Report Two: Part One

Project Website:
https://sites.google.com/scarletmail.rutgers.edu/restaurantautomation2018/home?authuser=1

Project Blog:
https://sites.google.com/scarletmail.rutgers.edu/restaurantautomation2018/blog

GROUP #4
Team Name: Little Bits
Team Members:

- Esraa Abdelmotteleb
- Manvi Agarwal
- John Alcantara
- Hagar Elshentenawy
- Lakshmi Sai Nithya Garudadri
- Kevin Honeker
- Zachary Joseph
- Fareen Pourmoussavian
- Dafna Shochat

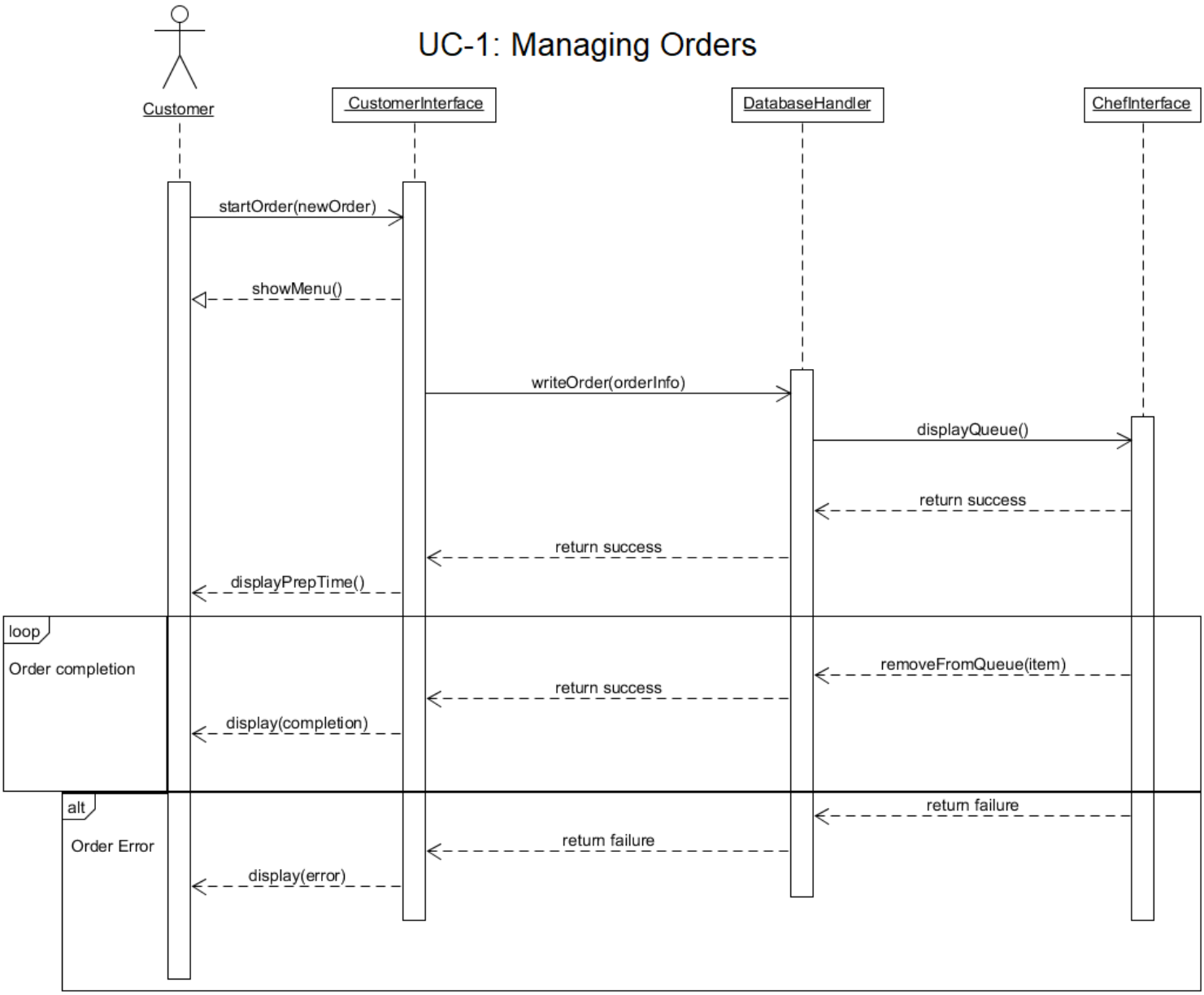
Contribution Breakdown

Section	Sub-Section	Pts	Esraa Abdelmotteleb	Manvi Agarwal	John Alcantara	Hagar Elshentenawy	Lakshmi Sai Nitya Garudadri	Kevin Honeker	Zachary Joseph	Fareen Pourmoussavian	Dafna Shochat	Teams
Project Management [18]	Document Merge	11	10%	10%	10%	10%	10%	10%	20%	10%	10%	Customer
	Project Coordination & Progress Report	5	0%	33%	11%	11%	11%	11%	0%	11%	11%	Employee
	Plan of Work	2	33%	0%	11%	11%	11%	11%	0%	11%	11%	Manager
Interaction Diagrams [30]	UML Diagrams	15	11%	11%	0%	16.50%	16.5%	11%	11%	22%	0%	
	Description of Diagram	15	11%	11%	22%	5.5%	5.5%	11%	11%	0%	22%	
Classes & Specifications [10]	Class Diagram	4	0%	33%	0%	33%	0%	0%	0%	33%	0%	
	Data Types & Operation Signatures	4	0%	33%	16.5%	33%	0%	0%	0%	16.5%	0%	
	Traceability Matrix	2	0%	0%	0%	0%	16.5%	33%	33%	0%	16.5%	
System Architecture & System Design [15]	Architectural Styles	4	0%	33%	0%	0%	0%	33%	0%	0%	33%	
	Identify Subsystems	2	33%	0%	17%	0%	0%	0%	0%	17%	33%	
	Mapping Subsystems to Hardware	2	33%	0%	0%	0%	33%	33%	0%	0%	0%	
	Persistent Data Storage	4	0%	25%	33%	0%	33%	0%	8%	0%	0%	
	Network Protocol	1	0%	50%	50%	0%	0%	0%	0%	0%	0%	
	Global Control Flow	1	0%	0%	0%	0%	33%	11%	33%	22%	0%	
	Hardware Requirements	1	10%	10%	20%	10%	10%	10%	10%	10%	10%	
Algoithms & Data Structures [4]	Algorithms	2	0%	33%	33%	17%	17%	0%	0%	0%	0%	
	Data Structures	2	0%	33%	0%	33%	0%	0%	0%	33%	0%	
User Interface Design & Implementation[11]	What Has Changed	6	0%	0%	11%	0%	0%	11%	33%	11%	33%	
	Ease to Use	5	16.50%	0%	0%	13%	0%	17%	16.50%	17%	20%	
Design of Tests [12]	Test Cases	4	33%	0%	0%	0%	33%	17%	0%	17%	0%	
	Coverage of Tests	4	33%	0%	0%	0%	33%	17%	0%	17%	0%	
	Integration Testing Strategy	4	0%	33%	33%	33%	0%	0%	0%	0%	0%	
	TOTAL	**100**	10.045	14.35	10.92	10.97	10.98	10.925	9.815	11.365	10.66	

Table of Contents

Interaction Diagram

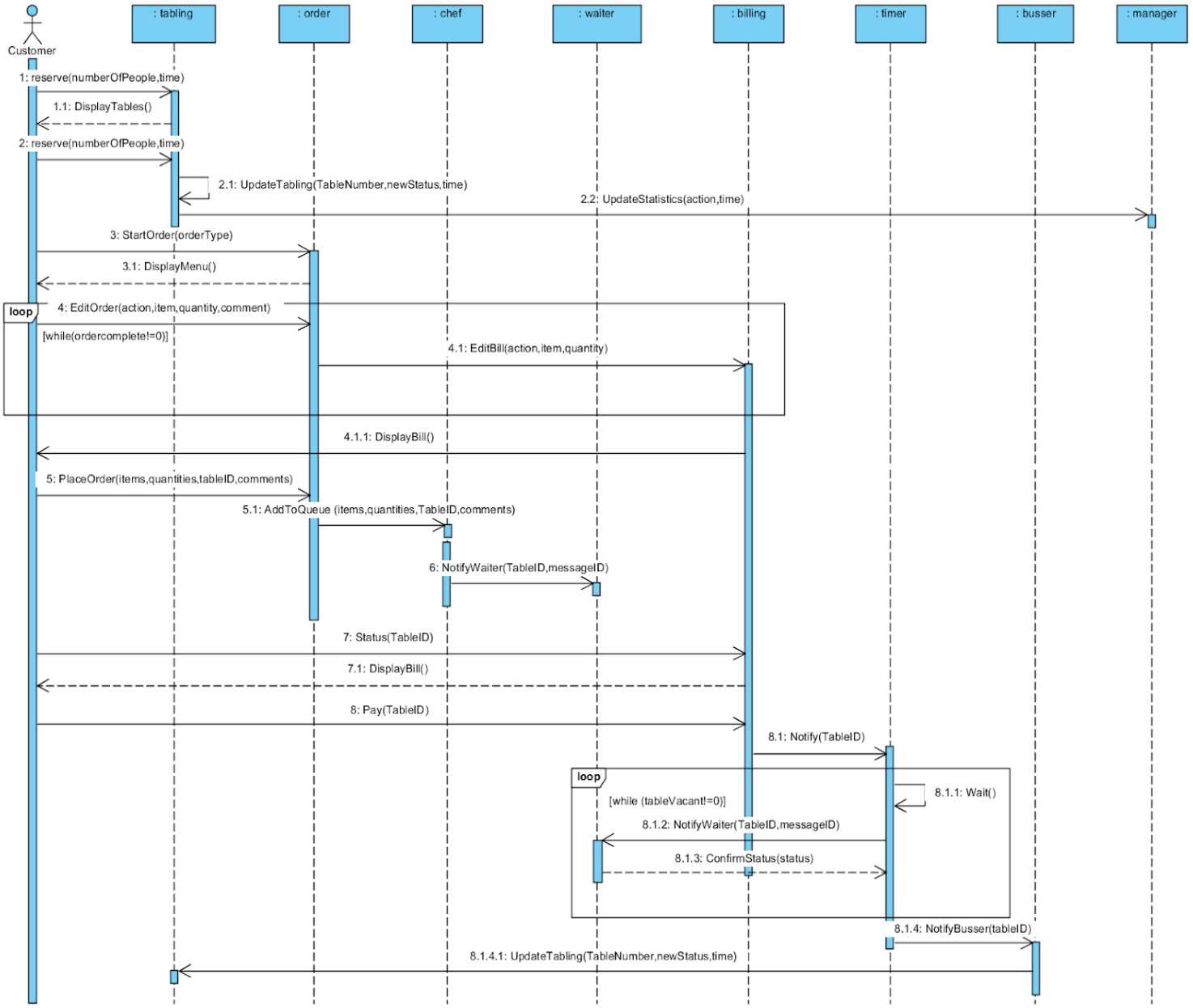
UC-1: Managing Orders



Explanation:

The customer places an order from the menu interface. Once the customer places the order, the order info gets sent to the chef's queue which is present on the chef's interface at all times. After the chef receives the order, a confirmation is sent back to the customer with an approximate time for completion. When the order is made, the chef send another notification back to the customer to to let them know that their order will be arriving shortly.

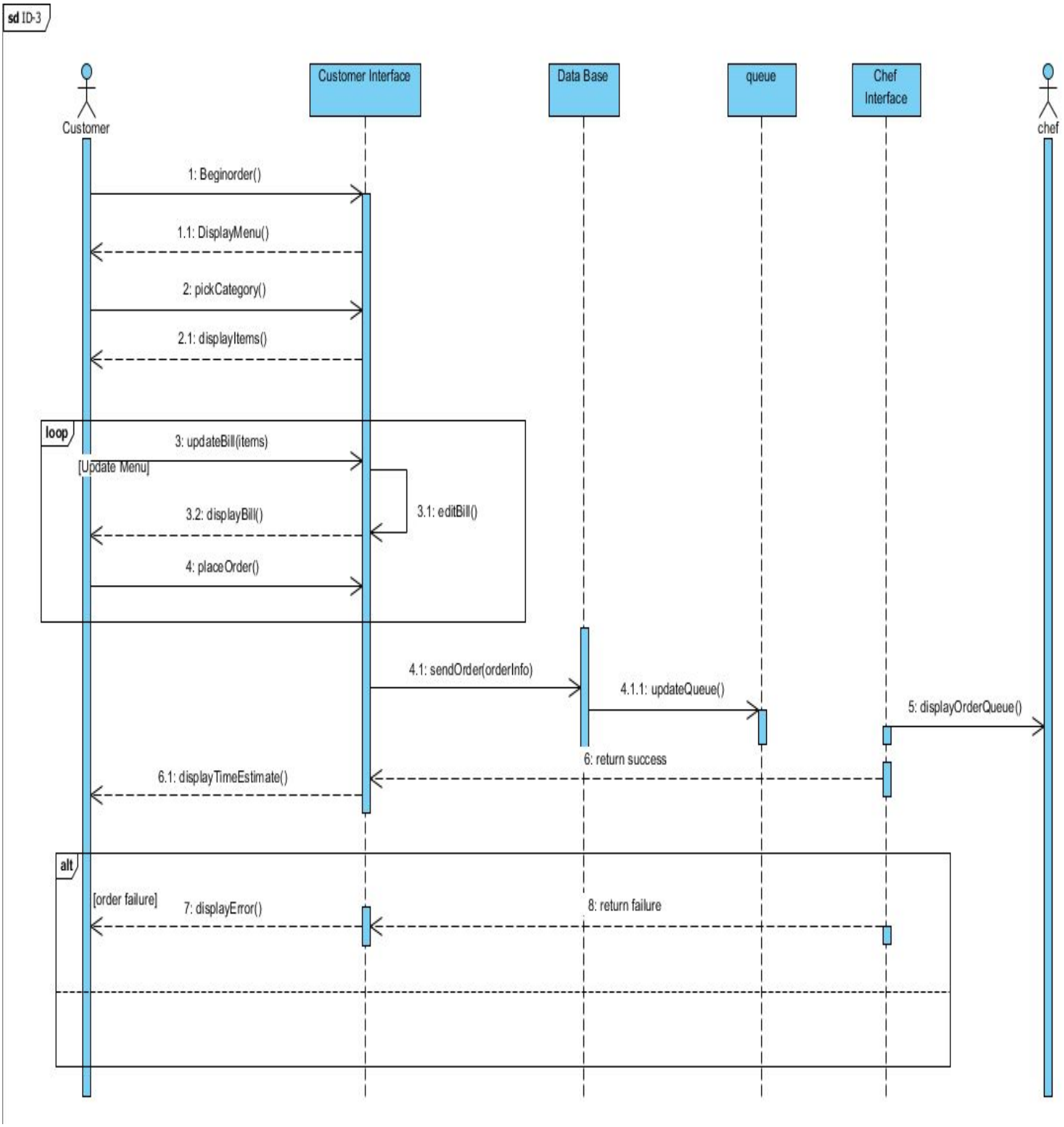
UC-2: Traffic Monitoring



Explanation:

The customer will go online to make a reservation. They click reserve to reserve a table. A map of the tables in the restaurant will show available tables marked as green. The customer will then choose the desired table and reserves it. The system will then update its records of which tables are available. Once the customer shows up, they can then be seated and can take a look at the menu. They can add or remove items from their order and they can attach notes to their order. Every time the order is edited, the bill for that order is also updated. When the customer is ready, they can place the order. The order is then added to the order queue and the chef is then notified about the order. When customers are done, they can pay their bill. Once the customers pay their bill and leave, there will be a timer which after a five-minute waiting period, the waiter can confirm if the customers left. If confirmed, then the table will be marked empty and dirty. If not confirmed, the timer will go for another five minutes until it is confirmed. This will notify the busser that a table needs to be cleaned. Once the table is cleaned, the busser will mark that it is ready for use, which will update the table map. Then customers ready to place reservations or employees looking for available tables can see open tables.

