because they are defined with the keyword *static* as it concerns memory management and we only want one instance of the static member that will be shared by other classes. Another thing to note is the {readOnly} at the end of the attributes, these are as such as the attributes are defined with the keyword *final*. As they are meant to specify the file path to the csv files, we do not want it to be manipulated and changed. Thus, setting it with the keyword *final* will help to ensure that.

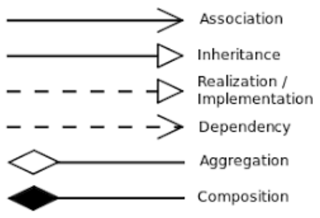


<<enumeration>>
Item
-itemId : String
-name : String
-price : doube
-category : Category
-description : String
+Item(itemId : String, name : String, price : double, category : Category, description : String) : Item
+Item(itemId : String, name : String, price : double, category : String, description : String) : Item
+getId() : String
+getName() : String
+getPrice() : double
+getCategory() : Category
+getDescription() : String
+setId() : void
+setName(name : String) : void
+setPrice(price : double) : void
+setCategory(category : Category) : void
+setDescription(description : String) : void
<<enum>> Category: {SIDE, SET_MEAL, BURGER, DRINK}

Example of Item Class

Item class has the special class type enumeration, the enumeration literals are defined at the bottom of the class. Enumerations are named constants that programmers mainly use to create their own pre-defined data type.

2.3 Types of Relations Used in UML Diagram:

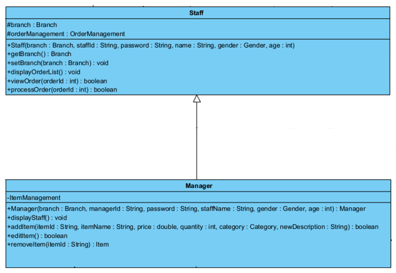


Example for the type of relations

Ordering from the strongest to weakest relation:

Inheritance → Implementation → Composition → Aggregation → Association → Dependency

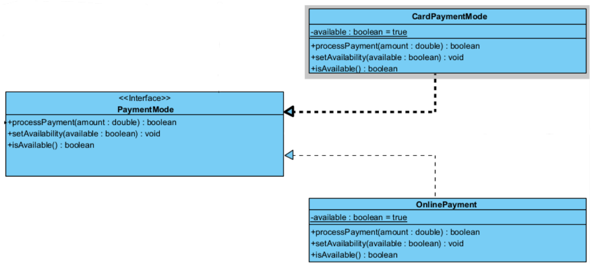
These are the relations used to represent how one class relates to another in the UML diagram.



Staff
#branch : Branch
#orderManagement : OrderManagement
+Staff(branch : Branch, staffId : String, password : String, name : String, gender : Gender, age : int)
+getBranch() : Branch
+setBranch(branch : Branch) : void
+displayOrderList() : void
+viewOrder(orderId : int) : boolean
+processOrder(orderId : int) : boolean
Manager
-ItemManagement
+Manager(branch : Branch, managerId : String, password : String, staffName : String, gender : Gender, age : int) : Manager
+displayStaff() : void
+addItem(itemId : String, itemName : String, price : double, quantity : int, category : Category, newDescription : String) : boolean
+editItem() : boolean
+removeItem(itemId : String) : Item

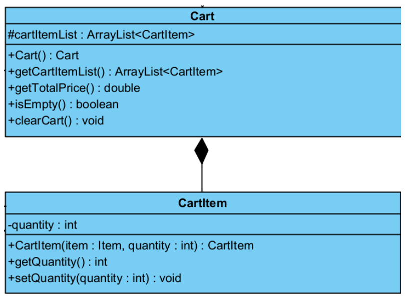
Example of Inheritance relation

According to the assignment pdf, Manager actions included all that the Staff is able to do. Therefore, the keyword *extends* is used to create an inheritance relation between Manager and Staff whereby Staff is the base class while Manager is the derived class.


CardPaymentMode
-available : boolean = true
+processPayment(amount : double) : boolean
+setAvailability(available : boolean) : void
+isAvailable() : boolean
<<Interface>>
PaymentMode
+processPayment(amount : double) : boolean
+setAvailability(available : boolean) : void
+isAvailable() : boolean
OnlinePayment
-available : boolean = true
+processPayment(amount : double) : boolean
+setAvailability(available : boolean) : void
+isAvailable() : boolean

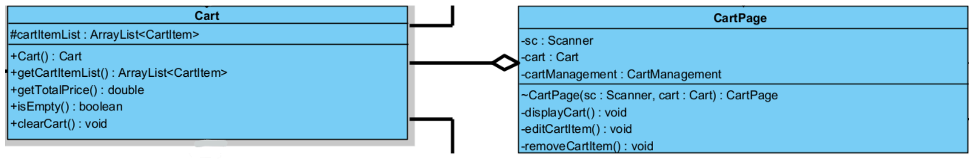
Example of Implementation relation

Since the PaymentMode class is defined as an Interface class, CardPaymentMode and OnlinePaymentMode are the classes to be implemented. More specifically, the methods are implemented from those declared in the Interface class.


Cart
#cartItemList : ArrayList<CartItem>
+Cart() : Cart
+getCartItemList() : ArrayList<CartItem>
+getTotalPrice() : double
+isEmpty() : boolean
+clearCart() : void
CartItem
-quantity : int
+CartItem(item : Item, quantity : int) : CartItem
+getQuantity() : int
+setQuantity(quantity : int) : void

Example of Composition relation

Composition relation is a subset of Association relation and it suggests a stronger relationship than Aggregation. It implies the whole and part relationship between Cart and CartItem, however, CartItem cannot exist without Cart. This idea agrees when viewed conceptually, in which that CartItem will have no meaning if it is not within a Cart (a CartItem will just be an Item without Cart).


