

Report 3 Part 1: Restaurant Automation

Food•E•Z (<https://sites.google.com/site/sefoodez/home>)

Group #3

Sujay Bandarpalle
Julian Esteban
Kanav Tahilramani
Omar El Warraky
Jonathan Du
Paolo Umali

All members contributed equally!

Summary of Changes

- Logo Updated and Changed
 - Labeled All Diagrams and Tables in a clearer fashion
 - Added Descriptions to each use case and traceability matrix
 - Showed complete evolution of UI and a description and step by step basis of how to go through it
 - Fixed Reference Section and Added Titles
 - Added Estimation using use case points
 - Added Object Constrained Language Contracts
 - Updated History of Work and Added Future Work
-

Table of Contents

1.Customer Statement of Requirements.....	7
1.1 Down Time	7
1.2 Wasted time trying to handle customers' transactions.....	9
1.3 Dissatisfaction due to wait time.....	10
1.4 Managing Profit and loss	9
1.5 Down Time	10
2.Glossary of terms.....	14
2.1 Technical Terms.....	14
2.2 Non-Technical Terms.....	14
3.User Stories.....	16
3.1 Manager	17
3.2 Chef.....	18
3.3 Busboy.....	18
3.4 Waiter/Waitress	19
3.5 Bartender.....	20
3.6 Takeout Cashier	20
3.7 Customer.....	21
3.8 General Employee.....	21
4.Functional Requirements Specification.....	23
4.1 Stakeholders	23
4.2 Actors and Goals.....	23
4.3 Use Cases.....	25
i) Casual Descriptions	25
ii) Use Case Diagram.....	25
iii) Traceability Matrix.....	27
iv) Fully Dressed Descriptions.....	29

5.Effort Estimation.....	46
6.Domain Analysis.....	49
6.1 Domain Model.....	49
i) Concept Definitions	49
ii) Association Definitions.....	51
iii) Attribute Definitions.....	52
iv) Traceability	
Matrix.....	55
v) Domain Model Diagram.....	56
6.2 System Contracts.....	57
6.3 Mathematical Model.....	59
7.Interaction Diagrams.....	63
8.Class Diagram and Interface Specification.....	68
8.1 Class Diagram	68
8.2 Data Types and Operation Signatures.....	71
8.3 Traceability Matrix.....	81
8.4 Design Patterns.....	82
8.5 Object Constraint Language.....	83
9.System Architecture and System Design.....	86
9.1 Architectural Styles	86
9.2 Identifying Subsystems.....	87
9.3 Mapping Subsystems to Hardware.....	88
9.4 Persistent Data Storage.....	88
9.5 Network Protocol.....	88
9.6 Global Control Flow.....	89
9.7 Hardware Requirements.....	90

10.Algorithms and Data Structures.....	91
10.1 Process and Prepare Order	91
10.2 Calculate Business Stats.....	93
10.3 Table Designation Algorithm.....	94
10.4 Information Modification Algorithm.....	95
10.5 Order Progress Queue Algorithm.....	96
10.5 Data Structures.....	97
11.User Interface and Implementation.....	98
11a.Details of Employee Web Portal.....	120
12.Design of Tests.....	121
12.1 Manager.....	121
12.2 Chef.....	124
12.3 Waiter.....	126
12.4 Busboy.....	127
12.5 Employee.....	128
12.6 Controller.....	130
12.7 Request Handler.....	131
12.8 Coverage and Integration Strategy.....	133
13.History of work, Current Status, Future Work.....	134
14.References.....	139

Customer Statement of Requirements:

The world is coming to an era of technology. Our restaurant is undergoing a renovation to implement technology into our stores. We have a deal made with Microsoft to implement tables and hand held tablets with top notch surfaces using the new Windows environment. We'd like to have a special app to be able to interact with our new windows environment. Below is a couple of problems we would like to be solved in this app that we can't find anywhere else.

Problem: Down Time Associated with Checking in on the Reservation List and Order Preparation Progress

In the restaurant industry, time is money. The amount of time it takes to serve customers determines the amount of groups served each day, which will determine the revenue generated from sales and tips. In order to cut down on time, we have come up with a few features that will remove the need for waiters to have to manually check in on reservation list at the front entrance as well as the kitchen. In addition, by walking around the restaurant more, a waiter exposes themselves to opportunities to be asked to for service from other customers, which will increase the amount of time between when orders are ready and when they are delivered, which increases the time that the table is occupied by the current party.

In order to solve this we came up with several features. This includes table availability schedule, the order progress queue, and the chef hotline.

Table Availability Schedule:

When customers walk in they are greeted by a tablet, in which they are asked to input their name and party size. This information is then sent back to our database, which utilizes an algorithm that assigns a table to each customer. This algorithm first uses party size to prioritize customers for each table. It does this by checking if any tables are available that can hold the specified party size. If no table is available, then the next largest party is prioritized, and so on.

When the party size prioritization part of the algorithm has ended, the next part consists of prioritizing by time. Here the algorithm checks through the available tables and if several party sizes match the specifications of an available table, then the party who arrived first is given the priority of being seated at the table. The algorithm

then proceeds to do this for the rest of the customers waiting to be seated as tables become available/unavailable.

At the end of our algorithm, the tablet at the front of the restaurant displays the name of the customer and shows a diagram of the restaurant with the table they are assigned highlighted. The customer is prompted to confirm that they have received the notification and will proceed to the table. This information is also updated for the waiters so that they know if one of their tables has recently been taken or if no confirmation has been made by the customer for a while, then to go to the front of the restaurant and manually look for the customer. This algorithm should also be made in a way that it will rotate the customers around the store. For example a medium sized store would be split into 4 with 4 waiters. Thus when giving the first customer a table it will send him to section 1 and then the next to section 2 giving this the ability to spread the customers around the store making the availability of waiters to a maximum and speeding up all processes at dining peak.

As the customer finishes dinner a tablet will be available for him to pay for his dinner. This payment process indicates that this table needs to be cleaned which will light up automatically on the application for waiters to know what tables need to be cleaned. When the table is cleaned the waiter will have to process this manually by a simple touch of a button letting the system know the table is ready to serve new customers.

Order Progress Queue:

The order progress queue will be a way for the waiters and chefs to have a synchronized electronic list of order progress and availability. It will consist of a database that stores meals. When a waiter places an order for a group, all of the meals will be put on the end of the queue in the database. Both the chef and the waiter can see the queue. The chef will be able to click on an meal, and have the option to select one of three statuses: stopped, in progress, or complete. The queue will also show how long it has been since the meal was started, so the waiter can estimate the time to completion. When the chef sets a meal's status to complete, it will send an alert to the waiter that placed the meal order so they know it is ready for pick-up.

Our queue will also be able to list orders for take-out, and will designate which meals are to be placed at the take-out window or the waiter pick-up window upon completion. Priority for which meal is to be prepped is assessed by the order in which they are placed, regardless of whether the meal is for take-out or dine-in.

Problem: Wasted time trying to handle customers' transactions

Many restaurant dwellers arrive at a restaurant with an empty stomach, ready to be seated and ready to devour anything in their path. Once seated, customers begin their watches, timing customer service and the time it takes for food to come out. As a restaurant employee, it is our job to ensure high-end customer service and minimize the time taken for food reach the customer. The problem at hand is time: time it takes for servers to take orders, place orders to the chefs, and bring out the food. With so much time wasted, customers will be seated longer and arriving guests will have to wait longer to be seated. To maximize profit and keep customers wanting to come back, we have come up with solutions that will speed up the guest visit.

Create a functional menu to effectively maximize profit and raise customer satisfaction

At restaurants like Applebee's or TGI Fridays, hosts/ess seat guests and hand each guest a menu. Once they have decided, servers take their orders and walk to the kitchen to give chefs the orders. Once orders are completed, servers come to the "ready area" and look for the orders to bring to hungry customers. While this is extremely time consuming, our restaurant automation application can significantly speed up the entire process and maximize customer satisfaction and restaurant profit.

With our restaurant automation application, we have a functional menu that servers can reference and will be able to place orders that will be sent directly to the chefs without having to walk to the kitchen to hand them orders. Having this functional menu will significantly speed up the time it takes for orders to travel from customer to server to chef. Once orders are completed by chefs, servers will be notified via our restaurant automation application. Having been notified by the chef that orders are completed, servers will not waste time checking up on orders and can designate more time tending to customers; tending to customer needs instead of constantly checking if an order is complete will raise customer satisfaction. The waiters will be holding the tablets for taking the orders so waiters can always give their personal opinion, customers usually like that. We would also like to have a personal app customers can order and it will show up that this customer has placed an order on the system himself so the waiter wouldn't have to go over to them. All in all, speeding up the process at which customers place orders and receive food will significantly reduce the time customers occupy a table; thus, allowing more guests to be seated and maximizing profit.

Problem: Dissatisfaction and unhappiness with regards to the wait time for paying the bill

Imagine that you and a group of friends are customers at a restaurant and you are all ready to pay the bill. You call over the server, but the server is waiting on another table. Five minutes pass by. The server finally comes and you tell the server that you are ready to pay. Another five minutes pass by where the server has to calculate the total cost of your meal. He hands you the check, but you wanted the bill split amongst you and the rest of your group. Another five minutes pass where the server has to recalculate the bill and split amongst all of you. Finally, you decide to pay with a credit card and another five minutes pass where he has to run to the swipe machine and process the bill. The overall process for paying the bill took twenty minutes. In that twenty minutes, another group of customers could have been seated, placed their orders, and possibly begun eating. Whether its splitting the bill or having multiple frustrating interactions with your server, paying the bill has always been a long and tiring process at restaurants.

With our restaurant automation application, we can significantly speed up the process of paying the bill. Once a server has placed an order, the order automatically gets placed in the corresponding table's bill. Furthermore, the application will keep track of what orders correspond to which guest within each table. Having the order kept in our database, we will eliminate the use of pencil and paper and the possibility of miscalculations on the bill (since our application will conduct all the arithmetic from the bill). If the server requests to split the bill, our application will prompt the server with the question "how many people would you like to split the bill amongst?". The app should be able to make a personal check for each of the customers on the table if they wish so. All of these features within the bill payment interface will eliminate the time it takes for servers to calculate the bill and significantly reduce the time needed for customers to pay the bill.

With less time required to handle transactions, more time can be allocated to seating and assisting new guests. Not only will we maximize profit, but speeding up the process of paying means customers will leave the restaurant with a smile and wanting to come back.

Problem of managing Profit/Loss statements efficiently and quickly:

Nowadays finding a complete system that deals with everything is becoming difficult. Managers want to see their profits and losses whenever they need it on a monthly basis to make sure everything is going well. Making a P/L calculator on an application will prove to be an essential use to the Manager as he will easily keep progress of what is occurring in terms of finance for his store wherever he is. Managers have many tedious things they must attend to so putting all finance matters linked directly with customer checks, payrolls and store expenses on the spot will allow the manager to see a budget and control when he needs to take action

or make a change on a day to day basis. This also will allow managers to view different profit margins for each store in the future to be able to grasp a general idea of each store's performance. This part of the application will also be able to process the tips separately from the profit and categorize it to each waiter so the manager can know exactly how much tips go to each individual.

Problem of Menu addition and Availability:

Having a part of this application to allow the manager to simply add and remove products off the menu will be very convenient. Managers won't need to buy new menus or update customer menus (takeaway), as for everything will be online and ready to go. We'd like the manager to have the ability to add an item or remove it with ease as well as keep track of the items availability in the kitchen and view how many orders are put on a particular item. Being able to access this from any place including his home or car allows the manager a lot of ease to carry out other tedious manners he needs to attend. Such as meetings or looking at other stores and so on. It also allows in the future managers to contact different stores in the chain and see the differences of Menu formats.

Employee Portal

Managers find trouble seeing employees checking in and out and watching their work schedules and seeing when and where they log in. Checking activity on spot wherever the manager is. Thus we need an Employee Portal.

The employee portal works as a universal scheduling manager and tool for workers at the restaurant. It will be accessible in two ways. Employees can login to the application on a tablet and view a calendar which shows all available shifts as well as those which they are signed up for. A similar interface will be available through a website as an online web portal.

This functionality enables several options for the staff. It allows them to sign up for future shifts, open a previously taken shift for coverage, and let go of a shift with a provided reason. This is used as opposed to manually taking note of all such information; it automates the documentation and shift management in the backend with minimal input required from employees.

In addition, there is possibility for expansion in that that employees can use either the tablet application or the website to clock their hours. The purpose behind this feature is to assist in automating the weekly paycheck distribution as it stores details pertinent to those calculations, i.e hours worked, sick leave, etc. This information can be used by managers to minimize the amount of time spent on related tasks per pay cycle.

Employees also find this a way to feel comfortable around technology. It helps them make sure their hours are all logged in correctly while as to see a clear schedule to their work and those around them.

The scheduler works to keep an organized log of work hours as opposed to keeping track of the shifts by hand, and the purpose behind the clock in/out is to reduce routine tasks and automate menial work that is required frequently. Lastly, the whole portal has added convenience of being accessible in two ways.

Plan of Work and Product Ownership

Within our group of 6 members, we have split into 3 pairs to maximize productivity. The pairs are as follow: Jonathan and Paolo, Julian and Sujay, and Omar and Kanav. Each pair will have specific tasks during the next few weeks that they will contribute. The main objective for the next few weeks will be to create the basis of the restaurant automation application before we branch off into sub-features using Windows.

Jonathan and Paolo: Our plan to accomplish in the next few weeks is to create a layout of the restaurant with tables. Within the layout, we want to add different functionalities which include:

- Create interface for customers' bill (<http://pay.opentable.com/>)
- Average Tip calculator for Managers to Gauge Waiter Service/Attractiveness
- Display of common tip denominations in the billing screen in order for a waiter to gauge the satisfaction of each party

Julian and Sujay: Our plan to fulfill is to create a login screen that will have different functionalities for different users.

- Speak to the Chef
- Create a functional menu for waiters/waitresses to use
- Table Availability and Wait Time
- Chef mode = The chef will be able to see the database listing the meals in the order of which they were placed. Upon clicking the meal, it will open up and show the individual items of the meal, along with three buttons that are Stopped, In Progress, and Complete. Shows time since start, estimated time to completion (can go negative which indicates a slower than usual prep time, and a likelihood of the meal finishing). The selection of these buttons will update the meal database queue for the waiter/waitress as well, so that they can see which meals are ready for pickup.

Omar and Kanav:

- Our plan to accomplish in the next few weeks is to design and create a menu that allows waiters/waitresses to place orders that can be directed to chefs.
-

- Online employee portal which handles scheduling tasks (add, remove w/ reason, swap, etc)
 - Profit/Loss tracking with database
-

Glossary of Terms:

Technical Terms:

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

Employee Portal: Schedule accessible by employees that lists their shifts and any open shifts they may cover.

Restaurant Automation: Use of an internal restaurant management application to automate and carry out major operations within a restaurant establishment.

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

Table Availability Schedule: A database that shows the availability of the tables in the restaurant.

Order Progress Queue: A priority queue that shows the progress of each order and what will be prepared first.

Non-Technical Terms:

Foodies: A person who has an ardent or refined interest in food and alcoholic beverages and seeks

Bartender: Serves beverages and maintains the supply and inventory of the bar.

Bill: A statement which contains details of the menu items that have been purchased and the money that is owed for them.

Busboys: Clears tables, takes dirty dishes to the dishwasher, sets tables.

Chef: Responsible for creating and planning menus, overseeing food preparation, and supervising the kitchen staff.

Chef Hotline: The functionality that allows customers to contact the chef from the waiter's tablet to inquire about menu items.

Customer: A person or party who visits the restaurant to dine, and can order food and place reservations from a smartphone or tablet app.

Customer Satisfaction: Measurement of how food preparation, customer service, and overall experience surpasses customer expectation

Dine-in: When a customer would like to eat in the store.

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

Menu: A list of dishes from which customers can choose from.

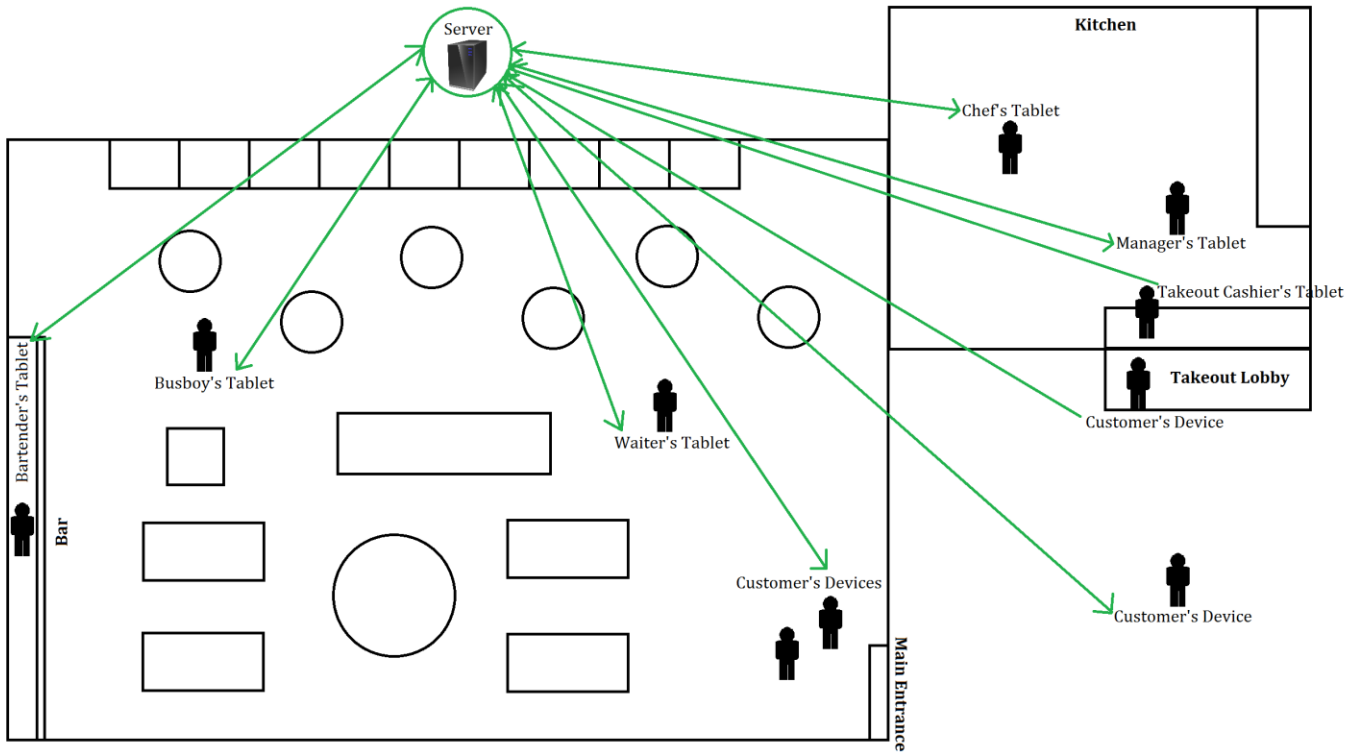
Profit: Revenue subtracted by expenses

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

Take-out: Orders that are prepared for customers that are not eating in the restaurant.

Waiter/ Waitress: Takes customers' orders, brings completed orders to customers, and marks recently vacated tables.

User Stories:



Size Key:

For each user story we include a size of 1 - 10 points. The sizes are rough estimates of how much effort is needed to implement each user story based on our group's technical skills and each user story's perceived complexity.

Priority Key:

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

ST-X-#: Middle priority (likely to be implemented)

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

As a manager.....

Identifier	User Story	Size
ST-M-1	I can modify the menu based on inventory and customer feedback so that we can serve available and desirable dishes..	8 points
ST-M-2	I can add and hire employees with a given reason.	7 points
ST-M-3	I can quickly find openings in the schedule that need to be filled so that I can easily ask employees to cover those openings.	4 points
ST-M-4	I can easily pull the profits and loss of the store for the week, month or year.	7 points
ST-M-5	I can add expenses and change salaries of employees.	5 points
ST-M-6	I can change item pricing.	5 points
ST-M-7	I can view the amount of tips accumulated from each employee to gauge good customer service	3 points
ST-M-8	I can track inventory lifetime and quantity.	4 points
ST-M-9	I can create new accounts and delegate/ revoke privileges to other employees based on employee rank.	3 points
ST-M-10	I can view timeclock sign-ins for all employees to see arrival and departure times.	5 points

As a chef.....

Identifier	User Story	Size
ST-Ch-1	I can see the queue of orders waiting to be prepared.	4 points
ST-Ch-2	I can mark orders as “In Preparation” and “Complete” .	5 points
ST-Ch-3	I can speak to the waiters/waitresses through their tablet	4 points
ST-Ch-4	I can modify the menu to make certain dishes available or unavailable if supplies are limited.	5 points
ST-Ch-5	I can record what orders and how many orders I made each shift to track accountability and performance.	3 points
ST-Ch-6	I can adjust and update the supply inventory to let the manager know of any lacking ingredients.	4 points

As a busboy.....

Identifier	User Story	Size
ST-B-1	I can view the tables that need to be cleaned.	4 points
ST-B-2	I can mark dirty tables, tables being cleaned, and clean tables	4 points

As a Waiter/Waitress....

Identifier	User Story	Size
ST-W-1	I can input my customers' orders quickly and have the chef notified of the order without walking to the kitchen.	7 points
ST-W-2	I can view my customers' bill and enter their payment information.	5 points
ST-W-3	I can apply coupons and discounts to the customers' bill.	6 points
ST-W-4	I can mark tables as recently vacated so that tables can be cleaned by busboys as soon as possible.	5 points
ST-W-5	I can see when a party has made their way to their table, and can mark it on the table availability page	6 points
ST-W-6	I can add special instructions to an order in case the customer has a specific request.	3 points
ST-W-7	I can see my average tip, as well as my tip percentage history in order to gauge the quality of my service	8 points

As a bartender.....

Identifier	User Story	Size
ST-BT-1	I can see the queue of drinks to prepare in the order they were placed, and mark each order as in progress or completed	5 points
ST-BT-2	I can pull up a list of cocktail recipes, in case a customer asks for something that I have never made	5 points
ST-BT-3	I can see and alter the status of the inventory of all wines, beers, hard liquor	5 points
ST-BT-4	I can click on a cocktail recipe, and it through color coding the ingredients, it will tell me which we still have stocked	5 points

As a Takeout Cashier.....

Identifier	User Story	Size
ST-T-1	I can enter the payment information for customers' ordering takeout.	5 points
ST-T-2	I can place orders for customers ordering takeout and send those orders to the chef's queue.	6 points
ST-T-3	I can view the progress of customer's takeout orders.	4 points

As a customer.....

Identifier	User Story	Size
ST-C-1	I can quickly order from a smartphone or tablet without having to wait for my waiter.	7 points
ST-C-2	I can remotely place reservations for my party to be entered into the seating queue through the app.	4 points
ST-C-3	I can set what time I would like my order to be added to the kitchen's list, so I am able to order long in advance	4 points
ST-C-4	I can walk in and place my party size and name into the tablet which will notify me where our table is when it's ready	5 points

As an Employee (General).....

Identifier	User Story	Size
ST-E-1	As an employee, I can view my shifts.	4 points
ST-E-2	As an employee, I can pick up open shifts.	5 points
ST-E-3	The menu should give descriptions of the items (ingredients, calories, etc.)	4 points
ST-E-4	As an employee, I can put my shift up for coverage.	6 points
ST-E-5	As an employee, I can sign-in and sign-out of my scheduled shifts.	4 points
ST-E-6	As an employee, I can view all shifts to see who I am working with.	3 points

Identifier	User Story	Size
ST-E-7	As an employee, I can access and view my paycheck stubs and important work-related forms (W-2 Wage and Tax form, Promotion Letter, Termination Letter, etc.)	8 points

Functional Requirements Specification:

Stakeholders

There are several types of stakeholders starting with end users, those who will be directly utilizing the system-to-be in an effort to automate periodic tasks. This would include any kind of employee, i.e waiters, bartenders, busboys, chef, and managers. Customers are stakeholders who do not directly make use of the system, but their interest is held in the enriched experience as a result of the system. Another set of stakeholders includes all those involved in development and management of the system. The interest comes from working to design and implement solutions to create the system.

Actors and Goals

Initiating Actors

Bartender

Role - The employee who is in charge of the bar and preparing drinks.

Goal - Manage the liquor inventory, and prepare drinks on the drinks queue.

Busboy

Role - The employee who is in charge of the cleanliness of the restaurant.

Goal - Manage the status of tables, and clean tables that are marked dirty on the app.

Chef

Role - The employee who is in charge of the kitchen and preparing meals.