UC-3: Customer Ordering



Explanation:

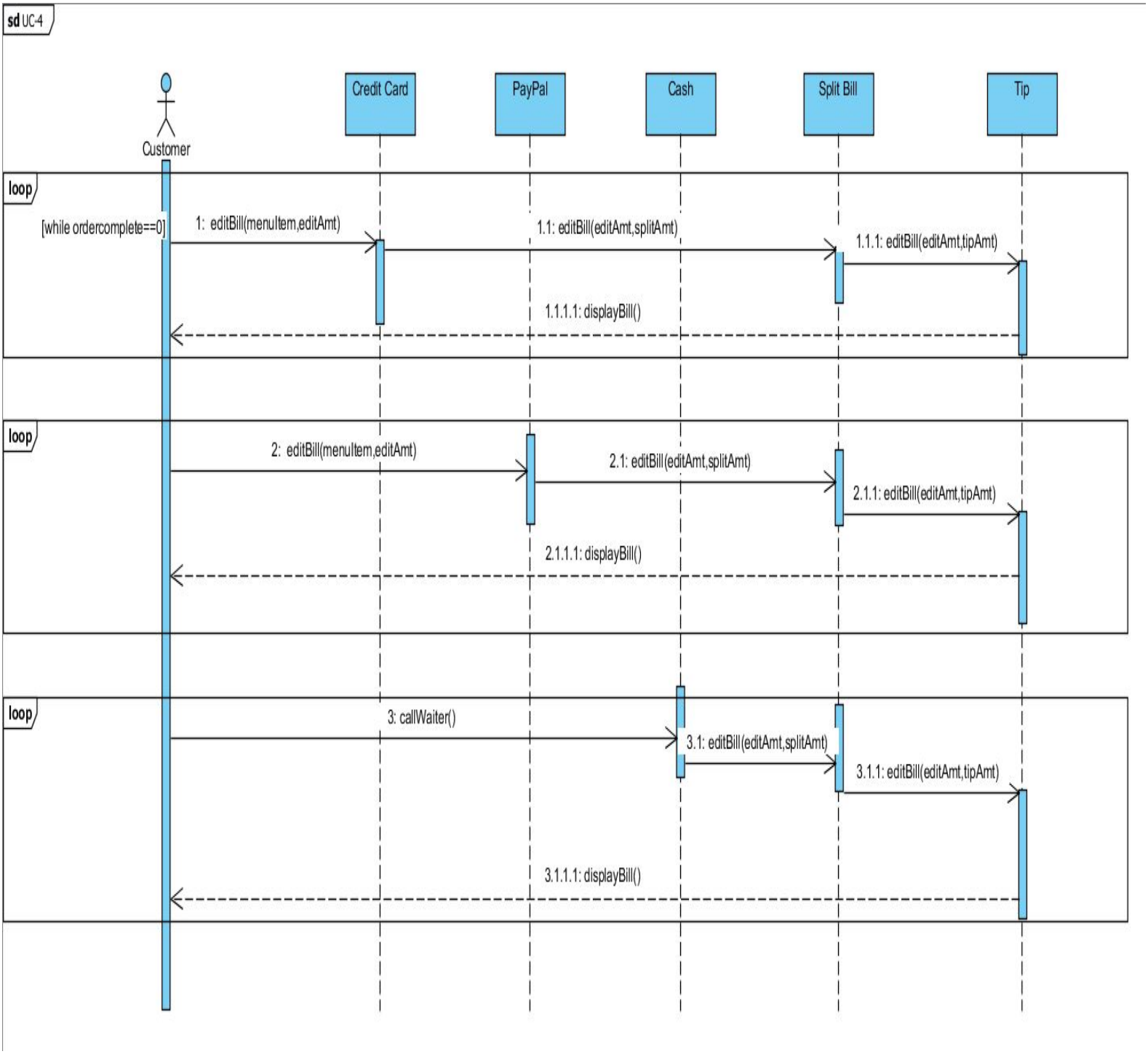
The main modules are menu display, creating an order, and playing games.

This case begins when the customer or user sits down at the table and initiates interaction with the tablet. Upon beginning an order the customer is shown several options including: Order Food, Play Games, and SOS. Each button has a very important function in the dining experience.

Ordering food is the main priority of this user case, this covers everything from choosing an entree to receiving the food, and everything in between. Upon selection of the “Order Food” option the system displays the menu categories such as Pasta Entrees, Drinks, Dessert, and Seafood. The customer can see the menu categories and browse them. Once the customer selects a category then their specific item list is displayed by the system for the customer to view. The customer can then select what items they want from the menu and write in any specific requests associated with the item such as food allergies or meat tenderness. Once the customer is ready to place the order for that item they select the “Order” button on the display. This will send the item and its specifications to the chef, the customer will then receive a notification stating how much time they should expect to wait till their food is ready.

Additional to the ordering the customer also has an “SOS” option where the customer can send a signal to their waiter indicating that they need help or have an issue that needs to be addressed. The waiter then receives the signal and can choose to respond to it from their personal tablet. Another option available to the customer is the “Play Games” option where customers can choose to play a free game while they wait for their food. Upon selecting a game on the display, the customer can begin to play and can choose to stop the game an any point.

UC-4: Payment Process



Explanation:

This is a continuation of the dining experience when the user is ready to pay their bill and leave the establishment.

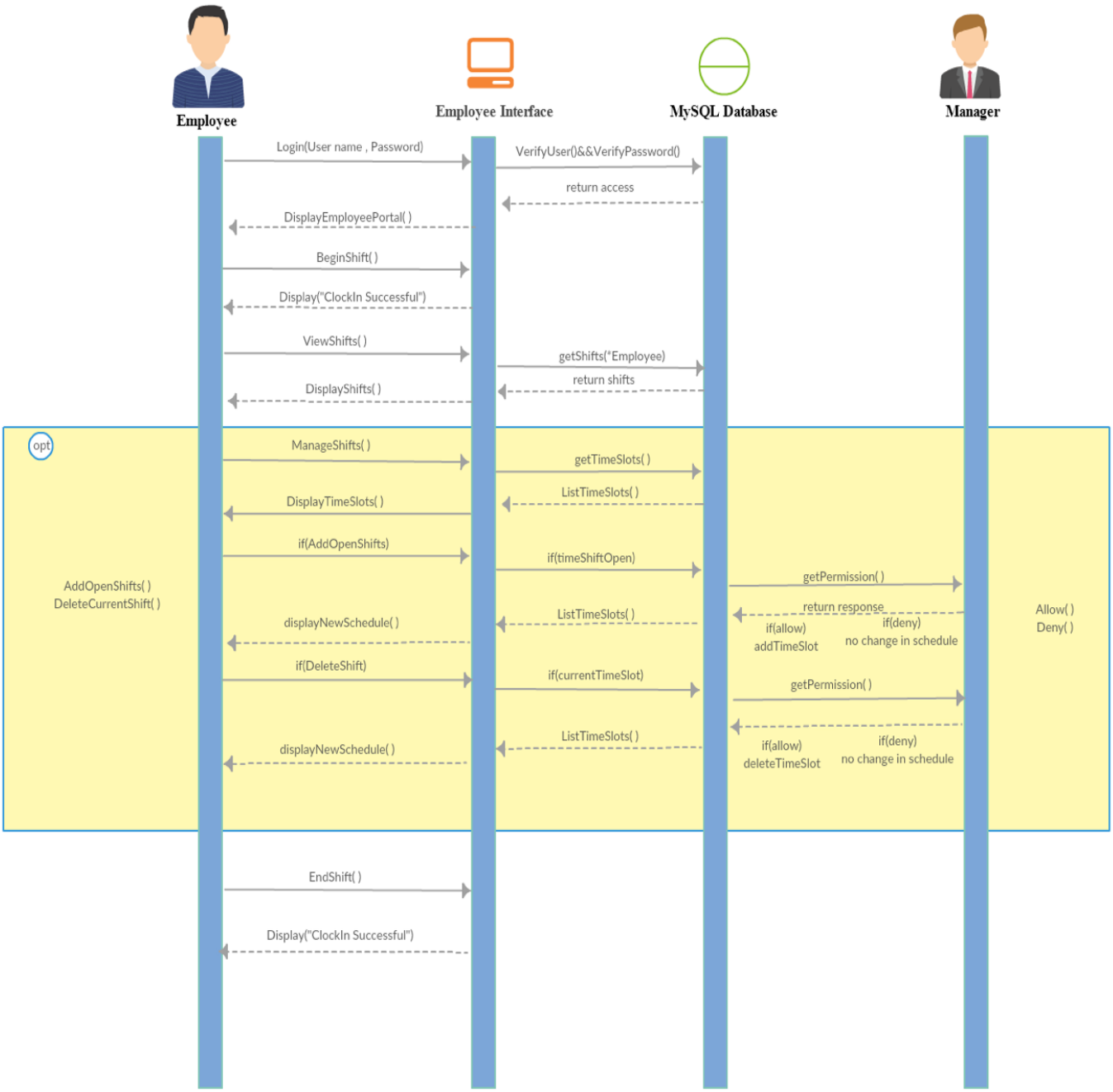
The customer can choose to pay their bill at any point after placing their food order. Most dining individuals do this once they have finished their food, this is convenient because here the customer can select if they need any takeout boxes if they have any food leftover that I want to take home. Once the customer is ready to pay they can select the option to pay their bill. At this point the display with show three methods to pay the bill: credit card, PayPal, or with cash. If the customer selects credit card the display with indicate that the customer is to swipe their card through a port on the tablet and indicate whether they are using Credit or Debit. If the customer selects to pay via PayPal then the following screen will ask the customer to enter their PayPal user ID and password.

If the customer selects to pay in cash, then the customer will be faced with a message stating that their waited is on their way. This will also send a message to the waiter's personal tablet saying that the table is ready to pay in cash. Once the waiter arrives at the table the customers can pay the waiter directly and receive change if needed.

Upon selecting a form of payment the customer is also able to select if they require any takeout boxes for any leftover food. They can select how many boxes they need depending on how many food orders were places. The waiters is notified by the customer's selection and will bring the boxes over when collecting cash payments or after Credit Card or PayPal payments have been processed.

If the customer wished to split the bill, then they may do so by selecting that option. The customer can then select which items they wish to move to the second bill. When paying for each bill a different payment method can be used on each bill. If the customer wished to split the bill more, then they can choose to call the waiter over and he/she can split the bill further.

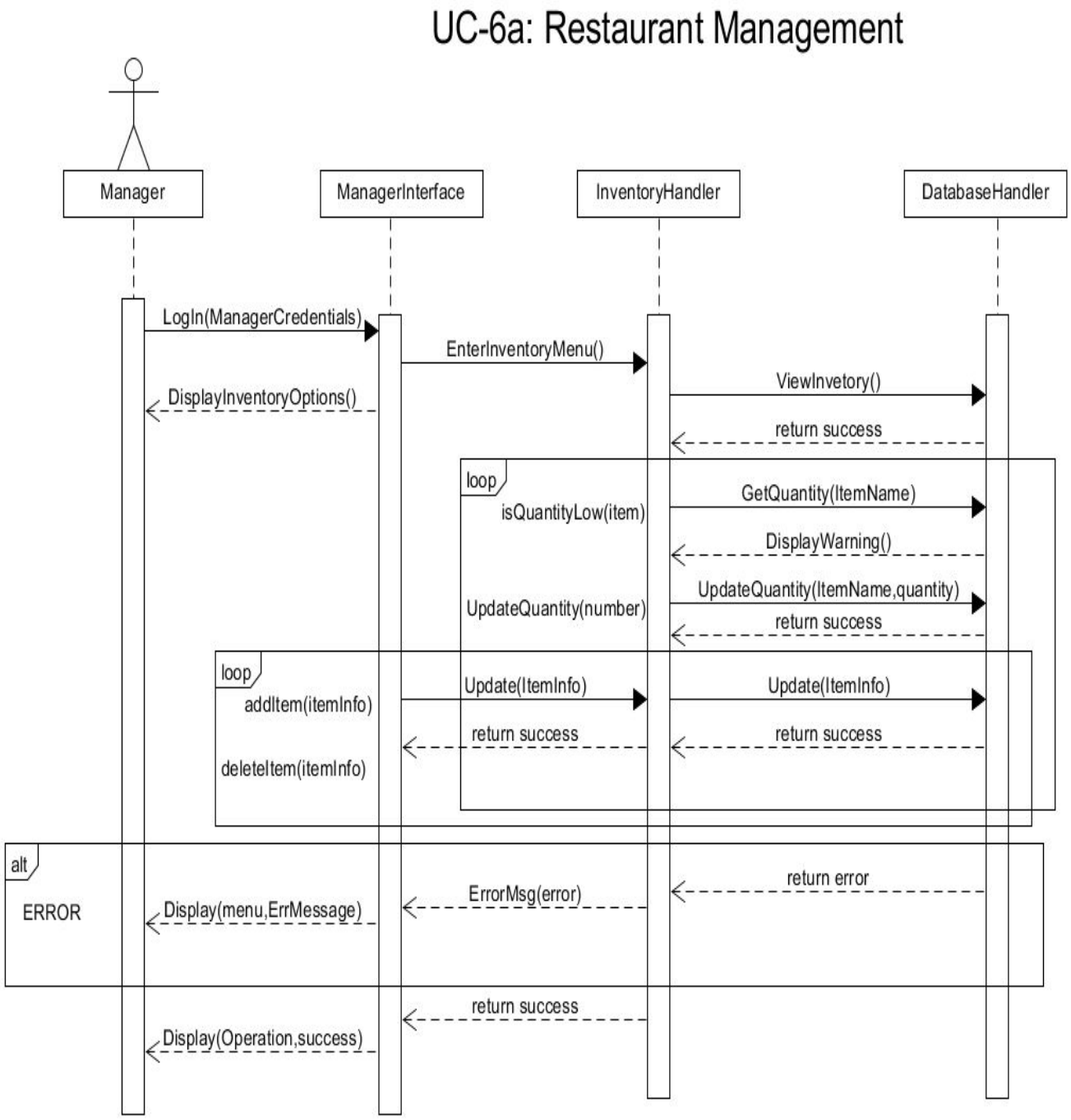
UC-5: Managing Employees



Explanation:

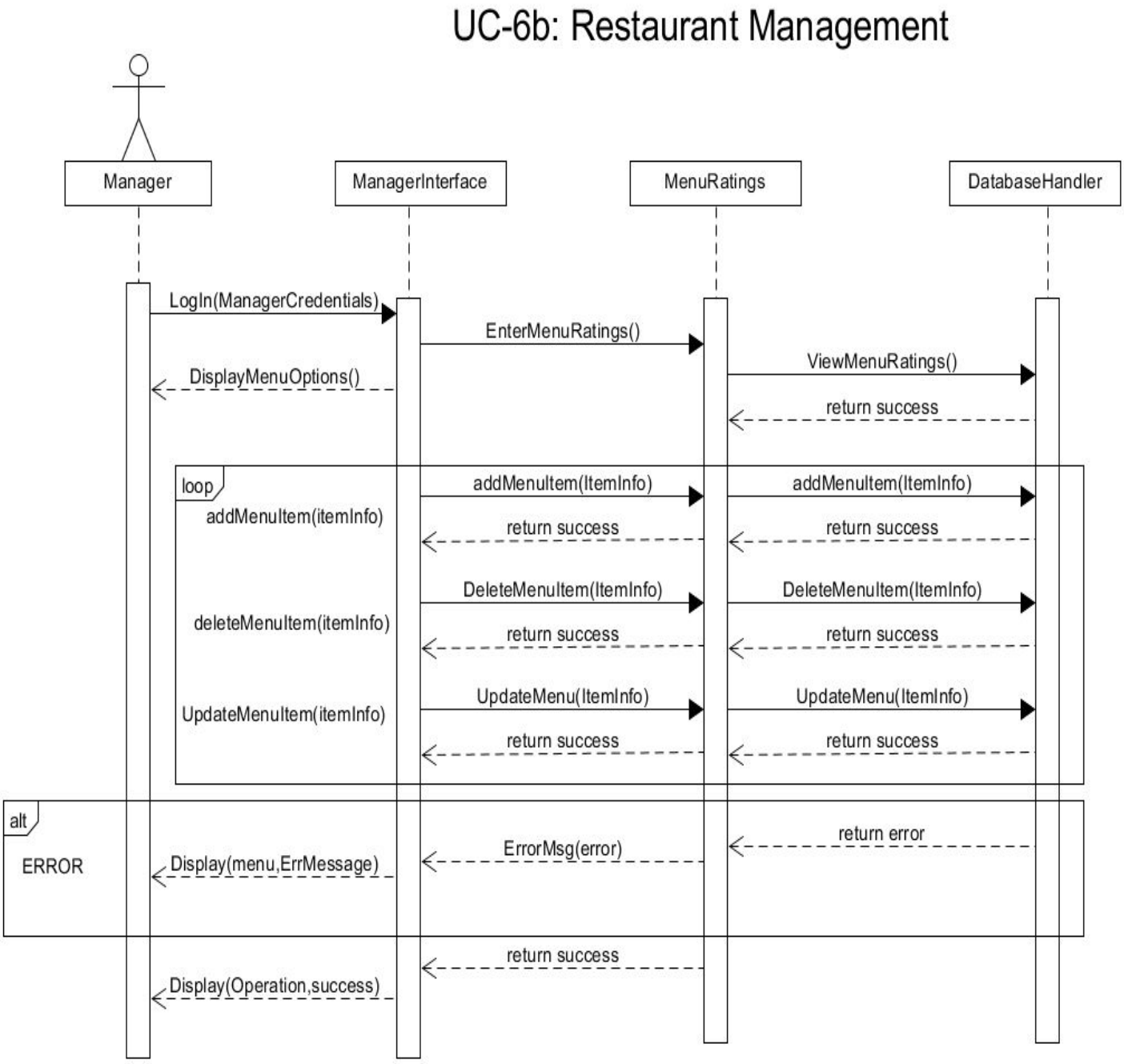
The employee (waiter, busser or chef) will first log into the employee portal. The employee can login by inputting their user identification on the log in page. Once they put in their identification code, they will be automatically clocked in. Once in the portal the employee can view their portal and all the available options for their position. There will also be a tab in the portal for schedule management. In this tab, employees can view shifts and also put preferences for future shifts. In the tab the employee will see open shifts that need to be assigned. Once the employee inputs their preference, the manager will be able to see all of the options and can create the schedule based on the employee's availability. Once the schedule is created, it will be posted so employees can see the new schedule and can confirm or deny.

