BRAC UNIVERSITY

Inspiring Excellence

CSE471 : System Analysis and Design

Lab Project Report

Project Title : Automated Restaurant Management System

Group Members

Group No : 01, CSE471 Lab Section : 02, Summer 2020		
Name	**Student ID**	**Contribution**
Shoaib Ahmed Dipu	17101482	Functional & Non Functional Requirements, Class Diagram & Data Flow Diagram
Mehadi Hassan	17101177	Use Case Diagram & Activity Diagram
Shemonto Das	17101447	Introduction, System Request, Sequence Diagram & Conclusion

Table Of Contents

Introduction

Our *Automated Restaurant Management System* aims to digitalize & automate the day to day processes of various restaurant operations including ordering, billing, kitchen, hall and inventory management.

Why We Intend To Build This System

We intended to build the system due to these reasons :

I. To improve the performance of the restaurant by eradicating the daily paperwork.

II. To perform the tasks in less amount of time and more efficiently.

III. To load balance effectively during rush hours.

IV. To minimize human error that occurs when performing tasks manually.

V. To assign tasks to chefs effectively so that congestion in the kitchen could be avoided.

VI. To keep track of the monthly/yearly by using the billing module and to see the trends in sales and profits.

System Request

A system request is a document that describes the business reasons for building a system and the value that the system is expected to provide. The project sponsor usually completes this form as part of a formal system project selection process within the organization. Most system requests include five elements: Project Sponsor, Business Need, Business Requirements, Business Value, and Special Issues or Constraints. Our System Request has also got these five elements.

System Request For Automated Restaurant Management System

Project Sponsor : Carl Edwards, CEO, DineOut Restaurant

Business Need :

An automated Restaurant Management System is required which will automate the day to day processes of various restaurant operations including ordering, billing, kitchen, hall and inventory management.

Business Requirements :

I. Customers should be able to scroll through the menu and select the dishes he/she wants.
II. Customers should be allowed to cancel/edit the order any time before its prepared.
III. The system will allow Customers to provide feedback regarding the food and overall service of the restaurant.
IV. Customers should be able to request for bill & ask for help through the system.
V. Head Chef should be able to assign the dishes in an order to chefs according to their specialties.
VI. Chefs should be able to check dish queues and their status.
VII. Chefs should be able to check the availability of the Ingredients.
VIII. The system will allow admin to perform CRUD (create, retrieve, update and delete) operations on Staff Members, Menu Items and Inventory.
IX. Head Chef should be able to mark orders complete & approve cancellation of dish or order.
X. Hall Manager should be able to mark the bill as paid.
XI. Hall Manager should receive notification when a particular order is complete.
XII. Hall Manager should be able to see/edit status of tables reserved and available and their capacities.
XIII. The system should include Popular Payment Gateways so that Customers should be able to pay Bills apart from Cash Payment.
XIV. Using the Payment Gateways, Management should be able to pay the employees.

Business Values :

I. Increase of sale : 5,00,000 / Month.
II. Reduced cost of Extra Employee : 1,20,000 / Month.
III. Advertisement on Menu Display : 30,000 / Month.
IV. Satisfaction of Customers.
V. Referral of Customers.

Constraints :

I. System should be compatible and should smoothly run on both older & latest versions of both Android and iOS.
II. Core system and its user interfaces should be compatible with tablets. However, running on small smart devices is not necessary.

Functional & Non Functional Requirements

A requirement is simply a statement of what the system must do or what characteristics it needs to have. There are two types of requirements, Functional and Non-Functional.

Functional Requirements can further be sub divided into two parts :

I. Process-oriented : A process the system must perform.

II. Information-oriented : Information the system must contain.

Non Functional Requirements can further be sub divided into four parts :

I. Operational : The physical and technical environments in which the system will operate.

II. Performance : The speed, capacity, and reliability of the system.

III. Security : Who has authorized access to the system under what circumstances.

IV. Cultural and Political : Cultural and political factors and legal requirements that affect the system.

In our System, there can be five different type of Users, Customer, Admin, Head Chef, Chef and Manager. We have provided Functional and Non Functional Requirements as per the Users.

Functional & Non Functional Requirements	
Functional Requirements	**1.Process Oriented** 1.1 Customers can scroll through the menu and select the dishes as per their preferences. 1.2 Customers can request to cancel or edit the order. 1.3 Customers can request for bill & ask for help through the system. 1.4 Customers can provide feedback regarding the food and overall service of the restaurant. 1.5 Head Chef can assign the dishes in an order to Chefs according to their specialties. 1.6 Chefs can check dish queues and their status. 1.7 Chefs can check the availability of the Ingredients. 1.8 System admin can perform CRUD(create, retrieve, update and delete) operations on Staff Members, Menu Items and Inventory. 1.9 Head Chef can mark orders as 'Completed' & can approve cancellation of dish or order. 1.10 Hall Manager can mark the bill as paid. 1.11 Hall Manager can receive notification when a particular order is complete. 1.12 Hall Manager can check or edit status of tables reserved and available and their capacities. 1.13 Popular Payment Gateways must be available in the system so that Customers can pay Bills apart from Cash Payment. 1.14 Management also can pay the employees using the Payment Gateways.
	2. Information-oriented 2.1 The system must show all available and unavailable dishes to the Customer and the operations of unavailable dishes must be disabled. 2.2 System will classify the dishes in the order according to category and add this dish on a particular Chef's queue in the kitchen screen. 2.3 For order cancellation, all the dishes will be presented for approval to the Head Chef. Only approved dishes will be dropped. 2.4 Hall manager can check the order no, table no and total payable amount. 2.5 Customers can check their order history of atleast last six months. 2.6 The system must include budgeted and actual sales and expense amounts for current year and last two years.
Non-Functional Requirements	**1.Operational** 1.1 The system can be run on Tablets and the Tablets would be present in the restaurant. It should be available both for Android and iOS.

	1.2 The system should be up and running for most of the time and server should not be down for more than a few minutes to avoid inconvenience of the customers. 1.3 The bill generated by the application must be accurate and the orders placed should exactly be the same which the customer has selected. 1.4 Management can easily install the App on devices and the system would run smoothly according to the requirements. 1.5 Interface of the software must be easy to use. It would not be complex since managers, chefs have a view, so interface should be simple.
	2. Performance 2.1 In case of scrolling through the menu there should be a delay of no more than 2 second before the next page of menu items is displayed otherwise our people's dining experience will be affected. 2.2 The order should be placed in pending orders and be visible to the Head Chef and Chefs in less than 1 second to start the preparation. 2.3 'Cancel Order' and 'Update Order' should be made with little delay to avoid delivery delay. 2.4 While connecting to the Firebase server the delay to make a successful connection should be less for effective real time communication. 2.5 The menu should have flexible fonts that can be zoomed so as to not over constrain the eyes.
	3. Security 3.1 Proper and encrypted login authentication for Head Chef and Admin should be available as sensitive information of Employees as well as inventory should be protected. 3.2 Information transmission should be securely transmitted to Firebase without any changes in information to avoid disturbances in orders and billing. 3.3 Only Admin can access to perform add, delete, update operations on the database for menu, inventory, employees and no other person can modify the data in the Database. 3.4 Chefs can only view the orders and cannot remove an order from their queue. Only Head Chef can interact with the queues containing orders. 3.5 Only Admin should have access to Customers' information.
	4. Cultural and Political 4.1 Customers should be able to pay bills in multiple currencies, such as, BDT and USD.