Cart
#cartItemList : ArrayList<CartItem>
+Cart() : Cart
+getCartItemList() : ArrayList<CartItem>
+getTotalPrice() : double
+isEmpty() : boolean
+clearCart() : void
CartPage
-sc : Scanner
-cart : Cart
-cartManagement : CartManagement
~CartPage(sc : Scanner, cart : Cart) : CartPage
-displayCart() : void
-editCartItem() : void
-removeCartItem() : void

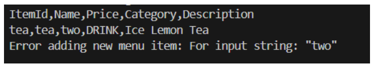
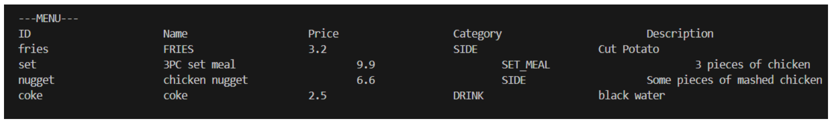
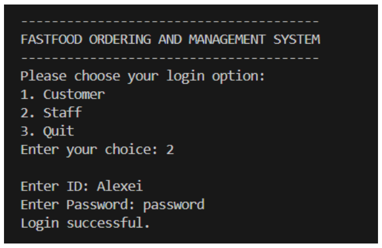
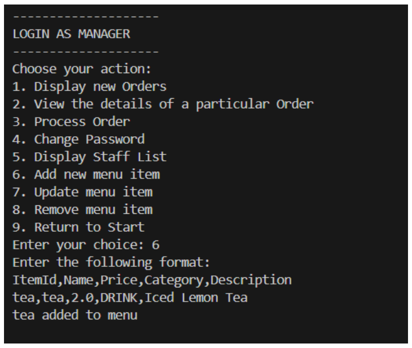
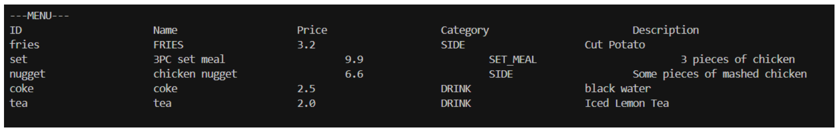
Example of Aggregation relation

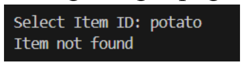
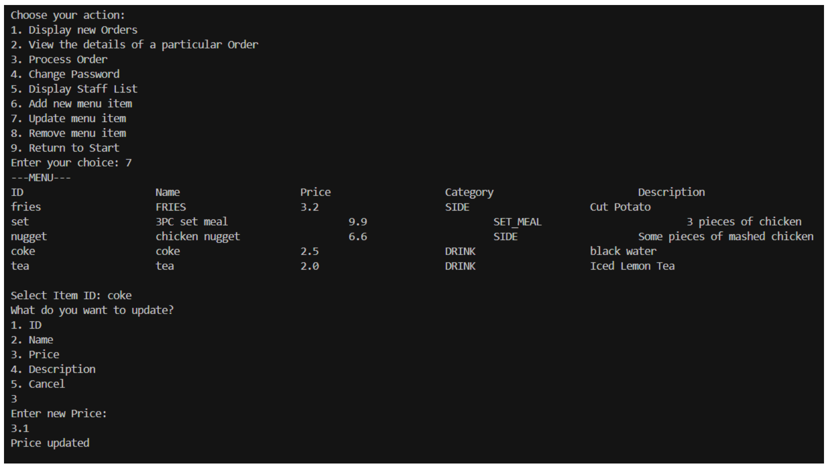
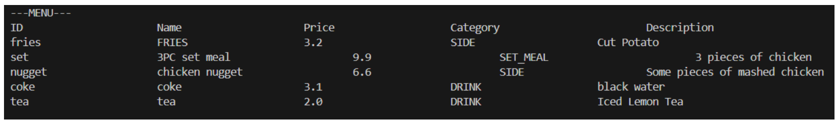
Cart and CartPage are related using Aggregation. Aggregation is also a subset of the Association relation, it represents the whole-part relationship, however the whole and part can exist independently. This aligns with logical understanding, as CartPage is a feature to interact with Cart however, Cart does exist for other purposes besides that of CartPage hence they can be separated.

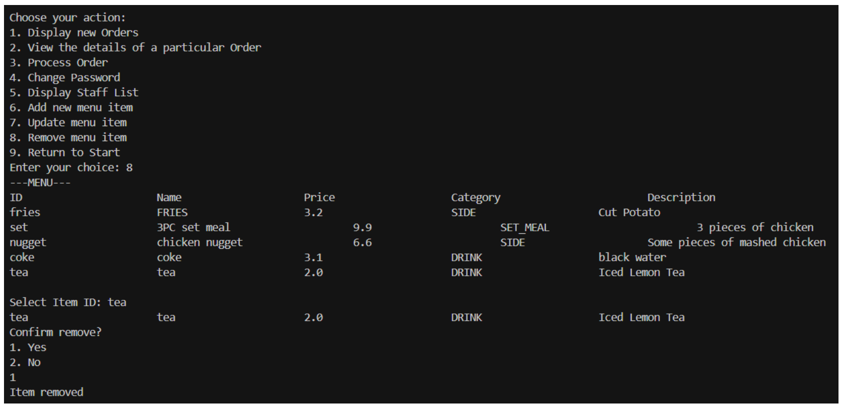
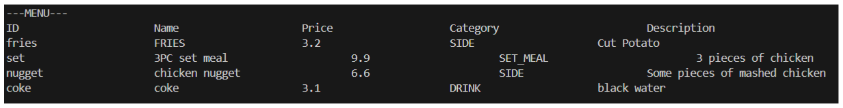
3. Test Case Demonstration