UC-6: Managing Restaurant



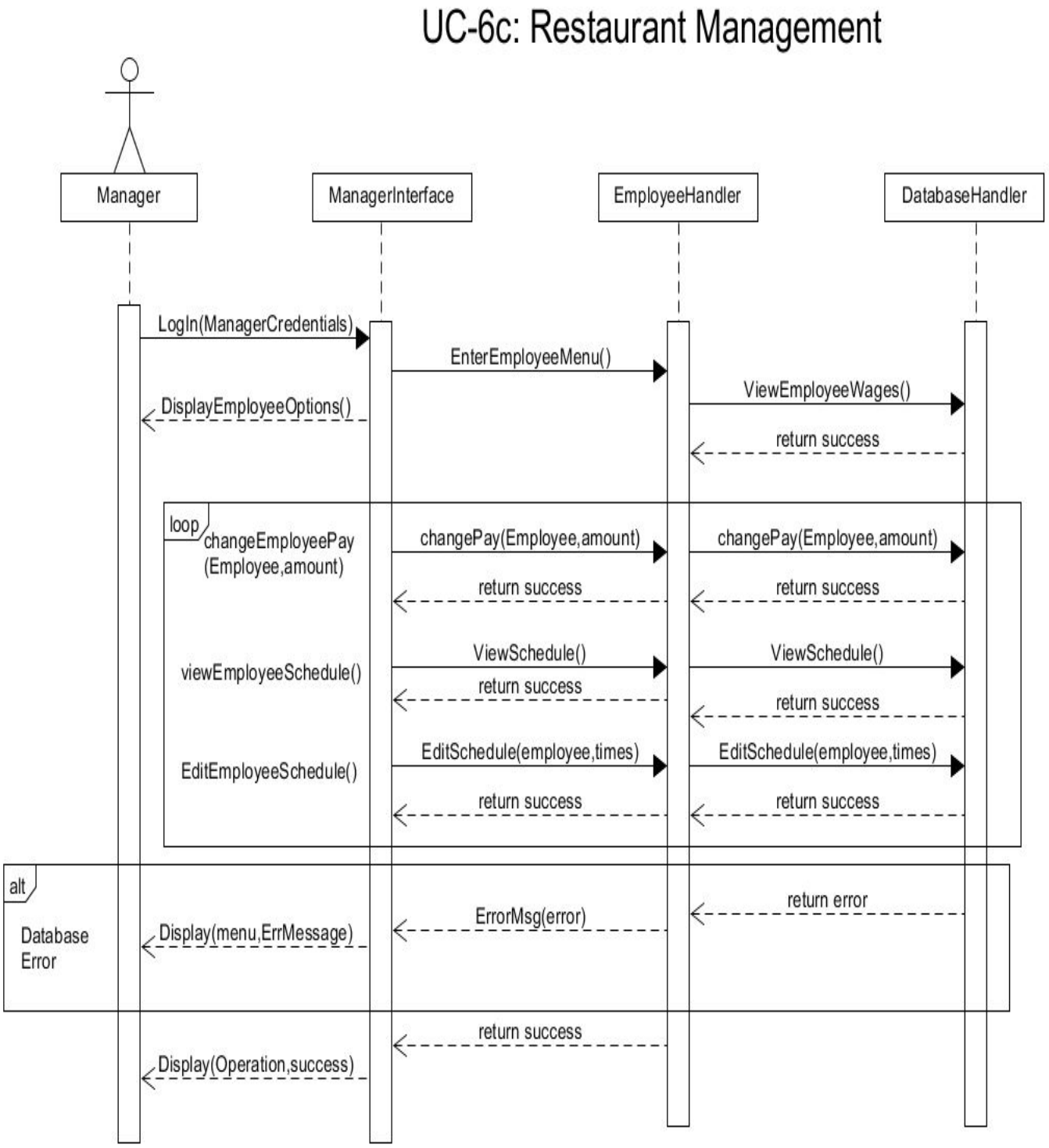
Explanation:

This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the inventory menu. After the manager prompts the system to display the inventory menu, they will be allowed to add or delete items from the inventory as is needed. While viewing the inventory, the system will display to the manager a warning if any of the items in the inventory is below a given threshold in quantity. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.


UC-6b: Restaurant Management
Manager
ManagerInterface
MenuRatings
DatabaseHandler
LogIn(ManagerCredentials)
EnterMenuRatings()
ViewMenuRatings()
DisplayMenuOptions()
return success
loop
addMenuItem(itemInfo)
addMenuItem(ItemInfo)
addMenuItem(ItemInfo)
return success
return success
DeleteMenuItem(ItemInfo)
DeleteMenuItem(ItemInfo)
deleteMenuItem(itemInfo)
return success
return success
UpdateMenu(ItemInfo)
UpdateMenu(ItemInfo)
UpdateMenuItem(itemInfo)
return success
return success
alt
ERROR
return error
ErrorMsg(error)
Display(menu,ErrMessage)
return success
Display(Operation,success)

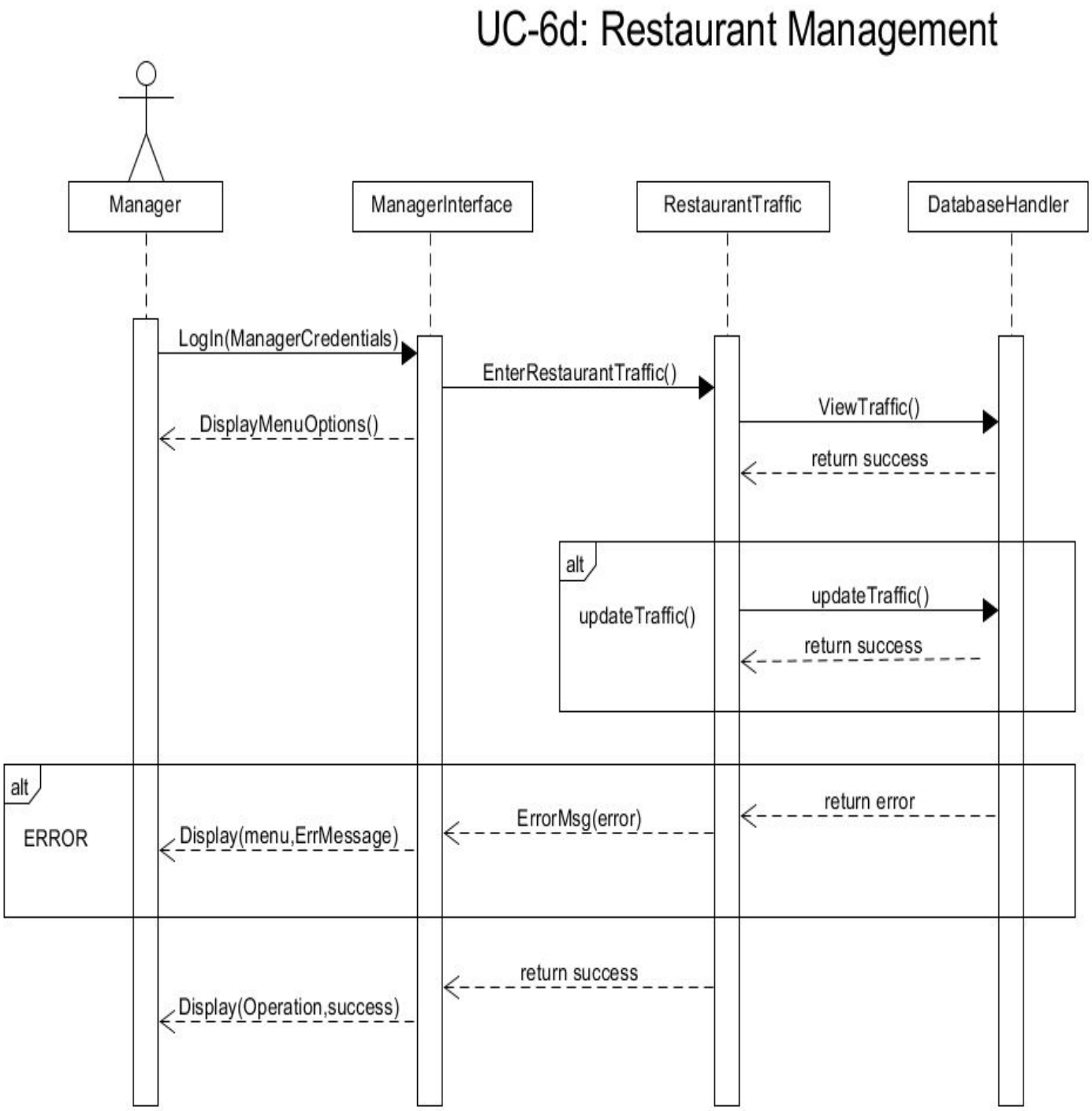
Explanation:

This diagram outlines the process the program goes through to display the restaurant's menu to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the restaurant's menu and editing options for the menu. From there, the manager may decide to view the ratings of each of the restaurant's menu items, or decide to edit the list of items on the menu itself. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.


UC-6c: Restaurant Management
Manager
ManagerInterface
EmployeeHandler
DatabaseHandler
LogIn(ManagerCredentials)
EnterEmployeeMenu()
ViewEmployeeWages()
DisplayEmployeeOptions()
return success
loop
changeEmployeePay
(Employee,amount)
changePay(Employee,amount)
changePay(Employee,amount)
return success
return success
viewEmployeeSchedule()
ViewSchedule()
ViewSchedule()
return success
return success
EditEmployeeSchedule()
EditSchedule(employee,times)
EditSchedule(employee,times)
return success
return success
alt
Database
Error
return error
ErrorMsg(error)
Display(menu,ErrMessage)
return success
Display(Operation,success)

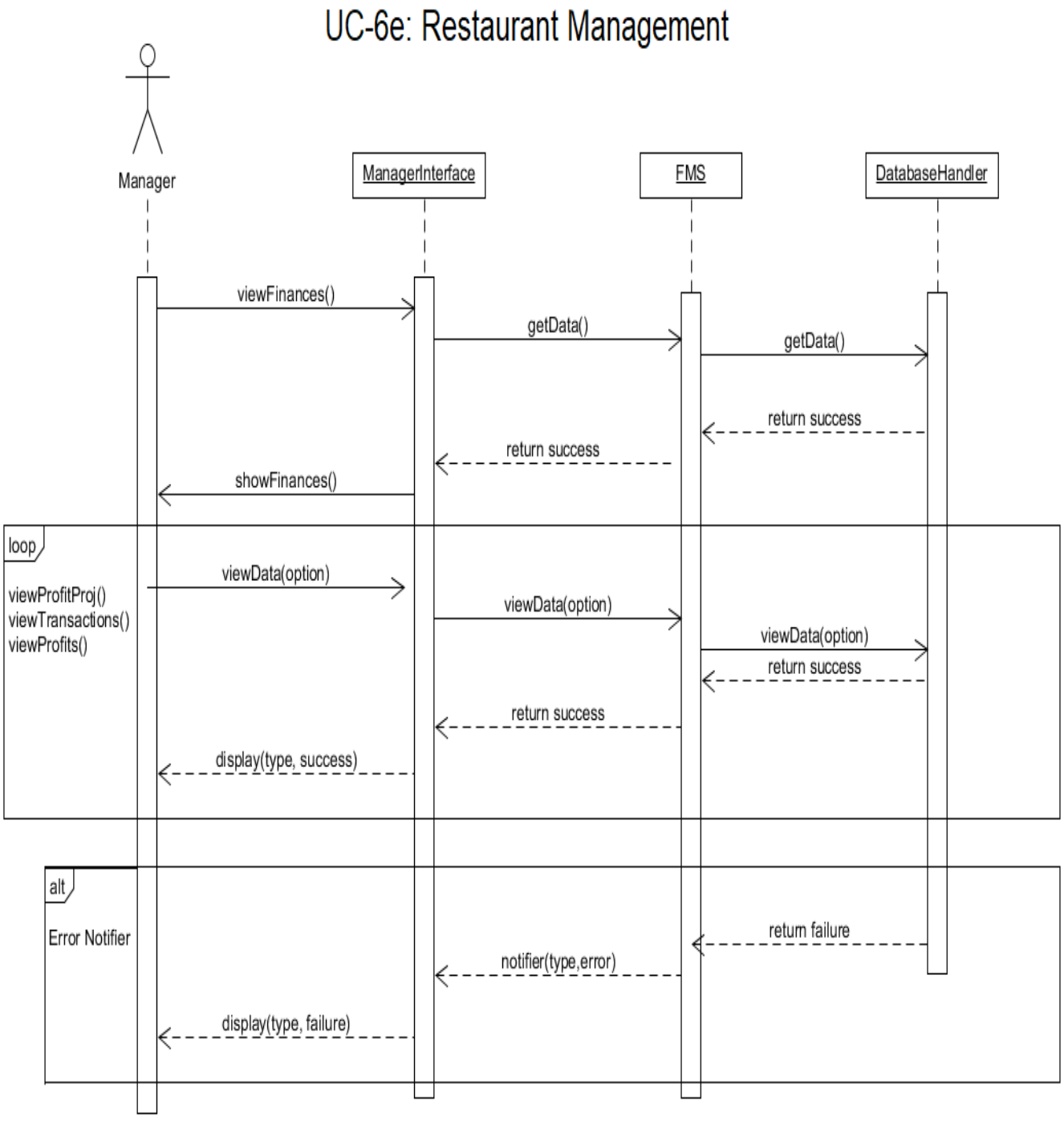
Explanation:

This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the employee management menu. From this point, the manager can decide to view the complete schedule of all employee shifts, to change the schedule of employee shifts, or to change a given employee's rate of pay. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.


UC-6d: Restaurant Management
Manager
ManagerInterface
RestaurantTraffic
DatabaseHandler
LogIn(ManagerCredentials)
EnterRestaurantTraffic()
ViewTraffic()
DisplayMenuOptions()
return success
alt
updateTraffic()
updateTraffic()
return success
alt
ERROR
Display(menu,ErrMessage)
ErrorMsg(error)
return error
return success
Display(Operation,success)

Explanation:

This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the floor traffic of the restaurant. The database will continually update the image of the floor traffic and continue to display said traffic to the requesting manager. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.


UC-6e: Restaurant Management
Manager
ManagerInterface
FMS
DatabaseHandler
viewFinances()
getData()
getData()
return success
return success
showFinances()
loop
viewProfitProj()
viewTransactions()
viewProfits()
viewData(option)
viewData(option)
viewData(option)
return success
return success
display(type, success)
alt
Error Notifier
return failure
notifier(type,error)
display(type, failure)

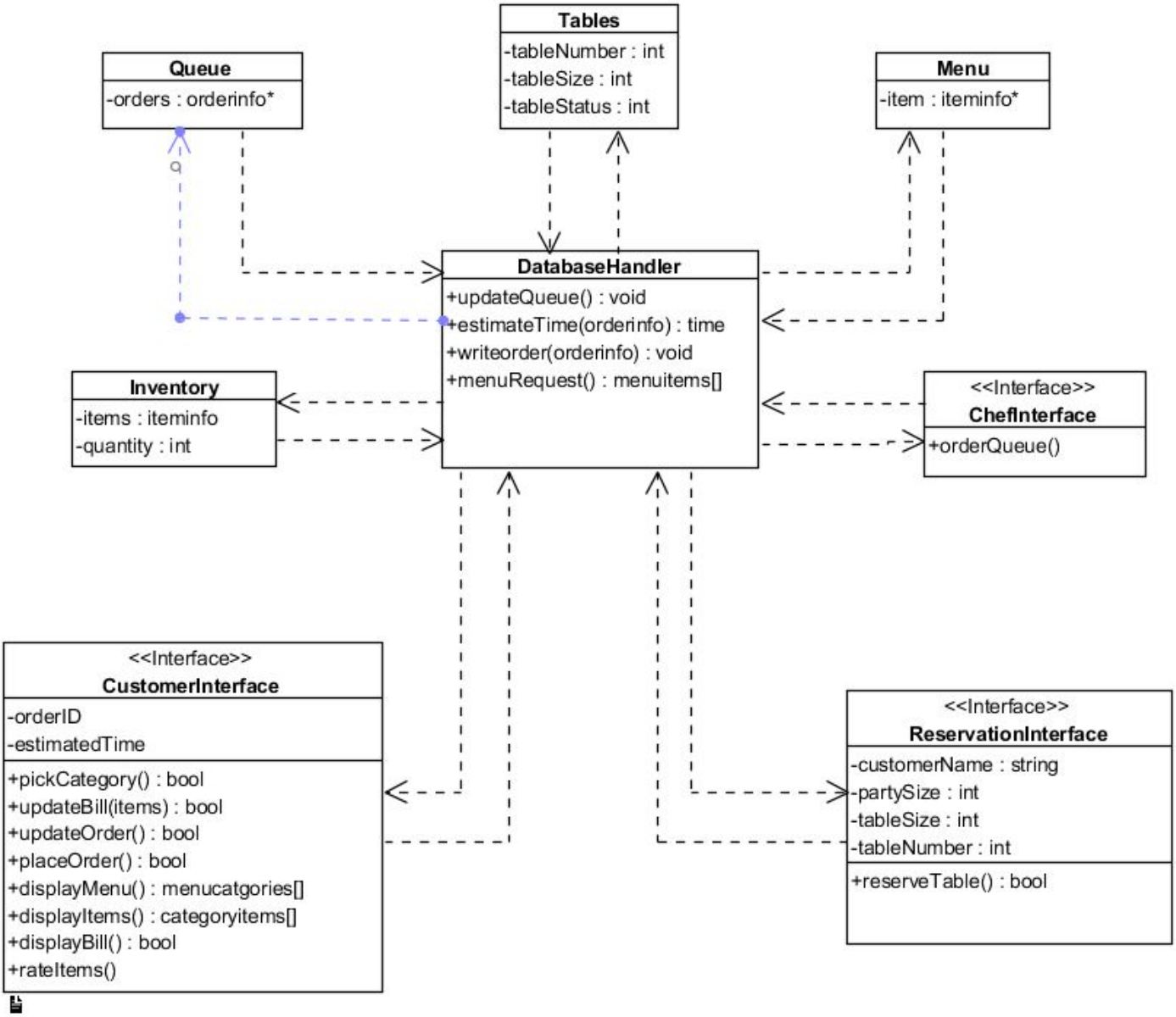
Explanation:

From the Manager interface, the manager can access the restaurant's financial records. The request is sent to the Financial Management System (FMS) where the manager can view profits, the transaction history, and a projection of future profits based on current data. The database is updated on a regular basis to ensure the manager receives up-to-date information.

