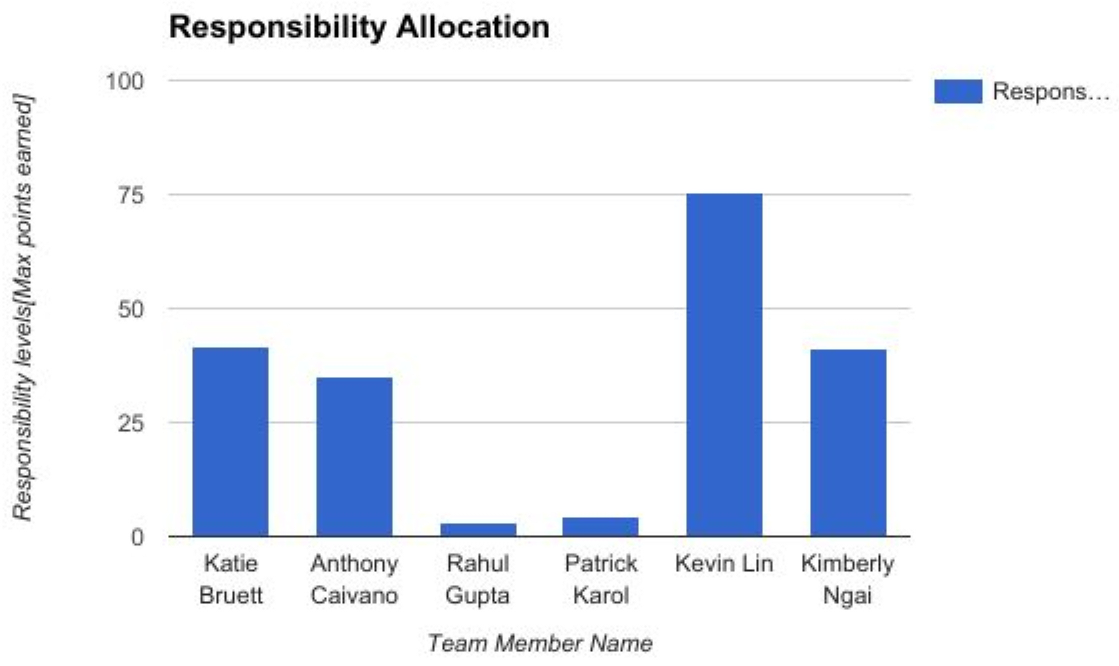
Software Engineering
Group #9
Restaurant automation
https://github.com/kevinlin47/Software-Engineering-Project

5/1/17
Katie Bruett
Anthony Caivano
Rahul Gupta
Patrick Karol
Kevin Lin
Kimberly Ngai

	Team Member Name					
	Katie Bruett	Anthony Caivano	Rahul Gupta	Patrick Karol	Kevin Lin	Kimberly Ngai
Customer Statement of Requirements (6 Points)	10%	15%		10%	65%	10%
Glossary of Terms (4 points)					100%	
System Requirements (6 points)				5%	95%	
Functional Requirements Specification (30 points)	13%	47%				40%
Effort Estimation using Use Case Points (4 points)					100%	
Domain Analysis (25 points)	60%	5%		10%	15%	10%
Interaction Diagrams (40 points)					99%	1%
Class Diagram and Interface Specification (20 points)		85%				15%
System Architecture and System Design (15 points)	65%				22%	13%
Algorithms and Data		10%				90%

Structures (4 points)						
User Interface Design and Implementation (11 points)	100%					
Design of Tests (12 points)			25%		50%	25%
History of Work (5 points)	20%	20%		20%	20%	20%
Summary of changes (5 points)	5%	5%			85%	5%
Project Management (13 points)			2%			98%



Summary of Changes:

Customer Statement of Requirements:

Added headings to customer statement of requirements as requested.

Glossary of Terms:

Removed some terms that did not seem relevant.
Added two terms that are relevant to the restaurant business.

User Stories:

Edited a few of the user stories to make them more clear.

Functional System Requirements:

Separated traceability matrices as requested by the grader.
Added details into fully-dressed descriptions to demonstrate more events.
Remade the "order done" sequence diagram.

Domain Analysis:

Re-wrote the attribute definitions.

Interaction Diagrams:

Added more detail in the description for interaction diagrams.
Updated some diagrams to reflect current program.
Explained design pattern choice when used, and added a diagram for customer and chef interaction as suggested.

System Architecture and Design:

Re-wrote "Persistent Data Storage" to reflect change in storage
Re-wrote "Network Protocol" to incorporate new communication protocol and reasoning
Added missing information about the timedb.

Algorithms and Data Structures:

Re-wrote data structures and included algorithms.

Design of Tests:

Added more detail to pass and fail criteria for test cases.

Class Diagram and Interface Specification:

Revised the class diagram to reflect the final design and updated descriptions.

User Interface Design and Implementation:

Updated the information of the user interface and added more detail about the changes and how the entire UI works together with a simple ease of use.

Additional Changes:

Included estimation using use case points.
Updated history of work and added future work.
Added additional references.
Included what sections the references were used for as requested.

1. Customer Statement of Requirements

1.1 Problem Statements

1.1.1 Wasting Time

Everyone says time is money, a statement that proves inaccurate in any business. Doing things quickly and efficiently promotes business, from increases in sales and transactions to outputting more customers who are satisfied with the time in which they were served and their experience in general at the business. In order to reap these benefits, implementing a restaurant management software would be beneficial to time efficiency. Areas in the restaurant that would benefit are inventory tracking, receiving customer orders, delivering customer orders,and table management.

1.1.1.1 Inventory Organization

Inventory tracking requires a significant amount of time because of the need to count each individual item and keep it logged on paper or other software, for example microsoft excel, taking up time that could be used elsewhere in the restaurant. A way to cut down the amount of time needed to track inventory would lead to more time improving other aspects of the restaurant. The accumulation of time acquired from employing this solution would later lead to an increase in the restaurant's savings because it can do more tasks and hire fewer employees. An example of this is emphasized in the inventory tracking section of the solution program wherein an autonomously tracked inventory would have to be initialized by a chef or manager. As orders are filled in the kitchen, the system would deduct from the inventory the quantity of ingredients that went into a particular dish. However, unless the chefs are extremely accurate in their measurements, this inventory would quickly turn into an approximation. To quickly adjust for this, the chefs should be able to adjust the inventory in real time, which would keep the rest of the kitchen (and to a lesser degree, the management) better informed on the available stock. Furthermore, on a time-scale appropriate to the restaurant, the inventory could be reset and reinitialized to further improve the accuracy. At the end of the day the inventory can be easily analyzed, and help in making restocking decisions.

1.1.1.2 Customer Service

Receiving customer orders and delivering customer orders come hand in hand and there are many ways to make this more efficient. Currently, the most common method among restaurant businesses is with pen and paper. The waiter writes down on their notepad the customers' desired order from the menu, and the waiter then proceeds and delivers it to the chefs in the kitchen. The waiters then come back and check to see if the order has been completed, and if so, the waiter can then bring the order to the correct customer. The saying “if it ain’t broke, don’t fix it” certainly can apply to this situation, but there are more efficient ways of doing this, utilizing both software and technology. Waiters will use a tablet to input orders to the kitchen. The chefs in the kitchen will then receive the orders through their computer system in the kitchen. The orders will appear in a queue style to keep them organized and fair. Once the order is complete, the chefs can notify the waiters through the software that their order is complete and ready to be delivered to the customer. This benefits the restaurant due to the fact that orders are sent through the computer, waiters do not have to enter the kitchen, except to pick up a completed order. After an order is initially placed, the tablet it left at the table for the customers to order any additional items they may need or have access to notify the waiters if they are needed for any reason. This saves the waiters time and allows them to be more attentive to their tables, resulting in better customer service reviews. Furthermore, during peak restaurant hours in a restaurant with a traditional ordering system, the kitchen can quickly be sent into a state of confusion and chaos, resulting in missed, delayed, or ruined orders; streamlining the ordering process would eliminate such a situation. Lastly, instead of waiters having to constantly check up on their submitted orders to see if they are ready, they can be instantly be notified by the chefs in the kitchen when their order is complete and ready for delivery.

1.1.1.3 Table Management

Table management is important in keeping the flow of the restaurant in order; having the tables cleaned and ready to serve customers as quickly as possible would only improve efficiency. The flow of customers coming in and out at a fast pace is very important, especially when there are many customers at the given moment. The restaurant does not want customers to have to wait for the tables to be cleaned. This drops customer satisfaction and income. The more customers the restaurant serves, the more money it makes, and the fewer the restaurant serves the less amount of money it makes. Also, if customers have to wait an extra few minutes they may leave and go somewhere else to eat. As resolution, waiters can have a graphical user interface on their computers or terminals to notify when a table needs to cleaned. This would let the manager or bus boys know that a table is vacant and needs to be cleaned immediately in preparation for the next incoming customer. This way the staff responsible for cleaning tables can do so right away, instead of missing the information that a table needs to be cleaned. Also waiters and managers will be able to set a table as occupied or vacant. This allows quick checking of table availability and will allow staff to take reservations.

1.1.2 Tracking Statistics

Tracking various restaurant related statistics is important in formulating good business decisions. Being able to track what items are being sold the most and what items are being sold the least is valuable. If the restaurant knows which items are being sold more frequently, it can make better decisions on ordering more inventory for items that are required to prepare this item. Conversely, knowing which items are being sold the least can help the restaurant decide to order less of the inventory required to prepare this item, or even just completely remove the item from the menu. This method of keeping an eye on popularity will allow the restaurant to know when exactly to start promotion of a failing item or lower the price in order to sell more and limit the amount of wasted inventory. This will save the restaurant money in the long run, and because the inventory is automatically kept track of in the system, this takes away the burden of tracking the information automatically.

Customer rating is a vital part in the efficiency of a restaurant. At the end of service, each customer would be presented with a survey on the tablet on their table, presenting them with questions concerning the different aspect of their service. This would allow management to track which departments of the restaurant are in need for improvement. It also gives management a clear idea of how customers are generally feeling about the restaurant in general or how the food is to see if any changes in the menu or recipes are needed.

1.1.3 Managing Employees

Tracking the amount of hours an employee works is needed to pay them correctly. Keeping track of hours can be done with a simple sheet of paper, the employee signs in when they show up, and signs out once they are done. While this method works, in a growing technology-based world this can be done by computer programs, which can save an incredible amount of time in calculating total employee hours per week. Any employee, be it a waiter, chef, busboy, or manager can use the built in feature of the software to make sure their work hours are correctly saved onto the system. Then once pay time comes along, individual pay can be easily calculated.

To operate this system, the employee punches in their assigned employee number at the start of their shift. The program automatically takes the current time on the computer as the start time, so the employee does not need to enter any information other than they are here to start their shift, and once their shift is done, the same procedure occurs the employee punches out signaling the system they are done with their shift. This procedure easily calculates the total time that was worked per day, per employee.

Doing it this way as some benefits, such as not requiring the employee to write down any information about time avoiding completely the situation if an employee was to fraud their start and end times. Using the computer time will avoid this problem and provide the most accurate start and end time. Also this avoids the possibility of losing the shift sheet, for any reason such as accidentally throwing it away, spilling food on it, or a multitude of other reasons. Once on the computer, it is safe from human errors. Once pay time comes along and the payroll must be done, the program will automatically calculate the employee's pay, taking into account overtime hours, and display it on the computer. Therefore, the manager would simply have to is handle the direct deposit to the employee or send the information to whoever is handling the employee checks.

1.2 Glossary of Terms

1.2.1 Technical Terms

Program: a sequence of coded instructions that can be inserted into a mechanism (as a computer).

Software: The entire set of programs, procedures, and related documentation associated with a system and especially a computer system.

Terminal: Any device for entering information into a computer or receiving information from it, as a keyboard with video display unit,either adjoining the computer or at some distance from it.

Queue: A waiting line especially of persons or vehicles.

Graphical User Interface: A computer program designed to allow a computer user to interact easily with the computer typically by making choices from menus or groups of icons.

Application: A program (such as a word processor or a spreadsheet) that performs a particular task or set of tasks.

Feature: A prominent part or characteristic.

1.2.2 Non-Technical Terms

Efficiently: Productive without waste.

Initial: Of or relating to the beginning.

Waiter: A person who waits tables (as in a restaurant).

Chef: A skilled cook who manages the kitchen (as of a restaurant).

Order: To give an order for <*order* a meal>.

Chaotic: A state of utter confusion.

Reservation: An act of reserving something.

Statistics: A collection of quantitative data.

Formulate: To prepare according to a formula.

Modification: The making of a limited change in something.

Dish: An individual order of food off of the menu.

Clock-in/Punch-in: To record the time an employee arrived at work for a shift.

Clock-out/Punch-out: To record the time an employee left work at the end of a shift.

2. System Requirements / User Stories

Priority Key:

ST-X-#: High priority (highly likely to be implemented)
ST-X-#: Middle priority (likely to be implemented)
ST-X-#: Low priority (highly unlikely to be implemented)

Points are on a scale of 1-10 and they indicate an estimate of the effort needed to complete the user story based on its complexity and the team's skills.

2.1 Manager

Identifier	User Story	Time Estimated to Complete
ST-M-1	As a manager, I can add and remove food items from the menu based on sales.	Points 7

🟩 **ST-M-2**	As a manager, I can add and remove employees from the system.	Points 6
🟨 **ST-M-3**	As a manager, I can see which items are selling well and which items are selling not so well	Points 6
🟨 **ST-M-4**	As a manager, I can change employee salaries	Points 5
🟩 **ST-M-5**	As a manager, I can adjust food prices.	Points 8
🟥 **ST-M-6**	As a manager, I can see when an employee started and ended their shifts this week.	Points 4
🟩 **ST-M-7**	As a manager, I can keep track of restaurant inventory.	Points 5
🟩 **ST-M-8**	As a manager, I can see the total amount of money earned from sales this week.	Points 7
🟥 **ST-M-9**	As a manager, I can see the cost of restocking inventory.	Points 6
🟥 **ST-M-10**	As a manager, I can set table reservations.	Points 5
🟨 **ST-M-11**	As a manager, I can schedule employee shifts.	Points 6
🟩 **ST-M-12**	As a manager, I can edit how much money is allocated to the budget	Points 7
🟥 **ST-M-13**	As a manager, I can see and approve or deny any employee request for days off.	Points 6
🟥 **ST-M-14**	As a manager, I can handle all maintenance requests	Points 5

2.2 Waiter

Identifier	User Story	Size
ST-W-1	As a waiter, I can create customer orders.	Points 7
ST-W-2	As a waiter, I can send customer orders to the chef	Points 8
ST-W-3	As a waiter, I can notify when a table needs to be cleaned.	Points 8
ST-W-4	As a waiter, I can notify when a table is being occupied.	Points 8
ST-W-5	As a waiter, I can find the total amount of a customer's order.	Points 6
ST-W-6	As a waiter, I can see the total amount of hours worked this week.	Points 6
ST-W-7	As a waiter, I can see my shift schedule for the week.	Points 7
ST-W-8	As a waiter, I can clock in and out of my shift.	Points 6
ST-W-9	As a waiter, I can request a day off.	Points 5

2.3 Chef

Identifier	User Story	Size
ST-Ch-1	As a chef, I can notify waiters	Points 8

🟩	when their orders are complete.	
🟨 **ST-Ch-2**	As a chef, I can see how much inventory is remaining	Points 9
🟥 **ST-Ch-3**	As a chef, I can notify the manager when inventory is getting low or completely gone.	Points 9
🟩 **ST-Ch-4**	As a chef, I can receive customer orders from waiters and have it displayed to me on a computer screen in a list .	Points 8
🟨 **ST-Ch-5**	As a chef, I can remove items from the menu if not enough inventory is available.	Points 9
🟩 **ST-Ch-6**	As a chef, I can designate the ingredients and cost of every dish on the menu.	Points 7
🟨 **ST-Ch-7**	As a chef, I can add new dishes to the menu.	Points 7
🟨 **ST-Ch-8**	As a chef, I can edit the inventory.	Points 6
🟩 **ST-Ch-9**	As a chef, I can see the popularity for each menu item.	Points 9
🟥 **ST-Ch-10**	As a chef, while viewing popularity of menu items, can search and sort items.	Points 5
🟩 **ST-Ch-11**	As a chef, I can clock in and out of my shift.	Points 6
🟩 **ST-Ch-12**	As a chef, I can see how many hours I've worked in the week.	Points 6

2.4 Customer

Identifier	User Story	Size
ST-Cu-1	As a customer, I can order more food items.	Points 8
ST-Cu-2	As a customer, I can complete a survey.	Points 9
ST-Cu-3	As a customer, I can pay my bill.	Points 7

3. Functional System Requirements

3.1 Stakeholders

Below is a list of all who might benefit from this system:

- Chef
  Chefs will be able to more efficiently run their kitchen and delegate tasks
- Waiter
  Waiters will be able to serve customers more efficiently
- Management
  Management will be able to handle finances and employees with ease and efficiency
- Customer
  Customers will be able to have a better dining experience

3.2 Actors

3.2.1 Initiating Actors

- Chef
  - Role: Chefs will be responsible for viewing orders and electronically notifying wait-staff when orders are finished.
  - Goal: Prepare food, take note of food inventory, and handle the menu
- Manager

    - Role: Managers will be responsible for running the restaurant
    - Goal: Manage the budgets and revenue, handle employees, and maintain restaurant
- Waiter
    - Role: Waiters will be in charge of interacting with the customers
    - Goal: Keep track of food progress for customers
- Customer
    - Role: Customers are visitors to the restaurant who will order food
    - Goal: Place orders, receive orders, and pay for them

3.2.2 Participating Actors

- Database
    - Role: The database stores all data that will be accessed and used by the system
    - Goal: Store any information/data regarding inventory, revenue, budgets, employees, maintenance, etc.