	4.2 Company policy has been set to buy equipment only from Ryans Computers and Star Tech & Engineering Limited. 4.3 Personal Information should be protected in compliance with the law of the Country. 4.4 Customers should be provided with the detailed information of Payment. Tax Paying Information of the Restaurant should be provided on Cash Memo.

Use Case Diagram

There are four major actors of our Restaurant Management System which are Customer, Hall manager, Chef, and Admin.

Customer Use Case Diagram :

To begin with, if we look at the use case diagram for the actor customer, customers can place an order but to do so the customer needs to select the dishes first that is why there is an include relation between place order and select dishes.

Customer can also cancel the order but for that, it needs to be approved by the head chef whether it can be canceled or not. So there is an include relation here as well.

Moreover, customers can update order, seek help, provide feedback along with checking his previous order.

To make payment customer needs to first request for the bill so it is presented with an include relation. The payment will be done using the secondary actor payment gateway.

Hall Manager Use Case Diagram :

If we move to our next actor which is the Hall Manager we can see manager can mark whether the payment was paid or not but for that customer needs to make payment first that is why there is an include relation between mark paid bill and make payment.

Manager can also active notifications when the chef completes the order or the customer seeks for help. As it is an optional action so it is shown with an exclude relation.

Furthermore, the manager can edit the table status, and to do that the table status needs to be checked first which is given with an include relation.

Lastly, the hall manager can check the payable amount as well.

Chef Use Case Diagram :

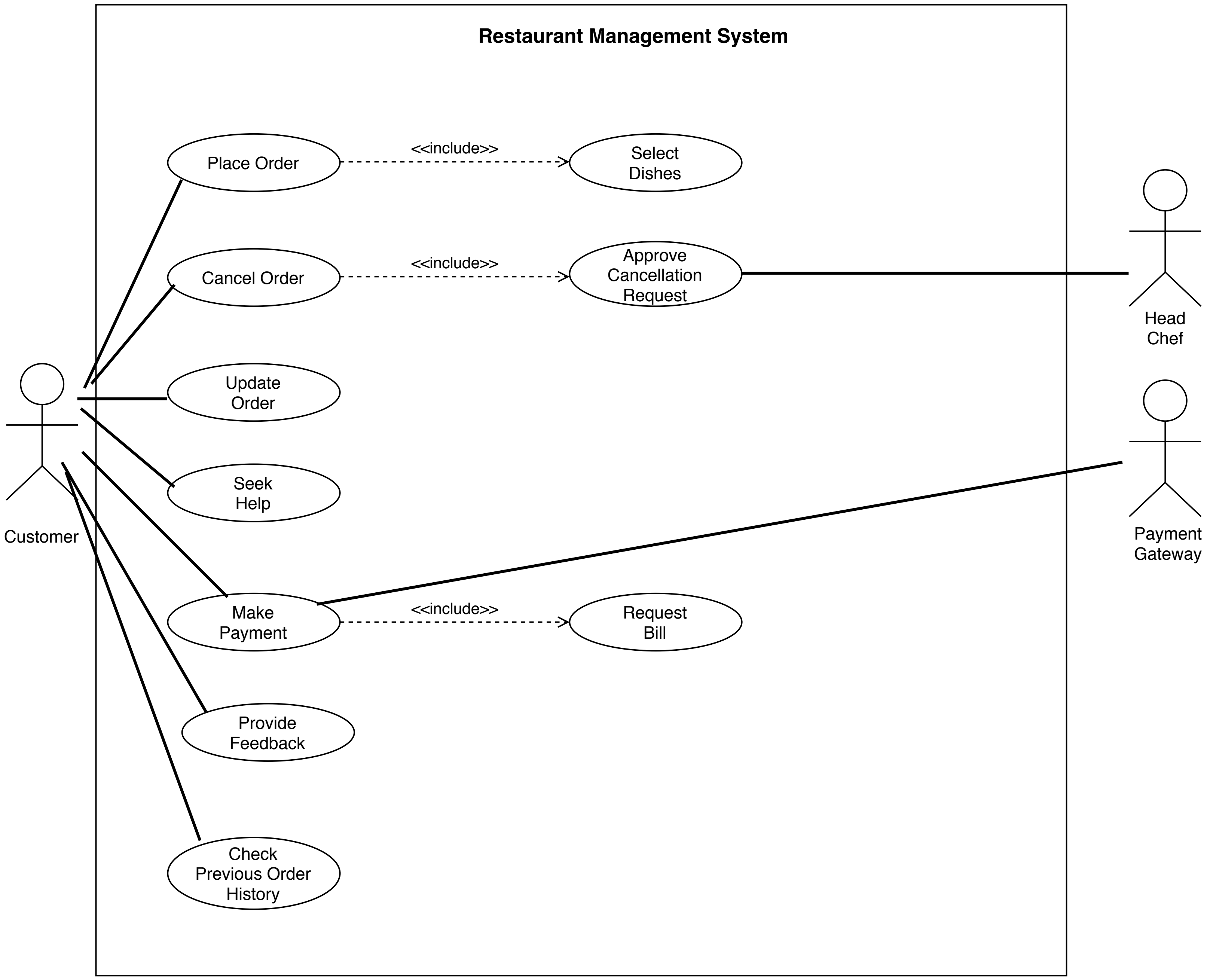
Chef is another important actor in our Restaurant management system. Firstly, the chef is generalized to Regular chef and Head chef and they both have different use cases. The head chef can mark the cooking as complete or not and for that cooking needs to be finished by the regular chef as it is a mandatory task, it is provided with an include relation. The same goes here, to approve the cancelation of dishes first the customer has to request a cancel order, and only then Head chef can approve it, as a result, it is also shown with include relation.

Furthermore, a Regular chef can prepare dishes that are assigned by the head chef which is also demonstrated with the include relation. Moreover, a Regular chef can check dish ques along with check the availability of the ingredients.