3.1 Manager's action: Menu Management:

Test Case	Expected Outcome	Result

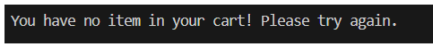
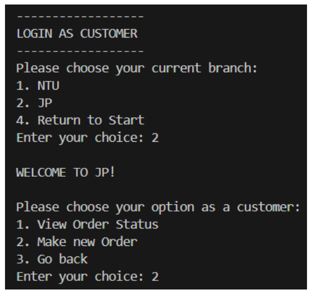
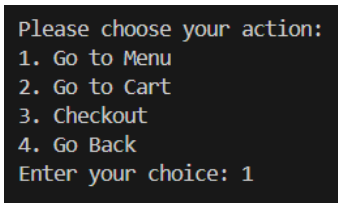
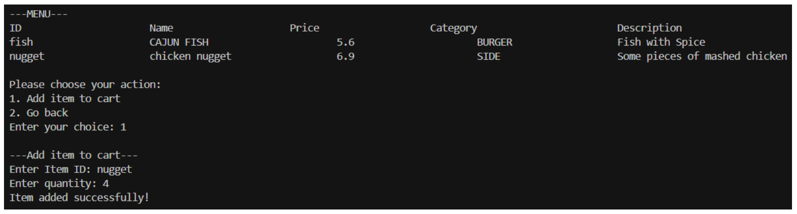
1. Add a new menu item with a unique name, price, description, and category.
• Verify that the menu item is successfully added.
	The manager can choose the option "6. Add new menu item" and key in required details for new item to be added.
	
	[Exception Handling]
	Invalid format of details entered like an invalid category or a non-digit price, it will throw and catch the error and print the error message
	
	`ItemId,Name,Price,Category,Description`
	`tea,tea,two,DRINK,Ice Lemon Tea`
	`Error adding new menu item: For input string: "two"`
	[Error Handling, Case 19]
	If the same ID is used for new item added, the menu will not be updated where the original item with the ID will not be changed and the new item is not added.
		
		`---MENU---`
		`ID  Name  Price  Category  Description`
		`fries  FRIES  3.2  SIDE  Cut Potato`
		`set  3PC set meal  9.9  SET_MEAL  3 pieces of chicken`
		`nugget  chicken nugget  6.6  SIDE  Some pieces of mashed chicken`
		`coke  coke  2.5  DRINK  black water`
		[Original Menu with 4 items]
		
		`-------------------------------------`
		`FASTFOOD ORDERING AND MANAGEMENT SYSTEM`
		`-------------------------------------`
		`Please choose your login option:`
		`1. Customer`
		`2. Staff`
		`3. Quit`
		`Enter your choice: 2`
		
		`Enter ID: Alexei`
		`Enter Password: password`
		`Login successful.`
		
		`-------------------`
		`LOGIN AS MANAGER`
		`-------------------`
		`Choose your action:`
		`1. Display new Orders`
		`2. View the details of a particular Order`
		`3. Process Order`
		`4. Change Password`
		`5. Display Staff List`
		`6. Add new menu item`
		`7. Update menu item`
		`8. Remove menu item`
		`9. Return to Start`
		`Enter your choice: 6`
		`Enter the following format:`
		`ItemId,Name,Price,Category,Description`
		`tea,tea,2.0,DRINK,Iced Lemon Tea`
		`tea added to menu`
		[Manager Login and adding new item process]
		
		`---MENU---`
		`ID  Name  Price  Category  Description`
		`fries  FRIES  3.2  SIDE  Cut Potato`
		`set  3PC set meal  9.9  SET_MEAL  3 pieces of chicken`
		`nugget  chicken nugget  6.6  SIDE  Some pieces of mashed chicken`
		`coke  coke  2.5  DRINK  black water`
		`tea  tea  2.0  DRINK  Iced Lemon Tea`
		[Updated Menu with 5 items]

