Design Document for Canteen Food Ordering System(CFOS)

1. Overview

Classes: (Basic building blocks of Canteen Food Ordering System)

Sl no.	Class	Principle Responsibility
1	Canteen Manager	Canteen owner controls and supervises the overall functioning of the RVCE Mingos Food Court. He supervises the food preparation and delivery process.
2	Customer	Customers are the end-users who purchase food items online through the web application. The customers include students, teaching and non-teaching staff of the college.
3	Menu	Stores and maintains the list of food items available in the canteen for purchase along with their pricing.
4	Order	Manages orders placed by the customer with the bill amount in real-time. Generates the unique token ID for each placed order upon payment confirmation.
5	Cart	Stores multiple food items that the customer wishes to order and provides the facility to add/delete/edit the quantity of food-items before checkout.
6	Payment	Authenticates transactions made by the customer and maintains the record of these transactions in real-time.
7	Wallet	Each user maintains a personal wallet to ensure end-to-end authentication of the completed transaction not only at the customer's end but also at the restaurant's end.

Note: *Other subsidiary classes may get added to the list in course of implementation for the purpose of load balancing and modularity.*

Actions:

Sl. no.	Action
1	Add/delete user
2	Add/delete/update menu items
3	View cart
3	Generate token ID
4	Payment authentication
6	Process debit
7	Place order
8	Generate report on sales statistics
9	Generate statement
10	Check sufficient balance
11	View available balance

Note: *There are other minor actions that do not play a major role in modeling.*

2. System Structure

2.1. Inheritance Structure

There does not seem to be any inheritance structure because of the lack of commonality between the classes. In some places inheritance seems intuitive, for example in specializing Security into BankSecurity and ShareSecurity and specializing Transaction into Buy and Sell. The figure below shows the inheritance structures.

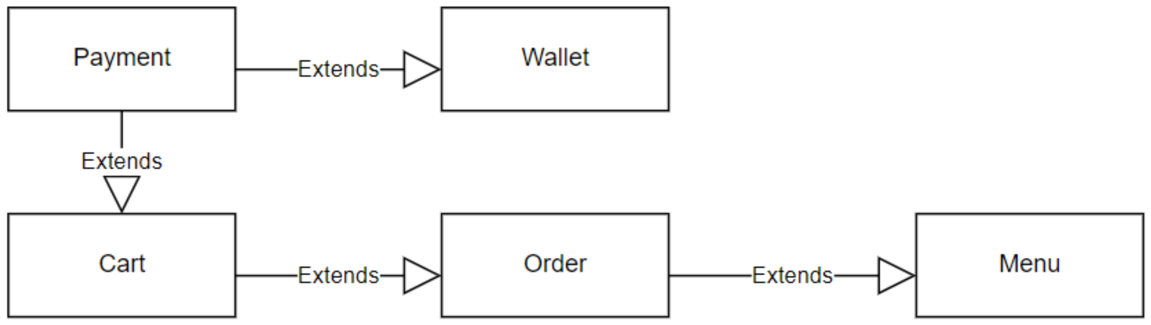

Fig 2.1: Possible Inheritance

2.2. Aggregations

The logical structure of CFOS suggests the following aggregation between the classes.