Goal - Manage the ingredient inventory, manage the menu, and prepare meals on the order queue.

Customer

Role - The restaurant visitor who orders food and drink from the restaurant for dining-in or taking-out.

Goal - Place, pay for, and receive orders quickly, and choose whether to eat-in or take-out.

Manager

Role - The employee who manages the entire restaurant.

Goal - Manage employees and scheduling, keep track of inventory, and keep track of profits/losses.

Takeout Cashier

Role - The employee who interacts with and serves take-out customers.

Goal - Place orders for take-out customers, keep track of the take-out order queue and charge take-out customers.

Waiter

Role - The employee who interacts with and serves dine-in customers.

Goal - Place orders for dine-in customers, keep track of the dine-in order queue and charge dine-in customers.

Participating Actors**Database**

Role - The object that stores information needed for the system.

Goal - Store and modify information pertaining to inventory, employees, profits, losses, etc..

Use Cases:

i) Casual Descriptions:

Our group's user stories will serve as the casual descriptions of our use cases.

ii) Use Case Diagram:

iii) Traceability Matrix:

User Stories	UC-1	UC-2	UC-3	UC-4	UC-5
ST-M-1					█
ST-M-2					
ST-M-3					
ST-M-4		█			
ST-M-5		█			
ST-M-6		█			█
ST-M-7					█
ST-M-8					
ST-M-9					
ST-M-10					
ST-Ch-1	█		█		
ST-Ch-2	█		█		
ST-Ch-3					
ST-Ch-4					█
ST-Ch-5					
ST-Ch-6					█

User Stories	UC-1	UC-2	UC-3	UC-4	UC-5
ST-W-1	█		█		
ST-W-2	█	█			
ST-W-3	█	█			
ST-W-4	█				
ST-W-5	█	█			
ST-W-6	█				
ST-W-7	█				
ST-B-1	█				
ST-B-2	█				
ST-BT-1	█	█	█	█	█
ST-BT-2	█				█
ST-BT-3	█				█
ST-BT-4	█				█
ST-T-1	█	█			
ST-T-2	█	█			
ST-T-3	█	█			

User Stories	UC-1	UC-2	UC-3	UC-4	UC-5
ST-G-1					
ST-G-2					
ST-G-3					
ST-G-4					
ST-G-5					
ST-G-6					
ST-G-7					
ST-C-1					
ST-C-2					
ST-C-3					
ST-C-4					

Matrix provides relationship with use cases and their user stories showing which user story incorporates which use case.

iv) Fully Dressed Use Cases:

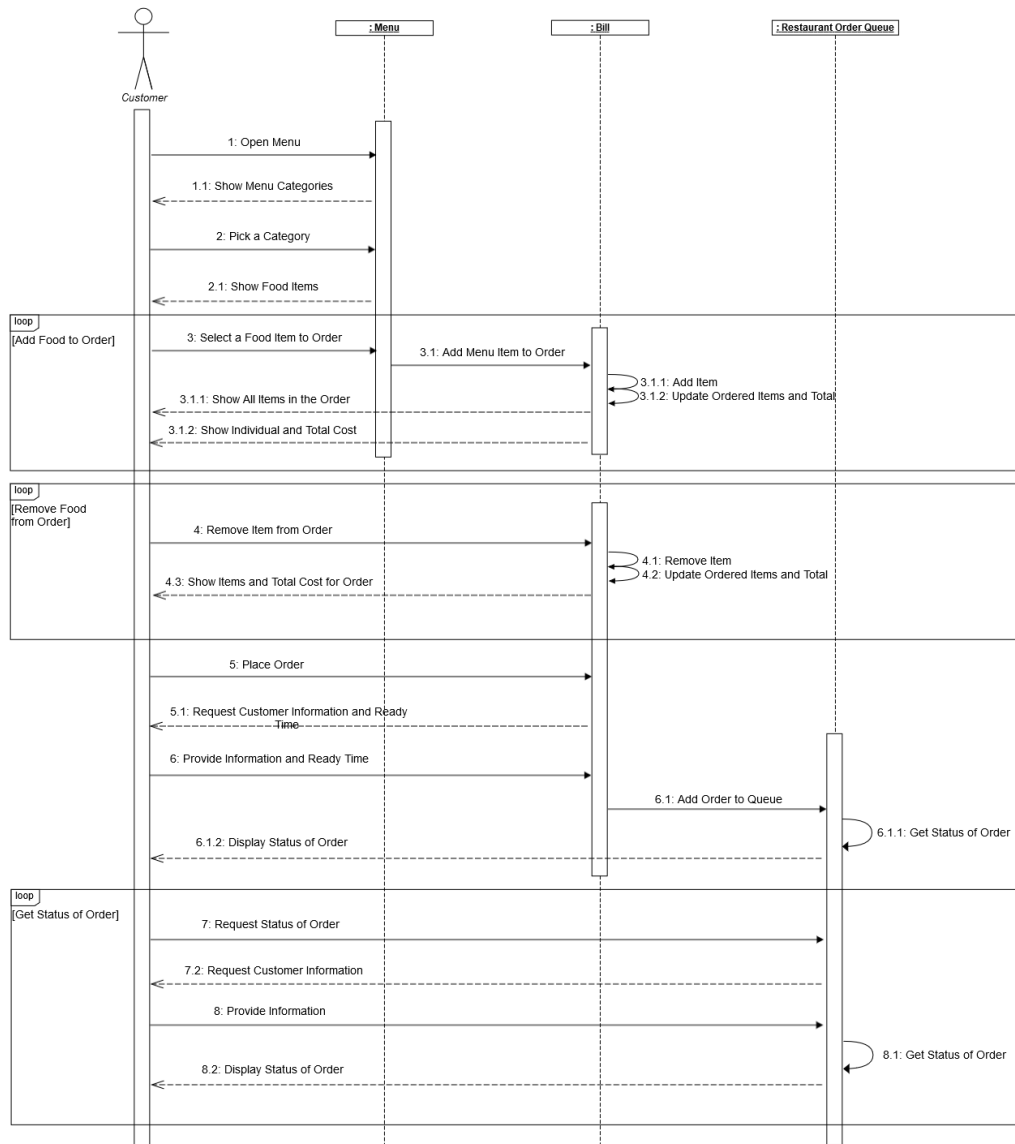
USE CASE UC-1: PLACE ORDER	ST-Ch-1to2, ST-W-1to3, ST-W-5, ST-BT-1, ST-T-1to3, ST-C-1
Related User Stories::	
Initiating Actor:	Customer
Actor's Goal:	To order food from the restaurant using the application on their smartphone or tablet.

Participating Actors:	Takeout Cashier, Waiter
Pre-Conditions:	The GUI is displayed with the menu of food items and an option to add and order them.
Post-Conditions:	The food status is shown on the screen.
Flow of Events for Main Success Scenario:	<p>→ 1) Customer picks the option to see the menu in the application.</p> <p>← 2) The system shows all the food and drink items in their respective categories, as well as the current order's bill on the right side.</p> <p>→ 3) Customer chooses which food category they would like to add to the order.</p> <p>← 4) The system displays all the items associated with the picked category, with an ability to view more information about an item (price and an option to see the nutritional facts) and an option to add to order.</p> <p>→ 5) Customer selects the option to add the desired item to the order.</p> <p>← 6) The system stays in the food and drinks menu page, with a newly updated bill on the right which includes every item added, their total cost, and options to remove an item or send the order request to the restaurant.</p> <p>→ 7) Customer selects the option to send the order request to the restaurant.</p> <p>← 8) System sends the order to the restaurant.</p> <p>← 9) System displays the status of the order (i.e. if it has been cooked or not), with an option to go back to the main menu.</p> <p>→ 10) The customer chooses to stay on the status page.</p>

Flow of Events for Extension(Alternate Scenarios):

→ 1a) Customer picks the option to see the status of a current order.
← The system prompts the customer to provide their order's unique ID.
→ Customer inputs their order ID.
← The system displays the status of the order.
→ 1b) Customer picks the option to see the status of a current order.
← The system prompts the customer to provide their order's unique ID.
→ Customer inputs an incorrect order ID.
← The system displays the message, "Invalid order ID entered, please check to make sure that you have put it in right."
→ 3a) Customer presses the back button to go back to the main menu.
← The system goes back to the main menu.
→ 5a) Customer selects option to see the nutritional facts of an item.
← The system displays the selected items nutritional facts.
→ Customer presses back button to return to the menu page.
→ 7a) Customer selects the option to remove an item from the bill.
← The system removes the item from the bill.
→ 7b) Customer selects adds another item to his order.
← The system stays in the food and drinks menu page, with a newly updated bill on the right which includes every item added.
→ 7c) Customer specifies the ready times of all food items.
← The system sends the order to the restaurant's queue with the order type: "Takeout", and the ready times for each item. It also sends the details of the order to the database for the takeout cashier to see.
← 10a) Customer selects the option to go back to the main menu.
← The system enables an option to view an existing order's status on the main menu.

Use Case UC-1: Place Order



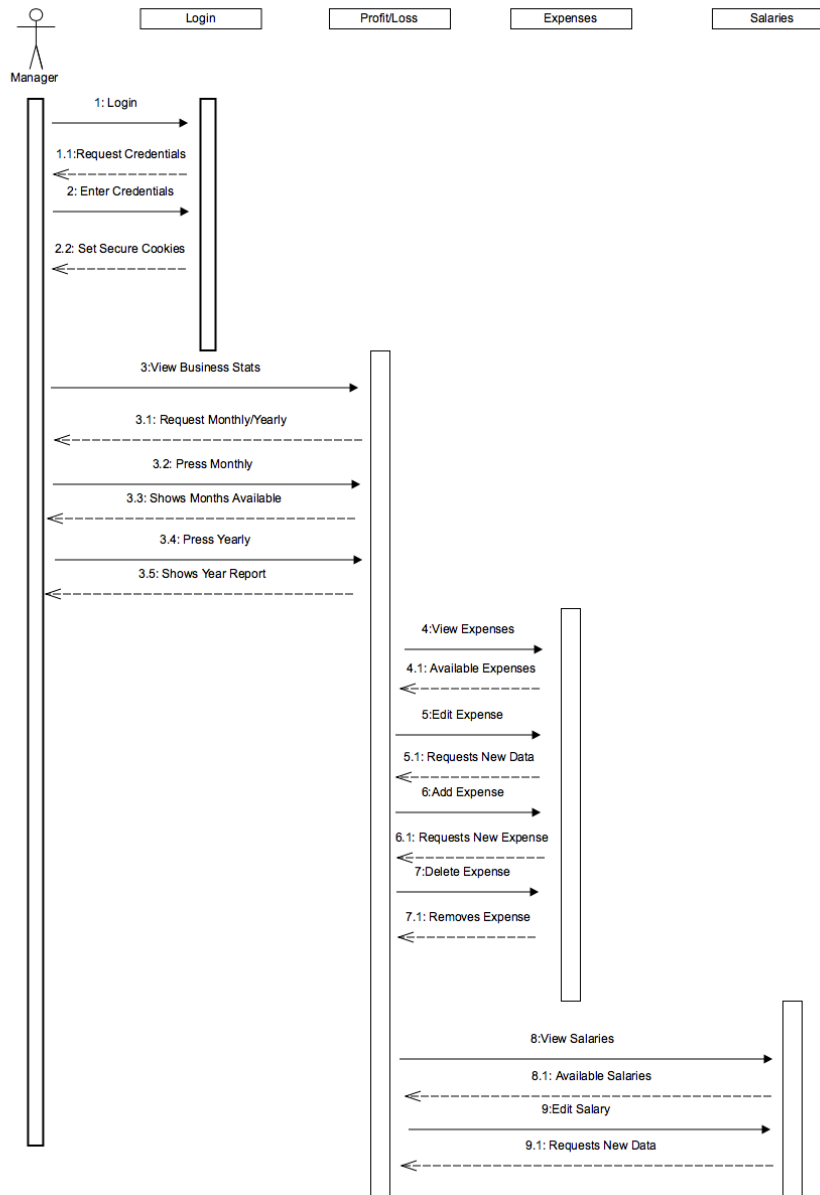
The Use Case Diagram shows how the customer interacts with the system and how the system reacts back such as placing the order in this case and how the system responds with the different scenarios as well as the next steps to be made.

<p align="center">USE CASE UC-2: VIEW BUSINESS STATS</p>	<p>ST-M-4,ST-M-5,ST-M-6</p>
<p>Related User Stories::</p>	
<p>Initiating Actor:</p>	<p>Manager</p>
<p>Actor's Goal:</p>	<p>To view profit/losses and edit expenses of the store and salaries of employees.</p>
<p>Participating Actors:</p>	<p>None</p>
<p>Pre-Conditions:</p>	<p>Manager Must be logged in. Screen Displays GUI of FoodEZ and allows manager to Enter Business Stats Section</p>
<p>Post-Conditions:</p>	<p>Goes back to Main Menu so Manager can Continue to do something else.</p>
<p>Flow of Events for Main Success Scenario:</p>	<p>→ 1) Manager Logs in to FoodEZ and chooses "View Business Stats". ← 2) System Opens up Profit/Loss Page → 3) Manager has the ability to choose stats of a single month or year. → 5) System Shows Options to choose from: Expenses or Salaries or Main Menu. ← 6) Manager Presses Main Menu → 7) System returns Manager to Main GUI Interface.</p>

Flow of Events for Extension(Alternate Scenarios):

- 3a) Manager Chooses “A Single Month”.
- ← The system prompts the customer to provide which month he would like to see
- Manager Chooses Month from drop down list
- ← Manager Can Return to Profit/loss Section
- 3a) Manager Chooses “Year”.
- ← Manager can choose “Chart” to view the flow chart of Profit/Loss of the year.
- ← Manager Can Return to Profit/loss Section
- 5a) Manager Selects “Expenses”
- ← Manager has ability to see total expenses of store.
- Manager presses “edit” to edit a single expense of the store (Food or Electricity)
- Manager presses back button to return to the Profit/Loss page.
- 5b) Manager Selects “Salaries”
- ← Manager has ability to view all Salaries of employees.
- Manager Presses “edit” to edit a salary of an employee.
- Manager presses back button to return to the Profit/Loss page.

Use Case UC-2: View Business Stats



The Use Case Diagram shows how the manager interacts with the system and how the system reacts back such as finding the profit and loss statements as well as editing the expenses and salaries of employees.

USE CASE UC-3: MANAGE QUEUE	ST-Ch-1, ST-Ch-2, ST-W-1, ST-BT-1
Related User Stories::	
Initiating Actor:	Waiter
Actor's Goal:	To add an order to the queue
Participating Actors:	Chef, Bartender
Pre-Conditions:	The waiter using the order screen to select the items that a seated party orders
Post-Conditions:	The order progresses through the various stages from “not started” to “ready for pickup”

Flow of Events for Main Success Scenario:

→ 1) Waiter acknowledges customer's desired item, and goes to the "Place Order" screen. adds each item to the list of the order, then submits

← 2) System displays the list of the various item classes (entree, dessert, beverages, appetizer).

→ 3) Waiter clicks on the class of the customer's desired item

← 4) System displays the list of items within each class. A "view order" option is present

→ 5) Waiter clicks on the item to add to the order, and option "ingredients" and "add note" is displayed next to each item

← 6) System adds the item to the list of the whole table's orders.

→ 7) Waiter repeats steps 1,3, and 5 until all of the table's desired items are in the order, then presses the "Place Order" option, which will add the order to the queue.

← 8) The system shows the whole queue, which can be navigated via scrolling. Each order that has been placed through the waiter's device will have an option to cancel the order or revise the order, as well as a progress status. Orders are added to the end of the queue when they are placed. Items that are prepared at the bar will be sent to the bartenders' devices, and items that the kitchen is responsible for will be sent to the chefs' devices.

→ 9) Chef or bartender finish their current task, then check the queue to see what needs to be prepared next.

→ 10) The chefs and bartenders can also see the queue. When an order is pressed, it will go to a new screen that shows each item and several options are displayed. The options are to set the progress status of the whole order. These options will be "not started", "in-progress", "on hold", or "ready for pick-up". They will set the item that they are preparing to "in-progress". There is also an option for "recipe"

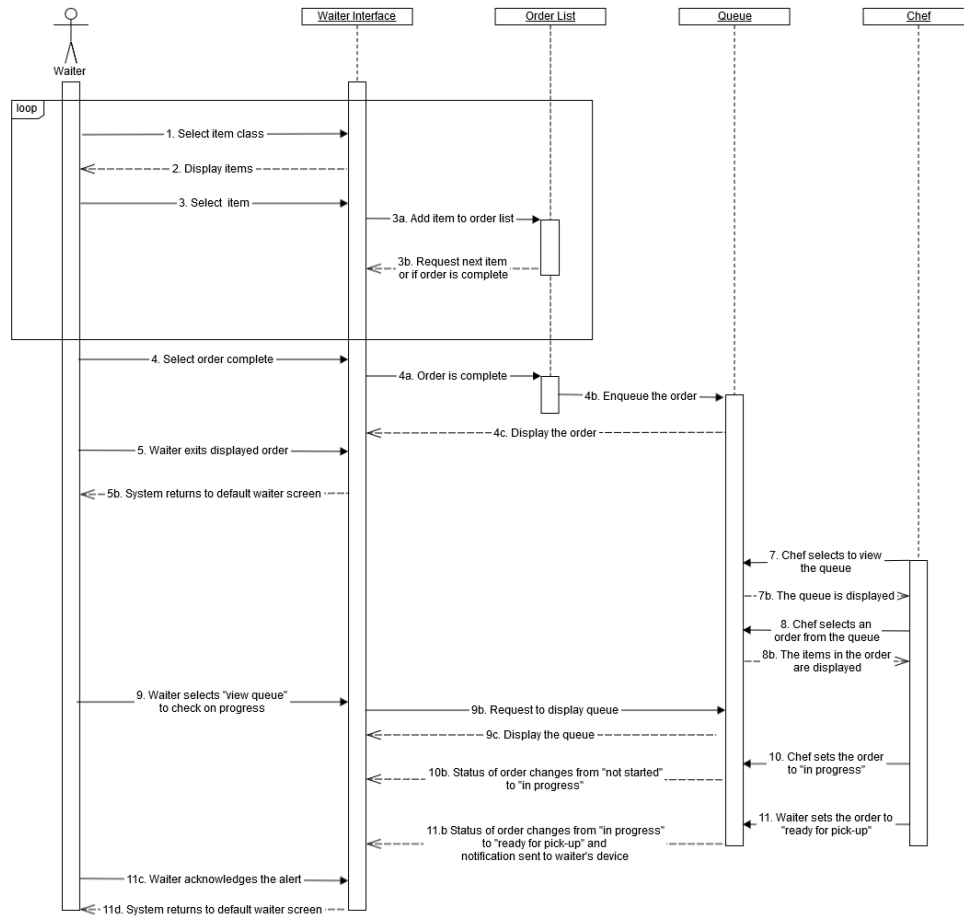
→ 11) Upon successful completion and delivery to the waiter pick-up area of the kitchen or bar, they will set the status to "ready for pick-up".

← 12) The device of the waiter who placed the order that is complete will be alerted via a notification, and they can retrieve the order to bring to the customers.

Flow of Events for Extension(Alternate Scenarios):

- 3a) Waiter can select “back” to return to the item class screen in the event that they selected the wrong item class
- ← System will return to the item class selection screen
- 4a) Waiter selects the “view order” tab to see all items.
- ← System will display the order list
- Waiter selects the “add note” in order to list details such as the rareness of steak or exclusion of certain ingredients
- ← System will add the item to the order list, with the note displayed under the item in the order list
- 5a) Waiter clicks on the item multiple times to increase the quantity of the item
- ← System will show multiple instances of the item in the order list
- 5b) Waiter selects the ingredients option because the customer inquired about the ingredients in a recipe for allergy/dietary reasons
- 10a) The chefs forgets the exact recipe for the item, and selects the “recipe”
- ← System displays ingredients and instructions for making the item

Use Case UC-3: Manage Queue



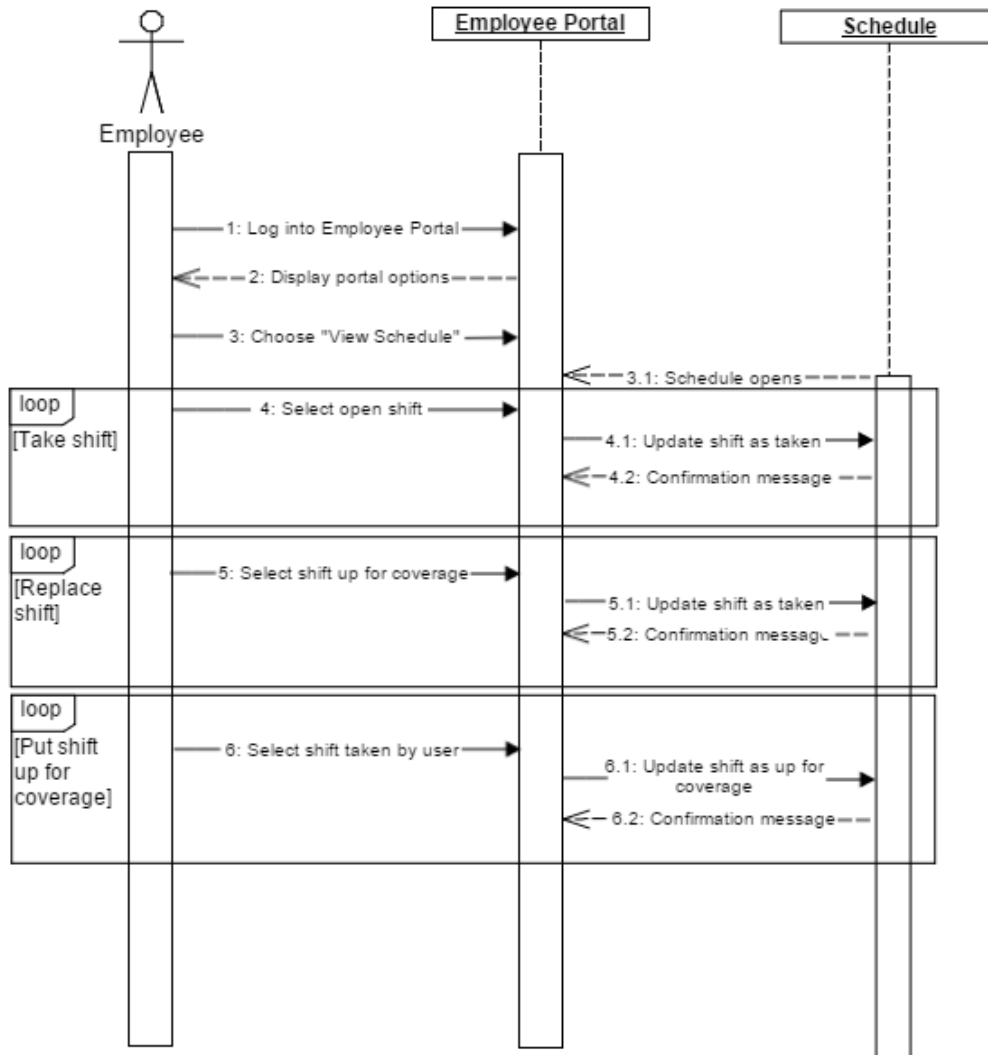
The Use Case Diagram shows how the waiter interacts with the system and how the system reacts back such as adding items into the queue and looking at the status also viewing the queue.

USE CASE UC-4: MANAGE SHIFTS	ST-G-1, ST-G-2, ST-G-4 to ST-G-6
Related User Stories::	
Initiating Actor:	Bartender, Busboy, Chef, Take-out Cashier, Waiter
Actor's Goal:	To add, replace, or request coverage for shift in the work week.
Participating Actors:	None
Pre-Conditions:	User has logged in as staff and has the "Employee Portal" open.
Post-Conditions:	Shift information has been updated and is visible by all users with access.
Flow of Events for Main Success Scenario:	<p>→ 1) User selects the "View Schedule" option.</p> <p>← 2) The system shows the schedule for the next two weeks by displaying all shifts in that time period – including those which are taken, up for coverage, or need to be filled.</p> <p>→ 3) User selects a shift which needs to be filled and clicks the corresponding timeslot.</p> <p>← 4) The system gives a confirmation that the shift is now theirs and updates the information accordingly in the backend which will show for any user on the employee portal in the future.</p>

Flow of Events for Extension(Alternate Scenarios):

→ 1a) User selects the “View Schedule” option.
← 2) The system shows the schedule for the next two weeks by displaying all shifts in that time period – including those which are taken, up for coverage, or need to be filled.
→ 3) User selects a shift which is taken by them and clicks the option “Put up for coverage.”
← 4) The system gives a confirmation that the shift is now up for coverage and updates the information accordingly in the backend which will show for any user on the employee portal in the future.
→ 1b) User selects the “View Schedule” option.
← 2) The system shows the schedule for the next two weeks by displaying all shifts in that time period – including those which are taken, up for coverage, or need to be filled.
→ 3) User selects a shift which is up for coverage and clicks “Fill.”
← 4) The system gives a confirmation that the shift is now theirs and updates the information accordingly in the backend which will show for any user on the employee portal in the future.

Use Case UC-4: Manage Shifts



The Use Case Diagram shows how any employee interacts with the system and how the system reacts back with managing their shifts with the database describing how they log in and update shifts and what messages will be displayed.

USE CASE UC-5: MANAGE MENU	ST-M-1, ST-M-6, ST-M-7, ST-Ch-4, ST-Ch-6, ST-BT-2, ST-BT-3, ST-BT-4
Related User Stories::	
Initiating Actor:	Managerial Staff, Chefs
Actor's Goal:	To modify the menu based on various different reasons.
Participating Actors:	None.
Pre-Conditions:	User has the "Edit Restaurant Menu" screen open.
Post-Conditions:	Restaurant menu is updated with user's changes and modifications
Flow of Events for Main Success Scenario:	<p>→ 1. Persons who have privileges to edit the restaurant menu select the "Edit Restaurant Menu" option in the "View Menu" screen</p> <p>← 2. Internal system displays existing menu with multiple options for editing purposes: add or delete menu items to each category, modify the ingredients in a dish (menu subject to changes due to ingredient availability), create new categories</p> <p>→ 3. User can modify category names and add or delete items from specific categories</p> <p>← 4. Internal system displays qualities of each menu item within the categories including the price, name, ingredients used, and ingredient count.</p> <p>→ 5. Persons who have privileges selects the "Add New Item" option</p> <p>← 6. Internal system prompts the user to enter information about the new item including "Name", "Price", "Ingredients Used", "Inventory Count", and "Customer Favorite".</p> <p>→ 7. Persons who have privileges clicks "Save and Add Item to Menu".</p> <p>← 8. Internal system adds the item to the menu based on price.</p> <p>→ 9. User saves changes and clicks "Return to Main Menu" after completion of editing the menu.</p> <p>← 10. Internal system returns to main menu in FoodEZ application interface.</p>

Flow of Events for Extension(Alternate Scenarios):

→ 2a. Persons with privileges selects "Delete Menu Category".

← Internal system prompts user with "Deleting X menu category will result in a loss of all proceeding menu items within X menu category. Will you proceed to delete X category?", where X is a category name. System will prompt the user to choose "Accept" or "Decline".

→ User selects "Accept".

←System deletes category, as well as all menu items within the category, and returns back to the editing menu.

→ 3a. Users select "Delete Item" within a category.

← System prompts user with "Are you sure you want to delete item: X", where X is an item name. System will prompt the user to chose "Accept" or "Decline".

→ User selects "Accept".

← System will automatically delete the item, and returns back to the editing menu.

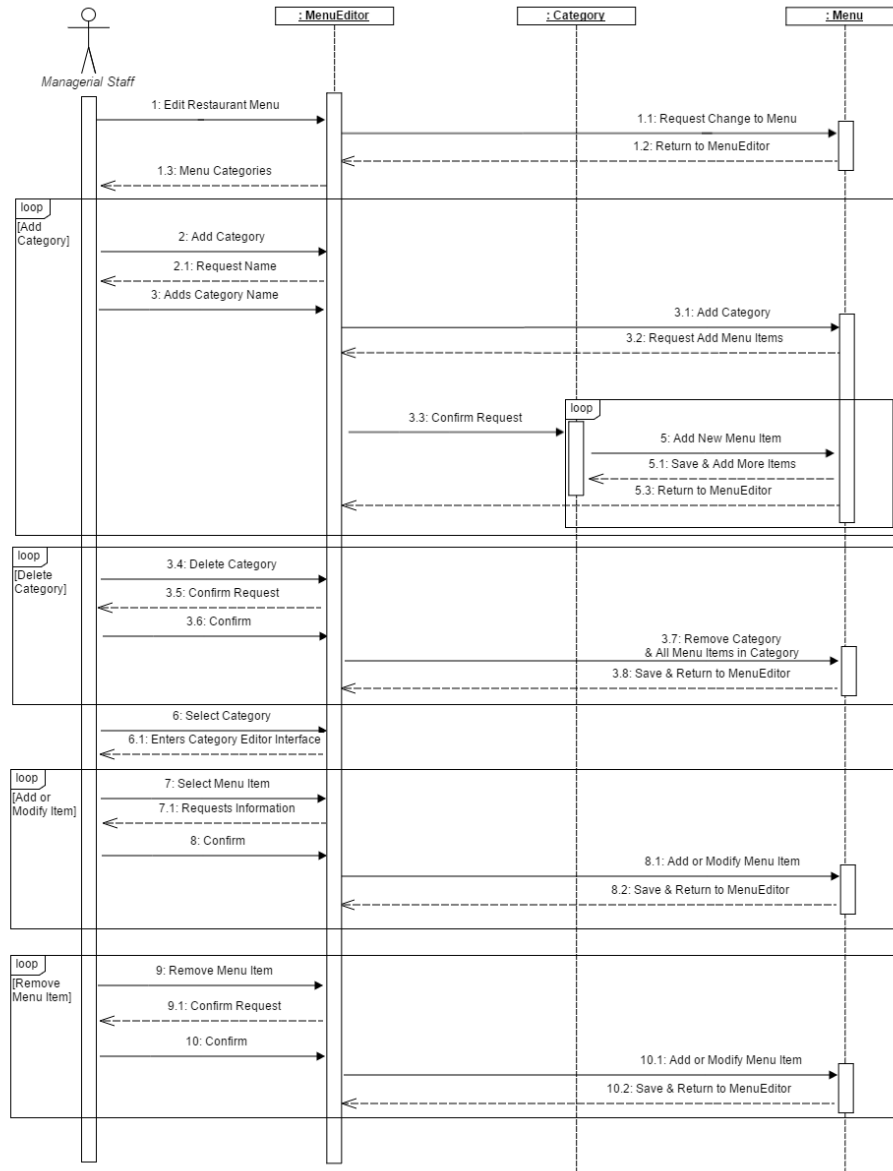
→ 4a. User selects "Edit Menu Item Properties", where user can alter the price, name, ingredients used, and ingredient count.

← Internal system prompts user with textboxes to modify the price, name, ingredients used, and ingredient count of the menu item.

→ Persons with privileges edits the appropriate information and clicks "Update All".

← System will update the edited information and return to the editing menu.

Use Case UC-5: Manage Menu



The Use Case Diagram shows how the manager interacts with the system and how the system reacts back with updating the menu.

User Effort Estimation:

Scenario 1: Waiter Places Order

1. Navigation: Select the option to go to menu.
 - a) Select the desired item category
 - b) Select the desired item, tap more than once to increase quantity
 - i. Waiter may choose to remove certain ingredients at customers request, or attach a brief note to the item.
 - c) Once all desired items are in the order, select “place order” to add the order to the queue.

Scenario 2: Chef Prepares Order

2. Navigation: Select the option to go to the queue.
 - a) Chef selects an order from the queue, the items within the order (including attached notes or ingredients alterations) are displayed
 - b) Chef selects “in progress” once the order preparation is about to begin
 - c) Chef selects “complete” once the order is ready for pick-up

Scenario 3: Busboy Cleans Table

3. Navigation: Select the option to go to the table layout.
 - a) Busboy selects a table that is “ready for cleaning” and click on it once to change its status to “cleaning in progress”
 - b) Busboy clicks on it once more to change its status to “ready for seating” upon completion of cleaning

Scenario 4: Customer enters their party into the seating queue

4. Navigation: Select “Add my party”.
 - a) Customer enter his/her name into the text box labeled “Party leader”.
 - b) Customer enter his/her party size into the text box labeled “Party size”.
-

$UCP = UUCP \times TCF \times ECF$

Weight: 5 -> Simple
 10 -> Medium
 15 -> Difficult

UC	Use Case Weight
UC -1: Place Order	5
UC -2: View Business Stats	10
UC -3: Manage Queue	15
UC -4: Manage Shifts	15
UC -5: Manage Menu	5

$UUCP = 50$

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

$TCF = 0.6 + (Total\ Complexity\ Factor \times 0.01)$
 $= 0.6 + (28 \times 0.01)$
 $= 0.88$

ECF	Environmental Complexity Factor weight	ECF perceived impact	Complexity Factor
-----	--	----------------------	-------------------

ECF	Environmental Complexity Factor weight	ECF perceived impact	Complexity Factor
1	0.5	2	1
2	1	3	3
3	2	4	8
4	2	4	8
5	0.5	2	1

$$\begin{aligned} \text{ECF} &= 1.4 - (0.03 \times \text{ECF}) \\ &= 1.4 - (0.03 \times 21) \\ &= 0.77 \end{aligned}$$

$$\begin{aligned} \text{UCP} &= \text{UUCP} \times \text{TCF} \times \text{ECF} \\ &= 50 \times 0.88 \times 0.77 \\ &= 33.88 \end{aligned}$$

$$\begin{aligned} \text{Duration} &= 33.88 \times 28 \\ &= 948.6 \text{ Hours} \end{aligned}$$

Domain Analysis:

1) Domain Model:

i) Concept Definitions:

Responsibility Description	Type	Concept
R-01: Coordinates scheduling, views profit/losses, delegates work to other employees	D	Manager
R-02: System knows all expenses and profits and losses	K	Profit/Loss
R-03: Can edit expenses of store	D	Manager
R-04: Customer accesses the menu online and places orders.	D	OrderItem
R-05: Knows all orders from all customers/tables.	K	OrderQueue
R-06: System sends orders to the restaurant.	D	Order
R-07: System knows and displays status of online order for customer viewing	K	OrderStatus
R-08: System knows when customer provides an incorrect order ID	K	Order
R-09: Change employee information including position, status, wage, contact information, etc.	D	InfoChanger
R-10: Modify menu	D	MenuModifier
R-11: Place food orders within the restaurant	D	Waiter
R-12: Place drink orders within the restaurant	D	Bartender

R-13: System knows count of ingredients	K	IngredientCount
R-14: System adds item to list of entire table's orders	D	Check
R-15: Waiter places table's orders	D	Order
R-16: System tracks order status within restaurant	K	OrderStatus
R-17: Chefs update order status	D	OrderStatus
R-18: System order status changed to "ready for pick-up"	K	OrderStatus
R-19: System keeps track of entire table's bill and tip calculations	K	Check
R-20: Waiter returns to system to retrieve check	D	Check
R-21: System receives payment and notifies busboy to clean table	D	CleanTable
R-22: Busboy alerts system upon table cleaning completion	D	CleanTable
R-23: System knows schedule of all employees	K	Schedule

ii) Association Definitions:

Concept Pair	Association Description	Association Name
Manager ↔ MenuModifer	Allows manager to modify menu	Modifies
Manager ↔ Profit/Loss	Manager tracks profit gain and profit loss	Provides Data
Manager ↔ IngredientCount	Manager requests updates on IngredientCount to manage menu items	Requests Updates
Manager ↔ InfoChanger	Manager alters confidential employee information	Modifies
Customer ↔ Waiter	Customer passes order requests to waiter	Conveys Requests
Waiter ↔ Chef	Waiter passes order requests to chef	Conveys Requests
Customer ↔ OrderItem	Customer adds a menu to the bill, and places the order.	Conveys Requests/Requests Save
Check ↔ OrderQueue	Bill provides the data of the ordered items to the OrderQueue	Provides Data
Waiter ↔ OrderStatus	Waiter periodically requests updates to OrderStatus	Requests Updates
Chef ↔ OrderStatus	Chef passes updates to OrderStatus	Provides Data
Waiter ↔ Check	Waiter retrieves and requests check for appropriate table	Conveys Requests/Provides Data
Waiter ↔ Customer	Waiter passes check and requests payment	Provides Data/Conveys Requests
BusBoy ↔ Check	BusBoy prepares cleans table once check is paid	Prepares
Waiter ↔ Tables	Waiter updates the the status of tables	Provides Data

iii) Attribute Definitions:

Concept	Attributes	Attribute Description
Manager	Name	Name of Manager
	Manager ID	Manager has a unique identification credential associated with login
	Privileges	Manager has specific privileges including menu modifications, hire/terminate employees, track profit/loss, etc.
Waiter	Name	Name of the waiter
	Waiter ID	Each waiter has a unique identification credential associated with them
	Table Association	The tables to which the waiter is serving
Employee	Name	Name of the employee
	Identity	Each employee has a unique identification credential associated with them
	Contact Information	The employee's address, email address, phone number, etc.
	Position	Position held by the employee at the restaurant
Profit/Loss	Total Gain	Total amount of income made from restaurant transactions
	Total Expenses	Total amount of costs spent by restaurant
	Net Profit/Net Loss	Profit or loss made by the restaurant
OrderQueue	Orders	Orders are placed on a queue within the system, which can be accessed by the chef

OrderItem	Name	Name of the menu item
	Cost	Price of the menu item
	Category	Category with which the menu item is associated
OrderStatus	Order Placed	OrderStatus changes to Order Placed
	Preparation	OrderStatus changes from Order Placed to Preparation
	Bake	OrderStatus changes from Order Placed to Bake (when applicable)
	Quality Check	OrderStatus changes from Bake to Quality Check
	Ready to Pick-Up	OrderStatus changes from Quality Check to Ready to Pick-Up
InfoChanger	Name	Allows for altering the name of Employee
	Wage	Allows for altering the payment
	Contact Info	Allows for altering the contact information of an employee
	Schedule	Allows for altering the schedule of an employee
	Position	Allows for altering the position of an employee
MenuModifier	Name of Item	Allows for altering the name of the selected item
	Ingredients	Allows for altering ingredients of a selected item
	Price	Allows for altering the price of a selected item
	Name	Name of the bartender

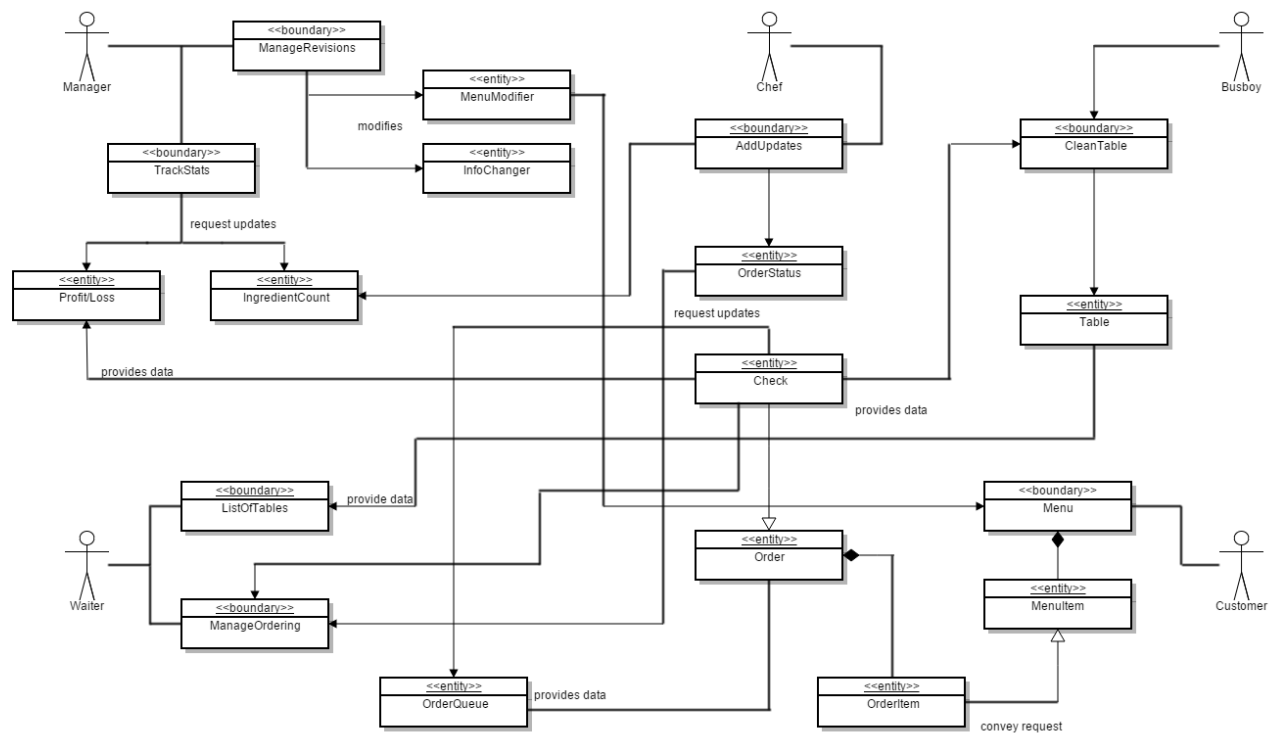
Bartender	Bartender ID	Each bartender has a unique identification credential associated with them
IngredientCount	Name	Name of the ingredient
	Total Stock	Amount of stock available in the kitchen for each ingredient
Order	Customer Information	Information about the customer who placed the order (if a takeout order)
	Order Number	The order has a unique identification number
	Table Number	The table number with which the order is associated
	Check	The most updated check for the order
Check	Order ID	The order ID with which the check is associated
	Number of Items	The total number of items ordered
	Total Cost	The total cost of all the items in the order
	Tip	Tip paid by customer
CleanTable	Status	Ready to be cleaned once customer departs
Schedule	Date	The various dates available on the schedule
	Time	The time slots available on each day
	Employee Name	Name of the employee working a certain shift

iv) Traceability Matrix

Use Case	PW	Domain Concepts														
		Manager	waiter	Employee	Profit/Loss	OrderItem	OrderQueue	OrderStatus	InfoChanger	MenuModifier	Bartender	IngredientCount	Order	Check	CleanTable	Schedule
UC1	9		X			X	X						X			
UC2	5	X		X	X									X		
UC3	8		X	X		X	X	X			X		X			
UC4	5	X	X	X	X				X		X					X
UC5	7	X		X						X		X				

The traceability matrix describes the relationship between the domain concepts we defined and the use cases implemented in the project. It shows how the use cases may contain many domain concept as well as the fact that a domain concept can be used by more than one use case.

v) Domain Model Diagram



System Operations Contract:

Name:	Customer Places Order
Responsibilities:	Order food from the restaurant using the application on their smartphone or tablet.
Use Cases:	UC-1
Exceptions:	None
Preconditions:	The application's GUI is displaying a menu with various food and drinks items, and an option to add them to bill and order them.
Postconditions:	The order is placed and added to the restaurant queue, after which its status is shown.

Name:	View Business Stats
Responsibilities:	View profit/losses and edit expenses of the store and salaries of employees.
Use Cases:	UC-2
Exceptions:	None
Preconditions:	Manager must be logged in. Screen displays GUI of FoodEZ and allows manager to enter the business stats section.
Postconditions:	Goes back to main menu so the manager can continue to do something else.

Name:	Manage Queue
Responsibilities:	Add an order placed by the customer to the queue.
Use Cases:	UC-3
Exceptions:	None
Preconditions:	The order screen on the Waiter's device allows him to select the items that a seated party orders.

Postconditions:	The order progresses through the various stages from “not started” to “ready for pickup”.
Name:	Manage Shifts
Responsibilities:	Add, replace, or request coverage for shift in the work week.
Use Cases:	UC-4
Exceptions:	None
Preconditions:	User has logged in as staff and has the “Employee Portal” open.
Postconditions:	Shift information has been updated and is visible by all users with access.

Name:	Manage Menu
Responsibilities:	Modify the menu based on various different reasons.
Use Cases:	UC-5
Exceptions:	None
Preconditions:	User has the "Edit Restaurant Menu" screen open.
Postconditions:	Restaurant menu is updated with user's changes and modifications

Mathematical Model:

Table Designation Algorithm

As customers enter the restaurant they are asked to specify their party size among other things into a tablet. This algorithm uses the party size and compares it against the available tables to see which table should be designated to the customer. The algorithm utilizes a sorted list (which is sorted and prioritized by a first come first serve basis) to take input from and then compares the party sizes of all customers and find a suitable table. If the table size matches the party size, then those two are matched, otherwise if the party size is less than the table size and no other party size matches the table then they are given the table.

Pseudo Code:

```
while (sorted list of customers does not equal 0)
{
    while (iterating through the list of available tables)
    {
        while (iterating through sorted list of customers)
        {
            if (size of table is equal to a party size in the customer list)
            {
                //Display customer name and table on the tablet
                //Along with map of restaurant highlighting the table
            }

            else if(size of table is greater than party size and no other party matches
table size)

            {
                //Display customer name and table on tablet
                //Along with map of restaurant highlighting the table
            }
        }
    }
}
```

Information Modification Algorithm

Whether it's hiring or terminating an employee, the manager will need the necessary options in order to modify information concerning an employee. Below gives a quick pseudocode algorithm on how a manager can edit and modify confidential information.

```
if (credentials entered match manager's credentials, allow access)
{
    //search for profile of employee manager wants to modify
    {
        while(1)
        {
            switch (manager chooses options)
            {
                case name:
                    //edit name of employee
                    break;
                case wage:
                    //edit wage of employee
                    break;
                case contact info:
                    //edit contact info of employee
                    break;
                case schedule:
                    //edit schedule of employee
                    break;
                case position:
                    //edit position of employee
                    break;
            }
            if (finished editing)
            {
                //save profile of employee
                //break and return to search screen
            }
            else
            {
                return to switch
            }
        }
    }
}

else
```

```
{
    print("Incorrect credentials.")
    return to login screen
}
```

Order Progress Queue Algorithm

Each order that is entered into the order queue most likely contains several items. Orders that have not yet been started have display an order status of “Not Started”. Upon beginning preparation of the items in an order, the order status associated with that specific order will become “In Progress (0%)”. Each item within the order also has a status, which can be either “Not Started” or “Complete”. During the time in which the order is “In Progress”, the percentage shown represents the amount of completed items within the order. Each time the chef marks an order as complete, the new percentage is displayed on the status bar of the order in the order queue. Once all items are complete, the order status changes to “Ready for Pick-Up”, at which time the device of the waiter who placed the order will receive a notification.

```
//assume order has data members numberComplete and numberIncomplete
//...and completionPercentage, as well as member function setPercent
//assume item has data members status and member functions setStatus

void changeItemStatus(order, item, newStatus)
{

    //if item was not started and is being started, change status

    if ( (if (item.status()=="Not Started") && (newStatus== "In
        Progress"))
    {
        item.setStatus("In Progress");
    }

    //if item was started and is now completed, change status and
    //... recalculate and display new completion percentage

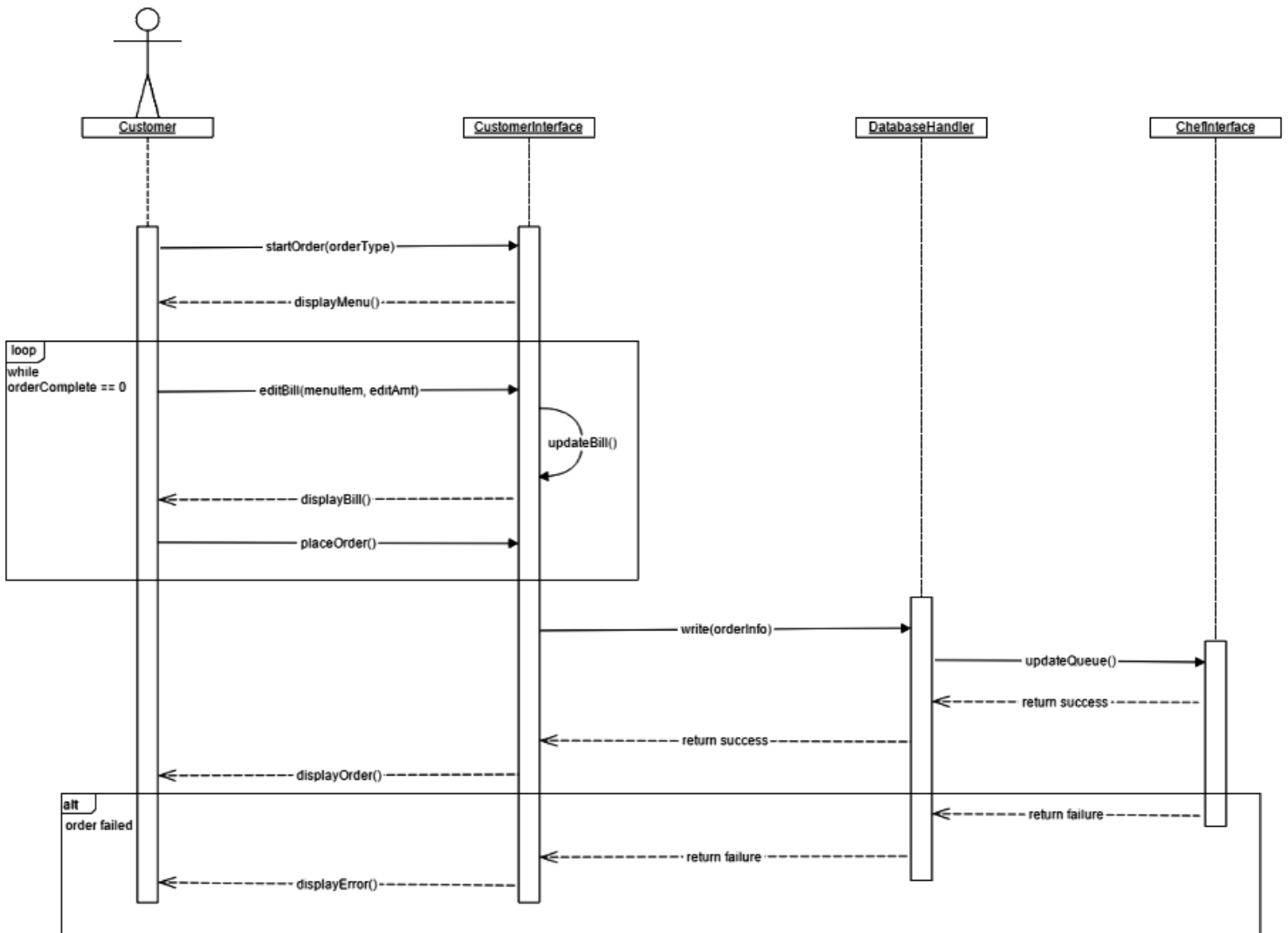
    if ( (if (item.status()=="In Progress") && (newStatus ==
        "Complete"))
    {
        item.setStatus("Complete");
        order.setPercentage( (order.numberComplete()/(order.numberComplete()
            +order.numberIncomplete()))*100 );
    }
}
```

```
//check if order is complete, if so, notify the waiter who  
//entered the order
```

```
if (order.completionPercentage==100)  
{  
    notifyWaiter(order);  
}
```

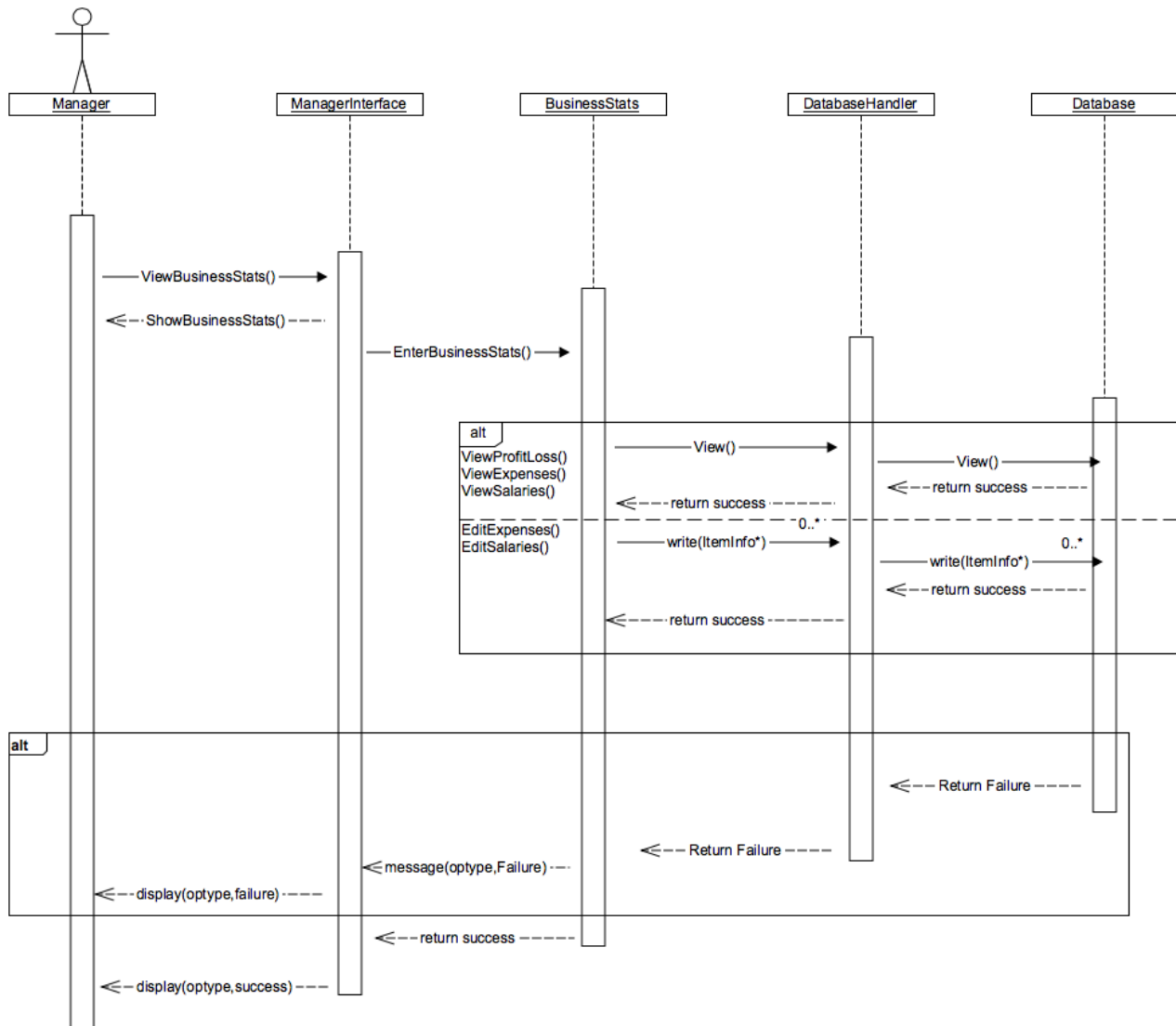
Interaction Diagrams:

UC1: Place Order



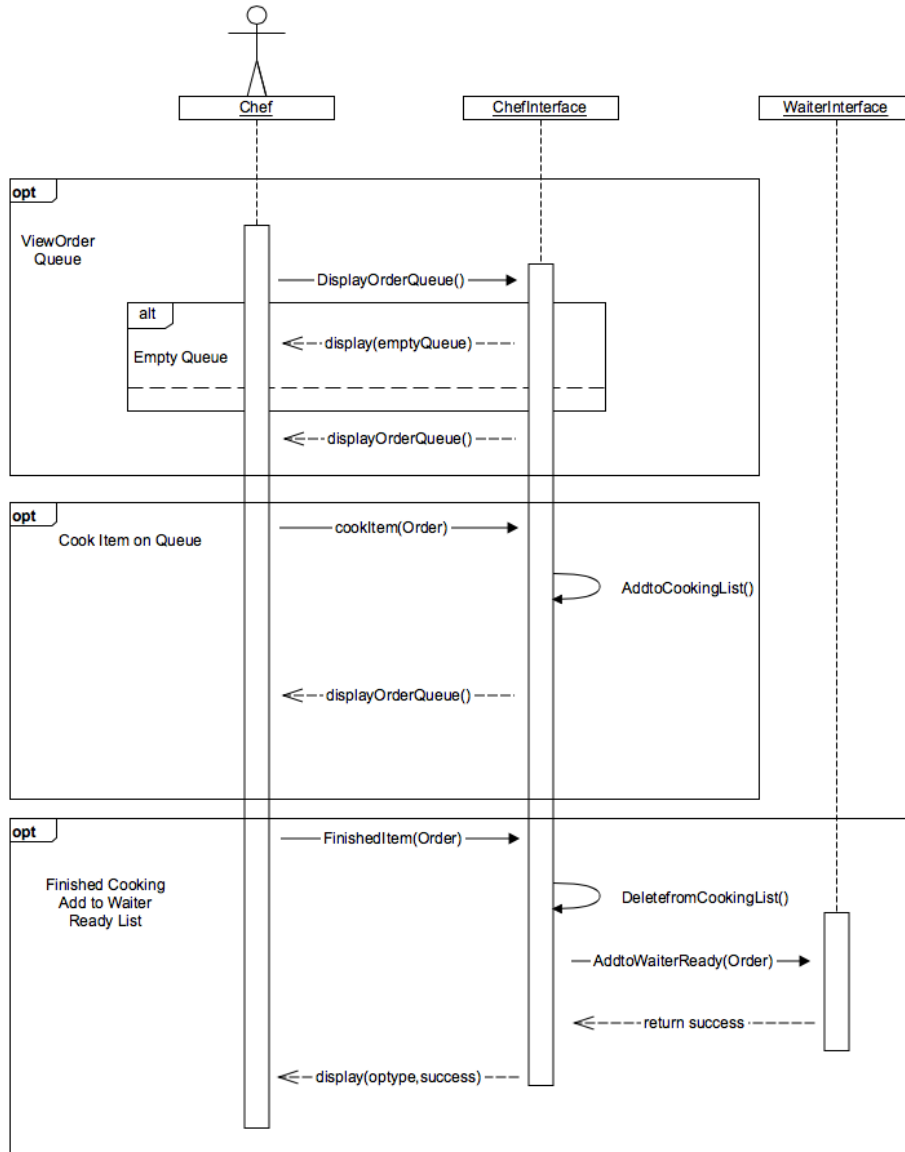
The Customer can start ordering by opening the GUI as a customer and the menu will be displayed. He can also edit it as long as his order has not been set complete. The order goes into the database and onto the queue for the waiter to be able to retrieve.

UC2: View Business Stats



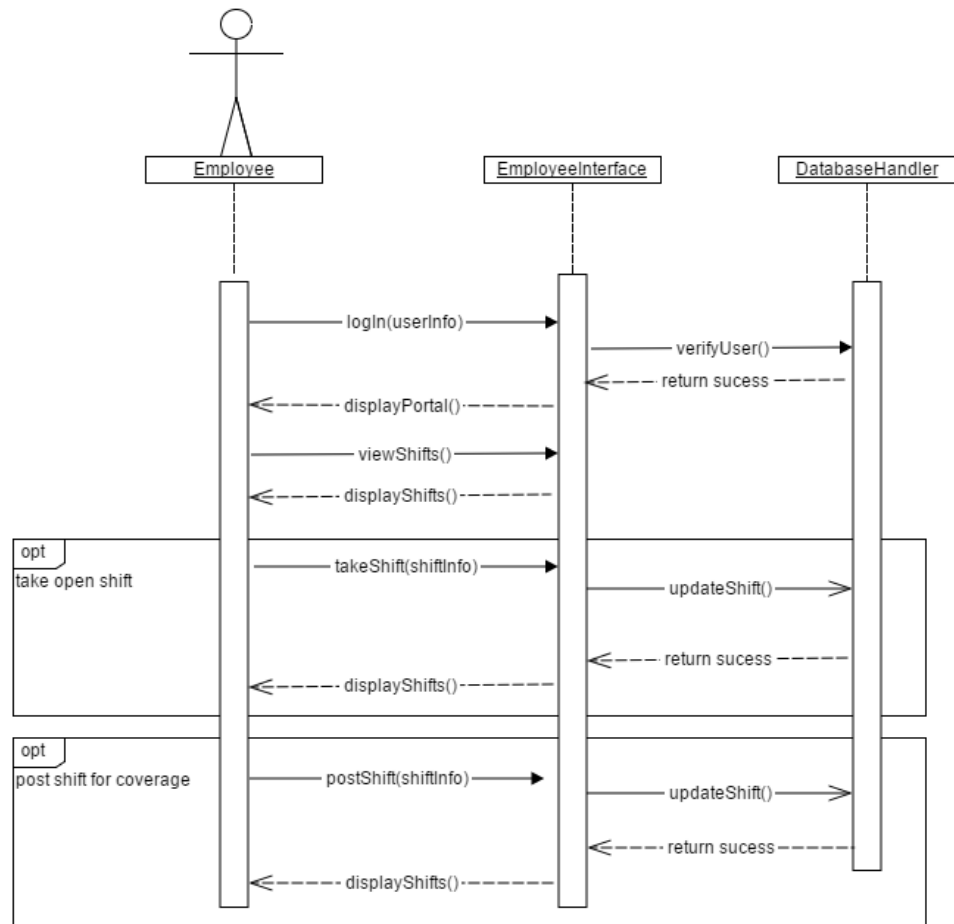
The manager can view the view business stats option from his GUI. Inside this option Manager can communicate with it by viewing or editing the salaries and expenses of the store. He can also view total Profit/Loss of the store. The information stored in and that comes out is all in the database that is held. The Program communicates efficiently with the database keeping everything up to date.

UC3: ManageQueue



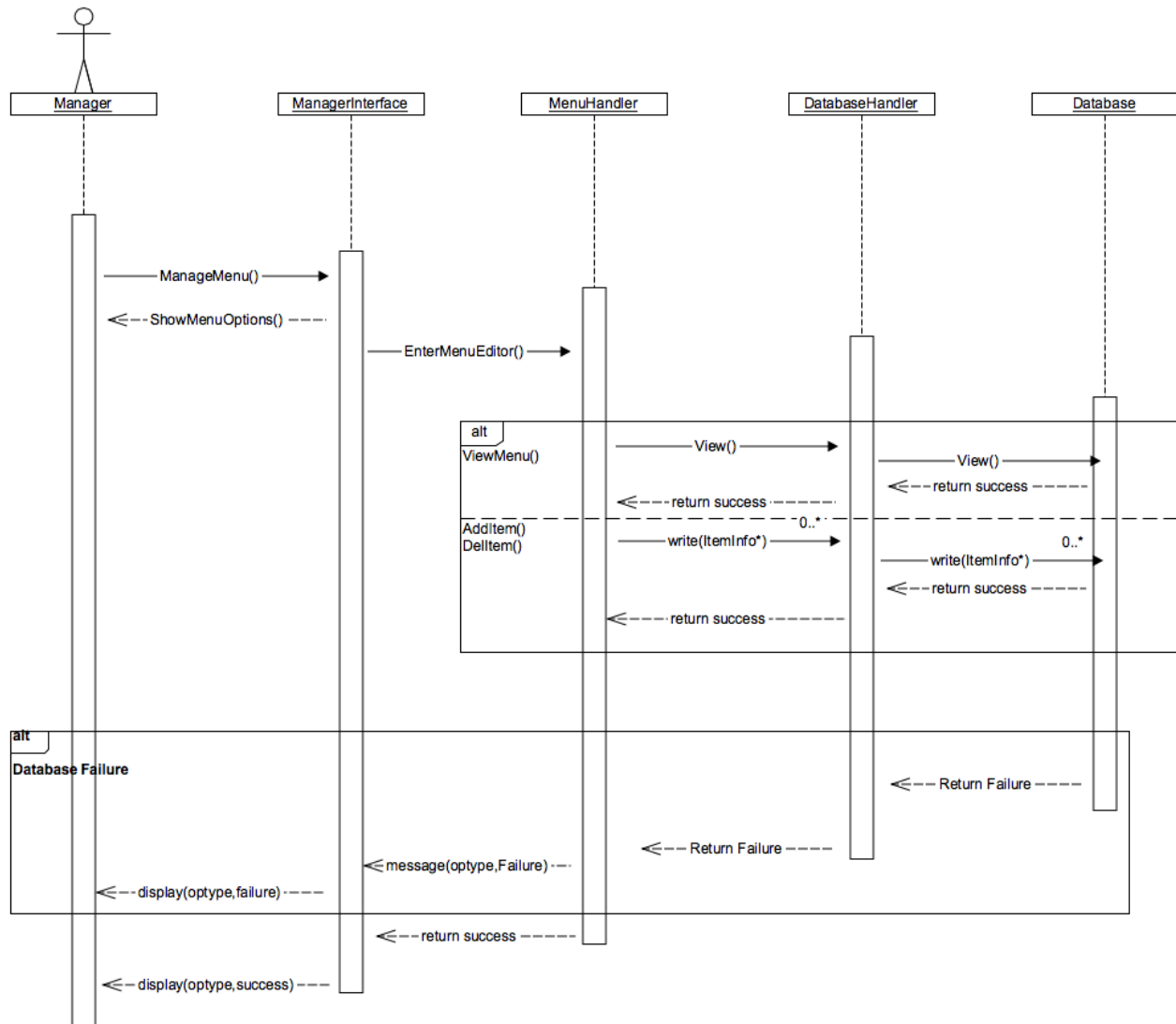
The Chef can view items in the queue added by the waiter or customer for order. The Chef can access his interface to view the queue. It will also return an empty error if there are no orders available. The chef can also interact with his interface by adding the item he will be cooking from the queue and thus adding it to the list of items in progress. After the chef finishes cooking he will Enter the finished item and thus remove form the list of Items in progress. This will also trigger communication with the waiter interface notifying him that the order is ready to be served.

UC4: ManageShifts



The Employee can access his interface and login to view his shifts which are saved on the database. He can also log which shifts he took or which shift he covered for or did extra time.

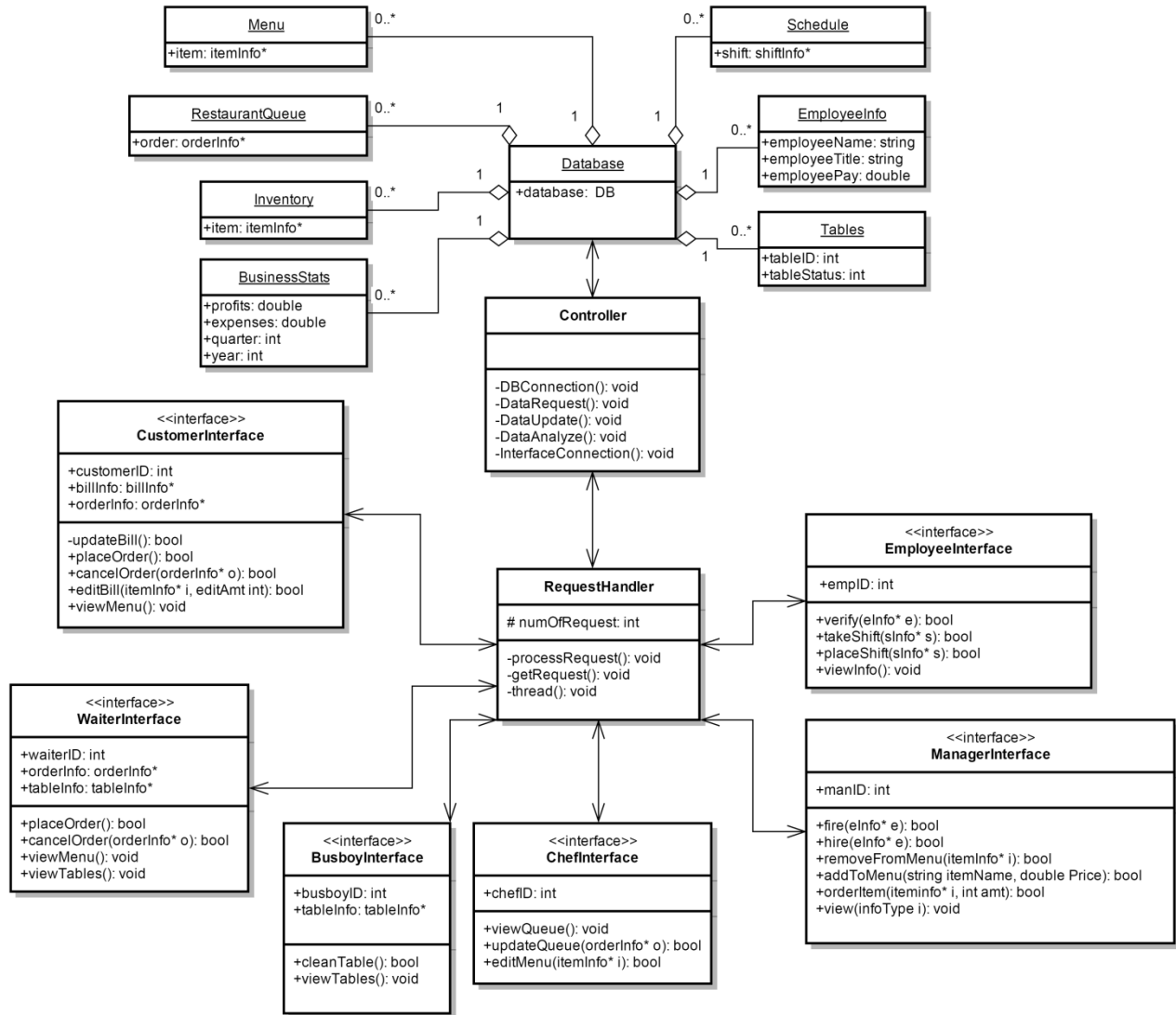
UC5: ManageMenu



The manager can interact with his interface to manage the menu. The manager has the ability to update the menu in the database whether by adding or deleting an item. The database will return the success showing the manager the success of his actions when doing so. It will also show failure in actions when deemed necessary or an error has occurred somewhere in the database.

Class Diagram and Interface Specification:

Class Diagram:



Class Descriptions:

The class diagram includes the following:

Menu - Menu is a class which contains all the menu items and each item's corresponding information.

RestaurantQueue - RestaurantQueue is a class which contains each table's food and/or drink orders, as well as specific information for each order.

Inventory - Inventory is a class that contains the quantity of items used in orders. Upon calling the function 'ItemList' within the Inventory class, it will display the current quantity of all items.

BusinessStats - BusinessStats is a class which contains functions that allow the Manager to view profits and expenses either quarterly or yearly.

Schedule - Schedule is a class that represents the timetable of all employees working during a given time. Upon calling the function 'shift' within the Schedule class, it will display which employees are working.

EmployeeInfo - EmployeeInfo is a class which contains information about each employee including the name, position, and pay rate.

Tables - Tables is a class which allows Busboys and waiters to view the status of specified tables.

Database - Database is a class which stores all information about FoodEZ including the schedule, employee information, inventory, business statistics, menu information, table information, etc.

Controller - The Controller requests, updates, and shares information between the RequestHandler and the Database.

CustomerInterface - This is the interface that is used by the customer. Through it, a customer is able to view and choose menu items to create their take out order. It contains attributes that give critical information such as the order items and the bill total.

RequestHandler - The request handler receives and processes all requests that come from the interfaces.

EmployeeInterface - The employee interface allows employees to view their shifts, to take on extra shifts or place unwanted shifts up for coverage. Each employee interface is tied to an employee via an ID number contained within the class.

Waiter Interface - The waiter interface is similar to the customer interface, in function however it is used by the employee to create orders for customers that are dining in the restaurant. Each employee interface is tied to an employee via an ID number contained within the class. Consequently, the waiter will be alerted when an order tied to their interface is completed.

BusboyInterface - The busboy interface shows all tables, along with their current status. The status is indicated by the color of the table as well as the text displayed on the table. The bus boy can view the whole restaurant layout with, as well as change the status of a table once its cleaning is “in progress” and when it is “completed”.

ChefInterface - The chef interface is utilized by the chef to prepare meals efficiently. Through it, he/she can view the queue and the contents of orders, as well as change the status of each meal as it goes from “in progress” to “completed”.

ManagerInterface - The manager interface is used to carry out essential tasks to the managing of the restaurant. Through it, the manager can fire or hire employees, track revenue and operating costs, alter the menu. The manager can also place orders for individual items through his/her interface that will be independent of any customer party’s bill, this is useful for customer services purposes when providing a free replacement or alternative item for customers who have complaints on meal preparation time, poor service, steak doneness, etc.

Data Types and Operation Signatures:

Customer Interface:

The first class, Customer Interface, is built so that a customer can interact with the system to place or cancel orders.

Attributes:

+customerID: Int	Integer variable corresponding to the customer's unique ID.
+billInfo: billInfo*	Struct containing information corresponding to the customer bill.
+orderInfo: orderInfo*	Struct containing information corresponding to the customer order.

Methods:

updateBill(): bool	Method updates the values in the billInfo* struct (e.g. the price change as the customer adds/removes items from an order).
+placeOrder(): bool	Method called to finalize an order and send the request to the order queue.
+cancelOrder(orderInfo* o): bool	Given a struct corresponding to an already placed order, this method requests that the order be canceled.
+editBill(itemInfo* i, int editAmt): bool	Will change the amount of the given item i by the amount given in editAmt on the customer's bill.
+viewMenu(): void	Method called to display the menu on the customer's interface.

Waiter Interface

The second class, waiter interface, is built such that a waiter has the ability to interact with the system to place orders, and find open tables.

Attributes:

+waiterID: int	Integer variable corresponding to the waiter's unique ID.
+orderInfo: orderInfo*	Struct containing information about the waiter's current customer's order.
+tableInfo: tableInfo*	Struct containing information about a certain table such as the status of the table and the table's id.

Methods:

+placeOrder(): bool	Method requests that the current order is placed to the queue.
+cancelOrder(orderInfo* o): bool	Given a struct corresponding to an already placed order, this method requests that the order be canceled.
+viewMenu(): void	Method called to display the menu on the waiter's interface.
+viewTables(): void	Method called to display the tables and their statuses on the waiter's interface.

Busboy Interface

The third class, busboy interface, is built such that a busboy has the ability to easily find dirty tables to clean.

Attributes:

+busboyID: int	Integer variable corresponding to the busboy's unique ID.
+tableInfo: tableInfo*	Struct containing information about a certain table such as the status of the table and the table's id.

Methods:

+cleanTable(): bool	Method called to request that a dirty table's status be changed to clean.
+viewTables(): void	Method called to display the tables and their statuses on the busboy's interface.

Chef Interface

The fourth class, chef interface, is built such that a chef can easily manage the queue and the menu.

Attributes:

+chefID: int	Integer variable corresponding to the chef's unique ID.
---------------------	---

Methods:

+viewQueue(): void	Method used to update the queue displayed on the chef's interface.
+updateQueue(orderInfo* o): bool	Method used to update the status of an order the restaurant queue represented by orderInfo* o.
+removeFromMenu(itemInfo* i): bool	Method to send a request to edit an item represented by itemInfo* i.
+addToMenu(string itemName, double Price): bool	Takes in an item name and its corresponding price. Creates the object itemInfo and adds it to the menu structure. Returns true if successfully added to the menu structure, false otherwise.

Manager Interface

The fifth class, manager interface, is built such that a manager can manage the menu, employees, and inventory.

Attributes:

+manID: int	An integer that is unique to each manager and is used to identify which manager is making changes, and which things he/she has can change.
--------------------	--

Methods:

+fire(eInfo* e): bool	Method used to update (remove) the employee database represented by eInfo* e
+hire(eInfo* e): bool	Method used to update (add) the employee database represented by eInfo* e
+removeFromMenu(itemInfo* i): bool	Method to send a request to edit an item represented by itemInfo* i.
+addToMenu(string itemName, double Price): bool	Takes in an item name and its corresponding price. Creates the object itemInfo and adds it to the menu structure. Returns true if successfully added to the menu structure, false otherwise.
+orderItem(itemInfo* i): bool	Takes in an object of item info as argument and requests to add it to the queue for preparation. Upon the order being marked as “complete”, the managers device is alerted. Returns true if order successfully added to the queue.
+view(itemInfo* i): bool	Displays the info for a given item, the string name of the item and the double price of the item

Employee Interface

The sixth class, employee interface, is built such that any employee can view their information and manage shifts (either placing their shifts for coverage or picking up).

Attributes:

+empID: int	An integer that is a unique ID number used to determine which employee is accessing the interface, and making changes to their scheduled shifts
--------------------	---

Methods:

+verify(eInfo* e): bool	Verifies the identity of the employee
+takeShift(sInfo* s): bool	Takes an argument sInfo (shift info), and adds it to the list of the specific employee's shifts, and removes it from the list of available shifts. Returns true, if successful, false otherwise.
+placeShift(sInfo* s): bool	Takes an argument sInfo (shift info), and removes it from the list of the specific employee's shifts, and adds it to the list of available shifts. Returns true if successful, false otherwise.
+viewInfo(): void	Displays the info on the current employee that is logged on.

Request Handler

The seventh class, request handler, is built such that multiple attempts to access the database from multiple interfaces is handle quick and concurrently using a multithreaded design.

Attributes:

#numOfRequest: int	Integer variable storing the number of request at a given time.
---------------------------	---

Methods:

processRequest(): void	Method used to send a request to the controller when the controller is ready to receive another request.
getRequest(): void	Method used to get an incoming request from a user interface.
thread(): void	Method used by threads to process multiple requests concurrently.

Controller

The controller class, sits between the request handler and the database and handles information sharing, and updating information in the database as well as in the user interfaces.

Methods:

DBConnection(): void	Method used to establish a connection with the database.
DataRequest(): void	Method used to request data from the database.
DataUpdate(): void	Method used to update data in the database and interfaces.
DataAnalyze(): void	Method used to run algorithms on data to analyze business stats, order queue efficiency, etc.

InterfaceConnection(): void

Method used to establish a connection with interfaces.

Database

The database object consist of multiple entries of different objects including tables, employee informations, schedule shifts, business statistics, items in the inventory, orders in the restaurant queue and items on the menu.

Attributes:

Database:database: DB	Database identifier (the ID of the database).
Tables:tableID: int	Integer variable that represent the unique ID of a table in the restaurant.
Tables:tableStatus: int	Integer variable that represents the current status of a table (0, 1 , 2, 3) corresponds to (Do Not Use, Open, In Use, Dirty).
EmployeeInfo:employeeName: string	String variable the represents the name of the employee represented by this object.
EmployeeInfo:employeeTitle: string	String variable the represents the title of the employee represented by this object.
EmployeeInfo:employeePay: double	Double value that tracks the pay that an employee receives
Schedule:shift: shiftInfo*	Object that contains the information for a specific shift, int day, int month, int year, int startHours, int startMinute, int endHours, int endMinutes.
BusinessStats:Profit: double	-A double that gives the value of the profits generated during this quarter, namely money and tips generated from sales and service.
BusinessStats:Expenses: double	A double that gives the value of all expenses incurred over this quarter, namely overhead, operating costs, inventory, and taxes.

Attributes:

BusinessStats:quarter: int	An integer value representing which quarter of the year this instance of BusinessStats represents
BusinessStats:year: int	An integer value representing which year this instance of BusinessStats represents
Menu:item: itemInfo*	Object that displays information about a menu item
RestaurantQueue:order: orderInfo*	Object that holds information about an order, int totalCost, string percentProgress, int itemCount, int waiterID
Inventory:itemList: list<iteminfo*>	List of all menu items which entails information about quantity, expiration date, type

Traceability Matrix

Domain Concepts	Software Classes															
	Menu	RestaurantQueue	Inventory	BusinessStats	Database	Tables	EmployeeInfo	Schedule	Controller	CustomerInterface	WaiterInterface	BusboyInterface	ChefInterface	ManagerInterface	EmployeeInterface	RequestHandler
Manager	x		x	x			x	x						x		
Waiter	x	x				x					x					
Employee	x	x						x							x	
Profit/Loss				x	x				x					x		x
OrderItem	x	x	x		x				x	x	x					x
OrderQueue					x				x	x	x		x			x
OrderStatus		x			x				x	x	x		x			x
InfoChanger					x				x					x	x	x
MenuModifier					x				x				x	x		x
Bartender	x					x										
IngredientCount			x		x				x				x	x		x
Order		x			x					x			x			
Check					x					x						
CleanTable					x	x			x			x				x
Schedule					x				x			x	x	x	x	x

The Concepts were derived very simply from the idea of a restaurant in sync with its employees. Everything is connected one way or another and it can be told by simply looking at the names of the classes. For example a Manager Interface was needed to put the concept of a manager and profit/loss and modifications in total. The rest of the classes were derived in the same manner with pure common sense.

Design Patterns:

The notification system uses the publisher-subscriber pattern to deliver notifications to all of the appropriate parties. When a notification is generated and sent to notificationHandler, the handler can parse the type of notification, its content, and its source and determine which interfaces need to receive the notification. It then publishes the notification out to those interfaces. For example, when the system detects that the stock level of an item in the inventory is empty, it would pass the name of the item whose stock has run out. It reads the input information and constructs the full notification message, then updates the rest of the interfaces. The notificationHandler will know each notification type and choose which subscribers should be notified. While notifications could have been implemented using direct communication, using the publisher subscriber pattern makes the system easier to maintain and update. If another user interface must be added for a new employee class, it is simple to include it in the subscribers list and include it in the notification system.

OCL: Object Constraint Language Contracts

```
waiterInterface:placeOrder() : bool
//The Waiter is the one placing the order
Invariants: self.Waiter -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Successful and pushed into Queue
Post-Conditions: placeOrder == true
```

```
waiterInterface:cancelOrder(orderInfo* o) : bool
//The Waiter is the one placing the order
Invariants: self.Waiter -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Deleted and removed from Queue
Post-Conditions: cancelOrder(orderInfo* o) == true
```

```
busboyInterface:cleantable() : bool
//The busboy is the one cleaning the table
Invariants: self.busboy -> True
//The table is dirty on it and not occupied
Pre-Conditions: table.occupied -> False
                table.dirty -> True
//Table Cleaned and updates table status
Post-Conditions: cleantable == true
```

```
chefInterface::addToMenu(string itemName, double Price): bool
//The chef is the one updating the menu
inv: self.chef -> True
//Item is not available on Menu
pre: MenuItem == NULL
//Update item in menu
post: addToMenu(string itemName, double Price) == true
```

```
chefInterface::removeFromMenu(itemInfo* i): bool
//The chef is the one updating the menu
inv: self.chef -> True
//Item is available on Menu
pre: MenuItem != NULL
//Removing Item succesful from menu
post: removeFromMenu(itemInfo* i) == true
```

```
managerInterface::fire(eInfo* e): bool
```

```
//The manager is the one firing the employee
inv: self.manager -> True
//employee is in database
pre: employeeinfo != NULL
//Removing employee succesful from database
post: fire(eInfo* e) == true
```

```
managerInterface::hire(eInfo* e): bool
//The manager is the one hiring the employee
inv: self.manager -> True
//employee is not in database
pre: employeeinfo == NULL
//Adding employee succesful from database
post: hire(eInfo* e) == true
```

```
managerInterface::addToMenu(string itemName, double Price): bool
//The manager is the one updating the menu
inv: self.manager -> True
//Item is not available on Menu
pre: MenuItem == NULL
//Update item in menu
post: addToMenu(string itemName, double Price) == true
```

```
managerInterface::removeFromMenu(itemInfo* i): bool
//The manager is the one updating the menu
inv: self.manager -> True
//Item is available on Menu
pre: MenuItem != NULL
//Removing Item succesful from menu
post: removeFromMenu(itemInfo* i) == true
```

```
managerInterface:placeOrder() : bool
//The manager is the one placing the order
Invariants: self.manager -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Succesful and pushed into Queue
Post-Conditions: placeOrder == true
```

```
managerInterface:cancelOrder(orderinfo* o) : bool
//The manager is the one placing the order
```

Invariants: self.manager -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Deleted and removed from Queue
Post-Conditions: cancelOrder(orderinfo* o) == true

System Architecture and System Design:

Architectural Styles

Our design architecture is based on the 2-tier client-server model in which a client directly connects with one integrated server. This kind of system works to establish unified infrastructure in that all correspondence and traffic runs through the server independent of the client attempting or requesting access.

The primary server acts as a database as well as a provider which fulfills requested services. From a database point of view, its role is to store information regarding employees, work schedules, and other details crucial to management. This is due to the implementation of the employee portal which lets clients login and adjust their individual particulars depending on access level.

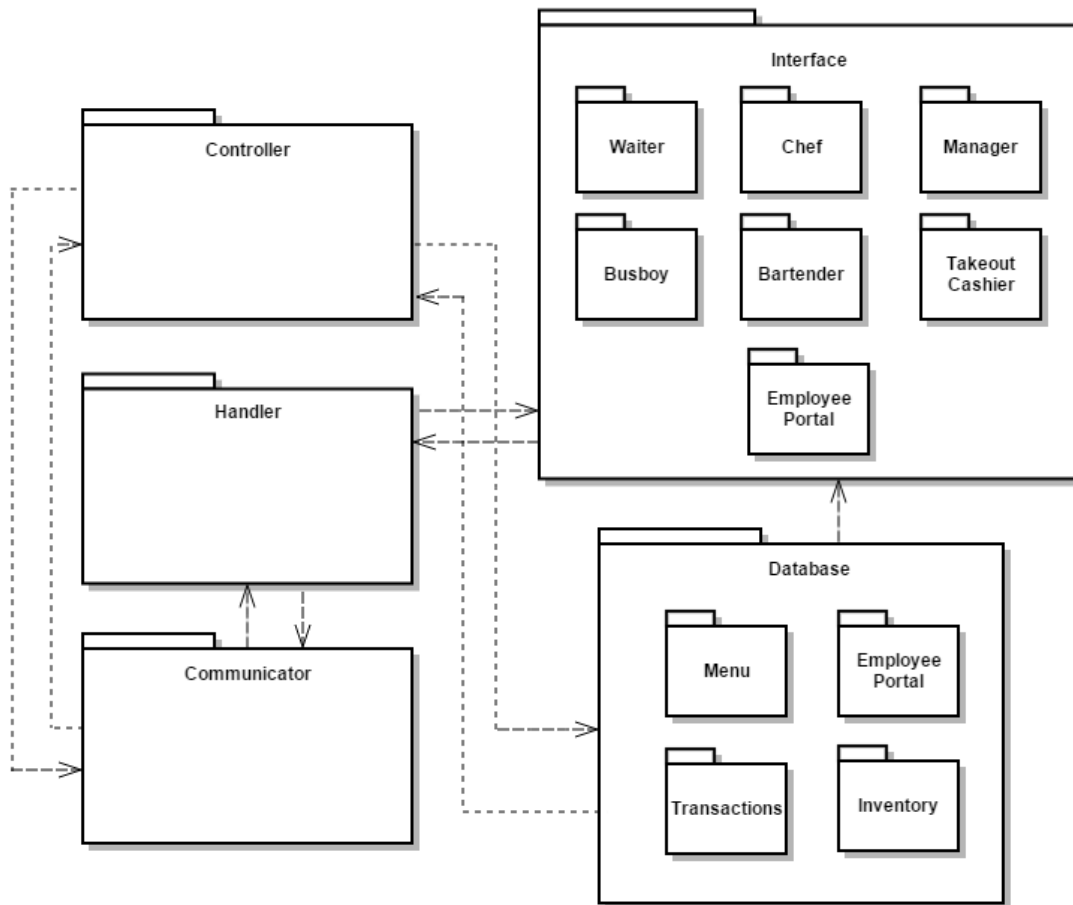
In the sense of restaurant automation, the system incorporates several different types of users (clients) and their corresponding functionality; credentials are first verified with the server followed by granting the client their appropriate permissions. An example of such a verification could be a manager logging into the system to make changes or verify paystub distribution. In another case, a waiter could request access in order to sign up for a shift in a certain work week.

The client-server model is supposed to be straightforward, but that is also one of its flaws. Due to its work processing allocation (i.e the server handles everything), constancy and uptime is affected. A fault in the server results in issues across the entire system. Compared to, for instance, component based architecture, client-server is essentially a one man's job.

A large positive is ease of maintenance in the future. By minimizing the number of objects interacting in the system, the need to maintain relationships between components is reduced. Client-server emphasizes this by leaving all work to the server.

Other than the database required for the employee portal, the server also keeps its local database which sends and receives information pertinent to the proceedings in the restaurant. With this system, the server (regardless indirectly or directly) handles cross-client requests. To illustrate this, a waiter may take a table's order; to relay it to the chef, the information is sent to the server which then proceeds to be accessible by the chef.

Identifying Subsystems



The package diagram consists of the composing elements in the system. Firstly, the interface package contains all types of employees and the portal as it presents different options and viewership depending on the user accessing its elements. The database holds all pertinent information for restaurant tasks, i.e transactions, inventory, and menu (changes). In addition, for the employee portal, it stores shifts, hours worked, and other related employee specific details. The controller handles flow of information transfer between the database and other packages which request or send anything. The communicator package, it acts as a medium through which the controller can receive authorization on interactions through package control. Lastly, the handler is responsible for spawning the interface and receiving information from the communicator about tasks that must be processed.

Mapping Subsystems to Hardware

Due to the nature of client-server architecture, the hardware hierarchy of the system is straightforward. Understanding that the system is running on the client-server model, the server runs on the master computer which has the database and all applicable data. Employees and management have tablets through which they access the restaurant automation application which all contain the communicator, interface processing, and controller. From a work allocation standpoint, this structure allows for each user to control what information they get (based on their permissions) and keeps the traffic/transfer of data simple.

Persistent Data Storage

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

Network Protocol

We will be implementing a Microsoft Azure Mobile Services SQL database which offers the ability to create highly functional mobile applications, such as FoodEZ. Microsoft Azure SQL database stores information in a cloud-based server, and allows for complete flexibility when programming for Windows Phones or tablets. By storing data on a cloud-based server, there is virtually no limit to the amount of space that we can use, our operation can be scaled according to current needs.

Global Control Flow

Execution Orderness:

“FoodEZ” is a procedure driven software. Practically everything executes in a linear fashion. The customer would enter the restaurant and push in the details (ex. Number of seats) allowing the program to seat him in a optimal position. Then the customer would make his order whether through tablet or waiters tablet. The order will be received by the Chef and when it is ready will notify the waiter to take the order directly to the customer. When the customer finishes eating he will pay the bill whether through the tablet or waiter and thus marking the table dirty for the busboy to come clean it. This clearly shows that all actions are made in a linear fashion all depending on the previous action.

Time Dependency

FoodEZ has an event-response time for the inventory alerts (Low Inventory), but for the rest of the system is is on real-time basis. This system is periodic as for everything done from being seated to paying the bill is done in a periodic manner. This is all time dependent because the time the customer makes the order places it in a queue for the chef to cook. Also calculation of wait time for the customer to receive the order is also on realtime basis. While as when the inventory is low will only respond when an item has reached a certain number.

Concurrency

FoodEZ will contain multiple threads, which will involve multiple systems running independently at once. Multiple customers will be placing orders at the same time which is why we need concurrency. The solution is to run multiple threads into the queue. Another reason is when an employee is editing the inventory or expenses. A thread will have to be spawned to handle the update in the database. Synchronization is not needed because each thread is independent of the other.

Hardware Requirement

With the portability of using a Windows Phone, employees and customers can now utilize all of the functionalities of FoodEZ in a small form factor. These types of functionalities require the following:

Windows Phone		
Hardware Component	Minimum Requirements	Recommended Requirements
Processor	1 GHz Dual-Core	1.5 GHz Dual-Core
RAM	512 MB	1 GB
Hard Drive Space	100 MB	200 MB
Network	Wi-Fi 802.11 b/g/n	Wi-Fi 802.11 a/b/g/n
Screen Size	4.3 inches	4.3 inches
Resolution	720 x 1280	720 x 1280

With a Windows tablet, employees and customers can now utilize all of the functionalities of FoodEZ. These tablets offer the same capabilities as the phone but in a more expansive form factor. These types of functionalities require the following:

Windows Tablet		
Hardware Component	Minimum Requirements	Recommended Requirements
Processor	1.8 GHz Quad-Core	2.0 GHz Dual-Core
RAM	1 GB	4 GB
Hard Drive Space	32 GB	64 GB
Network	Wi-Fi 802.11 b/g/n	Wi-Fi 802.11 a/ac/bg/n
Screen Size	7 inches	12 inches
Resolution	1280 x 800	2160 x 1440

Algorithms and Data Structures

Process and Prepare Orders

Once the system has validated a waiter's credentials, the system will allow the waiter to select the option to enter the menu. From there, the waiter can select the desired item category, desired item, increase or decrease quantity, and attach comments to the chef indicating specific requests made by the customer. After the order is ready to be sent to the chef, the waiter selects "place order" and the order will be sent to the chef's order queue.

Once the system has sent the order to the chef's order queue, the system displays the first order and its order's corresponding attributes (customer requests). When the chef begins preparing the order, the chef selects "in progress" to indicate that the order has begun preparing. After the order is completed and ready for pick-up, the chef selects "complete". Then, the system will notify the waiter that the order is ready for pick-up.

Pseudo Code:

```
// assume tableOrder is a queue of type itemInfo
// itemInfo is a struct containing info about the waiter's current table's order.
// assume menuItem is an object of type itemInfo
// assume the chef has already entered credentials and is viewing the chef's order queue on the chef interface.

if (credentials entered match the waiter's credentials, allow access to waiter interface)
{
    while (1) // allows the waiter to continuously view the waiter interface
    {
        // display waiter interface
        ...
        // assume that the waiter has selected "place a table's orders" within the waiter's
interface
        if (waiter selects "view menu" && entered a specific table number)
        {
            // waiter will be prompted with the entire FoodEZ menu
            ...
            // assume the waiter selected a specific menu item within the menu
            switch (waiter chooses different options for a selected menu item)
            {
case addItem:
// adds the menu item selected to the table's order
tableOrder.enqueue(menuItem)
break;
case addComment:
// adds a comment to the chef about the menu item
break;
case returnToMenu
// returns back to menu without adding item or comments
```

```
break;
    }
    if (waiter is finished entering the table's orders)
    {
        waiter.PlaceOrder();
        // waiter will place the completed table's order
        break;          // break from while loop
    }
    else if (waiter needs to place more menu items)
    {
        return to switch statement and continue adding items
    }
    else // if the waiter needs to access something else in the waiter UI
    {
        //break from switch statement
        return to the waiter interface
    }
}
}
```

Calculate Business Stats

When the manager is on the main menu, his login identification number is recognized as belonging to a person of authority and provides him the permission to access business data. Selecting the option to view business data will divert him to a screen that displays various metrics for determining how effectively the restaurant is running. The displayed values factor in incurred costs such as overhead (utilities, promotional costs, rent, etc), inventory, labor costs, as well as revenue generated from sales and gratuity. The algorithm that is documented here is the method through which the values in the “Business Data” page are displayed.

//assume that the database has several sections: overhead, inventory, labor, sales, tips, miscellaneous expenses.

```
recalculateAndDisplayBusinessStats(){
double revenue = 0;

    for (int i = newest entry in item sales; i == oldest entry in item sales; i++){
        revenue += entry i;
    }

for (int i = newest entry in tips; i == oldest entry in tips; i++){
    revenue += entry i;
}

double costs = 0;

for (int i = newest entry in inventory; i == oldest entry in inventory; i++){
    costs += entry i;
}

for (int i = newest entry in labor costs; i == oldest entry in labor costs; i++){
    costs += entry i;
}

for (int i = newest entry in overhead; i == oldest entry in overhead; i++){
    costs += entry i;
}

double profit = revenue-costs;

display("Total Revenue: $"<<revenue);

    display("Total Costs: $"<<costs);
display("Total Profit: $"<<profit);}
```

Algorithms we already discussed in our mathematical section in report 1:

Table Designation Algorithm

As customers enter the restaurant they are asked to specify their party size among other things into a tablet. This algorithm uses the party size and compares it against the available tables to see which table should be designated to the customer. The algorithm utilizes a sorted list (which is sorted and prioritized by a first come first serve basis) to take input from and then compares the party sizes of all customers and find a suitable table. If the table size matches the party size, then those two are matched, otherwise if the party size is less than the table size and no other party size matches the table then they are given the table.

Pseudo Code:

```
while (sorted list of customers does not equal 0)
{
    while (iterating through the list of available tables)
    {
        while (iterating through sorted list of customers)
        {
            if (size of table is equal to a party size in the customer list)
            {
                //Display customer name and table on the tablet
                //Along with map of restaurant highlighting the table
            }
            else if(size of table is > than party size and no other party matches table size)
            {
                //Display customer name and table on tablet
                //Along with map of restaurant highlighting the table
            }
        }
    }
}
```

Information Modification Algorithm

Whether it's hiring or terminating an employee, the manager will need the necessary options in order to modify information concerning an employee. Below gives a quick pseudocode algorithm on how a manager can edit and modify confidential information.

Pseudo Code:

```
if (credentials entered match manager's credentials, allow access)
{
    //search for profile of employee manager wants to modify

while(1)
{
    switch (manager chooses options)
    {
        case name:
            //edit name of employee
            break;
        case wage:
            //edit wage of employee
            break;
        case contact info:
            //edit contact info of employee
            break;
        case schedule:
            //edit schedule of employee
            break;
        case position:
            //edit position of employee
            break;
    }
    if (finished editing)
    {
        //save profile of employee
        //break and return to search screen
    }
    else return to switch

}
}

else
{
    print("Incorrect credentials.")
    return to login screen
}
```

Order Progress Queue Algorithm

Each order that is entered into the order queue most likely contains several items. Orders that have not yet been started have display an order status of “Not Started”. Upon beginning preparation of the items in an order, the order status associated with that specific order will become “In Progress (0%)”. Each item within the order also has a status, which can be either “Not Started” or “Complete”. During the time in which the order is “In Progress”, the percentage shown represents the amount of completed items within the order. Each time the chef marks an order as complete, the new percentage is displayed on the status bar of the order in the order queue. Once all items are complete, the order status changes to “Ready for Pick-Up”, at which time the device of the waiter who placed the order will receive a notification.

Pseudo Code:

```
//assume order has data members numberComplete and numberIncomplete
//...and completionPercentage, as well as member function setPercent
//assume item has data members status and member functions setStatus

void changeItemStatus(order, item, newStatus)
{
    //if item was not started and is being started, change status

    if ( (if (item.status()=="Not Started") && (newStatus== "InProgress"))
    {
        item.setStatus("In Progress");
    }
    //if item was started and is now completed, change status and
    //... recalculate and display new completion percentage
    if ( (if (item.status()=="In Progress") && (newStatus == "Complete"))
    {
        item.setStatus("Complete");
        order.setPercentage( (order.numberComplete()/(order.numberComplete()
            +order.numberIncomplete()))*100 );
    }
    //check if order is complete, if so, notify the waiter who
    //entered the order
    if (order.completionPercentage==100)
    {
        notifyWaiter(order);
    }
}
```


Data Structures

The system will be implemented using a few different data structures including priority queues, arrays, and hash tables. These will be used differently depending on the employee working and utilizing the application.

Waiters will be working with queues when attending to customers – particularly, this refers to how tables will be stored. Parties come in, get seated at a table, and that table is added to the waiter's queue structure to enable a FCFS (first come first served) order in the restaurant.

On the other hand, chefs use the priority queue to manage all orders that come in. This can be attributed to the idea that though tables are being handled in FCFS, the food may have to be cooked in a slightly different order. Considering that certain dishes take longer to cook than others, the priority queue allows the chef to perhaps begin cooking one dish before another even if the preceding dish was ordered by a party that came in at a later time.

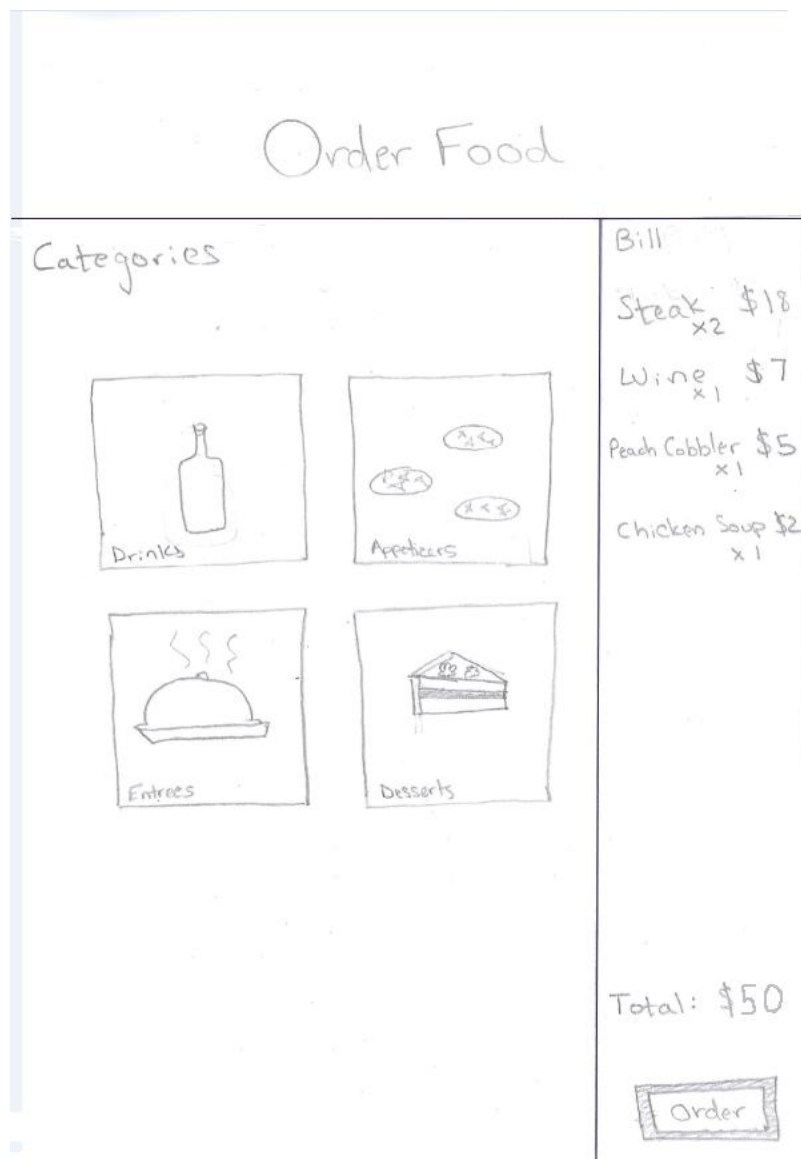
Menu items will be stored in an array. Looking at the variety of structures from a performance point of view, arrays clearly win for our purposes. The menu is important for performance due to its use in several different scenarios. Two quick examples of this would be customers browsing the menu and managers using it for statistical analysis on popularity of dishes and modifying the menu based on that. Both of these operations benefit greatly from high performance set, i.e the array.

In practical use, hash tables are generally preferred for fast data lookup due to their mapping nature. As a result, in the system, they will be implemented in the employee portal to map details of each employee (not limited to just payment and scheduling information). In such a case, each key in the hash table will represent an employee while the values correspond with pertinent details which can be accessed through the manager's portal.

User Interface and Implementation


Pre Evolution of UI


The user of the app will enter the main menu and be able to press order food. Allowing to pick a category and easily press the food of his/her choice while adding it to the bill simultaneously.




Order Food

Desserts

	<p>Cheesecake</p> <p>Nutritional Facts</p>	<input type="checkbox"/>
---	--	--------------------------

	<p>Peach Cobbler</p> <p>Nutritional Facts</p>	<input checked="" type="checkbox"/>
---	---	-------------------------------------

	<p>Hot Fudge Brownie</p> <p>Nutritional Facts</p>	<input type="checkbox"/>
---	---	--------------------------

Bill

Steak \$18
x2

Wine \$7
x1

Chicken Soup \$2
x1

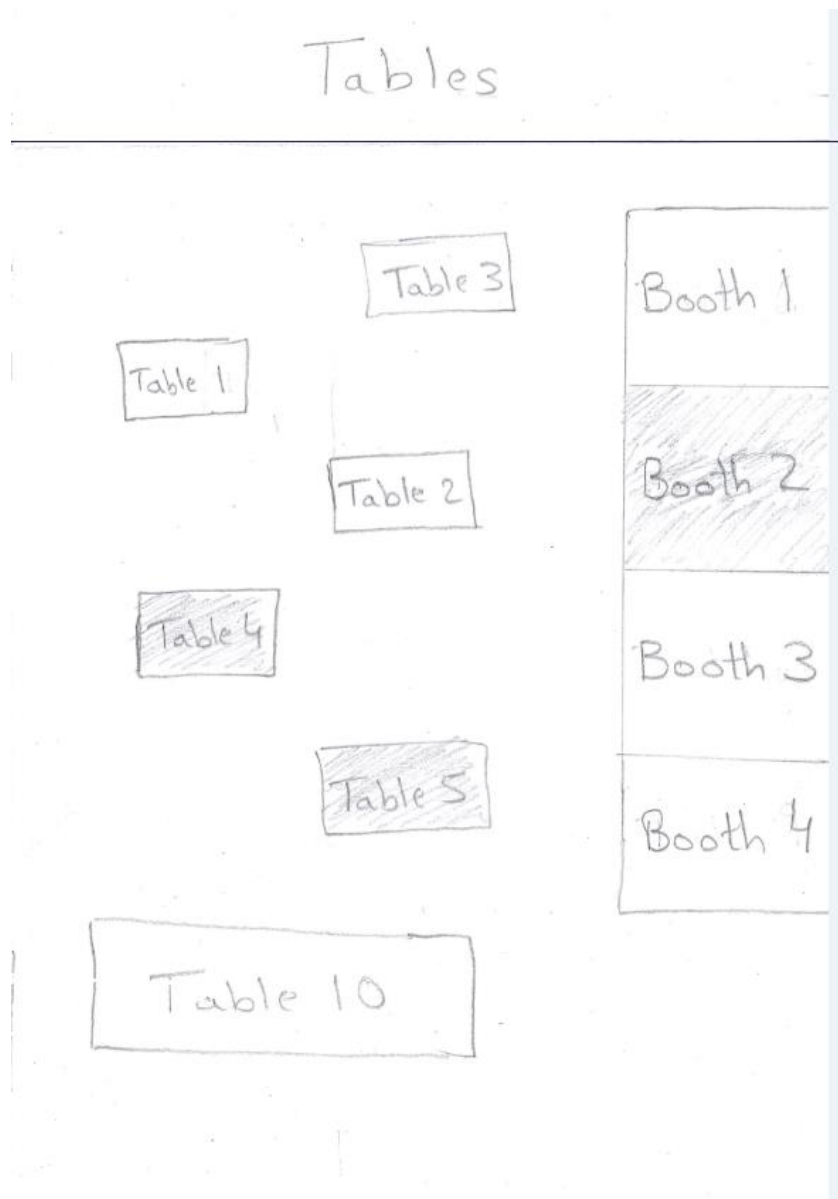
Add to Order

Total: \$45

The Queue will show which tables had ordered and be able to see which ordered first. The user of the app will also be able to press on the table of their choice to see what exactly this table has asked for and be able to update the progress of the table.

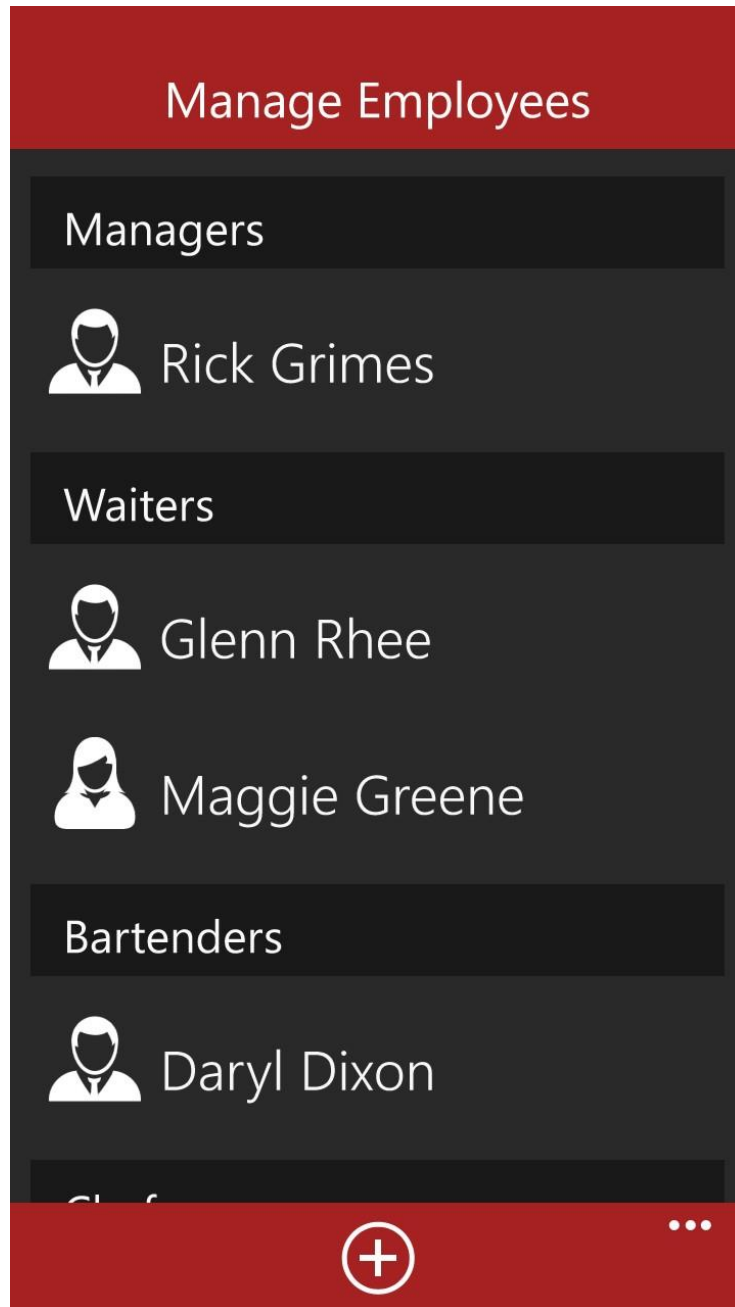
Queue		
Table 5		
Food Item	Qty.	Done
Steak	2	<input checked="" type="checkbox"/>
Peach Cobbler	1	<input type="checkbox"/>
chicken Soup		<input type="checkbox"/>

The waiters can also open from the main Menu the view of tables in the restaurant to view which is taken and which needs to be cleaned. Simply the user will press on View Tables and will be able to see it very clearly. Allowing him to edit the status of each table.




Full Evolution:

The User Interface of our application has gone through some changes, but the overall basis of the uses of each page has not changed much from the originally proposed design. Below is the manager's view of the employee list.



Employee Information







Personal Details Contact Info Work

First Name

Last Name

Gender

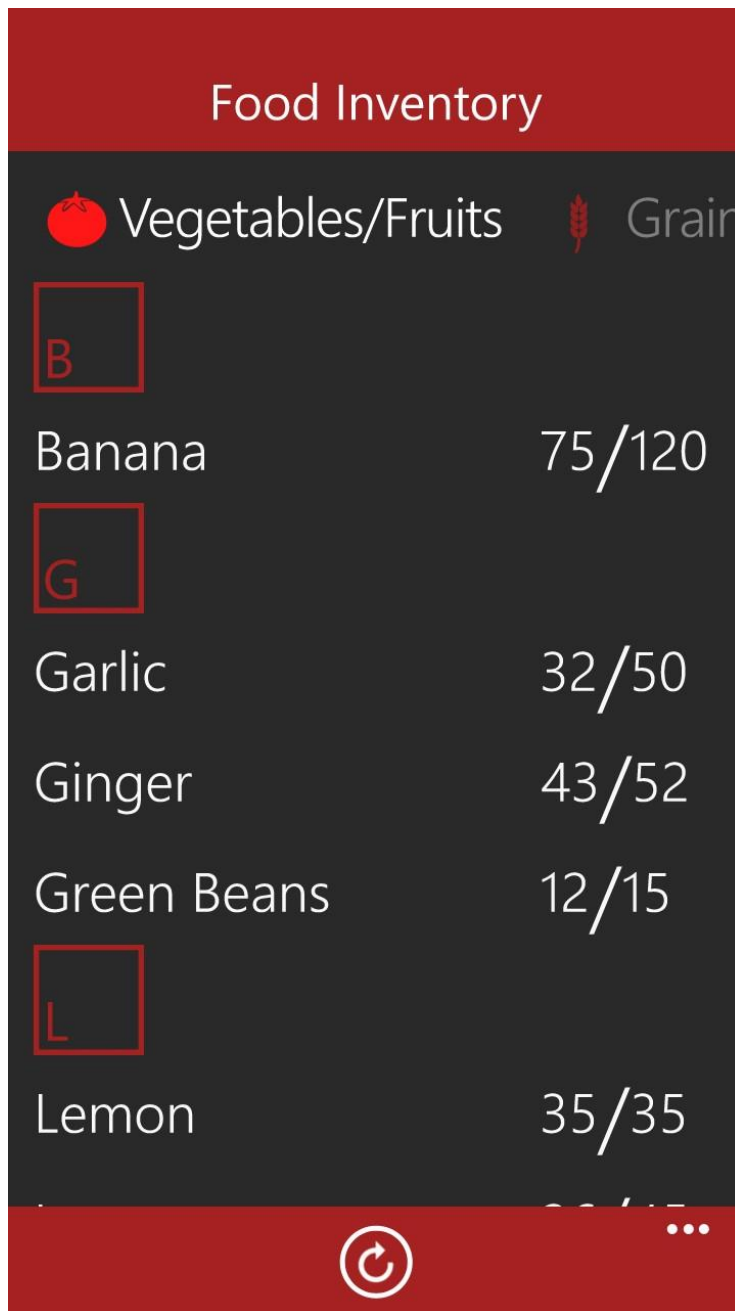
Date of Birth



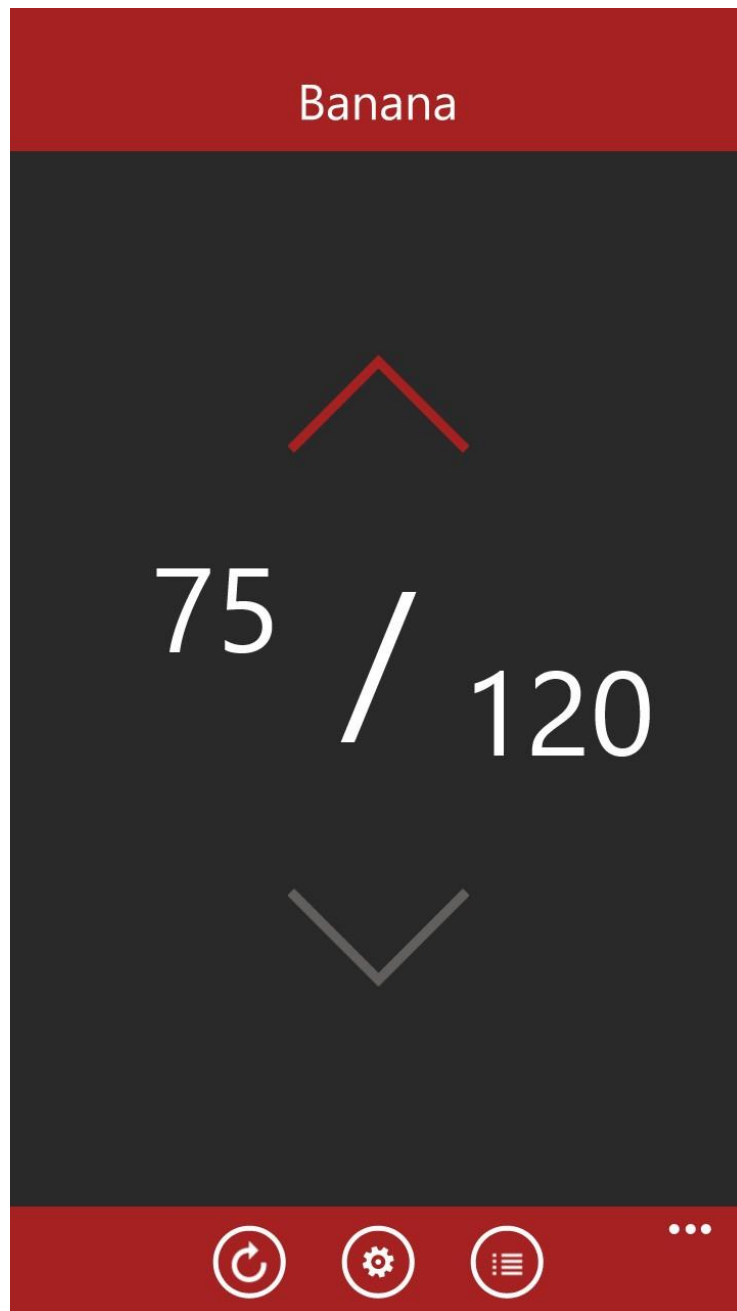
Editing of employee information



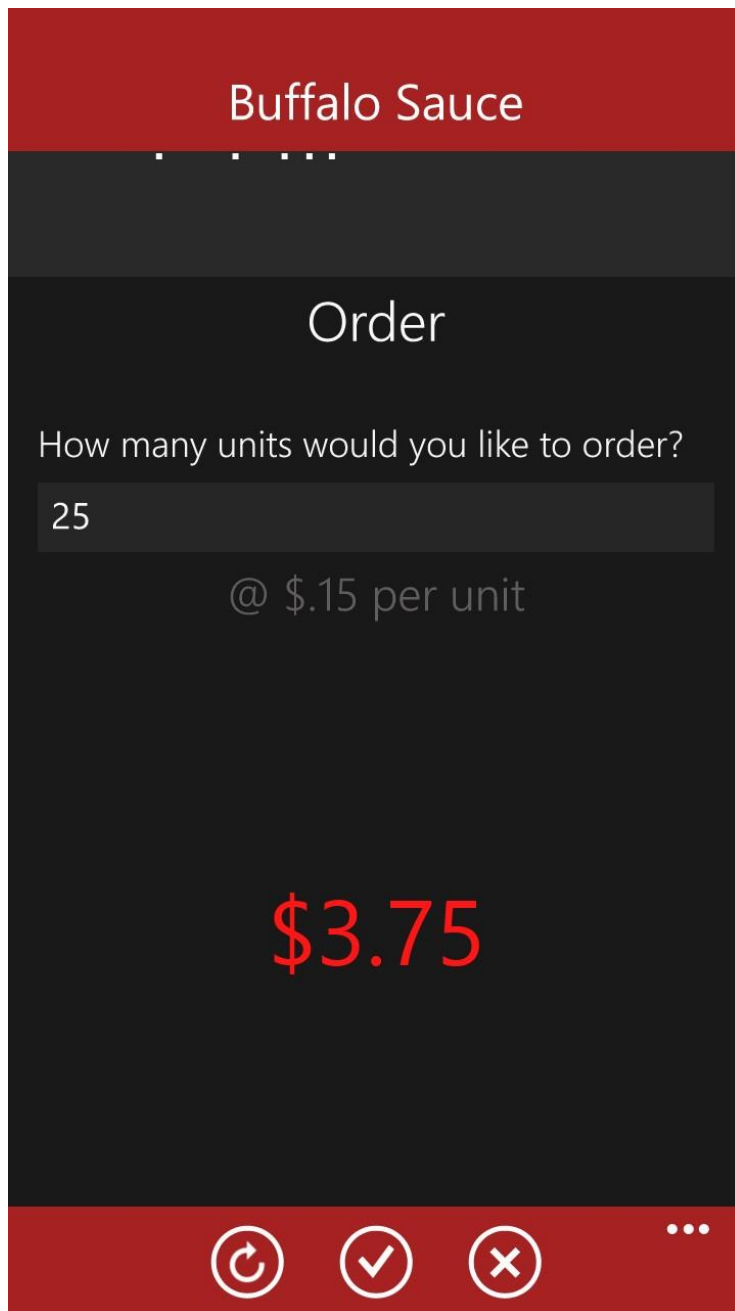
Tracking of business statistics



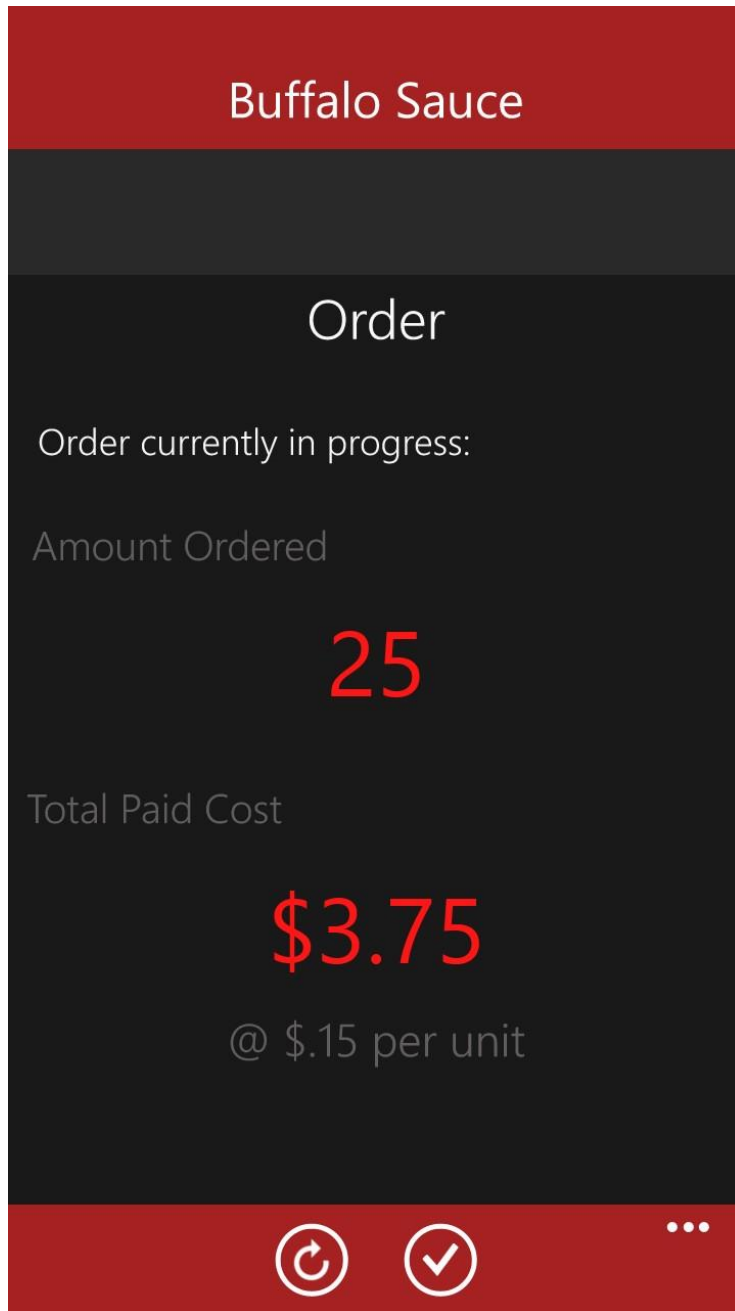
List of inventory amounts for each ingredient sorted by category