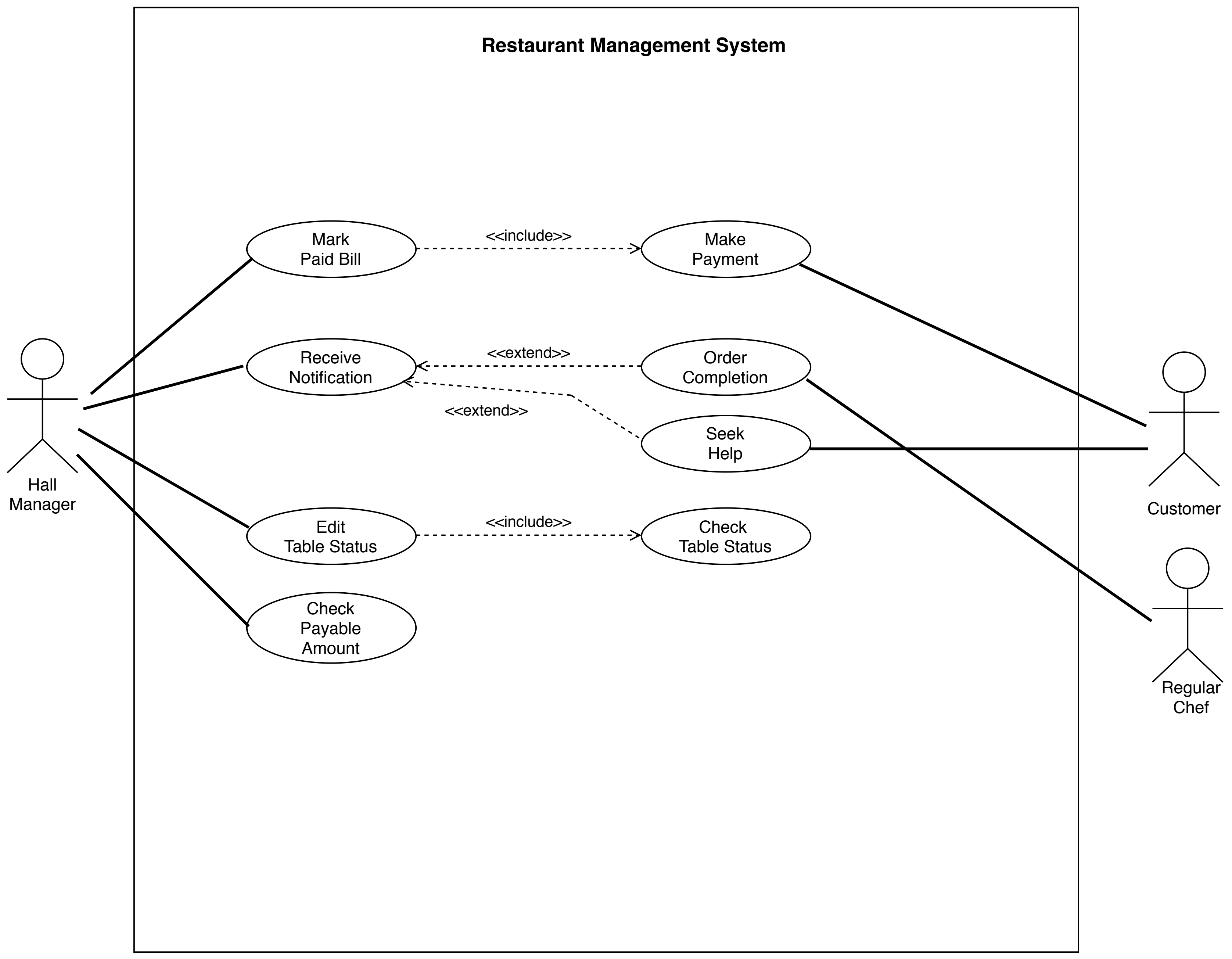
Admin Use Case Diagram :

Lastly, our final actor is Admin which performs create, retrieve, update and deletion of staff members, menu items, and inventory items.

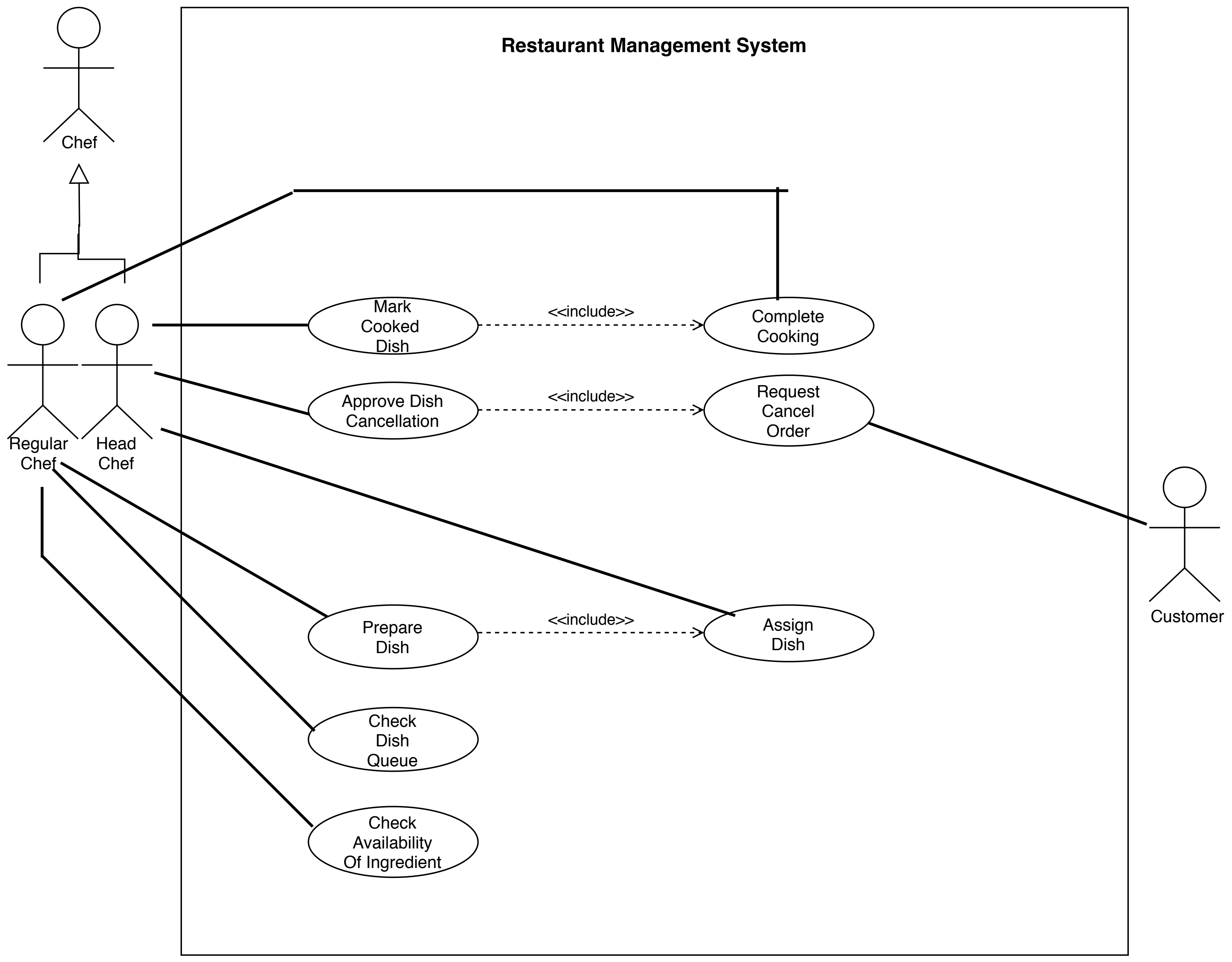
Use Case Diagram
Actor : Customer



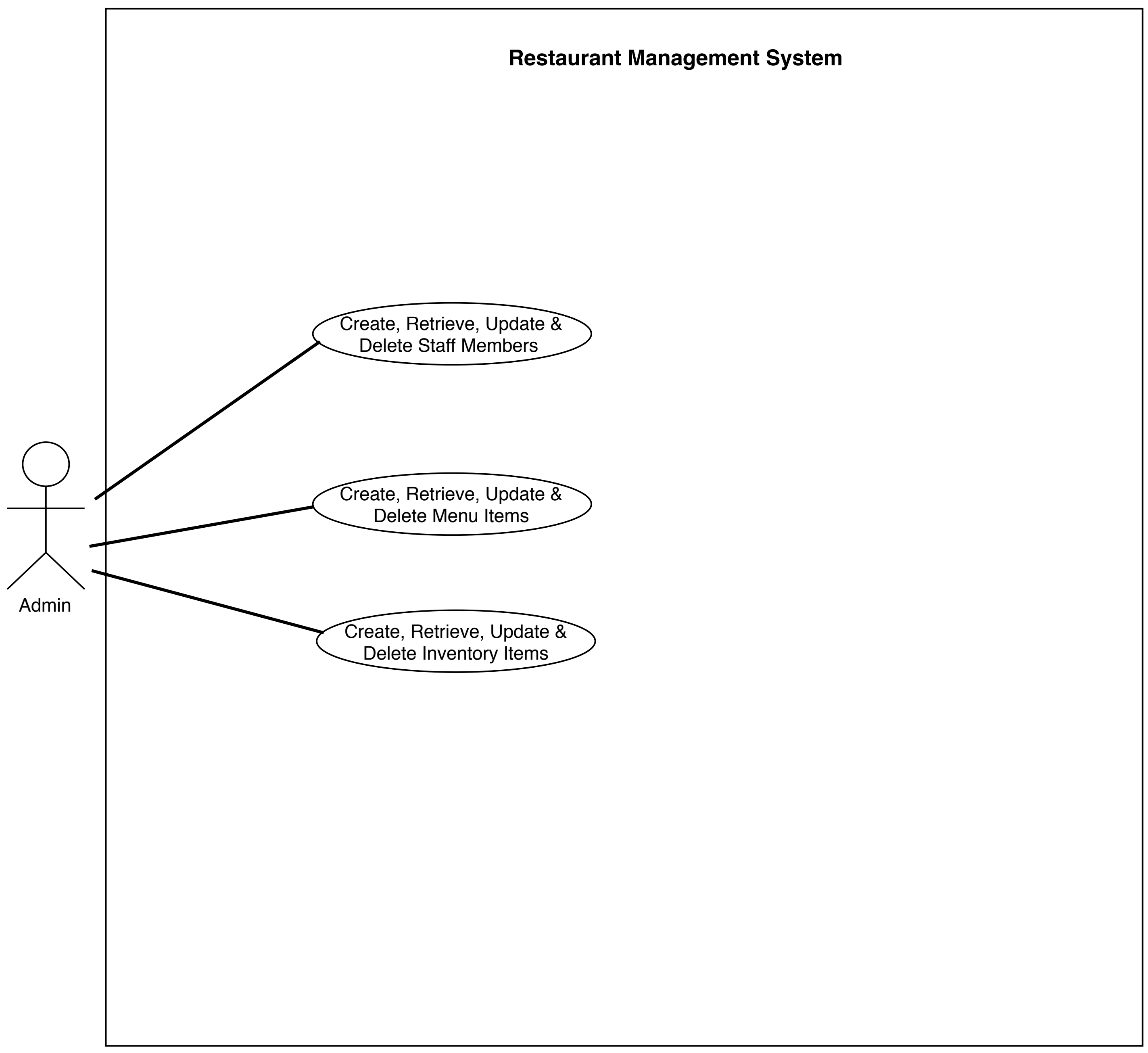
Use Case Diagram
Actor : Hall Manager



Use Case Diagram
Actor : Chef



Use Case Diagram
Actor : Admin


Restaurant Management System
Create, Retrieve, Update & Delete Staff Members
Create, Retrieve, Update & Delete Menu Items
Create, Retrieve, Update & Delete Inventory Items
Admin

Class Diagram

Relationships among the classes:

Generalization Relation : Parent Class : Staff, Child Class : HallManager, Chef.

Generalization Relation : Parent Class : Chef, Child Class : HeadChef, RegularChef.

Composition Relation : Between Customer class & OrderHistory class. Composition denotes a strong relationship between two classes. Here, Customer is the Master Class. There can be no existence of OrderHistory class without the existence of Customer class.

Composition Relation : Between Customer class & OrderCart class. Here, Customer is the Master Class. There can be no existence of OrderCart class without the existence of Customer class.

Aggregation Relation : Between OrderCart class and Bill class. Here, OrderCart is the Master Class. There can be existence of Bill class without the existence of OrderCart class.

Aggregation Relation : Between OrderCart class and OrderStatus class. Here, OrderCart is the Master Class. There can be existence of OrderStatus class without the existence of OrderCart class.

The rest of the Relationships are Association Relationship among the classes. From Multiplicity, we can get to know the lowest and highest possible number of instances of a Class.

Staff
- name : String
- email : String
# password : String
- salary : int
- staffID : int
+ calculateSalary() : void
+ withdrawSalary() : void
+ addBonus() : void
+ leaveRecord() : void
create, retrieve, update & delete
0..*
1
Admin
- name : String
- email : String
# password : String
- operation : String
+ create(operation : String) : void
+ retrieve(operation : String) : void
+ update(operation : String) : void
+ delete(operation : String) : void
create, retrieve, update & delete
1
0..*
InventoryItem
- inventoryItemType : String
- inventoryItemName : String
- inventoryItemID : int
- quantity : int
- inventoryItemPrice : int
+ addItem(itemID : int) : void
+ removeItem(itemID : int) : void
+ updateItem(itemID : int) : void
+ checkItemQuantity(itemID : int) : void
OrderCart
- itemList : ArrayList
- orderID : int
+ addItem(itemID : int) : void
+ deleteItem(itemID : int) : void
+ setQuantity(itemID : int) : void
+ cancelItem(itemID : int) : void
+ updateOrder() : void
1..*
1
1
yields
1..*
logs in
1..*
1
1..*
logs in
1
HallManager
- hallManagerID : int
+ markPaidBill(billID : int) : void
+ checkNotifications() : void
+ editTableStatus(tableID) : void
+ checkPayableAmount(billAmount : int) : void
Chef
- chefID : int
1
create, retrieve, update & delete
0..*
Menu
- itemList : ArrayList
+ sortMenuByCategory() : void
+ sortMenuByPrice() : void
SystemLogin
- email : String
# password : String
+ name : String
+ userType : String
+ emailVerification() : boolean
+ loginVerification() : boolean
+ checkUserType(userType : String) :
1
gets added to
order status
Item
- name : String
- itemID : int
- price : int
+ checkRecipe() : void
+ checkIngredients() : void
1..*
adds to
1..*
*
confirms order
Bill
- billAmount : int
+ generateBill() : void
+ updateBill() : void
1..*
gets paid
1..*
1..*
receives
checks
logs in
*
selects
HeadChef
+ markCookedItem(orderID : int) : void
+ approveDishCancellation(orderID : int) : void
+ assignItem(itemID : int, chefID : int) : void
RegularChef
+ checkDishQueue() : void
+ prepareDish(itemID : int) : void
+ checkAvailabilityOfIngredients(inventoryItemID) : void
1
1..*
1..*
1
1
Customer
- name : String
- address : String
- email : String
# password : String
- orderHistory : Queue
+ checkMenu() : void
+ confirmOrder() : boolean
+ updateOrder() : void
+ cancelOrderRequest() : boolean
+ seekHelp() : void
+ makePayment() : boolean
+ provideFeedback() : void
+ checkPreviousOrderHistory() : void
*
makes
1
PaymentGateway
- finalAmount : int
- credentialID : String
# credentialPass : String
+ makePayment() : boolean
+ confirmMessage() : String
+ provideCredential() : String
1
*
OrderStatus
- status : String
+ checkStatus(orderID : int) : void
+ approveCancelRequest(orderID : int) : boolean
*
*
Notification
- notificationList : ArrayList
- notificationType : String
+ showNotification() : void
+ sortNotificationByTime() : void
+ groupNotificationByCategory() : void
Table
+ tableID : int
+ checkTableStatus(tableID : int) : void
+ editTableStatus(tableID : int) : void
assigns tasks
receives tasks
1
have
*
1
1
1
OrderQueue
- orderList : ArrayList
- orderID : int
+ addOrder(orderID : int) : void
+ deleteOrder(orderID : int) : void
+ assignOrder(orderID : int, chefID : int) : void
+ cancelOrder(orderID : int) : void
+ updateOrder() : void
OrderHistory
- date : String
- orderNo : int
- orderHistory : Queue
+ checkOrderHistoryByDate() : void
+ checkOrderHistoryByMonth() : void

Activity Diagram

Customer Activity Diagram :

First of all, upon starting the system from start node customer will have to login to the system. The login request will be checked by the RMS. In the decision node, it will verify the request. If it is invalid it will be rejected and end the activity. On the other hand, if its approved customer will be greeted with choices like select dish or cancel dish in the decision node. If the customer selects dishes he will place the order and RMS will receive the order. If the customer wants to cancel the order, firstly the request will go to the RMS and then it will be forwarded to the head chef for approval. If the head chef approves the cancellation customer are notified by the RMS otherwise it gets rejected and ends the activity.

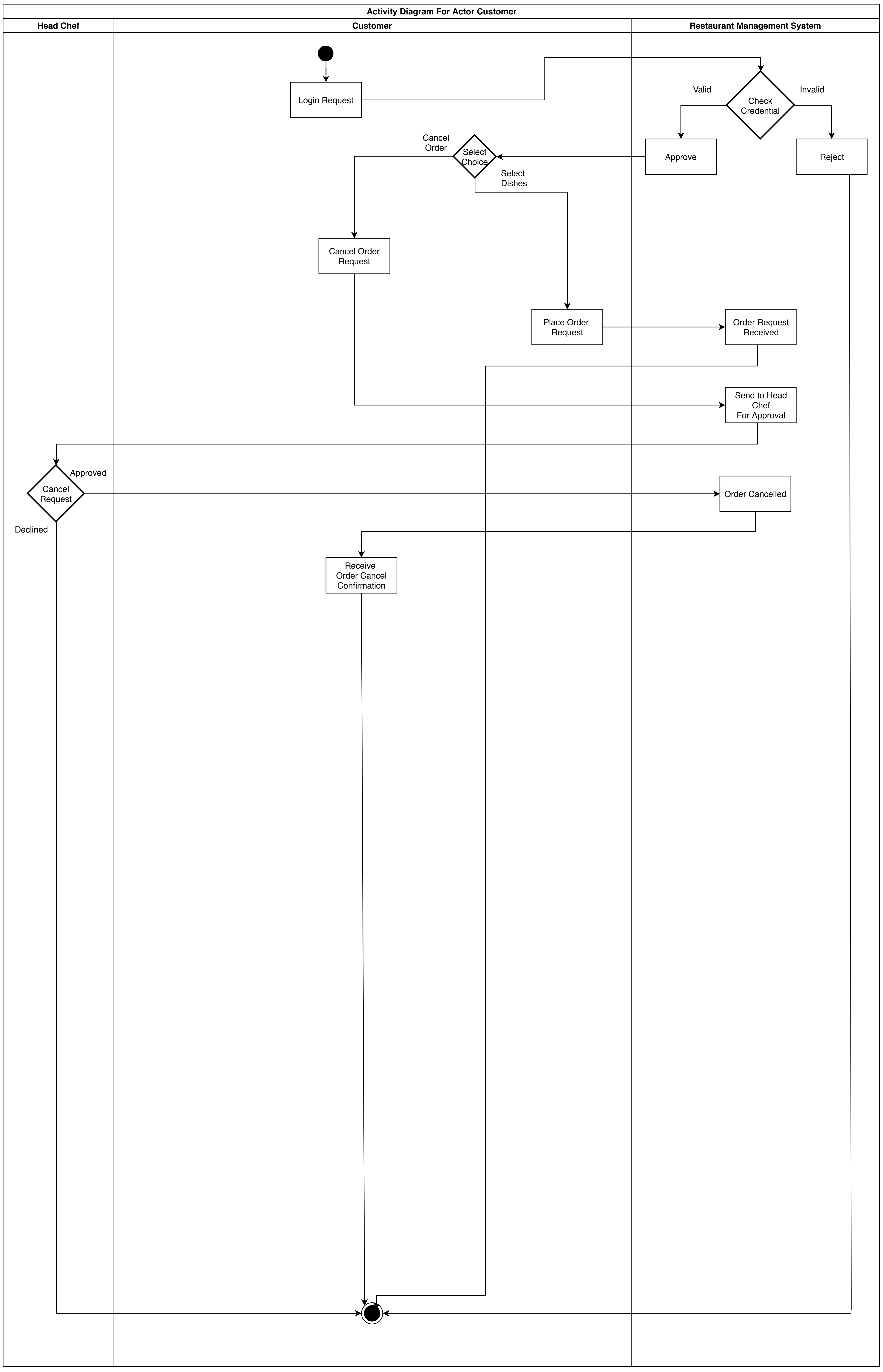
Hall Manager Activity Diagram :

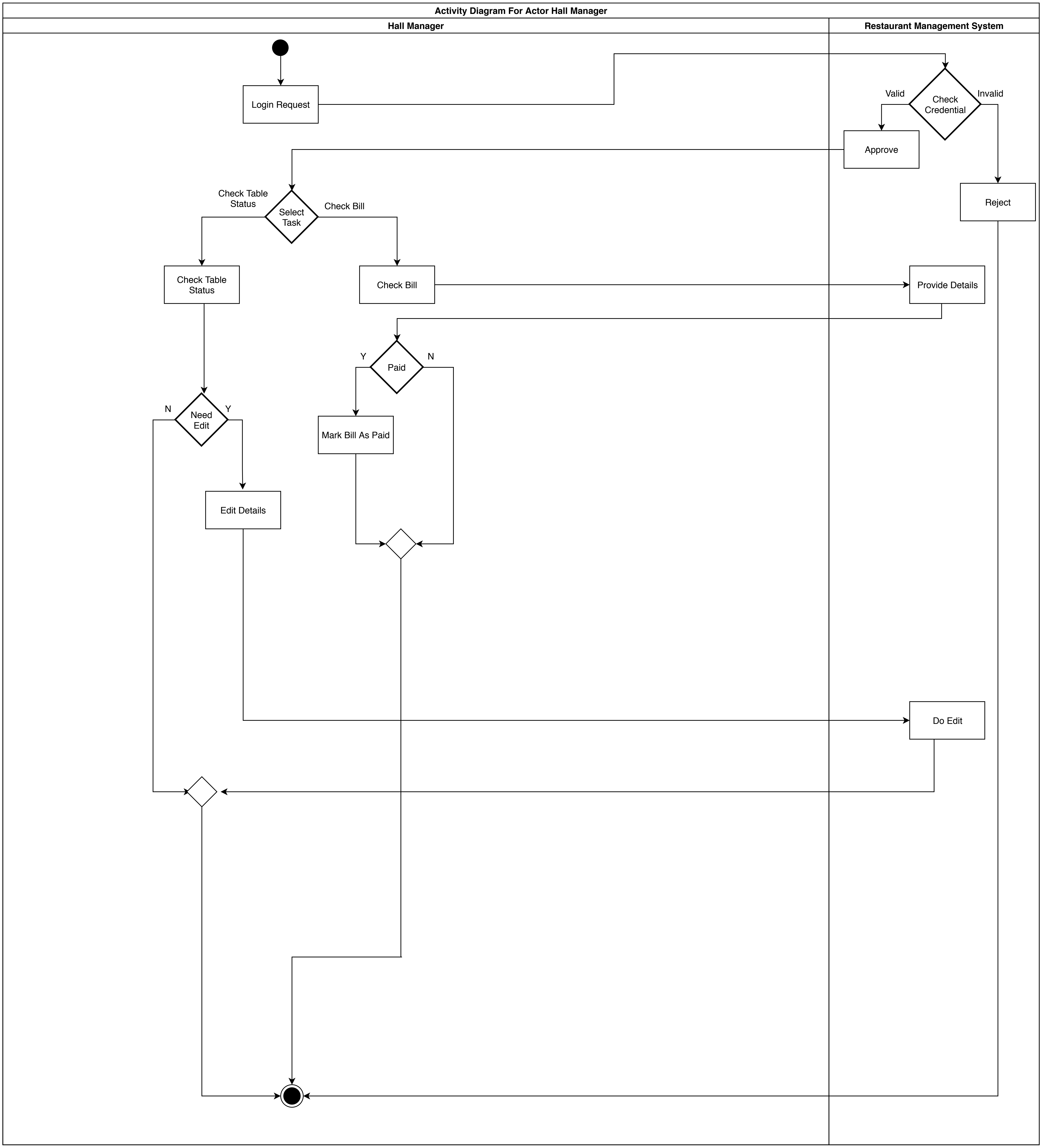
Like the customer, Hall Manager also have to login the system upon staring the system which gets verified by the RMS. If the credential are wrong then it rejects the requests and ends the activity otherwise on approval hall manager has to choose which task needs to be performed.

In the decision node if check table status is selected, then it goes to check table status activity and selects whether it needs to be edited or not in the decision node. If there needs any edit it is done with the help of RMS and both the node are merged in the merged node.

Same goes for the check bill activity where the details are provided by the RMS and the manager needs to decide is the payment is paid or not. If yes it marks it as paid and both the connection are joined with merge node like need edit decision.

In both the cases we used merge node as It is not used to synchronize concurrent flows but to accept one among several alternate flows. upon receiving any one of the flows from one of the sides it moves to next activity.


Activity Diagram For Actor Customer
Head Chef
Customer
Restaurant Management System
Login Request
Check Credential
Valid
Invalid
Approve
Reject
Select Choice
Cancel Order
Select Dishes
Cancel Order Request
Place Order Request
Order Request Received
Send to Head Chef For Approval
Cancel Request
Approved
Declined
Order Cancelled
Receive Order Cancel Confirmation


Activity Diagram For Actor Hall Manager
Hall Manager
Restaurant Management System
Login Request
Check Credential
Valid
Invalid
Approve
Reject
Check Table Status
Select Task
Check Bill
Check Table Status
Check Bill
Provide Details
Y
Paid
N
Mark Bill As Paid
N
Need Edit
Y
Edit Details
Do Edit

Data Flow Diagram

A data flow diagram (DFD) is a graphical representation of the movement of data between external entities, processes and data stores within a system.

For our system :

External Agents : Customer, Admin, Chef, Headchef, Manager

Datastores : Inventory, Employee, Order

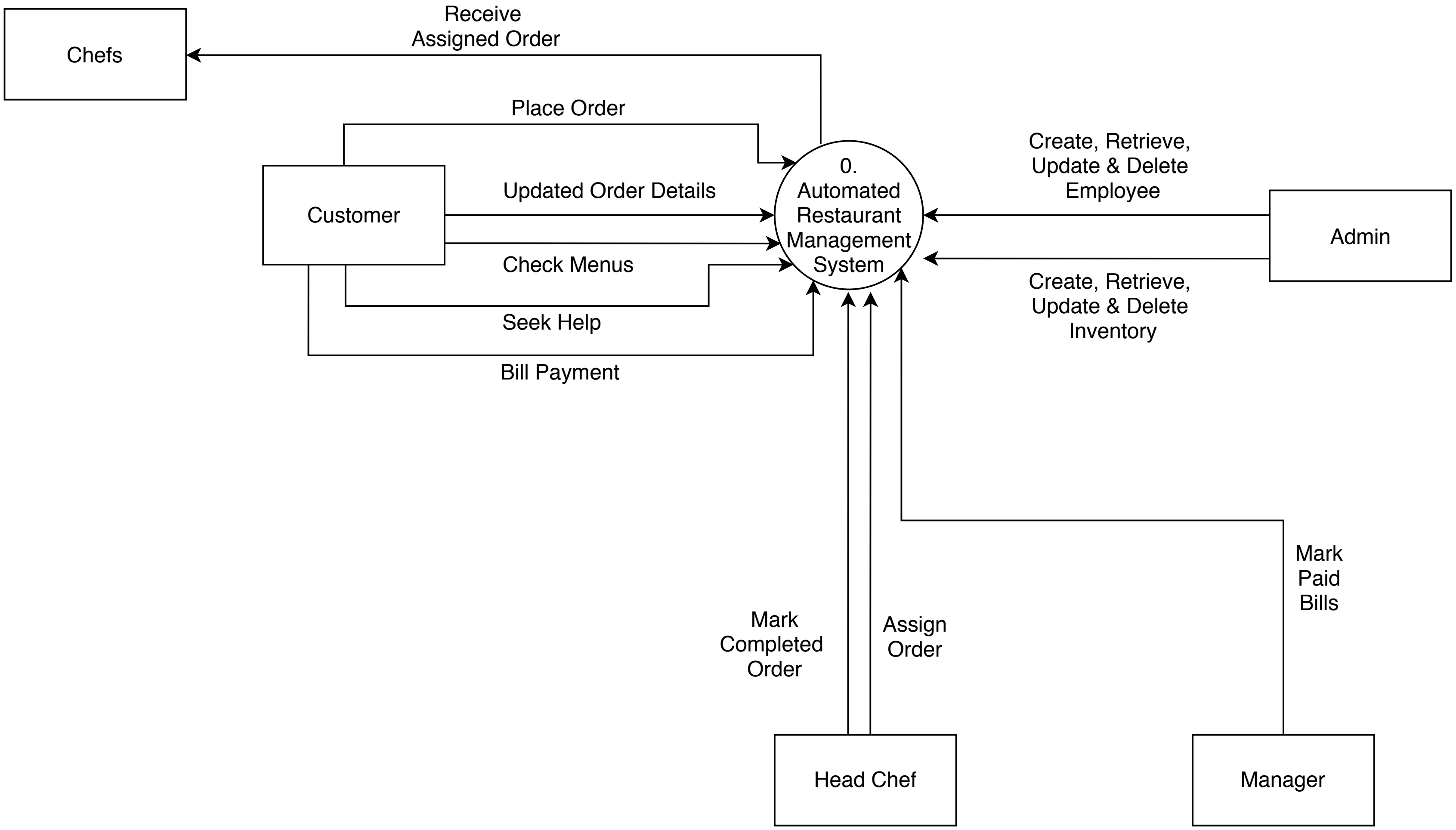
Processes : Update Order, Help Me, Order Items, Pay Bill, CRUD Inventory, CRUD Employee, Mark Completed Order, Assign Order, Mark Paid Bill and Calculate Employee Payment

These five External Agents are five types of users of this system. Through the ten processes, these five Agents perform different tasks.

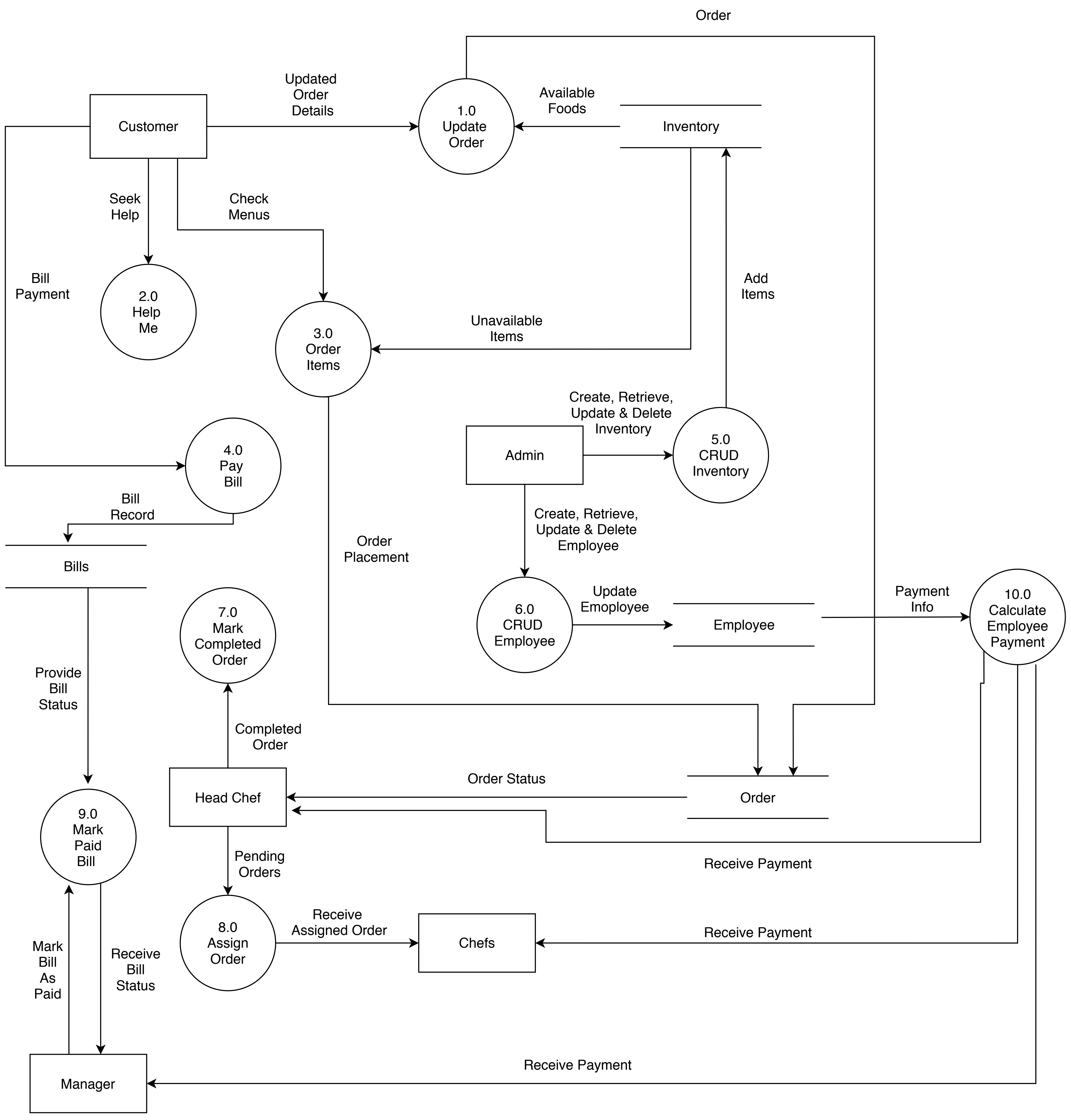
Level 0 or Context Diagram provides the overview of entire system. At this level, the system is considered as a single process. and we can identify external entities and data flow.

At Level 1 Diagram it is shown that all the processes that comprise a single process on the level 0 diagram. We demonstrate how information moves from and to each of these processes and shows in more detail the content of higher-level process. Level 1 diagrams may not be needed for all level 0 processes.

Data Flow Diagram : Level 0 [Context Diagram]



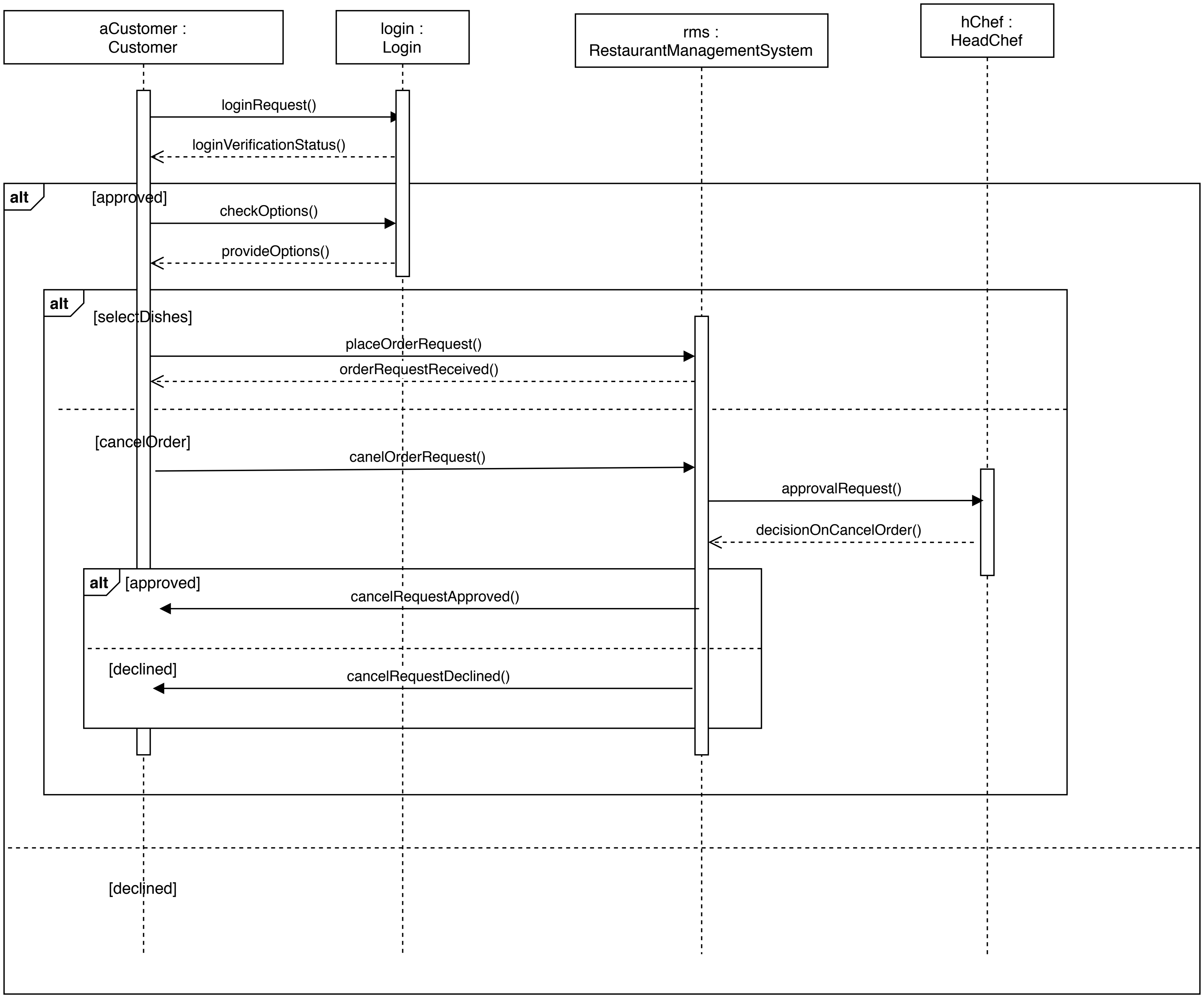
Data Flow Diagram : Level 1



Sequence Diagram

In the sequence diagram we have 4 objects as *aCustomer : Customer*, *login : Login, rms : RestaurantManagementSystem*, *hChef : HeadChef*. *aCustomer* sends a login request to the *login* object where an acknowledgement is also received by *aCustomer*. If the login request is approved then the system moves to the first alternate box where the customer requests for checking available options if not approved then its declined. Furthermore, the customer gets it reply regarding the available options. If the customer selects the *selectDishes* option then it sends request for placing order and also gets a corresponding reply for it. On the other hand, if the customer chooses *cancelOrder* option the request for cancelling the order is send to *rms* and *rms* will send it to *hChef* who decides & sends the decision to rms whether its approved or not and then *rms* will let the the *aCustomer* know about the decision.

Sequence Diagram



Conclusion

Due to time constraint issue of this semester, we didn't have the scope to implement it practically. So, in near future, we would like to implement this System practically and upgrade this to a Industry Grade System. Apart from that, we would like to launch this System & update it regularly.