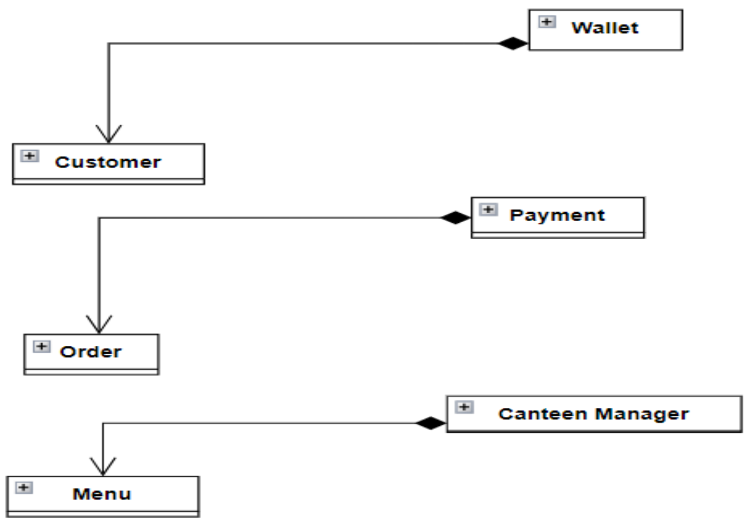

Fig 2.2.1: Aggregation Structure

2.3. Associations

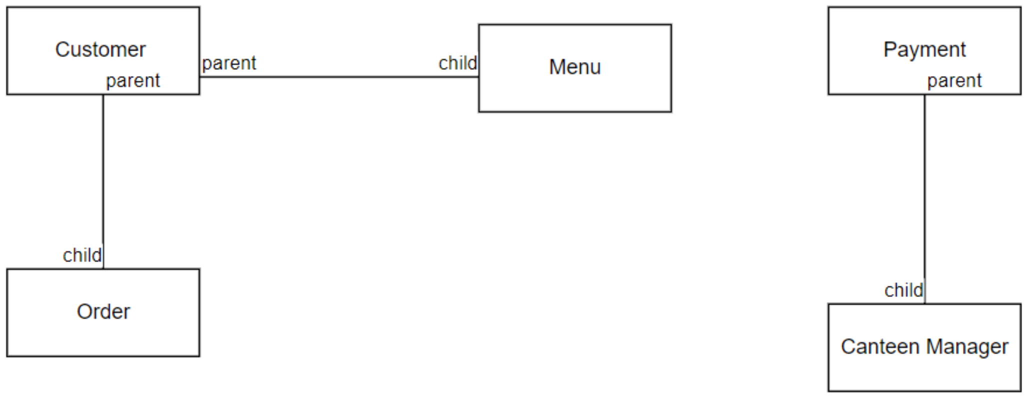

Fig 2.3.1: Class diagram showing associations

2.4. Complete class diagram

Finally after considering all the major actions the complete association + aggregation structure is arrived at.