3.3 Use Cases

3.3.1 Casual Descriptions

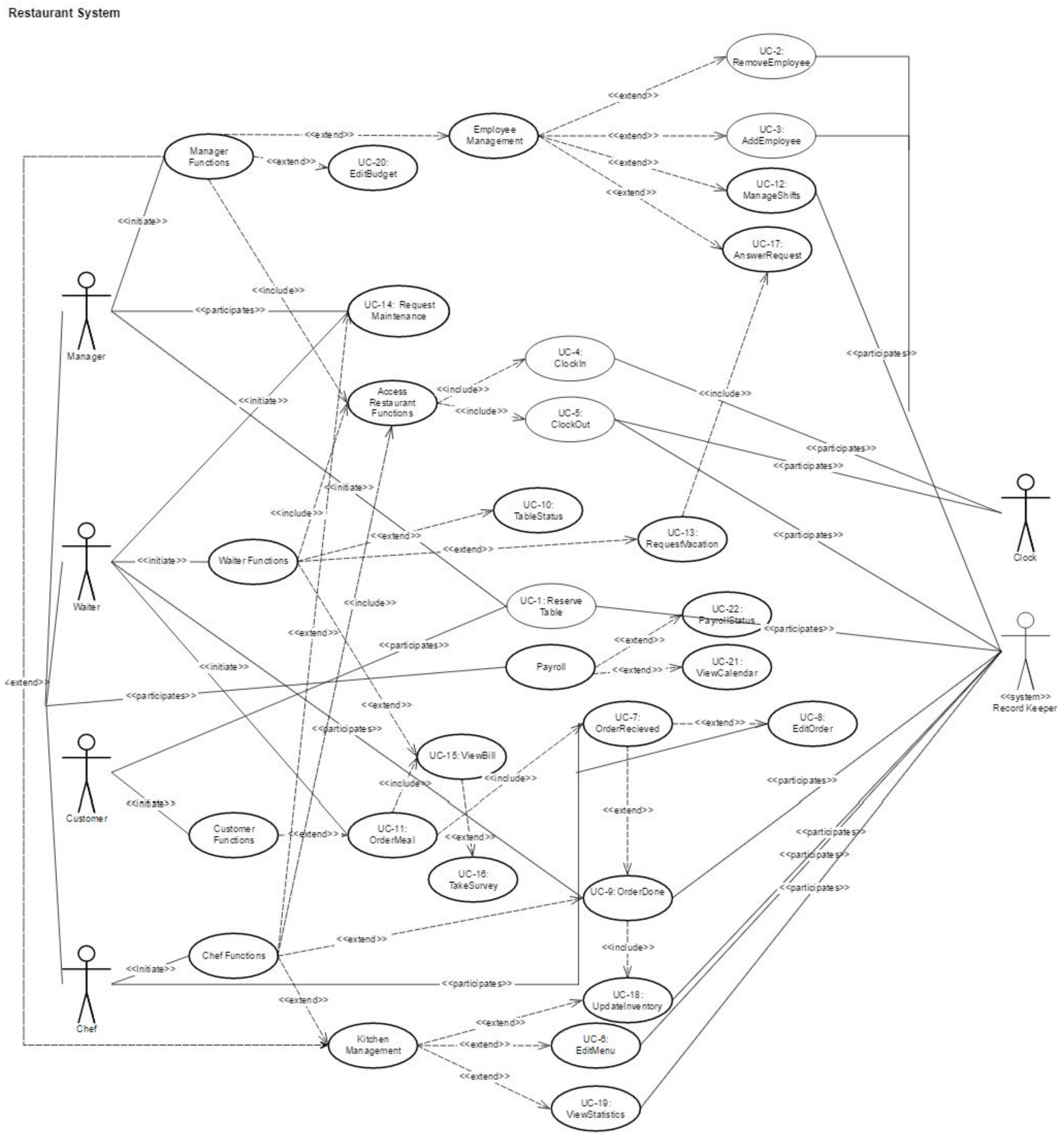
Use Case	Use Case Name	Description	Corresponding User Story
UC-1	ReserveTable	Shows available tables and can be selected to be reserved	**ST-M-10**
UC-2	RemoveEmployee	Removes an employee from the system when they are fired.	**ST-M-2**
UC-3	AddEmployee	Adds an employee to system when they are hired	**ST-M-2**
UC-4	ClockIn	Employees clock in for their shift	**ST-W-8, ST-Ch-11**

UC-5	ClockOut	Employees clock out of their shift	**ST-W-8, ST-Ch-11**
UC-6	EditMenu	Edit the menu by removing or adding items and adjusting prices	**ST-M-1, ST-M-5, ST-Ch-5, ST-Ch-6, ST-Ch-7**
UC-7	OrderReceived	Shows a customer's order as received by kitchen and puts into queue for chefs	**ST-W-1, ST-W-2, ST-Ch-4**
UC-8	EditOrder	Edits order that was already placed by the customer and notifies the chef and customer of change	**ST-Cu-1**
UC-9	OrderDone	Alerts waiters that order is done and ready to be picked up. Updates inventory to account for ingredients used and food made.	**ST-Ch-1**
UC-10	TableStatus	Shows status of table as clean, dirty, vacant, or occupied. If not clean, notifies waiters that it needs to be cleaned. Status can be changed by employees.	**ST-W-3, ST-W-4**
UC-11	OrderMeal	Places customer's order to the system and sends to chefs	**ST-Cu-1**
UC-12	ManageShifts	Managers can edit calendar to schedule	**ST-M-11**

		employees to shifts. Can add, remove, and edit employees to shifts.	
UC-13	RequestVacation	Lets employees notify/request manager of taking a day off	**ST-W-9**
UC-14	RequestMaintenance	Employees can request an item or equipment for maintenance	**ST-M-14**
UC-15	ViewBill	Allows customer to view and pay for their bills.	**ST-W-5, ST-Cu-3**
UC-16	TakeSurvey	After payment, shows a survey that needs to be filled out by customer and sends results to system.	**ST-Cu-2**
UC-17	AnswerRequest	Managers can look at any requests for vacation or maintenance and approve or reject	**ST-M-13**
UC-18	UpdateInventory	Keeps track of ingredient quantity as orders are placed and ingredients are bought. Also keeps track of food ordered and profit from food. Notifies chefs and managers if running low of ingredient(s).	**ST-M-7, ST-Ch-8**

UC-19	ViewStatistics	Displays budgets, revenue, food popularity, and inventory and inventory costs. Chefs can only view inventory and popularity.	**ST-M-3, ST-M-8, ST-M-9, ST-Ch-3, ST-Ch-9, ST-Ch-10**
UC-20	EditBudget	Allows manager to change how much money goes to budget	**ST-M-12**
UC-21	ViewCalendar	Allows employees to see their shift schedule for the day, week, or month	**ST-M-6, ST-W-7**
UC-22	PayrollStatus	Displays the total number of hours an employee worked for a chosen week and how much money they've earned	**ST-M-4, ST-W-6, ST-M-6, ST-Ch-12**
UC-23	CustomerOrder	Allows customer to order additional menu items from their table.	**ST-Cu-1**

3.3.2 Use Case Diagram



3.3.3 Traceability Matrices

3.3.3.1 Manager Matrix

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19	UC-20	UC-21	UC-22	UC-23
ST-M-1						X													X				
ST-M-2		X	X																				
ST-M-3																			X				
ST-M-4																						X	
ST-M-5						X																	
ST-M-6																					X	X	
ST-M-7																		X	X				
ST-M-8																			X				
ST-M-9																			X				
ST-M-10	X									X													
ST-M-11												X											
ST-M-12																				X			
ST-M-13																	X						
ST-M-14														X									

3.3.3.2 Waiter Matrix

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19	UC-20	UC-21	UC-22	UC-23
ST-W-1							X				X												
ST-W-2							X				X												
ST-W-3										X													
ST-W-4										X													
ST-W-5															X								
ST-W-6																						X	
ST-W-7																					X		
ST-W-8				X	X																		
ST-W-9													X										

3.3.3.3 Chef Matrix

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19	UC-20	UC-21	UC-22	UC-23
ST-Ch-1									X														
ST-Ch-2																			X				
ST-Ch-3																		X	X				
ST-Ch-4							X																
ST-Ch-5						X																	
ST-Ch-6						X																	
ST-Ch-7						X																	
ST-Ch-8																		X					
ST-Ch-9																			X				
ST-Ch-10																			X				
ST-Ch-11				X	X																		
ST-Ch-12																						X	

3.3.3.4 Customer Matrix

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19	UC-20	UC-21	UC-22	UC-23
ST-Cu-1								X			X												X
ST-Cu-2																X							
ST-Cu-3															X								

3.3.4 Fully-Dressed Descriptions

3.3.4.1 UC-6 (Edit Menu)

ID: UC-6
Title: Edit Menu

Primary Actor: Chef
Actor's Goal: To edit the menu
Secondary Actor: Database
Preconditions: Chef is logged in to the system
Success End Condition: Menu is edited
Failed End Condition: Menu is not updated with new items/edits
Main Success Scenario:

1. System displays main menu with various options for Chef to pick from
2. Chef presses "Manage Menu"
3. System shows list of items with corresponding prices
4. Chef presses "Add Item"
5. System prompts for name, ingredients, and price
6. Chef enters name, ingredients, and price of new food
7. Chef presses "Submit"
8. System displays confirmation that item was added
9. System returns to menu list
   Chef repeats Steps 4 to 8 until satisfied with changes
10. Chef presses "Return" to return to main menu
11. System brings Chef back to main menu

Extensions (Alternate Scenarios):
4a. Chef presses "Remove Item"

1. Systems prompts Chef to enter name of item to be removed
2. Chef enters name of existing food item
3. System prompts for confirmation
4. Chef presses "Yes"
5. System removes food from menu and displays confirmation that it was removed
6. System returns to menu list
   Chef repeats above steps until satisfied with changes
7. Chef presses "Return"
8. System brings Chef to main menu

4a2a. Chef enters name of non-existing food item

1. System prompts for confirmation
2. Chef presses "Yes"
3. System notifies Chef that item does not exist and goes to Step 4a1 of Alternate Scenarios

6a. Chef enters existing food name with different price and ingredients and presses "Submit"

1. System updates existing item with new price or ingredients
2. System displays confirmation that item was changed
3. System goes to Step 3 of Main Success Scenario

6b. Chef enters existing food name with same price and ingredients and presses "Submit"

1. System notifies Chef that item already exists with same values
2. System does not make any changes to menu/database
3. System goes to Step 3 of Main Success Scenario

7a. Chef presses "Cancel"

1. System returns to menu list and goes to Step 3 of Main Success Scenario

8a. System unable to process change due to connection error

1. System detects error and (a) cancels process, (b) displays error message to Chef, and (c) goes to Step 3 of Main Success Scenario

Frequency of Use: Once or twice a month
Priority: Low

3.3.4.2 UC-9 (Order Done)

ID: UC-9
Title: Order Done
Primary Actor: Chef
Actor's Goal: To notify waiters of completed order
Secondary Actors: Waiter, Database
Preconditions: An order has been placed, Chefs are logged in to the system with designated cooking stations, and cooking is underway
Success End Condition: Waiters are notified of a cooked meal and order list is repopulated
Failed End Condition: Unable to notify waiters of meal
Main Success Scenario:

1. Chef presses "Complete Component" when he is done cooking a food item
2. System notifies Chef that his contribution has been noted
   Repeat Steps 1 and 2 until all Chefs are done cooking the entire order
3. Chef presses "Order Done"
4. System notifies waiters that order is ready to be picked up
5. System updates order list with current unfinished orders

Extensions (Alternate Scenarios):
2a. System unable to receive Chef's update due to network error

1. System notifies Chef of error and goes to Step 1 of Main Success Scenario

4a. System unable to notify waiter due to network error

1. System notifies Chef of error sending notification and goes to Step 3 of Main Success Scenario

Frequency of Use: Several times per hour.
Priority: Medium

3.3.4.3 UC-19 (View Statistics)

ID: UC-19

Title: View Statistics
Primary Actor: Manager
Actor's Goal: To view statistics of restaurant
Secondary Actor: Database
Preconditions: Manager is logged into system.
Success End Condition: System displays desired statistics and Manager goes back to main menu
Main Success Scenario:

1. System displays Manager main menu
2. Manager presses "View Statistics" from main menu
3. System displays daily statistics that include revenue, food popularity, etc.
4. Manager presses "Return" to return to main menu
5. System returns Manager to main menu

Extensions (Alternate Scenarios):
3a. Manager presses "Weekly" option near top of displayed window

1. System displays weekly statistics
2. Manager presses "Return"
3. System returns to main menu

3b. Manager presses "Monthly" option

1. System displays monthly statistics
2. Manager presses "Return"
3. System returns to main menu

3c. Manager presses "Annual" option

1. System displays monthly statistics
2. Manager presses "Return"
3. System returns to main menu

Frequency of Use: Once or twice daily.
Priority: High

3.3.4.4 UC-22 (Payroll Status)

ID: UC-22
Title: Payroll Status
Primary Actor: Manager
Actor's Goal: To view the payroll status of all employees
Preconditions: Manager is logged in to the system
Secondary Actor: Database
Success End Condition: Manager sends payroll to be handled and Manager returns to Manager main menu
Failed End Condition: Manager is unable to send payroll

Main Success Scenario:

1. Manager selects “Payroll” from the main menu
2. System displays employee list with corresponding hours worked and subsequent pay
3. Manager selects “Submit”
4. System asks for confirmation
5. Manager presses “Yes”
6. System sends confirmation to Manager and sends the list to payroll and returns to main menu

Extensions (Alternate Scenarios):

3a. Manager selects “Return”

1. System returns to main menu

6a. System encounters network error

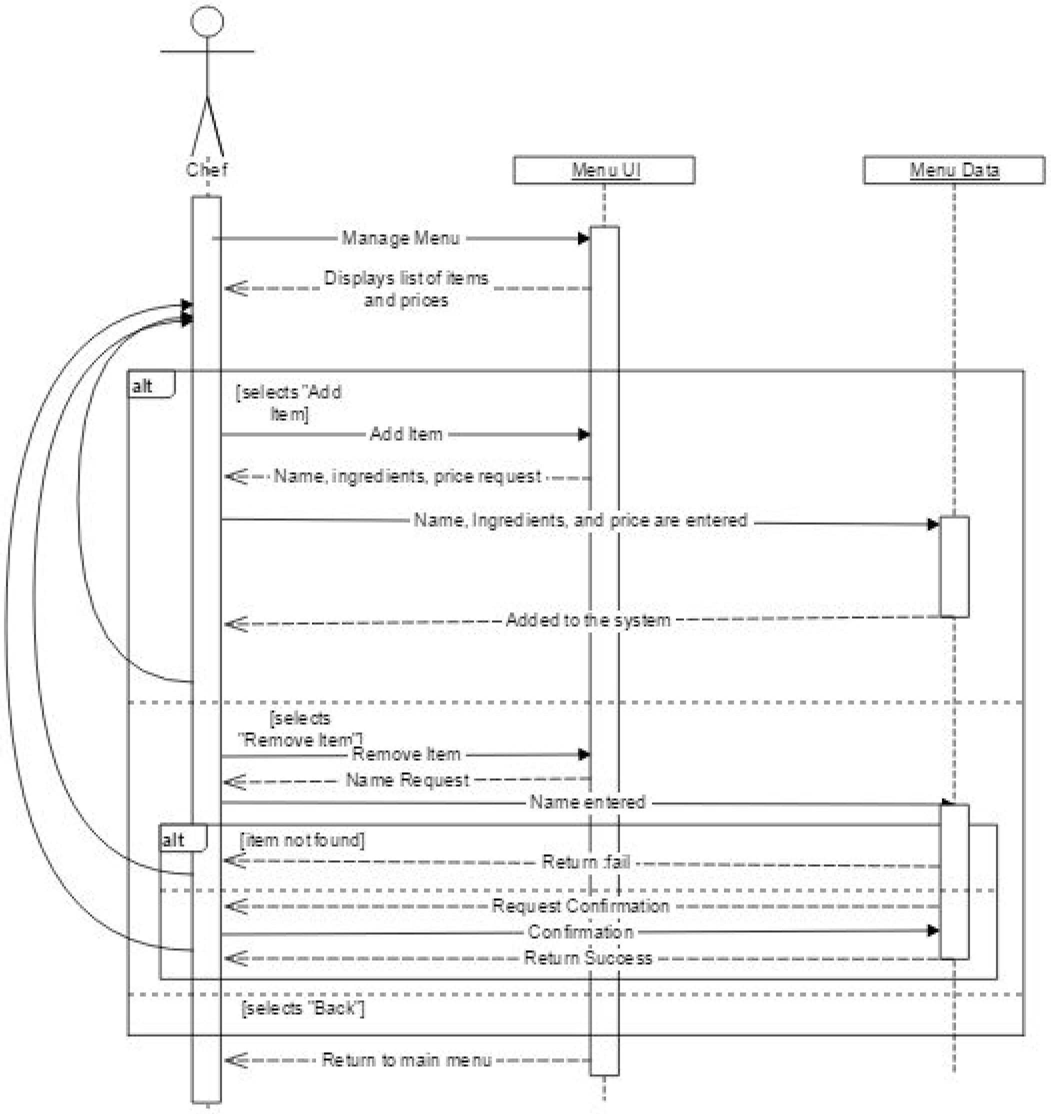
1. System notifies Manager of error and (a) doesn’t send the payroll and (b) goes to Step 2 of Main Success Scenario

Frequency of Use: Once a Week.

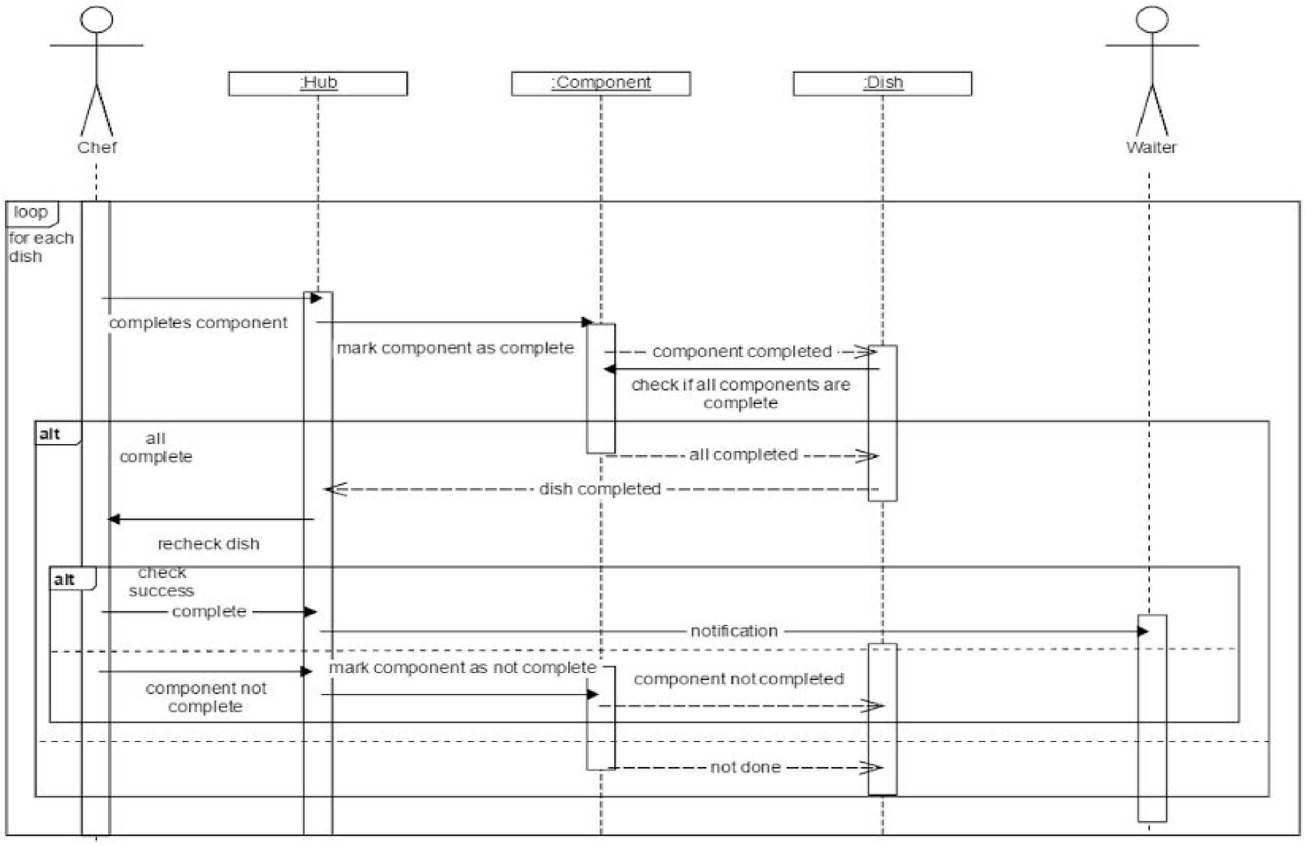
Priority: High

3.4 System Sequence Diagrams

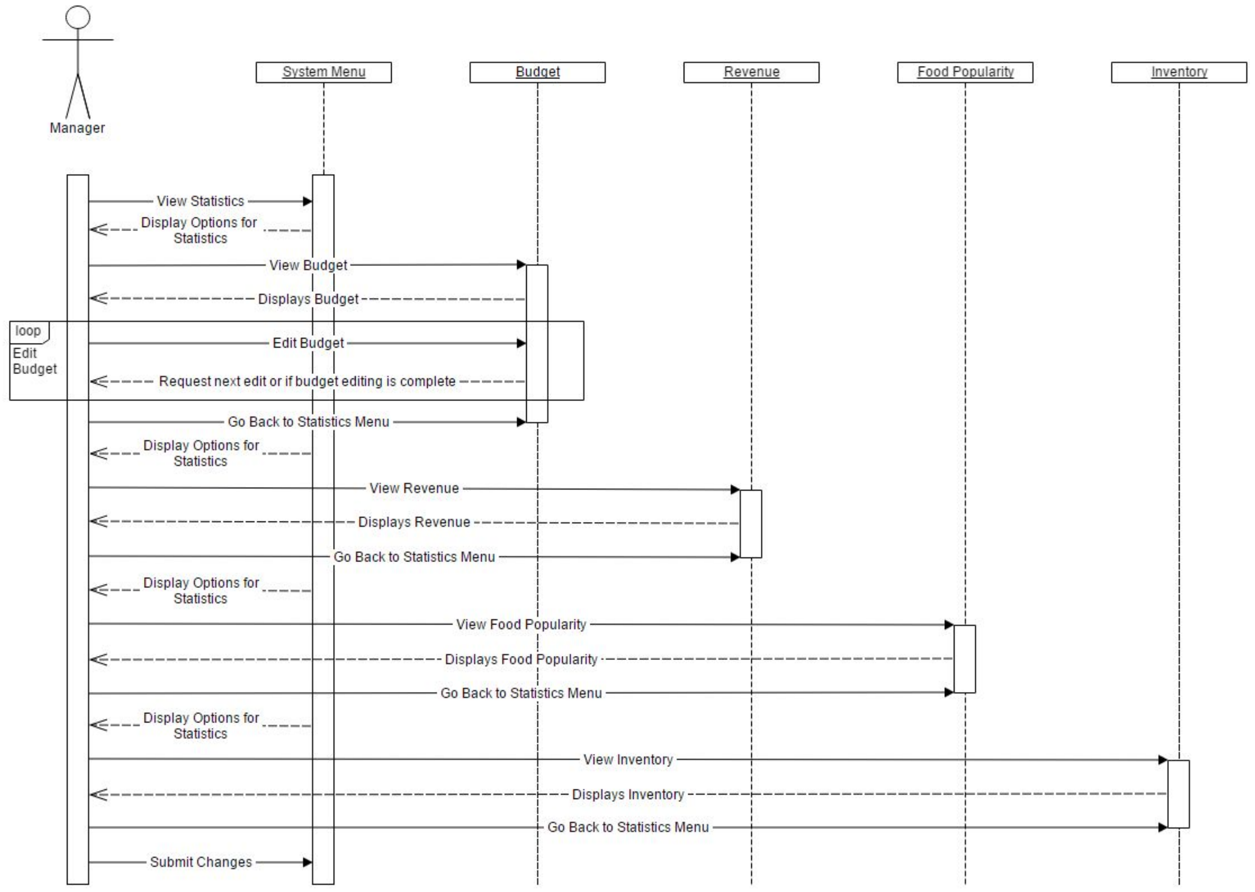
3.4.1 UC-6 (Edit Menu)