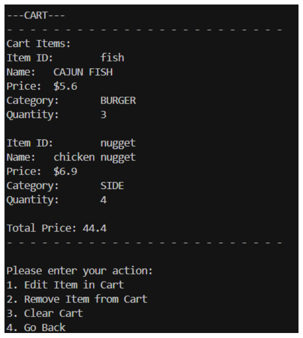
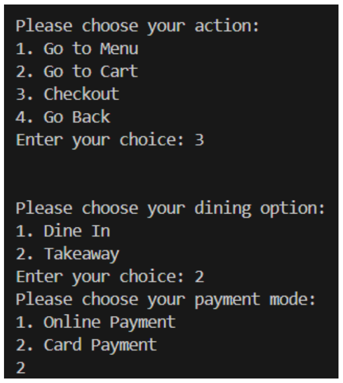
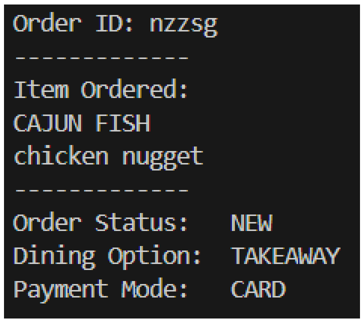
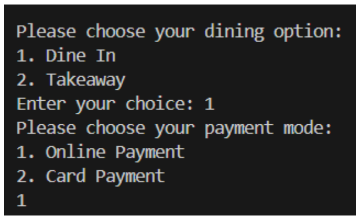
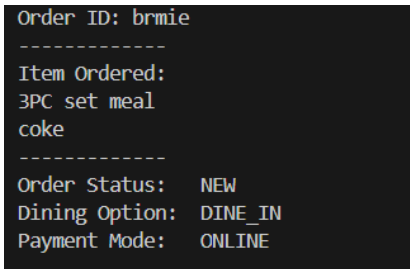
2. Update the price and description of an existing menu item. • Verify that the changes are reflected in the menu	The manager can choose the option "7. Update Menu Item" and choose the ID of the item they want to modify, followed by the attribute that needs to be updated.
	
	[Exception Handling]
	If incorrect ID is entered, it will return an error and go back to the Manager login page
	
	`Select Item ID: potato`
	`Item not found`
	
	[Backtracking]
	Managers can choose to go back out of the "Update Menu item" by choosing the cancel option available.
		
		`Choose your action:`
		`1. Display new Orders`
		`2. View the details of a particular Order`
		`3. Process Order`
		`4. Change Password`
		`5. Display Staff List`
		`6. Add new menu item`
		`7. Update menu item`
		`8. Remove menu item`
		`9. Return to Start`
		`Enter your choice: 7`
		`---MENU---`
		`ID  Name  Price  Category  Description`
		`fries  FRIES  3.2  SIDE  Cut Potato`
		`set  3PC set meal  9.9  SET_MEAL  3 pieces of chicken`
		`nugget  chicken nugget  6.6  SIDE  Some pieces of mashed chicken`
		`coke  coke  2.5  DRINK  black water`
		`tea  tea  2.0  DRINK  Iced Lemon Tea`
		
		`Select Item ID: coke`
		`What do you want to update?`
		`1. ID`
		`2. Name`
		`3. Price`
		`4. Description`
		`5. Cancel`
		`3`
		`Enter new Price:`
		`3.1`
		`Price updated`
		[Original menu with coke's price being 2.5, changing Item's price process by accessing item using itemID]
		
		`---MENU---`
		`ID  Name  Price  Category  Description`
		`fries  FRIES  3.2  SIDE  Cut Potato`
		`set  3PC set meal  9.9  SET_MEAL  3 pieces of chicken`
		`nugget  chicken nugget  6.6  SIDE  Some pieces of mashed chicken`
		`coke  coke  3.1  DRINK  black water`
		`tea  tea  2.0  DRINK  Iced Lemon Tea`
		[Updated menu, with coke's price now at 3.1]

3. Remove an existing menu item.
• Verify that the menu item is no longer available
	Manager can choose “8. Remove menu item” and enter item ID of item to be removed, where there will be a confirmation question before the item is removed.
	
	[Backtracking]
	If manager did not mean to type the itemID, they can choose “No” during the confirmation and item will not be removed
		
		`Choose your action:`
		`1. Display new Orders`
		`2. View the details of a particular Order`
		`3. Process Order`
		`4. Change Password`
		`5. Display Staff List`
		`6. Add new menu item`
		`7. Update menu item`
		`8. Remove menu item`
		`9. Return to Start`
		`Enter your choice: 8`
		`---MENU---`
		`ID          Name            Price          Category          Description`
		`fries       FRIES           3.2            SIDE              Cut Potato`
		`set         3PC set meal         9.9            SET_MEAL              3 pieces of chicken`
		`nugget      chicken nugget       6.6            SIDE              Some pieces of mashed chicken`
		`coke        coke            3.1            DRINK             black water`
		`tea         tea             2.0            DRINK             Iced Lemon Tea`
		
		`Select Item ID: tea`
		`tea         tea             2.0            DRINK             Iced Lemon Tea`
		`Confirm remove?`
		`1. Yes`
		`2. No`
		`1`
		`Item removed`
		[Original menu with 5 items shown, Item removal process with confirmation]
		
		`---MENU---`
		`ID          Name            Price          Category          Description`
		`fries       FRIES           3.2            SIDE              Cut Potato`
		`set         3PC set meal         9.9            SET_MEAL              3 pieces of chicken`
		`nugget      chicken nugget       6.6            SIDE              Some pieces of mashed chicken`
		`coke        coke            3.1            DRINK             black water`
		[Updated menu with 4 items]

3.2 Order Processing:

Test Case	Expected Outcome	Result

4. Place new order with multiple food items, customise some items, choose takeaway option
• Verify that order is created successfully
	Customers will first choose the branch, then select “2. Make new Order” and go to menu. They can then add an item to cart by item ID. Repeat steps to add another item into their cart.
	Once done, go to Cart and choose dining option “Takeaway” and payment mode. A receipt will be printed with Order ID.
	
	[Error Handling, Case 20]
	Return message when processing order with no item added to cart.
	
	`You have no item in your cart! Please try again.`
	
	[Backtracking]
	Customers can choose to backtrack and return to previous pages if they decide not to buy anything.
		