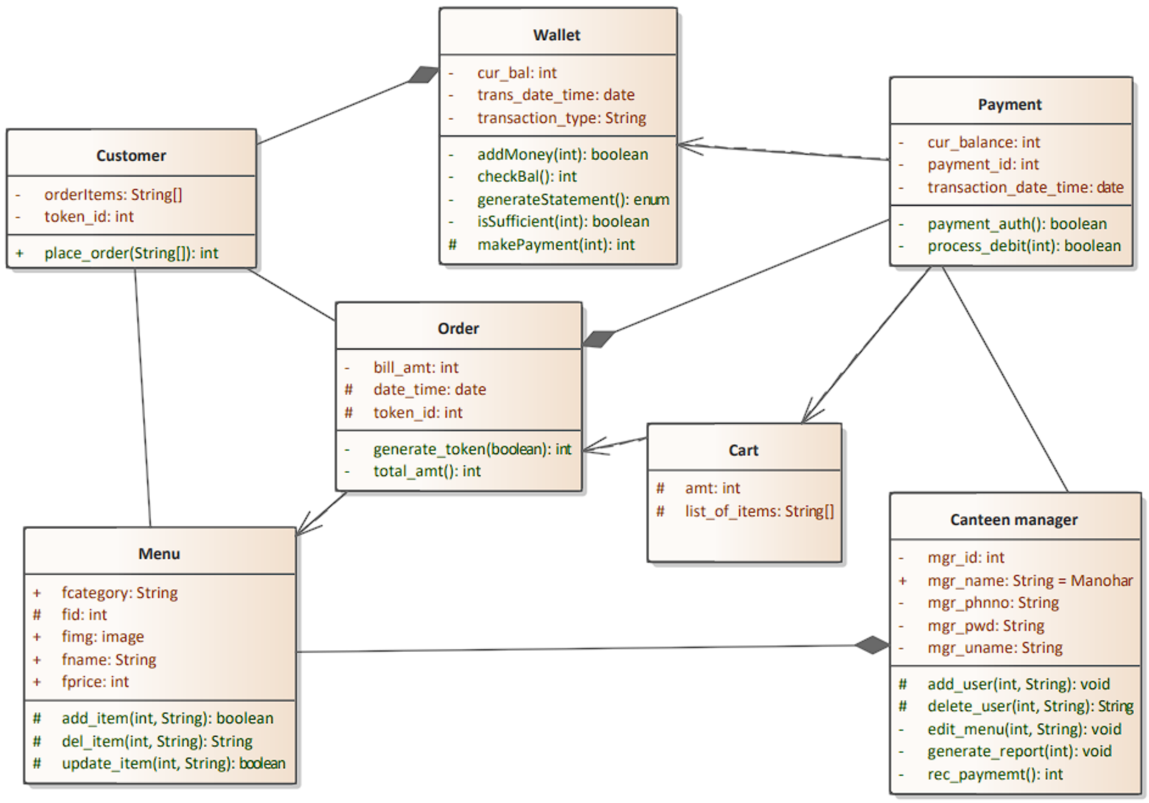
Wallet
- cur_bal: int
- trans_date_time: date
- transaction_type: String
- addMoney(int): boolean
- checkBal(): int
- generateStatement(): enum
- isSufficient(int): boolean
# makePayment(int): int
Payment
- cur_balance: int
- payment_id: int
- transaction_date_time: date
- payment_auth(): boolean
- process_debit(int): boolean
Customer
- orderItems: String[]
- token_id: int
+ place_order(String[]): int
Order
- bill_amt: int
# date_time: date
# token_id: int
- generate_token(boolean): int
- total_amt(): int
Cart
# amt: int
# list_of_items: String[]
Menu
+ fcategory: String
# fid: int
+ fimg: image
+ fname: String
+ fprice: int
# add_item(int, String): boolean
# del_item(int, String): String
# update_item(int, String): boolean
Canteen manager
- mgr_id: int
+ mgr_name: String = Manohar
- mgr_phnno: String
- mgr_pwd: String
- mgr_uname: String
# add_user(int, String): void
# delete_user(int, String): String
- edit_menu(int, String): void
- generate_report(int): void
- rec_paymemt(): int

Fig 2.4.1: Class diagram showing all classes and associations in the system

3. System Behavior

3.1 Sequence Diagram

The dynamic behavior of the system is modeled by figuring out the interactions between the classes involved in each principal action. The final diagram is being shown here. It should be remembered that these models have an impact in refining and enhancing the class diagrams.