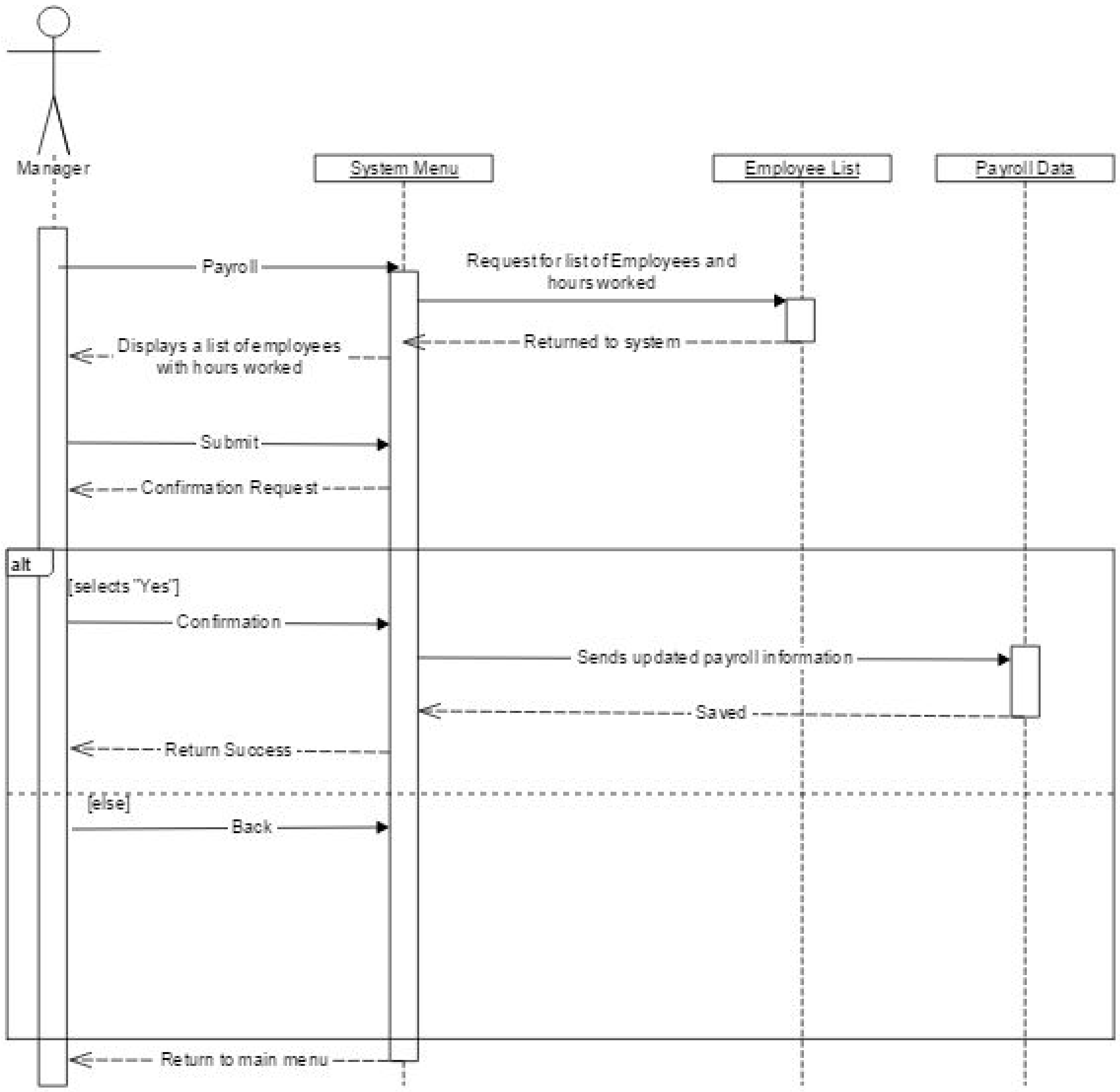
3.4.2 UC-9 (Order Done)



3.4.3 UC-19 (View Statistics)


Manager
System Menu
Budget
Revenue
Food Popularity
Inventory
View Statistics
Display Options for Statistics
View Budget
Displays Budget
loop
Edit Budget
Edit Budget
Request next edit or if budget editing is complete
Go Back to Statistics Menu
Display Options for Statistics
View Revenue
Displays Revenue
Go Back to Statistics Menu
Display Options for Statistics
View Food Popularity
Displays Food Popularity
Go Back to Statistics Menu
Display Options for Statistics
View Inventory
Displays Inventory
Go Back to Statistics Menu
Submit Changes

3.4.4 UC-22 (Payroll Status)



4. User Effort Estimation

Scenario 1: Waiter Places Order

1. Navigation: Select the order menu option.
    a) Select the desired items from each category

b) Once all desired items are in the order, select “Submit Order” to send the order to the chef’s computer display.

Scenario 2: Chef Prepares Order

2. Navigation: Select the “View Orders” option.

a) All current orders are displayed on screen to the chef in order of time being sent.

b) Chef selects the completed order in the list and then clicks “Done” once a order is complete, and notifies the waiter that it is ready to be delivered to the customer.

Scenario 3: Waiter requests table cleaned

3. Navigation: Select the “View tables” option.

a) All tables in the restaurant and their status (occupied, free,reserved, dirty) will be shown to the waiter on their screen.
b) Waiter selects a table on screen and selects the option to edit currently selected table’s status.
c) Waiter then chooses the status to “needs to be cleaned”.

Scenario 4: Customer responds to optional survey

4. Navigation: Select the “Take Survey” option.

a) Three questions will be displayed to the customer on screen. One relating to quality of the food, one relating to service time, and one relating to quality of service provided by the waiter.
b) Customer will select their response to each question by choosing a value from 1-5 on a drop down list. (1 = poor, 5 = Great).Finally a optional additional comments text area will be provided for any additional comments.

$UCP = UUCPxTCFxECF$

UCP = Use Case Points;
UUCP = Unadjusted Use Case Points;
TCF = Technical Complexity Factor;
ECF = Environment Complexity Factor;

Weight:
Simple → 5
Average → 10
Complex → 15

Use Case	Use Case Weight
UC-9: Order Done	10
UC-10: Table Status	10
UC-11: Submit Order	10
UC-16: Take Survey	5
UC-19: View Statistics	15

$UUCP = 10 + 10 + 10 + 5 + 15 = 50$

TCF	Technical complexity Factor Weight	TCF Perceived Complexity	Complexity Factor
1	1	4	4
2	1	4	4
3	1	4	4
4	0.5	2	1
5	2	5	10

$TCF = 0.6 + (0.01 * Total\ Complexity\ Factor$
$TCF = 0.6 + (0.01 * 23)$
$TCF = .83$

ECF	Description	Environmental Complexity Factor Weight	ECF Perceived Impact	Complexity Factor
E1	Beginner familiarity with the UML based development	1	2	2
E2	Some familiarity	1	3	3

	with application problem			
E3	Some knowledge of object-oriented approach	3	3	9
E4	Highly motivated, but some team members occasionally slacking	2	3	6
E5	Programming language of average difficulty will be used	2	4	8

$ECF = 1.4 - (0.0.3 * Total\ Complexity\ Factor)$
$ECF = 1.4 - (0.03 * 28)$
$ECF = .56$

$UCP = UUCPxTCFxECF$
$UCP = 50 * .83 * .56$
$UCP = 23.24$

$Duration = PF * UCP$

PF = Productivity Factor;

$Duration = 28 * 23.24$
$Duration = 650.72\ hours$

5. Domain Analysis

5.1 Domain Model

5.1.1 Concept Definitions

Description	**Type**	**Concept**
Contains information about vacancy and cleanliness of tables	K	TableStatus
Changes table to occupied, vacant, or dirty	D	ChangeTable
Contains information about food items being sold	K	Menu
Changes menu items, prices, etc.	D	ModifyMenu
Stores and displays any and all information regarding inventory	K	Inventory
Allows changes to be made to inventory	D	EditInventory
Employees can clock in and out of their shifts	D	ReportTime
Contains information about an employee’s pay and displays the hours they’ve worked	K	Payroll
Displays how budget is allocated	K	Budget
Edits amount of money towards the budget	D	EditBudget
Shows profit gain/loss for a certain period of time	K	Revenue
Displays lists of actions an actor can pick/do based on previous actions or current situation	K	SystemInterface
Adds or removes employee from the system and allows edits to be made to their salary	D	EditEmployee
Displays status of whether order is ready or not	K	OrderStatus
Places customer’s order and sends to chefs	D	Order
Chef updates order status as it is being made	D	OrderStatus

Notifies actor about recent change/action that needs to be taken	D	Notification
Contains information regarding the total cost of a customer's bill	K	Bill
Updates customer's bill as more food is ordered	D	UpdateBill
Allows customer to pay their bill	K	Payment
Contains questions that customers must answer at the end of their visit	K	Survey
Submits maintenance or vacation request from employees to manager	D	Request
Approves or denies request	D	UpdateRequest
Displays employees' shifts for given time	K	Shift
Allows manager to change shifts for all employees	D	EditShift

5.1.2 Association Definitions

Concept Pair	**Association Description**	**Association Name**
Change Table <-> Table Status	Change Table passes edits to table status	Edit Tables
Modify Menu <-> Menu	Modify Menu passes edits to Menu	Edit Menu
Edit Inventory <-> Inventory	Edit Inventory passes edits to Inventory	Edit Inventory
Report Time <-> Payroll	Report Time updates Payroll with hours	Edit Hours
Edit Budget <-> Budget	Edit Budget sends changes to Budget	Edit Budget
Revenue <-> Edit Budget	Changes in Revenue sent to Edit Budget	New Revenue
Edit Employee <-> Payroll	Changes in Employee list sent to Payroll	Edit Payroll
System Interface <-> Order	System Interface sends customer order to Order	Edit Order

Order Status <-> Notification	Order Status sent to notification of waiter	Notify Order
Bill <-> System Interface	Information on Bill sent to Customer System Interface	Customer Bill
Update Bill <-> Bill	Updates on Bill sent to Bill	Edit Bill
Payment <-> Revenue	Customer Interface allows payment to process and payment is sent into revenue	Customer Payment
Survey <-> System Interface	Customer Interface allows survey to be given to customer and answers are placed back into interface	Survey Answer
Notification <-> Request	Requests given to notification to Manager	Notify Request
Notification <-> Update Request	Manager approval or denial to request sent to notification to employee	Notify Answer
System Interface <-> Change Table	Employee Interface allows edit to be made to table statuses	Make Table edit
System Interface <-> Modify Menu	Employee Interface allows edit to be made to menu	Make Menu edit
System Interface <-> Edit Inventory	Employee Interface allows edit to be made to inventory levels	Make Inventory edit
System Interface <-> Edit Budget	Manager Interface allows edit to be made to budget	Make Budget edit
System Interface <-> Edit Employee	Manager Interface allows edit to be made to employee list	Make Employee edit
Shift <-> Edit Shift	Edits in shifts are sent to shift in order for employees to see their schedules	Change Shifts
System Interface <-> Edit Shift	Manager Interface allows edits to be made to employee schedules	

5.1.3 Attribute Definitions

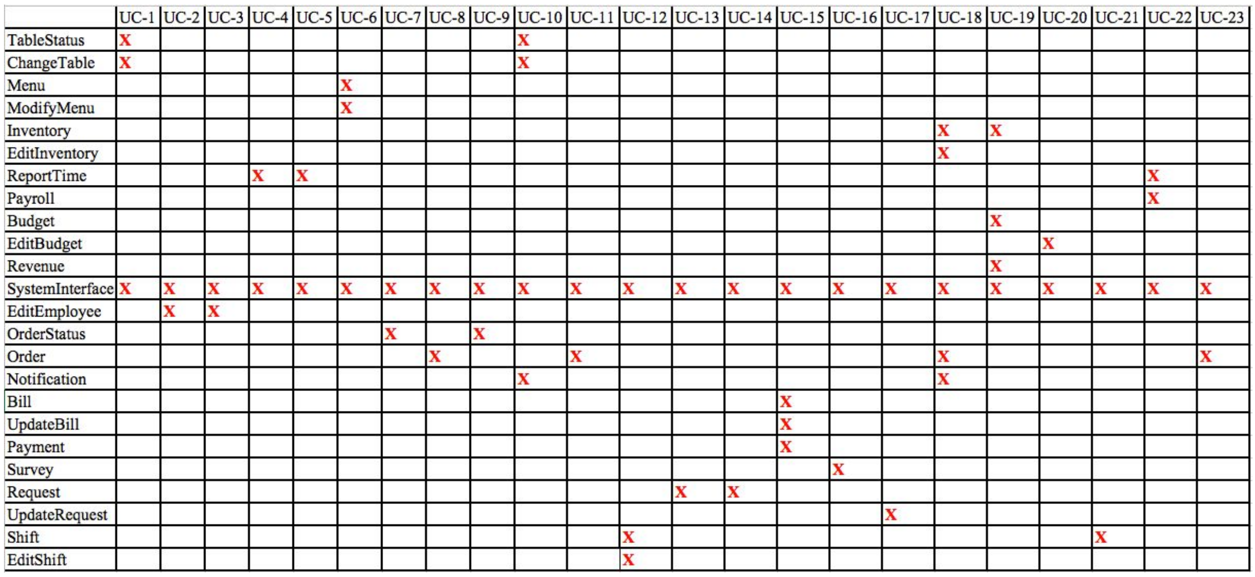
Concept	Attributes	Attribute Description
Table Status/Change Table	Cleanliness	Indication if the table has been cleaned since it was last used
	Status	Indication if a table is currently in use by a customer
	Table Number	The specific number of the table being asked about
Menu/Modify Menu	Type of Item	The dish's classification, such as appetizer, entree, dessert, or drink
	Name of Item	The name of the dish
	Description	The description paired with the item to be displayed on menu
	Price	The cost of the dish to be shown to the customers
Inventory/Edit Inventory	Ingredient List	The database of ingredients currently in stock and registered in the system
	Ingredient Amount	The current quantity of an ingredient. Unitless.
Report Time	Employee ID	The identification number of the specific employee who is logging his hours
	Arrival	Contains the time the employee logged into the system
	Departure	Contains the time the employee logged out of the system
Payroll	Employee ID	The identification number of the specific employee

	Hours	Contains total hours worked by the employee in the current week
	Pay	Contains current pay rate and calculated pay of the week to date
Budget / Edit Budget	Current Budget	Displays allocation of money towards savings, kitchen, management, and maintenance. Allows for altering of the allocations
Revenue	Income	The total amount of money earned from sales in the past week
	Expenditures	The total amount of money spent in the past week
	Profit/Loss	Displays difference in income and expenditures
System Interface	Accessor	The identity of the user accessing the system
	Choices	Depending on accessor, the user interface choices from which the user can choose to access
Edit Employee	Employee ID	The identification number of the specific employee
	Pay	Allows for the editing of the current pay rate of the employee in question
Order / Order Status	Order	The current dish that was ordered by a customer
	Table Number	The table number of the order being prepared and accessed in the system
	Status	Indicates if the order is

		finished being made
Notification	Employee ID	The identification number of the specific employee
	Change/Action	The description of the command or information to be sent to the employee
Bill / Update Bill	Order	The name and quantity of each dish ordered by the table
	Table Number	The identification number of the specific table
	Cost	The total cost of all the items in the order
Payment	Cost	The total cost of all items in the order
	Pay	Allows customer to pay the cost of the order
Survey	Questions	A set of questions for the customer to answer
	Choices	The responses a customer can give to a specific question
Request	Employee ID	The ID of the employee submitting the request
	Type	Indicates if the request is for a personal or vacation day
	Description	Contains entered information by employee of the request
Update Request	Request	Contains the request information submitted by an employee
	Approval	Gives the Manager a choice in accepting or denying a request from the employee

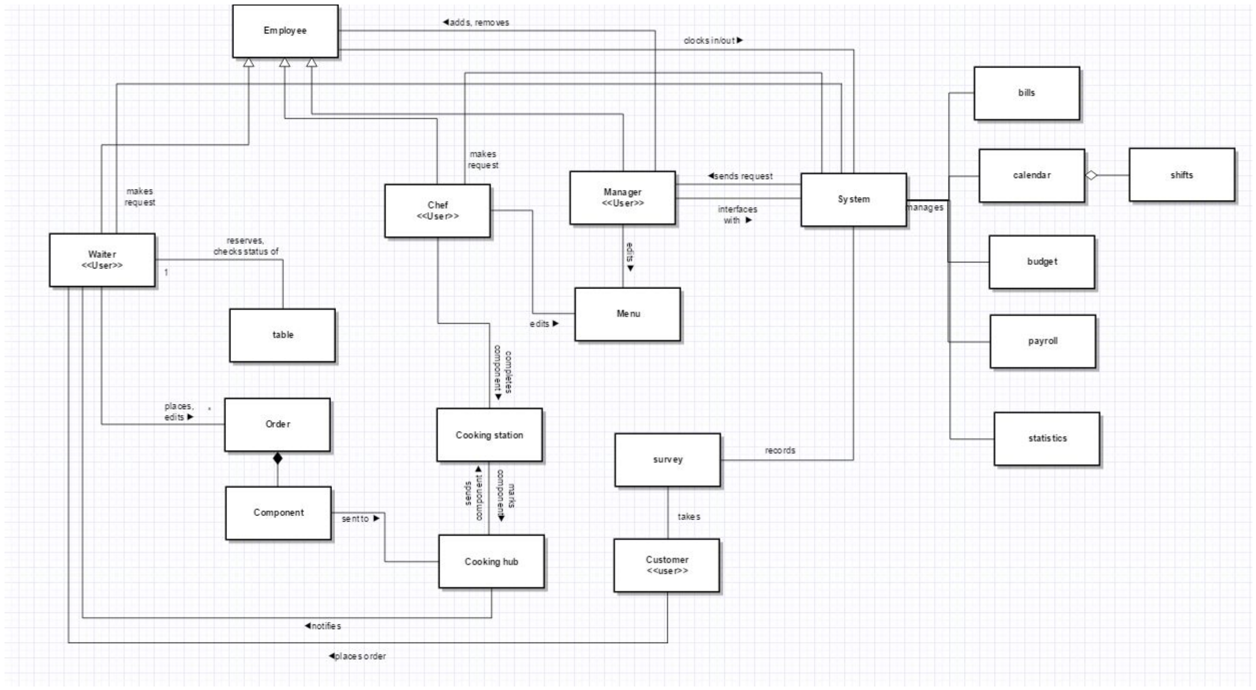
Shift / Edit Shift	Employee ID	The specific ID of employee whose schedule is being tracked
	Schedule	Given days of the week and corresponding times that the employee will be working

5.1.4 Traceability Matrix



	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17	UC-18	UC-19	UC-20	UC-21	UC-22	UC-23
TableStatus	X									X													
ChangeTable	X									X													
Menu						X																	
ModifyMenu						X																	
Inventory																		X	X				
EditInventory																		X					
ReportTime				X	X																	X	
Payroll																						X	
Budget																			X				
EditBudget																				X			
Revenue																			X				
SystemInterface	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
EditEmployee		X	X																				
OrderStatus							X		X														
Order								X			X							X					X
Notification										X								X					
Bill															X								
UpdateBill															X								
Payment															X								
Survey																X							
Request													X	X									
UpdateRequest																	X						
Shift												X									X		
EditShift												X											

5.1.5 Domain Model Diagram



5.2 System Operation Contracts

5.2.1 UC-6 (Edit Menu)

Name:	Edit Menu
Responsibilities:	Edit menu items using the application on a tablet.
Use Cases:	UC-6
Exceptions:	Manager can also perform this Use Case
Preconditions:	A chef is logged in to the system
Postconditions:	The menu is changed

5.2.2 UC-9 (Order Done)

Name:	Order Done

Responsibilities:	Notify waiters when their order is done.
Use Cases:	UC-9
Exceptions:	None
Preconditions:	An order has been placed and is marked as complete.
Postconditions:	Waiters are notified that their order is done, and the order is removed from the queue.