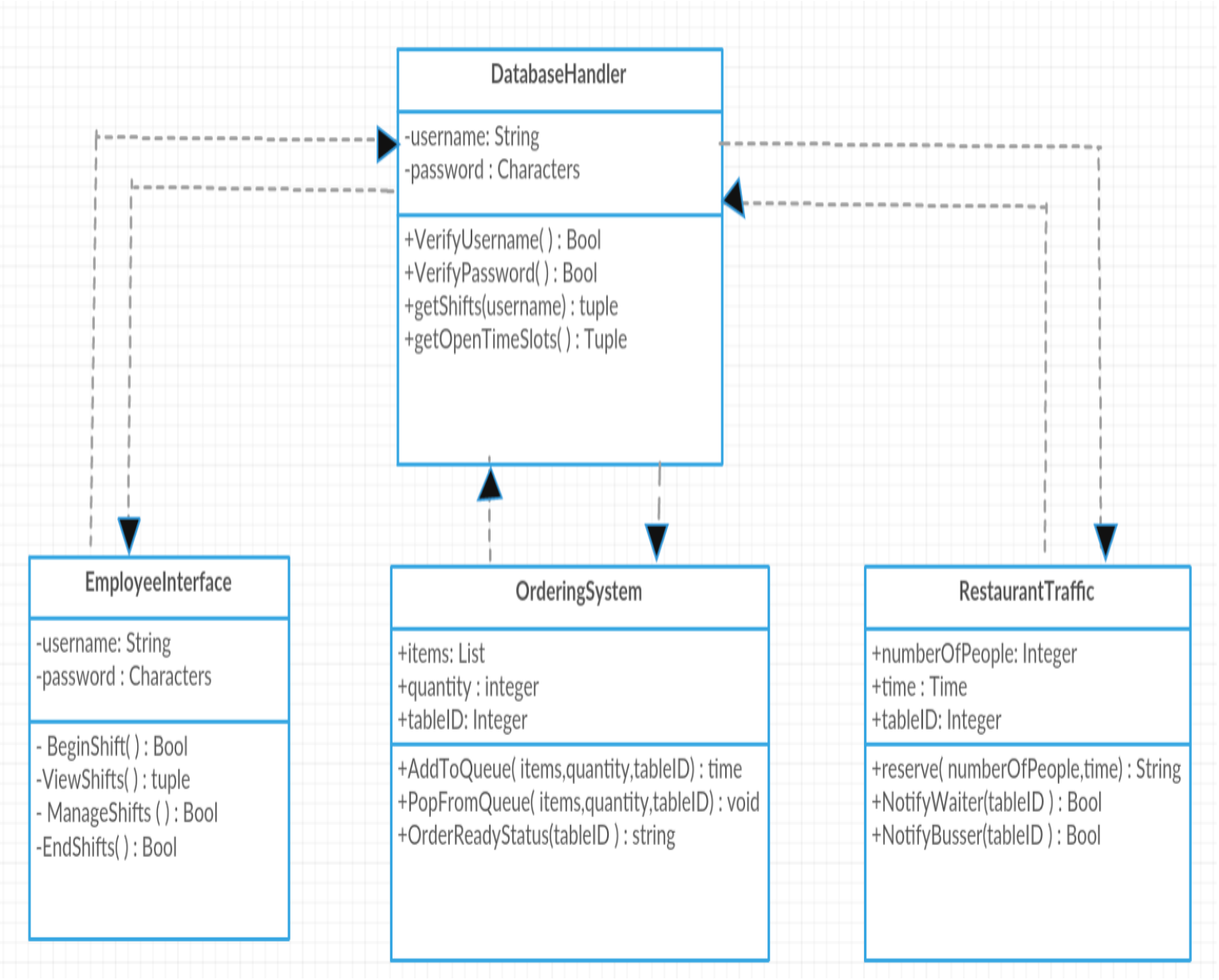
Class Diagram and Interface Specifications

Class Diagram

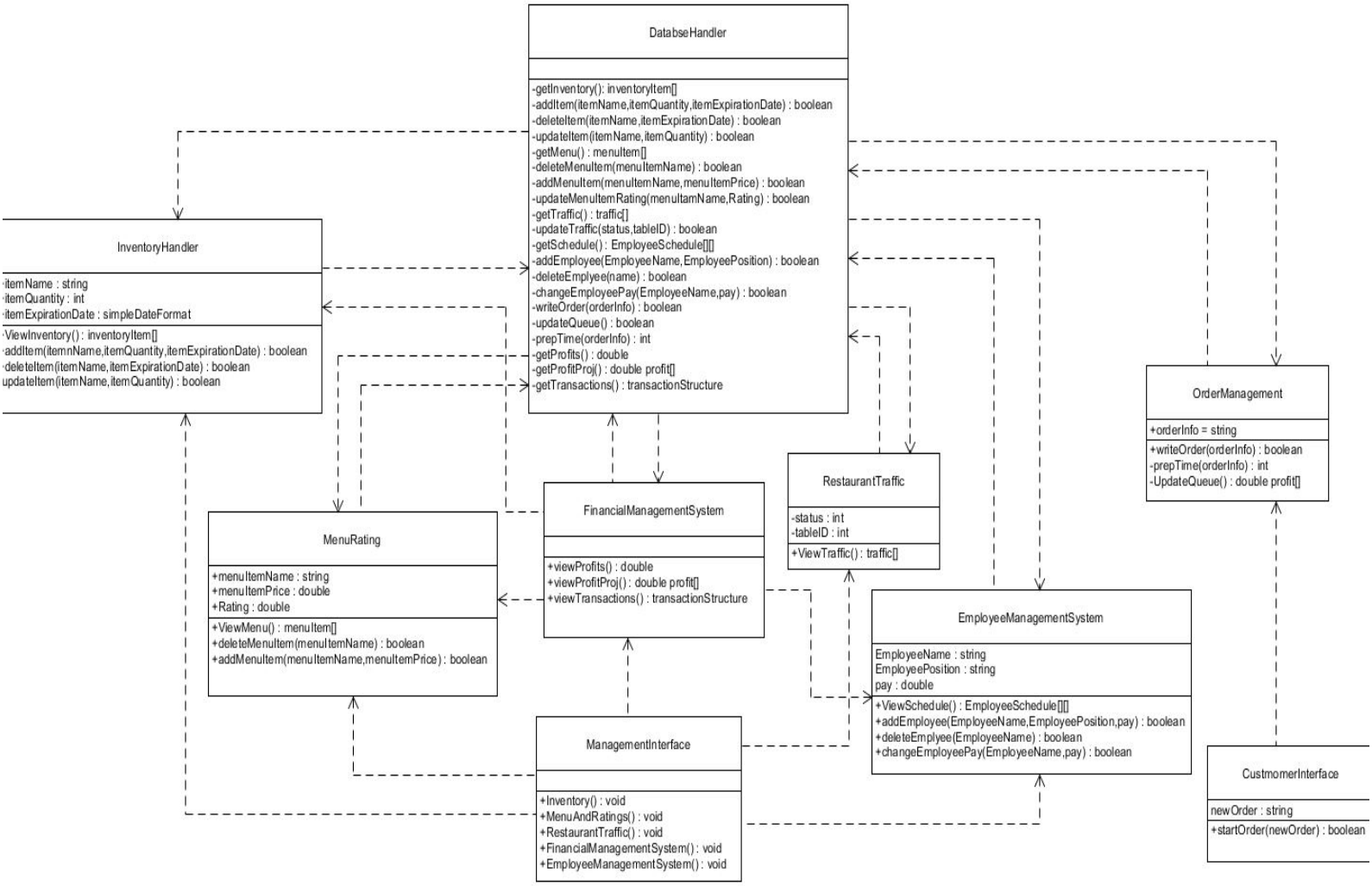
Customer:



Employees:


DatabaseHandler
-username: String
-password : Characters
+VerifyUsername( ) : Bool
+VerifyPassword( ) : Bool
+getShifts(username) : tuple
+getOpenTimeSlots( ) : Tuple
EmployeeInterface
-username: String
-password : Characters
- BeginShift( ) : Bool
-ViewShifts( ) : tuple
- ManageShifts ( ) : Bool
-EndShifts( ) : Bool
OrderingSystem
+items: List
+quantity : integer
+tableID: Integer
+AddToQueue(items,quantity,tableID) : time
+PopFromQueue(items,quantity,tableID) : void
+OrderReadyStatus(tableID) : string
RestaurantTraffic
+numberOfPeople: Integer
+time : Time
+tableID: Integer
+reserve(numberOfPeople,time) : String
+NotifyWaiter(tableID) : Bool
+NotifyBusser(tableID) : Bool

Manager:


DatabseHandler
-getInventory(): inventoryItem[]
-addItem(itemName,itemQuantity,itemExpirationDate) : boolean
-deleteItem(itemName,itemExpirationDate) : boolean
-updateItem(itemName,itemQuantity) : boolean
-getMenu() : menuItem[]
-deleteMenuItem(menuItemName) : boolean
-addMenuItem(menuItemName,menuItemPrice) : boolean
-updateMenuItemRating(menuItamName,Rating) : boolean
-getTraffic() : traffic[]
-updateTraffic(status,tableID) : boolean
-getSchedule() : EmployeeSchedule[][]
-addEmployee(EmployeeName,EmployeePosition) : boolean
-deleteEmplyee(name) : boolean
-changeEmployeePay(EmployeeName,pay) : boolean
-writeOrder(orderInfo) : boolean
-updateQueue() : boolean
-prepTime(orderInfo) : int
-getProfits() : double
-getProfitProj() : double profit[]
-getTransactions() : transactionStructure
InventoryHandler
itemName : string
itemQuantity : int
itemExpirationDate : simpleDateFormat
ViewInventory() : inventoryItem[]
addItem(itemnName,itemQuantity,itemExpirationDate) : boolean
deleteItem(itemName,itemExpirationDate) : boolean
updateItem(itemName,itemQuantity) : boolean
OrderManagement
+orderInfo = string
+writeOrder(orderInfo) : boolean
-prepTime(orderInfo) : int
-UpdateQueue() : double profit[]
RestaurantTraffic
-status : int
-tableID : int
+ViewTraffic() : traffic[]
FinancialManagementSystem
+viewProfits() : double
+viewProfitProj() : double profit[]
+viewTransactions() : transactionStructure
MenuRating
+menuItemName : string
+menuItemPrice : double
+Rating : double
+ViewMenu() : menuItem[]
+deleteMenuItem(menuItemName) : boolean
+addMenuItem(menuItemName,menuItemPrice) : boolean
EmployeeManagementSystem
EmployeeName : string
EmployeePosition : string
pay : double
+ViewSchedule() : EmployeeSchedule[][]
+addEmployee(EmployeeName,EmployeePosition,pay) : boolean
+deleteEmplyee(EmployeeName) : boolean
+changeEmployeePay(EmployeeName,pay) : boolean
ManagementInterface
+Inventory() : void
+MenuAndRatings() : void
+RestaurantTraffic() : void
+FinancialManagementSystem() : void
+EmployeeManagementSystem() : void
CustmomerInterface
newOrder : string
+startOrder(newOrder) : boolean

Data Types and Operations Signatures

Customer:

1. **Reservation Interface:**

Attributes:

-customerName: string	A string value corresponding to the customer reserving the table.
-partySize: int	An integer value corresponding to the party size of the customer reserving.
-tableSize:int	An integer the shows the amount of people that can be seated on the table.
-tableNumber:int	An integer that represents the table reserved number.

Methods:

+reserveTable():bool	Method that allows the customer to reserve available tables according to the party size and availability.

2. **Customer Interface:**

Attributes:

-orderId: int	An integer given to identify the customers placed order.
-estimatedTime: time	A time displayed to the customers display to give an estimation on the orders' readiness

Methods:

+pickCategory(): bool	Method called to display the items of each category
+updateBill(*items): bool	Method that updates the customer's bill when customers add or remove items.
+updateOrder(): bool	Method that allows the customer to add or remove items from their bill.
+placeOrder(): bool	Method that allows the customer to place their order when satisfied.
+displayBill(): bill info*	Method that gives information about the items ordered, their individual price and the total bill.
+displayMenu(): bool	Method that displays the menus categories and options.
+displayItems(): bool	Method that displays the items of each category and their price.
+rateItems()	Method that allows the customer to rate the items that they ordered.

Employee:

1. **Employee Interface:** The employees' login, clock in, view their shifts, request for changes, clock out and logout using the employee interface

Attributes:

-username:string	An employee's username that's used to gain access to the employees portal.
-password:characters	A password that should match with the one that the employee created initially to log in.

Operations:

-BeginShift( ) : void	This method lets the employee clock in letting the manager know when they began their shifts so that the manager can keep track of their working hours and pay them accordingly
-ViewShifts( ) : tuple	The employee can view their shifts and keep track of their schedules. This method returns a tuple that consists of the date, day and time.
-ManageShifts( ) : Bool	The employee can add or delete shifts after getting the manager's approval..
-EndShift( ) : Bool	This method lets the employee clock out stating that his shift has ended.

2. **Ordering System:** This class places the customer's order in the order queue, pops the order from the queue after the chef has prepared it and notifies the waiter.

Attributes:

+items : list	A list of items that the customer orders from the menu
+quantity : integer	Total number of items that the customer ordered
+tableID : integer	The table ID of the customer that placed the order.

Operations:

+AddToQueue(items, quantity, tableID) : time	A method that adds the customer's order to the order queue and the returns the amount of time it's going to take to prepare.
+OrderReadyStatus(tableID) : string	This method returns a string that is sent over to the waiter's interface notifying him that the order is ready.
+PopFromQueue( ) : void	This method removes that order that has been prepared from the order queue.

3. **Restaurant Traffic:** This class manages the restaurant traffic and makes reservations.

Attributes:

+numberOfPeople : integer	The number of people that are present so that the reservations can be made accordingly
+time : time	Time when the customer places a reservation

Operations:

+reserve(numberOfPeople , time) : String	This method places a reservation for the customer and returns the time when the customer first reserved the table.
+NotifyWaiter(tableID) : Bool	This method notifies the waiter for any calls from the customer.
+NotifyBusser(tableID) : Bool	This method notifies the busser as soon as the customer processes the payment for him to clean the table.

4. **Database Handler:** This class is the main class that has dependency with the action of other classes. It verifies the login credentials, and gets shifts and open time slots.

Attributes:

-username : string	Employee's username to login into the system
-password : characters	Employee's password to login into the system

Operations:

+VerifyUser( ) : Bool	This method checks if the username is correct
+VerifyPassword( ) : Bool	This methods checks if the password is right
+getTimeShifts(username) : tuple	This method returns a tuple whose elements are the employee's date, day and time of when he's going to work.
+getTimeSlots( ) : tuple	This method returns a tuple with all open shifts containing the date, day and time.