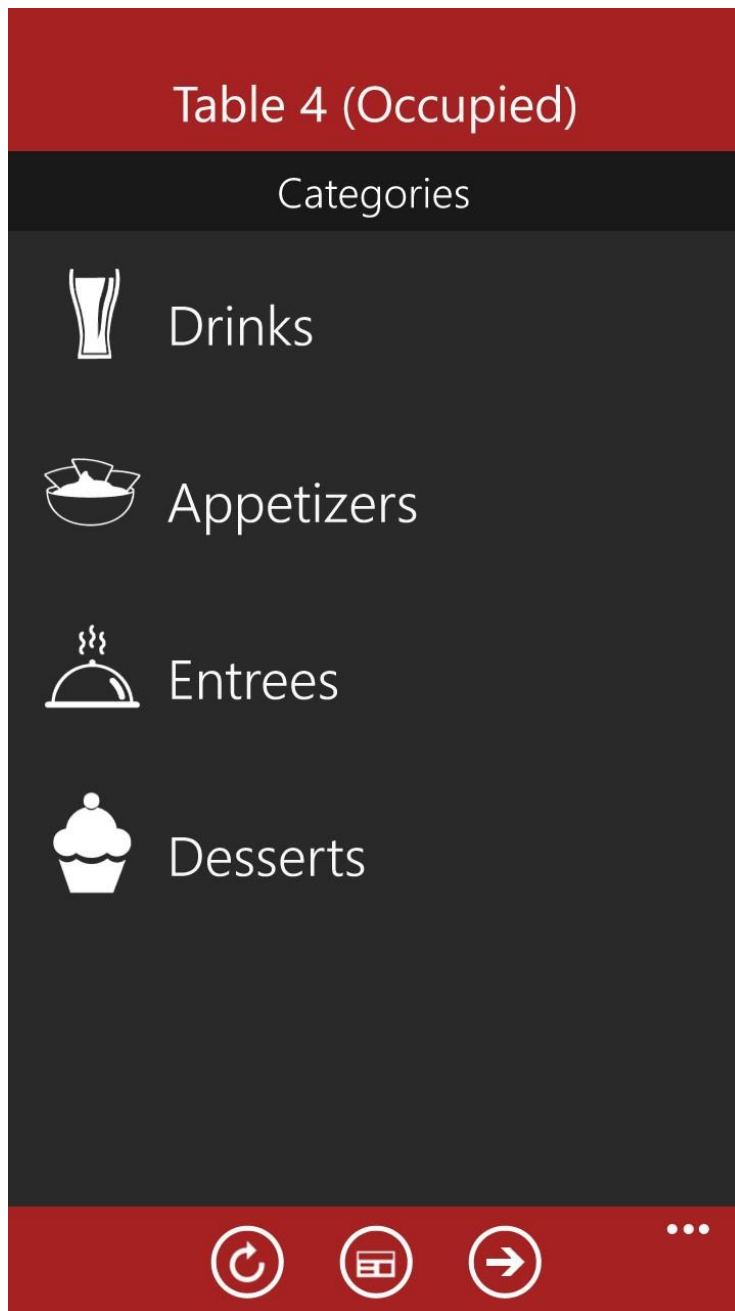
Manual changing of inventory amounts for bananas



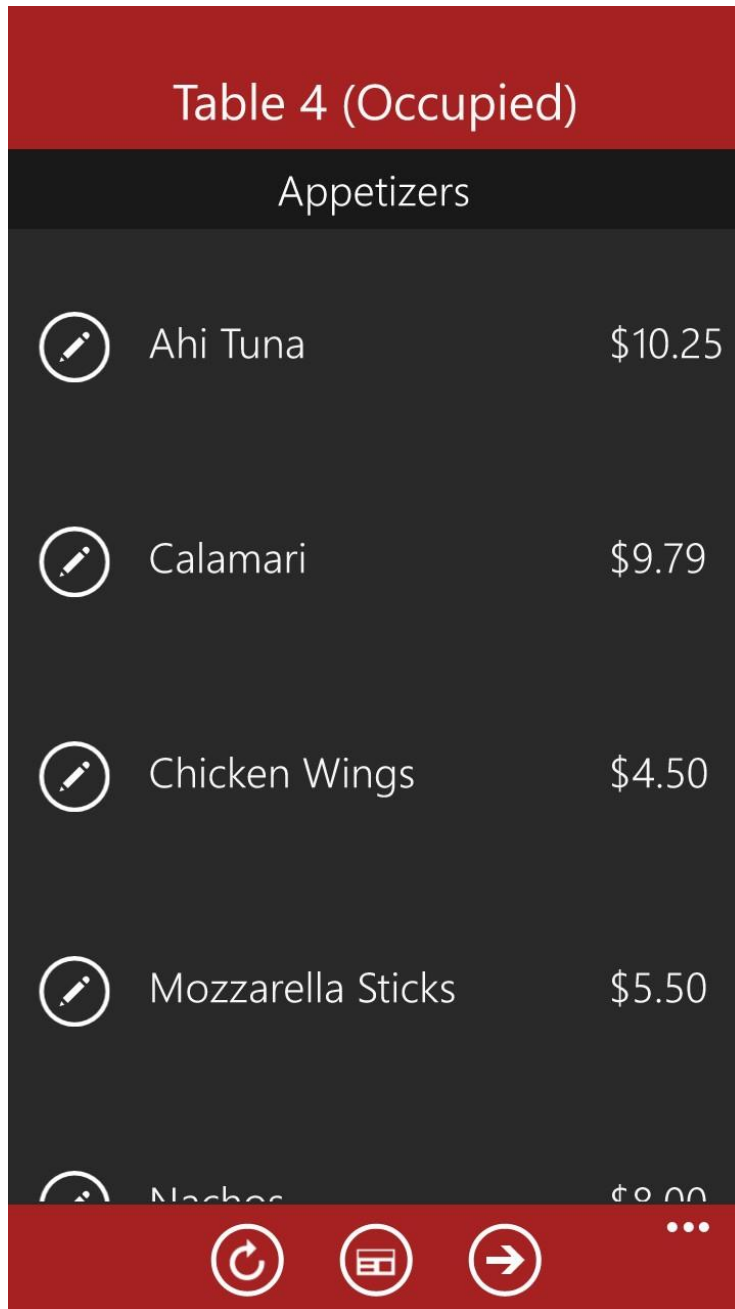
Manual ordering of inventory through the application



The order tab for buffalo sauce




The waiter's view of the menu category selection page



Displaying the appetizers

Calamari

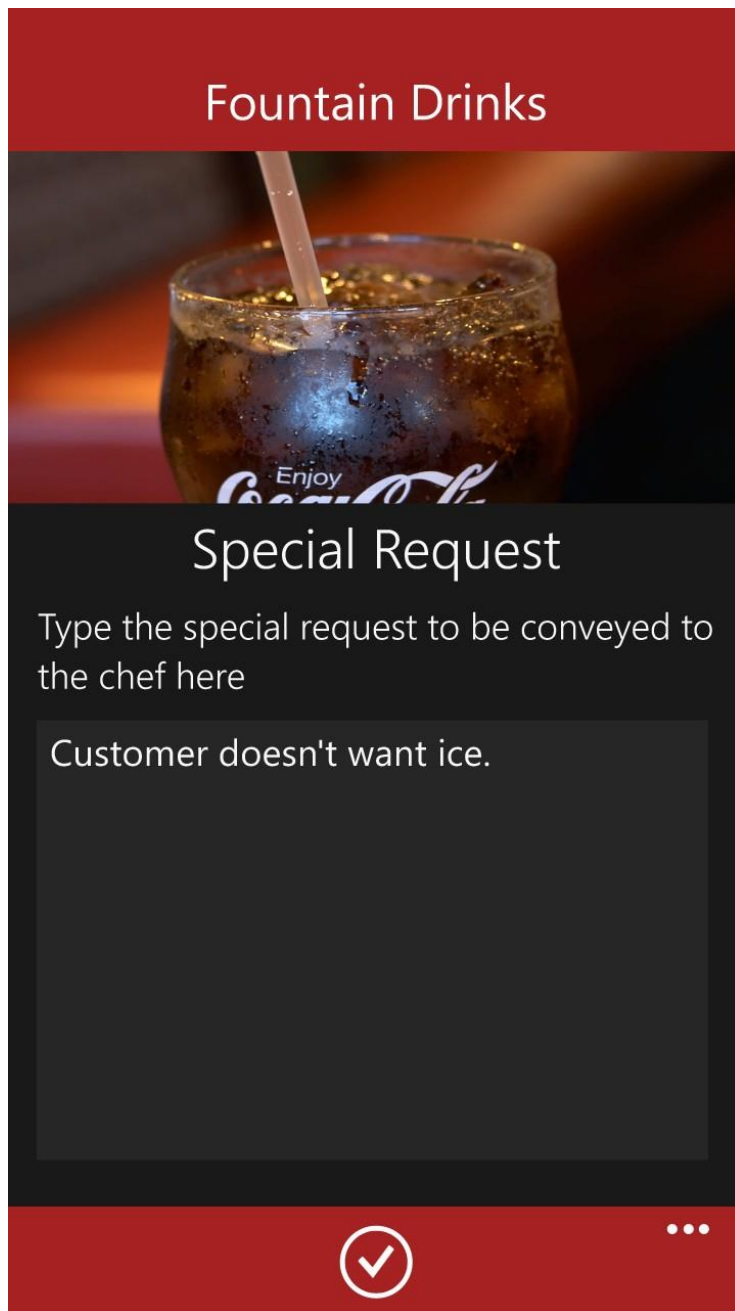


Check all the ingredients to be removed

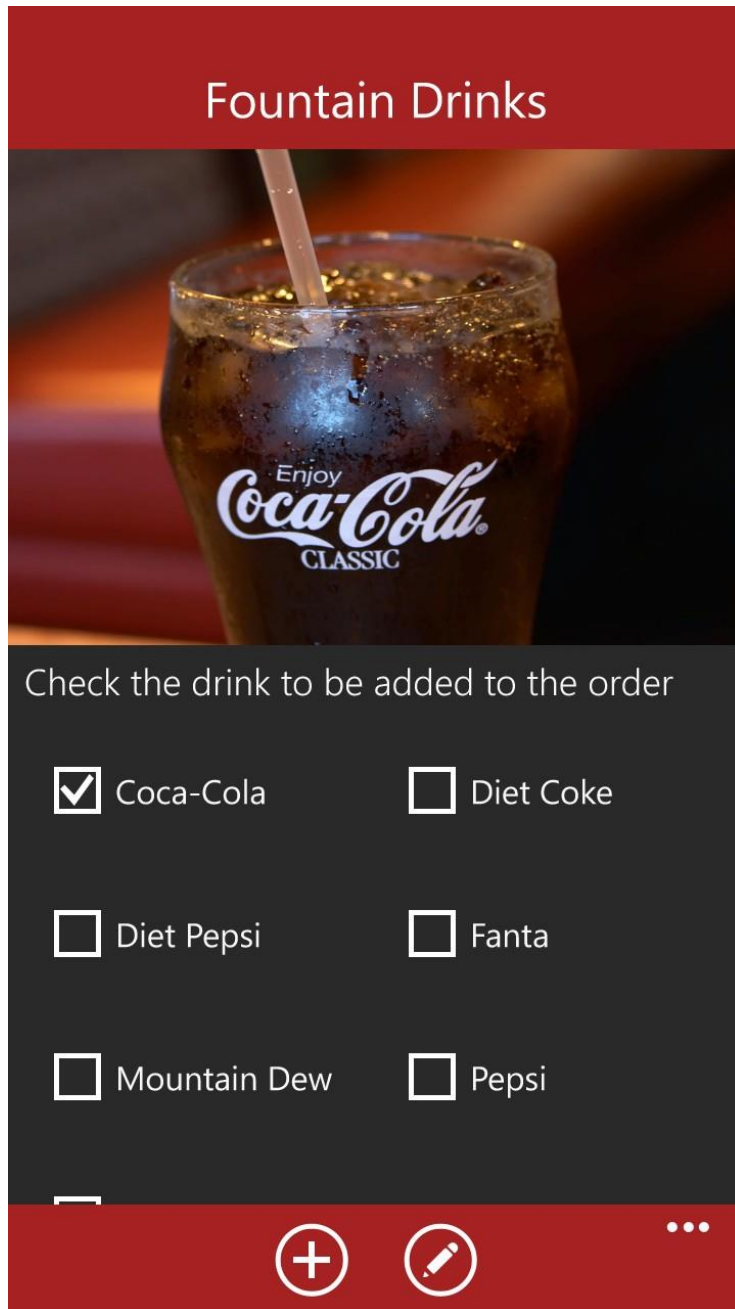
<input type="checkbox"/> Flour	<input type="checkbox"/> Lemon
<input type="checkbox"/> Marinara Sauce	<input type="checkbox"/> Pepper
<input type="checkbox"/> Salt	<input type="checkbox"/> Squid

+ ✎ ⋮

Removal of allergens/undesired ingredients from item in order



Setting a special request for an item in the order



Specifying the brand of a fountain drink in the order

Table 4: Fountain Drinks

Alteration Request

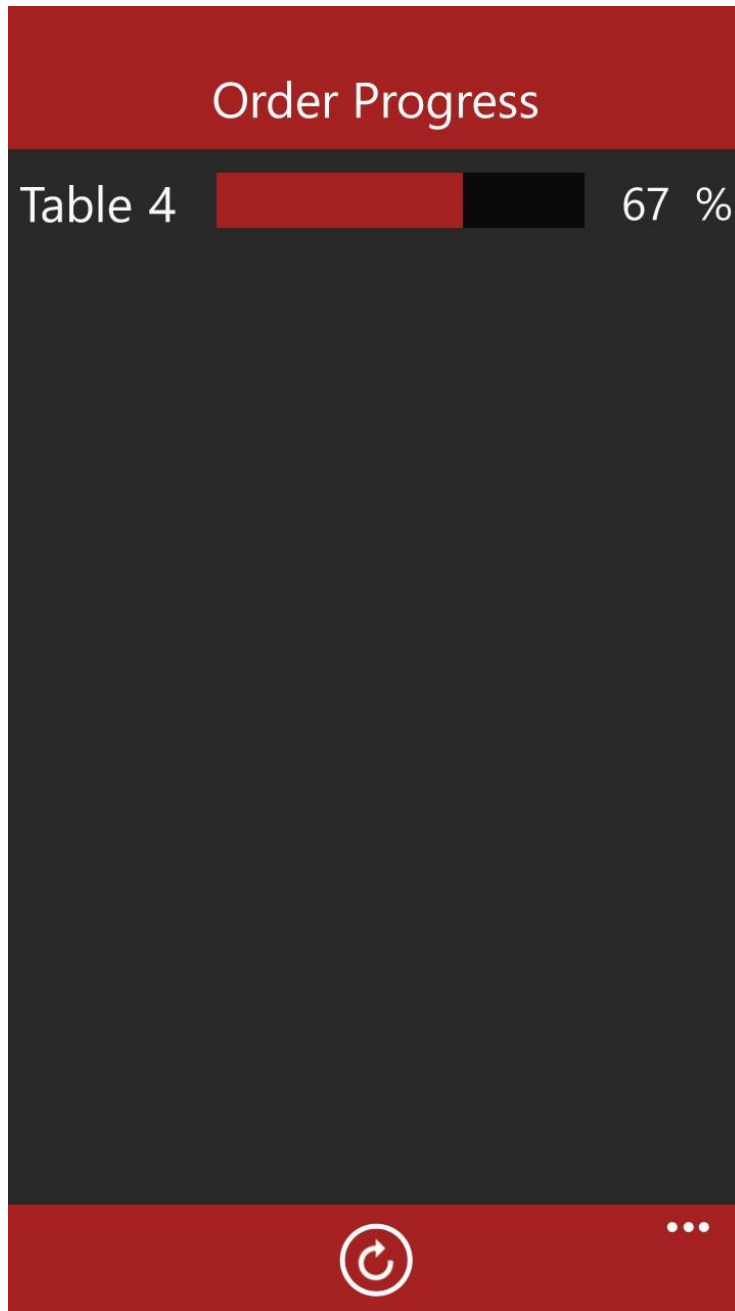
Coca-Cola

Special Request

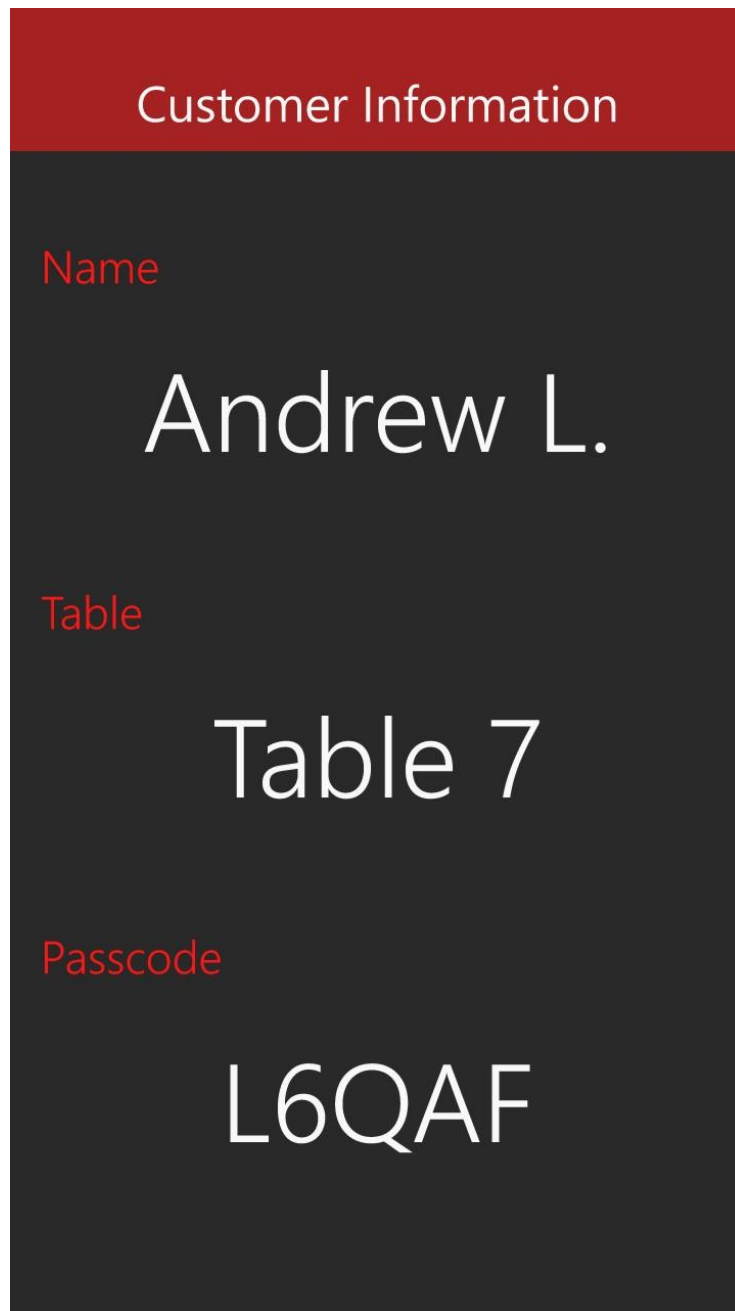
Customer doesn't want ice.



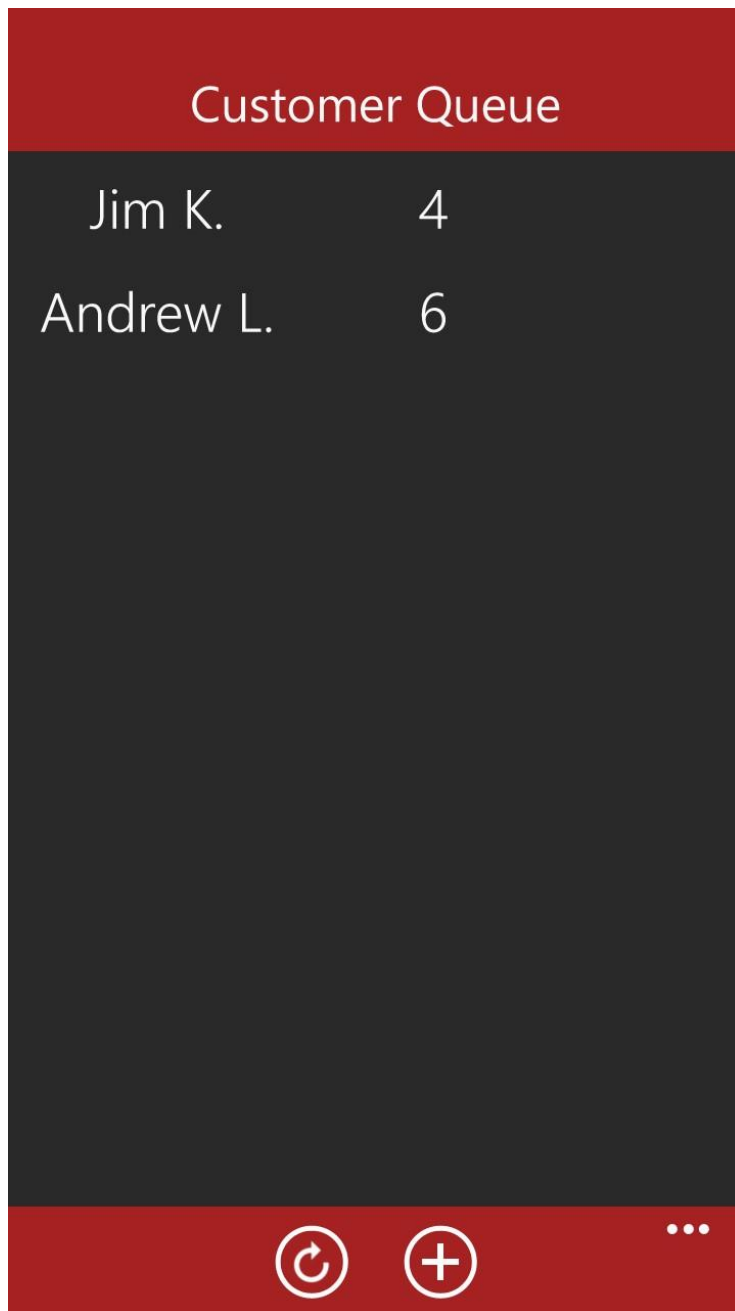
Bartender's view of the drink containing a special request



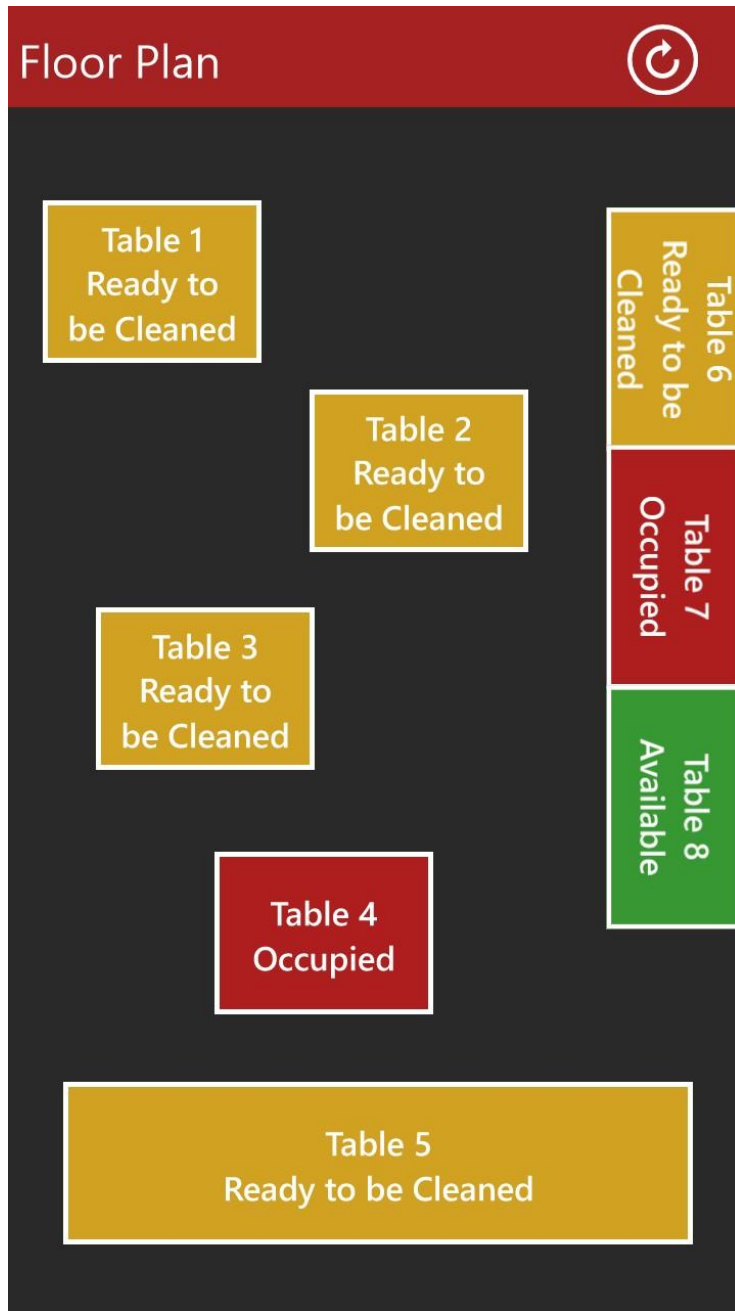
Waiter's view of the order progress percentage



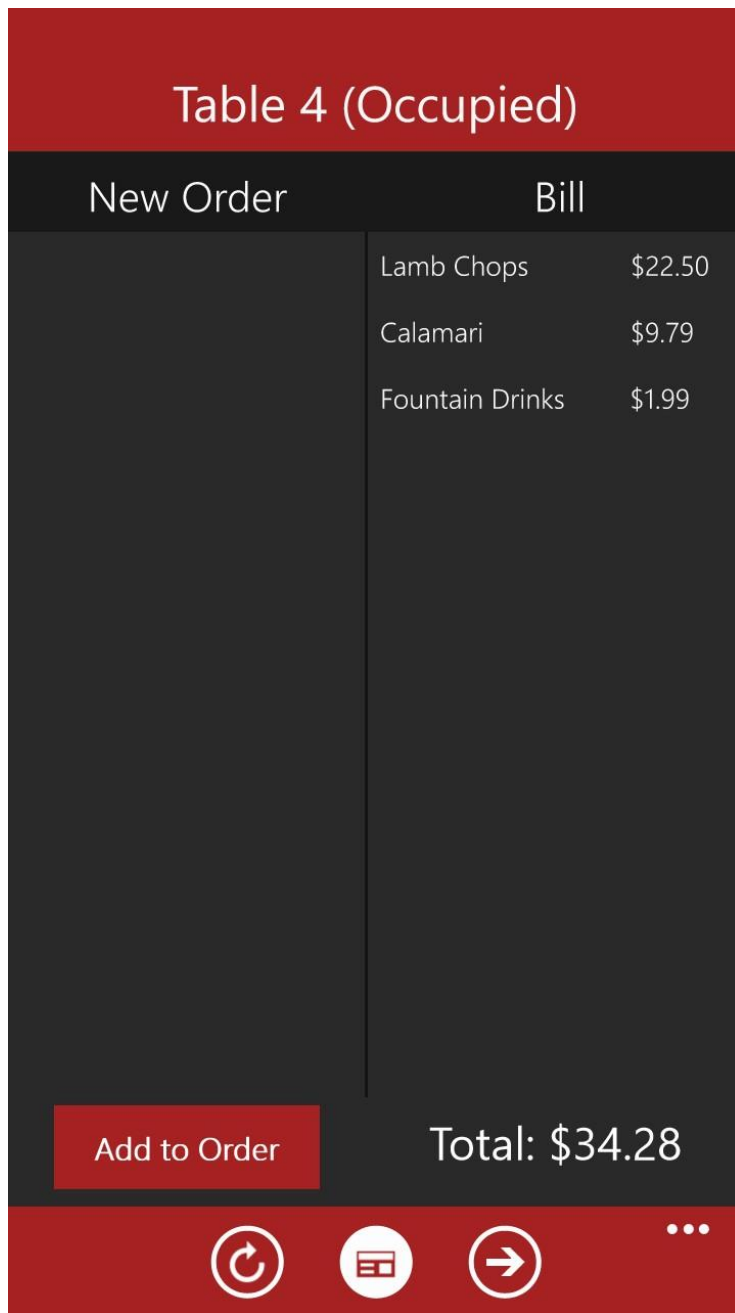
Host's view of customer table information, including customer passcode



Host's view of the customer queue for seated and waiting customers



Waiter and busboy view of the floor plan



Waiter's view of the bill page

Details of Employee Web Portal

The employee portal is a means for restaurant management to add an element of automation to the scheduling process by allowing workers to log in online and make use of features such as taking shifts, dropping shifts, and viewing upcoming shifts.

For stored data (i.e all shifts), MySQL was used as the database language of choice. Different databases represent the type of employee including waiters, chefs, etc. Each database has seven tables corresponding to each day of the week. Under each table, there exists a row for each date. For example, under the "sunday" table, three of the rows would coincide with 5/3/15, 5/10/15, and 5/17/15. Each row has ten different columns - an ID (auto incrementing for each entry), a date, and four pairs of a one character boolean matched with multi character "username" field.

The reason for these four pairs is that there are four shifts per day in the restaurant hours and scheduling: 12p-3p, 3p-6p, 6p-9p, and 9p-12a. The boolean for the corresponding shift serves to show whether the shift has been taken. If the boolean is false, the matching username is NULL, and if the boolean is true, the matching username field is populated with the username of the employee that picked up the shift.

These data fields are later used in the frontend to show those accessing the portal the names of the employees which have taken certain shifts, and it is especially helpful for the manager. One such case is when utilizing "View Upcoming Shifts" - PHP cycles through the database to appropriately print all shifts coming up in the week relevant to the employee which requests the information.

The front-end was primarily implemented using HTML, CSS, and JavaScript. One of the things required in the implementation was to bridge the gap between front-end and back-end due to the highly supported accessibility. As an example, if we consider "Take Shift" buttons that pair with shifts around the page, upon clicking the button, the database needs to be updated with relevant information and forcing a page reload. We solved this problem by making use of AJAX, or Asynchronous JavaScript and XML.

AJAX enabled interaction with back-end structures with nearly any kind of user action on the front-end. This meant that buttons, links, and the menu were all fair game in actively editing data and running PHP catalyzed by the user. It is important in this type of website that AJAX is asynchronous; by leaving the browser unlocked, user activity remains unmitigated while the database is receiving requests and being manipulated.

A large aspect of the automation involved in the website was that when a user logged in, their local time was stored as a persisting session variable in PHP. Independent of one moving around or flipping through weeks of shifts, the time remained as a global variable its main purpose being to create a

minimal maintenance model. As a result, an assortment of features blended in as convenience and ease of use.

The user is always presented with their current week of shifts when logging in by calculating the last Sunday (start of the week) from their date. Dropping shifts (also enabled by AJAX) is only possible forty-eight hours before the shift; the reasoning here was to allow time for management to find a replacement. It is inferred that anything sooner than forty-eight hours would rarely be enough time to properly populate all shifts that need to be taken. Shifts only from the current week and next week are open for interaction. Beyond that, shifts are not available so that users do not pick up shifts weeks in advance. For fairness, the next-next week opens up on Sundays at 12am.

On the topic of fairness, another feature which strongly supports this notion is that of implementing a hard cap for maximum hours allowed per employee in a week. It is set to twenty four, and employees are not able to take anymore shifts in a particular week once they have twenty four hours scheduled for that week; instead, they receive an informative popup to remind them of this cap.

In the scheduler, switching week functionality (i.e next week, previous week) is handled by keeping an ID number as a global, persisting variable. AJAX updates this when the user clicks the corresponding button to increment or decrement the ID so that the user is shown the appropriate week.

Employees other than the manager view the schedule for their type (i.e chefs see the chef schedule, waiters see the waiter schedule, etc.) while the manager can view the schedule for any type of employee. This is to enable proper management by giving them an interface to cleanly see any unclaimed shifts for the week.

Design of Tests

Manager:

Test-Case Identifier: TC-01

Function Tested: addToMenu(string itemName, double Price): bool

Pass/Fail Criteria: Test will pass if a new Item is added to Menu List.

Test Procedure	Expected Results
Call Function (Pass)	Item will be added to the menu.
Call Function (Fail)	If similar name and price is already there, it will return false. If only price is different it will update Price and return true.

Test-Case Identifier: TC-02

Function Tested: removeFromMenu(itemInfo* i): bool

Pass/Fail Criteria: Test will pass if an existing Item is removed from Menu List.

Test Procedure	Expected Results
Call Function (Pass)	Item will be removed from the menu.
Call Function (Fail)	If similar name is already there, it will return true and remove item. If this item is not located on menu it will return false and do nothing.

Test-Case Identifier: TC-03

Function Tested: hire(eInfo* e): bool

Pass/Fail Criteria: Test will pass if an employee is added to the database.

Test Procedure	Expected Results
Call Function (Pass)	Employee will be added to the database.
Call Function (Fail)	If employee info is correct or modified from a previous employee it will update database with employee and function returns true. (Employee is just on system still pending hiring process must be authorized later once more)

Test-Case Identifier: TC-04

Function Tested: fire(eInfo* e): bool

Pass/Fail Criteria: Test will pass if an employee is removed from the database.

Test Procedure	Expected Results
Call Function (Pass)	Employee will be added to the database.
Call Function (Fail)	If employee info is correct from a previous employee it will update database with employee and return true. If information does not match that on system it will return false and do nothing. (Employee is just on system still pending firing process must be authorized later once more after legal procedure and severance package)

Test-Case Identifier: TC-05

Function Tested: Business Stats in Database (UC-2)

Pass/Fail Criteria: Test will pass if change in expenses, salaries and viewing of profit/loss occurs.

Test Procedure	Expected Results
1. Enter a salary for a given employee	Salary will be changed.
2. Enter a new expense in database	Expense will be added.
3. View Profit/Loss	Profit/Loss Sheet will be viewed

Chef:**Test-Case Identifier:** TC-06**Function Tested:** addToMenu(string itemName, double Price): bool**Pass/Fail Criteria:** Test will pass if a new Item is added to Menu List.

Test Procedure	Expected Results
Call Function (Pass)	Item will be added to the menu.
Call Function (Fail)	If similar name and price is already there, it will return false. If only price is different it will update Price and return true.

Test-Case Identifier: TC-07**Function Tested:** removeFromMenu(itemInfo* i): bool**Pass/Fail Criteria:** Test will pass if an existing Item is removed from Menu List.

Test Procedure	Expected Results
Call Function (Pass)	Item will be removed from the menu.
Call Function (Fail)	If similar name is already there, it will return true and remove item. If this item is not located on menu it will return false and do nothing.

Test-Case Identifier: TC-08

Function Tested: viewQueue(): void

Pass/Fail Criteria: Test will pass if the queue is successfully displayed

Test Procedure	Expected Results
Call Function (Pass)	Queue is shown when the chef selects the view queue option in their interface
Call Function (Fail)	Queue is not displayed when the chef selects the view queue option in their interface.

Test-Case Identifier: TC-09

Function Tested: updateQueue(): void

Pass/Fail Criteria: Test will pass if the progress of an order is successfully updated.

Test Procedure	Expected Results
Call Function (Pass)	The progress indicator percentage of an order is updated when the function is called, a value of true indicating success is returned.
Call Function (Fail)	The progress indicator percentage of an order is not updated when the function is called, a value of false indicating failure is returned.

Waiter:**Test-Case Identifier:** TC-10**Function Tested:** placeOrder(): bool**Pass/Fail Criteria:** The test will pass if an order is successfully placed to the chef's queue.

Test Procedure	Expected Results
Call Function (Pass)	Information about the order is processed and sent to the chef's queue. The function placeOrder() will return true.
Call Function (Fail)	The menu item cannot be added, i.e. out of stock. The function placeOrder() will return false.

Test-Case Identifier: TC-11**Function Tested:** cancelOrder(orderInfo* o): bool**Pass/Fail Criteria:** The test will pass if the return value is true and the placed order is canceled.

Test Procedure	Expected Results
Call Function (Pass)	The placed order is canceled and removed from the chef's queue. The function returns true.
Call Function (Fail)	The chef has set the placed order from "In Progress" to "Completed" and is awaiting pick-up from the waiter. The order cannot be undone, and the function returns false.

Test-Case Identifier: TC-12**Function Tested:** viewMenu(): void**Pass/Fail Criteria:** The test will request the system to display the menu on the waiter's interface.

Test Procedure	Expected Results
Call Function (Pass)	The menu items are displayed on the waiter's interface.
Call Function (Fail)	No menu items are shown.

Test-Case Identifier: TC-13

Function Tested: viewTables(): void

Pass/Fail Criteria: The test will display all tables in the restaurant and their occupancy statuses on the waiter's interface.

Test Procedure	Expected Results
Call Function (Pass)	On the waiter's interface, a diagram of all the tables in the restaurant are displayed with corresponding colors to distinguish if a table is occupied or available: red signifies that a table is occupied, whereas green signifies that a table is available.
Call Function (Fail)	The system fails to retrieve data about the tables and their statuses.

Busboy:

Test-Case Identifier: TC-14

Function Tested: cleanTable(): bool

Pass/Fail Criteria: The test will pass if a dirty table's status changes to clean and the function returns true.

Test Procedure	Expected Results
Call Function (Pass)	The busboy taps the button to change the status of the table after cleaning it. The status of the table is changed to "Clean".
Call Function (Fail)	The busboy tries to change the status of the table to "Clean" but the table is already clean, so the function returns false.

Test-Case Identifier: TC-15

Function Tested: viewTables(): void

Pass/Fail Criteria: The test will pass if the tables and their statuses are displayed on the busboy's interface.

Test Procedure	Expected Results
Call Function (Pass)	The tables and their statuses are displayed on the busboy's interface.
Call Function (Fail)	The system cannot retrieve data about tables and their statuses.

Employee:**Test-Case Identifier:** TC-16**Function Tested:** verify(eInfo* e): bool**Pass/Fail Criteria:** The test passes if the employee's information is successfully verified.

Test Procedure	Expected Results
Call Function (Pass) with employee login information that is correct, meaning the password is the one that is associated with the login identification number that was entered.	The system recognizes the combination of password and login identification number, the system moves on to display the corresponding interface that is associated with the position of the employee, true is returned by the function since login was successful.
Call Function (Fail) with an incorrect combination of login information and password.	The system fails to recognize the information, and returns false as the login is unsuccessful.

Test-Case Identifier: TC-17**Function Tested:** takeShift(sInfo* s): bool**Pass/Fail Criteria:** The test will display all tables in the restaurant and their occupancy statuses on the waiter's interface.

Test Procedure	Expected Results
Call Function (Pass) on an sInfo object that is present in the list of shifts that are up for coverage	The sInfo* s object containing the shift information should be added to the list of shifts for the employee that called the function, true should be returned since it was successful.
Call Function (Fail) on an sInfo object that is not present in the list of shifts that are up for coverage	The list of sInfo* shift information for the specific employee should remain the same as it was prior to the function call, since this sInfo object is not present in the list of shifts for coverage. The function should return false since unsuccessful.

Test-Case Identifier: TC-18

Function Tested: placeShift(sInfo* s): bool

Pass/Fail Criteria: The test will display all tables in the restaurant and their occupancy statuses on the waiter's interface.

Test Procedure	Expected Results
Call Function (Pass) with a sInfo object that is in the employee's list of shifts.	The sInfo object that is passed as argument now appears in the list of shifts that are available for coverage, and is no longer present in the employee's list of shifts. The function returns true since the shift was successfully placed up for coverage.
Call Function (Fail) with an sInfo object that is not in the employee's list of shift	The system returns false, since the sInfo object is not present in the employee's list of shifts, meaning that they cannot put it up for coverage because it does not exist or it does not belong to them.

Test-Case Identifier: TC-19

Function Tested: viewInfo(): void

Pass/Fail Criteria: The information for a given

Test Procedure	Expected Results
Call Function (Pass) from the employee interface of an employee who has information to display.	The system displays the information associated with an employee's identification tag.
Call Function (Fail) from the employee interface of an employee who does not have information to display, such as a new hire.	The system does not display the information associated with an employee's identification tag.

Controller:**Test-Case Identifier:** TC-20**Function Tested:** DBConnection(): void throws exception**Pass/Fail Criteria:** The test passes if a the controller is able to connect to and access the database.

Test Procedure	Expected Results
Call function (Pass) as the system initializes while system is connected to network.	Controller is able to connect to the database
Call function (Fail) as the system initializes while system is not connected to network.	If connection cannot be established, function throws exception.

Test-Case Identifier: TC-21**Function Tested:** DataRequest(): void throws exception**Pass/Fail Criteria:** The test passes if a the controller receives valid data.

Test Procedure	Expected Results
Call function (Pass) when the controller is connected to the database.	Controller is able to receive data from the database
Call function (Fail) when the controller is not to the database	If request function fails, function throws an exception.

Test-Case Identifier: TC-22**Function Tested:** DataUpdate(): void throws exception**Pass/Fail Criteria:** The test passes if a the controller is able to update data in interface and database

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	Controller is able to update information in the database and interfaces.
Call function (Fail) when the controller has not established the needed connections.	If update fails, function throws exception.

Test-Case Identifier: TC-23

Function Tested: DataAnalyze(): void throws exception

Pass/Fail Criteria: The test passes if a the controller is able to run algorithms on data and store results in database.

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	Controller is able to run needed algorithms and updates information in database.
Call function (Fail) when the controller has not established the needed connections.	If analysis fails, function throws exception.

Test-Case Identifier: TC-24

Function Tested: InterfaceConnection(): void throws exception

Pass/Fail Criteria: The test passes if the controller is able to connect to and access interfaces(through the request handler).

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	Controller is able to connect to the interfaces.
Call function (Fail) when the controller is not connected to the network.	If connection cannot be established, function throws exception.

Request Handler:

Test-Case Identifier: TC-25

Function Tested: processRequest(): void

Pass/Fail Criteria: The test passes if the request handler successfully processes the request to the controller from an interface or vice-versa.

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	Request is processed as needed.
Call function (Fail) when the request handler is not connected to the network.	If request cannot be processed, function notifies calling interface/controller and request handler waits for/processes next process in queue.

Test-Case Identifier: TC-26

Function Tested: getRequest(): void

Pass/Fail Criteria: The test passes if the request handler successfully gets a request being sent from one of the interfaces or the controller

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	Next request is received by the request handler.
Call function (Fail) when the request handler is not connected to the network.	If request cannot be processed, function notifies calling object and request handler waits for/gets next request.

Test-Case Identifier: TC-27

Function Tested: thread(): void throws exception

Pass/Fail Criteria: The test passes if the request handler successfully creates a new thread to process a request.

Test Procedure	Expected Results
Call function (Pass) when the system is fully initialized.	New thread is created and able to process a given request.
Call function (Fail) when the system reaches max number of threads (out of memory).	If thread creation fails, function will throw exception and close system (can indicate memory leak).

Test Coverage:

The tests cover most essential classes implemented and some useless as well as synchronization of the database to the application. More tests will be designed when we see it needed in the future as we develop more into the app.

Integration Testing Strategy

We deemed our system to be best suited for bottom up testing. This approach is more practical due to the structure of the modules in the application. For instance, if we consider the possibility that there could be a problem with the interaction between the chef and waiter, i.e hiccups in the waiter sending the chef orders. If we take a different approach to testing, it would be much more difficult to understand whether that issue comes down to the actual interaction or integration of the modules or if it is a problem with how the individual classes were originally designed.

Bottom up testing alleviates this quite a bit by first solidifying the foundation and moving onto the top of the system. This comes down to hierarchy of the classes and more of how they relate to each other attribute wise rather than the interactivity. By understanding the relationships between the objects in the system, a bottom up approach makes testing more efficient and straightforward in that you quickly narrow down where the problem lies prepping it for remedy.

A concrete way of representing the components of our system and how they would relate in this context would be that we test each employee's independent, personalized job tasks in the application first – example being, for the chef, we would test the implementation of the queuing system and add make sure that it updates the server when the chef proclaims that a dish is cooked. After going through individual functionality, we test the features which call for interactions between more than one object or class, i.e the busboy being notified to clean a table after a party has paid their check and made their leave.

Project Management and History of Work

Merging and Collecting Contributions:

All information is first gathered on a google doc and each member contributes his part of the project onto the google doc. A team member then takes all information and forms a PDF separately to keep consistent with the rest.

Problems:

For computer engineering students, our workload this semester required excellent management. Another problem was that we had to learn the necessary skills in order to program the application, since we were not taught exceedingly relevant skills in the pre-requisites for the course. Coordinating times to meet and discuss our progress was difficult since some of us have very different schedules.

History of Work:

January 25 - January 29:

This team was made during the break before the semester started. We began to look through which projects seemed interesting and doable with the school schedule we had this semester. Restaurant Automation seemed to be the best decision due to the fact that one of our fathers' used to own a franchise and had an adequate amount of information of what is needed in a software like this. It also seemed to be the most interesting out of all the projects and doable in a sense where we can learn to program and make it functional. The proposal was finally written after several meetings and additions to google docs on what to add to this previously made idea.

January 30- February 22:

Feedback was given after submitting the proposal allowing us to continue in our report. We began to structure the report into parts and split features between the members. Each member was responsible of his own feature unless his feature was not included in the part that was being worked on thus he would help with the other teams. We finally were able to gather a report and complete Report 1.

February 23 - March 14:

We began analyzing lectures and previous reports to understand the Requirements for this report as it all seemed to be new information that we haven't encountered before. As usual our team would gather info on a doc to discuss easily and communicate using GroupMe to finalize our plans. The report ended and finally submitted. We began implementation during this period.

March 15 – April 29:

During this period we continued to work diligently on the documentation and implementation for our application in preparation demo 1. Then we continued to work on the documentation for the submission two days after the demo.

April 30 – May 2:

After a successful demo 1, we continued to improve upon our program and analyze our problem frame for key features that would highlight our solutions to the restaurant automation problem. We continued to work on the documentation as well, and completed the documentation for report 3, after demo 2.

Current Status:

We are currently finishing up report 3, and gathering the required contents for the Electronic Project Archive.

Future Status:

Though the second demo is over, we look forward to continue improving upon our product in the hopes of having a real marketable product in the near future and improving our skillset as developers and software engineers.

End of Project Responsibility Breakdown:

Sujay Bandarpalle and Julian Esteban:

- Microsoft Azure database setup and interaction
- Waiter floor plan and table status settings
- Waiter order placement, allergens or undesired ingredients removal from selected menu item, menu item note for special requests
- Food queue interaction by chef and order progress percentage tracking for waiter
- Automated inventory updating and reordering, manual inventory updating and reordering
- Tracking of business statistics (revenue, expenses, and profits)
- Host interface, waiting party list, and automated seating algorithm
- Log in and user authentication of employees and customers
- User interface and user experience throughout the application (UI/UX)

Jonathan Du and Paolo Umali:

- Customer order progress portal
- Busboy interface
- Busboy floor plan
- Manager addition and removal of employees
- Research to obtain ingredients for menu items
- Bill total calculation
- Waiter interface
- Menu pages and item selection

Kanav Tahilramani and Omar El Warraky:

- Employee portal
 - Employee portal database
 - Employee automatic scheduling algorithm and manual override for schedule changes
 - Bartender Interface
 - Drink Preparation Queue
 - Removal of items from bill
 - Removal of items from new order
 - Item addition confirmation
-

The Coordination of the integration will be a team effort where everyone helps each other while focusing on their own functions.

References:

1) Group 4's Report 2013 on format of Document:

<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2013-g4-report3.pdf>

2) The Software Engineering textbook by Ivan Marsic. Link at:

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

3) Wikipedia Definition of User Stories:

http://en.wikipedia.org/wiki/User_story

4) Group 4's Report 2014 on format of Document:

<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2014-g4-report3.pdf>

5) Group 1's Report 2013 on format of Document

<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2013-g1-report3.pdf>

6) UML Design: gliffy.com

7) Report1 Appendix A by Professor Marsic:

<http://www.ece.rutgers.edu/~marsic/Teaching/SE1/report1-appA.html>