5.2.3 UC-19 (View Statistics)

Name:	View Statistics
Responsibilities:	Display desired statistical information to the manager.
Use Cases:	UC-19
Exceptions:	None
Preconditions:	Manager is logged into the system.
Postconditions:	Manager is shown the statistics that he/she requested.

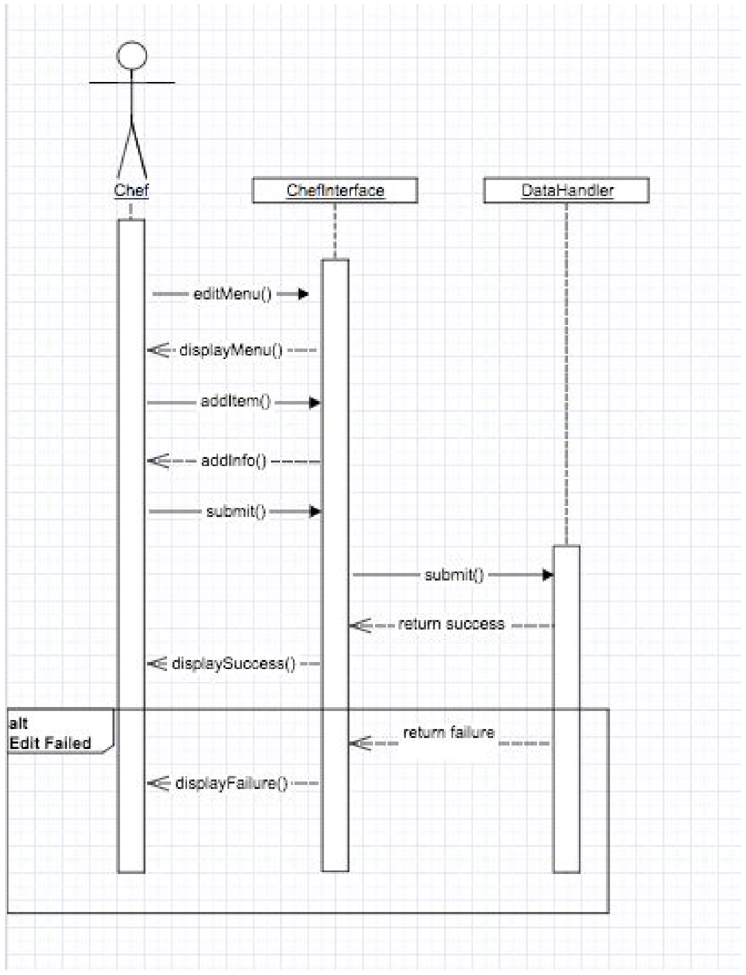
5.2.4 UC-22 (Payroll Status)

Name:	Payroll Status
Responsibilities:	Display information to the manager regarding each employee's total work hours this week and their corresponding pay.
Use Cases:	UC-22
Exceptions:	None
Preconditions:	Manager is logged into the system.
Postconditions:	Manager has a list of employees working hours for the week and the calculated pay for

	the week.

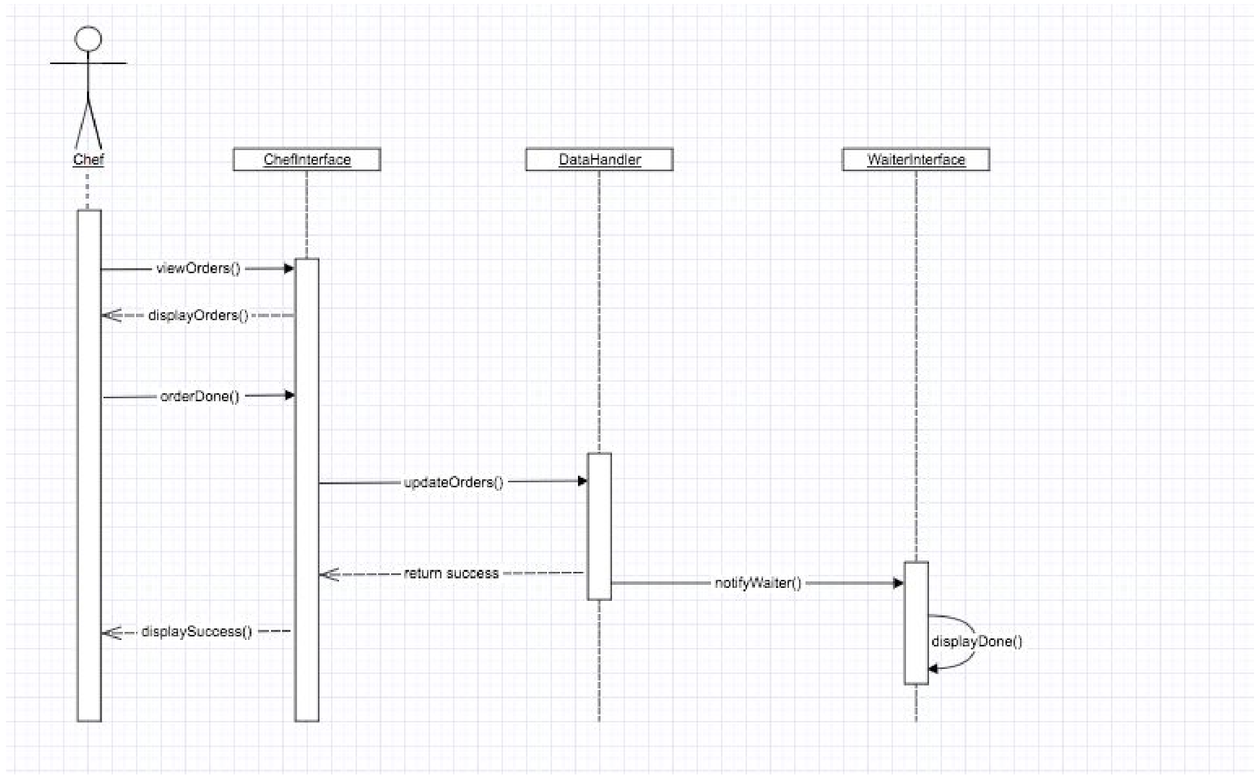
6. Interaction Diagrams

6.1 UC-6 (Edit Menu)



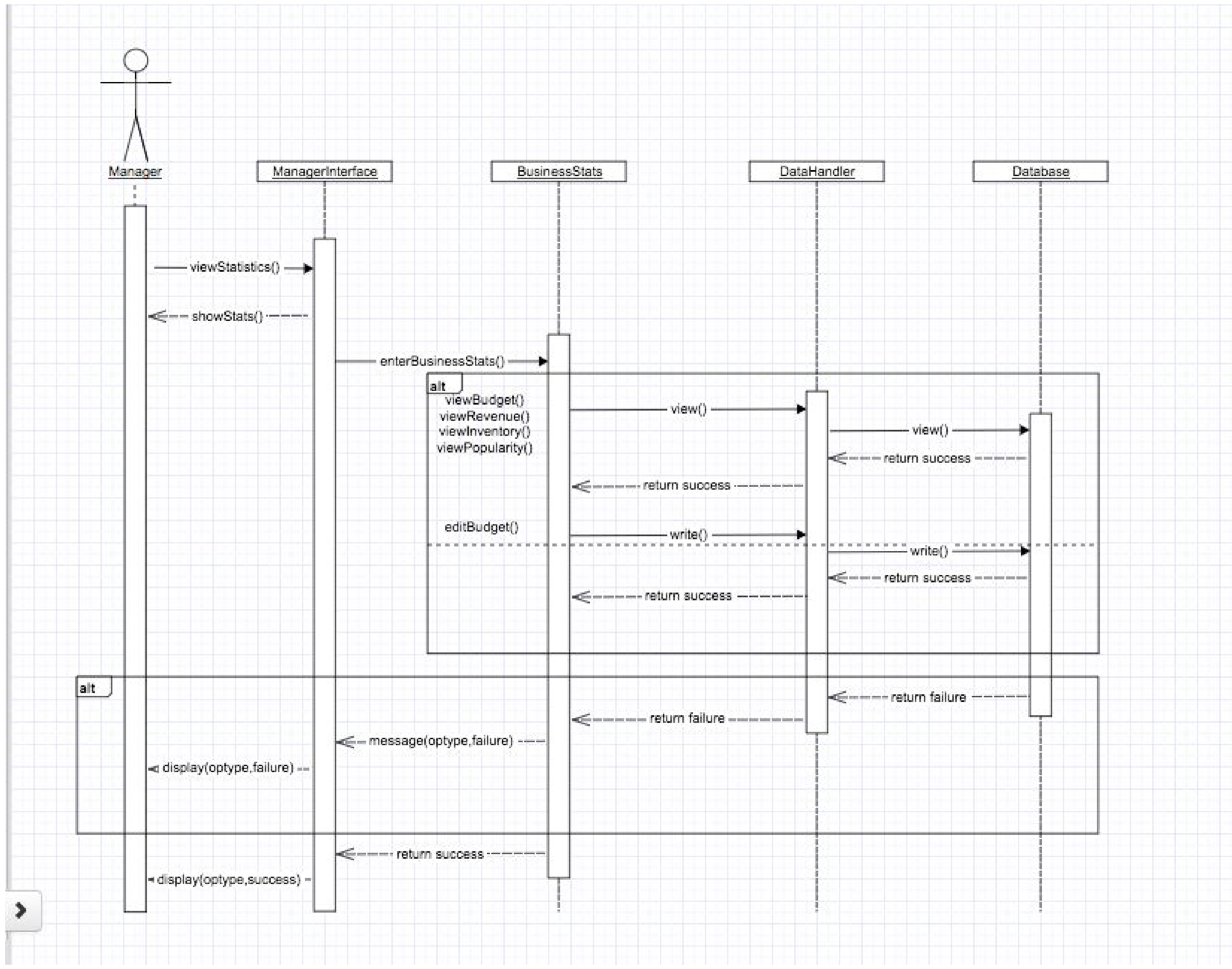
The chef can log into his UI form on his computer, and then select the edit menu option via a button click. He will then be displayed options to edit the menu (add, remove, edit items). He can add items by selecting add item, he will then be required to enter the new item name, price, and ingredients. The chef then clicks submit, the menu database gets updated and the waiter is notified on his screen if the change was successful, if the change was not successful then the waiter is notified on his screen that the edit failed. In this diagram, I used the Decorator pattern because the client (chef) initiates the call or request (submit()), and this causes a uniform calling of submit(). This improves the design because when a new function or service is added, client codes does not need to to be changed.

6.2 UC-9 (Order Done)



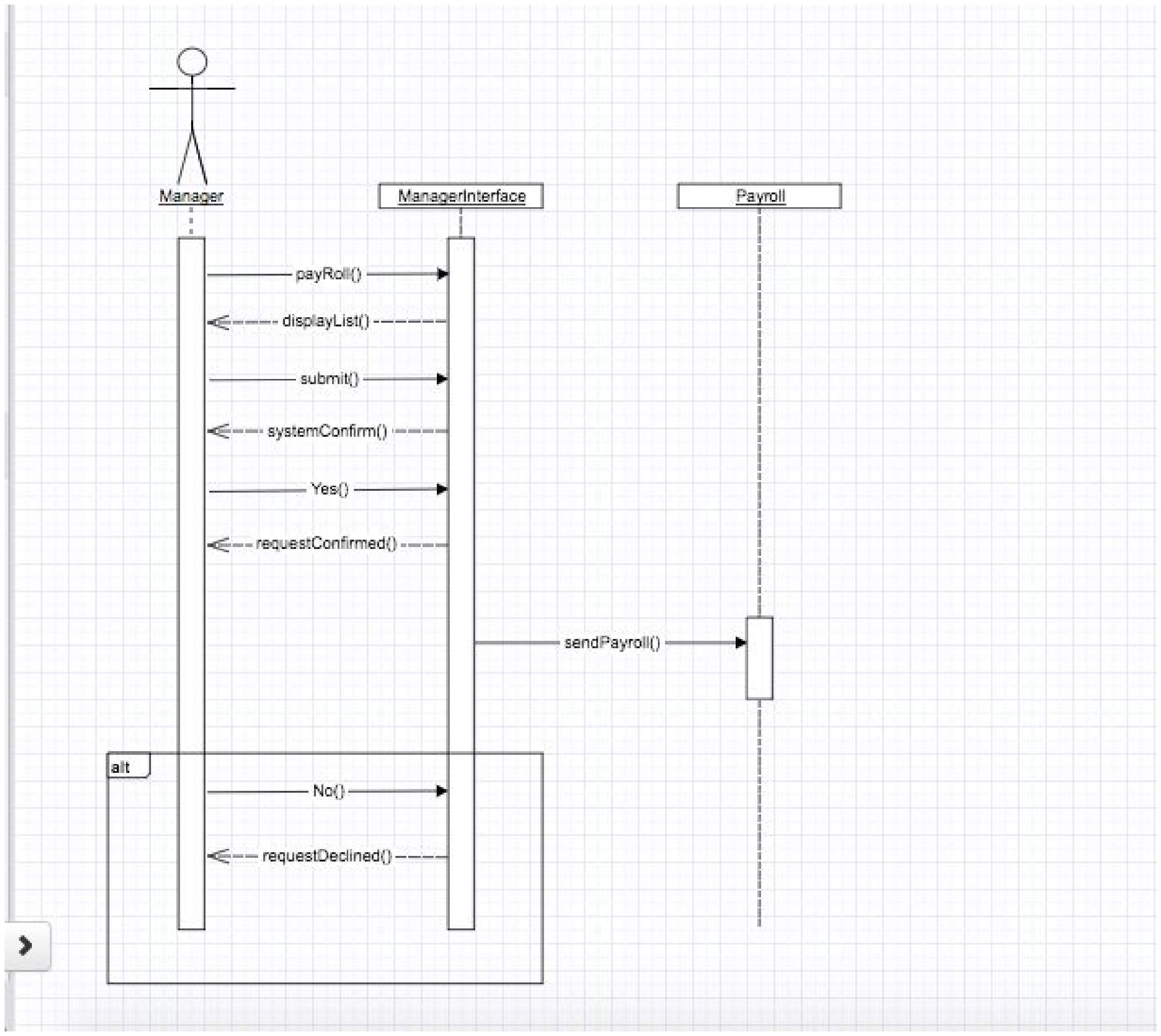
Once the chef is logged into his page on the computer, on his user interface, the chef can see the current queue of orders by clicking the “view orders” button. The list of orders will then be displayed to the chef in the order they were sent in. The chef and then select a order by clicking it and then clicking the “order done” button. If this is successful the list of orders gets updated, the chef is notified on his screen., and the waiter is then notified on his/her screen that an order has been completed.

6.3 UC-19 (View Statistics)



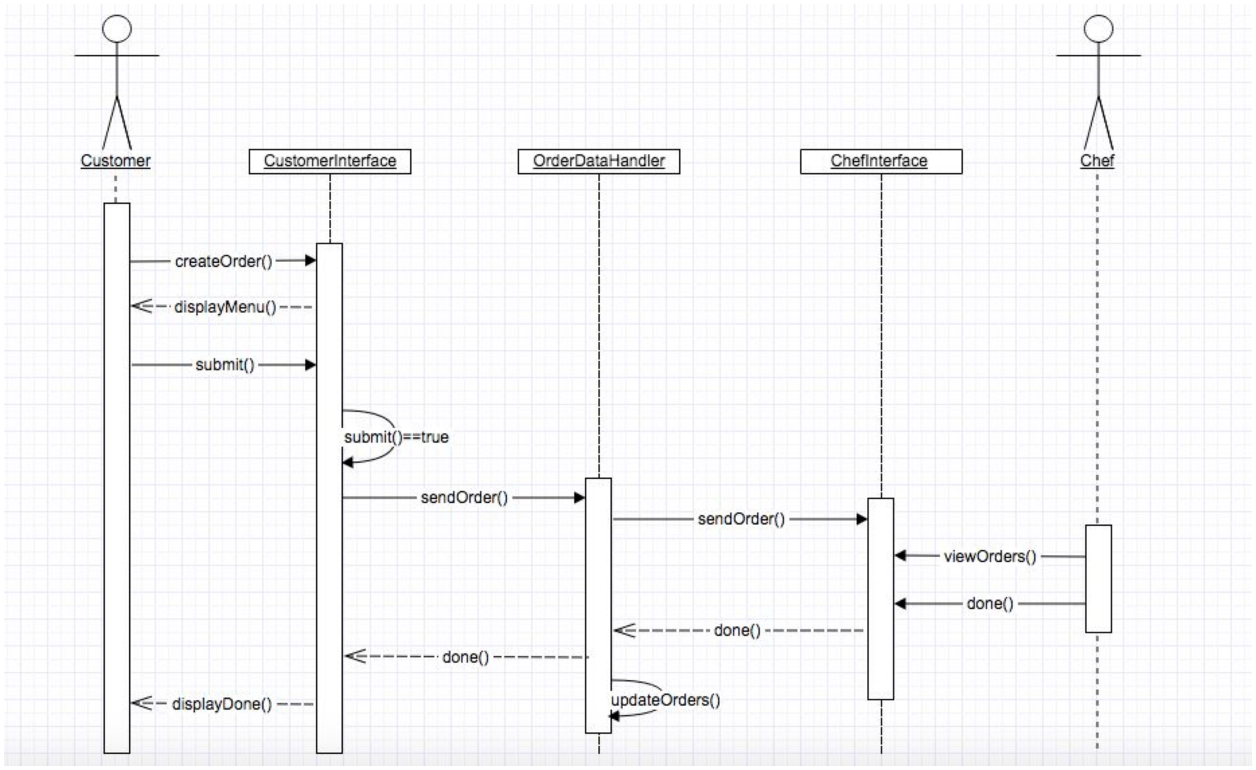
The manager is logged into his UI and he/she clicks the "view statistics" button. He/she is then shown a list of statistical categories he/she can chose (budget, revenue, inventory, popularity). The manager selects one of the choices and is shown the requested information pulled from the database, with a success message. If for some reason the request fails, due to possible not pulling information from the database, the manager will be shown a failure message. Alternatively, the manager can also edit the budget when accessing the budget statistic.