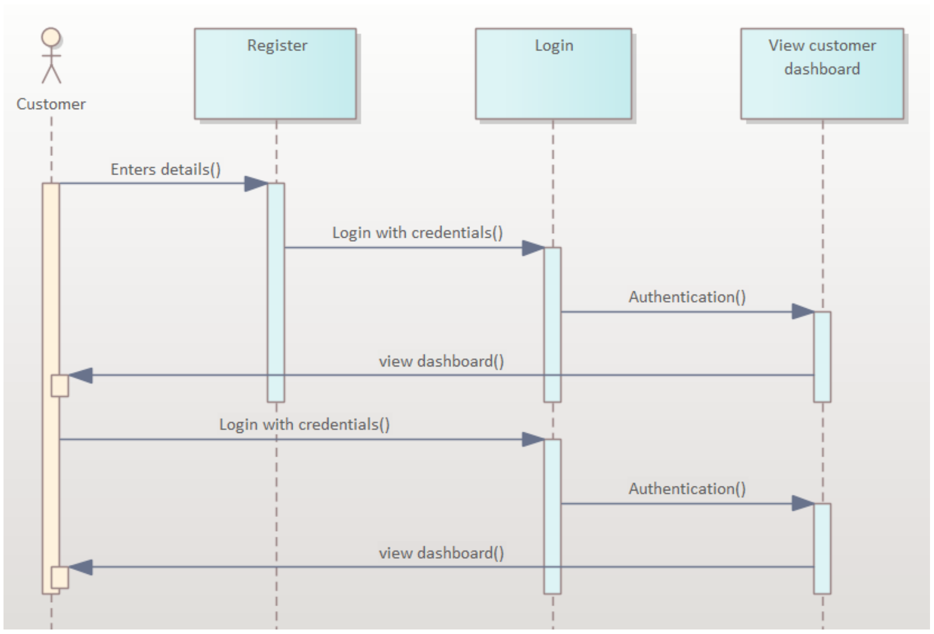

Fig 3.1 Principle Action: Login to the portal

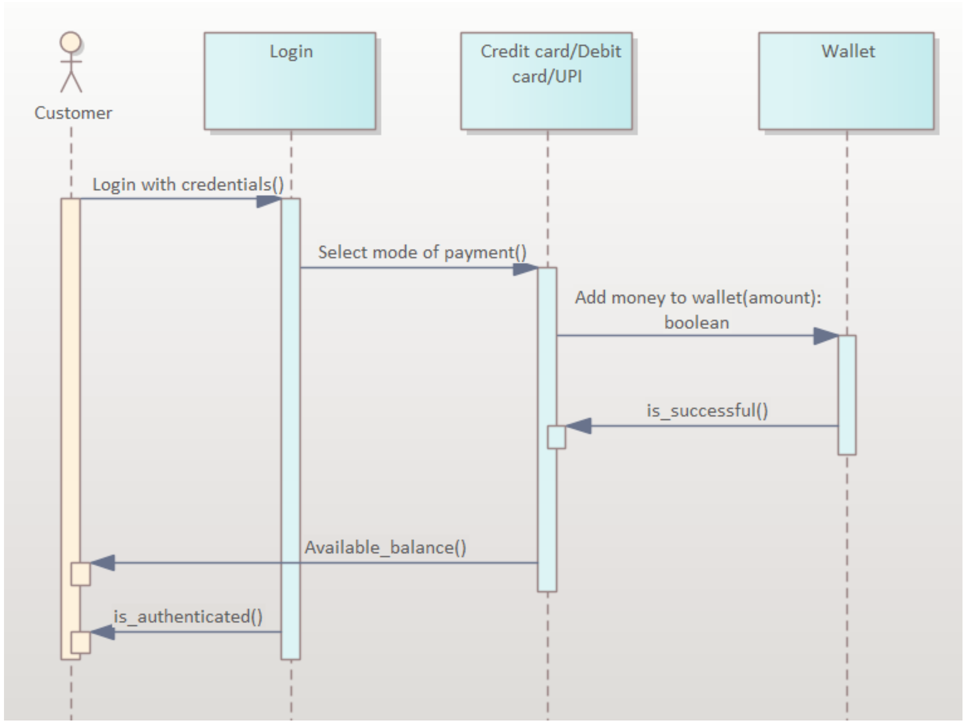

Fig 3.2 Principle Action: Adding money to wallet

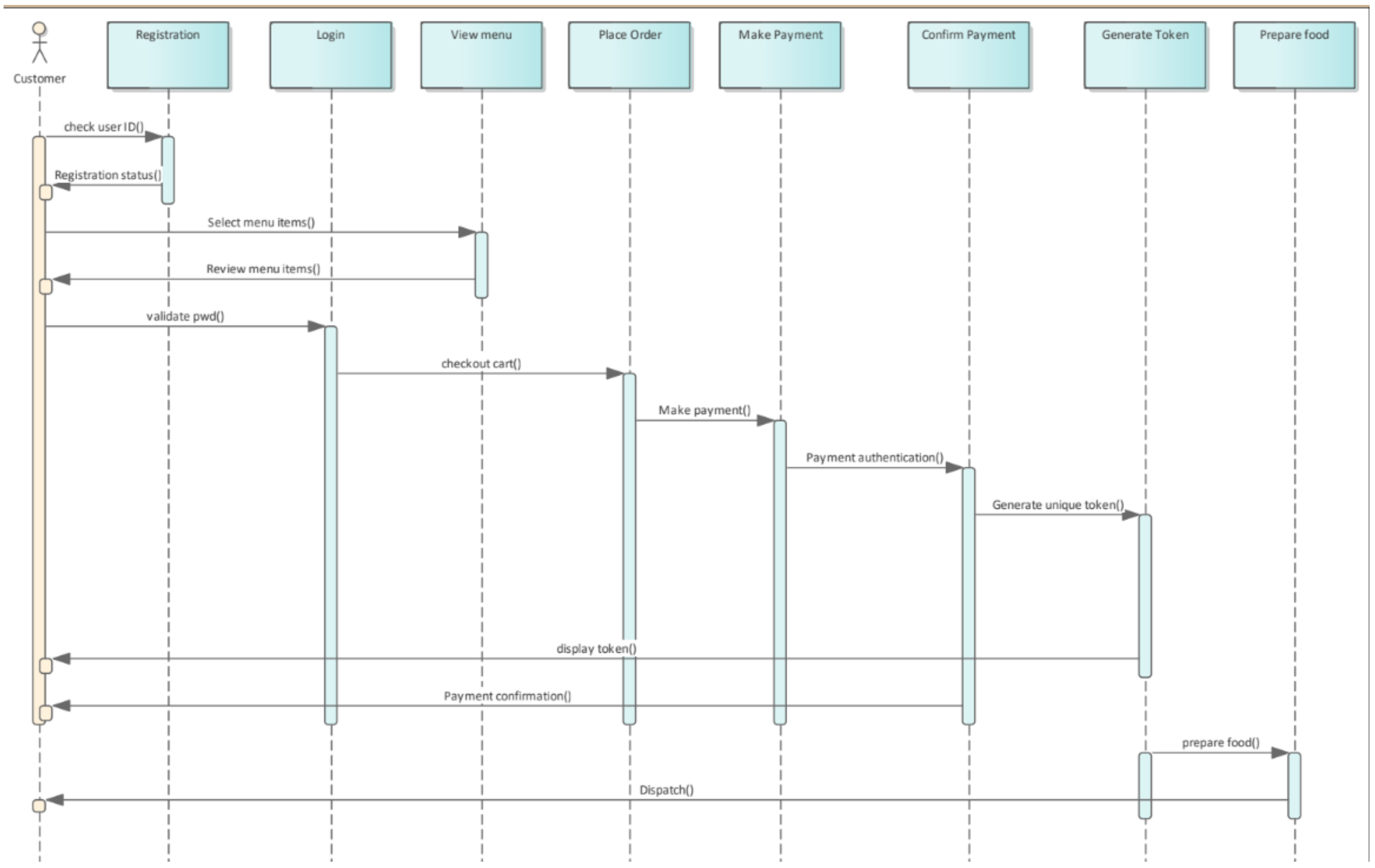

Fig 3.3 Principle Action: Place an order in the canteen

Now we are in a position to start with the design specification as we have all the attributes and methods of all the classes.