		`-----------------`
		`LOGIN AS CUSTOMER`
		`-----------------`
		`Please choose your current branch:`
		`1. NTU`
		`2. JP`
		`4. Return to Start`
		`Enter your choice: 2`
		
		`WELCOME TO JP!`
		
		`Please choose your option as a customer:`
		`1. View Order Status`
		`2. Make new Order`
		`3. Go back`
		`Enter your choice: 2`
		
		`Please choose your action:`
		`1. Go to Menu`
		`2. Go to Cart`
		`3. Checkout`
		`4. Go Back`
		`Enter your choice: 1`
		[Customer Page and going to Menu]
		
		`---MENU---`
		`ID          Name            Price          Category          Description`
		`fish        CAJUN FISH           5.6            BURGER            Fish with Spice`
		`nugget      chicken nugget       6.9            SIDE              Some pieces of mashed chicken`
		
		`Please choose your action:`
		`1. Add item to cart`
		`2. Go back`
		`Enter your choice: 1`
		
		`---Add item to cart---`
		`Enter Item ID: nugget`
		`Enter quantity: 4`
		`Item added successfully!`
		[Adding item into cart, repeat to add other menu items]

	They can also choose to “Go to Cart” for adjustments to cart. 	  **[Checkout and choose dining and payment option, with order receipt printed]**
5. Place a new order with dine-in option. • Verify that the order is created with the correct preferences	**Same as Case 4 but with ‘dine-in option’** **[Backtracking]** To add on, for the “Go to Cart” it prints out the current items added to cart and total price, where they can also remove items by their ID, or edit items in cart, where they can change the quantity of a specific item they included in cart previously.	**Same process as Case 4 (but in this case we used a different branch)**   **[Order receipt printed with different Order ID]**

```
---CART---
- - - - - - - - - - - - - - - - - - - - - - - -
Cart Items:
Item ID:        fish
Name:   CAJUN FISH
Price:  $5.6
Category:       BURGER
Quantity:       3

Item ID:        nugget
Name:   chicken nugget
Price:  $6.9
Category:       SIDE
Quantity:       4

Total Price: 44.4
- - - - - - - - - - - - - - - - - - - - - - - -

Please enter your action:
1. Edit Item in Cart
2. Remove Item from Cart
3. Clear Cart
4. Go Back
```

```
Please choose your action:
1. Go to Menu
2. Go to Cart
3. Checkout
4. Go Back
Enter your choice: 3


Please choose your dining option:
1. Dine In
2. Takeaway
Enter your choice: 2
Please choose your payment mode:
1. Online Payment
2. Card Payment
2
```

```
Order ID: nzzsg
-------------
Item Ordered:
CAJUN FISH
chicken nugget
-------------
Order Status:   NEW
Dining Option:  TAKEAWAY
Payment Mode:   CARD
```

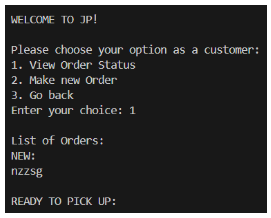
```
Please choose your dining option:
1. Dine In
2. Takeaway
Enter your choice: 1
Please choose your payment mode:
1. Online Payment
2. Card Payment
1
```

```
Order ID: brmie
------------
Item Ordered:
3PC set meal
coke
------------
Order Status:   NEW
Dining Option:  DINE_IN
Payment Mode:   ONLINE
```

3.3 Payment Integration:

Test Case	**Expected Outcome**	**Result**
6. Pay using credit/debit card.	**Same as Case 4**	**Same as Case 4 where Card Payment is selected**
7. Pay using online payment.	**Same as Case 5**	**Same as Case 5 where Online Payment is selected**

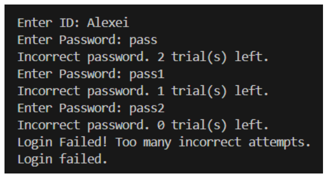
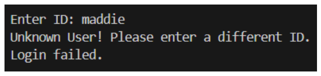
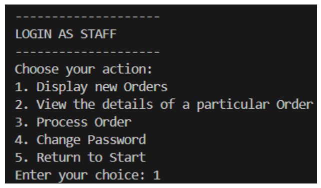
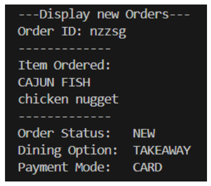
3.4 Order Tracking:

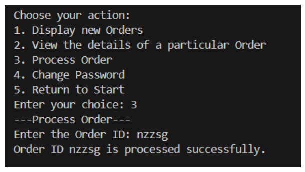
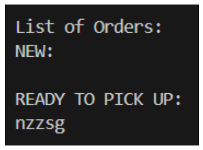
Test Case	**Expected Outcome**	**Result**
8. Track the status of an existing order using the order ID. • Verify that the correct status is displayed.	By selecting the branch, for example JP, and choose “1. View Order Status”, it will show the list of orders for that branch, where order IDs are categorised in “New” where orders are not processed and “Ready to pick up” when customers can pick up their orders.	

```
WELCOME TO JP!

Please choose your option as a customer:
1. View Order Status
2. Make new Order
3. Go back
Enter your choice: 1

List of Orders:
NEW:
nzzsg

READY TO PICK UP:
```

3.5 Staff Actions:

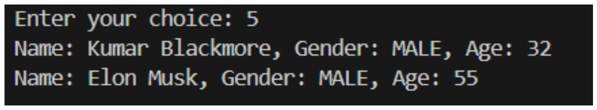
Test Case	Expected Outcome	Result

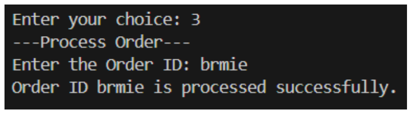
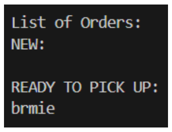
9. Login as a staff member and display new orders	Staff members can choose “3. Display new Orders” to view the details of any orders where the orderID and receipt is shown to the staff.
	