6.4 UC-22 (Payroll Status)



The manager is logged into his UI, he/she chooses the employee option, and is brought to the payroll page. The list of employees is shown with their current hours worked, their hourly rate, and their pay for the week. The manager submits the payroll which prompts the system to ask for confirmation. If the manager clicks “yes” then the payroll is sent to whoever handles and distributes the payroll. If the manager clicks “no” the payroll does not get sent.

6.5 UC-23 (Customer Order)



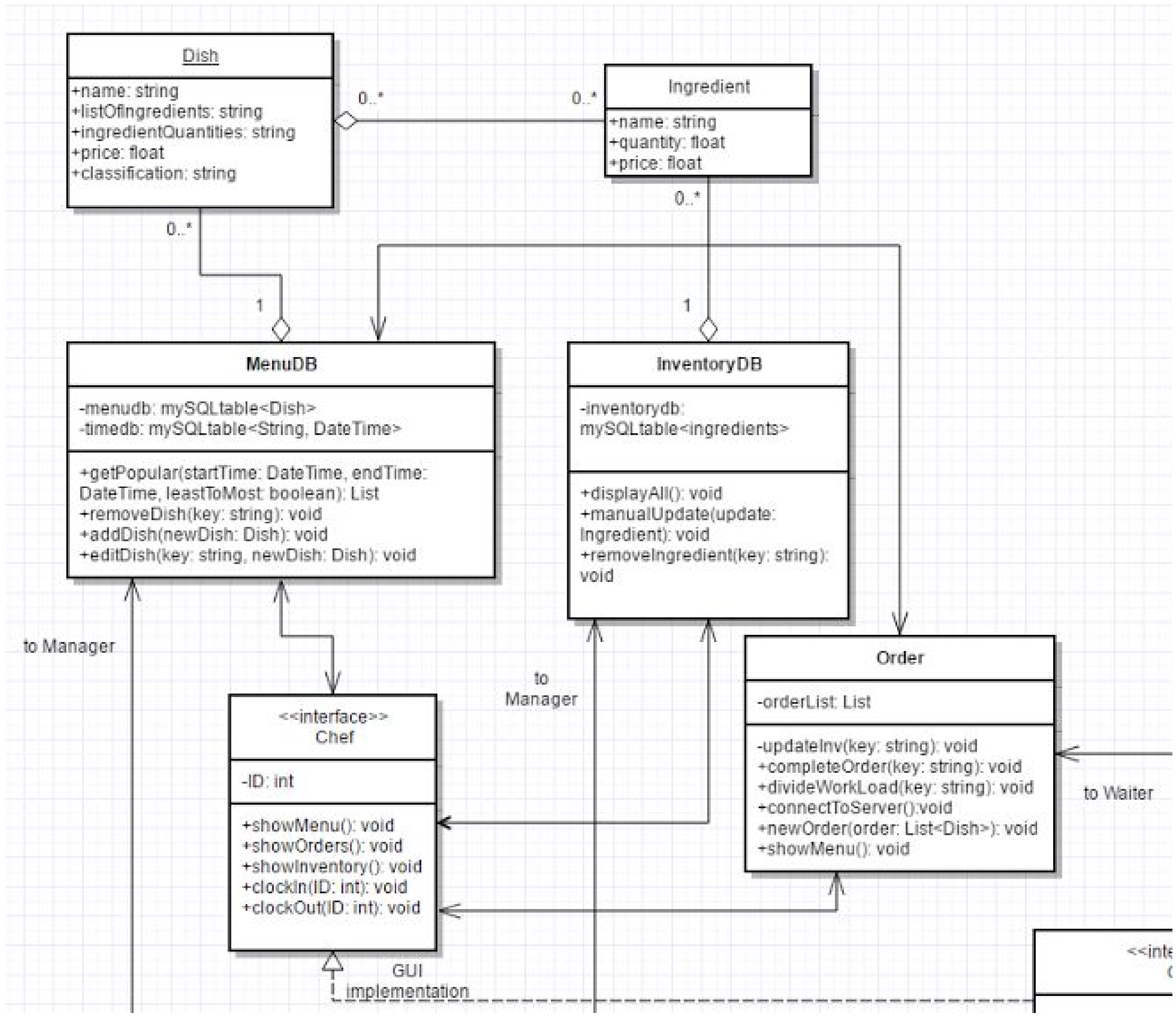
The customer can use the provided computer on their table to order additional items from the menu if desired. The customer clicks the "create order" button. They will be shown a menu, from there they can select what items they want. Once that is done they click the "submit" button, if that button is clicked it gets sent to the server to process, the server then sends the updated order list to the chef interface page. When the Chef clicks the "view orders" button, they will see the new updated order list, once the chef is done with that specific order they select it from list and click the "done" button. This sends a response to the server the server sends to the customer interface page that it was completed, and a message will be displayed to the customer on their computer screen. Also the server will update the order list to remove the completed order from the list. In this diagram I also used the Decorator design pattern, because the client customer interface calls the head of "linked list", and produces uniform calling of the "sendOrder" function. The advantage to this is adding a new function, does not require us to change the client code. We only need to implement the new function and insert it into the "linked list".

7. Class Diagram and Interface Specification

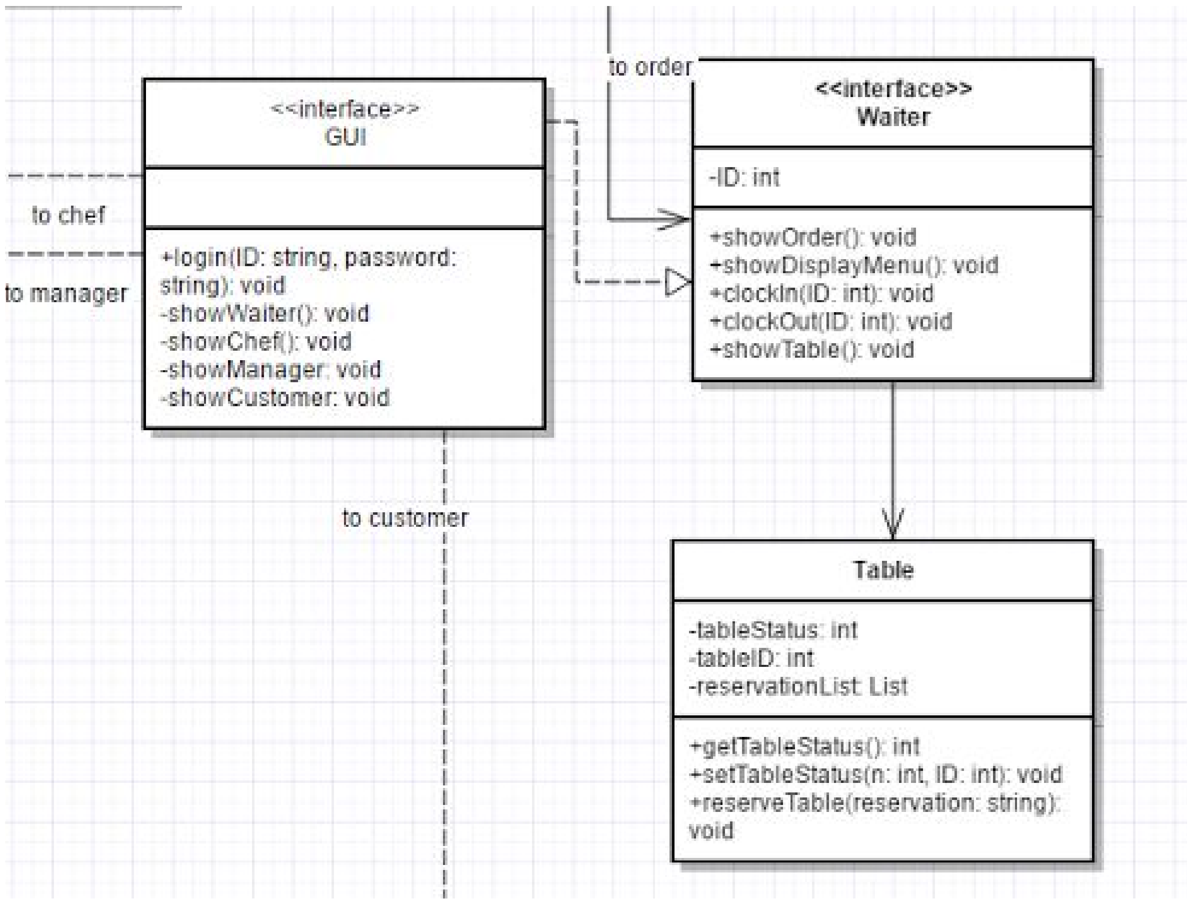
7.1 Class Diagram

The class diagram is displayed in three separate parts for clarity:

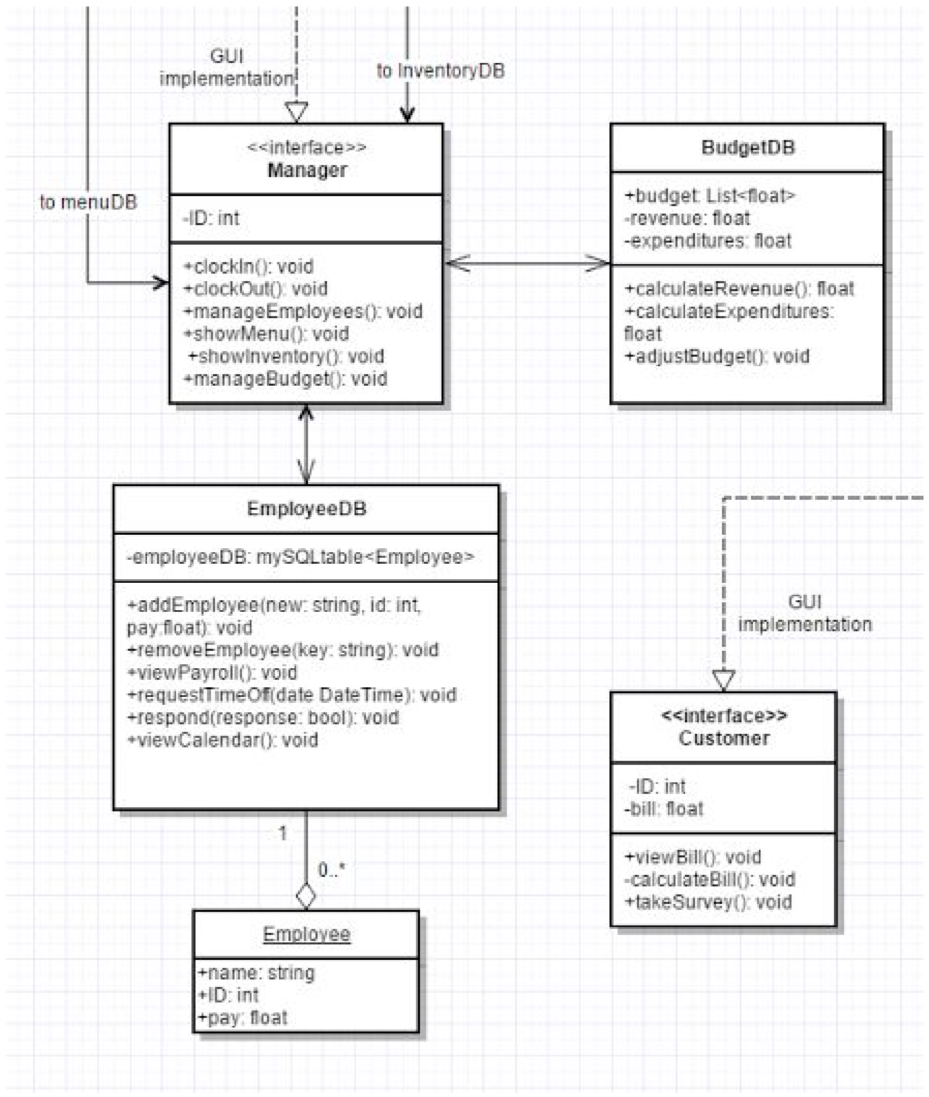
7.1.1 Class Diagram - Kitchen Portion


Dish
+name: string
+listOfIngredients: string
+ingredientQuantities: string
+price: float
+classification: string
0..*
0..*
Ingredient
+name: string
+quantity: float
+price: float
0..*
0..*
1
1
MenuDB
-menudb: mySQLtable<Dish>
-timedb: mySQLtable<String, DateTime>
+getPopular(startTime: DateTime, endTime: DateTime, leastToMost: boolean): List
+removeDish(key: string): void
+addDish(newDish: Dish): void
+editDish(key: string, newDish: Dish): void
InventoryDB
-inventorydb: mySQLtable<ingredients>
+displayAll(): void
+manualUpdate(update: Ingredient): void
+removeIngredient(key: string): void
to Manager
to Manager
Order
-orderList: List
-updateInv(key: string): void
+completeOrder(key: string): void
+divideWorkLoad(key: string): void
+connectToServer():void
+newOrder(order: List<Dish>): void
+showMenu(): void
to Waiter
<<interface>> Chef
-ID: int
+showMenu(): void
+showOrders(): void
+showInventory(): void
+clockIn(ID: int): void
+clockOut(ID: int): void
GUI implementation

7.1.2 Class Diagram - Waiter and General GUI portion



7.1.3 Class Diagram - Management and Customer Portion



7.2 Data Types and Operation Signatures

7.2.1 Dish Object Type

The Dish object is the baseline object for what a customer will order, and what is stored in the menu database.

Attributes:

+name: string - A string that has the name of the dish

+listOfIngredients : string - A list of Ingredients that are in the dish, separated by commas so it can be later serialized by Order's updateInv() function.
+ingredientQuantities: string - A list of quantities for each ingredient, separated by commas so it can be later serialized by Order's updateInv() function.
+price: float - The price, in dollars of the dish.
+classification: string - Indicates what the meal is - for example: entree, appetizer, etc for menu organization.

7.2.2 Ingredient Object Type

Each dish contains ingredients, and the store's inventory contains a stock of ingredients.
Attributes:
+name: string - A string that has the name of the ingredient.
+quantity: float - The amount that the restaurant has in stock
+price: float - The cost of the ingredient

7.2.3 Menu Database

This class keeps track of each dish on the menu, and also is able to determine the most popular item on it.
Attributes:
-menudb: mySQLtable<Dish> - Online data storage that contains all of the dishes in the restaurant, and can be modified by the user.
-timedb: mySQLtalbe <String, DateTime> - Table that is automatically updated each time and order is completed. Stores the name of a dish and the time that it was ordered, and used to track popularity.
Methods:
+getPopular(startTime: DateTime, endTime: DateTime, leastToMost: boolean): List - Used to determine which dishes are the most popular within a given timeframe. Can also return the list in reverse order if leastToMost is true.

+removeDish(key: string): void - Allows the user to remove a dish from the menu.

+addDish(newDish: Dish): void - Allows the user to add a dish to the menu.

+editDish(key: string, newDish: Dish): void - Allows the user to edit a dish already on the menu with new information.

7.2.4 Order

This class allows the user to place an order, and the chef to mark it as complete when finished.
Attributes:
-orderList: List - A copy of the list of the current orders, which is updated and maintained by the server.
Methods:

-updateInv(key: string):void - Updates the inventory when the order is completed. First serializes the ingredients of the item that was submitted, and then updates the ingredient database.
+completeOrder(key:string): void - Allows the chef to complete a dish. Sends a message to the server that the order was completed, and calls updateInv().
+divideWorkLoad(key:string): void - Serializes the order and distributes it to each workstation depending on the dish.
+connectToServer():void - Creates a server connection to the other devices in the restaurant that can order. The server contains all of the waiting orders.
+newOrder(order: List<Dish>): void - Pushes an order to the server with a list of the dishes that were ordered.
+showMenu(): void - Displays the menu.

7.2.5 Inventory Database

This database keeps track of the restaurant's inventory, which can update automatically or manually.
Attributes:
-inventorydb: mySQLtable<ingredients> - A table of ingredients that the restaurant keeps in stock.
Methods:
+displayAll(): - Displays the contents of inventorydb.
+manualUpdate(update: Ingredient): void - Allows the user to update the inventory manually, subverting Order's updateInv().
+removeIngredient(key: string): void - Removes the specified ingredient from the database.

7.2.6 Chef Interface

This interface will give the chefs in the restaurant access to vital functions for kitchen and order management.
Attributes:
-ID: int - The identification number of the chef who is currently logged in.
Methods:
+showMenu(): void - Displays the user interface for the MenuDB
+showOrders(): void - Displays the user interface for the Orders
+showInventory(): void - Displays the user interface for the InventoryDB
+clockIn(ID: int): void - Allows the chef to clock into his shift
+clockOut(ID: int): void - Allows the chef to clock out of his shift

7.2.7 GUI Interface

This interface will serve as the general GUI on the main screen of the application, and will direct users to the proper screen, based on entered login information.
Methods

+login(ID: string, password: string): void - Allows the user to enter his credentials which will call one of the four following functions, depending on what the user inputs. If the username and password are incorrect, the user will remain on this screen.
-showWaiter(): void - Displays the user interface for the Waiters
-showChef(): void - Displays the user interface for the Chefs
-showManager: void - Displays the user interface for the Managers
-showCustomer: void - Displays the user interface for the Customers

7.2.8 Waiter Interface

This interface will give the waiters in the restaurant access to vital functions for ordering and table management.

Attributes

-ID: int - The identification number of the currently clocked in employee

Methods

+showOrder(): void - Displays the user interface for Ordering.

+showDisplayMenu(): void - Displays the menu for ordering

+clockIn(ID: int): void - Allows the waiter to clock in.

+clockOut(ID: int): void - Allows the waiter to clock out.

+showTable(): void - Displays the table interface.

7.2.9 Table

This class has information regarding each table in the restaurant.

Attributes

-tableStatus: int - Enumerated variable that indicates if the table is vacant, occupied, dirty, or out of service.
-tableID: int - Identification for a table.
-reservationList: List- List of times where the table will be reserved by someone.

Methods

+getTableStatus(): int - Returns tableStatus.
+setTableStatus(n: int): void - Sets tableStatus.
+reserveTable(reservation: string): void - Adds a reservation to the list.

7.2.10 Manager Interface

This interface will give the managers in the restaurant access to vital functions for general restaurant management.

Attributes

-ID: int - Identification number of the currently clocked in manager.

Methods

clockIn(ID: int): void - Allows the waiter to clock in.

+clockOut(ID: int): void - Allows the waiter to clock out.