Manager:

1. **Database Handler:**

Methods	Description
• getInventory(): inventoryItem[]	The Database Handler will retrieve all inventory data stored in the main restaurant database for use by the Inventory Handler
• addItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will allow for the manager to add an item to the restaurant inventory given a name, quantity of the item and the item's expiration date
• deleteItem(itemName, itemExpirationDate): boolean	This function will allow for the manager to delete a given item from the restaurant inventory given a name and the item's expiration date
• updateItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will allow for the manager to update an item's quantity given the item's name and expiration date
• getMenu(): menuItem[]	The Database Handler will retrieve all menu data stored in the main restaurant database for use by the Menu Management System
• deleteMenuItem(menuItemName): boolean	This function will allow for the manager to delete a given menu item from the restaurant menu given a name
• addMenuItem(menuItemName, price): boolean	This function will allow for the manager to add a given menu item to the restaurant menu given a name and price
• getTraffic(): traffic[]	The Database Handler will retrieve all floor traffic data stored in the main restaurant database for use by the Traffic Monitoring System
• getSchedule(): employeeSchedule[][]	The Database Handler will retrieve all employee scheduling data stored in the main restaurant database for use by the Employee Management System

• addEmployee(employeeName, employeePosition, pay): boolean	This function will allow for the manager to add an employee to the restaurant employee database given a name, position and pay rate
• deleteEmployee(employeeName): boolean	This function will allow for the manager to delete an employee from the restaurant employee database given a name
• changeEmployeePay(employeeName, pay): boolean	This function will allow for the manager to change the pay rate of an employee given the employee's name
• writeOrder(orderInfo): boolean	This function will send a customer's finished order to the order queue, where it will be interpreted by a chef to cook
• updateQueue(signal): boolean	This function will receive a signal from the chef interface to update the queue based on the completion of the chef's assigned order
• prepTime(orderInfo): int	Given order information, this function will return an estimated preparation time for the order
• getProfits(): double	The Database Handler will retrieve all profit data stored in the main restaurant database for use by the Financial Management System
• getProfitProj(timePeriod): double profit[]	The Database Handler will take all of the relevant financial information and input the information into the profit projection formula, which will in turn, return the set of profit projections for a given set of times
• getTransactions(): transactionStructure	The Database Handler will retrieve the restaurant's entire transaction history stored in the main restaurant database for use by the Financial Management System

2. **Inventory Management System:**

Attributes	Description
• itemName: string	The name of the inventory item will be stored as a string
• itemQuantity: double	The quantity of a given item will be stored as a double
• itemExpirationDate: simpleDateFormat	The expiration date will be stored as a date/time data type, which is available in Python libraries
Methods	**Description**
• viewInventory(): inventoryItem[]	This function will take the data given by the Database Handler and send the array of inventory items to the Manager Interface to be displayed to the Manager
• addItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will request the Database Handler to add an item to the restaurant inventory database given an item name, quantity, and expiration date
• deleteItem(itemName, itemExpirationDate): boolean	This function will request the Database Handler to delete an item from the restaurant inventory database given an item name and expiration date
• updateItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will request the Database Handler to update an item's quantity in the restaurant inventory database given the item's name, quantity, and expiration date

3. Traffic Monitoring System:

Attributes	Description
• status: int	This variable will hold an integer value serving as an indicator as to what state a table is in (dirty, clean, occupied, etc.)
• tableID: int	This variable will hold a given table's ID number
Methods	**Description**
• viewTraffic(): traffic[]	This function will request traffic information from the Database Handler to be used to display the tabling diagram to the requesting Manager

4. Menu Management System:

Attributes	Description
• menuItemName : string	Name of the item on the menu
• menuItemPrice : double	Price of the menu item
• Rating : double	Number of stars given to a dish
Methods	**Description**
• ViewMenu() : menuItem[]	Displays the whole menu which will include the menu item, price and rating associated with it
• deleteMenuItem(menuItemName, menuItemPrice) : boolean	Deletes the item on the menu that has the corresponding name and price and returns true if it worked and false otherwise
• addMenuItem(menuItemName, menuItemPrice) : boolean	Adds an item to the menu specifying the name and price associated with it. If it works then it returns true and if it does not work then it will return false

5. **Financial Management System:**

Attribute	Description
• timePeriod : simpleTimeFormat	Amount of time starting from projection starting date to ending date that will display days,weeks,months,years
Methods	**Description**
• viewProfits() : double	Displays the amount of money made
• viewProfitProj(timePeriod): double profit[]	Displays the how much money the owner will make from starting date to the ending date and returns an array that contains the amount of money the owner is projected to make
• viewTransactions() : transactionStructure	Displays transactions made such as paying employees, cost of food that is in the inventory now, amount of money gained/lost each day. This will return a structure of transactions made so the owner is aware where his money is going

6. **Management Interface:**

Methods	Description
• Inventory() : void	Displays the inventory options
• MenuAndRatings(): void	Display the options for menu and ratings
• RestaurantTraffic() : void	Display options for Restaurant traffic
• FinancialMangementSystem() : void	Display financial options
• EmployeeMangement() : void	Shows different options for employees

7. **Employee Management System:**

Attributes	Description
• EmployeeName : string	Name of the employee
• EmployeePosition : string	What the role of the employee is
• Pay : double	Amount of money paid to employees
Methods	**Description**
• ViewSchedule() : EmployeeSchedule[][]	Displays the schedule of employees who will be working on what days. It returns a 2D array with names and times of when people will be working
• addEmployee(EmployeeName, EmployeePosition,pay) : boolean	Adds an employee to the system specifying their name, position they will be working and their rate of pay. This method will return true if it worked and false otherwise
• deleteEmployee(EmployeeName, EmployeePosition,pay) : boolean	Adds an employee to the system specifying their name, position they will be working and their rate of pay. This method will return true if it worked and false otherwise
• changeEmployeePay(EmployeeName, pay) : boolean	Change the rate of pay for the specified employee. If it was altered correctly it will return true and false otherwise

8. **Order Mangement System:**

Attributes	Description
• orderInfo : string	Lists the items that were ordered
Methods	**Description**
• writeOrder(orderInfo) : boolean	Writes down everything that was ordered by reading in the order information returning true if it happened correctly and false if any problems came up
• prepTime(orderInfo) : int	Time it takes to prepare the food
• UpdateQueue() : boolean	Updates the queue of orders that are placed

9. **Chef Interface:**

Attributes	Description
• Signal : int	Signal that tells the system to update the queue
Methods	**Description**
• updateQueue(orderInfo) : int	Updates the queue of orders. The chef will remove the order from the queue once it has been done. Sends the signal to update the queue.

10. **Customer Interface:**

Attributes	Description
• newOrder : string	Describes what the new order is
Methods	**Description**
• startOrder(newOrder) : boolean	Writes down everything that was ordered by the customer and sends it to the queue.

Traceability Matrix

Customer:

	Software Classes			
	Order	TimeEstimation	OrderItem	Check
Domain Concepts				
Queue	x	x		
Tables	x		x	x
Menu	x		x	x
Inventory			x	
ChefInterface		x		
ReservationInterface		x		
CustomerInferface	x			x
DatabaseHandler		x	x	x

The Concepts were derived very simply from the idea of a restaurant in sync with its customers. Everything is connected one way or another and it can be told by simply looking at the names of the classes. For example: The matrix shows that for ordering purpose we use queue, tables and the menu. We need a time interface for queue, all the user interfaces as well as the database handler. For ordering an item, we use the data such as tables, menu, inventory, and the database. And lastly, when a customer leaves, we check for the table number, the items he ordered, his user interface for the ratings and reviews and the database data.

Employees:

Classes → Domain concepts ↓	Chefs Interface	Customer Interface	Table Management	Order Management	Employee Task Management
Order Status	X	X		X	
Order Display		X		X	
Order Queue	X	X			
Clean Table			X		X
Table Record	X		X	X	X

The Domain Concepts came from simply just taking a look at the various different functions of our system. We saw what we needed to accomplish and created various tasks. Order Status is done to give customers a description of how long their food wood take and in what stage it is in. Order Display displays the Order Status to the customer as well as the waiter. Order Queue is for the chef to see what order the orders are placed so he can make the food in the correct order it came in. This will affect Order Status which in turn will affect Order Display. Clean Table is there so we can properly manage tables. Managing tables entails cleaning, marking tables dirty, and also will be needed in customer reservations. Table Record is made for the waiters to mark a table empty and dirty.

Manager:

Classes → Domain concepts ↓	Database Handler	Inventory Handler	Financial Management System	Menu Ratings	Manager Interface	Restaurant Traffic	Employee Management System	Order Management	Customer Interface
Controller	X	X	X	X	X	X	X	X	X
Page Maker			X	X	X	X	X	X	X
Interface Page					X				X
Info Changer	X	X	X	X			X		
Notifier	X	X			X	X			
Database Connection	X	X	X	X	X	X	X	X	X
Investigation Request	X		X	X			X	X	
Traffic Display						X			
Predictor	X		X						

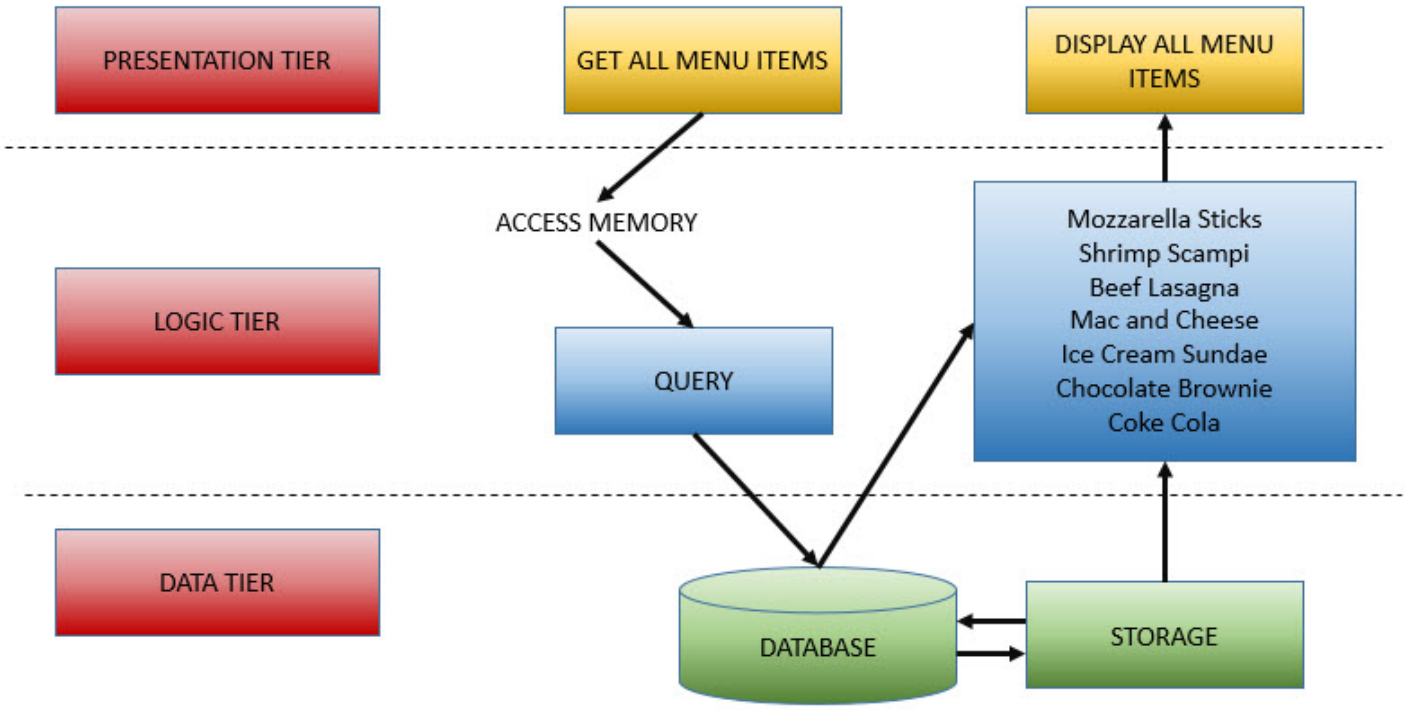
The Controller is essential to all parts of the program, as it coordinates everything and allows the different classes to communicate with each other. The Page Maker displays the UI for all of the relevant classes, so it is connected to everything that has a UI. The Interface Page allows users to interact with the program, so it is a part of any interface. The Info Changer is related to any part that allows saved data to be updated, this includes employee records, inventory items, menu items, etc. The Database Connection is another essential part of the system, as all aspects classes need access to the database. Notifier shows messages to the user, so it is connected to interface changes and the database. The Investigation Request has been expanded to anything that may request information from the database. Traffic Display, only relates to the restaurant traffic monitor, but it is accessed by both employees and the manager. The Predictor makes financial predictions based off of information stored on the database, so it is connected to both the financial management system and the database itself.

System Architecture and System Design

Architectural Styles

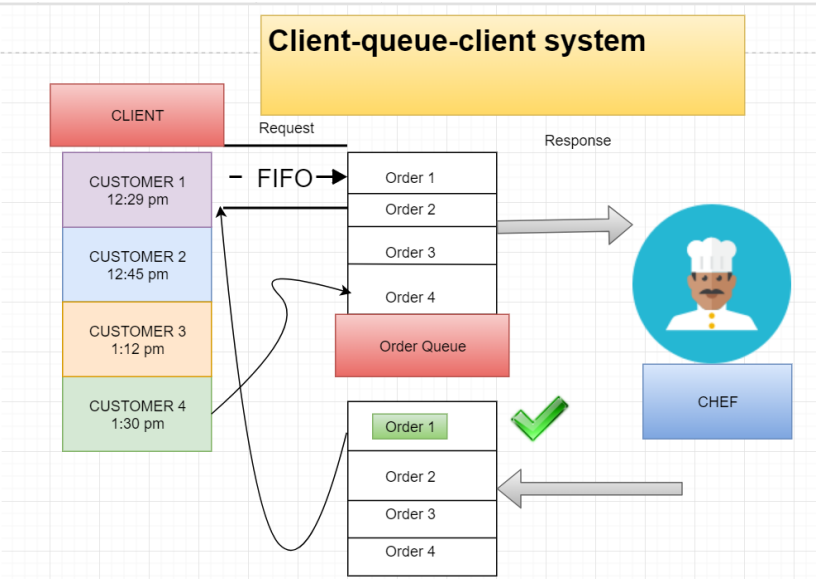
Customer:

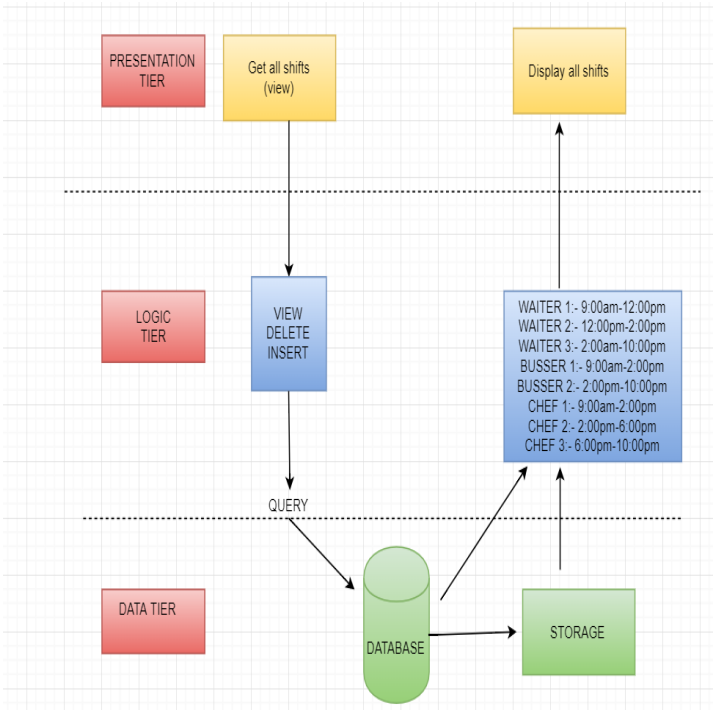
The customer's architecture styles are a combination of various forms of architectures. Starting from reservation, this is client server architecture when the client makes a choice what what table the want based off the ones shown as available by the server. Following that customers would be fronted with the menu interface where the layered architecture comes into play. After finishing the meal, the customers deal with payment for the meal and rating of their experience. Paying for the meal is component based architecture because it is dependent of the customer's choice for what form of payment the wish to use. Rating the experience and food is client server and message bus architecture. When used in harmony, the customer is able to enjoy their experience.

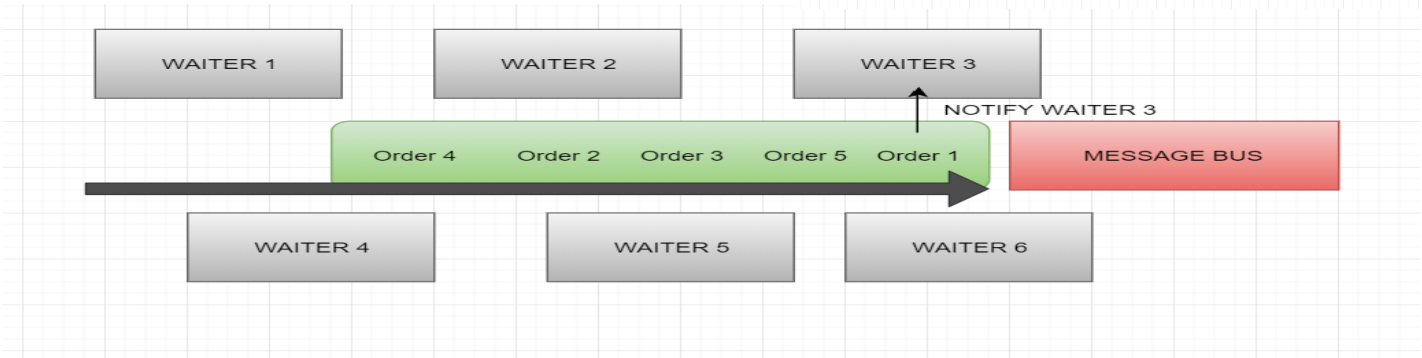


Employees:

The architectural style that we will be using is not limited to a single architectural style and is a combination of component based, service-oriented, multi-layered and client-server architecture. The ordering system uses the client-queue-client system architecture. Instead of communicating directly, the client and server exchange data with one another by storing it in a queue on the server. The waiters check the status of the order by using a message bus because a message bus specializes in transporting messages between applications (chef and the waiter). The login and logout as well as the tabling system uses a client server architecture with the clients being the waiters, customers, and bussers who update the current status of the table as independent clients and interact with a common server that records the statuses. The employees view their shifts and request modification using a multi-layered architecture/multi-tier with the presentation tier i.e. the employee interface requesting to view the shifts. The logic tier identifies employee's request to modify shifts and the data tier retrieves all the shifts.