	[Error Handling, Case 24]
	
	`Enter ID: Alexei`
	`Enter Password: pass`
	`Incorrect password. 2 trial(s) left.`
	`Enter Password: pass1`
	`Incorrect password. 1 trial(s) left.`
	`Enter Password: pass2`
	`Incorrect password. 0 trial(s) left.`
	`Login Failed! Too many incorrect attempts.`
	`Login failed.`
	
	`Enter ID: maddie`
	`Unknown User! Please enter a different ID.`
	`Login failed.`
	If password is wrong or Id unknown, gives error message
		
		`-------------------`
		`LOGIN AS STAFF`
		`-------------------`
		`Choose your action:`
		`1. Display new Orders`
		`2. View the details of a particular Order`
		`3. Process Order`
		`4. Change Password`
		`5. Return to Start`
		`Enter your choice: 1`
		
		`---Display new Orders---`
		`Order ID: nzzsg`
		`------------`
		`Item Ordered:`
		`CAJUN FISH`
		`chicken nugget`
		`------------`
		`Order Status:   NEW`
		`Dining Option:  TAKEAWAY`
		`Payment Mode:   CARD`

10. Process a new order, updating its status to "Ready to pickup."	Staff members can choose “4. Process Order” and enter the order ID so that it is processed successfully.
	The Order Tracking process for customers will also be updated, where for branch JP, the status of order nzzsg is updated.
		
		`Choose your action:`
		`1. Display new Orders`
		`2. View the details of a particular Order`
		`3. Process Order`
		`4. Change Password`
		`5. Return to Start`
		`Enter your choice: 3`
		`---Process Order---`
		`Enter the Order ID: nzzsg`
		`Order ID nzzsg is processed successfully.`
		
		`List of Orders:`
		`NEW:`
		
		`READY TO PICK UP:`
		`nzzsg`

3.6 Manager Actions:

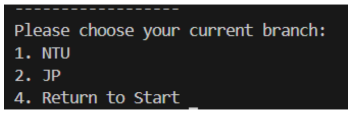
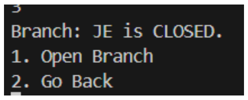
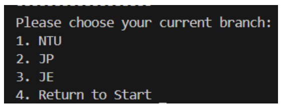
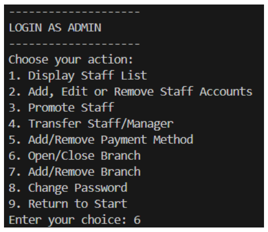
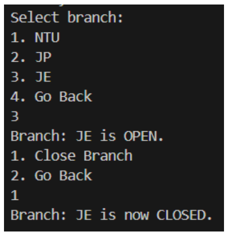
Test Case	Expected Outcome	Result

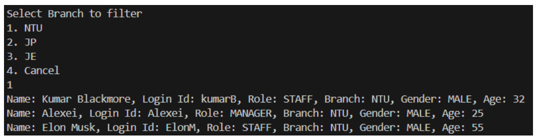
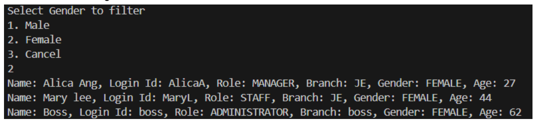
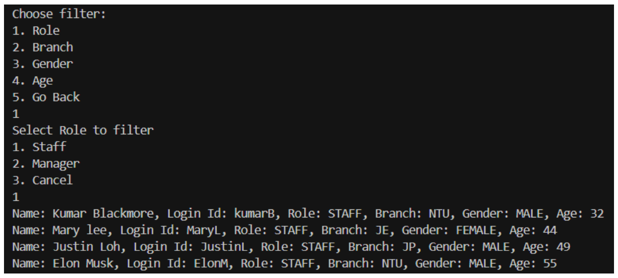
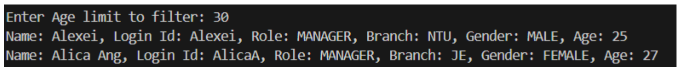
11. Display the staff list in the manager’s branch	Manager can choose “5. Display Staff List” as seen in actions available in Case 1. This case shows staff from NTU like manager.	
		`Enter your choice: 5`
		`Name: Kumar Blackmore, Gender: MALE, Age: 32`
		`Name: Elon Musk, Gender: MALE, Age: 55`

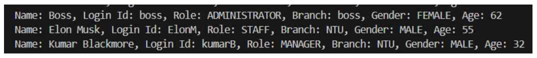
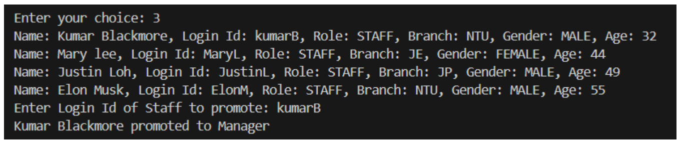
12. Process order	Manager can also process order using “3. Process order” and tracking process is updated like in Case 5 but for NTU branch.	
		`Enter your choice: 3`
		`---Process Order---`
		`Enter the Order ID: brmie`
		`Order ID brmie is processed successfully.`
		