+manageEmployees(): void - Displays the user interface for the Employee Database
+showMenu(): void - Displays the user interface for the menu.
+showInventory(): void - Displays the user interface for the inventory
+manageBudget(): void - Displays the user interface for the Budget Database

7.2.11 Employee Object Type

This object contains employee information
Attributes
+name: string - The name of the employee
+ID: int- An identification number for the employee.
+pay: float - The per hour pay rate of the employee

7.2.12 Employee Database

This class holds information about every employee, the total payroll, and other information.
Attributes
-employeeDB: mySQLtable<Employee> - A table of each employee who is working at the restaurant
Methods
+addEmployee(new: string, id: int, pay:float): void - Adds a new employee to the existing list.
+removeEmployee(key: string): void - Searches for and removes an employee from the list, if found.
+viewPayroll(): void - Displays the payroll data on the user interface.
+requestTimeOff(date DateTime): void - Allows the user to make a vacation request.
+respond(response: bool): void - Allow the manager to respond to the vacation request.
+viewCalendar(): void - Displays a calendar

7.2.13 Budget Database

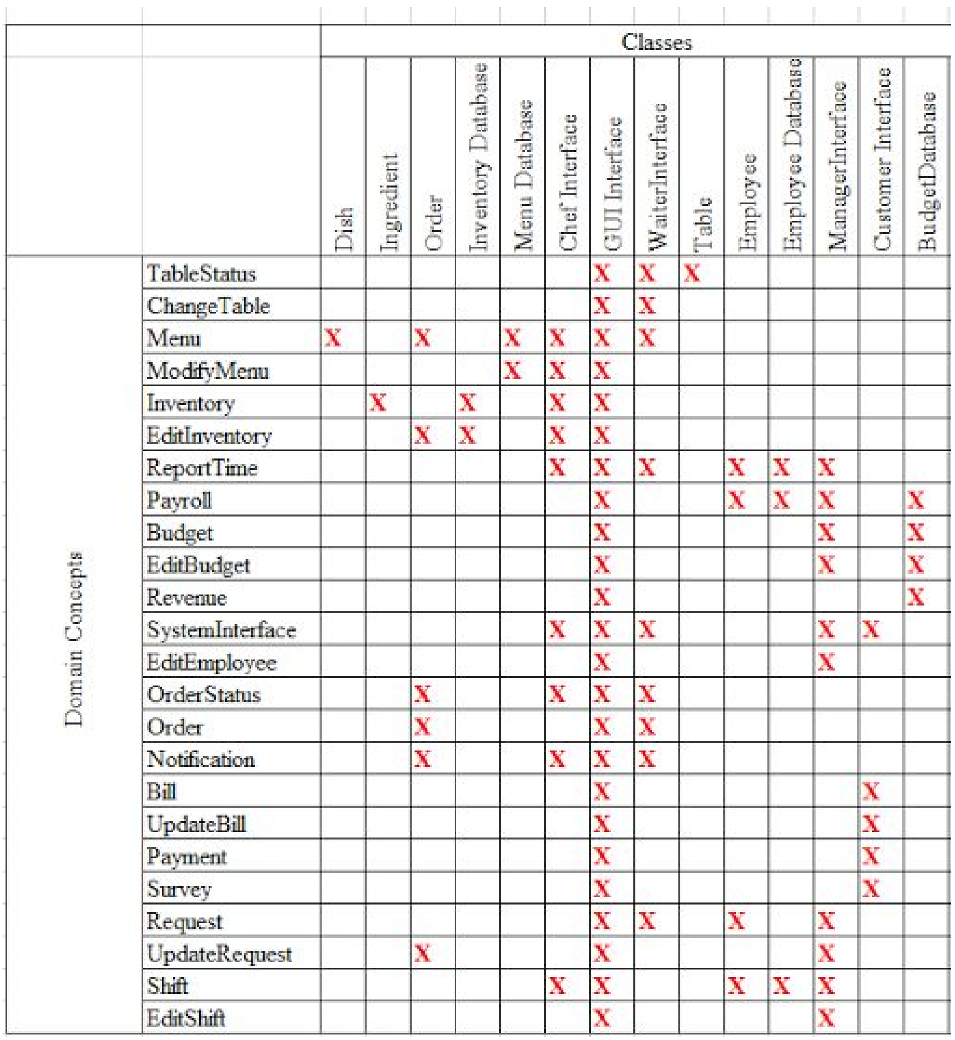
This database holds information about each component of the budget, and can make show adjustments in real time.
Attributes
+budget: List<float> - List of quantities of each section of the budget
-revenue: float - Running total for the month based on the amount of orders that have been processed
-expenditures: float - Running total for the month based on how many ingredients have been ordered, and needs to be manually adjusted.
Methods
+calculateRevenue(): float - Calculates revenue based on the timedb.
+calculateExpenditures(): float - Calculates revenue based on how many ingredients have

been ordered
+adjustBudget(): void - Allows the user to reallocate funds

7.2.14 Customer Interface

This interface is simple, but will allow the user to perform a few basic functions
Attributes
-ID:int - Customer identification number.
-bill: float - Running total of the bill
Methods
+viewBill(): void- Allows the customer to view what he has ordered and the price.
-calculateBill(): void - Called when the user wants to view the bill, and recalculates if any new orders have been submitted
+takeSurvey(): void - Allows the user to take a survey of the quality, etc. of the restaurant.

7.3 Traceability Matrix



	Classes													
Domain Concepts	Dish	Ingredient	Order	Inventory Database	Menu Database	Chef Interface	GUI Interface	WaiterInterface	Table	Employee	Employee Database	ManagerInterface	Customer Interface	BudgetDatabase
TableStatus							X	X	X					
ChangeTable							X	X						
Menu	X		X		X	X	X	X						
ModifyMenu					X	X	X							
Inventory		X		X		X	X							
EditInventory			X	X		X	X							
ReportTime						X	X	X		X	X	X		
Payroll							X			X	X	X		X
Budget							X					X		X
EditBudget							X					X		X
Revenue							X							X
SystemInterface						X	X	X				X	X	
EditEmployee							X					X		
OrderStatus			X			X	X	X						
Order			X				X	X						
Notification			X			X	X	X						
Bill							X						X	
UpdateBill							X						X	
Payment							X						X	
Survey							X						X	
Request							X	X		X		X		
UpdateRequest			X				X					X		
Shift						X	X			X	X	X		
EditShift							X					X		

A majority of the domain concepts were lumped together into logical groups to create classes, and further show which users would use which domain, e.g. a generalized Menu Database was derived from grouping all the menu concepts together. Because the GUI handles all requests and changes, it was derived from all the domain concepts. The remaining classes were derived from the concepts using common sense and generalizations; it made the most sense to have domain concepts realized in classes that would be most convenient for the user.

7.4 Design Patterns

Our system uses the Decorator pattern to handle requests made by the user on the interface. With the Decorator pattern, we are able to design and implement new features and easily incorporate them into our system. It also allows easy editing to fix any minor parts of a code without hindering the entire system. An example would be future implementation of a nutritional facts database for the customers to use. This feature would be designed and implemented to work as needed and then it would be added to our system by simply creating a corresponding button on the interface which would call the new feature. Editing any existing features will follow the same procedure and will require no major change to any other files or code.

7.5 Object Constraint Language (OCL) Contracts

7.5.1 Remove Dish

context MenuDB **inv**:
    self.menudb > 0
    --Menu must not be empty when removing a dish
context MenuDB::removeDish(key: string): void
    pre: key == self.Dish.name
    --User input (key) exists in menu
    --Dish removed from menu
    post: self.menudb = self.menudb - 1
    --Removing dish causes menu to have one less item than before

7.5.2 Add Dish

context MenuDB::addDish(newDish: Dish): void
    pre: newDish != self.menudb
    --New dish is not found in menu already
    post: self.menudb = self.menudb + 1
    --Adding dish to menu increases menudb quantity by one

7.5.3 Add Employee

context EmployeeDB::addEmployee(new: string, id: int, pay: float): void

pre: new != self.Employee.name
  id != self.Employee.ID

--Name or ID does not already exist

--Adds employee to database

post: self.employeeDB = self.employeeDB + 1

--Employee database increases by one after adding employee

7.5.4 Remove Employee

context EmployeeDB **inv**:

self.employeDB > 0

--Employee database must not be empty before executing function

context EmployeeDB::removeEmployee(key: string): void

pre: key == self.Employee.name

--User input matches existing employee

--Removes employee from database

post: self.employeeDB = self.employeeDB - 1

--Employee database has one less row after removing employee

7.5.5 Calculate Revenue

context BudgetDB::calculateRevenue(): float

pre: self.revenue != NULL

--Value exists for revenue

post: result = self.revenue

7.5.6 Clock In

context ManagerInterface **inv**

self.manager == true

--Manager is one clocking in

context ManagerInterface::clockIn(id: int): void

pre: id == self.ID
  self.lastPunch != "IN"

--Entered id matches ID in database

--Last recorded punch in database is not IN

--Manager is clocked in

post: self.lastPunch = “IN”

--Changes last recorded punch to IN to show successful clock in

7.5.7 Clock Out

context ManagerInterface **inv**

self.manager == true

--Manager is one clocking out

context ManagerInterface::clockOut(id: int): void

pre: id == self.ID

self.lastPunch != “OUT”

--Entered id matches ID in database

--Last recorded punch in database is not OUT

--Manager is clocked in

post: self.lastPunch = “OUT”

--Changes last recorded punch to IN to show successful clock out

7.5.8 Get Table Status

context Table **inv**

self.WaiterInterface == true

--Waiter must be accessing table status from their interface

context Table::getTableStatus(): int

pre: self.tableID != NULL

--Table ID must exist

--Retreives table status

post: result = self.tableStatus

--Returns the int that correlates to the table status

7.5.9 New Order

context Order **inv**

self.WaiterInterface == true

--Waiter must be ordering from their interface

context Order::newOrder(order: List<Dish>): void

pre: order != NULL

--Check to make sure order exists in menu database

--Sends order

post: self.OrderList = self.OrderList + 1

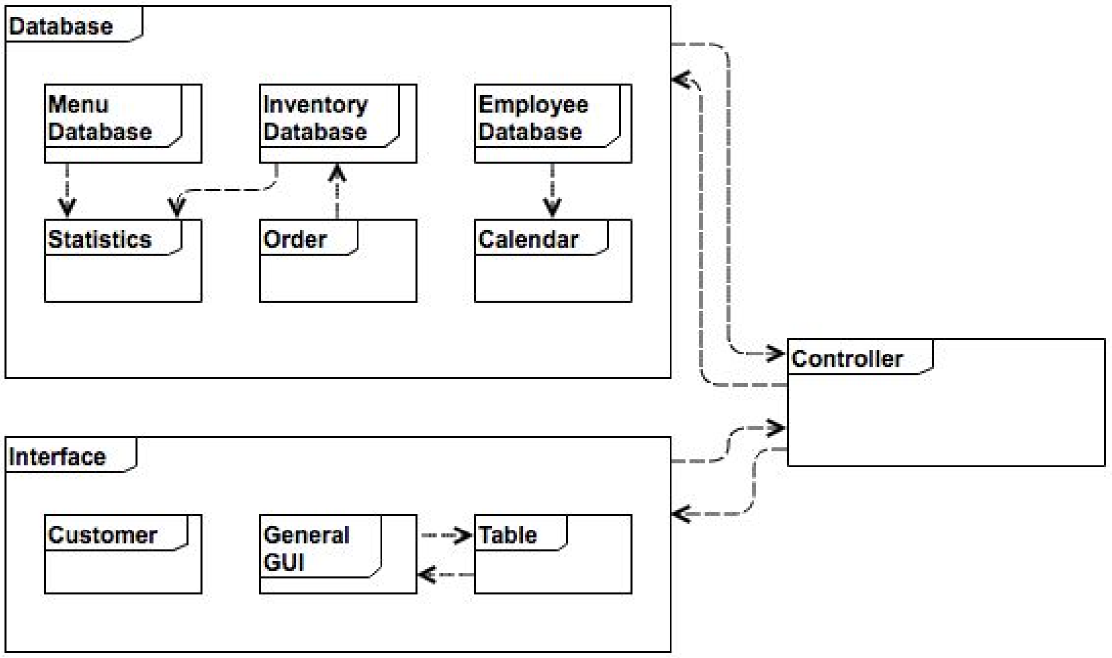
--List increases by one to show successful order

8. System Architecture and Design

8.1 Architectural Styles

Our project utilizes a number of architectural styles in its design. The two main styles that we implemented were the Client/Server style and the Object-Oriented Style. The Client/Server style is basically a separation of the system into two interconnecting parts. In our system, it will feature a desktop user interface which would communicate to a database containing all the relevant information about the restaurant's kitchen, management, and customer service. This is also known as a 2-Tier architecture style. The system will grant access to certain data based on user identification, therefore increasing the security of the system. This system also boasts a simplicity factor based on the fact that all the relevant data is stored on one central server. The other main style we utilized is the Object-Oriented style. This method is all about individual objects being independent from other objects, making the system more comprehensive and easy to understand. In our system, these independent objects will be all the different options on the main menu of the system. By separating the kitchen tasks from the managerial, it improves simplicity and efficiency for both the user and the system in general. These styles work together to make up our system architectural style.

8.2 Identifying Subsystems



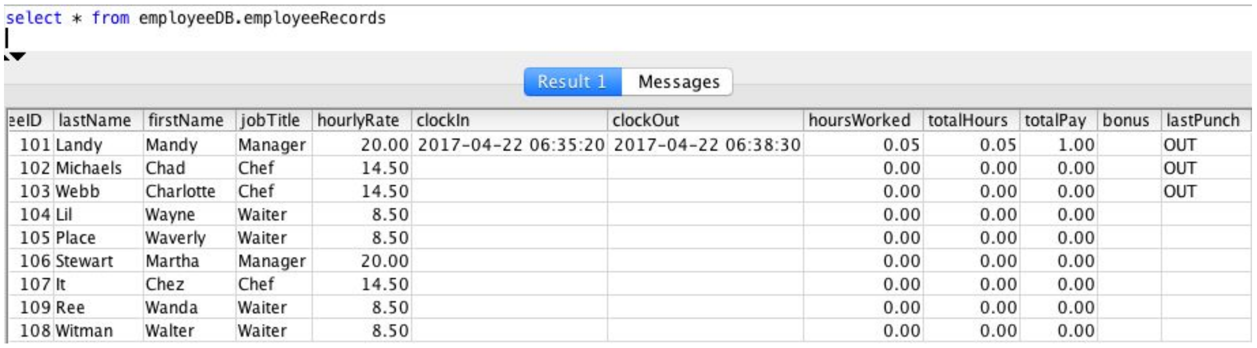
Based on the class diagram, the system will have three packages: Interface, Database, and Controller. The Interface package is comprised of the Customer and the General GUI Interfaces where both Interfaces can access/input different options based on the user. The Database contains all the information needed to run the restaurant, e.g. the menu and employee information, and also information regarding dishes, ingredients, and any business statistics. The last package is the Controller which acts as the mediator between the Database and the Interface. The controller is in charge of handling all requests made from the interfaces and transfers any information between the Database and Interface.

8.3 Mapping Systems to Hardware

Our system will need to be run on multiple computers as the system is based around the idea of each table of customers having their own system, the kitchen having one, the waiters having one, and the manager having his/her own. These would be identifiers of the subsystems. The main server will be used by the manager on a computer system, with granted access to the entire system. The customer subsystem would be in place on the tablet given to each table and then would be given access to the menu, bill, and survey. The waiter subsystem would be on the tablets used by the waiters and given access to the menu and bill also, as well as ability to send and track orders with the kitchen. Lastly, the kitchen subsystem would be on a computer system within the kitchen with access to orders, inventory, menu, and a communication link to the waiter subsystem to notify waiters of completed orders.

8.4 Persistent Data Storage

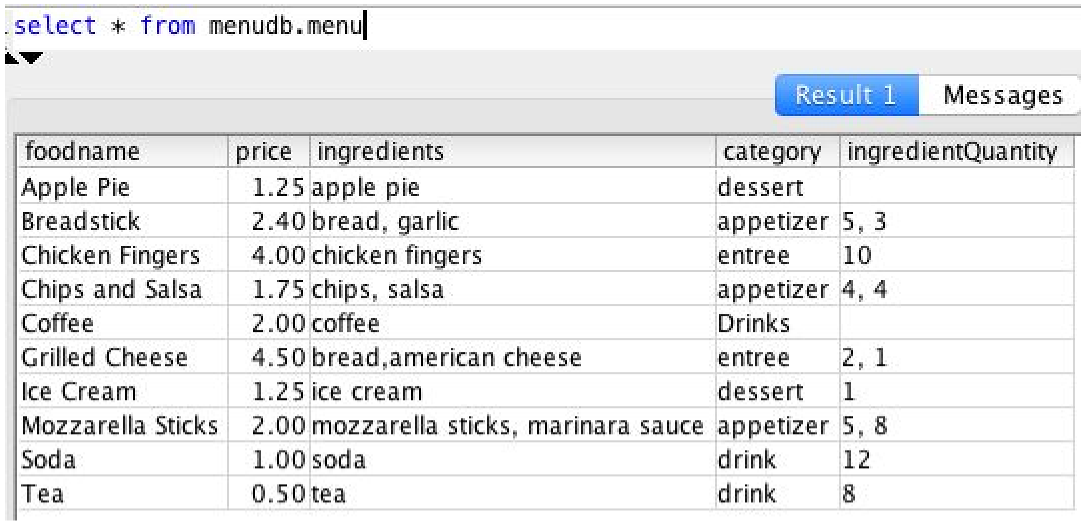
For this system, our data will contain information on the menu, inventory, and employee records, so they need to be preserved for future use and access. We decided to approach this problem by using SQL databases and tables. Separate databases will be made for separate sections, so a menu database for menu-related items for example. The employee database will contain the employee's ID, name, job title, hourly wage, a punch-in, punch-out "system", hours worked, and pay. The menu database will contain the name of the food item, its price, ingredients, and course type, e.g. dessert, entree, etc. The inventory database will contain an ingredient name, quantity, and cost. The time database will contain the dish that was ordered at a given timestamp, for use in popularity analysis. Screenshots of the databases and tables are below.



```
select * from employeeDB.employeeRecords
```

eeID	lastName	firstName	jobTitle	hourlyRate	clockIn	clockOut	hoursWorked	totalHours	totalPay	bonus	lastPunch
101	Landy	Mandy	Manager	20.00	2017-04-22 06:35:20	2017-04-22 06:38:30	0.05	0.05	1.00		OUT
102	Michaels	Chad	Chef	14.50			0.00	0.00	0.00		OUT
103	Webb	Charlotte	Chef	14.50			0.00	0.00	0.00		OUT
104	Lil	Wayne	Waiter	8.50			0.00	0.00	0.00		
105	Place	Waverly	Waiter	8.50			0.00	0.00	0.00		
106	Stewart	Martha	Manager	20.00			0.00	0.00	0.00		
107	It	Chez	Chef	14.50			0.00	0.00	0.00		
109	Ree	Wanda	Waiter	8.50			0.00	0.00	0.00		
108	Witman	Walter	Waiter	8.50			0.00	0.00	0.00		

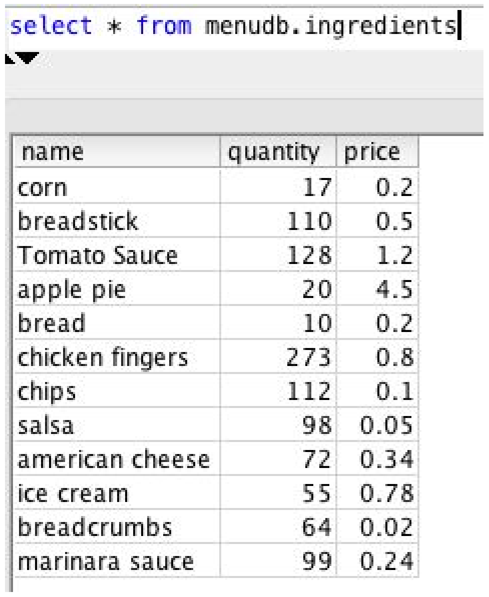
Employee Data



```
select * from menudb.menu
```

foodname	price	ingredients	category	ingredientQuantity
Apple Pie	1.25	apple pie	dessert	
Breadstick	2.40	bread, garlic	appetizer	5, 3
Chicken Fingers	4.00	chicken fingers	entree	10
Chips and Salsa	1.75	chips, salsa	appetizer	4, 4
Coffee	2.00	coffee	Drinks	
Grilled Cheese	4.50	bread,american cheese	entree	2, 1
Ice Cream	1.25	ice cream	dessert	1
Mozzarella Sticks	2.00	mozzarella sticks, marinara sauce	appetizer	5, 8
Soda	1.00	soda	drink	12
Tea	0.50	tea	drink	8

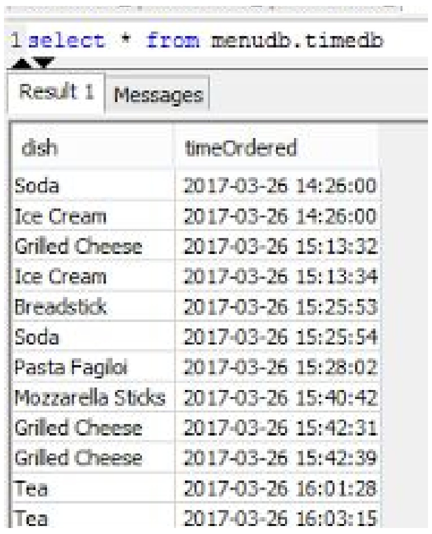
Menu Data



```
select * from menudb.ingredients
```

name	quantity	price
corn	17	0.2
breadstick	110	0.5
Tomato Sauce	128	1.2
apple pie	20	4.5
bread	10	0.2
chicken fingers	273	0.8
chips	112	0.1
salsa	98	0.05
american cheese	72	0.34
ice cream	55	0.78
breadcrumbs	64	0.02
marinara sauce	99	0.24

Inventory Data



```
select * from menudb.timedb
```

Result 1 | Messages

dish	timeOrdered
Soda	2017-03-26 14:26:00
Ice Cream	2017-03-26 14:26:00
Grilled Cheese	2017-03-26 15:13:32
Ice Cream	2017-03-26 15:13:34
Breadstick	2017-03-26 15:25:53
Soda	2017-03-26 15:25:54
Pasta Fagiloi	2017-03-26 15:28:02
Mozzarella Sticks	2017-03-26 15:40:42
Grilled Cheese	2017-03-26 15:42:31
Grilled Cheese	2017-03-26 15:42:39
Tea	2017-03-26 16:01:28
Tea	2017-03-26 16:03:15

Time Data

8.5 Network Protocol

Our system relies heavily on communication between computers and tablets. In order to accomplish this communication, we will be using MYSQL databases and Java JDBC to connect to the databases for information. Although the group members have more experience with C++ programming, Java offers a more interactive UI which is perfect for this project. MYSQL was chosen for its compatibility with various operating systems, so the case of a Mac communicating with a Windows or Android tablet will pose no problem, and for its easy implementation and access for Java where we will display the database contents on Java GUI forms. As custom with other SQLs, MYSQL also provides an online server to hold information. In case of an emergency with a tablet or computer, the data will stay intact and uncorrupted. The database will also allow certain parts of the restaurant share specific data to other parts. An example would be the menu. The chef can edit the menu with new items or remove current ones and the waiters and customers will be shown the new updated menu as soon as the database processes the change.

