14:332:452:01 Software Engineering Spring ‘14
Group 4
Final Report

WHY WAIT

A Restaurant Automation System
http://mitulgada.wix.com/whywait

Group Members:
Amgad Armanus
Jake Chou
Mitul Gada
Avni Patel
Nirjan Thayaparan
Diego Urquiza
Christian Youssef

Project Management

All team members contributed equally.

Summary of Changes

- **Functional Requirements - Use cases**
- **Effort Estimation**
- **Domain Analysis - Concept Definitions and Traceability matrix**
- **User interface design and implementation**
- **Design pattern**
- **Future work**

Table of Contents

1) Customer Requirements

a.) Problem Statements

The following are problems faced in restaurants today and are divided by the different positions.

Chef

Problem -Because chefs must constantly remain in the kitchen, they cannot be bothered with the placement time of orders. There is a great inefficiency between the time customers order food to the time when they receive the food. The waiter has to take the order and deliver that order to the kitchen, then the chef must make sure that that meal are prepared on time while simultaneously preparing other meals as well. The food then sits in the kitchen until it is picked up by the waiter for table delivery. The inefficiency is even more faulted since there are multiple customers who face these same problems. For the chef, whose priority is to cook the food, it is arduous to worry about these external issues especially as there is a multiple of customers and just as many meals to cook.

Solution - The chef would want a system that would allow them to get orders automatically, prioritize the meals ordered, and then notify a waiter that the meal is ready to be taken.

Problem - Between cooking and taking orders, there is a ton of room for error when it comes to supplies. One day there can be too many carrots and another there can be too little onions. This can entail a huge loss in profit since a customer can order something and the supplies may not be present. There could also be a loss of profit if we oversupply and the ingredients are essentially wasted. Lastly, there are certain days of the week when more supplies are needed than they are on other days. These days, the chances of not having the supplies and evidently losing profit from making the food is more likely.

Solution - It would be very beneficial if chefs could log the amount of ingredients used and what is needed. Also if there was a way to analyze when more supplies are needed on certain days, this would make it easier and take away the guesswork on how much is needed.

Problem - In the kitchen, the chef is preparing multiple meals all at the same time. Be it a gourmet salmon dish or fresh bread, the chef must focus on each detail to ensure quality, but also making sure it is delivered as quickly as possible. Thus, a problem arises when the chef must focus on cooking meals and ingredients with different timings while simultaneously estimating the time for each to complete. With all the different foods that all begin cooking at different times, it is hard for the chef to keep track of the times of the food, which can lead to reduced quality in meals and slow down preparations for upcoming meals.

Solution - The chef would want the system to assist in keeping tabs on when multiple ingredients or meals are being cooked for a certain amount of time, and when they have completed. This way chefs would not have to worry about cooking time and could instead be notified when food is finished cooking.

Customer Sign-in

Problem - Restaurants these days are always trying to find a way to satisfy customers. One thing they always look at is the waiting and seating arrangements in the beginning. Often when restaurants are packed, customers that come in have to wait. The problem is, restaurants don't know exactly how long it will take so they estimate it. Restaurants want to be more organized with their table management and keep track of their customers. This also leads to another problem with the status of the tables. Restaurants want to know if tables are ready to clean immediately so they can be attended to. Restaurants will also want to know how long customers have been sitting at the table. This will help tell them if they are almost done so the busboys can clean the table. Overall the problem restaurants are trying to fix these days is the satisfaction of the customers by giving them seating faster, and the organization of all the

tables so they can be attended faster.

Solution - A solution that is appealing to the restaurant will to have a customer seating PC that will be in GUI. This customer seating PC will be in real time so it can show not only the customers, but the restaurant workers all the tables occupied or not. This way new customers can pick the available tables to their liking. For example, some customers might prefer sitting near a window, or booth, etc. And since this will be in real time, the exact times of when customers were seated will show up. This will overall give a much more accurate estimate of how much longer customers will take at the table since the time is there. Customers can see this when arriving and have a good idea when customers might leave and restaurant workers can use this information to inform busboys that a table is about to be ready to clean. This organized system will not only help restaurant workers being faster and efficient, but improve customer satisfaction.

Cashier

Problem - When doing transactions, restaurants aim for a fast and errorless execution to make it easier for the customer. One thing restaurants are worried about is receipt handling. Some customers might prefer an electronic form for a receipt so they won't lose the hard copy. Many restaurants only offer hard copies for receipts which may be unsatisfying for some customers. Another problem is the amount of change given. Often times the receptionist takes longer handling the change and might even make a mistake forcing to fix the issue which ultimately slows down the business. It is in the restaurant's best interest to handle the final transaction professionally to satisfy the customers and to organized the business.

Solution - A simple solution that restaurants will love is an automated cashier tablet that will have an option to either print out a receipt, have it emailed, and even both if wanted. This option will be appealing to all customers. The customers who want the hard copy can get it and the ones who often lose things or unorganized are able to get an electronic form to access easily. This also helps the restaurant be organized since they can keep the files tabbed and

easily accessible as well as saves money by printing out less receipts. Another problem it solves is the slower process and human error. By having an automatic change dispenser, it eliminates the error of the receptionist giving out the wrong change. This not only speeds up the process but also saves the transaction time which ultimately improves customer service.

BusBoy

Problem - A lot of times, busboys would be standing around waiting for tables to be cleaned once the customers leave. Restaurants around would want busboys to clean the tables as soon as the customers left to speed up the service. Also, sometimes the waiter will also need help bringing out dishes to customers with large parties. Busboys will be needed to help out the waiters. Restaurant's want to fix the problem where the busboy will always be ready and there fast to help out the customers.

Solution - A solution restaurants will need is the busboy tablet. This tablet will notify the busboys of the tables needed to clean immediately so they can get straight to it. This will speed up the business and improve customer satisfaction since the tables will be ready faster. The tablet will notify the busboys of all duties needed to perform which improves efficiency. Another example is when a waiter needs help to carry food out. The tablet will tell the busboys and they will go to help the waiters serve food making the restaurant an altogether team helping out one another to improve the business.

Manager

Problem - Restaurant managers spend a lot of their time managing inventory and manually counting food items at the end of each day in order to record data in the log book. Managers also generate payroll for each employee at the end of each week. Although it is difficult to track popular food items by manually going through the order receipts and counting the number of times an item is ordered, managers need a simple way to get statistics about food items, employee performance, restaurant revenue and average wait times.

Solution - We believe that managers should be able to view statistics about the restaurant more efficiently without doing the tedious work of gathering the information themselves. Our tools will allow the manager use these statistics to make the best business decisions. Whether its to discontinue an unpopular menu item or to speak to an employee about their lack of performance or attitude our tools will provide an accurate representation on how the restaurant is functioning. An inventory system that will monitor incoming orders and adjust the inventory accordingly is ideal for accomplishing these task. It will also allow the manager to adjust the inventory manually whenever they need to. An automated system can send a low stock alert to the manager whenever an item reaches a certain threshold.

Problem- Managing payroll can be difficult for any one person to precisely handle. Usually the Manager is in charge of payroll but he has a lot of other responsibilities to handle beside managing hours. And during stressful/hectic days the manager has a lot of deal with and has a higher chance of making mistakes on an employee's payroll. Not to mention the old fashion way of doing payroll is with paper and pencil. The same usually goes for the calendar schedule of all employee shifts. Managing the calendar can at times be a big hassle especially when shifts need to be moved around and/or updated.

Solution- Technology is the way to go for more a precise and accurate database of information. Using a server will keep track of every employee's work hours that will stored into a database that will be backed up. This means that manager can easily access all this information without having to worry about updating unless he needs to. Using automatic software that updates itself reduces the chance of pencil and paper error alongside saving the manager a lot of time he can use on running the restaurant instead. Using a calendar interface is far more simplistic, easy to use, and cleaner to use and manager. The calendar can easily be shared among workers and offer the ability to allow workers to cover or change shifts in case of emergencies.

Waiter

Problem- In a customer service oriented business, customer satisfaction is the greatest asset. A common problem restaurants face is customers having to wait a long time to be able to speak with their waiters. This is caused by many issues: the wait staff might not be the fastest people around, they are constantly going back and forth to the kitchen so they aren't around to be able to help customers, also they have so many things to do at once so, sometimes they forget that a customer didn't get their food or something. These are all things that affect a restaurant's customer satisfaction rating because no one likes having to wait for their food. When people come to a restaurant, they come hungry. So serving them as soon as possible is essential otherwise customers will be unhappy.

Solution- To fix this, waiters should get some sort of reminder every few minutes to check the tables that they were assigned to. Also, it would be great if waiters could place and check the status of the orders without having to go to the kitchen. Anything to keep the waiters on the floor and out of the kitchen would help.

Problem- Another issue is sometimes waiters don't have the best handwriting. This creates miscommunication between the customers and the chefs. Customers who order one thing and get something different, they would not be happy with that. Also, this significantly decreases profit margins because food is thrown away and more money is spent making another order.

Solution- It would be good if there could be something to make communication easier between the waiters and chefs.

Problem- Sometimes another problem is the wait staff doesn't have the entire menu memorized. So, if a customer has a question about what ingredients are in the food, waiters have to go to the kitchen to ask the chefs. This is a big waste of time, and time is money in any industry. Also, the menu changes from time to time so its difficult to keep track of what is

actually on the menu.

Solution- It would be beneficial to have a menu with the ingredients listed so that the waiter wouldn't have to make the extra trip to the kitchen every time a customer has a question.

Problem- Another matter in restaurants is a customer complaining about the wait time after their done eating. At this point customers want to leave as soon as possible and waiters are generally busy taking care of other customers. This process takes a long time. Waiters will come to the customers with their bill and in the mean time take care of some other work. Then they go back to get the payment, take care of some more work, and then finally give the customer a chance to give a tip. The entire process takes anywhere from 5-15 minutes. Also, a lot of times restaurants will get big groups and splitting the bill is a hassle. Sometimes one person gets charged for another person's bill or the split is incorrect. This is very taxing on the waiters who are thinking about the other tables they are waiting on and sometimes customers just walk away without paying. This increasing problem of theft is solely due to the latency between waiters giving the bill and getting the payment from customers.

Solution- If there is a way for the waiters to immediately charge the customers for their tab with a portable credit card reader that would allow the waiters to split the bills that would make it easier for both parties. This would decrease theft because payment would be taken on the spot from the customers.

2) Glossary of Terms

Technical Terms

Database: the file where the menu items, inventory, scheduling and orders are stored.

Order Queue: a list of orders that are placed in first in, first out order. These orders are sent to the Chef PC when the chef can prepare them.

Inventory System: a system that can be accessed by the manager or the chef, this system is used to manage the inventory of food items.

Walk-in Queue: a list of table reservations that are placed in first in, first out order. These reservations are made on the Customer PC when the customer walks into the restaurant.

Graphical User Interface (GUI) - interface that allows allows easier user communication via pictures and texts

Non-Technical Terms

Customer: Any person that orders an item from the menu or walks into the restaurant.

Waiter: The person that takes the orders of the customers, enters the orders into the system and delivers the orders to the customers.

Manager: The person that is responsible for inventory management, employee scheduling, payroll and customer satisfaction.

Chef: The person that receives the orders on the Chef PC and prepares the food.

Cashier: The person that charges the customers after their meal.

Ingredient: Inventory items that are used to prepare menu item.

3) System Requirements

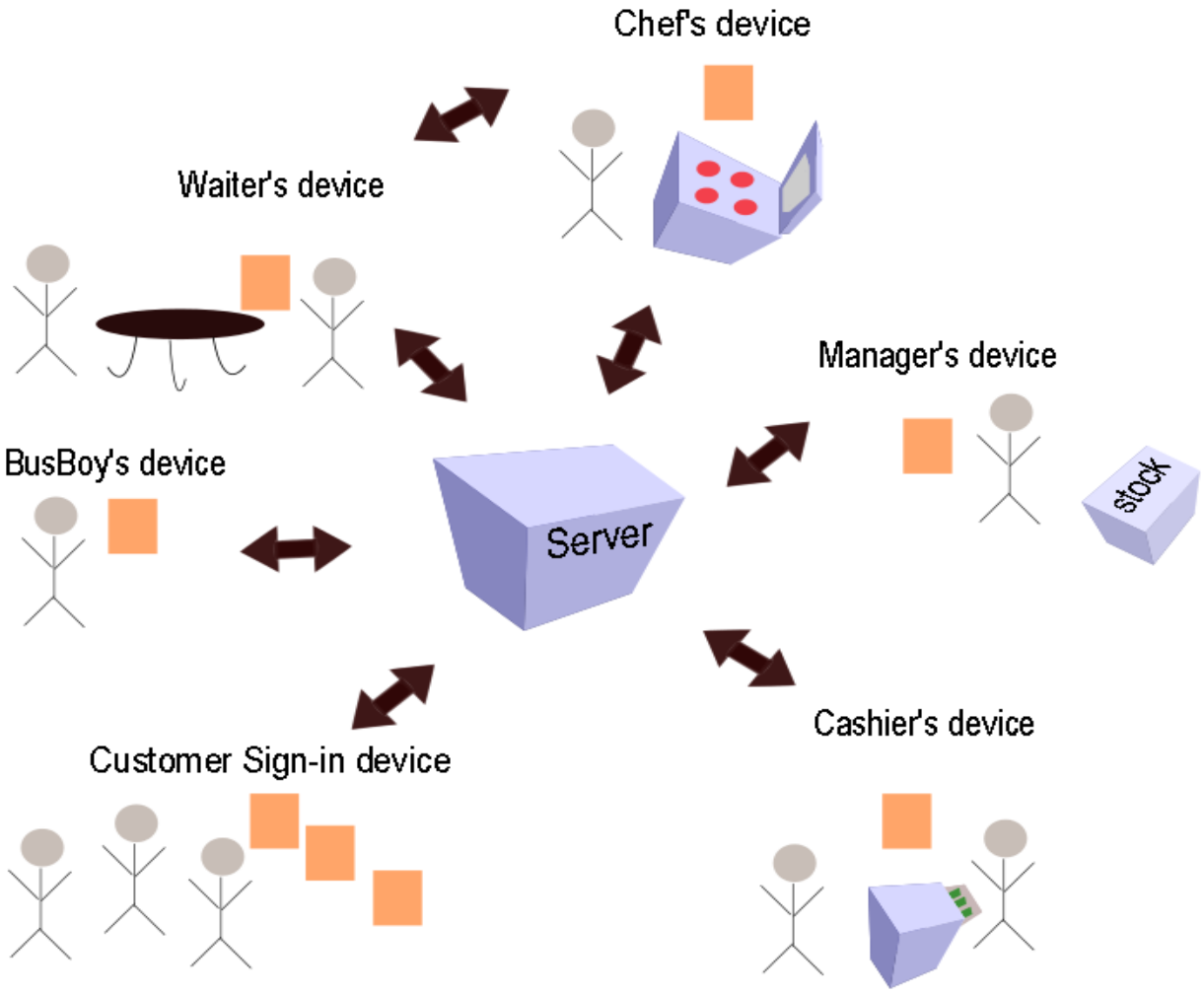


Figure 1.
This is the general flow of how the restaurant will work with the Server. Each peach rectangular object represents a

tablet/device that will communicate with the server

a.) Enumerated Functional Requirements

Identifier	PW	Requirement
REQ - 1	1	The customer sign in PC shall keep track of the customers at the table and record the time they've been waiting.
REQ - 2	1	The cashier tablet shall have an option for the customer on what kind of receipt they want (printed copy or email)
REQ - 3	2	The busboy tablet shall notify the busboy what tables to clean.
REQ - 4	4	The chef PC will see customers meals prioritized by order time(when did they order)
REQ - 5	5	The chef PC shall receive orders from waiters
REQ - 6	3	The chef PC must be able to notify the waiter that a meal is ready to be delivered
REQ - 7	2	The chef PC must be able to set cooking timers and be notified when they are done
REQ - 8	4	The chef PC should notify the system to update inventory when a meal is ready to be delivered
REQ - 9	2	The manager PC shall be able to analyze and predict supply usage
REQ - 10	3	The manager PC will keep track of all employee hours for payroll alongside a calendar GUI with everyone's work schedule
REQ - 11	4	The manager PC will dynamically update inventory as orders ready to be delivered. Also, when ordering shipments of supplies, quantity of each supply with be updated.
REQ - 12	1	The manager PC will have statistics on food popularity and employee performances.

REQ - 13	4	The manager PC will have the ability to add, edit or remove items from the menu.
REQ - 14	2	The manager's PC shall alert the manager when the inventory has a low stock of a particular item.
REQ - 15	2	The manager's PC shall keep track of daily, weekly, monthly, and annual revenue.
REQ - 16	5	The waiter PC will allow the menu to be viewed with what ingredients are used.
REQ - 17	4	The waiter PC will be able to place separate orders per customer.
REQ - 18	1	The waiter PC will allow bills to be split and paid on the spot with credit card reader.
REQ - 19	1	The waiter PC will send reminders to user to go check on table periodically.
REQ - 20	3	The customer sign in PC will let waiter PC know which customers came first.
REQ - 21	4	The waiter PC must be able to notify the chef PC of any food allergies/special instructions

b.) Enumerated Non-Functional Requirements

Identifier	PW	Requirement
REQ - 22	1	The program should be aesthetically pleasing and meet the standard of the restaurant.
REQ - 23	5	Security measures need to be made to make sure non-users do not login.
REQ - 24	4	Users should read the manual before using the program.
REQ - 25	5	The system should be backed up in case of failures to avoid losing inventory count and orders.

REQ - 26	5	The system should have a low mean time between failures(MTBF)
REQ - 27	2	Time between different screens on the program should be minimized.
REQ - 28	1	The system should work on any type of device or PC.
REQ - 29	3	The system should be easy to use by users that are not technologically advanced.
REQ - 30	4	The system should be easy to debug.
REQ - 31	1	The system should be compatible with the assistance button.
REQ - 32	2	The device each employee uses should be sized based on what would be most convenient to them.

c.) On- Screen Appearance Requirements

Identifier	PW	Requirements
REQ - 33	4	The system shall display real time table availability.
REQ - 34	2	The system should display average wait time for the next available table.
REQ - 35	5	The system shall display menu items, ingredients and price.
REQ - 36	5	The system shall display the order total price.
REQ - 37	4	The system shall display inventory for the manager and chef.
REQ - 38	1	The system should display employee schedules.
REQ - 39	4	The system shall display order queue for the kitchen.
REQ - 40	3	The system shall display revenue, employee and order statistics for the manager.
REQ - 41	3	The system shall display available payment options.

REQ - 42	2	The system shall display a comment box for allergy/special requests.

4) Functional Requirements

a.) Stakeholders

Restaurant Owners: The restaurant owners would be most interested in this system because it immediately affects their business and he wants to improve efficiency and customer satisfaction.

Software Designers: The software designers will also be interested since it will be their job to design it the best to their ability and sell it to people who need it.

Customers: The customers are the ones who will rely on the system to order their food and for their food to arrive as quickly as possible while also desiring the best possible restaurant experience

Restaurant Employees: The employees will have an interest in the system because they will rely on it to ease the burden of their responsibilities and help create more efficiency within the restaurant

b.) Actors and Goals

Initiating Actors

Manager: The employee in charge of the restaurant who must manage the system. The goal of the manager is to make sure the system is up-to-date, properly functioning, keep track of employee information and activities, and manage the restaurant

Waiter: The employee who must attend to customers and service them. The goal of the waiter is to take and deliver the orders to the customers.

Chef: The employee who must cook the food. The goal of the chef is to make the food and maintain ingredients.

Customer: The person who places an order. The goal of the customer is to place their order and pay for their meal.

Customer Sign-In: The employee who checks in customers. The goal of the customer sign in is to seat customers faster and efficiently.

Cashier: The employee who processes checks. The goal of the cashier is to make sure the customer pays their bill in the quickest and most efficient way.

Busboy: The employee who cleans the tables. The goal of the busboy is to clean a table

right after a customer leaves and to help the waiters delivering food if necessary.

Participating Actors:

Database: Stores the data of the system. The goal of the database is to update inventory, keep track of employee information, and manage any other data

Timer: Keep track of cooking time for an order. The goal of the timer is to make sure an order/ingredient is cooked properly

c.) Use Cases

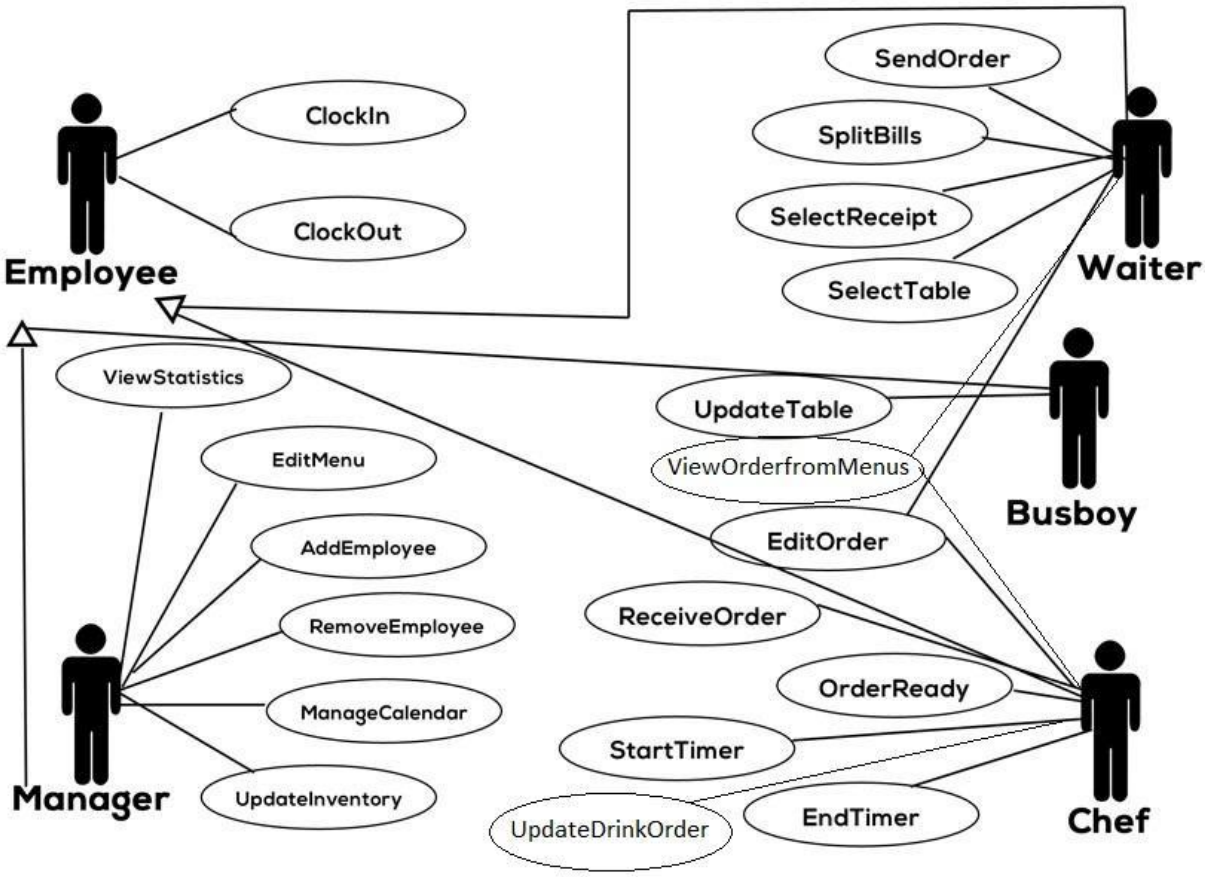
i) Casual Descriptions

Use Case #	Use Case Name	Description
UC -1	UpdateTable	The waiters can change the status of all tables to available or not
UC -2	SelectTable	The tables will be shown and customers can select which one to sit
UC -3	SelectReceipt	The option of receipt with either paper or email is given.
UC - 4	StartTimer	The chef will start the timer to cook an order
UC - 5	EndTimer	The timer will end, notifying the chef

UC - 6	ReceiveOrder	The chef receives the order which is automatically prioritized
UC - 7	OrderReady	The order is ready to be delivered to the customer and inventory must also be update
UC - 8	EditOrder	The chef can edit any order or remove them as well
UC - 9	SendOrder	Waiter places customer order to the system
UC - 10	SplitBills	Waiter charges customer using a credit card swiper, which can be used to split bills if necessary
UC - 11	ManageCalendar	Manager can make changes to the Calendar GUI in order to move/add/remove shifts
UC - 12	UpdateInventory	Manager can manually update the Inventory system. System will also update inventory as orders are placed and as supplies are ordered.
UC - 13	EditMenu	Manager can modify menu items or add/remove an item to the menu
UC - 14	ViewStatistics	Manager can look at all stats of the restaurant ranging from food popularity to payroll
UC - 15	AddEmployee	Enter a new employee into the system
UC - 16	RemoveEmployee	Remove an employee from the system

UC - 17	ClockIn	Employee clocks into their shift
UC - 18	ClockOut	Employee clocks out of their shift
UC - 19	UpdateDrinkOrder	Bartender can update drink orders of customers
UC - 20	ViewOrderfromMenus	Show how drink orders are separate from food orders

ii) Use Case Diagram



iii) Traceability Matrix

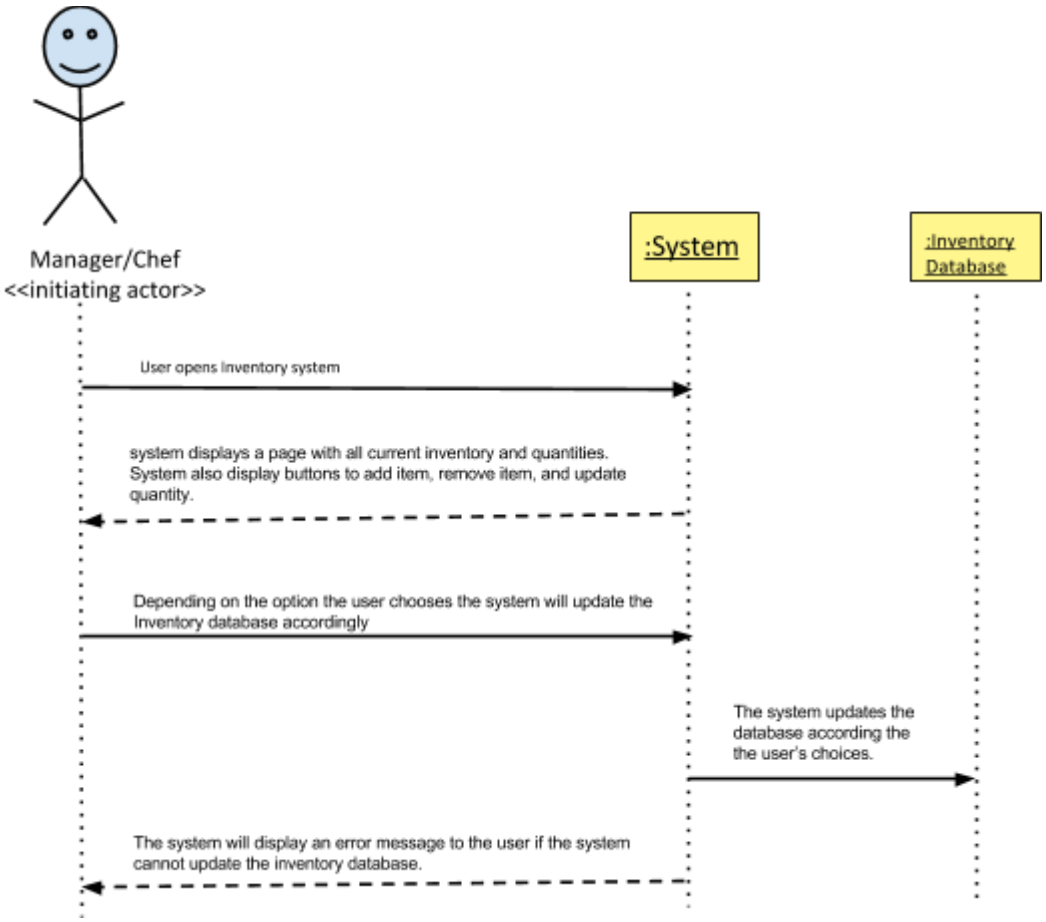
	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16
RQ1	X	X														
RQ2		X	X							X						
RQ3	X	X														
RQ4						X			X							
RQ5						X		X	X							
RQ6							X	X				X				
RQ7				X	X			X								
RQ8								X				X				
RQ9												X	X	X		
RQ10											X				X	X
RQ11												X	X	X		
RQ12											X	X	X	X	X	X
RQ13				X	X			X				X	X	X		
RQ14							X					X	X			
RQ15										X	X	X	X	X	X	X
RQ16								X					X			
RQ17						X			X	X						
RQ18			X							X						
RQ19	X															

RQ20		X														
RQ21						X		X	X							
RQ22											X			X		
RQ23						X	X	X				X	X		X	X
RQ24					X	X	X	X				X	X		X	X
RQ25						X	X		X			X		X		
RQ26					X	X	X	X				X	X			
RQ27							X	X				X	X			
RQ28											X	X	X	X	X	X
RQ29						X	X		X			X		X		
RQ30		X				X	X		X		X	X		X		
RQ31						X	X	X	X			X		X		
RQ32						X		X	X				X	X		
RQ33	X	X								X						
RQ34	X	X								X						
RQ35						X		X	X				X	X		
RQ36			X			X	X			X				X		
RQ37							X	X				X	X	X		
RQ38											X					
RQ39				X	X	X	X									
RQ40											X			X		
RQ41			X							X				X		
RQ42								X	X			X	X			
MX PW	4	4	5	4	5	5	5	5	5	5	4	5	5	5	5	5

d.) Fully Dressed Descriptions with Sequence Diagrams

Manager

Use Case UC#: 12	UpdateInventory
Related Requirements:	REQ-11
Initiating Actors:	Manager, chef
Actor's Goal:	To update inventory in real-time.
Participating Actors:	Database
Preconditions:	Inventory database is set up and contains item that are in stock, but changes need to be made to the quantities.
Postconditions:	Inventory database is updated
Failed End Condition:	
Flow of Events for Main Success Scenario:	
--------- 1.	The inventory database contains current quantity of each item in stock.
--------- 2.	After the user open the inventory system they have the option to add an item, remove an item, change quantity of an item.
--------- 2a.	The user selects the item they would like to update.
2a-1.	The system display the current quantity currently in stock of the selected item and displays options to update quantity.
--------- 2b.	The user chooses to add an item.
2b-1.	The system asks for item information such as name, type, available quantity and low-quantity threshold.
--------- 2c.	The user chooses an item and clicks remove item.
2c-1.	The system deletes the item.
--------- 3.	The system updates the inventory database.
--------- 4.	Once a customer menu order is delivered the system automatically updates the inventory by subtracting one from the quantity of each ingredient.
--------- 5.	Once the manager places and order for food supplies the system will update the inventory database of each item ordered.
Flow of Events for Extension(Alternate Scenarios):	
When the system fails to contact the inventory database or is unable to make the desired changes it will display an error message. When a menu order is placed and the quantity for an item in stock reaches a certain threshold a notification is sent to the manager.	


Manager/Chef
<<initiating actor>>
:System
:Inventory Database
User opens Inventory system
system displays a page with all current inventory and quantities. System also display buttons to add item, remove item, and update quantity.
Depending on the option the user chooses the system will update the Inventory database accordingly
The system updates the database according the the user's choices.
The system will display an error message to the user if the system cannot update the inventory database.

Customer Sign-In/Busboy

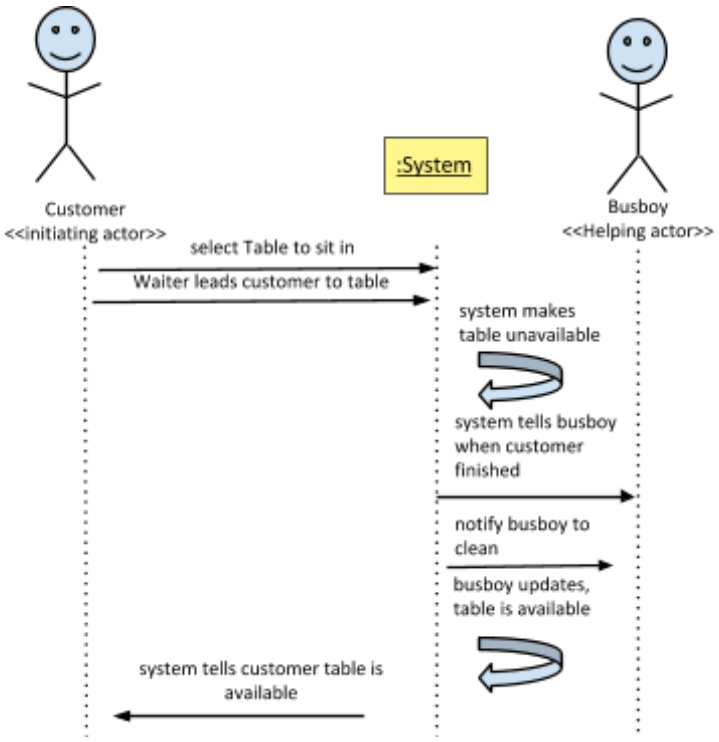
Use Case UC#:	SelectTable
Related Requirements:	Select Table, Update Table
Initiating Actors:	Customer, busboy
Actor's Goal:	To select and update the tables
Participating Actors:	Customer, Waiters, busboys
Preconditions:	Table is ready to be selected
Postconditions:	Table is ready to be cleaned
Failed End Condition:	

Flow of Events for Main Success Scenario:

1. Customer chooses table to sit in
2. Waiter leads customer to sit there and table becomes unavailable
3. Customer finishes eating and leaves
4. Busboy cleans the table to prepare for next customer
5. Table becomes clean and busboy notifies system that table is ready to be used again

Flow of Events for Extension(Alternate Scenarios):

Customer selects unavailable table. System notifies customer that they cant sit there. Customer is forced to wait or select another table.



Chef

Use Case UC#:	**OrderDone**
Related Requirements:	REQ6
Initiating Actors:	**Chef**
Actor's Goal:	To notify the waiter that food is ready.
Participating Actors:	**Waiter**
Preconditions:	Food is ready.
Postconditions:	Food is no longer listed as orders that still need to be cook.

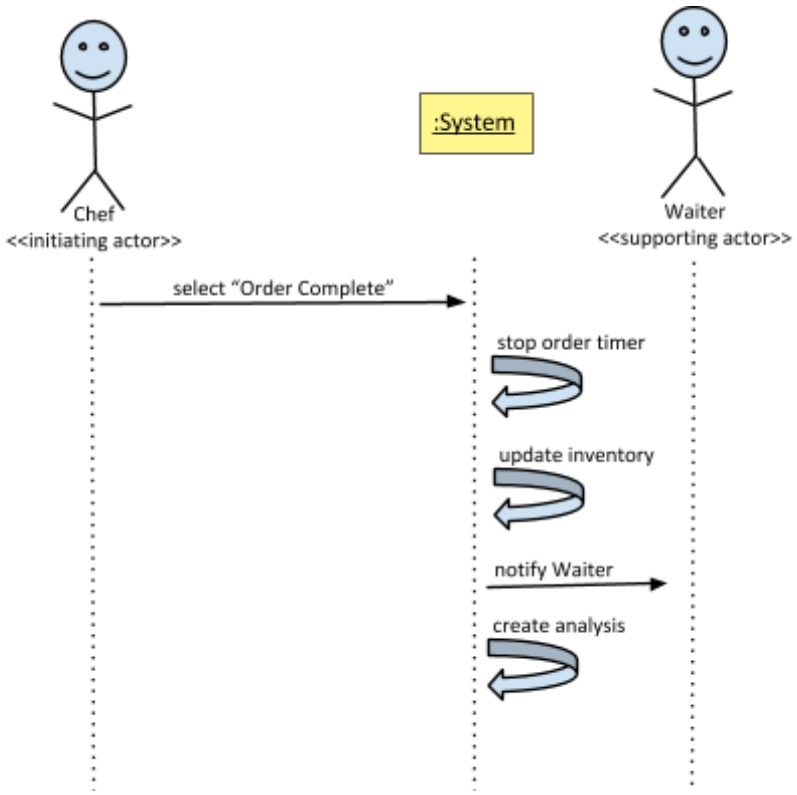
Failed End Condition:

Flow of Events for Main Success Scenario:

1. **Chef** selects that an order is ready.
2. (a) **System** stops timer for that order.
   (b) **System** updates inventory based on order.
   (c) **System** notifies waiter that food is ready.
   (d) **System** creates analysis for the order.
5. **Waiter** receives notification on the ready order.

Flow of Events for Extension(Alternate Scenarios):

1. Chef selects OrderDone for wrong order
   a) Chef must notify immediately of mistake to Waiter
   b) Chef must then fix the mistake on his tablet
2. Chef selects OrderDone prematurely
   a) Chef must notify immediately of mistake to Waiter



Waiter

Use Case UC#:	SendOrder
Related Requirements:	REQ-17
Initiating Actors:	Waiter
Actor's Goal:	To notify the chef to make the food.
Participating Actors:	**Chef**
Preconditions:	Food has to be on menu.
Successful End Conditions:	Order has been sent to Chef's PC
Failed End Condition:	Order did not get sent to Chef's PC and waiter is notified of failure.

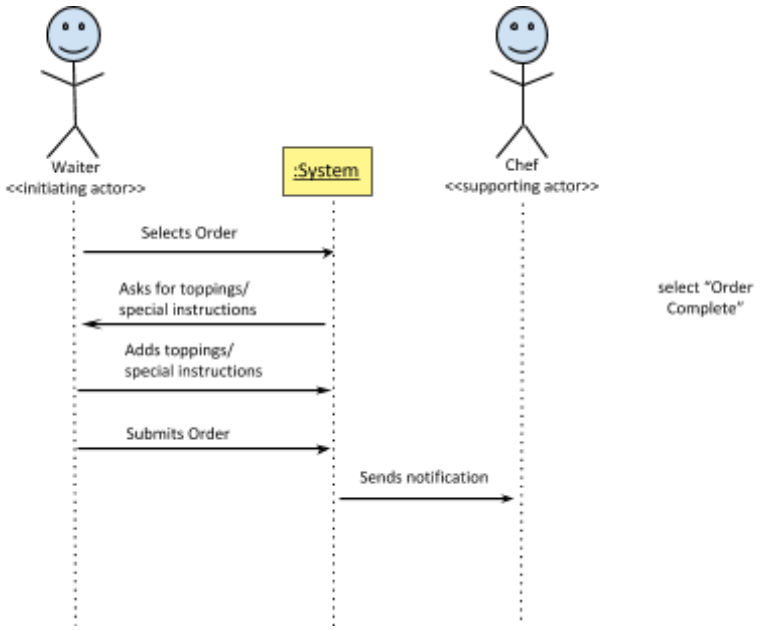
Flow of Events for Main Success Scenario:

1. **Waiter** selects what customer wants to order.
2. **System** asks for any extra toppings and/or special instruction.
3. **Waiter** adds extra toppings and/or special instruction.
4. **Waiter** submits order.
5. **System** sends notification of order to **Chef.**

Flow of Events for Extension(Alternate Scenarios):

1a. **Waiter** selects what customer wants to order, but is no longer in **System**.

1. Waiter requests customer to make a different selection.



5) Effort Estimation

	Use Case Weight
UC - 1	15
UC - 2	10
UC - 3	5
UC - 4	10
UC - 5	10
UC - 6	15
UC - 7	15

UC - 8	10
UC - 9	15
UC - 10	5
UC - 11	10
UC - 12	15
UC - 13	5
UC - 14	10
UC - 15	10
UC - 16	10
UC - 17	5
UC - 18	5
UC - 19	10
UC - 20	5
UUCP	195

TCF	Technical Complexity Factor Weight	TCF perceived complexity	Complexity Factor
1	2	5	10
2	1	1	1
3	1	5	5
4	1	2	2
5	1	4	4
6	0.5	3	1.5
7	0.5	3	1.5

8	2	4	8
9	1	3	3
10	1	2	2
11	1	1	1
12	1	2	2
13	1	1	1
		TCF total Points	42
		TCF	1.02

ECF	**Environmental Complexity Factor Weight**	**ECF perceived impact**	**CF**
1	**1.5**	**4**	**6**
2	**0.5**	**3**	**1.5**
3	**1**	**3**	**3**
4	**0.5**	**2**	**1**
5	**1**	**3**	**3**
6	**2**	**4**	**8**
7	**-1**	**1**	**-1**
8	**-1**	**3**	**-3**
		ECF total points	**18.5**
		ECF	**0.845**

Adjusted UCP	196.865
PF	28
Duration	5512.2

5.2 User Effort Estimation:

Scenario 1:

Customer Sign in PC

1. Customer wanting to select a table to sit in. (2 taps)
A. Customer selects the table they want
B. Customer taps choose table.

2. Busboy updating the table. (2 taps)
A. Busboy selects the table
B. Busboy taps table is available

Scenario 2:

Manager Tablet

1. Manager ordering supplies (5 taps)
A. Manager selects inventory section
B. Manager selects item
C. Manager selects order
D. Manager picks quantity
E. Manager confirms order

2. Manager deleting an item (4 taps)
A. Manager selects inventory section
B. Manager selects an item
C. Manager selects delete
D. Manager confirms the deletion

3. Manager adding an item (4 taps)
A. Manager selects inventory section

B. Manager selects add item
C. Manager types in the item
D. Manager confirms the addition

4. Manager adding an employee (multiple taps)
A. Manager selects employee list
B. Manager selects add employee
C. Manager fills out information about employee
D. Manager confirms the addition

5. Manager deleting an employee (3 taps)
A. Manager selects employee list
B. Manager clicks on employee that he/she wants to delete
C. manager confirms the deletion

Scenario 3:

Waiter Tablet

1. Waiter splitting the bill (3 taps)
A. Waiter selects table(if not already on the current table)
B. Waiter selects split bill
C. Waiter selects how many people to split bill for

2. Waiter sending the order (multiple taps)
A. Waiter selects table(if not already on the current table)
B. Waiter selects food/drink items depending on what the customer asks
C. Waiter adds/removes toppings from item
D. Waiter confirms the order

3. Waiter selects table where customer is sitting (1 tap)
A. Waiter selects table

4. Waiter selects receipt (4 taps)
A. Waiter selects table(if not already on the current table)
B. Waiter selects pay and charges all customers on respective table
C. Waiter gives receipt option in form of paper or email
D. Waiter confirms the option

5. Waiter cancels current order (3 taps)

A. Waiter selects table(if not already on the current table)

B. Waiter selects Void

C. Waiter selects Yes on 'Are you sure?' pop up

6. Waiter quick search (food/drink)items (2 taps)

A. Waiter selects quick search and can type in any item

B. Item will appear in menu section where he can select the item

7. Waiter logs out/clocks out

A. Waiter selects option

B. waiter can choose whether he wants to clock out or log out

Scenario 4:

Any Employee Tablet (hitting the 'home' or 'back key)

1. Employee clocks in the system (2 tap)

A. Employee clicks clock in

B. Employee enters his/her own specific code

C. Employee presses done

2. Employee clocks out of the system (1 tap)

A. Employee clicks clock out and enters code

B. Employee enters his/her own specific code

C. Employee presses done

3. Employee can check table layout (1 tap) if privileged user

A. Employee clicks table icon

Scenario 5:

Chef Tablet

1. Chef selects item that is ready to be queued for cook timer (1 taps)

A. Chef toggles a check mark button next to the item that is ready

2. Chef selects order ready (1 tap)

A. Chef selects Food Ready: Yes? when current meal is ready for the table

3. Chef switches between Order Pending or Order Complete (1 tap)

A. Chef can hit the corresponding tab labeled ‘Order Pending’ or ‘Order Complete

Scenario 6:

Cashier Tablet

1. Customer wants to pay (4-5 taps)

A. Customer selects pay transaction

B. Waiter hits pay button

C. Select cash or credit

D. Select paper or email receipt then submit

E. (if email route) enter email then submit

6) Domain Analysis

a.) Domain Model

i) Concept Definitions

Responsibility	Type	Concept
R-01: Prompt customer to choose what table they would like to sit at	D	Table
R-02: Update status of table (Available, Not Available, Dirty)	D	Table
R-03: Prompt waiter to go to customer's table, give menus	K	Menu
R-04: Prompt waiter to take customer's order and send to Chef's Tablet	K	Order
R-05: Notification telling waiter to check on customer's table	K	customerNotification
R-06: Notification telling waiter to pick up customer's order from kitchen	K	Order
R-07: Update status of customer's order (Ready or Not Ready)	D	Order
R-08: Notify Chef of customer's order	K	Order
R-09: Notification to manager when items are: almost finished, almost expired, finished, and expired	K	Inventory
R-10: Display list of inventory (including quantity)		Inventory
R-11: Change Inventory (add, remove, modify expiration date)	D	Inventory
R-12: Prompt waiter to get payment from customers including if customer wants to split bill	K	Payment
R-13: Prompt waiter to ask customer if they want receipt printed or emailed	K	Payment

R-14: Displays employee information (position, status, wage, contact info, add, remove) as well as status (Working or Off)		EmployeeInfo
R-15: Change employee information (position, status, wage, contact info, add, remove)	D	EmployeeInfo
R-16: Change menu (add, remove, modify)	D	Menu
R-17: Display Statistics		Statistics
R-18: Employees can either log in or out of the system	D	Clock
R-19: System that the waiter uses as the liaison between the customer and the chef	D	WaiterTablet
R-20: System that customers choose their table with	D	CustomerSigninTablet
R-21: System that the manager uses which oversees everything happening in the store	D	ManagerTablet
R-22: System that chef/bartender uses for all food/drink updates	D	ChefTablet
R-23: System that the Busboy uses for all table updates	D	BusBoyTablet
R-24 System that waiter uses, separates drink menu from food menu	D	WaiterTablet

ii) Association Definitions

Concept pair	Association Description	Association Name
CustomerSigninTablet < - > Table	CustomerSigninTablet updates the table based on the Table's availability	Updates the table status
WaiterTablet < - > Table	Table updates the WaiterTablet and prompts waiter to get menus for customer	Gives customers menu

WaiterTablet < - > customerNotification	customerNotification sends notification to WaiterTablet to go to table and place order	Placing order
ChefTablet < - > Order	Order sends a notification of order to ChefTablet	Sends Order
WaiterTablet < -> Order	Order sends a notification to WaiterTablet to pick up food from kitchen	Notifies waiter to pick up food
WaiterTablet < - > customerNotification	customerNotification sends a notification to WaiterTablet to check on table, give bill, and get payment	Pay bill
WaiterTablet < - > Payment	Payment sends notification to WaiterTablet for the waiter to ask the customer if they want a paper receipt or e-receipt using	Print Receipt
BusBoyTablet < - > Payment	Payment updates BusBoyTablet that the table is dirty after customer pays	Marks table as dirty
ManagerTablet < - > Inventory	ManagerTablet uses Inventory to add, remove, and modify expiration date of inventory using Inventory	Updates Inventory
ManagerTablet < - > Inventory	Inventory sends a notification to MangerTablet based on status (almost expired, almost finished, expired, and finished) using inventoryNotification	Notifies manager about inventory
ManagerTablet < - > EmployeeInfo	ManagerTablet uses EmployeeInfo to update employee information (position, status, wage, contact info, add, remove) using updateEmployeeInfo	Updates employee information
WaiterTablet < - > Menu	WaiterTablet can display ingredients and modify customer's orders on Menu	update/display menu
ManagerTablet < - > Statistics	ManagerTablet uses Statistics to display item popularity, wait time, and dine in time	displays statistics of store

ManagerTablet < - > Clock	For use of the ManagerTablet, Manager uses his specific employee number and Clocks in or out	Clock in or out
ChefTablet < - > Clock	For use of the ChefTablet, Chef uses his specific employee number and Clocks in or out	Clock in or out
BusBoyTablet < - > Clock	For use of the BusBoyTablet, Busboy uses his specific employee number and Clocks in or out	Clock in or out
WaiterTablet <- > Clock	For use of the WaiterTablet, Waiter uses his specific employee number and Clocks in or out	Clock in or out

iii) Attribute Definitions

Concept	**Attribute**	**Description**
1. Table	selectTable	Where the customer sits and eats
2. Sit in Tablet	OrderQueue getFood	requests customers send in Subsistence customers consume
3. Employee	OrderQueue	Worker at the restaurant
4. Order	getFood OrderQueue	Items ordered by customer Order is added to the kitchen's order queue
5. Notification	TabletAlert	Alert that is sent from customer's table to waiter's PC
6. Notification	KitchenAlert	Alert that is sent from the kitchen the waiter's PC
7. order	OrderQueue	Order is added to the kitchen's

		order queue
8. inventory	LowStockNotification	The system notifies the manager about low stock items.
9. Inventory	InventoryCheck	The system queries the inventory database to check on an item's stock.
10. Inventory	StockItems	Items in stock are added, removed or updated in the inventory database.
11. Payment	OrderTotal PaymentType	Displays the order total for the waiter Displays accepted payment types
12. Receipt	RecieptType	Provide an option to print a paper receipt or email the receipt to the customer or both.
13. Employee	EmployeeName EmployeeID EmployeeAddress EmployeeTitle	Stores employee name in payroll database Stores employee ID in payroll database Stores employee address in payroll database Allows the manager to update an employee's title such as, waiter, busboy, chef, manager.
14. updateEmployee	EmployeeName EmployeeID EmployeeAddress	Allows manager to update employee name. Allows manager to update employee ID. Allows manager to update employee address.
15. Menu	AddItem	Add an item to the menu

	RemoveItem SuspendItem Ingredients	Remove an item from the menu Temporary suspend item from menu due to an out of stock ingredient. Update an item's ingredients on the menu.
16. Statistics	ItemPopularity WaitTime DineInTime	The number of times an item was ordered. An average wait time for each party to be seated. The average time each party stays at the restaurant.
17. Clock	ClockIn ClockOut	when workers start their shift when workers end their shift

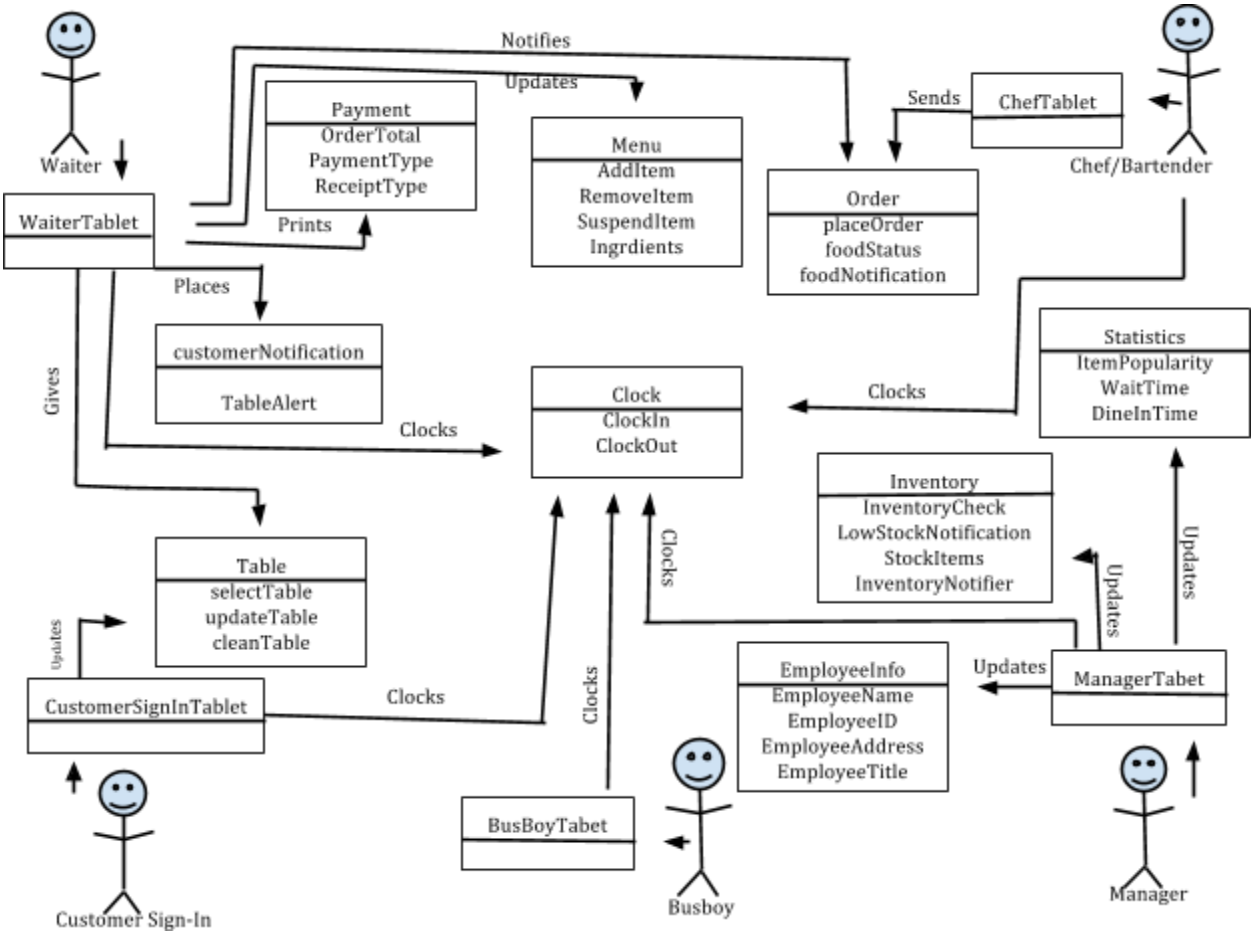
iv) Traceability Matrix

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17
UC-1		X	X														
UC-2	X																
UC-3												X					
UC-4								X									
UC-5						X											
UC-6						X		X									

UC-7						X											
UC-8				X													
UC-9				X													
UC-10												X					
UC-11													X				
UC-12									X	X	X						
UC-13															X		
UC-14																X	
UC-15														X			
UC-16														X			
UC-17																	X
UC -18																	X
UC -19				x		x	x										
UC -20															x		

Our traceability matrix coordinates our domain model with our use cases. It shows which use cases interact with each classes to help while writing the code. The updated traceability matrix includes our updated usecases and how they interact with each class. Most of these were derived from the functions of each class.

v) Domain Model Diagram



b.) System Contracts

Name:	Update Inventory
Responsibilities:	To update inventory in real- time
Use Case:	UC- 12
Exceptions:	none
Preconditions:	Inventory database is already created and already inputted with data. Each item has either 0 or n amount.
Postconditions:	Inventory either increments or decrements the quantity of item(s). Inventory will notify if there is an item out of stock

Name:	Select Table
Responsibilities:	To select and update tables to either sit customers or to be cleaned
Use Case:	UC- 2
Exceptions:	none
Preconditions:	Table must be empty, either before cleaning or after cleaning
Postconditions:	Table is ready to be cleaned or table can now sit customers

Name:	Send Order
Responsibilities:	To notify chef to make the meal for the customer
Use Case:	UC- 9
Exceptions:	None
Preconditions:	All items to be ordered are from the menu. Customer(s) notifies waiter what to order and waiter creates the order(s)
Postconditions:	Chef receives the orders in the order that they were made

Name:	Order Ready
Responsibilities:	To notify the waiter that the meal is ready to be delivered to the customer and to automatically update inventory
Use Case:	UC- 7
Exceptions:	None
Preconditions:	Food is ready to be delivered. Inventory is ready to be updated
Postconditions:	Meal is no longer listed as an order to be cooked. Inventory is updated

c.) Mathematical Models

Chef: Algorithm for Orders

Orders must be cooked based on the time they come in. Customers who order their food first should receive their food before customers who order later. Also orders from a single table should come out at the same time. In order to do this, orders should be placed into a queue where a single tables orders are placed into one location in the queue. These table orders should then be placed into a sub-queue for just the table. New orders get placed at the end of the queue.

```
while ( queue is not empty) {
        if (table order is at the beginning of queue)
                place table orders into a sub-queue based on longest cooking time
        while( table order is not complete) {
                start cooking timer for order in the front of the sub-queue
                start cooking order in the front of the sub-queue
                if (chef is idle)
                        start cooking next order in the sub-queue
                if (order is complete)
                        end cooking timer for the order in the sub-queue
                        remove item from sub-queue and start next order in the sub-queue
        }
        remove table order from the beginning of the queue and start next table order
}
```

“Chef is idle” - The chef is able to start another order because the current order is being cooked and does not require the chef’s attention. This can happen more than once per order so the chef should be able to see the list of the all items in the sub-queue until the order is completed.

Customer sitting: Order to sit customers and clear tables

Customers are sat down based on first come. The customer will be able to select any available table they choose. Once selected, the table will become unavailable to other customers. Customers and waiters will see the real time status of all the tables. Once the customer is finished, the waiter or busboy will update the system to show that the table is available again.

```
print (array{all tables})
        if( table has no customer )
                show table as green and available
        else( table has customer )
                show table as red and unavailable

if (customer chooses table)
        change table to red and unavailable
if (customer leaves table)
        change table to green and available
```

Manager Inventory prediction

For the manager inventory prediction algorithm, we decided to use what Group #1 from Spring 2013 semester started. Algorithms are difficult to formulate and much harder to perfect as it can be an ongoing process even after the system is done. So as a team we are going to try and improve on what last year's team did.

The predictions will be based off of a recursive algorithm that will estimate ingredient usage for foods. We can say that 'i' will be ingredients for food on any day 'n' which will be based off of estimated previous usage(day n-1,n-2,…,n-7) and weekly usage (week w-1,w-2,w-3,w-4). Going with a four week schedule helps to better predict popular seasons such as fall, winter, spring,

and summer since the popularity of items will fluctuate.

For predictions of daily ingredient usage, every time an order is placed it will tally up the popularity of ingredient ‘i’ until the end of the day ‘n’. Weekly tallies of ingredients will be based on what the manager decided to manually order at the end of each week ‘w.’ These prediction will take some time (about 1-2 months) to begin working properly as the system will need some data to predict off of.

Below is an idea of how the prediction will be calculated based off of last year’s formula that we will be looking to improve. The daily usage will take into account the day you are on over the average of all the other days to give the manager an estimate. The weekly average will be based off of what the manager ordered and will be estimated over month periods.

Day n proportional usage:

$$\frac{U(n-7)}{(U(n-1)\ +\ U(n-2)\ +\ U(n-3)\ +\ U(n-4)\ +\ U(n-5)\ +\ U(n-6)\ +\ U(n-7))}$$

Weekly average :

$$\frac{U(w-1)+U(w-2)+U(w-3)+U(w-4)}{4}$$

Ui(n) =

$$\frac{U(n-7)}{(U(n-1)+U(n-2)+U(n-3)+U(n-4)+U(n-5)+U(n-6)+U(n-7))} * \frac{U(w-1)+U(w-2)+U(w-3)+U(w-4)}{4}$$

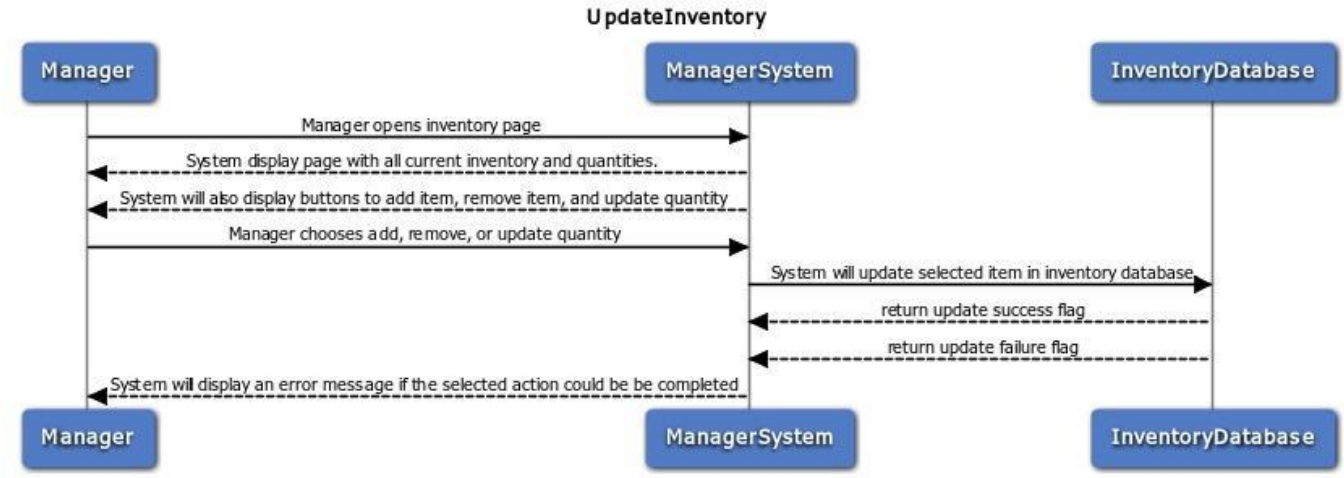
Note* U(week) is the actual weekly usage taken from the inventory system’s data, and NOT a value returned by the recursive function.

Plan of Work:

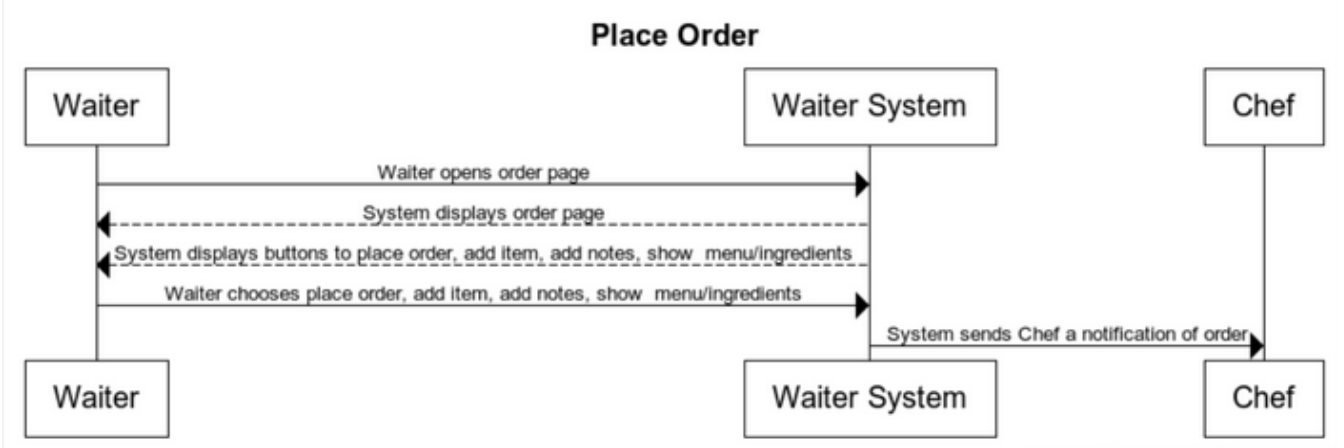
Task	Start Date	Deadline	Estimated time(days)	Responsibility
Part 1 (Interaction Diagrams)		**3/2/2014**		
a. Interaction Diagrams	2/25/2014	3/1/2014	4D	ALL
b. Describe bubble diagram	2/25/2014	3/1/2014	4D	ALL
Part 2 (Class Diagram and System Architecture)		**3/9/2014**		
a.Class Diagram	3/4/2014	3/6/2014	2D	MG
b. Data Types and Operation Signatures	3/4/2014	3/6/2014	2D	AP
c. Traceability Matrix	3/4/2014	3/6/2014	2D	NT
System Architecture				
a. Architectural Styles	3/4/2014	3/6/2014	2D	ALL
b. Identifying Subsystems	3/4/2014	3/6/2014	2D	JC
c.Mapping Subsystems to Hardware	3/4/2014	3/6/2014	2D	AA
d.Persistent Data Storage	3/4/2014	3/6/2014	2D	MG
e.Network Protocol	3/4/2014	3/6/2014	2D	NT,AP
f.Global Control Flow	3/4/2014	3/6/2014	2D	JC
g.Hardware Requirements	3/4/2014	3/6/2014	2D	CY
Part 3		**3/9/2014**		
a.Algorithms	3/5/2014	3/6/2014	1D	ALL
b.Data Structures	3/6/2014	3/7/2014	1D	AA,DU
c.User Interface Design and Implementation	3/6/2014	3/7/2014	1D	DU,CY
d.Design of Tests	3/6/2014	3/7/2014	1D	ALL
f. Merge Project	3/7/2014	3/8/2014	1D	ALL
Full Report# 2	3/11/2014	**3/15/2014**	3D	ALL
First Demo	3/16/2014	**3/28/2014**	10D	ALL
Reflective essay	3/29/2014	**4/1/2014**	1D	ALL
Part 1 (Sections)	3/30/2014	**4/4/2014**	3D	ALL
Second Demo	4/5/2014	**4/7/2014**	2D	ALL
Full Report# 3 (rest of sections)	4/5/2014	**4/8/2014**	2D	ALL

AA= Amgad Armanus
JC= Jake Chou
MG= Mitul Gada
AP= Avni Patel
NT= Nirjan Thayaparan
DU= Diego Urquiza
CY= Christian Youssef
ALL= all team members

7) Interaction Diagrams

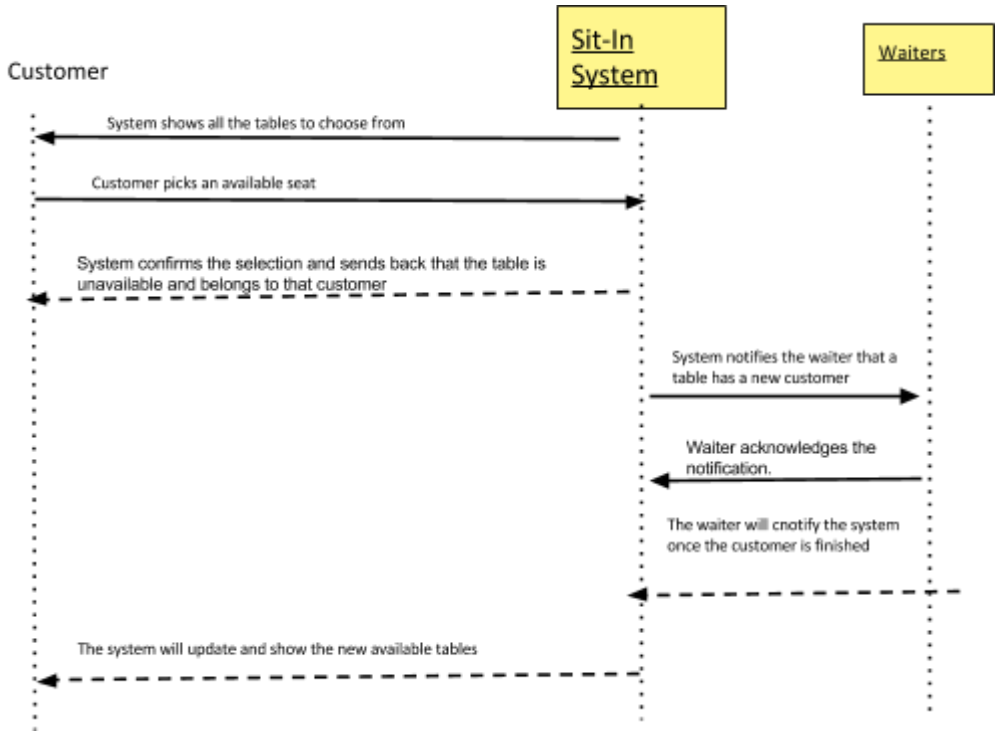


The ManagerSystem interacts with the inventory database and displays the results to the manager screen. In case of a failure to update the database, the ManagerSystem will display an error message to inform the manager.

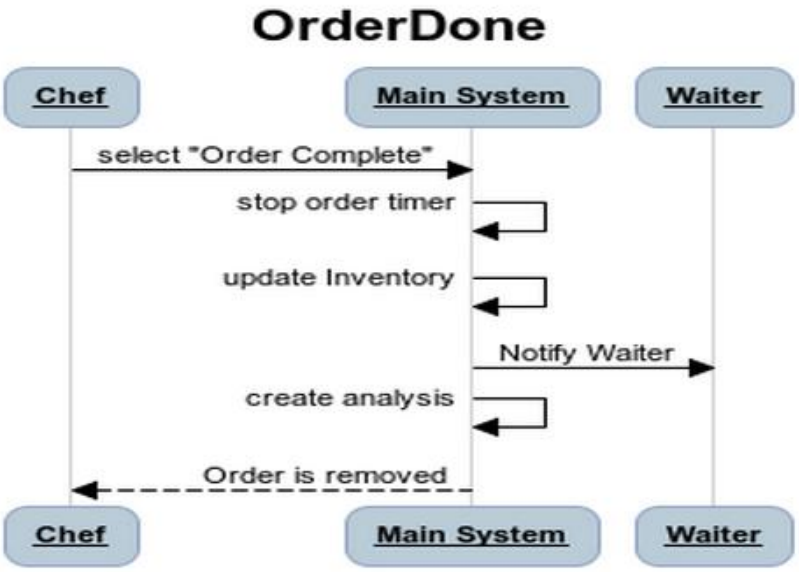


The Waiter System interacts with and sends a notification the Chef's PC. The Waiter Screen allows the Waiter to place an order, add items, add notes, and see the menu/ingredients.

UpdateTable:



The customer sign in PC will be used by both the customers and the waiters. The customers will see the status of all the tables in real time and be able to choose the available ones. The system will then change it so that the next customers will see that the table is unavailable. Once finished, the waiter will update the system so it can send back to the customer PC when an unavailable table will be ready.



The Chef PC will utilize this sequence in order to send a notification through the system to the waiter that an order has been completed and is ready to be served to the customer. The chef will press the "Order Complete" option, which will tell the system to stop the order times for that specific order, update the inventory in response to the order completed, notify the waiter that the meal is ready to be served, and create an analysis based on the order. Through this option, efficiency throughout will be increased because of all the actions that happen at once through the system instead of manually and one by one.

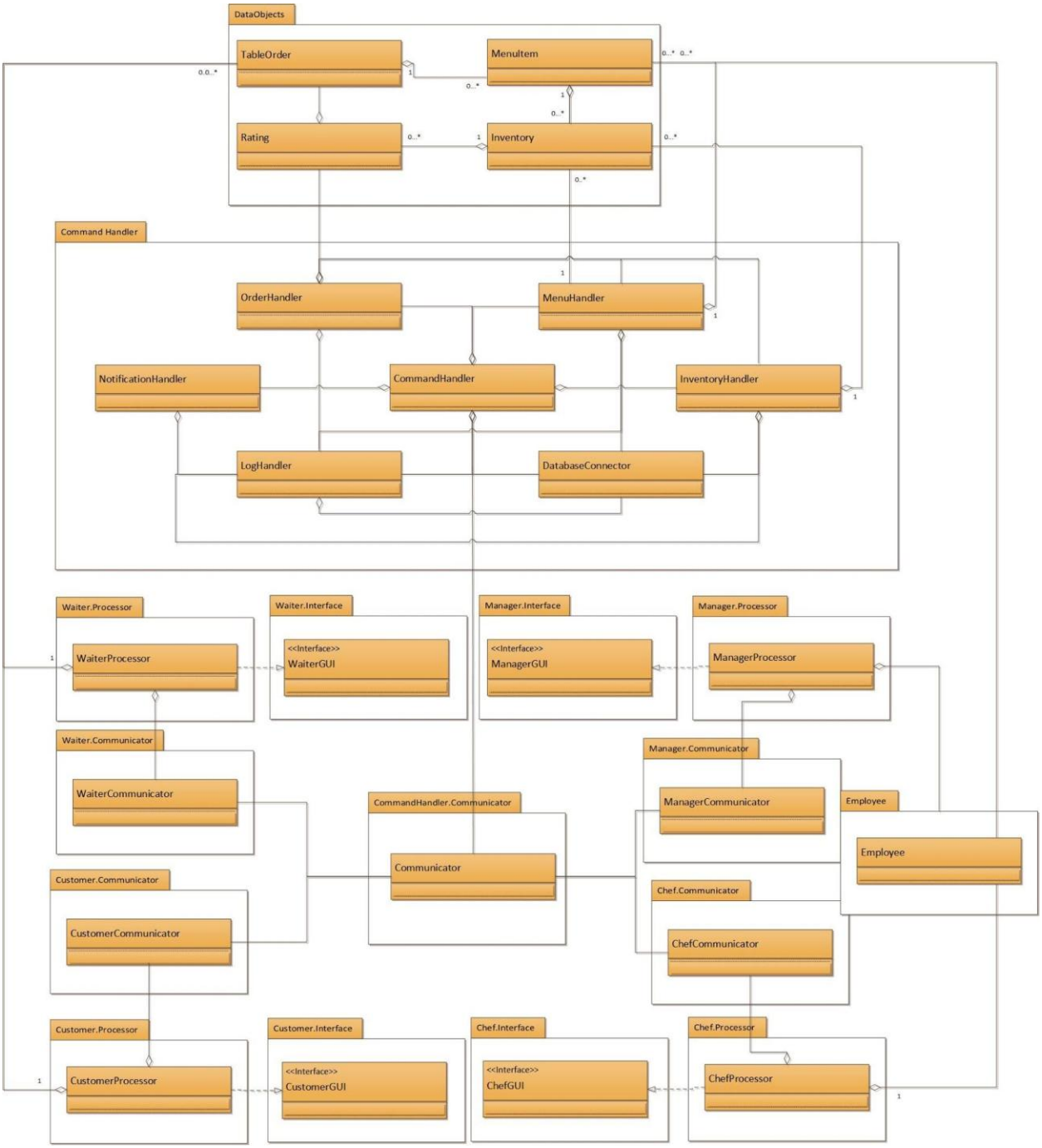
a.) System Design Pattern

A design pattern that we used in our project is the publisher-subscriber pattern. This specific design pattern is used to implement indirect communication between our software objects. In other words, the object cannot or does not want to know the identity of the object whose method it calls. The purpose is to enable reusable components and it can facilitate separation of the business logic of objects. For our project we used this pattern for mainly our system and when it communicates with other employees. When sending commands and signals, our system will be able to incorporate it and handle the source and content to link it to the

appropriate interface. For example, if a manager is updating the inventory or if the chef is sending back completed food orders, the system will separate each command and recognize that the first one being the manager and needing to send the notification for the inventory around while the second one being sending the notification to the waiter signaling that the chef completed the order. This pattern will help keep our system more organized and easier to use. We can send back multiple commands and reuse it and the system will take it accordingly. In our system the one-to-many dependency between objects occurs when the waiter's tablet sends out the order and the other parts of the system get notified and updated automatically.

8) Class Diagram and Interface Specification

a.) Class Diagram

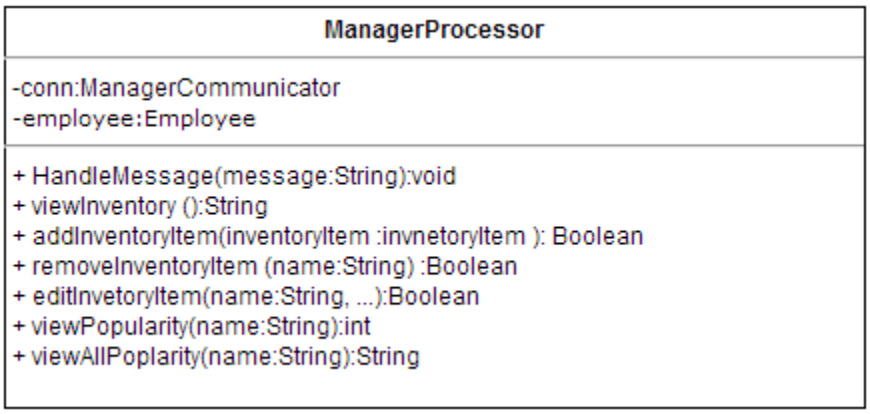


b.) Data Types and Operation Signatures

Note For all of our classes we are going to be using spring 2013 'Auto-Serve' as reference for their classes and we will be optimizing and adding newer functions.

ManagerProcessor

ManagerProcessor is responsible for handling the requests given by the CustomerGUI.



Attributes

-conn:ManagerCommunicator

-employee:Employee

Methods

+HandleMessage(message:String):void

Used to handle requests that are being passed to this function.

+viewInventory():String

Views inventory items that are in the system.

+addInventoryItem(inventoryItem:Inventory):Boolean

Allows the user to add an inventory item to the system.

+editInventory(name:String,…):Boolean

Allows used to edit a selected inventory item.

+viewPopularity(name:String):int

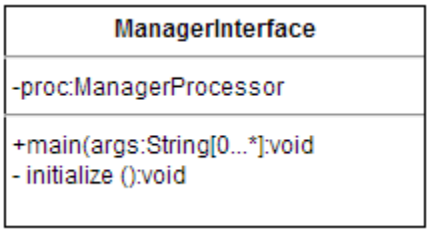
Displays the popularity of the selected menu item.

+viewAllPopularity(name:String):String

Displays popularity of all items on the menu.

ManagerInterface

ManagerGUI is the what the manager interacts with on their display. This is responsible for communication between the manager and the system.



Attributes

-proc:ManagerProcessor

Processes all requests made and calls the appropriate function.

Methods

+main(args:String[0….*]):void

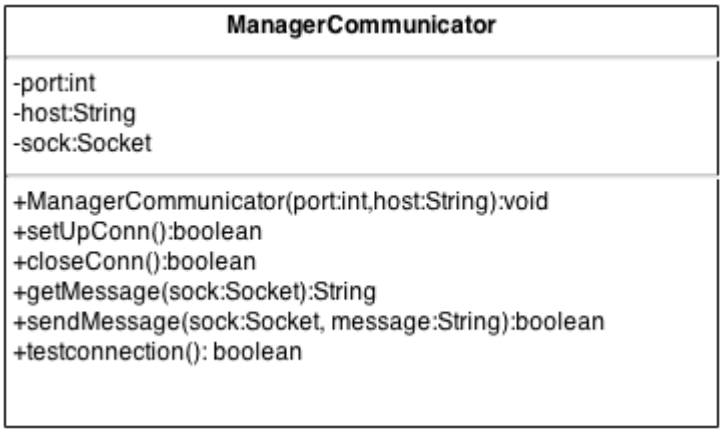
The main function that calls the initialize to prompt the initialization of the GUI.

-initialize():void

Creates the GUI that is displayed for the manager.

ManagerCommunicator

The ManagerCommunicator class will take care of all the connections to the server from the client side. The functions in this class will be responsible for sending requests to the CommandHandler as well as sending information to the ManagerGUI.



Attributes

-port:int

The port number of the client.

-host:String

protocol string that will be string compared in order to make sure it has the name pipe sql protocol

-sock:Socket

The socket of the client used to communicate to the server

Methods

+ManagerCommunicator(port:int,host:String):void

ManagerCommunicator will set up the port number and the host string

+setUpConn():boolean

This function will use port number and string initialized from MangerCommunicator in order to set up the socket connection

+closeConn():boolean

Terminates the socket connection

+getMessage(sock:Socket):String

Gets request coming into the client socket

+sendMessage(sock:Socket,message:String):boolean

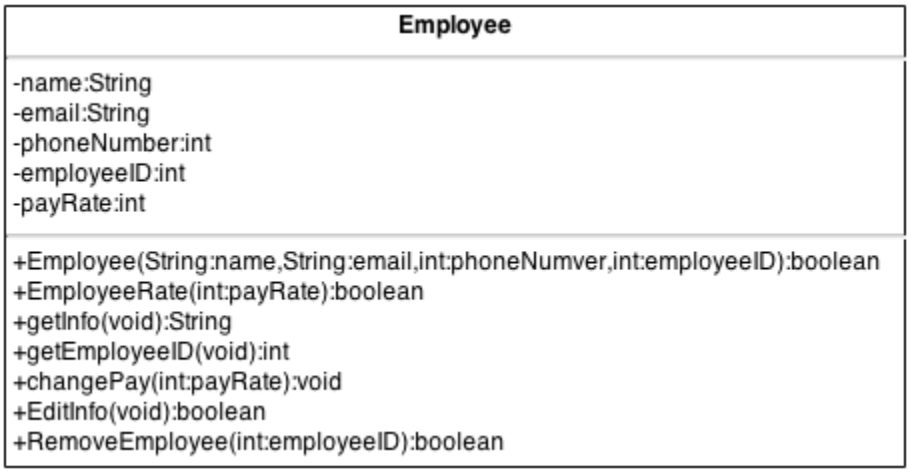
Takes the socket information and sends it as a string which is formated to sql protocols to the server alongside with a message.

+testConnection():boolean

Test the connection to the server. If the value returns false as in failed connection, it will invoke setUpConn

Employee

The Employee class will set up all or remove employees from the system. This class will be friends with the ManagerProcessor class since we want the manager to have the ability to change employee information on the go.



Attributes

-name:String

First and last name of the employee.

-email:String

Employee primary email information.

-phoneNumber:int

Employee primary contact phone number.

-employeeID:int

ID that will be used in order to keep better track of employees.

-payRate:int

Hourly rate of employees

Methods

+Employee(String:name,String:email,int:phoneNumber,int:employeeID):boolean

This function will set up a new employee into the system based off of there name,email, and phone number. After successfully entering a valid new employee the system will display employee information as will as generated employee ID.

+EmployeeRate(int:payRate,int:employeeID):boolean

Manager can set initial pay rate for the employee.

+getInfo(int:employeeID):String

Pull up all of employee's information.

+getEmployeeID(void):int

Bring up a list of all current employees organized by ID order.

+changePay(int:payRate,int:employeeID):void

Change the hourly rate of employees payroll.

+EditInfo(int:employeeID):boolean

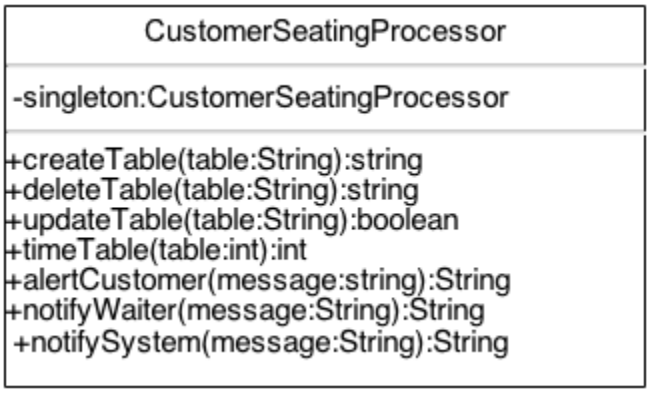
Allows manager to edit basic employee information (name,email, or phonenumber).

+RemoveEmployee(int:employeeID):boolean

Removes employee completely from the system (except payroll information).

CustomerSeatingProcessor

The Customer Seating Processor's task organizes all of the tables and gives the status of each one to the customer and waiters. The customers will be able to see all of the tables and the status of each and choose the available ones. These requests will be sent to the waiters to notify them to bring the customer to the selected table. The system will then update for the next customer.



Attributes

-singleton:CustomerSeatingProcessor

The function only calls to itself

Methods

+createTable(table:String):string

This method will create a new string and insert a new table.

+deleteTable(table:String):string

This method will delete an existing string and delete a table

+updateTable(table:String):boolean

This method will change the status of a table. There are only 2 statuses, available or unavailable.

+timeTable(table:int):int

This method will show the current time of the table of how long the customer was at

that table to show the customers and waiters and estimate time of how much longer they will take.

+alertCustomer(message:string):String

This alert will show the customer when a table has become available again if a customer at the table has just left or change the table they just selected to unavailable for the next customer to see.

+notifyWaiter(message:String):String

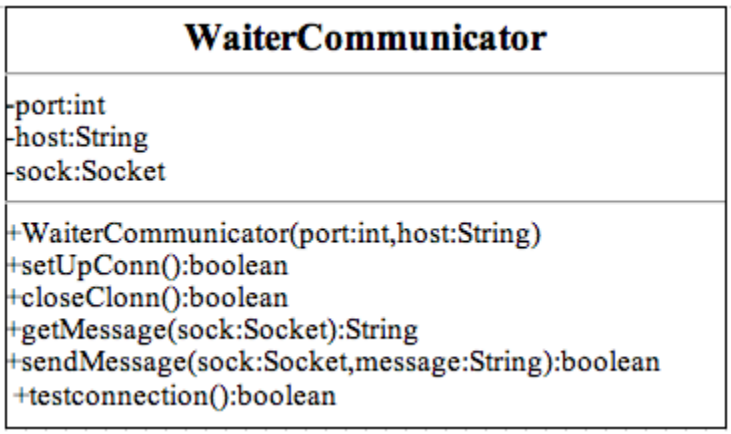
This alert will notify the waiter that a customer has just chosen a table and is ready to be seated.

+notifySystem(message:String):String

The system will update accordingly to give real time updates and statuses of every table

Waiter.Communicator

The WaiterCommunicator is responsible for all communications between the CommandHandler and the WaiterGUI. This class is actively listening for requests from the CommandHandler and is responsible for all changes made to the WaiterGUI.



Attributes

-port:int

The port through which the class listens through.

-host:String

The Hostname of the local computer

-sock:Socket

The socket that sends and receives requests on.

Methods

+WaiterCommunicator(port:int,host:String)

The constructor.

+setUpConn():boolean

The method used to setup the connection for the socket.

+closeConn():Boolean

The method used to close the connection on the socket.

+getMessage(sock:Socket):String

The method used to receive a request on a connected socket

+sendMessage(sock:Scoket,message:String):Boolean

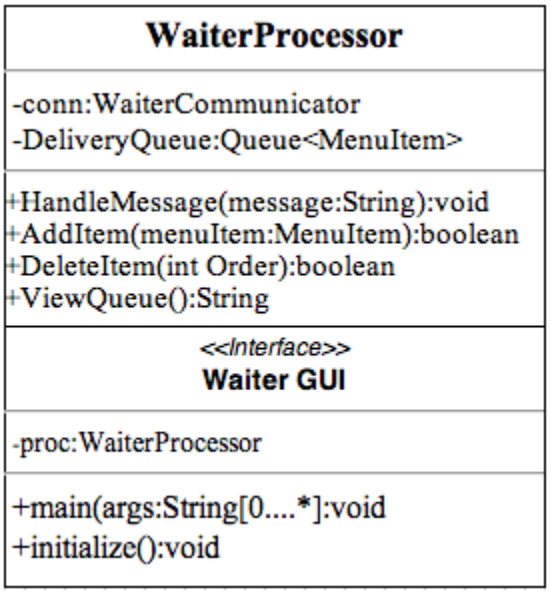
The method used to send a message from the socket.

+testConn():Socket

The method used to test the socket's connection.

Waiter.Processor

The WaiterProcessor is responsible for the maintaining the OrderQueue locally while also handing the requests given by the WaiterGUI.



Attributes

-conn:WaiterCommunicator

This is the socket connection used to communicate with other components in the system.

-DeliveryQueue:Queue<MenuItem>

This is the container for the queue that holds the items that were ordered and are ready to be

delivered to customers.

Methods

+HandleMessage(message:String):void

This method handles all the messages the waiter sends to the Chef.

+AddItem(int Order):Boolean

This method adds an order to the Deliveryqueue and gives an order number.

+DeleteItem(int Order):Boolean

This method removes an order form the Deliveryqueue based on the order number.

+ViewQueue():ArrayList<MenuItem>

This methods returns the current delivery queue.

WaiterGUI

The WaiterGUI is the front end of the system. It is the interface that the Waiter uses and is the liaison between the Waiter and the system. This class will be using the WaiterProcessor to aid in processing the requests by the Waiter.

Attributes

-proc:WaiterProcessor

This object is used to process all the requests.

Methods

+main(args:String[0….*]):void

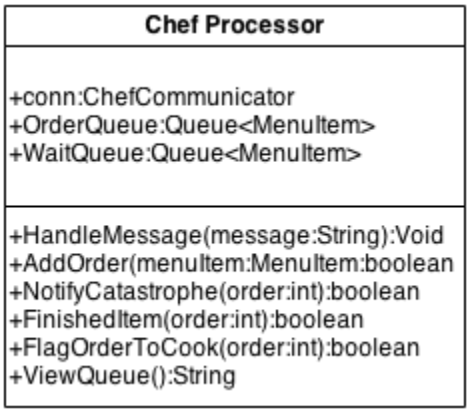
This method is used to initialize the GUI.

-intialize():void

This method creates the GUI

Chef.Processor

The ChefProcessor is responsible for the maintaining the OrderQueue and BeingCookedQueue locally while also handling the requests given by the chefGUI. An example of a request is: Flagging order done which has to send the order to the WaiterGUI.



Attributes

+conn:ChefCommunicator

This object is used to send and receive requests.

+OrderQueue:Queue<MenuItem>

This object holds the menuItems on the OrderQueue

+WaitQueue:Queue<MenuItem>

This object holds the menuItem on the Orders to cook.

Methods

+HandleMessage(message:String):void

This method handles any message passed to the chef. 93

+AddOrder(menuItem:MenuItem):boolean

This method add a menu item to the chef's ready queue .

+DeleteItem(order:int):boolean

This method removes an item from the chef's ready queue and puts it on the waiter queue.

+NotifyCatastrophe(order:int):Boolean

This method will notify the controller of a catastrophe and to halt the current queue.

+FinishedItem(order:int):Boolean

This method will take an item from the wait queue and flag it as done. This will send a message to the controller to forward the item to the waiter to be delivered.

+FlagOrderToCook(order:int):Boolean

This method will flag an order to be cooked which will move it to the wait

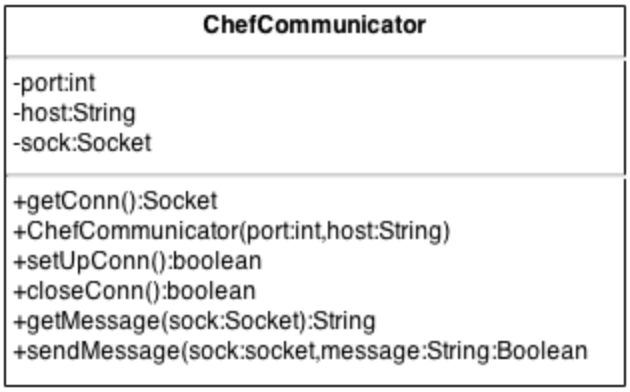
queue.

+ViewQueue():String

This method will return the current queue for the chef.

Chef.Communicator

The ChefCommunicator is responsible for sending and receiving any communication between the ChefGUI and the CommandHandler. This class is actively listening for requests from the CommandHandler and can be responsible for any changes made in the ChefGUI.



Attributes

-port:int

The port through which the class is going to listen through.

-host:String

The Hostname of the local computer

-sock:Socket

The socket that going to be used to send and receive requests on.

Methods

+ChefCommunicator(port:int,host:String)

The constructor used to initialize. 94

+setUpConn():boolean

The method used to setup the connection on the socket.

+getConn():Socket

The method used to listen and return any incoming information

+closeConn():Boolean

The method used to close the connection on the socket.

+getMessage(sock:Socket):String

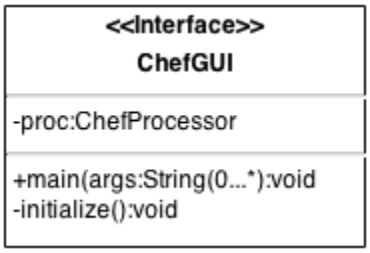
The method used to receive a request on a connected socket

+sendMessage(sock:Scoket,message:String):Boolean

The method used to send a message on the socket.

Chef.Interface

The ChefGUI is the front end of the system and is responsible for the interface between the Chef and the system. This class will be using the ChefProcessor to aid in processing the requests by the chef.



Attributes

-proc:ChefProcessor

This object is used to process all the requests.

Methods

+main(args:String[0….*]):void

This method is used to initialize the GUI.

-intialize():void

This method creates the GUI.

c.) Traceability Matrix

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Concept #	TableOrder	MenuItem	Inventory	OrderHandler	MenuHandler	NotificationHandler	CommandHandler	InventoryHandler	LogHandler	DatabaseConnector	WaiterProcessor	WaiterGUI	WaiterCommunicator	ManagerProcessor	ManagerGUI	Employee	ManagerCommunicator	CustomerProcessor	CustomerGUI	CustomerCommunicator	ChefProcessor	ChefGUI	ChefCommunicator	Communicator
1	x										x					x		x						x
2	x					x	x				x		x			x	x	x	x	x				x
3		x			x						x	x	x		x	x			x	x				
4						x															x	x		
5						x															x	x		
6						x															x	x	x	x
7						x															x	x	x	x
8		x		x	x						x	x	x											
9				x							x	x	x											
10														x	x	x	x							x
11							x		x						x	x								
12			x					x																
13		x			x									x	x	x								x
14									x	x														
15														x	x	x								x
16														x	x	x								x
17									x	x														
18									x	x														

9) System Architecture and System Design

a.) Architectural Styles

An architectural style is a set of principles that provides an abstract framework for a set of systems. The main purpose is to improve partitioning and promote design to our problems by providing detailed solutions. We will be focusing on multiple architectural styles that correspond to our project. These topics will include communication, deployment, domain and structure.

Communication: Service-Oriented Architectural Style
Service-oriented architecture enables application functionality to be provided as a set of services and creation of applications that make use of software services. they basically focus on providing a message based interaction with an application through interfaces that are applicable. Our project will have a main home system that will communicate and interact with all the PCs and tablets. However, these tablets will be autonomous since they all will have a different task and job that will notify the system to update the overall components. Services are also distributable since these portable PCs can be carried around throughout the restaurant and used whenever. Services will also share contracts when communicating and not internal classes.

Deployment: Client/Server

The client/server architecture will distribute the system that involves a separate client and server system with the overall connecting network. It describes the relationship between them whereas the client will initiate requests and wait for the reply from the system. We will use our system as the client and the tables as the server. The servers will be able to request information from the system, and the system will distribute accordingly. Our communication protocols will also have a common language when deploying and we are looking at either C or C++.

Domain: Event Driven Design

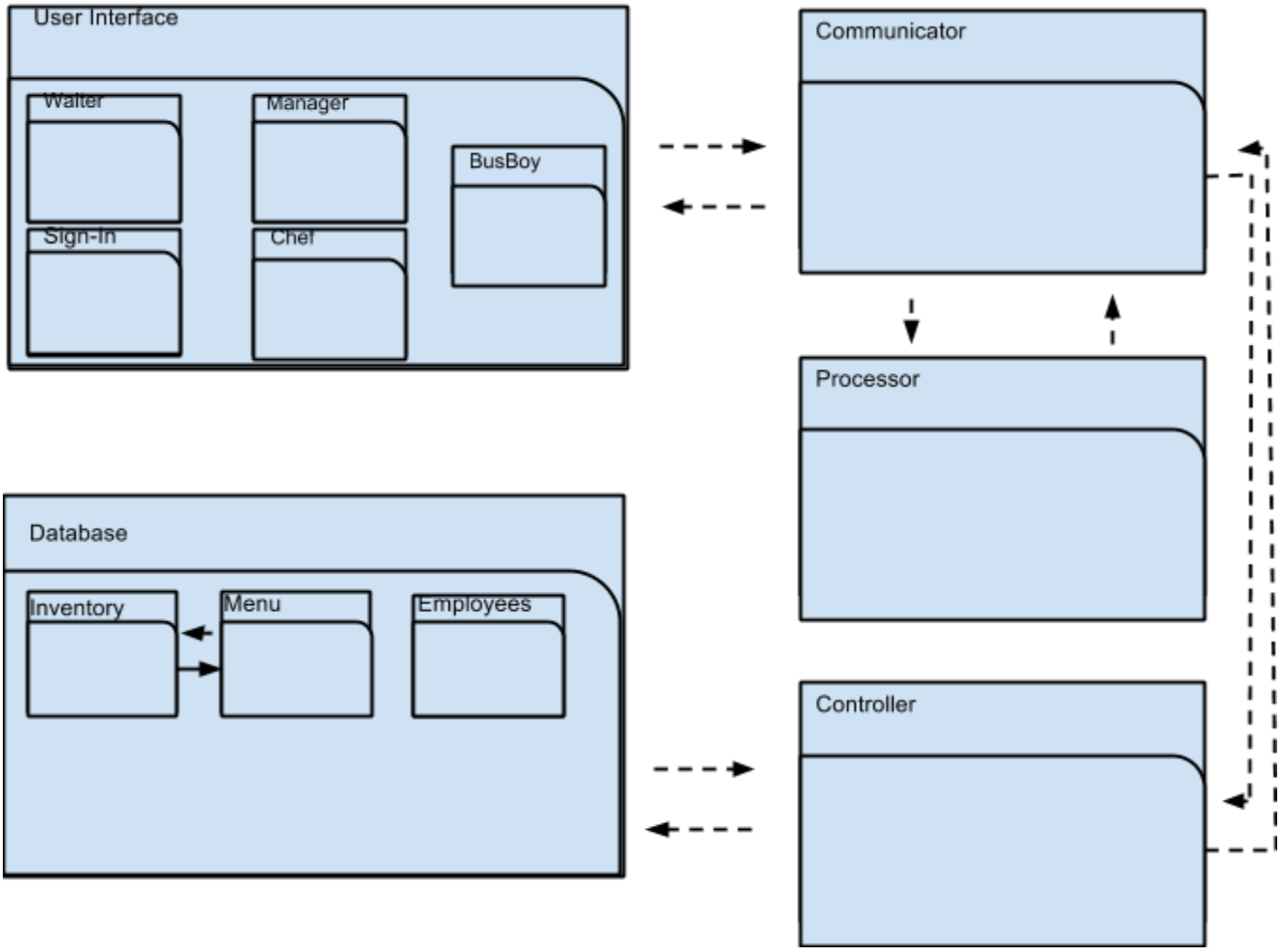
Event Driven Design is a software pattern that helps promote the production, detection, consumption, and reaction to all events. An event can be considered a significant change of an object or state of it embedded in the system. Our system will have notifications that will move the events along. For example, our chef will have notifications that will send out to the waiters when the food is ready. The waiters will be notified once the customers are seated at a certain table. The manager will be notified if an item in the inventory is about to run out or expire. All these main events and more will be crucial for the restaurant business and the event design will be sent out by the system by sending notifications.

Structure: Object-Oriented Architectural Style

Object-oriented architecture is a design pattern that divides all the responsibilities of an application or system into individual reusable objects that still maintains the data. These independent objects will communicate through our interfaces by calling methods or accessing properties of other objects by sending and receiving messages. Our structure is composed of many different tablets and PCS. Referring to abstraction, the manager tablet will have functions like Get() and Update() with the inventory and items. Encapsulation makes it easy for the tablets to delete or update items to have the newest possible update. Inheritance allows objects to be functional throughout since a single update on any machine will influence other machines by updating the system instead of one individual tablet.

b.) Identifying Subsystems

Package Diagram



Based on the packaging diagram above, the packages are divided into 5 main parts. The user interface contains the 5 different interface: Chef, Waiter, Manager, Busboy and Customer Sign-In. These are the 5 different interfaces that an employee or customer may access. There is a separate package for database. This contains all the information on the menu and inventory as well as information on the employees. The database may also contain other folders. The communicator package is the accessor of information. It sends interactions from the interface to the controller. The controller delegates what the communicator can do. It accesses the database and uses the information to interact with the communicator and the processor.

c.) Mapping Subsystems to Hardware

From our subsystem diagram above, our subsystem will be mapped to our hardware in a very simple way. The database will be a MySQL server that will run either on a portable or a centralized computer. The controllers will also be on this computer as well in order to bridge the gap between the user interfaces and the database. Both the database and controller will serve the back end of the process. But each user interface which will serve as the front end, such as the chef, the waiter,or the manager, will be set on individual computer or separate tablets for each user. Each of these interfaces will incorporate the necessary corresponding processor and communicator.

d.) Persistent Data Storage

For our database, all the data is permanently saved and updated frequently. So, persistent data storage is essential in the transactions done in our system on a daily basis. Some transactions are customer's orders, menu changes, and inventory changes. For our system, all the transactions depend on the table selected. Each table will hold different transactions, which go into our database. This is all done using SQL, which will maintain our database and everything does coincide with each other in some way. Our database works in a way that the applications in our system acts as the customer to the sever. This does not interact with the database directly, but simply just asks the server to perform an operation. With this built in server, our data is maintained and allows alteration of features.

e.) Network Protocol

For our system, we are going to have an application which can run on a computer(for prototyping) that will be communicating back and forth with a server. We decided to go with a Microsoft SQL server since the backend will be on C++ and the front end of the computer application will also be in C++. This way we can maximize performance and compatibility by communicating locally through a network. Since we will not need to go through the internet, we can avoid TCP/IP and go straight for a Named Pipe. Named Pipes have an advantage over TCP/IP because they are usually faster for sending information, and have more free network stack resources. Named pipes are easily configured through the SQL server and the client through options and code.

Sample code to set up server and database:

```
QCoreApplication a(argc, argv);

QString servername = "LOCALHOST\\SQLEXPRESS";    //server name
QSTRING dbname = "test";              //database name
QSqlDatabase db = QSqlDatabase::addDatabase("QODBC");   //database driver
db.setConnectionOptions();             //set connections

QString dsn = QString("DRIVER={SQL Native Client};SERVER
=%1;DATABASE=%2;Trusted_Connection=Yes;").arg(servername).arg(dbname);
 //connection string

db.setDatabaseName(dsn);  //database name to connection string
```

f.) Global Control Flow

Execution Orderness

"Why Wait" is procedure-driven. Everything executes in the following linear fashion:
When the customer first comes in, they will choose a table which is empty using the Customer Sign in PC. Once they are seated the waiter will come to get the customer's order. The waiter will then place the order. The Chef will receive the order on the Chef's PC and begin preparing the food. The inventory will get adjusted as needed. Once the food is ready the Chef will use the Chef's PC to notify the Waiter PC. The Waiter will come get the food and deliver it to the customer's table. Once the customer is finished eating the waiter will get the payment from the customer on the Waiter's PC. Once the payment goes through, tables will be marked as dirty and the Busboy PC will call the busboy to come clean.

Time Dependency

"Why Wait" has an event-response time for the inventory alerts, but the rest of the system is a real-time system that is periodic. The procedure shown in the Execution Orderness section is what is periodic. The customers choosing a table and ordering their food, the chef preparing their food, the waiter delivering the customer's food and taking the payment from the customer, and the BusBoy cleaning the table is all periodic. All of the previously mentioned processes are time dependent as the time that each actor takes to do their function will be taken into consideration for our system processes. Another factor where real-time plays a role

is when the system estimates the amount of time the customer will have to wait for their order to be ready.

The inventory functions are also time dependent because when an ingredient comes close to expiration date the system will send out a notification to the manager. Also, if the inventory is low, a notification gets sent to the manager.

Concurrency

"Why Wait" will use multi-threading because there are multiple subsystems running independently of each other. All interactions between the subsystems are controlled through a central server. Multiple customers will be taken care of at once so multiple orders will be placed at the same time. The event where multiple customers are served will be taken care of by running the different threads through the order queue. Another situation where multi-threading would be used is when the manager is checking the inventory and updating it. The manager would have to spawn one thread for checking the inventory and another thread to handle the update request in the database. The synchronization of these threads are not necessary because they would not be working together.

Hardware Requirements

The system will need a server that would store the databases and allow communication between different subsystems. The system will also need a tablet for each of the waiter, host, manager, and chef.

Server:

Hardware	**Minimum Requirements**
Processor	Intel Xeon E5 2.3 GHz
Hard Drive	500 GB
RAM	8 GB
Network Card	10/100/1000Mbps

Tablets:

Hardware	**Minimum Requirements**
Processor	Intel i5 E5 2.4 GHz

Hard Drive	32GB
RAM	2 GB
Display	Multi-touch screen - 1024 x 760 Resolution
Network Card	802.11b/g/n Wireless LAN

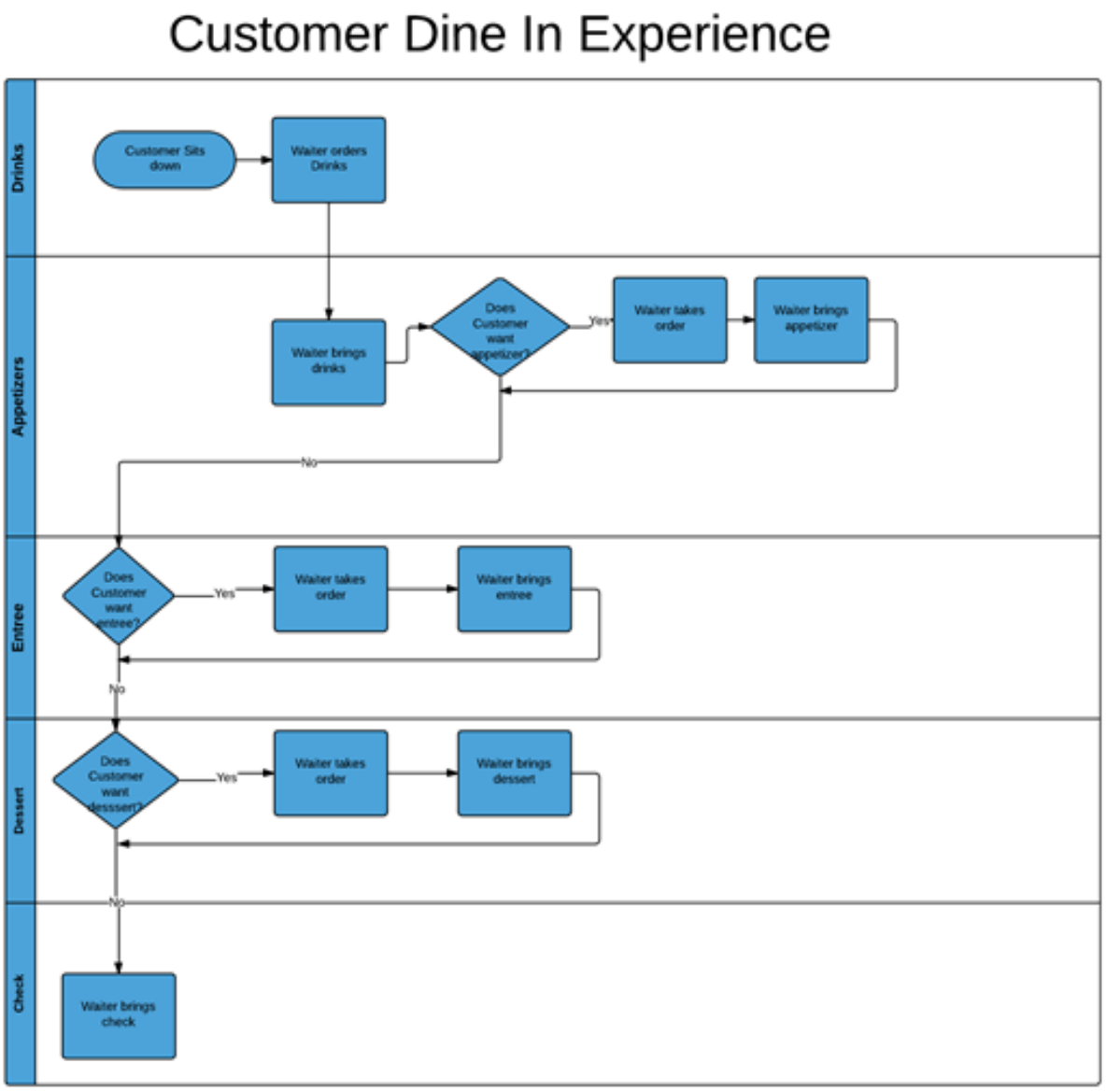
10) Algorithms and Data Structures

a.) Algorithms

Waiter:

If (timeSinceLastVisit==8)
return notification;

The way this algorithm would work is if the waiter has not checked on the table in under 8min, the WaiterPC tablet will notify the waiter to check on the table. When the customer is seated, the waiter will greet the customers after the 1st minute; bring drinks after the 3rd minute; appetizers by the 8th minute if ordered; entree by the 20th minute but if no appetizers are ordered, then by 15th minute; and desserts 15 minutes after the entree; If no dessert then check within 15 after entree, otherwise 5 minutes after dessert is served



Manager:

In the manager system there are some simple algorithms being used, one of which updates the inventory as items are used or restocked. Therefore, the inventory system will increment the quantity by one or decrement it by one automatically. Another algorithm is the not that queries the database, this is used the the database management system and this communicates with the database tables themselves. Some of these algorithms include, adding an new item to the database, removing an item from the database, and updating an item's status. These database tables include, user account information, food menu information, table size and occupancy, and inventory items.

Chef:
The chef uses an algorithm for cooking orders in a queue. There is a queue within a queue as described in the mathematical section. The chef also uses an algorithm when an order is cooked to decrement the supplies as they are used. When an order is cooked, the ingredients are automatically subtracted and the database is updated.

Customer:
The customer will use a given algorithm for the seating process. The tables will have a toggle option that will show whether it is available or unavailable. The customer will pick the a table that they will like but it has to be unavailable. There will also a timer function that goes along with it that keeps going up and counting when a table is selected. It will then reset once the customer leaves.

b.) Data Structures

DIfferent data structures are needed for each of the user. Array lists are used for storing data such as the items in stock or the employee information. Array lists make it easy to store data. Items can be removed and added to the list. Queues are used for prioritizing primarily by the chef and waiter. Queues are essential to make sure that orders that come first are served so that customers have shorter and more equal waiting times.

Manager
The manager would use array lists. This is because of flexibility. In an array list. Items can be deleted and added to any part of the queue therefore it would be easier to implement and easier to manipulate in the future. It is also essential in keeping track of employees who also need to be added and deleted.

Chef
The chef uses queues for making sure that orders that come first are made to order before orders that come after. This is because of performance. If an order comes earlier and is cooked later, then the customer will have a long waiting time and the waiting times of all customers will be uneven and disorganized. The strict format of the queue allows these processes to be easier.

Waiter
The main data structure for the WaiterPC is for placing the order. Each table's order will be a multi-dimensional array. I.e.

Table 1	Table 1	Table 1	Table 1
Customer 1	Customer 2	Customer 3	Customer 4
Drink	Drink	Drink	Drink
Appetizer	Appetizer	Appetizer	Appetizer
Entree	Entree	Entree	Entree
Dessert	Dessert	Dessert	Dessert

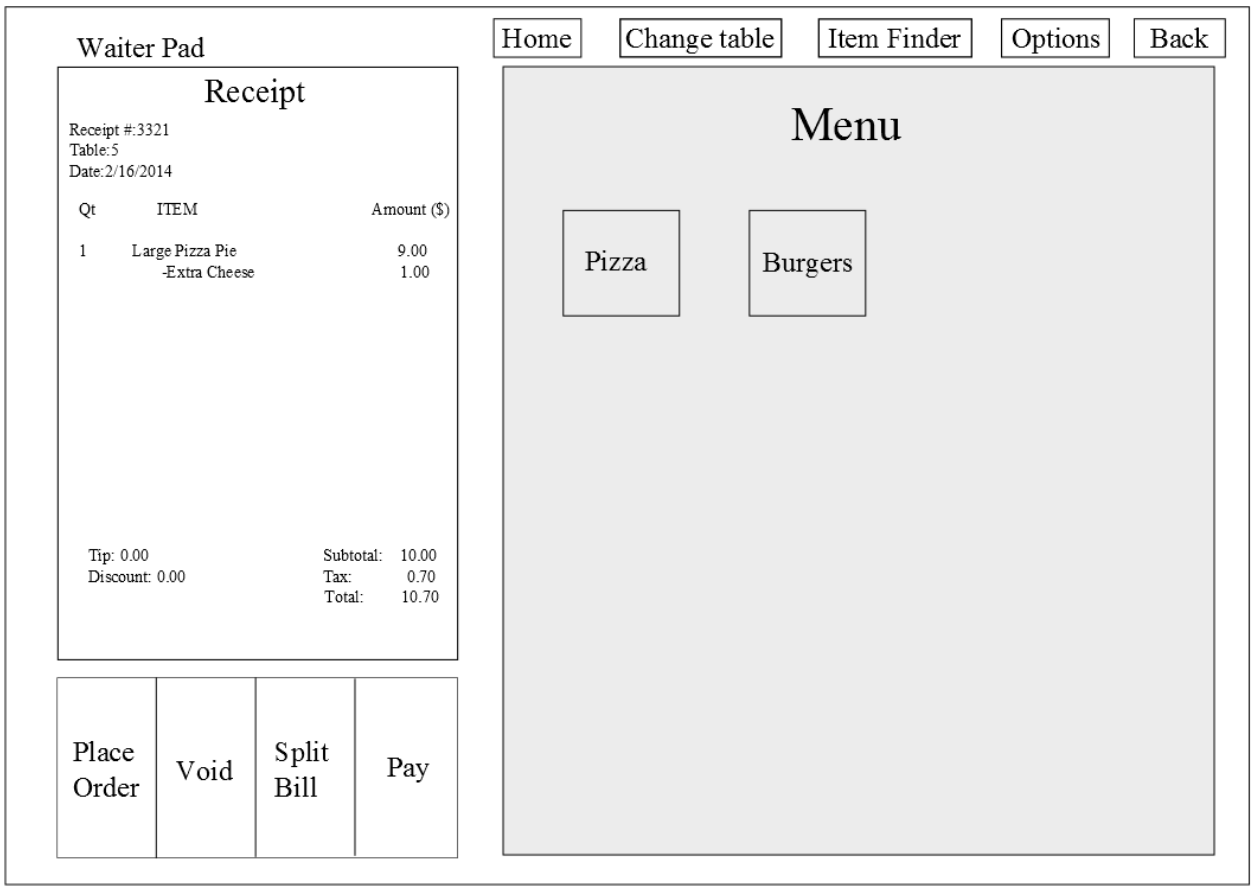
This design was used based on the flowchart shown in the algorithm section for the WaiterPC.

Customer:
The tables given for the customers will be an array list. This will make it easier for employees to add or delete tables at a given time. Selecting the tables will be a toggle or a boolean. It simply will allow a customer to select the table or not, depending on the 2 statuses of either available or unavailable.

11) User Interface Design and Implementation

During the initial interface design of our system we took careful consideration on minimizing user effort while keeping Ease-of-use in mind. Our main objective was have our system do exactly what the customer requirements entailed for each specified goals. Therefor, we have not made any significant chances to the chef, bus boy, or waiter GUI implementation. However, we feel that the manager side of the system needs some more work. The manager GUI implementation has a difficult Ease-of-use because as admin you need to make changes to the system on the fly. As far as the user effort for the manager, we feel that the system is optimized and straight to the point.

Preliminary Design(draft images)



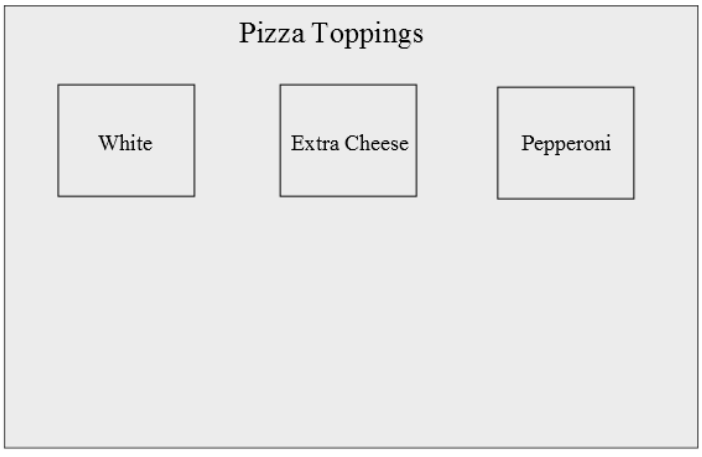


Figure 1
Image of the waiter's GUI(graphical user interface) and the image below is the pop up interface if the user would hit the pizza option on the menu.

We are going to look at the Use case #9 *SendOrder* from the perspective of the waiter as a user. In figure 1 above**(Page25)**, after the waiter gets an order from the customer he can use the Menu section to navigate to a specific item and what toppings they

would like. In this particular case the customer choose to order a 'Large Pizza Pie' and requested 'Extra Cheese.' So using the menu the user would navigate to the 'Pizza' item that will invoke a pop up that will have all of the available toppings. After Selecting a topping the receipt section will automatically update to show a draft of the Receipt which can be used for the user to easily repeat back or edit the order. When the order is ready to be placed, the user can 'Place Order' which will invoke the *SendOrder* function. This will send a signal to the server and then back to the chef.

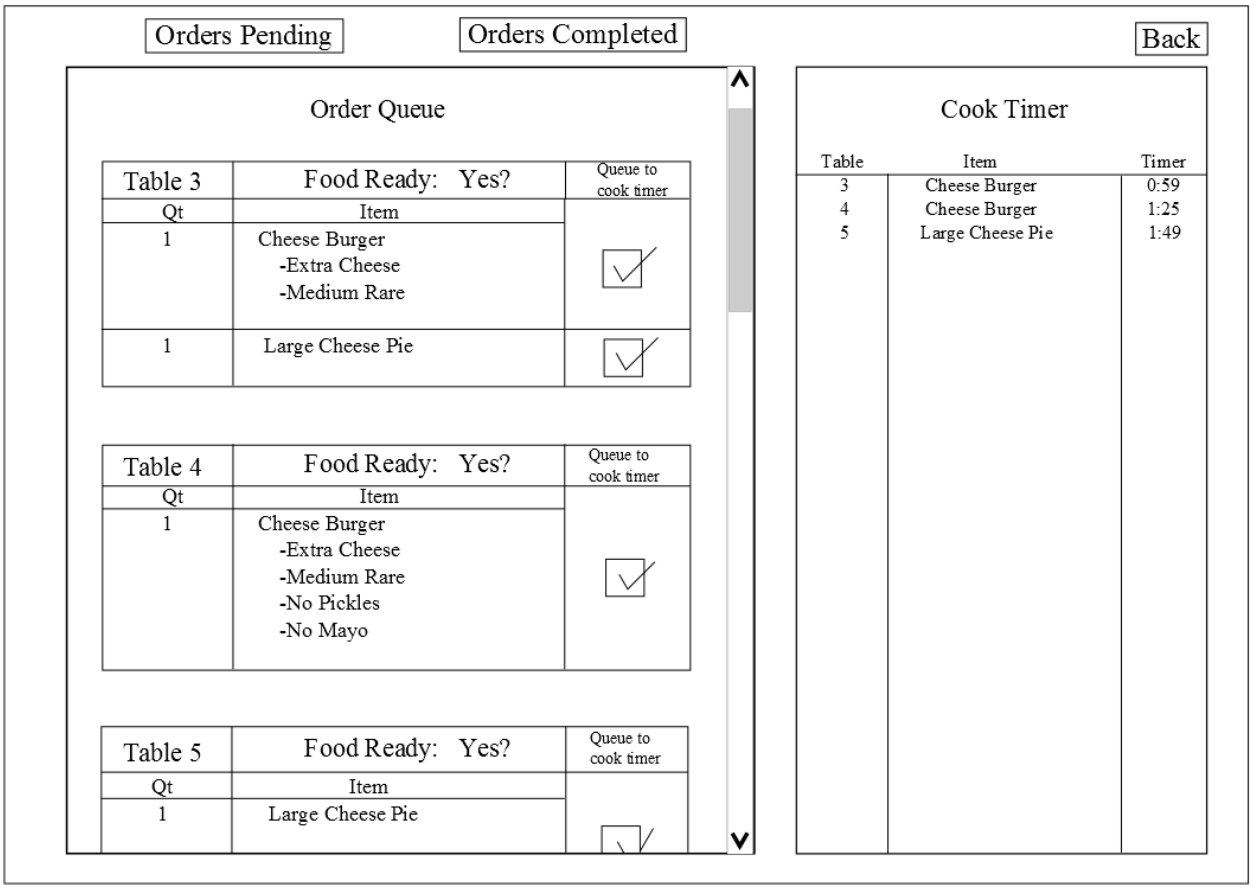


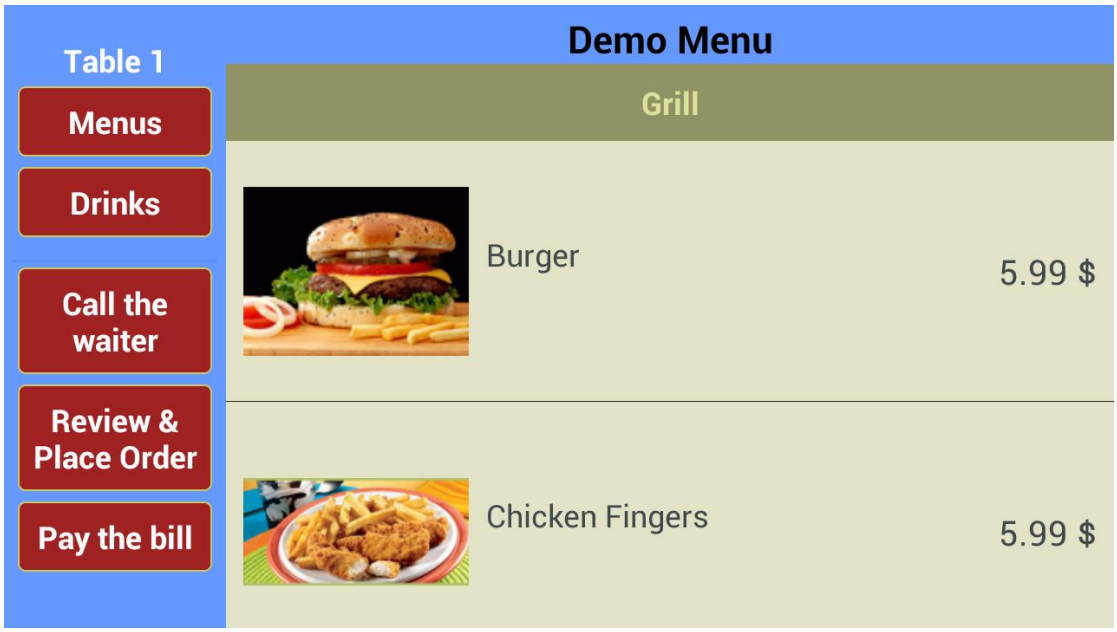
Figure 2
Image of the Chef's GUI and all his/her incoming orders.

The Chef will be receiving automatic updates of new orders once the system receives *SendOrder.* In figure 2 above, the interface will automatically update itself on new incoming orders based off of Use Case #6 *RecieveOrder*(more detail later). Here the Chef can Queue in items to the Cook Timer once prepared and ready to cook. After the chef feels that the order is ready they can use the 'Food Ready: Yes?' button in order to invoke Use Case #7 *OrderReady.* The Waiter's pad will receive a pop up notification with the table # in which the food is ready. After the order has been sent the interface will update and move the order into the 'Orders Completed' tab.

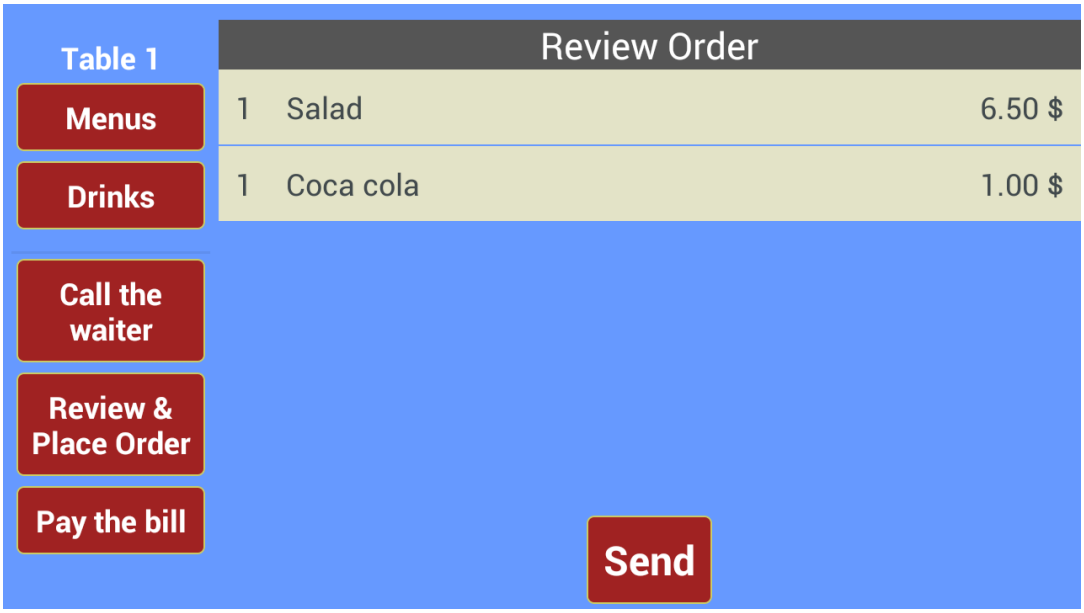
Final user interface design

We have simplified the interface to fit as many features as possible on the waiter's tablet or even the customer's phone application.

On this screen the waiter or the customer is able to selected their table. After they select their table they will be prompted to log in. The waiter can log in using their username and password to any table, the customer can log in using the specified log in information located on each table.

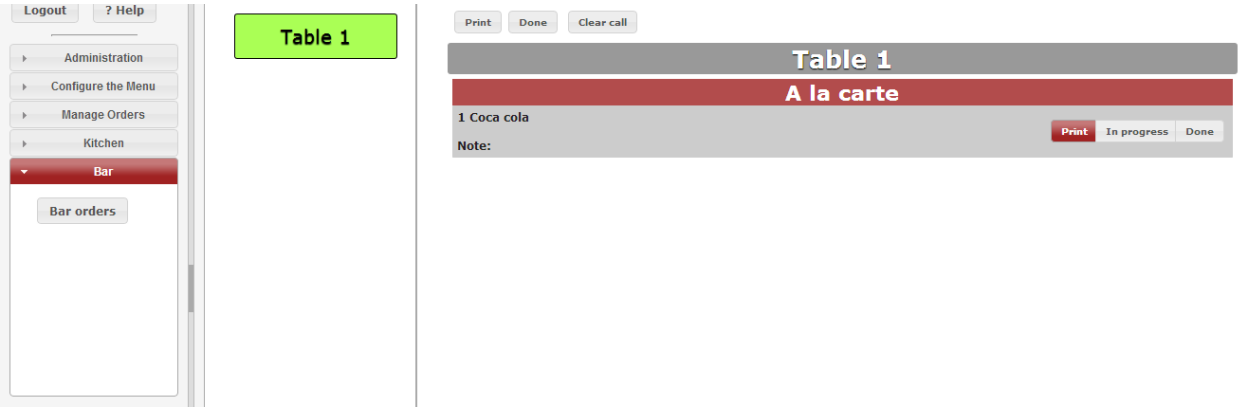


After the user logs in successfully they will be able to access the menu and select the items they would like to order.



After the user selects the items they would like to order they will be able review the items in their cart before they place the order.

After the order is placed, the kitchen will see the order on their queue. The chef can change the status of the order and mark it as, in progress or done.



If the order includes any drink items, the bartender will be notified and the drink items will show up on their system. The bartender is also to change the status of the drink order to, in progress or done.

12) Design of Tests

Note For the following Test case diagrams we decided to use Spring 2013 template since we felt it was clean, easy to understand, and straight to the point. We are also going to implement some of their test cases.

Manager

For the manager system we plan on testing every function in the system besides the few we highlighted below. Since the manager has admin rights to the system, all the changes he or she makes will be reflected across all the devices. In the testing phase will be consistently adding and removing items such as employees and food items in order to make sure everything is

working as intended. For algorithm testing in the case of the manager will be Inventory prediction. we plan manualing inputting dummy variables and changing dates over a long period of time in order to see how well it can predict food outcomes.

Test-case Identifier: TC - 01 **Function Tested**: addInventoryItem(inventoryItem:inventoryItem) :Boolean **Pass/Fail Critera**: the test will only pass if new item appears on list

Test Procedure	**Expected Results**
-Call Function(Pass) -Call Function(Fail)	-New item shows up on an updated inventory list -Function will return NULL value if no new item is added to the list

Test-case Identifier: TC - 02 **Function Tested**: removeInventoryItem(name:String) :Boolean **Pass/Fail Critera**: This test will only pass if item is successfully removed

Test Procedure	**Expected Results**
-Call Function(Pass) -Call Function(Fail)	-Old item does not show up on the updated inventory list -Function will return NULL value if item fails to be removed

Test-case Identifier: TC - 03 **Function Tested**: viewInventory():ArrayList **Pass/Fail Critera**: the test will only pass if a list with all inventory items appear

Test Procedure	Expected Results
-Call Function(Pass) -Call Function(Fail)	-All inventory items are return to screen -Function will return NULL value if nothing is returned

Test-case Identifier: TC - 04 **Function Tested**: viewPopularity(name:String):int **Pass/Fail Critera**: the test will only pass if the popularity stat of an item is returned

Test Procedure	Expected Results
-Call Function(Pass) -Call Function(Fail)	-The corresponding item will be returned to screen with its popularity -Function will return -1 value if the stat does not successfully return

Customer

The customer will have many options and functions that will implement how the restaurant system will go. They will be able to select the tables they want which will set the flow for the restaurant. There will also be a function to add or remove the tables. There will also be the status for the tables, showing if they are either available or unavailable. In addition, they can also view the wait time for each table and the time the customers are there for satisfaction.

Test-case Identifier: TC - 05 **Function Tested:** Customer::SelectTable(Table): **Pass/Fail Criteria:** The system will pass if customer selected available table, it will fail if customer selects a table that can't be selected.

Test Procedure:	Expected Results:

-Call Function(Pass)	The system will pass if customer selects correct table, it will be given to customer and the table will become not available.
-Call Function(Fail)	The system will fail if customer selects a table that is already taken, the system will return an error message requesting customer to select a different table.

Test-case Identifier: TC - 06
Function Tested: Customer::viewWaitTime (Menuitem m) : int throws exception
Pass/Fail Criteria: The test passes if the correct wait time in seconds is returned from the controller for the menu item passed as a parameter.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	Customer will be able to see correct wait time and time of customer.
-Call Function(Fail)	An error will occur and the time will be failed to sent, function will throw exception.

Test-case Identifier: TC - 07
Function Tested: Customer::viewStatus()
Pass/Fail Criteria: The system will pass if it shows the correct status, it will fail otherwise.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	Customer will be able to see correct status of the table.
-Call Function(Fail)	An error will occur and the status will be failed to sent, function will throw exception.

Test-case Identifier: TC - 08 **Function Tested:** addTable(Table) **Pass/Fail Criteria:** The test will pass if a table is added correctly, otherwise an error will occur it if it doesn't.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	The new table will be added correctly and it will show up on the system.
-Call Function(Fail)	An error will occur and the table won't be added and the system will ask for you to try again or quit.

Test-case Identifier: TC - 09 **Function Tested:** deleteTable(Table) **Pass/Fail Criteria:** The test will pass if a table is deleted correctly, otherwise an error will occur it if it doesn't.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	The current table will be deleted correctly and it will not show up on the system.
-Call Function(Fail)	An error will occur and the table won't be deleted and the system will ask for you to try again or quit.

Test-case Identifier: TC-10 **Use Case Tested:** Waiter::notification (Table) **Pass/Fail Criteria:** The test passes if the system is able to notify the waiter according to our algorithm

Test Procedure:	**Expected Results:**
-Call Function(Pass)	WaiterPC tablet gets a notification if customer is at table and waiter has not

-Call Function(Fail)	checked in 8 minutes Waiter tablet doesn't get a notification because no customers at table or an error occurred in the system

Test-case Identifier: TC-11 **Use Case Tested:** Waiter::AddItem (Menuitem m) **Pass/Fail Criteria:** The test passes if the item is added to the queue

Test Procedure:	**Expected Results:**
-Call Function(Pass) -Call Function(Fail)	The correct data is sent through and the item that needs to be added goes on the queue of the ChefPC and as well as an added item on the specific table for the waiter to view Item is not available or an error occurred in the system

Test-case Identifier: TC-12 **Use Case Tested:** Waiter::DeleteItem (Menuitem m) **Pass/Fail Criteria:** The test passes if the item is deleted from the queue

Test Procedure:	**Expected Results:**
-Call Function(Pass) -Call Function(Fail)	The correct data is sent through and the item that needs to be removed comes off on the queue of the ChefPC and as well as deleted on the specific table for the waiter to view Item is not available or an error occurred in the system

Test-case Identifier: TC-13 **Use Case Tested:** Waiter::ViewQueue () : ArrayList<MenuItem> **Pass/Fail Criteria:** The test passes if the menu items are viewed from each table on the queue for the Waiter

Test Procedure:	**Expected Results:**
-Call Function(Pass) -Call Function(Fail)	The correct data is sent through and the items ordered by each customer on each table can be viewed on the queue for the Waiter Table is empty or an error occurred in the system

Chef

The chef will have basic actions and functions to utilize in the testing phase. The chef will only need to communicate with the waiter in order to receive the order. Within the testing phase, the chef will be able to also view the order queue of the current orders. The chef will also be able to add or delete items within an order in accordance to the customers preferences. However, the chef can ultimately determine when the order has been completely prepared and be ready to be taken to the customer using the function of itemfinished as call to the waiter that the order is done.

Test-case Identifier: TC-14 **Use Case Tested:** Chef::ViewQueue () : ArrayList<MenuItem> **Pass/Fail Criteria:** The test passes if the menu items are viewed from each table on the queue for the Waiter

Test Procedure:	**Expected Results:**
-Call Function(Pass)	The correct data is sent through and the items ordered by each customer on each

-Call Function(Fail)	table can be viewed on the queue for the Waiter Table is empty or an error occurred in the system

Test-case Identifier: TC-15 **Use Case Tested:** Chef::ItemFinished () : ArrayList<MenuItem> : Boolean throws exception **Pass/Fail Criteria:** The test passes if the item is removed from temporary storage and passed to the waiter via the controller

Test Procedure:	**Expected Results:**
-Call Function(Pass)	Correct data to be sent is passed, function returns true if the controller successfully passes the MenuItem to the waiter to be delivered.
-Call Function(Fail)	MenuItem incorrect or controller error, function returns false. Message fails to be sent, function throws exception

Test-case Identifier: TC - 16 **Function Tested:** Chef::AddOrder (TableOrder o) : Boolean **Pass/Fail Criteria:** The test passes if the menu items in the order pass in as an argument are successfully scheduled into the chef queue.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	Correct data to be sent is passed, function returns true if all the menu items part of the table order passed are scheduled successfully.

-Call Function(Fail)	If functions fails to schedule all the orders, returns false

Test-case Identifier: TC - 17 **Function Tested**: Chef::RemoveOrder (TableOrder o) : Boolean **Pass/Fail Criteria**: The test passes if the menu items in the order pass in as an argument are successfully removed from the chef queue.

Test Procedure:	**Expected Results:**
-Call Function(Pass)	Correct data to be sent is passed, function returns true if menu item is removed.
-Call Function(Fail)	Function returns false, if menu item is incorrect or cannot be removed.

13) History of Work, Current Status, and Future Work

a.) Merging Contributions From Individual Team Members

In order to merge our work together and be organized, we made a google document. This was the best way since we are able to work together in different locations and have our work saved at the same time on one final document we are able to edit together. Some issues that we encountered were that our format would be a little different and we would be unorganized since our parts would be at different locations. This problem was not that hard to solve. We fixed the format issue by agreeing on a specific font and spacing. We fixed the unorganization part by labeling everything beforehand when making the template so we are able to put the correct parts in the same place.

b.) Project Coordination and Progress Report

History of Work

January 21st- January 29th

Our team decided to take on the restaurant automation project and created the proposal of how we would go about tackling this project and how we would decide to implement this. We began by reading through past projects and understanding where they did well, and where they also had weaknesses, and we decide to create a system that would build upon those weaknesses

February 1st- February 23rd

We received feedback on how to improve our proposal and how we could plan better for our project as well. Our aim was to build upon the new proposal and incorporate those new ideas into our first report. A theoretical model of our system was built using the report guidelines given, incorporating Customer Statement of Requirements, Glossary of Terms,
Functional Requirements, Effort Estimation, and Domain Analysis. We divided the work evenly and pieced together the report at the end, submitting our report on February 23rd

February 24th- April 19th

The team then began planning for the second report, building on what we learned from drafting our first report on the specification of our system. This time we would focus on the design aspect of the system. Through the drafting of the second report, we began to develop part of the system that we would be using during the first demo. Through the Interaction Diagrams, the Class Diagrams and Interface Specification, System Architecture and System Design, Algorithm and Data Structures, User Interface Design and Implementation, and Design of Tests, we drafted our second report and began to build our system. We focused on building the server and the GUI and divided the work evenly between all team members.

Current Status

Our system is currently being developed, the building being separated into two components, the server and the GUI. We have divided the overall team into two teams to work separately on either component. As of right now, we do not have any functional components, but we are focusing our maximum effort into completing parts of the server and the GUI in time for our first demo presentation

Future Work

Currently, we are in the middle of developing a working version of our server and GUI for the first demo presentation. For the demo, we hope to have a server and GUI that interact with each other, and communicate with other modules as well, such as the chef

and waiter modules. Achieving this communication is a top level priority because that communication is what will allow the entire system to work as a whole. But it also poses a problem as the modules will be worked on with separate groups, so the modules won't work together at first due to maybe different programming languages or even errors in the coding of the modules. But ultimately, we want to have our server and GUI combine with all of the modules and interact as a single system.

Below is a pseudo code on our chef queuing algorithm. The way it would work is the chef would start his/her system which would invoke StartChefPrep() function. The StartChefPrep() function has an infinite for loop that is going to listen for signals from the waiter. Once the waiter sends in a stream of order information, the algorithm separates the appetizer from main courses and then starts queueing in food according to predetermined cook times. The way the function figures out the what order to cook the times in is by finding the largest cook time item and then finds the difference in time between the other items and queues them accordingly. Once the chef hits the exit button, the function closes.

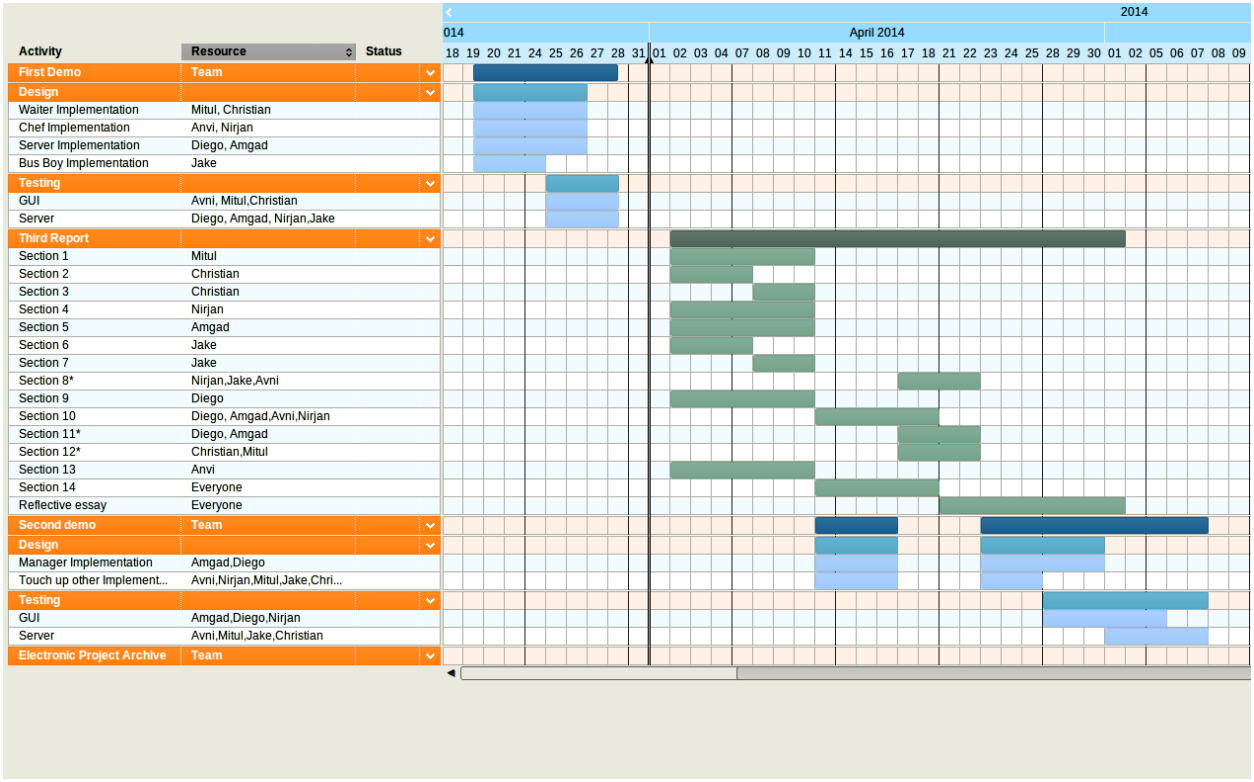
```
//we are going to assume that all appetizer take a set time

void StartChefPrep(void)
for( ; ; )
{
 prompt close button = false
 listen for signal from waiter
 read order
 if(appetizer in order)
 {
        prompt a prep button = false
        while( button press == false)
        {
                if(button pressed)
                        {
                                Queue in appetizer and set a buffer timer of 5 minutes
                                button press = true
                        }
        }
 button press = false
```

```
 }

 prompt a prep button = false
 while( button press == false)
 {
        if(button is pressed)
                {
                        Take in all items into cooktimerorder()
                        button press = true
                }

 }
 button press = false

 if (close button == true)
{
        {
                close chef screen
        }
}
}

cooktimeroder()
{
 find longest cook time item
 for(each item i )
        {
                new timer = longest item cook time - item(i)
                if( new timer == 0)
                        {
                                break
                        }
                queue in item(i) in new timer minutes
        }
}
```

One feature for future work is to implement an inventory database. This database would

update in real-time as soon as an order is placed. This would allow the manager to track the inventory and order inventory items whenever the stock reaches a certain threshold.

Another feature we would like to implement for the future is the integration of a credit card reader. This credit card reader would process the transaction on the spot instead of the waiter going to another PC like what is usually done in restaurants. This credit card reader will also be able to split payments and refunds. This will further improve efficiency as well as security and increase customer satisfaction because customers don't normally see the waiters charging their credit cards in front of them and also since they will be charged right after the customers are done eating it will speed up the whole payment process.



c.) Breakdown of Responsibilities

WaiterPC Responsibilities:	Mitul	Christian	Amgad	Diego	Jake	Nirjan	Avni
Customer Statement of Requirements (CSR)	X	X	X	X	X	X	X

System Requirements	X	X	X	X	X	X	X
Functional Requirements Specification	X	X	X	X	X	X	X
User Interface Specification	X	X	X	X	X	X	X
Domain Analysis	X	X	X	X	X	X	X
Interaction Diagrams	X	X	X	X	X	X	X
Class Diagram and Interface Specification	X	X	X	X	X	X	X
System Architecture and System Design	X	X	X	X	X	X	X
Algorithms and Data Structure	X	X	X	X	X	X	X
User Interface Design and Implementation	X	X	X	X	X	X	X
Design of Tests	X	X	X	X	X	X	X
Project Management	X	X	X	X	X	X	X
Plan of Work	X	X	X	X	X	X	X
References	X	X	X	X	X	X	X

ManagerPC Responsibilities:	Mitul	Christian	Amgad	Diego	Jake	Nirjan	Avni
Customer Statement of Requirements (CSR)	X	X	X	X	X	X	X

System Requirements	X	X	X	X	X	X	X
Functional Requirements Specification	X	X	X	X	X	X	X
User Interface Specification	X	X	X	X	X	X	X
Domain Analysis	X	X	X	X	X	X	X
Interaction Diagrams	X	X	X	X	X	X	X
Class Diagram and Interface Specification	X	X	X	X	X	X	X
System Architecture and System Design	X	X	X	X	X	X	X
Algorithms and Data Structure	X	X	X	X	X	X	X
User Interface Design and Implementation	X	X	X	X	X	X	X
Design of Tests	X	X	X	X	X	X	X
Project Management	X	X	X	X	X	X	X
Plan of Work	X	X	X	X	X	X	X
References	X	X	X	X	X	X	X

ChefPC Responsibilities:	Mitul	Christian	Amgad	Diego	Jake	Nirjan	Avni
Customer Statement of Requirements (CSR)	X	X	X	X	X	X	X
System Requirements	X	X	X	X	X	X	X

Functional Requirements Specification	X	X	X	X	X	X	X
User Interface Specification	X	X	X	X	X	X	X
Domain Analysis	X	X	X	X	X	X	X
Interaction Diagrams	X	X	X	X	X	X	X
Class Diagram and Interface Specification	X	X	X	X	X	X	X
System Architecture and System Design	X	X	X	X	X	X	X
Algorithms and Data Structure	X	X	X	X	X	X	X
User Interface Design and Implementation	X	X	X	X	X	X	X
Design of Tests	X	X	X	X	X	X	X
Project Management	X	X	X	X	X	X	X
Plan of Work	X	X	X	X	X	X	X
References	X	X	X	X	X	X	X

BusBoyPC Responsibilities:	Mitul	Christian	Amgad	Diego	Jake	Nirjan	Avni
Customer Statement of Requirements (CSR)	X	X	X	X	X	X	X
System Requirements	X	X	X	X	X	X	X

Functional Requirements Specification	X	X	X	X	X	X	X
User Interface Specification	X	X	X	X	X	X	X
Domain Analysis	X	X	X	X	X	X	X
Interaction Diagrams	X	X	X	X	X	X	X
Class Diagram and Interface Specification	X	X	X	X	X	X	X
System Architecture and System Design	X	X	X	X	X	X	X
Algorithms and Data Structure	X	X	X	X	X	X	X
User Interface Design and Implementation	X	X	X	X	X	X	X
Design of Tests	X	X	X	X	X	X	X
Project Management	X	X	X	X	X	X	X
Plan of Work	X	X	X	X	X	X	X
References	X	X	X	X	X	X	X

14) References

"Concepts: Requirements." Concepts: Requirements. Polytechnique Montreal, 2012. Web. 5 Feb. 2014. -(Used for Non-Functional Requirements)

Nick Leshi. (2010). Good Restaurants Come and Go. Available: http://open.salon.com/blog/kikstad/2010/06/25/good_restaurants_come_and_ go. Last accessed 8th Feb 2014. -(Used for Cover picture).

Group#1 Spring 2013. (2013). Inventory usage rate estimation and runout date estimation. Auto-Serve. 1 (2), 75-116.

Group#1 Spring 2013. (2013). Inventory usage rate estimation and runout date estimation. Auto-Serve. 1 (2), 129-151.

easymenu-restaurant-menu
https://code.google.com/p/easymenu-restaurant-menu/