		`List of Orders:`
		`NEW:`
		
		`READY TO PICK UP:`
		`brmie`

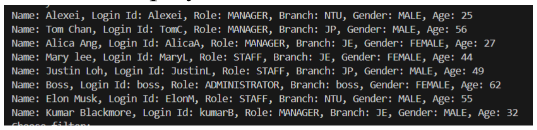
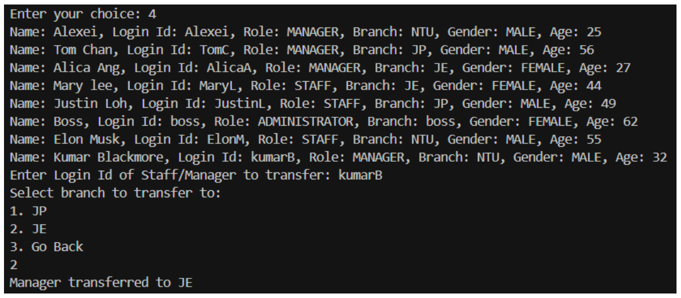
3.7 Admin Actions:

Test Case	Expected Outcome	Result

13. Close Branch	Admin is able to choose “6. Open/Close Branch” to Open or Close branch
	
	[Updated version of available branches]
	
	`------------------`
	`Please choose your current branch:`
	`1. NTU`
	`2. JP`
	`4. Return to Start _`
	[To reopen]
	
	`3`
	`Branch: JE is CLOSED.`
	`1. Open Branch`
	`2. Go Back`
		
		`Please choose your current branch:`
		`1. NTU`
		`2. JP`
		`3. JE`
		`4. Return to Start _`
		[Original branches customers can choose]
		
		`--------------------`
		`LOGIN AS ADMIN`
		`--------------------`
		`Choose your action:`
		`1. Display Staff List`
		`2. Add, Edit or Remove Staff Accounts`
		`3. Promote Staff`
		`4. Transfer Staff/Manager`
		`5. Add/Remove Payment Method`
		`6. Open/Close Branch`
		`7. Add/Remove Branch`
		`8. Change Password`
		`9. Return to Start`
		`Enter your choice: 6`
		
		`Select branch:`
		`1. NTU`
		`2. JP`
		`3. JE`
		`4. Go Back`
		`3`
		`Branch: JE is OPEN.`
		`1. Close Branch`
		`2. Go Back`
		`1`
		`Branch: JE is now CLOSED.`
		[Process of Admin closing branch]

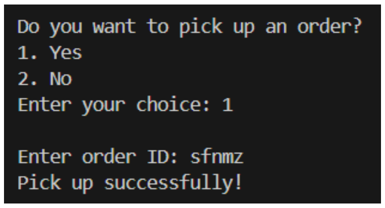
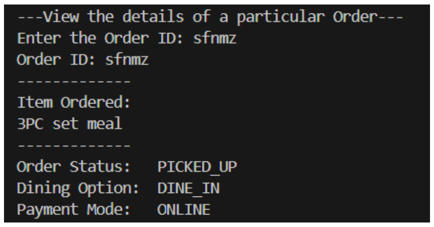
14. Login as an admin and display the staff list with filters (branch, role, gender, age).	Admin can filter staff list:
	
	
	`Select Branch to filter`
	`1. NTU`
	`2. JP`
	`3. JE`
	`4. Cancel`
	`1`
	`Name: Kumar Blackmore, Login Id: kumarB, Role: STAFF, Branch: NTU, Gender: MALE, Age: 32`
	`Name: Alexei, Login Id: Alexei, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 25`
	`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
	[Filter by branch ‘NTU’]
	
	`Select Gender to filter`
	`1. Male`
	`2. Female`
	`3. Cancel`
	`2`
	`Name: Alica Ang, Login Id: AlicaA, Role: MANAGER, Branch: JE, Gender: FEMALE, Age: 27`
	`Name: Mary lee, Login Id: MaryL, Role: STAFF, Branch: JE, Gender: FEMALE, Age: 44`
	`Name: Boss, Login Id: boss, Role: ADMINISTRATOR, Branch: boss, Gender: FEMALE, Age: 62`
	[Filter by gender ‘Female’]
		
		`Choose filter:`
		`1. Role`
		`2. Branch`
		`3. Gender`
		`4. Age`
		`5. Go Back`
		`1`
		`Select Role to filter`
		`1. Staff`
		`2. Manager`
		`3. Cancel`
		`1`
		`Name: Kumar Blackmore, Login Id: kumarB, Role: STAFF, Branch: NTU, Gender: MALE, Age: 32`
		`Name: Mary lee, Login Id: MaryL, Role: STAFF, Branch: JE, Gender: FEMALE, Age: 44`
		`Name: Justin Loh, Login Id: JustinL, Role: STAFF, Branch: JP, Gender: MALE, Age: 49`
		`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
		[Filter by role ‘Staff’]
		
		`Enter Age limit to filter: 30`
		`Name: Alexei, Login Id: Alexei, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 25`
		`Name: Alica Ang, Login Id: AlicaA, Role: MANAGER, Branch: JE, Gender: FEMALE, Age: 27`
		[Filter by ‘Age’, Eg. limit of 30]

16. Promote a staff to a Branch Manager	Admin choose “3. Promote Staff”
	
	`Name: Boss, Login Id: boss, Role: ADMINISTRATOR, Branch: boss, Gender: FEMALE, Age: 62`
	`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
	`Name: Kumar Blackmore, Login Id: kumarB, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 32`
	Zooming into the new staff list, it shows kumarB now being a manager.
		