In addition to databases, we will also be using Java sockets in order for the waiter to send order information to chefs and for chefs to notify waiters when their submitted order is completed and ready to be delivered to the table. The server machine will be one of the computers at the restaurant, but for this project we designated one of the group member's laptop as the server machine. This machine's IP address will be the one used for the server class and all client classes will be sending information to that IP address. All devices must be connected onto the same internet connection for communication between programs. In the restaurant it will be whatever their wifi setup is, and for our case we used a mobile phone hotspot. Doing it this way makes it so that employees cannot access the software from outside of the restaurant, say for example from their home.

8.6 Global Control Flow

8.6.1 Execution Orderness

Our system is a procedure-driven system. This means that the system waits for an employee or customer to go through a given set of steps in order to initiate a command. Each user, depending on their status, goes through the same steps in order to complete the task they need to accomplish. For example, if a customer wants to pay their bill, every customer will go through the same set of commands to complete their task: Click “Pay Bill”, Click “Confirm”, Swipe Card, Write Signature, and take the survey. This applies to each user completing each task.

8.6.2 Time Dependency

Our system works with timers in relation to the kitchen aspect as the manager and chef need to gauge how long a customer has been waiting for their order to be made and delivered to their table. So the kitchen aspect is a real-time system whereas the other sectors of the system are event-response type. Each dish is given an individual completion time estimation that the chefs utilize in order to finish all meals for one table around the same time. So since these times are varying, the system is not periodic. The constraint of time on each table is around 30 minutes from the time of ordering. The chefs will be encouraged to finish the food at a much faster pace than 30 minutes, but 30 minutes is the time of which the manager would be signaled to go to the table, apologize, give the table their meal for free, and thus cost the restaurant the price of the ingredients to make the food on the table.

8.6.3 Concurrency

Our system will utilize threads as the main goal of the system is to be able to have multiple users inputting commands and working on the system at once. If we do not utilize threads, each command would have to finish before the next would begin which would result in an inefficient restaurant. We would obviously use a different thread for manager tasks, customer tasks, and kitchen tasks as those need to be completed simultaneously. These threads would interconnect as they need to communicate commands between the three subsystems. The kitchen would also need multiple threads so that the chef could cook more than one table’s food at a time.

8.7 Hardware Requirements

Our system would need to implement a color display as it will be using colors to display the cleanliness and availability of tables in the restaurant. It does not require a high resolution display as the objects on the screen will be of little detail. Since the system will be running on tablets, the screen resolution should be around 1024 x 600, or around the same as a low price tablet. The system, although relatively simple, is composed of a large database of information from the employee information to the inventory to the survey answers, so the hard drive needs to be able to contain all that information. The main computer, the one which the manager is using and stores the load of the information, should contain at least 100 GB of space in order to hold everything. The tablets, on the other hand, will wirelessly transmit their information directly to the main computer system, requiring that they only have around 2GB of hard drive space. The kitchen computer must also be able to store a large amount of information and should have around 10GB of space. Although the restaurant does not have an excessive number of employees, or tables for that matter, it will rely on a fast and efficient network bandwidth to ensure the data is transmitted quickly from one subsystem to the next, requiring a minimum of 2Mbps network bandwidth.

9. Algorithms and Data Structures

9.1 Algorithms

For this system, a majority of the algorithms used will be for communicating with databases to retrieve or update any information stored in them.

9.1.1 Waiter

The waiters will have an algorithm that deals with table status. This requires the algorithm to communicate with the server to mark a table as clean, dirty, or occupied which would then update all the waiters' tablets with the new status.

9.1.2 Chef

Chefs will utilize an algorithm for putting orders into a queue. This allows orders to be cooked on a mainly first-come, first-serve basis and allow efficiency for serving customers, but also taking into consideration the cooking time durations of certain items. Another algorithm will be used to update the inventory as orders are cooked. When a chef marks an order as complete, it is then removed from the queue and the ingredients used are automatically subtracted from the inventory. In the case of restocking the inventory, the chefs can order more ingredients and have the inventory update with new quantities. The algorithms that modify the inventory access the data stored in the SQL database. In the case of the inventory update algorithm, the string containing order information is serialized into individual dishes, and then each ingredient is serialized from the list of ingredients for that particular dish from the database. Then, the new ingredient quantity is calculated and committed into the database. To display data to the user, the database fetches information based on user defined parameters, and each column is displayed in its particular user interface location.

9.1.3 Manager

The manager system contains many simple algorithms to help maintain and update any necessary data, such as employee records and payment. Most of the algorithms deal with maintaining the SQL databases and the database tables themselves to automatically update with all changes done from the management side. Adding items, removing items, and updating items in the databases are some of the algorithms used. For example in calculating an employee's payroll, the time difference between when they "clock-in" and "clock-out" is found in hours and then multiplied by their hourly rate. The times can be retrieved from the databases and the algorithm will update the databases with the hours and pay.

9.1.4 Customer

A queue-like algorithm will be used to seat customers based on time of arrival, party size, and available table size. Customers will be able to order their meals on tablets and send them to the chefs. This is done with an algorithm to communicate with the system to input the customer's order into the chef's cooking queue. A simple adding algorithm will also be able used to get the overall price of the customer's meal which the customer can then pay off at the end. There will also be a counter algorithm to keep track of number of tables served for the restaurant to get an exact number of customers they had. Optional surveys are also given at the end of their meals. Customers will be asked to rate on their experience on a scale of 1-5 and their answers are stored in the databases as well as the number of surveys taken. An algorithm is used to find the average ratings which requires retrieving data from the databases.

9.2 Data Structures

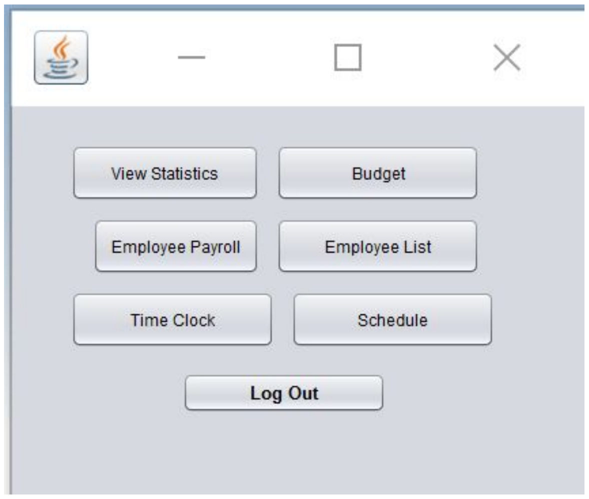
The main data structures that will be used by this system are databases, more specifically SQL databases, and arraylists used as a "queue". This allows a tidier storage of data that can easily be accessed to view outside of the system itself. It also eliminates the dangers of storing all crucial information on any arrays which might lead to a memory leak or error. Databases allow simple communication between the different parts of restaurant, e.g. the kitchen keeps track of how many dishes were made, saves that information into the database periodically, and the management can access that data for revenue and budget purposes. Arraylists offer the ease of manipulating data that is queue sensitive, i.e. it is easy to add or remove items from a queue when using arraylists. Vectors will be used to temporarily hold information that need to be displayed to any tables, but that is the extent of their use.

Using arraylists as a queue, customers waiting for a table can be added or deleted from the queue instantly which allows quick changes to the queue. For chefs, in the event an order is cancelled by the customer or waiter or needs to be cancelled the chef themself, an arraylist would be able to handle deleting one of its objects while maintaining the order of the queue. This means that any slight or extreme changes to the arraylist will not affect the overall efficiency of the chefs.

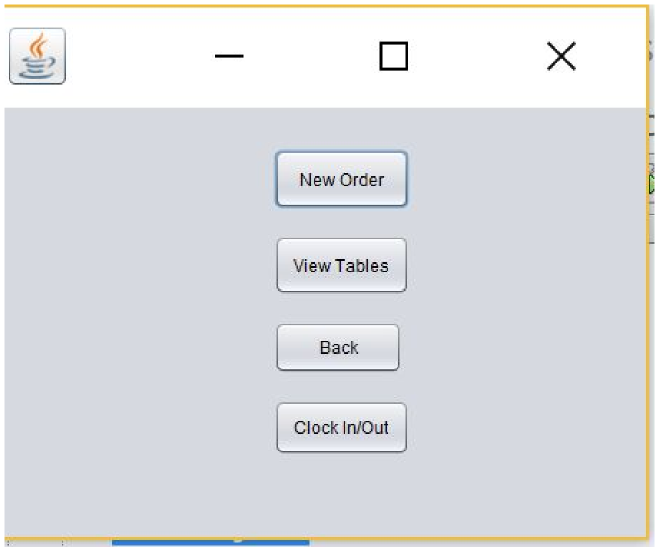
The SQL databases will be used for the remaining portion of the system. They will contain the "backbone" data for the restaurant, such as the menu, inventory, and employee records. With databases, it is flexible in the sense that data is easily created, stored, and edited by parties with the correct access. It also allows a nice visual representation of tables made and a user can see what data is being stored.

10. User Interface Design and Implementation

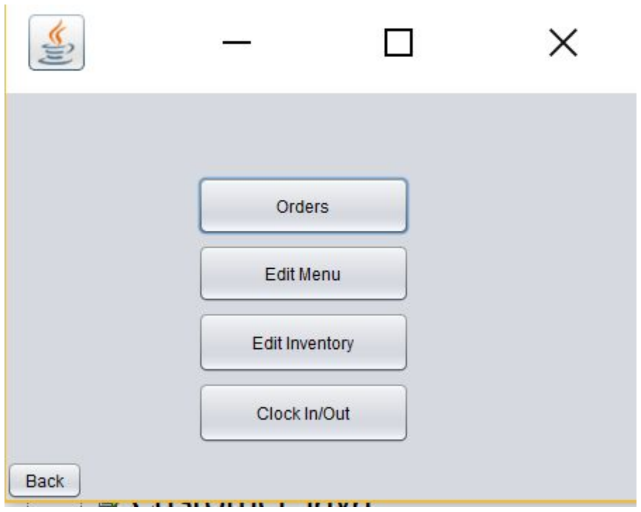
Compared to report two, the user interface of the system has been updated to accommodate all of the features outlined in this report. It has kept the same overall look as was introduced in report two for the login system. What has really changed since the last report is the matter of ease of use for the user. In our system, each type of user (manager, waiter, customer, etc.) has their own specific menu screen to allow simple functionality for each user. For the manager, they are presented with the following screen:



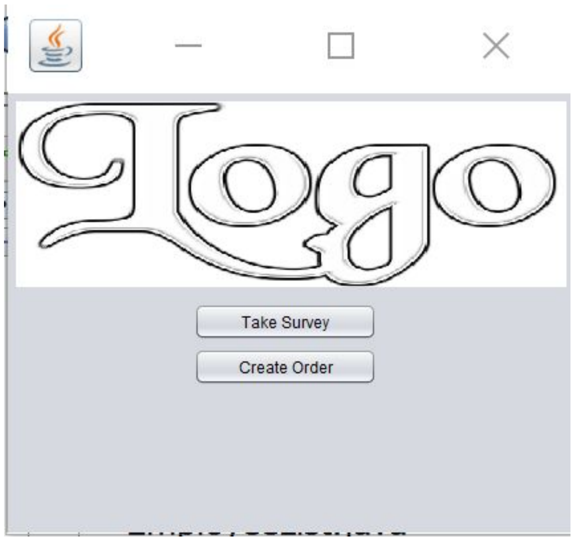
This screen allows direct navigation tools for the manager to use in order to easily access whichever section of the restaurant they want to control. Each button on the menu leads to a separate screen that completes the function that is needed. For ease of use, the UI was differed for each of the possible login combinations so that each employee would be able to access the information that they were allowed to see as well as the information they most needed to see in order to do their job efficiently. This difference can be seen in the information given to the waiter when he/she logs into the system:

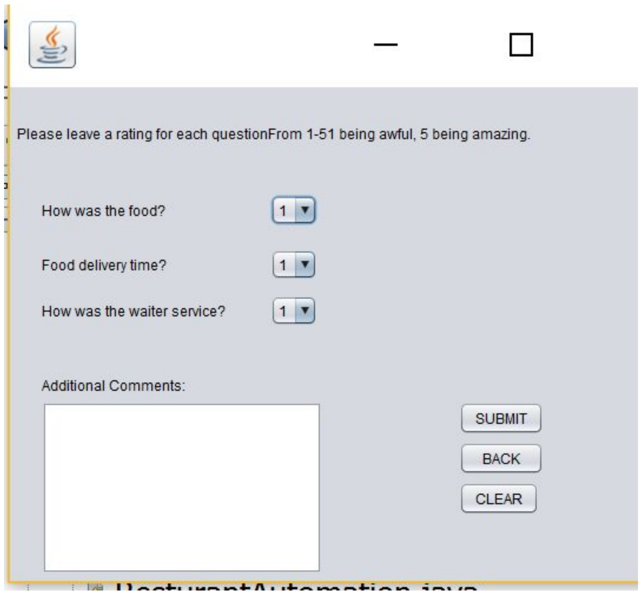


This interface allows for the waiter to see the status of the food tables, create new orders to send to the kitchen, while also giving the waiter the ability to log in and out of the system for the day. For the chef, it was useful for them to see what food is needed to complete, give them access to viewing and editing of the menu and the inventory and also give them a place to log their hours for the day. Because of these requirements, a separate UI menu was created for them to access these specific features:



The last type of user that has the ability to log into the system is the customer. This UI was designed to give the customer access to nothing within the system sans an option to take a survey at the end of their meal to rate their experience and an option to add additional items to their meal they may have forgotten to tell the waiter. The survey is a simple 3-question survey that takes minimal effort from the user to complete:




Please leave a rating for each questionFrom 1-51 being awful, 5 being amazing.

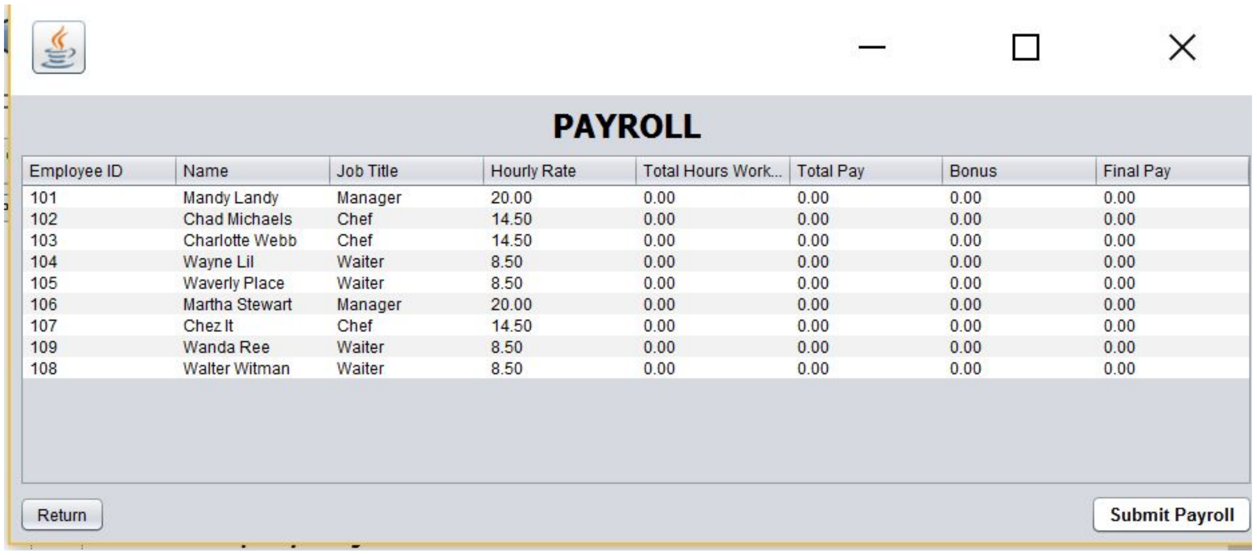
How was the food? 1

Food delivery time? 1

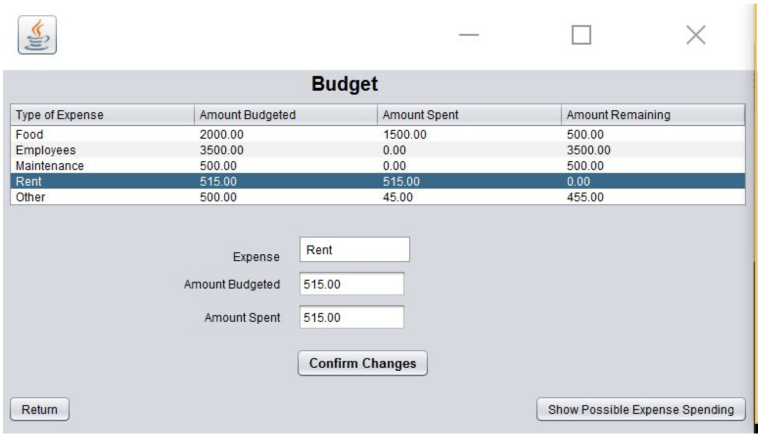
How was the waiter service? 1

Additional Comments:

To give a more in-depth look at the simple user interface and ease of use within our system, it can be seen pretty easily in the sub-menus of one of the logins. For this purpose, the main areas of each specific type of employee will be looked at. For the manager, the most important tasks to look at are the employee payroll and budget, respectively seen below:

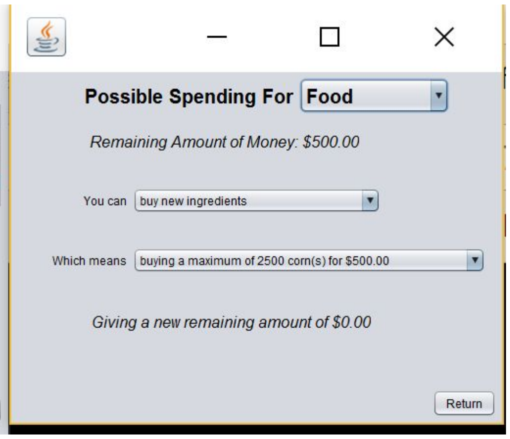

PAYROLL

Employee ID	Name	Job Title	Hourly Rate	Total Hours Work...	Total Pay	Bonus	Final Pay
101	Mandy Landy	Manager	20.00	0.00	0.00	0.00	0.00
102	Chad Michaels	Chef	14.50	0.00	0.00	0.00	0.00
103	Charlotte Webb	Chef	14.50	0.00	0.00	0.00	0.00
104	Wayne Lil	Waiter	8.50	0.00	0.00	0.00	0.00
105	Waverly Place	Waiter	8.50	0.00	0.00	0.00	0.00
106	Martha Stewart	Manager	20.00	0.00	0.00	0.00	0.00
107	Chez It	Chef	14.50	0.00	0.00	0.00	0.00
109	Wanda Ree	Waiter	8.50	0.00	0.00	0.00	0.00
108	Walter Witman	Waiter	8.50	0.00	0.00	0.00	0.00


Budget

Type of Expense	Amount Budgeted	Amount Spent	Amount Remaining
Food	2000.00	1500.00	500.00
Employees	3500.00	0.00	3500.00
Maintenance	500.00	0.00	500.00
Rent	515.00	515.00	0.00
Other	500.00	45.00	455.00

Expense	Rent
Amount Budgeted	515.00
Amount Spent	515.00


Possible Spending For Food

Remaining Amount of Money: $500.00

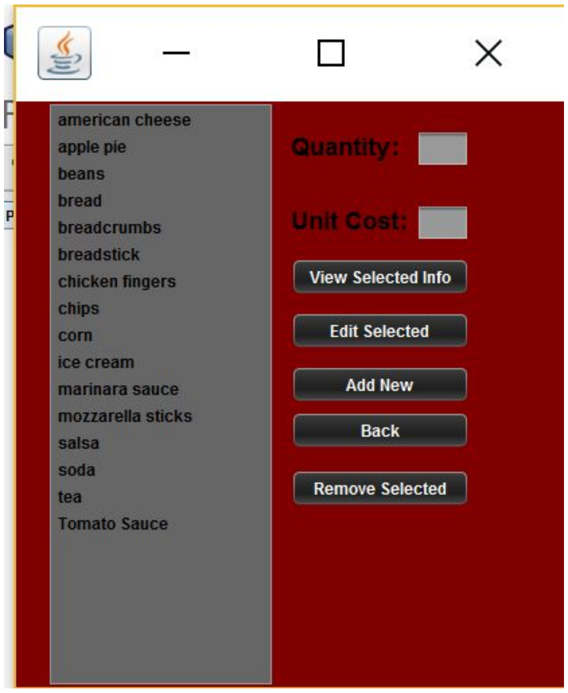
You can buy new ingredients

Which means buying a maximum of 2500 corn(s) for $500.00

Giving a new remaining amount of $0.00

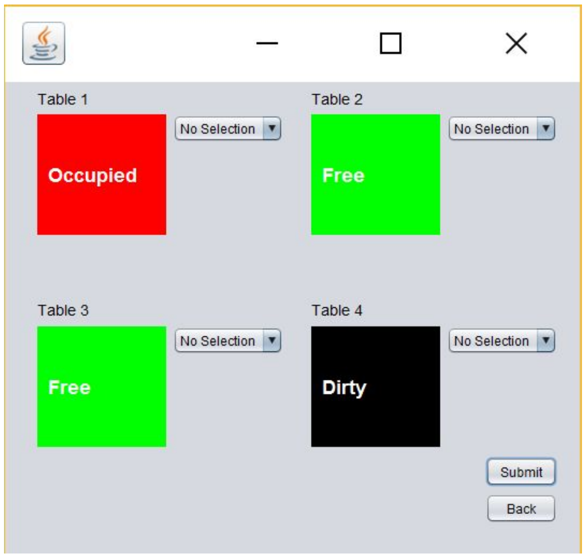
In terms of ease of use for the payroll, it can be seen that little needs to be done in order to submit the payroll for the employees. A simple confirmation button sends the payroll into the system. For budgeting, it was originally difficult to keep an easy enough ease of use for the manager. But it became automated enough to simply choose which budget you would like to see at the moment and change the . A function for the manager to see what any extra money that was originally allocated to an area could be used for can easily be accessed through the “Show Possible Expense Spending”, which allows the manager to easily view how he/she could rearrange the budget.

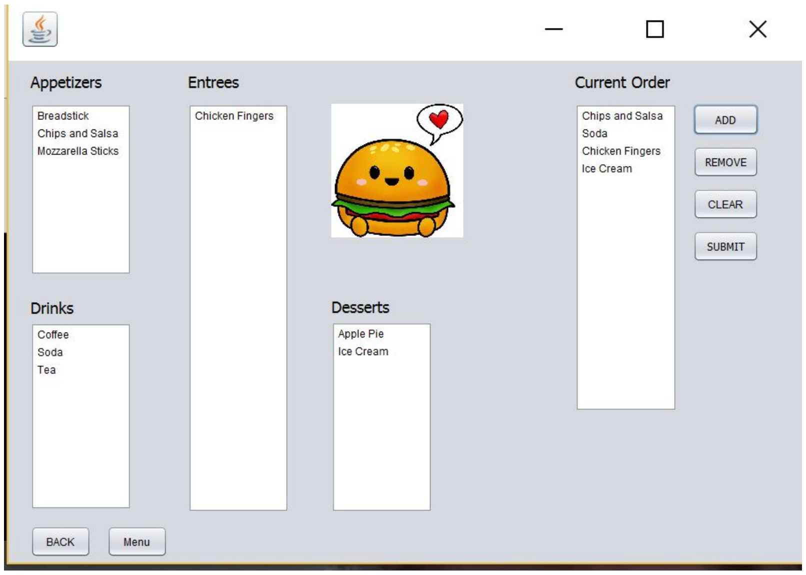
For the chef, the most important functions are the viewing/editing of the inventory as well as the current orders that need to be made and served. These two sub-menus can be seen below:


american cheese
apple pie
beans
bread
breadcrumbs
breadstick
chicken fingers
chips
corn
ice cream
marinara sauce
mozzarella sticks
salsa
soda
tea
Tomato Sauce

The order menu, although empty in the above picture, is an easy to use system that lists the current orders that have yet to be finished. When the chef finishes an order, he/she will be able to select the specific dish they finished and press the “Done” button. The inventory easily stores every ingredient that is needed to create each dish on the menu. Information can be seen easily by selecting the specific ingredient and pressing the “View Selected Info”. Adding new ingredients is easily able to be added also via the “Add New” button. The other buttons are straightforward enough and have simple ease of use.

The waiter’s most important tasks are to view tables and to create orders. These interfaces are shown below:





The waiter must be able to view the tables in the restaurant and always know the status of each table (available, occupied, or dirty). They can simply change the status of a table when they seat a new customer or allow the busboys to know when a table is dirty and needs to be cleaned.

As for the menu, the ease of use is simple. The waiter simply selects which items from the menu the customer would like and selects add and submits the order when everything is ordered.

Overall, the ease of use of our system is very simple and straightforward for the user to understand, no matter their role in the running of the restaurant. This easily usable system makes for an efficient and automated restaurant.

11. Design of Tests

11.1 Test Cases

11.1.1 Chef

Test-Case Identifier: TC-01
Function Tested: addToMenu(string itemName, double Price): bool
Pass/Fail Criteria: Test passes if the new item is added to the menu list, with a popup dialog message. Fails if popup dialog says failure.

Test Procedure	**Expected Results**
Call Function(Pass)	Item will be added to the menu
Call Function(Fail)	If item already exists it will return false

Test-Case Identifier: TC-02
Function Tested: editMenuItem(string itemName, double Price): bool
Pass/Fail Criteria: Test passes if existing item is updated, and updated information is reflected in the MySQL table.

Test Procedure	**Expected Results**
Call Function(Pass)	If item name already exists and has a different price, the menu item is updated with the new price.
Call Function(Fail)	If item already exists with the same price or if the item does not exist, the function will return false.

Test-Case Identifier: TC-03
Function Tested: removeFromMenu(string itemName): bool
Pass/Fail Criteria: Test passes if an existing item is removed from the menu list. And on the waiter menu screen the item is indeed removed.

Test Procedure	Expected Results
Call Function(Pass)	Item is removed from the menu.
Call Function(Fail)	Returns false if the item name is not found in the menu list.

Test-Case Identifier: TC-04
Function Tested: viewOrders(): void
Pass/Fail Criteria: Test passes if the orders are correctly displayed on the chef's computer screen.

Test Procedure	Expected Results
Call Function(Pass)	The orders are displayed when the function is called.
Call Function(Fail)	The orders are not displayed when the function is called.

Test-Case Identifier: TC-05
Function Tested: doneOrder(): void
Pass/Fail Criteria: Test passes if order is removed from the order list.

Test Procedure	Expected Results
Call Function(Pass)	Order is removed from the order list.
Call Function(Fail)	Order does not get removed from the order list.

11.1.2 Waiter

Test-Case Identifier: TC-06
Function Tested: submitOrder(): bool
Pass/Fail Criteria: Test passes if the order is successfully sent to the chef's order list. Waiter screen will recieve a dialog message saying it was submitted. Also chef's screen will see the recently submitted order.

Test Procedure	Expected Results
Call Function(Pass)	The order information is sent to the chef's order list and the function returns true.
Call Function(Fail)	The order information did not get sent to the chef's order list and the function returns false.

Test-Case Identifier: TC-07
Function Tested: orderMenu(): void
Pass/Fail criteria: The menu is correctly displayed on the waiter's interface.

Test Procedure	**Expected Results**
Call Function(Pass)	The menu items are correctly displayed on the waiter's interface.
Call Function(Fail)	The menu items are not shown or not shown correctly on the waiters interface.

Test-Case Identifier: TC-08
Function Tested: viewTables(): void
Pass/Fail Criteria: Test passes if the status of each table is displayed on the waiter's interface.

Test Procedure	**Expected Results**
Call Function(Pass)	On the waiter's interface, he/she is shown the status of each table.
Call Function(Fail)	Table status is not shown to the waiter on their interface.

11.1.3 Manager

Test-Case Identifier: TC-09
Function Tested: viewStatistics(): void
Pass/Fail Criteria: Test passes if the statistics are displayed properly on manager's interface.

Test Procedure	**Expected Results**
Call Function(Pass)	Restaurant statistics are properly display on the manager interface.
Call Function(Fail)	Statistics are not displayed or displayed incorrectly.

Test-Case Identifier: TC-10
Function Tested: editBudget():bool
Pass/Fail Criteria: The budget is changed appropriately to show the user's new information.

Test Procedure	**Expected Results**

Call Function(Pass)	The budget is changed and the function returns true.
Call Function(Fail)	The budget is not changed or changed incorrectly and the function returns false.

Test-Case Identifier: TC-11
Function Tested: payroll():void
Pass/Fail Criteria: The payroll is displayed properly on the user's interface. Test fails if the information displayed is incorrect.

Test Procedure	Expected Results
Call Function(Pass)	The payroll is displayed correctly with the correct information concerning the payroll (employee names, hours worked, etc.)
Call Function(Fail)	The payroll or the information on it is not displayed or displayed incorrectly.

Test-Case Identifier: TC-12
Function Tested: sendPayroll():bool
Pass/Fail Criteria: The payroll is sent to whoever will manage it. Popup dialog with message "Success" will be shown to user if passes. Otherwise popup dialog with message "Failure" will mean failed.

Test Procedure	Expected Results
Call Function(Pass)	The sending process is executed properly and the payroll is sent to the recipient and the function returns true.
Call Function(Fail)	The sending process is not executed properly and the payroll is not sent to the recipient. The function returns false.

11.2 Test Coverage

The test cases mentioned above cover the important functions of our system. Those functions are the backbone to any restaurant, so most of our other use cases previously mentioned in the report coincide with these test cases. A majority of our functions deal with communication to a database and making sure the correct information is displayed or changed, and apply to all major operations within restaurant business. Along with the test cases, the functions can also be tested for successful execution by looking at the databases themselves to check for the same values or objects being displayed by our interface. As more functions are implemented to our system in the future, more test cases will be designed if the above cases do not cover them, derived from our already existing major functions and test cases.

11.3 Integration Testing

Our main Integration Testing strategies include big-bang integration testing and the bottom-up approach.

A specific type of big-bang integration testing that has proved optimistic is usage model testing. Having three pairs of groups within our team working on separate restaurant components, we have been able to conduct isolated testing on our own components and proceed with a test of all components combined. Of course, this approach assumes that there are few problems with the individual components themselves, and saves time since the components been isolated and tested before.

The bottom-up approach has been as effective in its combination with the big-bang approach when we have needed it most. This method involves testing the lowest-level components first, in order to facilitate higher-level component testing. We can apply this technique to our testing of budget functionality and menu functionality. For example, test the functionality of items on the menu, and that they can be viewed properly, before testing if the menu can be edited. This minimizes room for error and saves times as well.

12. Project Management, History of Work, Current Status, and Future Work

12.1 Merging the Contributions from Individual Team Members

As a means of compiling our individual work, we utilized Google Drive as a tool to both communicate with each other and simultaneously work on report components in separate documents and spreadsheets. Some issues that we encountered and were able to overcome were grammatical errors, formatting issues such as varying font and structure, and initial organization errors (fits into the concept of varying font and structure as well). We were able to easily resolve organization and formatting issues by making use of a Table of Contents at the beginning of every component document/report and using a similar template each time. By creating a Table of Contents and distributing work amongst pairings within the group, we were able to find and keep track of our work in the report, allowing us to check easily for grammatical errors and resolve any other human-error associated issues. We also utilized the app GroupMe as a means of communication between the team. This allowed any team member to have his/her question or message answered in a timely manner and also served as an archive of any conversations not held in person that any member could go back through if they wanted to double check anything.

12.2 History of Work

Jan 22 - Jan 29

We formed our group and chose to take on Restaurant Automation as our project focus. We met in person and discussed/brainstormed ideas for each of our three main restaurant components: Food Preparation, Management, and Customer Service. With prior research done both individually and collaboratively on previously submitted Restaurant Automation reports, we were able to devise a proposal including an extensive conceptual definition/pre-layout of our plan of action.

Jan 30 - Feb 19

Utilizing our proposal and ideas we built upon over the course of the few days leading up to the introduction of the first report, we developed a more structured and defined layout for our plan, which included, according to the outline provided to us, a problem statement, functional and nonfunctional system requirements and on-screen appearance requirements, requirements specification (use cases, traceability matrix, and system sequence diagrams), user interface specification, and domain model, as well as a plan of work for each.

Feb 20 - Feb 26

Our group started drafting and completed the first part of the second report. Using our first report, we created UML interaction/sequence diagrams for our fully-dressed use cases, specifically UC-6 (Edit Menu), UC-9 (Order Done), UC-19 (View Statistics), and UC-22 (Payroll Status). A design for each fully-dressed use case implementation (each of the four above) is being worked on currently.

Feb 26 - Mar 5

Our group drafted and completed the second portion of our second report. Continuing what we started in the first part, we focused on two main components: Class Diagram and Interface Specification and System Architecture and System Design. We included a class diagram, list of data types and signatures, network protocol, global control flow, and architectural styles that we employed as a result of weeks of thought and planning.

Mar 6 - Mar 12

During this week, we compiled all of our work and planning from the past three weeks to produce the final version of our second report, for which we focused on algorithms, user interface design and implementation, and design of tests (which included our test cases and integration strategy).

Mar 13 - Mar 26
In this time period, we spent our time coding and designing the basic UI and functions of our system for Demo #1. With the help of each other and the internet, we were able to gain a basic understanding of Java and MYSQL. We also spent the two days after Demo #1 gathering and working on our documentation for submission.

Mar 27 - Apr 23
After our presentation of our system in Demo #1, we have since received feedback from the reviewers on what to improve and add to make our system more marketable. As a group, we continued to think of ways of how our system could solve the restaurant automation problem. We added new functions to our system to further emphasize the automation portion and fixed some UI problems. We also took the graders' comments on our previous reports and worked to fix them for Report 3 Part 1. Below are details from the sub-teams/members about improvements they had made in their respective section during this time period.

Inventory/Menu: moved from local storage (text files) to server storage (SQL database). Most of the implementation was already completed for the first demo, so UI improvements were focused on in the second.

Kitchen: increase user's control over the program, allowing user to configure the program to better suit their needs. Improved prioritization mechanism to provide more reliable and modifiable functionality.

Waiter/Customer Service: added table status/management for waiters. They can notify other waiters using the software when a table is free, occupied, or dirty. Also added a optional survey for customers to take, results to these questions are stored in a SQL table.

Management: migrated from arraylist to SQL databases to hold information. Implemented a functional punch-in, punch-out system, a payroll system, a budget viewer and editor, as well as the employee records. Took feedback from first demo and increased automation from management point of view.

Apr 24 - May 2
We presented our system for Demo #2 and since then, we have been working on the final version of report 3 as well as our overall project submission to the archive and our reflective essays.

12.3 Current Status

Currently, we are compiling our documentation for the electronic web archive and finishing report 3. Individual group members are also finishing their reflective essays.

12.3.1 Accomplishments

Since the beginning of this project, our team has accomplished many feats. We improved our programming skills in an unfamiliar language as well as our documentation skills. Below is a bulleted list of our other accomplishments.

- Setup SQL databases as the backbone to our server and communication between parts of a restaurant
- Implemented a working GUI that works well with our programming
- Used Java sockets as another means of communication
- Implemented a functional electronic menu, inventory, and employee database
- Created a way to delegate tasks between chefs to allow cooking efficiency
- Utilized the SQL databases in a way to offer real-time updates to other employees

12.4 Future Work

Now that our second demo is over and submission is near, our main goal for future work will be focusing on improving our system. Even though we implemented our system to have a basic functionality, we would like to continue on improving our product after Demo #2 in terms of UI and actual functions. Future plans for the separate restaurant sections are below.

Inventory/menu: automatic ordering new of inventory from an online store, automatic estimation of nutritional facts, recipe database with step by step instructions.

Kitchen: develop GUI, integrate with server to receive waiters' orders, integrate with inventory to access full menu and inventory information.

Waiter/Customer Service: add the ability for waiters to customize customer orders. For example if a customer orders a cheeseburger and they want no tomatoes, extra onions, etc. Also allow customers to order additional food items from their table if desired. And allow waiters to edit a submitted order within about 5 minutes, just in case there was a mess up.

Management: a more in-depth helper for the budget to show more detailed options on spending excess money, implementation of a working request form for employees to request a vacation or other requests, better shift scheduling, and automation for statistics.

13. References

1. **2015 Group 3's report for format of document (referenced for overall report)**
   http://eceweb1.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf

2. **Wikipedia's explanation of interaction diagram (referenced for Section 6 Interaction Diagrams)**

https://en.wikipedia.org/wiki/Interaction_overview_diagram

3. Architectural Styles overview (referenced for Section 9 System Architecture and Design)
https://msdn.microsoft.com/en-us/library/ee658117.aspx

4. User Story explanation on Wikipedia (referenced for Section 2 User Stories)
https://en.wikipedia.org/wiki/User_story

5. Java Socket Documentation (referenced for Section 9.5 Network Protocol)
https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html