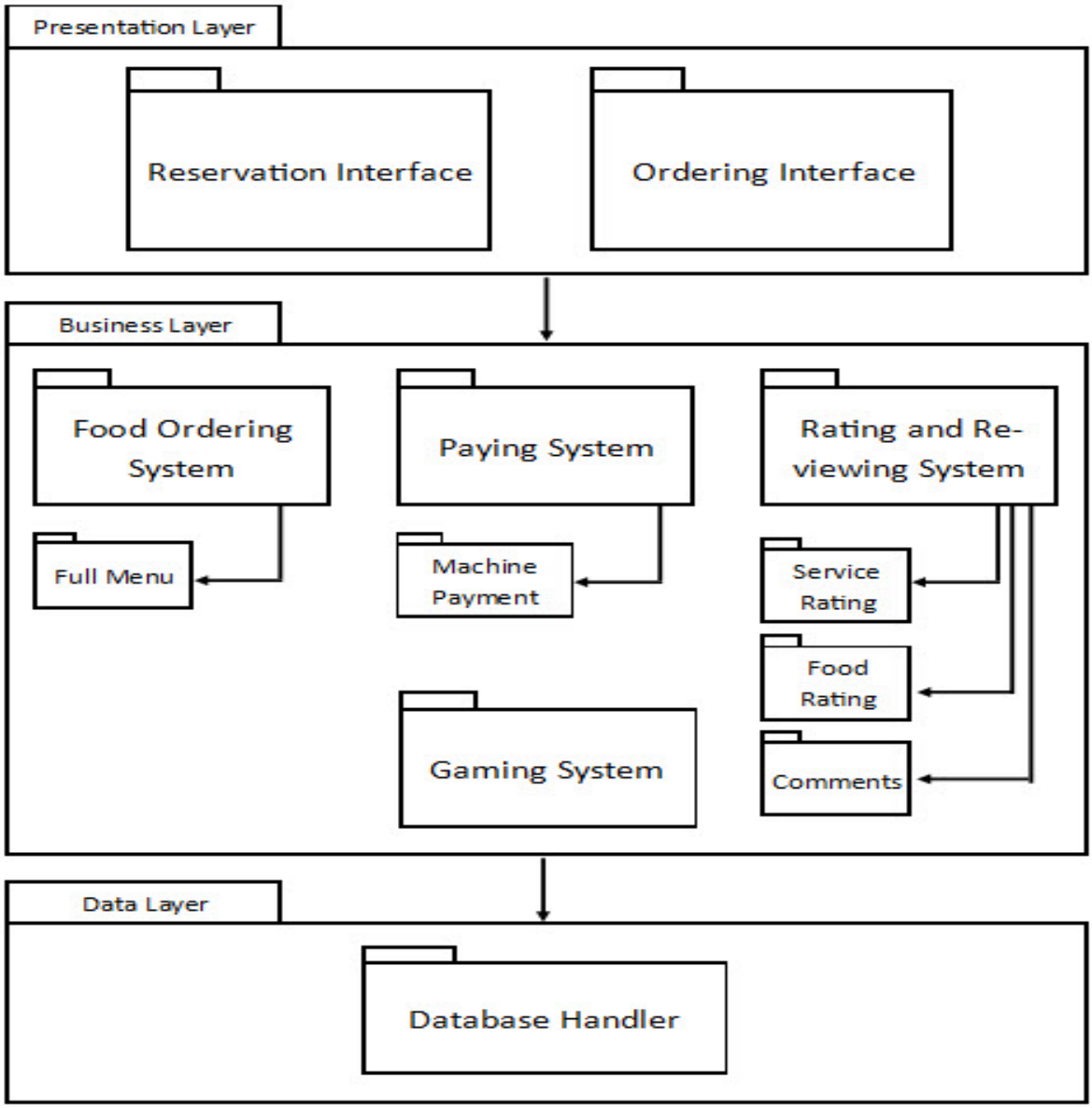


Managers:

The financial management system, the employee management system, and the inventory management system will need to use a multi-layered architecture for calculating and presenting the restaurant's financial information, employee scheduling and, the inventory system. The presentation tier will need to provide an easy to read interface for the managers. The logic tier will process requests from the manager, and later calculate and process the relevant information from the data tier. A client -server model will be necessary for storing the manager's login credentials as well as providing a view of the restaurant floor and every employee's status (i.e. viewing who is assigned to any given table). The menu rating and editing system will also run off of a multi-layered architecture. From the presentation tier, the manager sends a request to view ratings or edit the menu, the request is processed in the logic tier, which then retrieves the relevant information from the database. The information is processed, and sent back to the presentation tier for the manager.

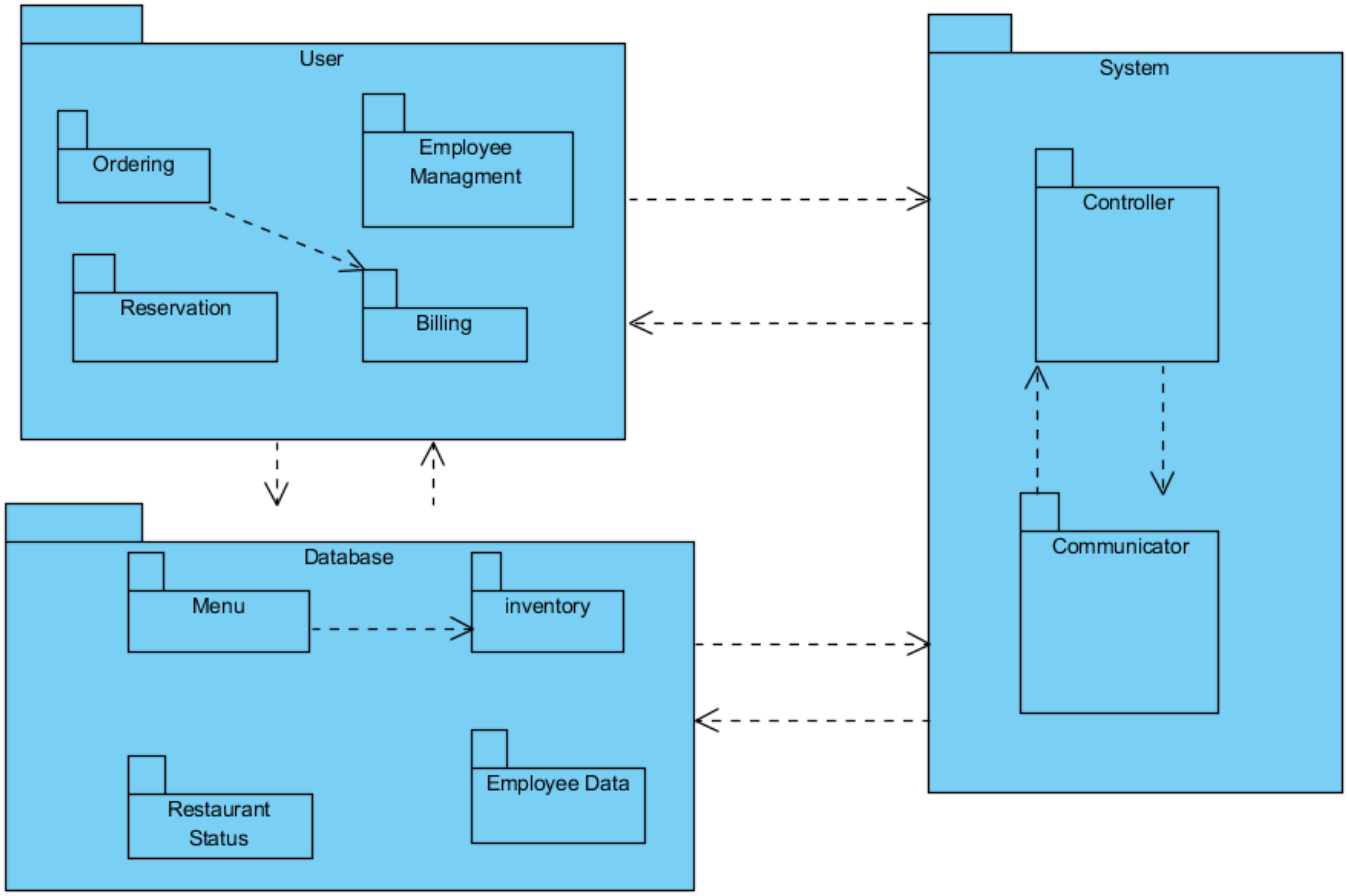
Identifying Subsystems

Customer:



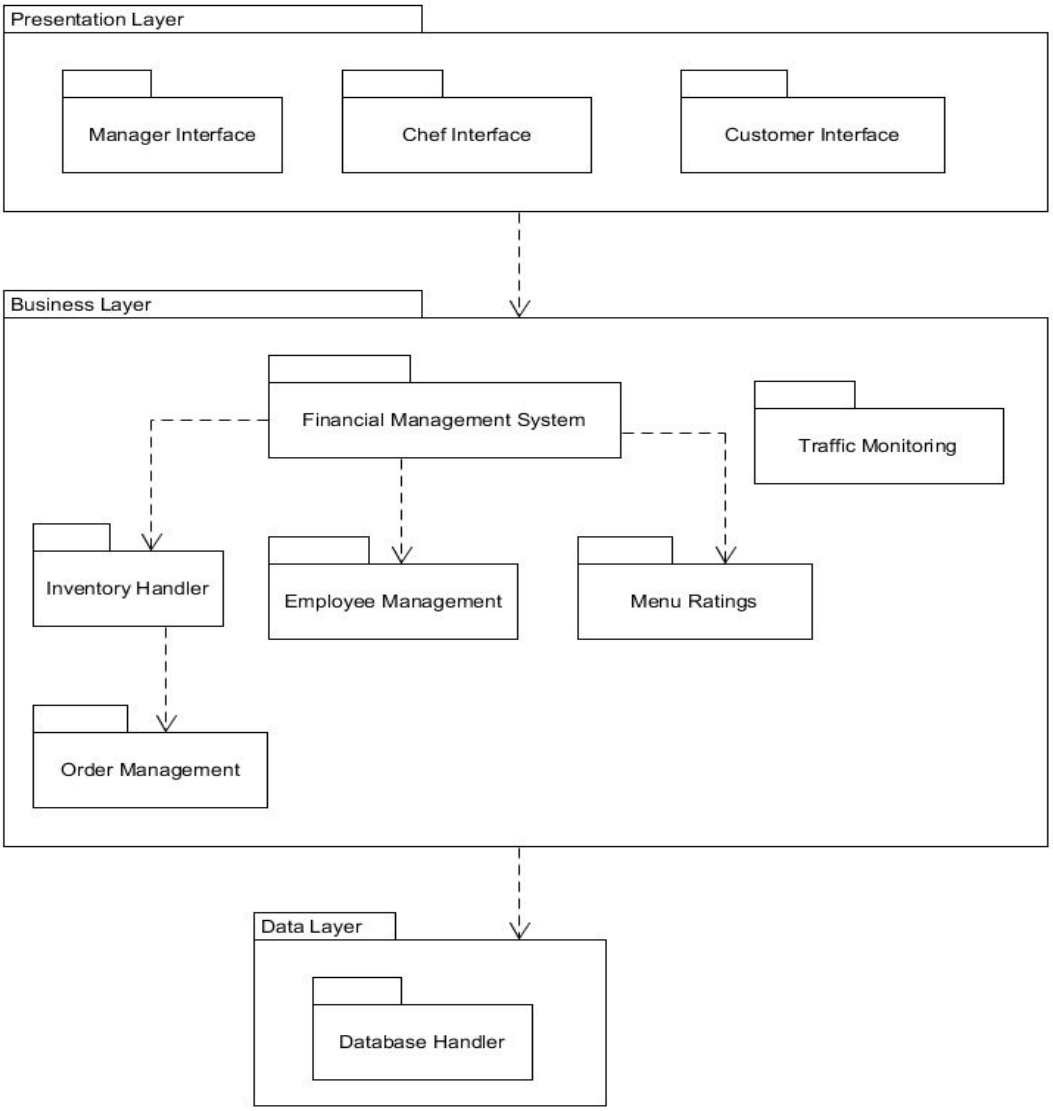
From arrival at the establishment to departure the customers is faced with two interfaces: Reservation Interface and Ordering Interface. In the Reservation Interface the customer is able to select where they wish to dine in the restaurant. The Ordering Interface brings in the various business layers. Food Ordering System allows the customer to select items off the menu that they wish to order. The Payment System deals with the electronic payment by the customer, this is unrelated to cash payments. The Rating and Reviewing System allows for the restaurant to be able to reanalyze their food and their employees based on their customers' experiences. All of the systems rely on the data found in the restaurant database, so they rely on the Database Handles to exchange data between various subsystems.

Employees:



We have a user package that includes all the packages that users need direct access to like ordering, reserving, billing and Employee management. We then have database where all our data is stored. This includes the menu, employee data, our inventory and the restaurant status: that is, which tables are available, how many workers are currently in the restaurant and all other data pertaining to the current status of the restaurant. Finally, there's the system package. This includes the controller and the communicator. All of our main packages have mutual dependencies and some of our sub-packages have dependencies as well.

Manager:



This UML package diagram describes the entire manager application within the larger restaurant automation software. The manager application is responsible for displaying relevant information to the manager, customer, and chef interfaces. The application utilizes six main subsystems to accomplish its functionality, which are the financial management system, the inventory management system, the employee management system, the menu management system, the order management system, and the traffic monitoring system. Each of these systems rely on data from the main restaurant database, and thus they rely on the database handler to exchange data between the subsystems and the database.

Mapping Subsystems to Hardware

Customer:

Due to the nature of client-server architecture, the hardware hierarchy of the system is straightforward. Understanding that the system is running on the client-server model, the server runs on the master computer which has the database and all applicable data. Customers have tablets through which they access the restaurant automation application which all contain the communicator, interface processing, and controller. From a customer's standpoint, this structure allows for each user to control what he/she wants to order and keeps the traffic/transfer of data simple.

Employees:

Although our architecture style is complex as it combines multiple styles, our hardware mapping is quite simple. There is a master computer that stores our database and runs our main program. Every table has a "table tablet" that can only access the customer interface of our system and runs all the related process. All waiters have "waiter tablets" that can only access the waiter interface of our system and runs all the related process. Similarly, every chef has a tablet that can only access the chef interface of our system and runs all the related process.

Manager:

Aside from having a desktop that has full access to all subsystems, managers will also have a portable tablet that will have some of the subsystems on it. For example, the financial management system will not be accessible on the tablet for privacy and security reasons, but managers will be able to pull up the restaurant floor traffic view. On the tablet, managers will have access to the restaurant floor traffic view, inventory view, and employee scheduling view. Both the desktop and the tablet will have access the database, which is on a server. On the desktop, managers will have access to the financial management system, which includes profits, employee wages, and profit projections. The desktop will also have access to the inventory manager, which will allow the manager to view and update the inventory. Managers will have the ability to edit menu options and view the ratings of menu dishes. Managers have the ability to create and edit employee scheduling and information from the desktop as well. The restaurant floor view will be the same on the desktop as it is on the tablet. It is notable that everything that is available on the tablet will also be available on the desktop.

Persistent Data Storage

Customer:

The system stores information which needs to be longstanding. In particular, we document details which make management's work easier and also provide clean documentation for future reference. This includes reviews and comments given by the customers, changes made to restaurant proceedings (i.e. the menu), ratings provided by the customers. In SQL terms, we have tables and fields which are to hold this data permanently with backups planned. Backups ensure long term reliability and establish that maintenance of the data is of paramount importance which is the backbone of persistent data storage acting highly appropriate for our purposes.

Employees:

For the employees to be able to view their shifts at any time of the day, we need to save the data that outlives a single execution of the system. Python allows us to permanently store content. A file can contain structured data or unstructured. Since we will be recording the shifts systematically, we will be using a structured database. The database would include columns as the days of the week and the rows as the entry of each employee. The SQL Database will be required for adding, deleting and updating the records which requires manager's authentication and approval. The following diagram given an idea of how the shifts are visible on the employee's interface.

Staff	Mon 18	Tue 19	Wed 20	Thu 21	Fri 22	Sat 23	Sun 24
Employee 1	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 2							
Position							
Employee 3		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Supervisor		Supervisor			
Employee 4	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 5	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	
Employee 6		3:00 AM - 7:00 AM					
Position		Trainee					
Employee 7	3:00 PM - 7:00 PM	7:00 AM - 3:00 PM		3:00 PM - 7:00 PM		7:00 AM - 3:00 PM	
Position	Manager	Manager		Manager		Manager	
Employee 8	3:00 AM - 7:00 AM				7:00 AM - 3:00 PM		
Position	Sales Associate				Sales Associate		
Employee 9		3:00 AM - 7:00 AM	3:00 PM - 7:00 PM	3:00 PM - 7:00 PM		11:00 AM - 3:00 PM	
Position		Cashier	Cashier	Cashier		Cashier	
Employee 10							
Position							
Employee 11	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 12							
Position							
Employee 13		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Sales Associate		Sales Associate			
Employee 14	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 15	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	

Managers:

Our system will need to outlive a single execution of the system given this software will be the core to the operation of a restaurant. The management side of the software will need to store login information for all employees and managers working at the restaurant, inventory information, menu information, and financial information for the restaurant, which will all be stored in separate sections of a large relational database. Login information will be stored in a private table, which will associate a given login username and password with a valid employee or manager object, which itself will contain all personal information pertaining to said employee or manager. Inventory information will be stored in a public table with each item being described by name, quantity, cost per unit, and low quantity threshold. Menu information will also be held in a public table described by name, price, and average customer rating. Financial information will be held in a private table containing the total transaction history of the restaurant, which is described by money lost/gained and date of transaction; daily earnings/losses are entered into this table as singular transactions rather than each customer order being entered as its own transaction.

Network Protocol

Our system will be running on multiple terminals/machines (for the chef, waiters, customers and bussers) and therefore we will require a communication protocol. Python provides two levels of access to network services, one of which is at a low level where you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. We will be using the HTTP protocol with port number 80 because we're using web pages to develop our software which is HTTP's main functionality. The python modules used are httplib, urllib, xmlrpclib. Writing our communication protocol in Python will allow for easy portability with the several other modules throughout the restaurant automation software that are also written in Python.

Global Control Flow

Execution Orderness:

"Little Bits" is a procedure driven software. Practically everything executes in a linear fashion. In the restaurant business you have many different sub teams. For the sake of simplicity, we combined this into three teams. An employee team, a Customer team, and a Manager Team. The customer would enter the restaurant and push in the details (ex. Number of seats and time) allowing the program to seat him in a optimal position. Then the customer would make his order whether through his/her tablet or waiter's tablet. The order will be received by the Chef and when it is ready will notify the waiter to take the order directly to the customer. When the customer finishes eating he will pay the bill whether through the tablet or waiter and thus marking the table dirty for the busboy to come clean it. This clearly shows that all actions are made in a linear fashion all depending on the previous action.

For the employee teams it is linear because all the various employees need to access all relevant systems with ease. All actions are based off of each other. When a customer places a reservation, the table management has to change and a waiter is notified a table they are assigned. Then once the order is placed it is sent to the order queue for the chef to make the orders. Then the order time is calculated and displayed. Finally, once a customer leaves, the table is marked dirty using a timer and a busser is notified. This is a simple explanation showing the linearity of the system.

For the Managers team, this system must be procedure-driven. It must execute in a linear fashion in order to maintain consistency. In a fast paced restaurant setting a manager will need to access everything with ease, so a system that changes with different events is not optimal.

Time Dependency:

This system is periodic as for everything done from being seated to paying the bill is done in a periodic manner. For the customer's team, this is all time dependent because the time the customer makes the order places it in a queue for the chef to cook. Also calculation of wait time for the customer to receive the order is also on real time basis. For the employee's section, there is a simple timer. The timer is started in real time when a customer pays their bill. Every five minutes a notification will be sent to the waiter, which the waiter confirms or ignores if the customer has left the table. Until the notification is accepted that the customer leaves, the timer will then repeat until it is completed. Finally, for the manager's team, there are no timers in our system due to the fact our system is an event-response type. Their system does not depend on real time. The manager system does not need to be constantly updated with respect to time. All functions on the manager aspect of the applications depend on events initiated by relevant users.

Concurrency:

Little Bits will contain multiple threads, which will involve multiple systems running independently at once. Multiple customers will be placing orders at the same time which is why we need concurrency. The solution is to run multiple threads into the queue. Synchronization for the customer's team is not needed because each thread is independent of the other. For the employee's team, they will also contain multiple threads, which will involve multiple systems running independently at once. Multiple tables will be placing orders at one time and this will need to be sent to the Chefs tablet which is why we need concurrency. The solution is to run multiple threads into the OrderQue. Synchronization for this team is needed because some threads such as the ones for the OrderStatus and OrderQueue will need to work together. For the manager's team it will use threads to manage login sessions between multiple managers that are accessing the system simultaneously. There will be synchronization so that one manager doesn't overwrite the other one. For example, if one manager is adding an item to the inventory database and another is removing or adding another item there will be locks imposed so that the data will not be overwritten.

Hardware Requirements

1. Ziosk Z-400 Tablet
    a. Display: 1024x600 WSVGA LCD
    b. Battery: 7.4 VDC Li-Ion polymer 8720 mAh custom rechargeable pack
    c. Wireless: 802.11 a/b/g/n/ac plus Bluetooth 4.0 (LE)
    d. Card Reader: 3DES DUKPT Encrypting Magnetic Stripe Reader
    e. Description: The Ziosk Z-400 Tablet will need to have an eight inch diagonal, to hold a minimum of 1 GB of hard drive space, and to handle a network bandwidth of around 60 Kbps to handle the transfer order information to and from each tablet.
2. Dell Inspiron Desktop
    a. Processor: 7th Generation Intel® Core™ i7-7700 processor (8MB Cache, up to 4.20 GHz)
    b. Operating System: Windows 10 Pro 64-bit English
    c. RAM: 16GB, 2400MHz, DDR4
    d. Hard Drive: Dual Storage (3.5' 1TB HDD + 2.5' 128GB SSD)
    e. Wireless: Dell Wireless 1707 Card 802.11bgn + Bluetooth 4.0
    f. Video Card: NVIDIA® GeForce™ GTX 1050 with 2GB GDDR5 graphics memory

Algorithms and Data Structures

Algorithms

Table Reserving Algorithm:
As customers enter the restaurant they are asked to specify their party size among other things into a tablet. This algorithm uses the party size and compares it against the available tables to see which table should be designated to the customer. The algorithm utilizes a sorted list (sorted and prioritized by a first come first serve basis) to take input from and then compares the party sizes of all customers and find all the vacant and suitable tables. The customer gets to choose one among them according to its position in the restaurant (People preferring a view). If the party size is less than the table size and no other tables are available, then they are given the table. In case the party size is larger than the capacity of the vacant tables or if the tables which can accommodate the party size are busy, then the customers have to wait until they are emptied.

```
while (sorted list of customers does not equal 0)
{
        while (iterating through the list of available tables)
        {
                while (iterating through sorted list of customers)
                {
                        if (size of table is equal to a party size of the customers)
                        {
                                //Highlight allowed tables the fit the party for the customer to
                        reserve
                                //Allow the customer to reserve and display reservation number

                        }
                        else if(size of table is greater than party size and no other party
                        matches      table size)
                        {
                                //Highlight allowed tables the fit the party for the customer to
                        reserve
                                //Allow the customer to reserve and display reservation number
                        }

                }
        }
}
```

Bill Payment Algorithm:

As customers place their order the bill amount gets added up according to the price and the quantity of the item ordered. When the customer wants to pay the bill, the total bill amount is displayed with three modes of payment. Based on the customer's preferred payment method, the algorithm matches the mode of payment selected by the customer with the payments options in the database and displays the corresponding payment page. On each page, the customer has the option to tip and rate the food and share his/her customer experience in place of the conventional method of payment of the bill.

```
while (bill of customer not equal to zero)
{
        if (payment method equals credit card)
        {
                // Display Tipping option according to the bill amount
                //calculate bill amount accordingly and display payment page on tablet
                // Acknowledge payment
                //Display rating option
        }
        if (payment method equals paypal)
        {
                // Display Tipping option according to the bill amount
                //calculate bill amount and display paypal page
                // Acknowledge payment
                //Display rating option
        }
         if (payment method equals cash)
        {
                //send notification to the table waiter to assist the customers
                //Display rating option
        }
}
```

Order Status Algorithm:

```
class OrderQueue:
        def  init  (self):
                self.orderItems = [ ]
        def isEmpty(self):
                return self.orderItems = [ ]
        def enqueue(self, orderItems)
                self.orderItems.insert(0,orderItems)
        def dequeue(self):
                return self.orderItems.pop()
        def size(self):
                return len(self.orderItems)

        q = OrderQueue()
def orderStatus:
        while(q.size!=0)
                if (q.isEmpty == True):
                        print (“No orders in progress”)
                else:
                        if (q.size ==1)
                                print (“Cooking...”)
                                setTimer()
                                //as soon as the timer goes off, print (“Order is ready”)
                                q.dequeue()
                                //notify waiter
                                print (“No more remaining orders”)
                        else:
                                print (“order number 1 out of q.size() is cooking”)
                                print (“(q.size-1) orders are currently in progress”)
                                setTimer()
                                //as soon as the timer goes off, print (“order is ready”)
                                //notify waiter
                                q.dequeue()
                                q.enqueue(order #x)
```

Polynomial Regression

For the financial management system, a mathematical model will be used in making profit predictions of the restaurant will experience given the customer traffic in a given amount of time, and the popularity of the restaurant's menu items. With the two variables being quantity of items sold and total income collected, we will estimate total profits collected over a time period specified by the user which is the manager. Polynomial regression is a technique which will be used which takes the form:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \square$$

Where X represents the total quantity of items sold, Y represents the total income collected by said menu items, $\in$ represents the error and *h* represents the degree of the polynomial. The manager will be able to choose the degree of the polynomial to use in making a projection on the profits of the restaurant. Using established data points taken over a period of time, six months for example, the system will begin to make calculations on profit projections for months and/or years in advance, given input for the desired time period from the manager.

Data Structures

For the customer section, they will use a variety of data structures. For viewing and identifying the occupied tables for the reservation interface, a queue is storing information needed (such as availability) transfers this information to the customer reserving. Database is also used for storing a variety of information. The database stores table information (table size, number), it also stores the menu items and their quantities. For the employees' section, all orders placed by a customer are added to a queue which is accessible by the chef. The queue will be implemented using the data structure called priority queue where the chef can assign priorities manually and use the "order status algorithm" to notify the customers. Finally, for the manager segment, they use data structures in order to increase performance. They will be using arrays for now to make sure the system works for simplicity purposes. Then later on, once everything is working, they will possibly switch to linked lists which are similar to arrays but are more dynamic and help save more space and be more efficient, or switch to trees to increase the speed of our system. This will be up for discussion later after the system is fully functioning and operational for the design.

User Interface Design and Implementation

What Has Changed

Customer:

Some features come standard with every dining experience, such as: coming into the restaurant, ordering food, and paying for food. Each step is very important and caters to the customer's various needs such as getting a table, getting food, and paying to they may go home. From the original proposal we have added a few new features to this experience. As expected, the interface will be an altered version of the original draft but still completing all the same functions. Additionally, during the customers' food ordering experience another feature we have added is the ability to play a game while you wait for the food to be prepared. This is important to the experience for the customer because they then do not notice the amount of time it takes to create their food and they are just about to relax.

Employees:

In the employees' team, we decided to keep things very simple. We have mainly stayed with our original ideas. One of the ideas that we added was a timer system for the bussers. This system is a timer that when a customer pays a bill, a timer is started. Until the waiter confirms that the customers have left, the timer will continue to run. We have also solidified our testing layout for a restaurant. We determined that we would have 10 tables with 5 waiters, 2 Chefs, and 2 Bussers. This simplified how we would code and create our system.

Managers:

For the manager team, nothing has changed.

Ease of Use

Customers:

Reservation
- CUSTOMER ARRIVES
- Type in name and party size
- Select table location preference
- Given wait time

At the Table
- Either open menu or play games
- In menu, select category
- Then select item and add specifications if applicable
- When finished selecting items, press the ORDER button
- Screen displays wait time for all food in total
- In games, select game and stop whenever you want
- Also SOS button available

When Ready to Pay
- Selecting Pay option
- Select form of payment or splitting the check
- Pay or call waitress for cash
- CUSTOMER LEAVES

Employees:

To create a system that is easy to use, we tried to eliminate having a multitude of functions. We created a simple interface that simply, when an employee logs in, they will be automatically directed to their positions portal. The user interface is very user friendly. Each tab takes allows the user to access one or more of our system's functions. Employees can easily enter their availability through clicking on a calendar to mark the days they are available and can then enter the specific times they are available (during which they desire to have their shifts) through choosing the start and end time from a drop down menu. We wanted to make as many things automated as possible, to help manage tasks so that the employee can focus on customer satisfaction.

Managers:

For the managers in particular when the user interface is implemented, management interface will be simple and user friendly, where it will only take a few clicks to see your restaurant menu, finances, inventory, employees and restaurant traffic. The managers will need to work with a system that is quick to learn and easy to memorize. A quick and simple interface will allow managers to spend less time staring at a screen and more time engaging with both the employees and the customers.

Design of Tests

Test-Case Identifier: TC-1
Function Tested: reservingTable(): void throws exception
Pass/Fail Criteria: The test passes if a customer is able to reserve a table successfully by entering all the required input details

Test Procedure	Expected Results
Call function (Pass)	Table will be reserved
Call function (Fail)	If the party size is more or less than the available table sizes or if the restaurant is full or if the customer enters wrong details

Test-Case Identifier: TC-2
Function Tested: billPayment(): void throws exception
Pass/Fail Criteria: The test passes if a customer is able to pay his/her bill successfully by choosing one of the payment options and entering the required details

Test Procedure	Expected Results
Call function (Pass)	Bill will be paid
Call function (Fail)	If the customer enters wrong details or is unable to choose one of the available payment options

Test-Case Identifier: TC-3
Function Tested: Notify waiter (waiterID, messageID)
Pass/Fail Criteria: The chef can indicate when the order is finished and the waiter is notified

Test Procedure	Expected Results
Call function (Pass)	1- the waiter gets a notification to pick up the order and deliver it to the customer. 2- the order disappears from the order cue.
Call function (Fail)	1- The waiter doesn't get a notification to pick up the order and deliver it to the customer. 2- the order doesn’t disappear from the order cue. 3- the wrong waiter gets notified.

Test-Case Identifier: TC-4
Function Tested: Notify waiter (waiterID, messageID)
Pass/Fail Criteria: A waiter is notified five minutes after a customer pays their bill

Test Procedure	Expected Results
Call function (Pass)	1- A popup message shows to the specific waiter who we want to be notified. The message says "is table X vacant?" where X is the table number of the customer. 2- If the waiter chooses "No," result 1 is repeated after 5 minutes till the waiter chooses yes. 3- if the waiter chooses "Yes," a popup message shows to the busser saying "Table X is vacant. Please clean it."
Call function (fail)	The waiter is not notified or the wrong waiter is notified.

Test-Case Identifier: TC-5
Function Tested: notifyBusser (TableID)
Pass/Fail Criteria: The waiter can notify the busser to clean a table

Test Procedure	Expected Results
Call function (Pass)	a popup message shows to the busser saying "Table X is vacant. Please clean it."
Call function (fail)	The pop up message does not show.

Test-Case Identifier: TC-6
Function Tested: UpdateTabling (TableID)
Pass/Fail Criteria: A busser can mark a table as clean

Test Procedure	Expected Results
Call function (Pass)	The restaurant layout shows the table as clean
Call function (fail)	The restaurant layout shows the table as dirty, inUse or not available.

Test-Case Identifier: TC-7
Function Tested: DisplayTimeSlots();
Pass/Fail Criteria: Employee can see the time slots they can sign up for.

Test Procedure	Expected Results
Call function (Pass)	Available time slots show
Call function (fail)	Available time slots don't show or unavailable time slots show as available

Test-Case Identifier: TC-8
Function tested Tested: GetPermissions()
Pass/Fail Criteria: Employee can ask for permission to add/delete a time slot to their schedules.

Test Procedure	Expected Results
Call function (Pass)	In the tab "schedules" in the manager portal, a message shows saying "X is asking to add /drop the the following time slot." X is the employee.
Call function (fail)	The message doesn't show up

Test-Case Identifier: TC-9
Function Tested: payEmployee()
Pass/Fail Criteria: Manager can pay employees and can adjust the pay

Test Procedure	Expected Results
Call function (Pass)	1- manager can see the number of hours every employee worked and the recommended pay 2- manager can adjust pay 3- a money transfer is sent to the employee with the amount the manager chose.

Test-Case Identifier: TC-10
Function Tested: ViewInventory(): inventoryItem[][]
Pass/Fail Criteria: The test passes if the manager is able to view the inventory successfully

Test Procedure	Expected Results
Call function (Pass)	Display Inventory
Call function (Fail)	Database failure

Test-Case Identifier: TC-10a
Function Tested: addItem(itemName, itemQuantity, itemExpirationDate): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to add an item to the inventory successfully by entering all the required item details

Test Procedure	Expected Results
Call function (Pass)	Item added to inventory
Call function (Fail)	Item amount is less than to equal to 0, no name

Test-Case Identifier: TC-10b
Function Tested: deleteItem(itemName, itemExpirationDate): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to delete an item from the inventory successfully by telling the system which item he wants removed

Test Procedure	Expected Results
Call function (Pass)	Item deleted from inventory
Call function (Fail)	Item does not exist, item name blank

Test-Case Identifier: TC-10c
Function Tested: updateItem(itemName, itemQuantity, itemExpirationDate): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to update an item in the inventory successfully by entering in the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item in the inventory is updated
Call function (Fail)	Item does not exist, quantity is less than 0

Test-Case Identifier: TC-11
Function Tested: ViewMenu(): menuItem[]
Pass/Fail Criteria: The test passes if the manager is able to view the menu

Test Procedure	Expected Results
Call function (Pass)	Display Menu
Call function (Fail)	Database Failure

Test-Case Identifier: TC-11a
Function Tested: addMenuItem(menuItemName, menuItemPrice): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to add item to the menu, by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item added to menu
Call function (Fail)	itemPrice was $0.00 or less or no price or no name

Test-Case Identifier: TC-11b
Function Tested: deleteMenuItem(menuItemName, menuItemPrice): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to delete item to the menu, by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item removed to menu
Call function (Fail)	Incorrect item name and price, price is less than 0, item does not exist

Test-Case Identifier: TC-11c
Function Tested: UpdateMenuItem(menuItemName, menuItemPrice): boolean
Pass/Fail Criteria: The test passes if the manager is able to update an menu item by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item is updated
Call function (Fail)	Incorrect item name and price, price is less than 0, item does not exist

Test-Case Identifier: TC-12
Function Tested: ViewSchedule(): EmployeeSchedule[][]
Pass/Fail Criteria: The test passes if the manager is able to see the employee schedule

Test Procedure	Expected Results
Call function (Pass)	Displays Employee Schedule
Call function (Fail)	Database failure

Test-Case Identifier: TC-12a
Function Tested: addEmployee(EmployeeName, EmployeePosition,pay): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to add an employee to the system

Test Procedure	**Expected Results**
Call function (Pass)	Employee added to system
Call function (Fail)	Incorrect position or pay<0 is inputted

Test-Case Identifier: TC-12b
Function Tested: deleteEmployee(EmployeeName, EmployeePosition,pay): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to remove an employee from the system

Test Procedure	**Expected Results**
Call function (Pass)	Employee is removed from the system
Call function (Fail)	Incorrect name or position or pay<0 is inputted

Test-Case Identifier: TC-12c
Function Tested: ChangeEmployeePay(EmployeeName, EmployeePosition,pay): boolean throws exception
Pass/Fail Criteria: The test passes if the manager is able to change an employee's wages

Test Procedure	**Expected Results**
Call function (Pass)	Employee's pay is updated
Call function (Fail)	Incorrect name or position or pay<0 is inputted

Test-Case Identifier: TC-13
Function Tested: updateQueue(orderInfo): int throws exception
Pass/Fail Criteria: The test passes if the queue is updated correctly by the chef

Test Procedure	Expected Results
Call function (Pass)	Displays updated queue
Call function (Fail)	Queue is not updated, Queue is updated incorrectly

Test-Case Identifier: TC-14
Function Tested: inventory (): void throws exception
Pass/Fail Criteria: The test passes if the system launches the inventory interface

Test Procedure	Expected Results
Call function (Pass)	Displays inventory menu
Call function (Fail)	System crashes

Test-Case Identifier: TC-14a
Function Tested: MenuAndRating(): void throws exception
Pass/Fail Criteria: The test passes if the system launches the Menu and Ratings interface

Test Procedure	Expected Results
Call function (Pass)	Displays Menu and Ratings options
Call function (Fail)	System crashes

Test-Case Identifier: TC-14b
Function Tested: RestaurantTraffic(): void throws exception
Pass/Fail Criteria: The test passes if the system launches the Restaurant Traffic interface

Test Procedure	Expected Results
Call function (Pass)	Displays restaurant traffic view
Call function (Fail)	System crashes

Test-Case Identifier: TC-14c
Function Tested: FinancialManagementSystem(): void throws exception
Pass/Fail Criteria: The test passes if the system displays the Financial Management System interface

Test Procedure	Expected Results
Call function (Pass)	Displays FMS menu
Call function (Fail)	System crashes

Test-Case Identifier: TC-14d
Function Tested: EmployeeManagementSystem(): void throws exception
Pass/Fail Criteria: The test passes if the system correctly displays the employee management system interface

Test Procedure	Expected Results
Call function (Pass)	Displays Employee management options
Call function (Fail)	System crashes

Test-Case Identifier: TC-15
Function Tested: startOrder(newOrder):boolean
Pass/Fail Criteria: The test passes if the customer is able to submit a new order

Test Procedure	Expected Results
Call function (Pass)	Order is sent to the queue
Call function (Fail)	Database failure, invalid order (i.e. order item is out of stock)

Test-Case Identifier: TC-16
Function Tested: viewTraffic(): traffic[]
Pass/Fail Criteria: The test passes if the system displays correct information about the current restaurant traffic and table statuses

Test Procedure	**Expected Results**
Call function (Pass)	Displays restaurant information
Call function (Fail)	Database failure, displays incorrect or outdated floor view/information

Test-Case Identifier: TC-17
Function Tested: writeOrder(orderInfo): boolean throws exception
Pass/Fail Criteria: The test passes if there are no problems with the customer's order

Test Procedure	**Expected Results**
Call function (Pass)	Order is sent to the queue and returns a confirmation to the customer
Call function (Fail)	Order is not sent to the queue and an error message is returned to the customer

Test-Case Identifier: TC-17a
Function Tested: prepTime(orderInfo): int throws exception
Pass/Fail Criteria: The test passes if the system displays the correct preparation time for the order

Test Procedure	Expected Results
Call function (Pass)	Displays the correct approximate time for order completion
Call function (Fail)	Database failure, displays no or incorrect preparation time for order

Test-Case Identifier: TC-17b
Function Tested: updateQueue(): boolean throws exception
Pass/Fail Criteria: The test passes if the queue is correctly updated with the customer's order

Test Procedure	Expected Results
Call function (Pass)	Displays correct order queue
Call function (Fail)	Database failure, queue is updated incorrectly or not updated at all,

Test-Case Identifier: TC-18
Function Tested: viewProfits(): double throws exception
Pass/Fail Criteria: The test passes if the manager is successfully able to obtain profits

Test Procedure	Expected Results
Call function (Pass)	Displays profits of type double
Call function (Fail)	Database failure, profits are not updated

Test-Case Identifier: TC-18a
Function Tested: viewProfitProj(timePeriod): double profit[]
Pass/Fail Criteria: The test passes if the manager is successfully able to see his profit projections

Test Procedure	Expected Results
Call function (Pass)	Displays profit projections
Call function (Fail)	Database failure, profit projections are not calculated, not enough

Test-Case Identifier: TC-18b
Function Tested: viewTransactions(): transactionStructure
Pass/Fail Criteria: The test passes if the manager is successfully able to see his profit projections

Test Procedure	Expected Results
Call function (Pass)	Displays transactions made such as paying employees, cost of food that is in the inventory now, amount of money gained/lost each day
Call function (Fail)	Database failure, transactions not up to date

Test Coverage

The tests cover most essential classes such as Reservation, Ordering and Payment processes implemented by the customers. More tests will be designed when we see it needed in the future as we develop the customer interface in the Web application. Some of the tests cover the basic possibilities within UC2 and UC5 including sign. For the manager section the test covers the important classes which deal with inventory management, employee management, restaurant traffic, financial management systems. More tests may be required when the system is implemented and needs to be tested for durability.

Integration Testing Strategy

Big Bang integration testing is an integration testing strategy where all units are linked at once resulting in a complete system. We will not use this as our primary testing strategy but instead use big bang integration testing to test our fully developed software. This is useful to test if the back end database and the front end interaction works smoothly. Big bang testing also helps to look for loopholes in the software when all components are integrated together. For example: Combining all the applications from the customer side (placing orders, SOS call button, processing the payment and reserving the tables), the employee's side (marking the status of the table, order queue, and accessing the schedules) and the manager's side, all these functionalities must be embedded together and work in the right order.

We will be using mixed(sandwich) testing for testing our primary components. Sandwich testing combines both top down and bottom up approach. We break the application into three parts- customers, employees and manager, where using permutations each and every application pair is tested separately and the corresponding results are recorded. The pairs include the customer chef pair where tests include the ordering queue and the processing time, the customer busser pair where after the payment is processed, the busser is notified, the waiter busser pair where both the parties are responsible for updating the table status, the manager employee pair where the schedules/shifts are managed, the manager chef pair where they coordinate with the ingredients. This strategy is useful and works quite well because before we combine everything and move on to big bang, we need to make sure that two components interact with each other in an orderly manner.

A few functionalities like the financial management system are independent and their testing is done using the big bang testing strategy.

Project Management

Merging the Contributions from Team Members

Overall merging has been completed by one person for continuity. However, the report for each part has been checked over by each member of the group. As a team we have also had each sub-team; the Customer team, the Employees team, and the Managers team, complete their own team report which was then compiled into one large report. It has been difficult to make sure that every part is the same. We try to make sure spacing is the same, fonts are all the same, and the formatting is done the same way every time. With the amount of information that needs to be compiled it gets increasingly difficult. To manage this we made each sub-team do their own mini report to make merging simplified. We also created a basic template to use for each report, and for this final report we just have to insert the old files into one large file.

Project Coordination and Progress Report

Customer

The customer team has completed creating the first draft of user interface expectations and step by step page layout. We have also finished creating the reservation pages and work to make them functional from the customer's end. We have to start putting together the menu to have the user interface we originally sketched out. We also need to create a functional reservation system for the customers' convenience. Secondly we need to create databases to address; Reservations and selection of specific tables, ordering through the digital menu and adding specifications, Showing the bill and averaging a tip, Reviewing and leaving comments regarding the food and experience, and an SOS button to call the waiter

Employees

The employees section has been dividing the work based on skills as well as points was difficult at first but as the course progressed it became a lot easier since everyone was up to date with the course material and flexible with the tasks being allotted.

Since we are coding in different languages we require a framework so we settled on using Django- a python framework. Currently we're learning more about Django on how to use it. We're also done with the tedious Wamp server installation. We plan to divide our work based on teams and the employees team is currently focusing on coding the order status algorithm and working on the employee login system. We developed an outline of how to code UC2(traffic monitoring) and UC 5(managing employees) and will implement the two soon.

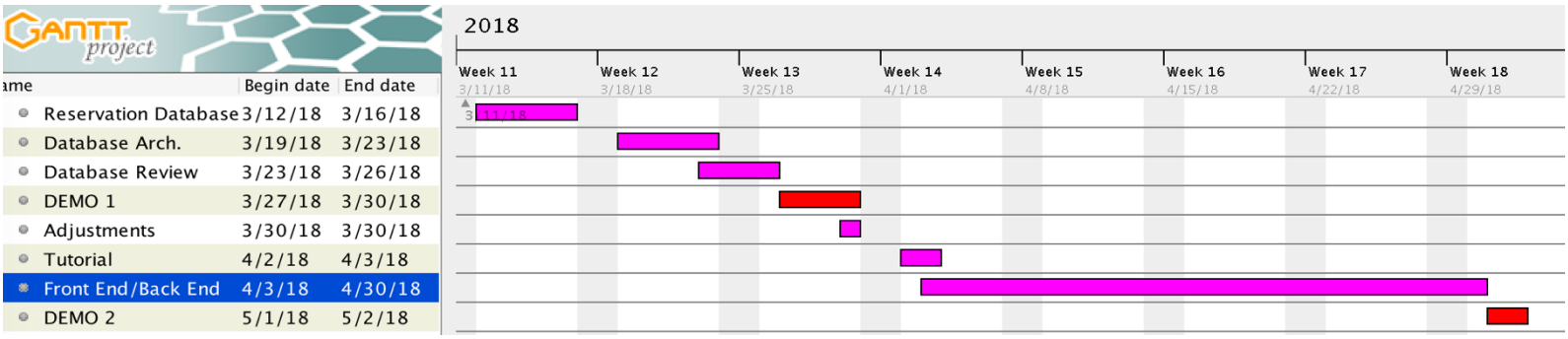
Manager

The team was easily able to divide up the task on who should do what and it was made in a way such that we would each get the same amount of points (assuming everyone only did the parts they needed to). If there were any obstacles that we faced, we either consulted each other or asked the group for clarification about the obstacle we were facing in order to find possible solutions to the issue at hand. We made sure to look at each other's work and to provide feedback so each of us know whether something needs to be edited before we submit any of our work.

Since we are coding in different languages, we as a team discussed using a framework which ended up being Django which is a python framework which makes sense since we are coding in python. Currently we're learning more about Django on how to use it. All of us have set up a Wamp server. We plan to divide our work based on teams and the manager team is currently focusing on coding the database with sample tables that will be going into the database as well as coding parts of the employee system.

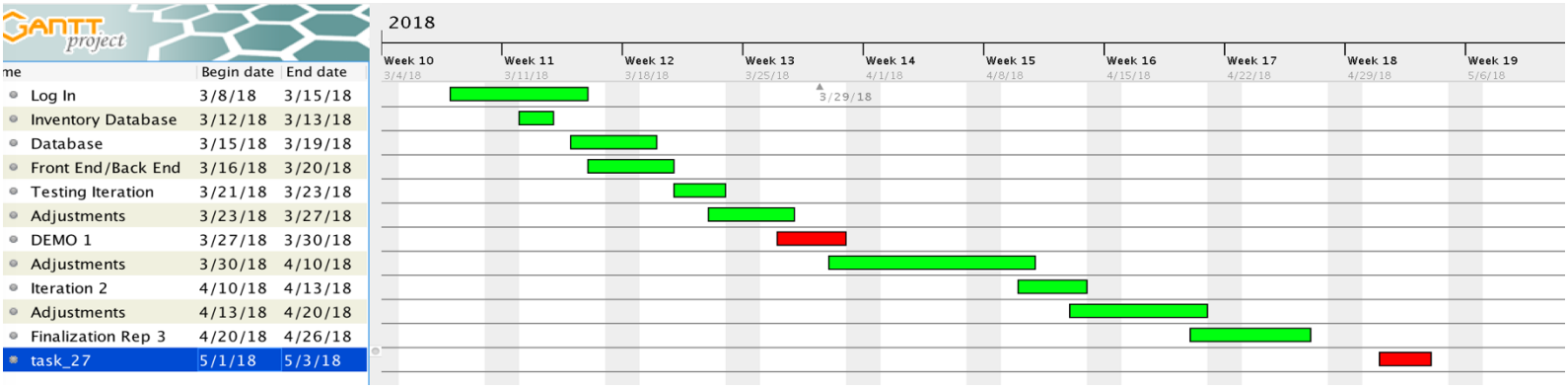
Plan of Work

Customer



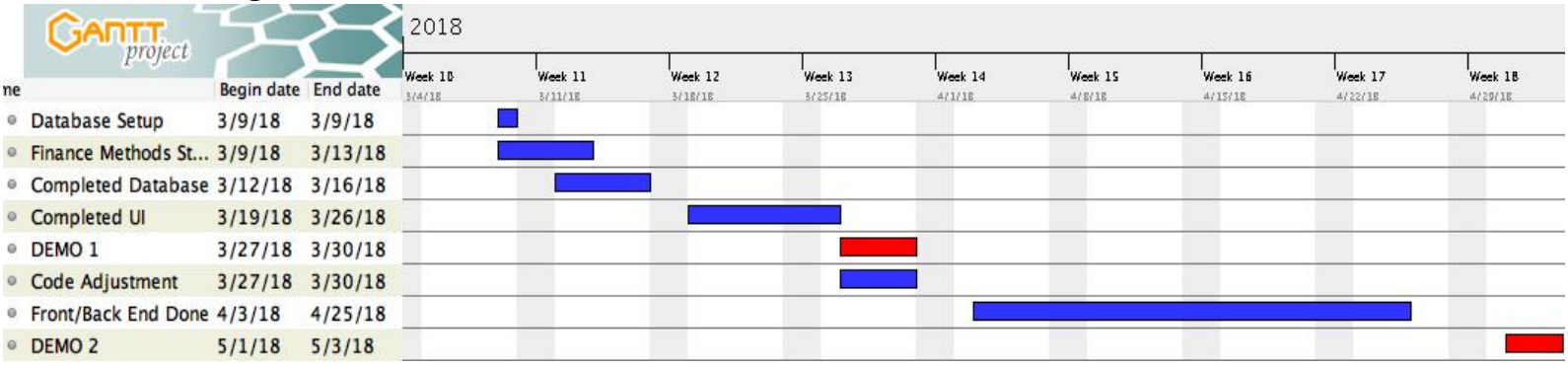
ame	Begin date	End date
Reservation Database	3/12/18	3/16/18
Database Arch.	3/19/18	3/23/18
Database Review	3/23/18	3/26/18
DEMO 1	3/27/18	3/30/18
Adjustments	3/30/18	3/30/18
Tutorial	4/2/18	4/3/18
Front End/Back End	4/3/18	4/30/18
DEMO 2	5/1/18	5/2/18

Employee



me	Begin date	End date
Log In	3/8/18	3/15/18
Inventory Database	3/12/18	3/13/18
Database	3/15/18	3/19/18
Front End/Back End	3/16/18	3/20/18
Testing Iteration	3/21/18	3/23/18
Adjustments	3/23/18	3/27/18
DEMO 1	3/27/18	3/30/18
Adjustments	3/30/18	4/10/18
Iteration 2	4/10/18	4/13/18
Adjustments	4/13/18	4/20/18
Finalization Rep 3	4/20/18	4/26/18
task_27	5/1/18	5/3/18

Managers



ne	Begin date	End date
Database Setup	3/9/18	3/9/18
Finance Methods St...	3/9/18	3/13/18
Completed Database	3/12/18	3/16/18
Completed UI	3/19/18	3/26/18
DEMO 1	3/27/18	3/30/18
Code Adjustment	3/27/18	3/30/18
Front/Back End Done	4/3/18	4/25/18
DEMO 2	5/1/18	5/3/18

Breakdown of Responsibilities

Manvi, Zachary and Esraa:

- Our plan is to code the order status algorithm.
- Program the traffic monitoring system
- Program the employee interface (log in and log out)
- Program the chef interface
- Program the employee shifts where on manager's permission, the employees can change shifts.
- Program the customer rating system.

Dafna, Hagar, and Nitya

- Develop payment and review system pages
- Develop the database connecting all the pages
- Collect all the data sent from each of the customer to the chef each time any customer orders any item.
- Developing the game for the interested customers.
- Work on displaying the time estimation for each order placed
- Developing an interface between the waiter and the customer in case they need help for ordering or for payment purposes.

John, Fareen, and Kevin

- We will be responsible for creating the database and coding the manager system which includes the employee management system, the financial management system, the menu and ratings management system, the inventory management system, and the traffic monitoring system.
- Each one of us will test the integration to see if our methods work when the database is integrated.

References

Hardware Req. Reference: https://fccid.io/XOX-Z400/User-Manual/Users-Manual-2576495

http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf

http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

https://www.tutorialspoint.com/software_testing_dictionary/big_bang_testing.htm

https://neutrium.net/mathematics/least-squares-fitting-of-a-polynomial

https://onlinecourses.science.psu.edu/stat501/node/324

https://fccid.io/XOX-Z400/User-Manual/Users-Manual-2576495