4. Detail Design Specification:

It consists of a list of main classes and their attributes and methods with proper comments.

```
1.  class Canteen_Manager{
            //attributes//
            - mgr_id: int                          //Manger ID
            + mgr_name: String = Manohar           //Manger_name initialized to the current manager
            name
            - mgr_phnno: String                    //Manger Phone number
            - mgr_uname: String                    //Manager's Username for login
            - mgr_pwd: String                      //Manager's password to login

            //methods//
            # add_user(int, String): void          //Add a user
            # delete_user(int, String): String     //Delete a user
            - edit_menu(int, String): void         //Edit the day's menu to update the items
            - generate_report(int): void           //Generate sales report monthly
            - rec_payment(): int                   //Receive payment from the customers
    }

2.  class Wallet{
            //attributes//
            - cur_bal: int                         //Current balance in the wallet
            - trans_date_time: date                //Date of the transaction
            - transaction_type: String             //Transaction type- credit/debit

            //methods//
            - addMoney(int): boolean               //Returns true if money is added to the wallet
            successfully
            - checkBal(): int                      //Returns the current Balance of the wallet
            - generateStatement(): enum            //Generate statement of the previous transactions
            - isSufficient(int): boolean           //returns true if the balance is meeting the initial
            balance criteria else false
            # makePayment(int): int                //Make payment to checkout
    }
3.  class Order{
            //attributes//
            - bill_amt: int                        //The total bill amount after checkout
            # date_time: date                      //Date of the order
            # token_id: int                        //Unique token ID generated for each order

            //methods//
            - generate_token(boolean): int         //return the Unique token ID generated
            - total_amt(): int                     //Calculate the total amount
    }
```

```
4.  class Customer{
            //attributes//
            - orderItems: String[]                  //List of items you add to cart
            - token_id: int                         //Unique token ID received by customer

            //method//
            + place_order(String[]): int            //Place an order and re-receive the total
            bill amount before confirmation
    }
5.  class Payment{
            //attributes//
             cur_balance: int                       //Current balance after the recent payment
            - payment_id: int                       //Payment ID of the checkout
            - transaction_date_time: date           //Transaction Date and time

            //methods//
            - payment_auth(): boolean               //returns true if payment is authorized
            - process_debit(int): boolean           //returns true if the debit is successful from the
            wallet
   }
6.  class Menu{
            //attributes//
            + fcategory: String                     //Food category-(South Indian/North
            Indian/Chaats)
            # fid: int                              //Food ID
            + fimg: image                           //Food image
            + fname: String                         //Food name
            + fprice: int                           //Price of food

            //methods//
            # add_item(int, String): boolean        //Returns true if item is added successfully else
            false
            # del_item(int, String): String         //Returns the deleted item from the menu
            # update_item(int, String): boolean     //Returns true if the item is updated successfully
            else false
   }
7.  class Cart{

            //attributes//
             + amt: int                             //Total amount of all items in the cart

        //Method//
            # list_of_items: String[]               //List of food items in the cart
   }
```

5. Use Case Diagram

Details of the use case diagram for Canteen Food Ordering System:

Use Case #	Use Case	Description
1	View Menu	The user gets to view the menu along with illustrations and prices.
2	Order Item	Select preferred items and add them to the cart.
3	Payment At checkout	Pay the total bill amount through the wallet in the application.
4	Accept Order	After the payment is accepted, order is accepted and thus received by the kitchen staff.
5	Generate Token	A token number is automatically generated based on the number of orders on that day.
6	Prepare Order	The order gets prepared by the kitchen staff.
7	Dispatch Order	Kitchen staff checks the generated token and delivers the order at the counter.
8	Add Money To wallet	A minimum balance along with an extra top-up amount can be added to the wallet through Third party payment apps like GPay.
9	View Transactions	The canteen manager can view all the transactions being performed.
10	View Reports on sales statistics	Canteen manager gets reports on the sales statistics for a custom duration.
11	Edit Menu	Canteen manager gets to add or delete a dish from the menu.

Details of each use case in the below given format

Use Case	**1. View menu**
Description	Allows a member to view the menu along with the image and pricing for the same with or without logging into the portal.
Assumptions	None
Actors	● Customer (students, teaching and non-teaching staff)
Steps	1. User opens the website 2. User views all the dishes 3. User selects a category 4. User views dishes according to category
Variations	The menu could be viewed with or without logging into the website.
Non-functional	The System must support multiple sessions at once. The system must not crash. Loading speed of the website must be minimum to ensure a seamless user experience.
Issues	Images may not be displayed due to slow internet.

Use Case	**2. Order Item**
Description	Allows a member to select preferred items and add them to the cart.
Assumptions	At Least 1 item is selected. The User has logged in.
Actors	● Customer (students, teaching and non-teaching staff)
Steps	1. User logs into the website. 2. User views all the dishes. 3. User selects all the required dishes. 4. User adds the selected dishes into the cart. 5. User proceeds to checkout.
Variations	No variations available.
Non-functional	Unique token ID for the order is only generated on successful payment at checkout.
Issues	A condition may occur such that the number of items available is less than the quantity chosen.

Use Case	**3. Payment at Checkout**
Description	Pay the total bill amount through the wallet in the application and receive a confirmation after the payment has been made.
Assumptions	The user has logged into their account.
Actors	● Customer (students, teaching and non-teaching staff)
Steps	1. The user Clicks on “Proceed to checkout”. 2. The user checks his balance in the wallet. 3. The user clicks on “Make Payment”. 4. The user waits to receive a confirmation.
Variations	None
Non-functional	Give alerts to the customer if the current balance in the wallet is less than the required minimum balance and ensure sufficient balance in the wallet at all times before checkout.
Issues	The user might not have enough balance in his wallet.

Use Case	**4. Accept Order**
Description	After the payment is made, order is accepted and thus received by the kitchen staff.
Assumptions	An order is placed by the user.
Actors	● Customer (students, teaching and non-teaching staff)
Steps	1. Payment has been made. 2. The confirmation is received by the user. 3. Order to be prepared by the kitchen staff is displayed in their system.
Variations	None
Non-functional	The System must convey orders correctly.

Issues	The payment may not be confirmed on the server side.

Use Case	**5. Generate Token**
Description	A unique token is generated for the user using which the order can be collected when dispatched.
Assumptions	The user has placed an order and paid for it. The kitchen staff has received the order.
Actors	● Customer (students, teaching and non-teaching staff) • Kitchen Staff
Steps	1. The user makes payment for his order 2. The kitchen staff receives the order. 3. The user gets a token for his order.
Variations	None
Non-functional	Time taken to receive a request must be minimum.
Issues	The user's payment might not be successful.

Use Case	**6. Prepare Order**
Description	The order gets prepared by the kitchen staff.
Assumptions	The kitchen staff have received the order details.
Actors	● Kitchen Staff
Steps	1. The user waits for the order to be prepared by the kitchen staff.
Variations	None
Non-functional	Prominent display of the placed order with the token ID in real-time to ensure no wait time for the kitchen staff to receive the request for the next order.
Issues	The menu might not be edited when the order is placed(in case the dish is out of stock).

Use Case	**7. Dispatch Order**
Description	Kitchen staff checks the generated token and delivers the order at the counter.
Assumptions	The kitchen staff have received the order details and prepared the order.
Actors	● Customer (students, teaching and non-teaching staff) • Kitchen Staff
Steps	1. The user shows the token number on his phone 2. The kitchen staff checks for the token number. 3. The kitchen staff dispatches the order to the customer. 4. The User collects his order at the counter.

Variations	None
Non-functional	Uniqueness of the Token ID to ensure the correct positioning of the parcel on the table counter is critical to the dispatch process. Customers should easily access the parcel and should be able to quickly leave with the parcel in a seamless manner.
Issues	A wrong item might be delivered to the customer.

Use Case	**8. Add money to wallet**
Description	A minimum balance along with an extra top-up amount can be added to the wallet through Third party payment apps like GPay or credit,debit cards.
Assumptions	The user has logged into their account.
Actors	● Customer (students, teaching and non-teaching staff)
Steps	1. Customer logs into his account and views the current balance. 2. Customer clicks on the Add money icon which redirects to the payment gateway. 3. The customer can select any mode of payment [UPI, Credit/Debit card or net banking] to transfer money to the wallet. 4. Upon successful payment, the amount is added to the current balance of the wallet.
Variations	None
Non-functional	Delivery response time must be less.
Issues	The user may add an amount that is less than the minimum that must be added.

Use Case	**9. View Transactions**
Description	The canteen manager can track all the transactions being performed.
Assumptions	● The canteen manager has logged into their account. ● At Least one transaction must be taking place.
Actors	● Canteen manager
Steps	1. Login into the portal. 2. Click on “View transactions” 3. View the transactions.
Variations	None
Non-functional	Time taken to deliver the response must be fast.
Issues	Multiple transactions made by the customers at a given time may result in data inconsistency.

Use Case	**10. View Reports on sales statistics**
Description	Canteen manager gets reports on the sales statistics for a custom duration.
Assumptions	● The manager logs into the website correctly.

	● The duration specified must be for a week or a month.
Actors	● Canteen manager
Steps	1. Login into the portal. 2. Click on "View reports" 3. Set the custom duration 4. View the sales and statistics for the specified duration.
Variations	In case the manager does not set a specific duration, the report for the previous month will be generated.
Non-functional	A comprehensive analysis of the revenue generated must be presented.
Issues	Sales duration specified may include holidays when the college remains closed.

Use Case	**11. Edit Menu**
Description	Canteen manager gets to add or delete or update a dish from the menu.
Assumptions	The manager logs into the website correctly.
Actors	● Canteen manager
Steps	1. Canteen manager logs into the system as admin. 2. Canteen manager views the menu and selects the item to modify. 3. Upon selection of the item, an item can be updated or deleted from the menu. 4. A new item can be added by the manager.
Variations	An item can be deleted, updated or added into the menu.
Non-functional	Only the admin is authorized to change the menu.
Issues	● The manager adds an item that already exists in the menu. ● The manager modifies an item that doesn't exist in the menu.

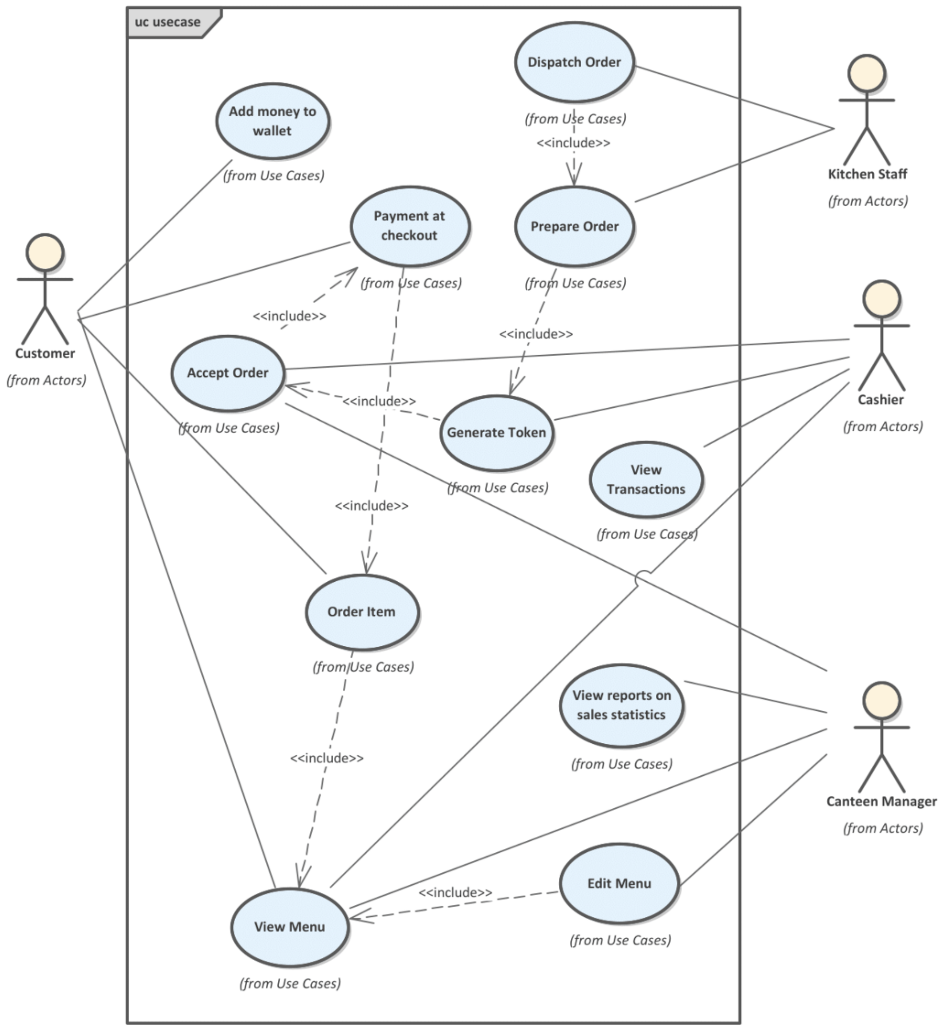

Fig 5.1: Use Case diagram: Complete overview of the canteen management system