		`Enter your choice: 3`
		`Name: Kumar Blackmore, Login Id: kumarB, Role: STAFF, Branch: NTU, Gender: MALE, Age: 32`
		`Name: Mary lee, Login Id: MaryL, Role: STAFF, Branch: JE, Gender: FEMALE, Age: 44`
		`Name: Justin Loh, Login Id: JustinL, Role: STAFF, Branch: JP, Gender: MALE, Age: 49`
		`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
		`Enter Login Id of Staff to promote: kumarB`
		`Kumar Blackmore promoted to Manager`

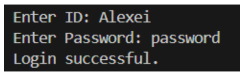
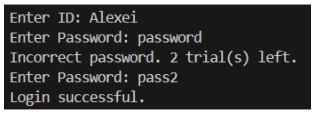
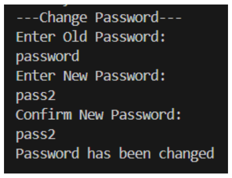
17. Transfer a staff/manager among branches	Admin can choose “4.Transfer staff/manager”
	
	If Admin displays new staff list, it will show
	
	`Name: Alexei, Login Id: Alexei, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 25`
	`Name: Tom Chan, Login Id: TomC, Role: MANAGER, Branch: JP, Gender: MALE, Age: 56`
	`Name: Alica Ang, Login Id: AlicaA, Role: MANAGER, Branch: JE, Gender: FEMALE, Age: 27`
	`Name: Mary lee, Login Id: MaryL, Role: STAFF, Branch: JE, Gender: FEMALE, Age: 44`
	`Name: Justin Loh, Login Id: JustinL, Role: STAFF, Branch: JP, Gender: MALE, Age: 49`
	`Name: Boss, Login Id: boss, Role: ADMINISTRATOR, Branch: boss, Gender: FEMALE, Age: 62`
	`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
	`Name: Kumar Blackmore, Login Id: kumarB, Role: MANAGER, Branch: JE, Gender: MALE, Age: 32`
	Where kumarB is now in branch JE
		
		`Enter your choice: 4`
		`Name: Alexei, Login Id: Alexei, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 25`
		`Name: Tom Chan, Login Id: TomC, Role: MANAGER, Branch: JP, Gender: MALE, Age: 56`
		`Name: Alica Ang, Login Id: AlicaA, Role: MANAGER, Branch: JE, Gender: FEMALE, Age: 27`
		`Name: Mary lee, Login Id: MaryL, Role: STAFF, Branch: JE, Gender: FEMALE, Age: 44`
		`Name: Justin Loh, Login Id: JustinL, Role: STAFF, Branch: JP, Gender: MALE, Age: 49`
		`Name: Boss, Login Id: boss, Role: ADMINISTRATOR, Branch: boss, Gender: FEMALE, Age: 62`
		`Name: Elon Musk, Login Id: ElonM, Role: STAFF, Branch: NTU, Gender: MALE, Age: 55`
		`Name: Kumar Blackmore, Login Id: kumarB, Role: MANAGER, Branch: NTU, Gender: MALE, Age: 32`
		`Enter Login Id of Staff/Manager to transfer: kumarB`
		`Select branch to transfer to:`
		`1. JP`
		`2. JE`
		`3. Go Back`
		`2`
		`Manager transferred to JE`

3.7 Customer Interface:

Test Case	Expected Outcome	Result

18. Place a new order, check the order status using the order ID, and collect the food	Customer will pick up orders by orderID  ```Do you want to pick up an order? 1. Yes 2. No Enter your choice: 1 Enter order ID: sfnmz Pick up successfully!```	 ```---View the details of a particular Order--- Enter the Order ID: sfnmz Order ID: sfnmz ------------ Item Ordered: 3PC set meal ------------ Order Status: PICKED_UP Dining Option: DINE_IN Payment Mode: ONLINE``` **[Order status PICKED_UP represents completed]**

3.8 Login System:

Test Case	Expected Outcome	Result
25. Change the default password, and log in with new password	Passwords are changed by each individual staff member, without going through a third party.  ```Enter ID: Alexei Enter Password: password Login successful.``` changes to  ```Enter ID: Alexei Enter Password: password Incorrect password. 2 trial(s) left. Enter Password: pass2 Login successful.```	 ```---Change Password--- Enter Old Password: password Enter New Password: pass2 Confirm New Password: pass2 Password has been changed```

Test cases 22, 26 and 27 passes, where we did not display results in the report.

4. Reflection

When we first started the project, we initially wanted to build our UML diagram first before starting on our code. However, we realise that it is quite tough where relationships between classes and functionalities are not clear and we are unable to test them out. Hence we changed our strategy to start building our code around the 4 agents to understand more on the interactions among them, where our UML code will be updated as we code, making the process easier and more efficient. However we still have guidelines so that we have SOLID principles in mind so that most of our classes exhibit low coupling and high cohesion.

We gained a deeper understanding on how to incorporate SOLID principles when coding, and also the use of new functionalities like reading and writing to files, and also the need to consider potential errors that can occur and update the system to allow it to catch those errors.

Further enhancements that we would want to explore is password encryption for user accounts to improve system's security to reduce security risk in case of data breach, which can help reduce impact on business.

5. References

1. https://blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams/