Why W8

Restaurant Automation

https://github.com/SE-Group-4/Why-W8

Group 4

Report 1 Full Report

Stephan Dimitrovski
Michael Haas
Jimmy Jorge
Nikhil Jiju
Kyungsuk Lee
Yi Xie

Individual Contributions Breakdown

	Project Category		Team Member Name					
			Stephan	Michael	Jimmy	Nikhil	Kyungsuk	Yi
Responsibility Levels	Project Management *10 Points*		40%		40%		20%	
	Sec 1. Customer Statement of Requirements	Problem Statement *5 Points*			100%			
		Glossary *4 Points*		50%			20%	30%
	Sec 2. System Requirements	Functional Requirements *2 Points*	100%					
		Nonfunctional Requirements *2 Points*					100%	
		UI Requirements *2 Points*				100%		
	Sec 3. Functional Requirements Specification	Stakeholders, Actors, Goals *2 Points*	30%		50%		20%	
		Use Case Casual Descriptions *8 Points*	100%					
		Use Case Diagram *5 Points*	100%					
		Use Case Full Descriptions *10 Points*					100%	
		System Sequence Diagrams *5 Points*		15%		15%		70%
	Sec 4. User Interface Specification	Preliminary Design *11 Points*		33%		34%		33%

		Effort Estimation ***4 Points***		15%	70%	15%		
	Sec 5. Domain Analysis	**Concepts** ***10 Points***	20%	15%		15%	50%	
		Associations ***5 Points***		25%				75%
		Attributes ***5 Points***		25%	75%			
		Contracts ***5 Points***				50%		50%
	Sec 6. Plan of Work ***5 Points***				100%			

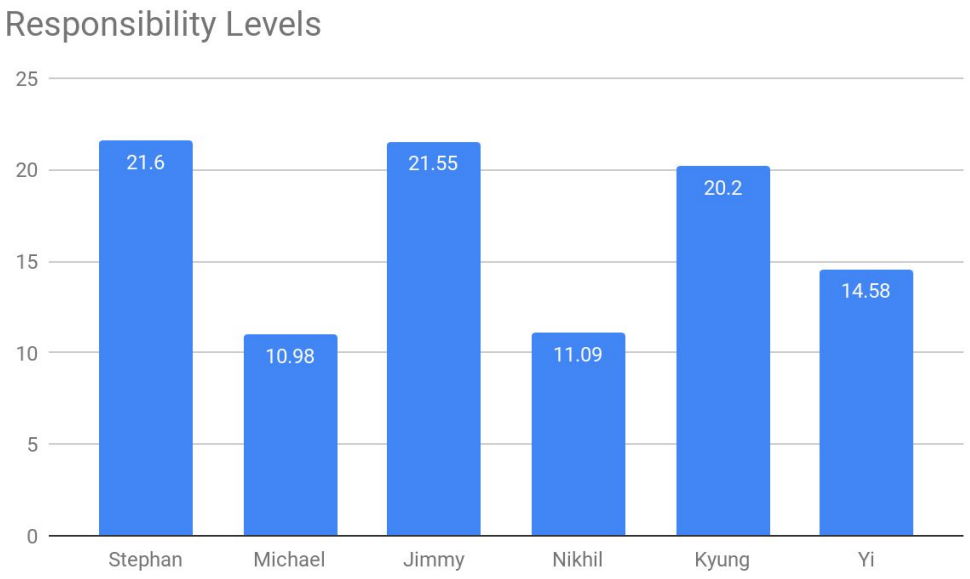


Table of Contents

1. Customer Statement of Requirements

1.1 Problem Statement

Actor - Host/Hostess:

By being the first person the customer sees, after coming through the front door, there is a paramount of importance placed on hospitality that is served by the host. As a host, I have to try to make the customer feel welcome, and have them seated in a calm and collected manner. However, this feat can be more difficult than it sounds when factoring a large influx of customers and limited seating, along with the confusion of people moving about. If there was a program that could keep track of the open seats, and where they were located, I could serve newly entering customers much faster. What would be even better is if the customer had reserved a seat for themselves prior to arrival. That way I wouldn't even have to search for an open seat, and all I would have to do is confirm that they did indeed reserve a seat and guide them to it.

The Why W8 solution: Why W8 allows the customer to select their own table before even stepping foot inside the restaurant. The process is simple, the customer will select their desired time and table. If available, the customer will reserve said table at said time. Upon arrival the customer will simply give their name and the hostess will see when and where to seat them.

Actor - Manager:

Running a business is not easy by any means. If there is anything that could help minimize the amount I have to spread myself across tasks, and help me keep the business profitable, then I'm all ears. I was personally thinking about something that can help the restaurant keep track of the flow of customers, and perhaps what they ordered. That way I know when to keep extra staff on hand, and what to order more of in order to keep a stocked inventory.

That being said, there is also the issue of my having to keep tabs on my employee, to make sure that they are working in a timely manner. I hear that you have some kind of table tracker that you might be able to implement. Well, if you could add in something that keeps track of how long it takes the busboy to clean off the tables and how long it takes for the guests to be seated along with being served, that would make my day. Saves me some time from having to look over the camera records to make sure there isn't any major slacking.

However, while all of the aforementioned is nice and dandy, I really don't want to have to juggle around another program. Sometimes I get so lost when I have to sift through so many different programs in order to manage payrolls, inventory, shift management, expenditures, et cetera. If there is any way you lot can combine all of these tools for me in one compact program, I would be eternally grateful.

The Why W8 solution: The overall design of Why W8 keeps track of practically every action made in the restaurant by both the employees and the customers. This makes relaying data back to the management much easier. For example, Why W8 will be able to recognize how many orders were placed, how many dishes are ordered per order, how much of the inventory was used per order, and how many employees are on staff between a given interval of time. With knowledge of this information, Why W8 can easily create charts for management to keep track of the day to day running of the restaurant.

Actor - Busboy:

Most people think living the life of a busboy is easy - and it is, for the most part. But, there are some days that just have me turned about, with having so many customers and all. With the hostess asking you for an informative layout of the tables in the restaurant, I was wondering if you could put in some information that shows which tables have to be cleared off and cleaned. You know, that way I could quickly come over and do whatever has to be done to make the place spiffy for the next customer.

The Why W8 solution: Why W8 offers an interactive display which will keep track of every step along the way of a successful dining experience. With that being said, the Busboy(s) will know when a customer has left the restaurant, which will trigger the Busboy(s) to clean the now unoccupied table, once the table is clean the Busboy(s) will be able to set the table as a clean “Available” table.

Actor - Customer:

I’m really picky about my food - I prefer knowing that the food I’m about to get is near perfection, at least to most people. Some kind of rating system might help me with decisions regarding just that. In consideration of the foregoing, I also want to know what exactly is in the food, in the case of allergies and for calorie counting purposes. All this talk about food has made me hungry, so I’ll probably call somewhere for food; actually, that gives me another idea! If there was a digital menu, I wouldn’t have to go through the hassle of having to speak with someone and waste time trying to understand them as they try to understand me - my phone isn’t especially good, so this is more of a problem than you might think. And perhaps if I could order ahead of time, that would cut down my waiting to get the food - fantastic! Speaking of waiting, if the menu also displayed how long an item would take to make, then I would know when to arrive at the restaurant to pick up the food when it’s nice and hot. Although, there are also occasions when I like to just eat at the restaurant, but the line goes through the door sometimes! If only there was a way everyone already knew where to sit and could just get to it. What if there was a way to combine some of those ideas! Hey, let’s say I order off of this menu, then maybe I can also reserve a seat for myself, and pay the bill while I’m eating. That way, I already have a seat and food ready to go - presto, time saved.

The Why W8 solution: Why W8 offers a digital touch menu with numerous features to ensure customer satisfaction. An integrated rating system will show the customer what previous customers thought about a specific dish, this will guide the customer into selecting a dish that will help them leaving satisfied. The menu will also display what ingredients are in each dish which will avoid any allergy issues. To remove the uncertainty of a wait time, Why W8 has the time it will take for each dish to be made displayed on the menu. Another great feature Why W8 provides is a table selection option. This option allows the customer to reserve a specific table at a specific time.

Actor - Chef:

Day in and day out, I work in the kitchen - being a chef isn't terrible, but boy is it stressful. Having to make sure orders are filled out in a timely fashion and keeping track of dishes that have a few modifications requested to them by the customer really does a number on me sometimes. Not to mention, sometimes what the waiter writes down looks like downright chicken scratch, it all just makes the job harder than it has to be. So, what I would like to get from you guys is something that will help me read incoming orders clearly, with any possible modifications done to them in a different color. That way, I don't have to rely on the waiter's chicken scratch, and I can clearly see what is different about this meal from the standard. Next on my list would be something to help tell the waiter that the food is ready to pick up - that's another one of my pet peeves; I have a fresh meal ready for a customer and it just sits around for 5 minutes, taking up space on the counter for more dishes and getting cold all the while.

Now since that's all settled, there is one more idea that I have, but I'm not sure if it's worth the effort. I was thinking about having the incoming orders laid out in such a way so that I could tell what ingredients are common among them. That way, I could prepare larger batches of whatever it is that the meals require, say sautéed mushrooms for instance. If an order for a burger with mushrooms came in, and then an order of creamy mushroom on fettuccine, I would like to be able to see that they share sautéed mushrooms in common, so I could make more in one go rather than having two sautéing sessions. But, that's just my own little idea - I'm not even 100% sure it would work the way I think it could.

The Why W8 solution: Because Why W8 is a connected application, when the customer places an order it immediately goes into a queue until it is ready to be made. Once the order is ready to be made the Chef will be able to click “Start Order.” This will bring up the recipe for each dish. Once the Chef completes the order they will simply click “Order Complete” which will signal the server to pick up the food.

Actor - Waiter/Waitress:

Man do I hate when the chef yells at me - he gets so agitated when I don't pick up stuff from him in the time he wants me to. It's not like I diddle-dally my way around the restaurant. I have customers to attend to as well, taking their orders and, well, waiting on them. I hear you guys are already implementing some sort of order ahead system - that is absolutely fantastic, since it will take a bit of the load off of me. I was just wondering if there was some way you can have me fill out orders digitally, and I would just send them off to the kitchen without even my having to go there. Also, if you guys could have a little list for me indicating which meal is done, and to which table it should go to, that would be fabulous. In fact, if you could have some way of indicating which table has yet to be served, or is in the middle of eating, or is wanting to pay their bill, or ... well, you get the idea. Something that tells me the status of each table, so I know where to run to next that too would be extremely helpful. Maybe with all of these things done, the chef would no longer have a reason to yell at me.

The Why W8 solution: Because of the integration of all components of the app, once the customer orders their food it will be relayed directly to the kitchen eliminating the middleman where things can

get lost in translation. Once an order is ready, a notification will alert the server prompting them to deliver the food to the corresponding table. Once the food is delivered the server simple presses "Delivered" for the corresponding table which will remove it from the list.

What Makes Why W8 Better?

Taking a look at [1], we see that this group's proposed solution involves a tablet at the front of the restaurant where they input their information. This can cause lines to form as people wait for others to finish in front of them, defeating the purpose of trying to speed up the dining process. Why W8 addresses this problem by allowing customers to download our application directly to their phone and select a table for a specific group size without having to use a tablet. Of course, there will be backup tablets for the special occasions where customers do not own a smartphone.

Further, in [1] the payment is done through the tablet when the customers finish their meals, which we believe can cause some dishonest customers to avoid paying their bill. We address this by having the customer pay on their phone as soon as their order is placed, otherwise, the order does not go through in our system for the chef to cook. We also added QR codes on the tables to allow customers to confirm their table when they enter the restaurant, as well as scan the code when they are leaving the restaurant to let our system know that the table has opened up. If for some reason the customers to do not scan the QR code upon leaving, the waiter will be able to scan it to override the system.

Why W8 also has its own features not seen in other apps, such as the rating and favorites system. Customers are given the option to create an account on the app so that they can keep track of their "favorite" foods for easy re-order and also rate foods that they have tried. This gives all customers, even those who do not have an account, the ability to view the top rated foods as well as the most favorited foods. These appear at the top of the menu when the customers get to the ordering screen.

1.2 Glossary of Terms

Admin
The restaurant owner/manager utilizing the reports generated by the application.

Application (App)
A software program designed to perform specific functions for a user.

Busboy
A busboy is a person who works in the restaurant and catering industry clearing tables, taking dirty dishes to the dishwasher, setting tables. In this report, a busboy would get an alert when the customers finished.

Chef
A chef is a trained professional cook who is proficient in food preparation, often focusing on a particular cuisine. In this report, the chef is the person prepares the food after the customers confirm their order in our app.

Cross-Platform
An application that can be used on multiple types of devices, such as varying phone models.

Customer
A person or organization that buys goods or services from a store or business. In this report, the customer is the person utilizing the application, and not the one requesting the proposed system.

Customer Account
A user account to access the application as a customer that allows them to reserve tables and stores their preferred meals, ratings, etcetera.

Customer Service
Customer service is the process of ensuring customer satisfaction with a product or service. In this report, customer can contact the restaurant or give some comments or complaint in the interface of customer service.

Database
A structured set of data stored on the Why W8 network, certain parts of which are accessible to certain user accounts.

Dine-In
To order and eat food within a restaurant.

Favorite Rating
A score that reflect customers' satisfaction. The application shall allow the customer to rate and "favorite" meals. "Favorite" meals will be highlighted in the customer's account for future orders.

Floor Layout
Shows all tables in the restaurant along with their respective status for the customer's viewing purposes.

Guest Account
An account almost identical to a customer account minus the features of storing information such as favorite meals and personal ratings.

Host
Assigns seats to people who come to the restaurant, especially those who wish to not use the app. Changes the status of tables from "Occupied" to "Vacant".

Menu
In this report, a menu is a list of food with pictures and favorite rates. Customers can view or order food in the menu.

Manager
Manages inventory, payroll, employee list and charts, customer's bills to override any issues, and the log-in interface to prevent unauthorized access of admin panel.

Order Queue
A list of orders that are placed in first in, first out order (FIFO). The orders are sent to the chef's PC, where the chef can then view and prepare the meals. [6]

QR Code
QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix barcode (or two-dimensional barcode. In this report, the QR code is used for customers confirming their table number. [5]

Rating
A position or standing of something, such as how popular some meal is amongst customers determined by their feedback and stored on the Why W8 database.

Reservation
An arrangement to secure a table ahead of time.

Restaurant System
Customers can view all available tables and reserve a specific table at a specific time using the android application.

Screen
A specific window displayed on the Why W8 user interface that provides convenient functionality for the user.

Tablet
A portable thin computer that utilizes a touchscreen as its primary user interface.

Take-Out
Food that is sold at a restaurant and packaged so that the customer can eat it elsewhere.

Tip
A small percent of money to be given to the waiter/waitress.

User Account
A private location on a network server for an individual who utilizes the application to store information such as their username, password, etcetera.

User Interface
The visual aspect of the android application on the user's phone that allows user interaction with the system.

Waiter
A waiter is the person who works at a restaurant, attending customers—supplying them with food and drink as requested. In this report, waiters are working on delivering food to tables and serving for the special requirements.

Walk-in Queue
A list of table reservations that are placed in first in, first out order (FIFO). These reservations are made on the customer's end of our application, which is on their android phones.

2. System Requirements

2.1 Enumerated Functional Requirements

Table 2.1 Functional Requirements

Identifier	Priority	Requirement
REQ-1	5	The application shall allow customers to choose an open table based on the restaurant's seating chart. Once a table is chosen, customers shall scan a QR Code at the selected table to confirm their seat.
REQ-2	5	The application shall provide a menu once the table is confirmed, where the customer places their order.
REQ-3	5	The application shall send the order to the chef queue to be cooked once the customer has paid for their meal.
REQ-4	4	The application shall provide the option to make a table reservation at the restaurant.
REQ-5	3	The application shall show the customer an approximation of the remaining wait time for their meal.
REQ-6	5	The application shall alert the waiter when an order has been completed. The waiter will then be able to deliver the order to the designated table.
REQ-7	4	The application shall allow the customer to choose a take-out option, where orders are placed and picked up.
REQ-8	2	The application shall allow the customer to rate and "favorite" meal options. "Favorite" meals will be highlighted in the customer's account for future orders.
REQ-9	1	The application shall display to the admin food, customer, and inventory reports. Popularity of meals, number of customers, and remaining supplies will be available to view for the admin.
REQ-10	1	The application shall allow the customer to view the menu options without choosing a table or placing an order.
REQ-11	4	The application shall request the customer to scan the QR code found on the table once they are ready to leave.
REQ-12	4	The application shall ask the customer if they would like to leave a tip once they have confirmed they are leaving the restaurant. There will be suggested tip percentages, as well as customer input.
REQ-13	5	The application shall alert the busboy when a table is to be cleaned.

		When the customer scans the QR code at the table signalling that they are leaving the restaurant, the busboy will receive a notification.
REQ-14	2	The application shall request a login, or ask the customer to use a guest account to proceed.

In Section 1.1, we propose the solutions we believe to be the answer to the customer's statement of the problem. As we can see the host/hostess use case is satisfied by REQ-1, as the user is able to select a table on our app even without having to step inside the restaurant. The manager use case is satisfied by REQ-9, as the application gathers the data from the database of user ratings, inventory, favorite foods, etc. and displays detailed reports for the day. The busboy use case is satisfied by REQ-13, as the busboy receives a notification once the customer has left the restaurant that their table is dirty. The most important use case, for the customer, is satisfied by REQ-1, REQ-2, REQ-4, REQ-5, REQ-7, REQ-8, REQ-10, REQ-11, REQ-12 and REQ-13. All of these requirements give the customer the options to use our application to speed up the dining process by using our app for every step of their experience. The chef use case is satisfied by REQ-3, REQ-5, and REQ-6. Our app gives the chef the ability to start an order and give updates on the progress of a customer's meal. Finally, the waiter/waitress use case is satisfied by REQ-3, REQ-6, and REQ-12. The waiter/waitress does not have to write down every order or bring a receipt to the customer as everything is taken care of by our app. Find the acceptance test cases for each of the requirements below.

Table 2.2 Acceptance Test Cases for Functional Requirements

Identifier	Acceptance Test Case
REQ-1	If the user chooses to dine-in, the seating chart will be shown. A customer selects an open table, designated by the color green. If a customer selects a green colored table, they will be asked to scan a QR code, otherwise, they will not be sent to the next screen if they choose a red colored table.
REQ-2	Once the table is confirmed, the customer is directed to a QR scanner where they must scan the QR code given on their selected table. If the correct QR code is scanned, the menu screen shows where customers can place their order, otherwise, they will not be able to proceed.
REQ-3	After placing their order, the customer will be asked to pay for their meal. If they make a payment, the order will be sent to the chef queue to be cooked, otherwise theri meal will remain on hold until a payment is made.
REQ-4	At the beginning screen, there will be a reservation option. Clicking this will show the seating chart for available tables at certain times. Once a free table is chosen, our database will be updated with the reservation.
REQ-5	After the chef starts cooking a customer's meal, the status of the meal, from preparation to completion will be displayed for the customer to see. There will also be an approximate time remaining shown for the customer.

REQ-6	When the chef updates the status of a meal as done, the waiter should receive an alert to pick up the meal and bring it to the appropriate table.
REQ-7	At the beginning screen, there will be a take-out option. Choosing this will bring the customer to the order screen where they choose their meal. After receiving payment, the order is sent to the chef and is prepared, giving the customer an estimated time to pickup.
REQ-8	Rating a food on a scale from 1 to 5 adds the rating to the database and updates the average rating for the food item. Favoriting a meal adds a heart next to the food and highlights the food the next time the customer uses the app.
REQ-9	The application gathers data from the current day from the database and produces an accurate report based on that data.
REQ-10	At the beginning screen, there will be a menu option. Choosing this allows the customer to view the menu without having to make an order.
REQ-11	After making payment, the app will ask the customer when they are ready to leave. Once ready, they will be taken to a QR scanner. Scanning the QR code marks the table as dirty for the busboy and free the table in the seating chart.
REQ-12	Scanning the QR code to confirm leaving should bring up a tip screen. The customer can choose to leave a tip by picking the suggested tips, or input their own tip.
REQ-13	Scanning the QR code to confirm leaving should send a notification to the busboy that the table is dirty and should be cleaned.
REQ-14	The application uses a customer account to keep track of ratings and favorite foods, but if the customer does not want an account, they can use a guest account. Picking either will bring them to the same menu order screen.

2.2 Enumerated Nonfunctional Requirements

Table 2.3 Nonfunctional Requirements [2]

Identifier	Priority	Requirement
REQ-15	1	The application should be cross-platform for maximum usability to the general audience
REQ-16	5	The application should have a high mean time between failure (MTBF) and high mean time to recovery (MTTR) to ensure reliability
REQ-17	3	The application should be intuitive and easy to use
REQ-18	3	The application should look aesthetically pleasing and modern

REQ-19	4	The application should have minimal transitions from each screen/action
REQ-20	5	The application should run as a service to save resources on the user's device
REQ-21	3	The application should be designed to handle an overestimated number of users for consistent throughput and availability
REQ-22	3	The application should be designed to make updates and fixes easy to implement

2.3 On-Screen Appearance Requirements

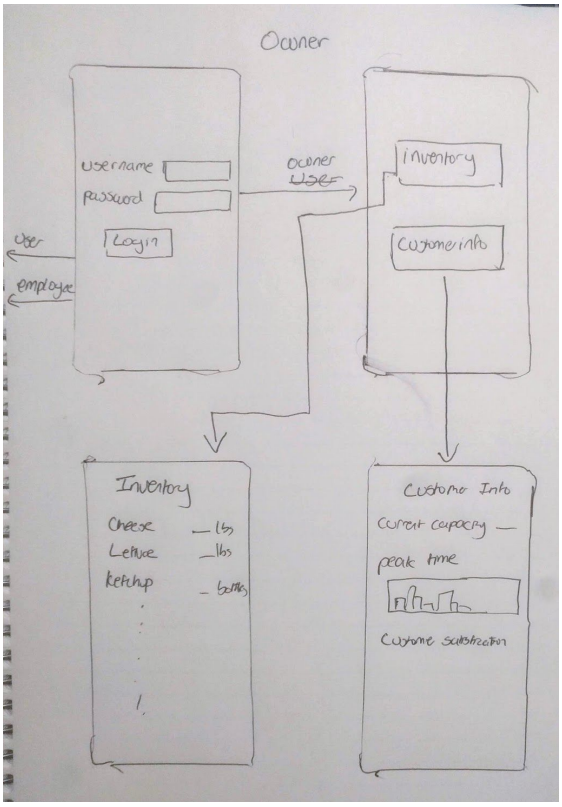


Figure 2.1 User interface and path for the owner/admin

Table 2.4 On-Screen Requirements for the Owner/Admin

Identifier	Priority	Requirement
REQ-23	1	Two text fields are used to take in values for username and password. A button is used to send the information to the system which determines the authorization of the user.

REQ-24	1	On next screen, two buttons are used to determine type of information the owner wants to look at. Clicking on the button takes owner to the page linked with that button.
REQ-25	3	On the Inventory screen, there is a list of items in stock which is updated in real time. Items running low are marked accordingly.
REQ-26	3	On the Customer info screen, there are statistics such as the current number of people in the restaurant. There is also a graph showing the amount of customers for each hour.

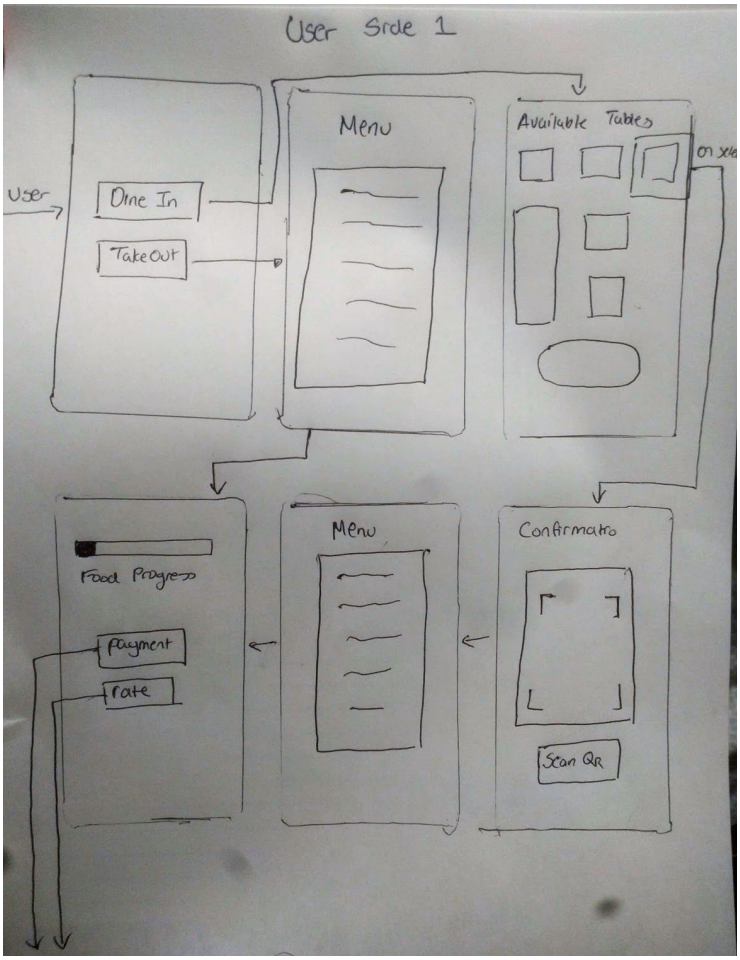


Figure 2.2 User interface and path for the customer

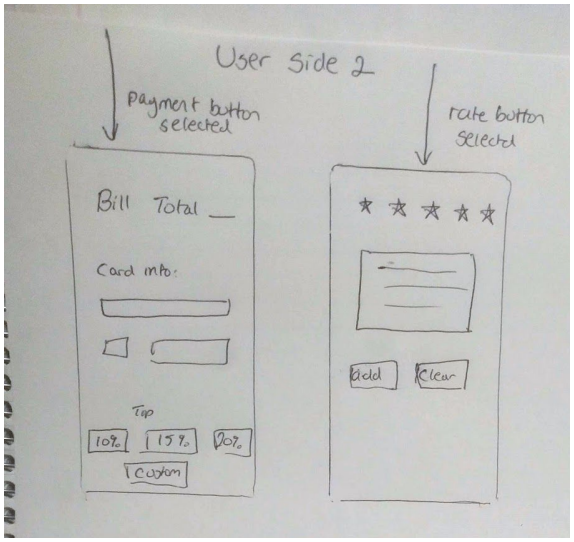


Figure 2.2 (Cont.) User interface and path for the customer

Table 2.5 On-Screen Requirements for the Customer

Identifier	Priority	Requirement
REQ-27	1	Two buttons appear for customer to choose to dine in or take out. Button leads to corresponding pages.
REQ-28	1	If the takeout button is clicked, the menu page appears for customers to select what they want to order. There is a list of food with specifications which show when an item is clicked.
REQ-29	1	If the dine in button is clicked, the available tables appear. This shows the tables available labeled as green. The customer can choose the table they want and sit there.
REQ-30	2	Upon table selection, confirmation page appears. This requires customers to scan the qr on the table to prove they are sitting in the correct table. There is an on screen camera and the scan QR button takes the picture.
REQ-31	1	After confirmation, user is taken to menu page again as with take out.
REQ-32	5	Once food items are selected, this page appears. Contains a food progress bar which is pre-programmed to a timer of expected time. There are also two buttons each linked to corresponding screens.
REQ-33	2	If the payment button is selected, payment screen appears. This includes bill total that is printed as text. There are also text fields that take in card information. There are also buttons to add in tips to the total.
REQ-34	5	If the rate button is selected, the review page appears. Here customers can leave a review for the food they ate. There is a text field for the customer to

write the review on as well as a button to submit the review.

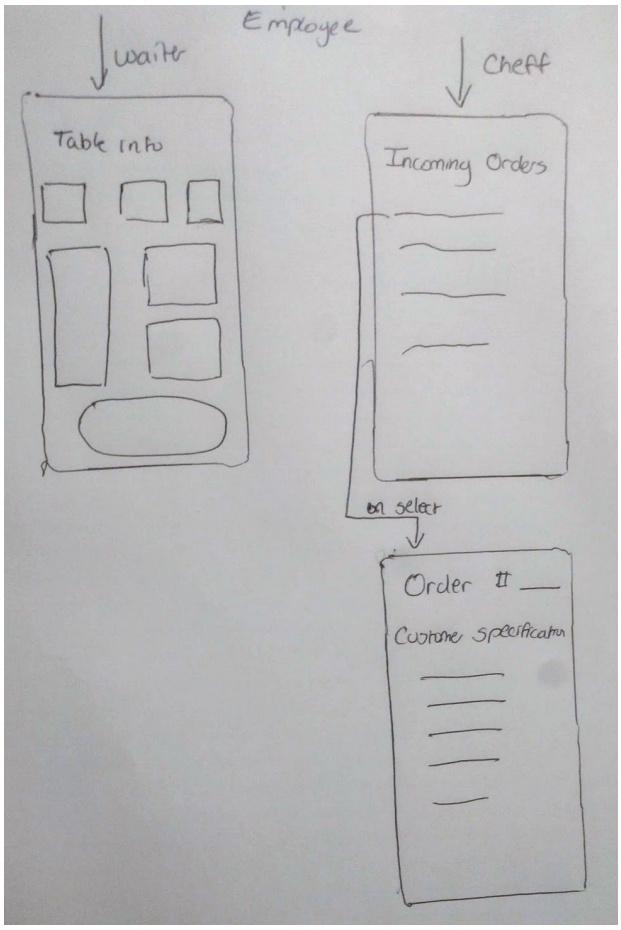


Figure 2.3 User interface and path for the employee (general)

Table 2.6 On-Screen Requirements for the Employee (General)

Identifier	Priority	Requirement
REQ-35	1	If the employee is a waiter, a screen with table info appears. These tables are marked according to duties the waiter must perform. If the table is dirty, the waiter needs to clean it. If the table has an order attached that is ready the waiter will be informed to deliver the food there. The waiter also has the ability to make tables available.
REQ-36	1	If the employee is a chef, this screen with incoming orders appear. It contains a queue of what the customers ordered. The chef can select to get more information which brings up the next screen.
REQ-37	2	If an order is expanded, it appears on this screen. This contains text information about the order details which includes customer specification. The chef can press a button to confirm he is currently making that dish.

3. Functional Requirements Specification

3.1 Stakeholders

The following are the stakeholders who will have the most interest to design and run the system efficiently.

- ➢ Restaurant owner
- ➢ Employees i.e Manager, Waitress, Busboy, Host, Chef, and other employees
- ➢ Software Developers i.e front-end, back-end, full-stack developers/programmers

Alongside the restaurant owner seeing their business become more efficient, all of the employees simply would benefit from a streamlined process. On top of this, the customer, who is one of the most important aspects of the restaurant world, would make great use of the all-in-one application for their restaurant outings. Lastly, software developers on our team have a great interest in assuring our application runs smoothly and makes the whole restaurant process truly efficient. With this in mind, future software developers will be able to build and add more features in the future with the restaurant owner/manager in mind; specific customizations, niche features, etc.

3.2 Actors & Goals

Table 3.1 Initiating Actors

Actor	Role	Goal
Customer	The customer uses the application to either choose to order food for take-out, or dine in and select their table before entering the restaurant. The customer also places their order in the system.	The goal of the customer is to have an excellent experience with minimal wait times for all aspects of the restaurant dining process.
Guest	The guest is a special type of customer which does not have an account. The guest has the same role as the customer, without being able to favorite food.	The goal of the guest is to have an excellent experience with minimal wait times for all aspects of the restaurant dining process.
Manager	The manager uses the application to review the daily reports associated with their restaurant, including trends, ratings, inventory, customer peak times, etc.	The goal of the manager is to understand the important aspects of their restaurant as easily and quickly as possible.

Table 3.2 Participating Actors

Actor	Role

Chef	The chef waits for the customer to place their order(s) if there are no orders in the queue. Otherwise, the chef begins cooking the top meal in the queue. The chef also updates the status of the customer's meal so that an estimate for the remaining cooking time can be relayed to the customer.
Waiter/ Waitress	The waiter/waitress receives notifications from the database that a customer's order is ready. Once notified, the waitress retrieves the customer's meal and brings it to their table.
Busboy	The busboy receives notifications from the database that a customer has left their table, thus marking the table dirty. Once notified, the busboy goes to the appropriate table to clean up.
Database	The database records a customer's orders, table selection, favorite foods, and ratings.

3.3 Use Cases

3.3.1 Casual Description

The summary use cases are as follows:

UC-1: Dine-in - Allows the customer to dine inside the restaurant.
Derived from requirement REQ-1, REQ-2, REQ-3, REQ-5, REQ-6, REQ-8, REQ-11, REQ-12, and REQ-13.

UC-2: Take Out - Allows the customer to order food from their home for pick-up.
Derived from requirement REQ-7.

UC-3: Reservation - Allows the customer to reserve a table for a later date or time without having to call the restaurant.
Derived from requirement REQ-4.

UC-4: View Menu - Allows the customer to view the menu (mandatory sub use case, «include» from UC-1: Dine-in and UC-2: Take Out). Extension Point: the customer is able to rate food items. Extension Point: the customer is able to favorite a food item.
Derived from requirement REQ-2, REQ-8, and REQ-10.

UC-5: Table Selection - Allows the customer to select an open table in the seating chart (mandatory sub use case, «include» from UC-1: Dine-in and UC-3: Reservation).
Derived from requirement REQ-1 and REQ-4.

UC-6: QR Scan - Allows the customer to confirm their table by scanning the QR code associated with the table they chose in UC-5 (mandatory sub use case, «include» from UC-1: Dine-in).
Derived from requirement REQ-11

UC-7: Payment - Allows the customer to pay for their meal, after which the chef will begin cooking (mandatory sub use case, «include» from UC-1: Dine-in and UC-2: Take Out).
Derived from requirement REQ-3.

UC-8: Rate Food - Allows the customer to rate food they have eaten at the restaurant on a scale from 1 to 5 (optional sub use case, «extend» UC-4: View Menu).
Derived from requirement REQ-8.

UC-9: Favorite Food - Allows the customer to favorite a food on the menu so that it is highlighted the next time they order at the restaurant (optional sub use case, «extend» UC-4: View Menu).
Derived from requirement REQ-8.

UC-10: Login - Allows the customer to utilize the restaurant options and track their rated and favorited foods (mandatory sub use case, «include» from UC-1: Dine-in).
Derived from requirement REQ-14.

UC-11: Generate Report - Allows the manager to view important aspects about the restaurant generated from the daily customers in easy to follow charts and graphs.
Derived from requirement REQ-9.

UC-12: Register - Allows a guest to create an account so that he or she can access the application features.
Derived from requirement REQ-14.

UC-13: Estimate Time - Allows the chef to estimate the remaining cooking time for a customer's meal.
Derived from requirement REQ-5.

It is important to note how the waiter/waitress and busboy are not given their own use cases because they are not initiating actors. They only participate in the actions which the customers initiate, and the database relays notifications to them. Dismissing notifications is not shown here for simplicity.

3.3.2 Use Case Diagram

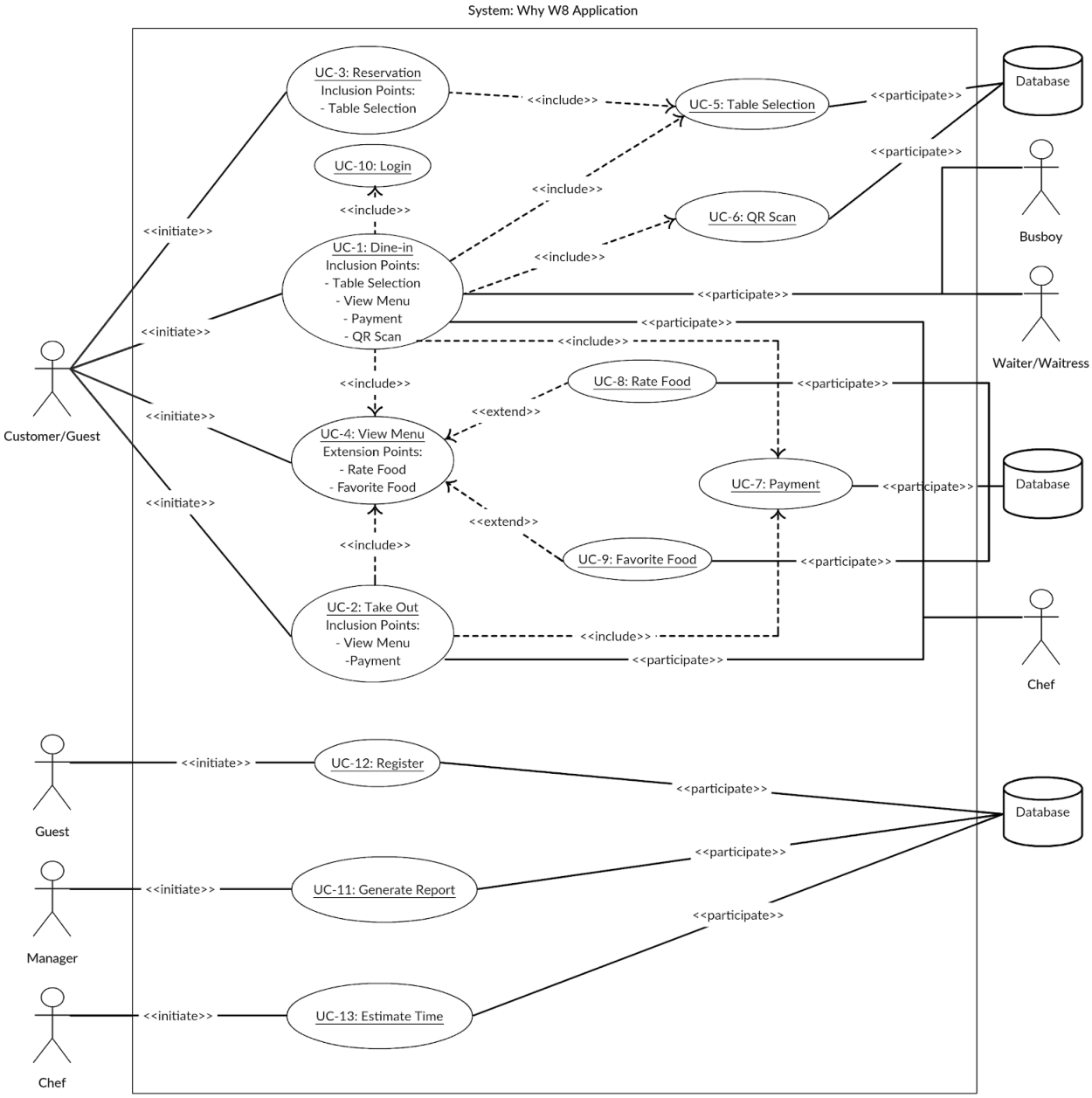


Figure 3.1 Use Case Diagram for the Why W8 application

3.3.3 Traceability Matrix

Req't	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13
REQ-1	5	X				X								
REQ-2	5	X			X									
REQ-3	5	X						X						
REQ-4	4			X		X								
REQ-5	3	X												X
REQ-6	5	X												
REQ-7	4		X											
REQ-8	2	X			X				X	X				
REQ-9	1											X		
REQ-10	1				X									
REQ-11	4	X					X							
REQ-12	4	X												
REQ-13	5	X												
REQ-14	2										X		X	
Max PW		5	4	4	5	5	4	5	2	2	2	1	2	3
Total PW		38	4	4	8	10	4	5	2	2	2	1	2	3

Figure 3.2 Traceability matrix mapping the system requirements to use cases

3.3.4 Fully-Dressed Description

UC-1 : Dine-in
Related Requirements: ● REQ-1, REQ-2, REQ-3, REQ-5, REQ-6, REQ-8, REQ-11, REQ-12, REQ-13 **Initiating Actor:** ● Customer, Guest **Actor's Goal:** ● To choose to dine-in at the restaurant **Participating Actors:** ● Waiter/waitress, busboy, database

Preconditions:
- User has application loaded

Postconditions:
- User is shown available tables

Flow of Events for Main Success Scenario:
1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks login
3. ← The system prompts for account information
4. → The customer inputs account ID/PW
5. ← The system displays option for dine-in, take out, reservation, view menu
6. → The customer clicks dine-in

Flow of Events for Alternate Success Scenario:
1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks guest user
3. ← The system displays option for dine-in, take out, reservation, view menu
4. → The customer clicks dine-in

UC-5 : Table Selection

Related Requirements:
- REQ-1, REQ-4

Initiating Actor:
- Customer, Guest

Actor's Goal:
- To select an open table for dining in

Participating Actors:
- Database

Preconditions:
- User has application loaded
- User has chosen to dine-in

Postconditions:
- User is prompted to confirm table

Flow of Events for Main Success Scenario:
1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks login
3. ← The system prompts for account information
4. → The customer inputs account ID/PW
5. ← The system displays option for dine-in, take out, reservation, view menu
6. → The customer clicks dine-in
7. ← The system checks database to find available tables
8. ← The system displays available tables for customer
9. → The customer clicks an available table
10. ← The system confirms table, updates database

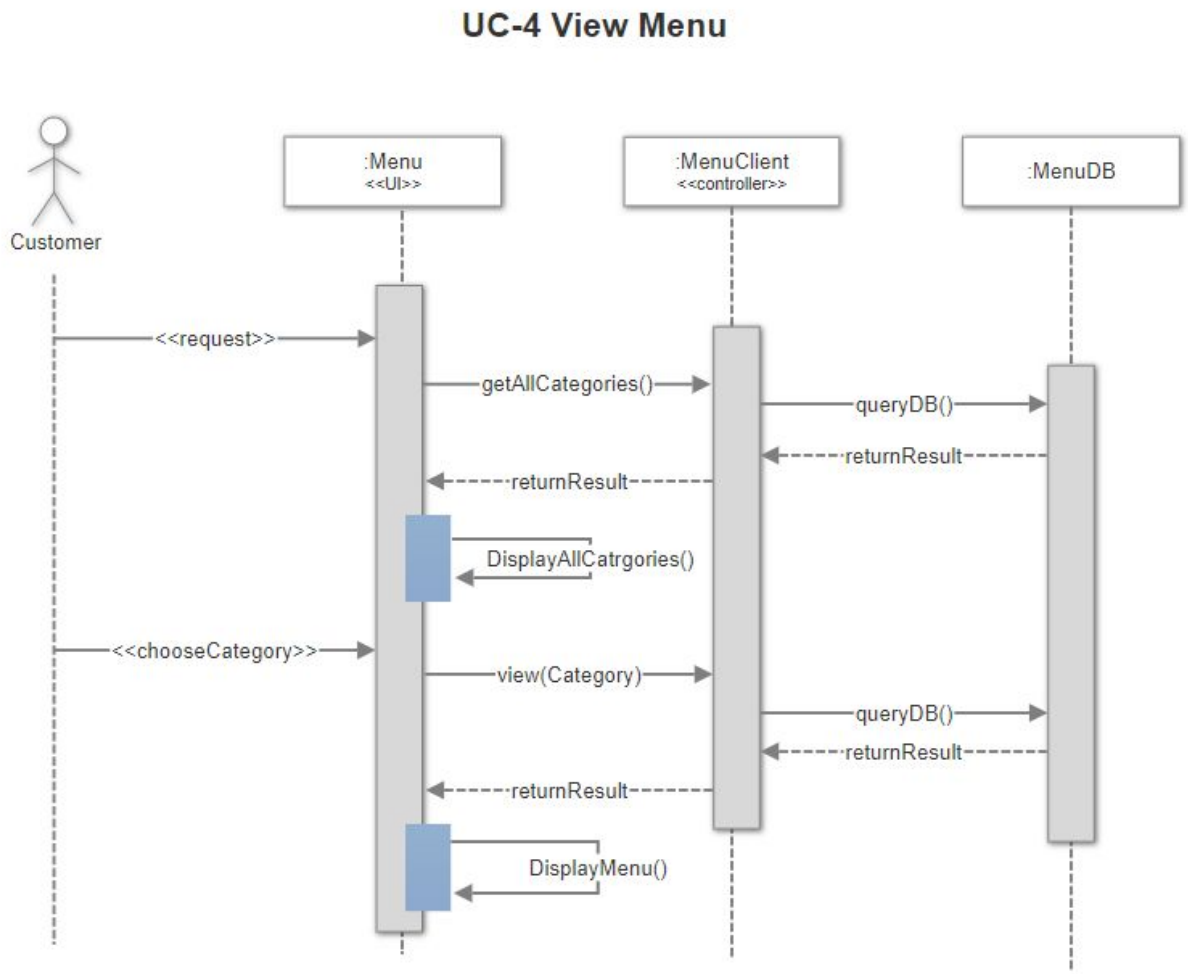
Flow of Events for Alternate Success Scenario:

1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks guest user
3. ← The system displays option for dine-in, take out, reservation, view menu
4. → The customer clicks dine-in
5. ← The system checks database to find available tables
6. ← The system displays available tables for customer
7. → The customer clicks an available table
8. ← The system confirms table, updates database

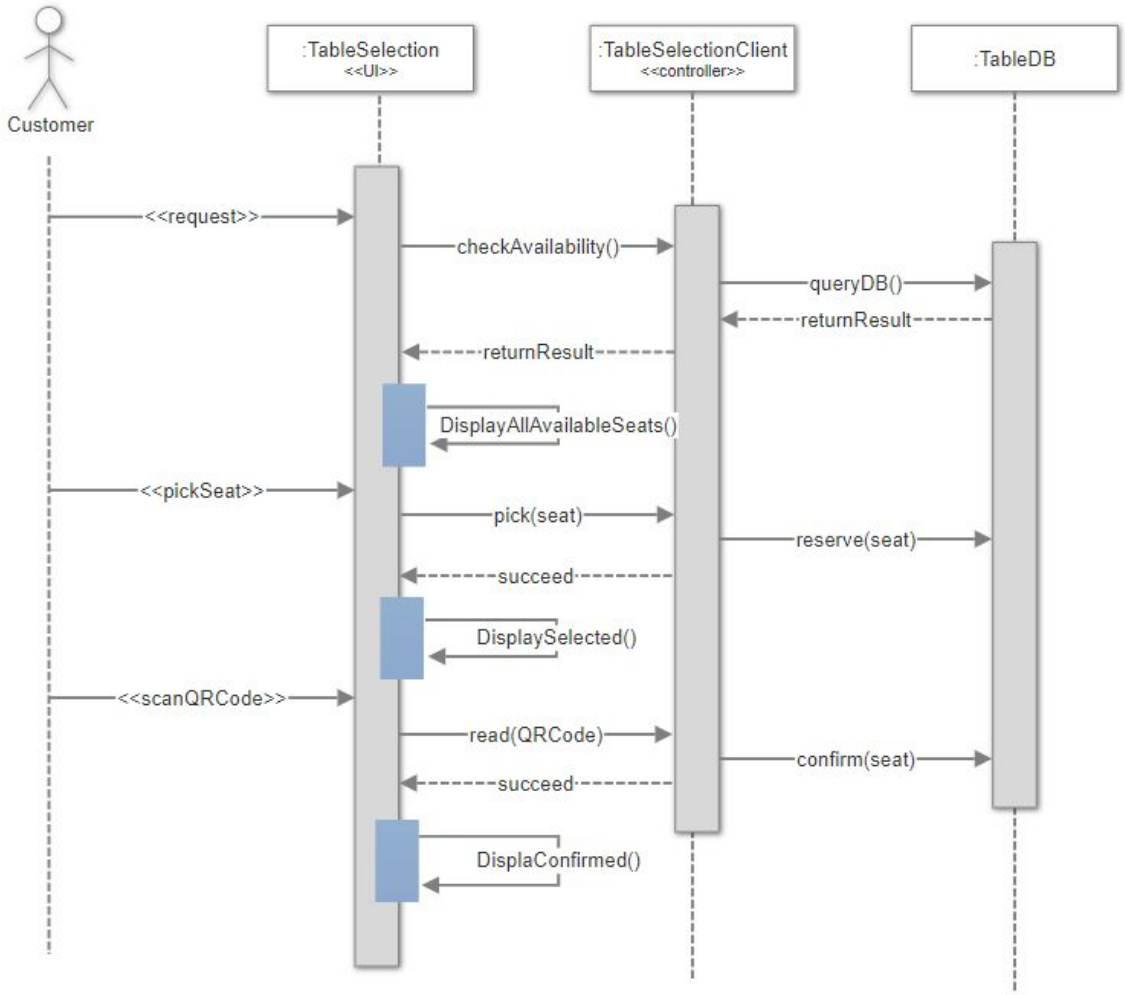
UC-4 : View Menu
Related Requirements: ● REQ-2, REQ-8, REQ-10 **Initiating Actor:** ● Customer, Guest **Actor's Goal:** ● To view the menu of food/drinks **Participating Actors:** ● Database **Preconditions:** ● User has application loaded **Postconditions:** ● N/A
Flow of Events for Main Success Scenario: 1. → The customer opens the application to the homepage for login/guest user 2. → The customer clicks login 3. ← The system prompts for account information 4. → The customer inputs account ID/PW 5. ← The system displays option for dine-in, take out, reservation, view menu 6. → The customer clicks view menu 7. ← The system displays the menu **Flow of Events for Alternate Success Scenario:** 1. → The customer opens the application to the homepage for login/guest user 2. → The customer clicks guest user 3. ← The system displays option for dine-in, take out, reservation, view menu 4. → The customer clicks view menu 5. ← The system displays the menu

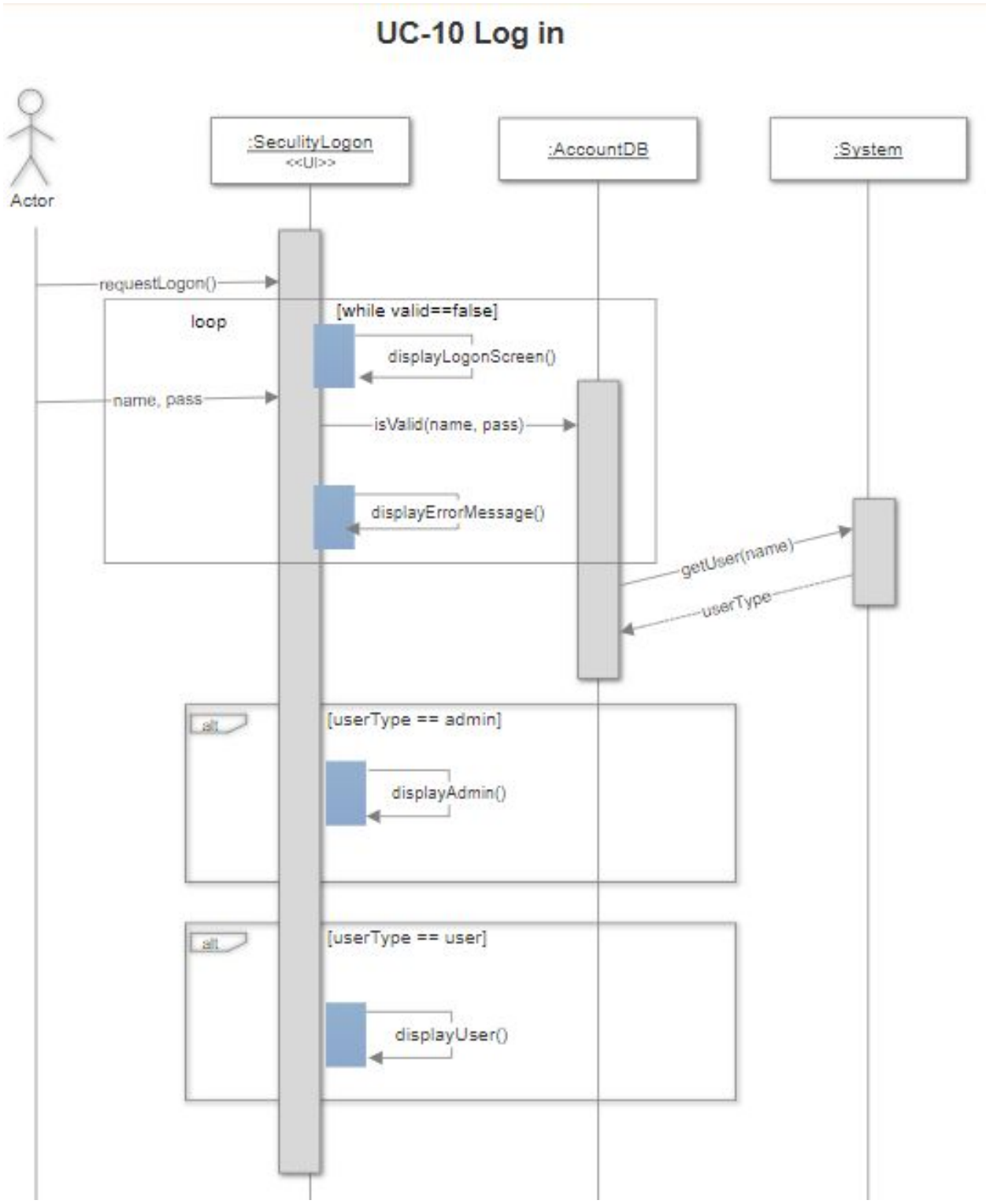
3.4 System Sequence Diagrams

UC-1: Dine-in is the most important use case, however it contains a combination of other use cases through inclusions and extensions, so it is not demonstrated here due to the system sequence diagrams showing the flow of the system components. Since UC-1: Dine-in includes other use cases, we show those important use cases instead, as they show the flow of the components in our system.



UC-5 Table Selection




UC-10 Log in
Actor
:SeculityLogon
<<UI>>
:AccountDB
:System
requestLogon()
loop
[while valid==false]
displayLogonScreen()
name, pass
isValid(name, pass)
displayErrorMessage()
getUser(name)
userType
alt
[userType == admin]
displayAdmin()
alt
[userType == user]
displayUser()

4. User Specification

4.1 Preliminary Design

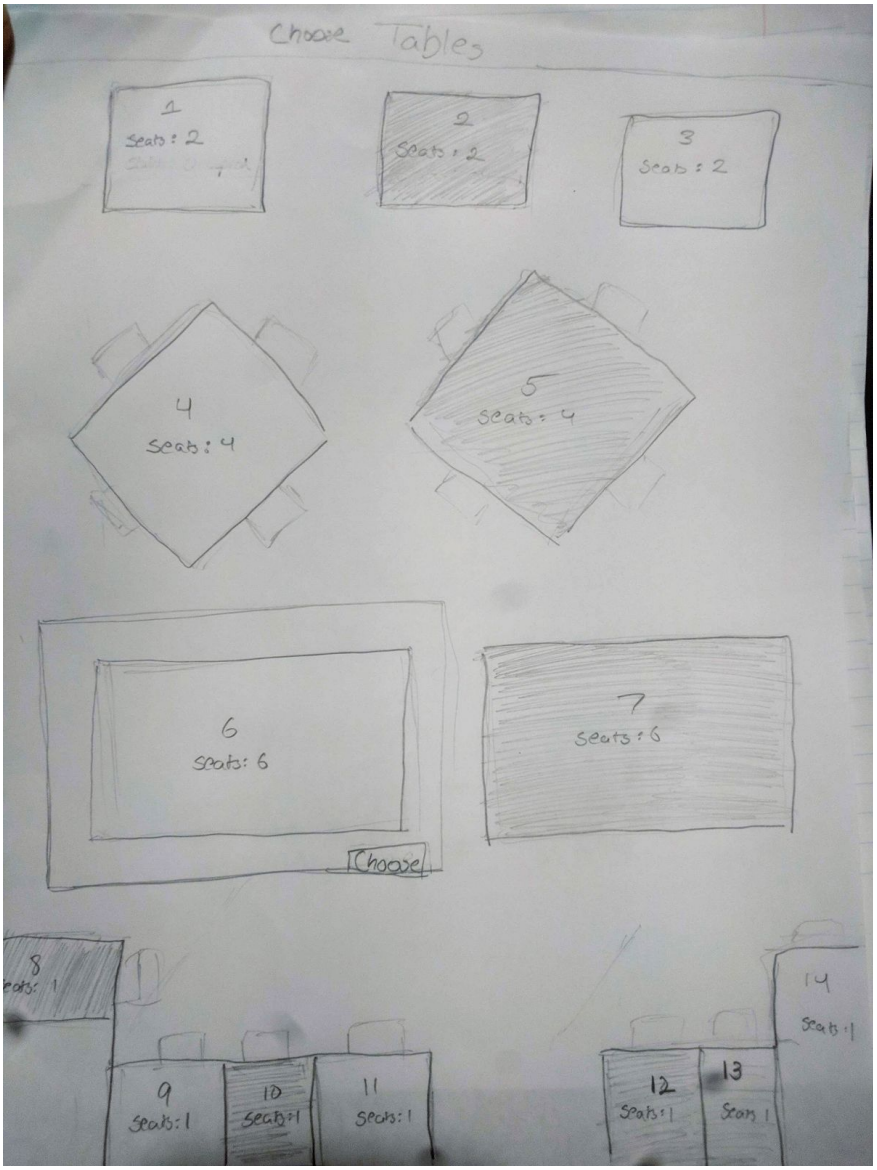
When designing an application the designer must keep the user in mind at all times. This requires the designer to make the interface as simple as possible for the user while still carrying out the expected functions.

Upon opening the app, the first screen will be the login page, where the user can either create an account, sign in to an already existing account, or use the app as a guest.

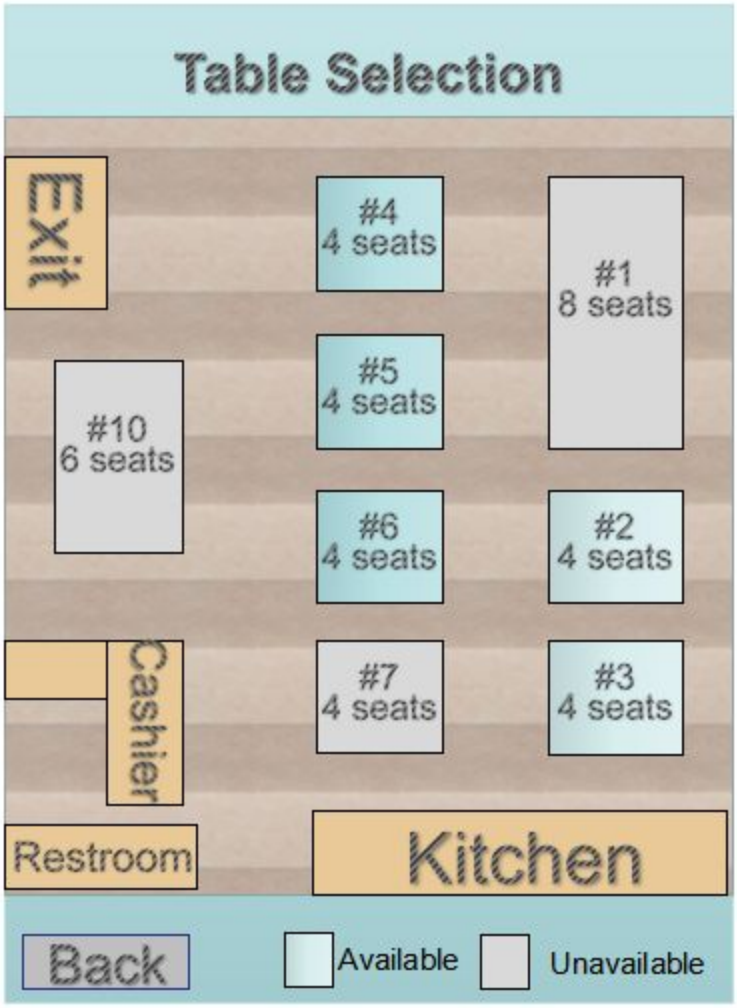
After signing in, the following screen will open. The customer can choose to either dine in or take out by clicking one of the corresponding buttons. If the customer simply wants to see the menu, they can click on the view menu button which will take them to a text screen with only the menu and no other functionality.

If the customer selects Dine In, they will be taken to the following screen shown below. The customer is given the option to select any table that is not shaded. The shaded tables indicate occupation and are disabled from being selected. The information for how much people each table can seat is also printed as a text on top of the table. The available tables are selectable entities, allowing the customer to click on the one at which they desired to be seated. Upon selection, times at which the table is available are shown as a list, from which the customer is required to select one of the available time slots.

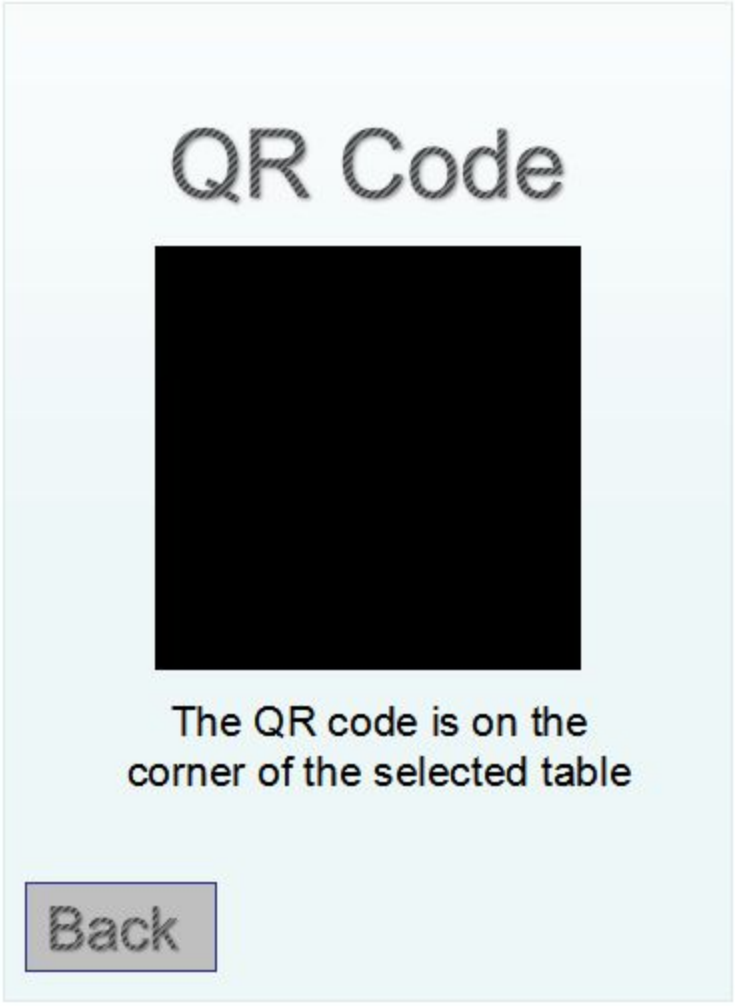
The waiter has access to a similar screen, but they also have the ability and responsibility to mark tables as available on the app. Once a customer leaves and the table is cleaned, the waiter does this by selecting said table and declaring it available for future customers.



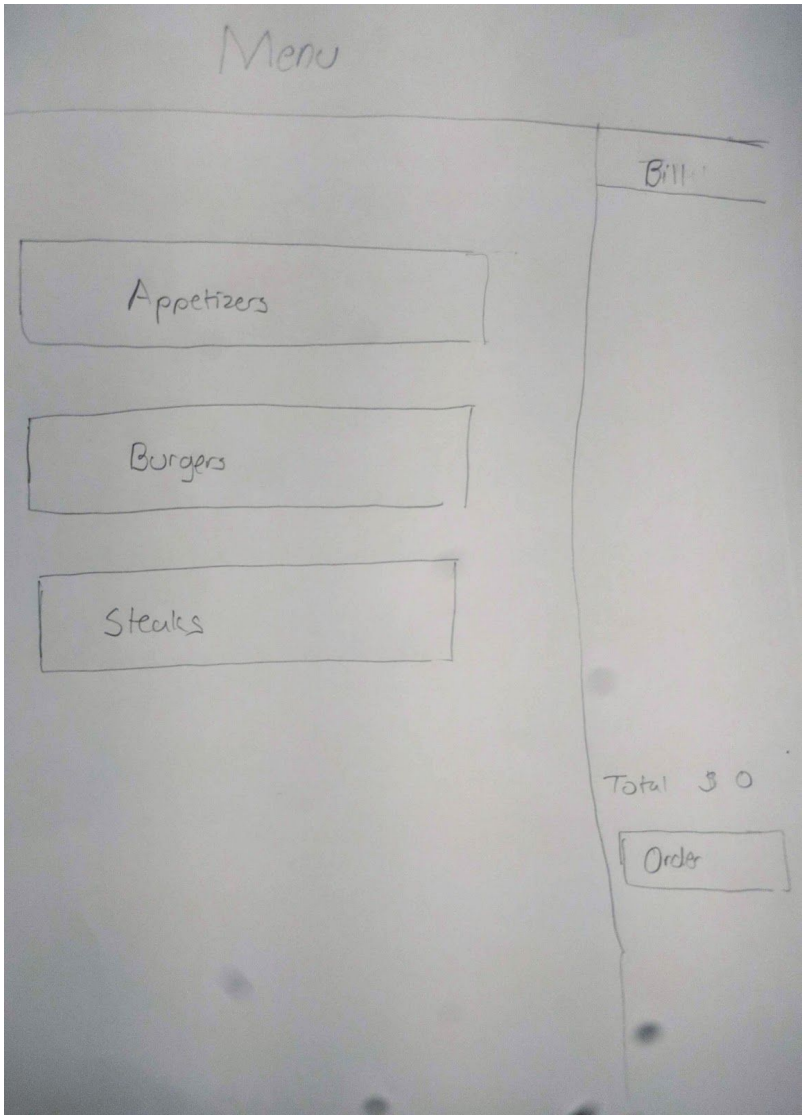
After listening to waiter suggestions as well as customer inputs, we organized the tables around to be more accessible by waiters. We also provided more references to better show where the customer can find the table. The changes are displayed on the screen below.



After selecting a table and time, the customer is taken to the screen below. This screen accesses the customer's phone camera and will request permission the first time it is used. The customer is prompted to center the QR code inside the frame given and then click the scan code button. This button will take a picture of the code and then confirm to see if it is associated with the right table.



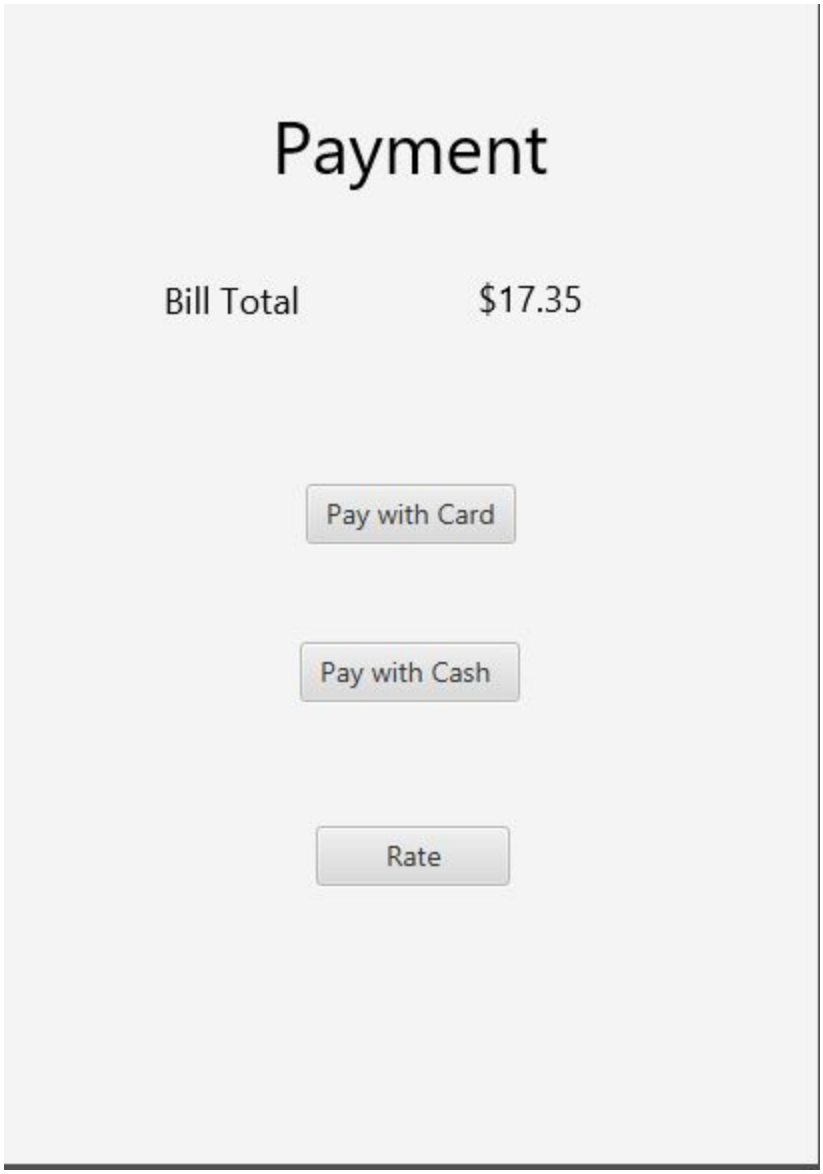
After confirming a table, the Menu Screen shows up. This same screen can also be accessed if the customer selected ‘Take-Out’ or ‘View Menu’. If the customer selected view menu, they can click navigate through the categories and view the food; they cannot click ‘Order’ as this button will be disabled. If the customer clicked take-out, they will follow through the menu just as if the customer had clicked dine in. The customer has the option to choose a category and click on the button to expand on said category.



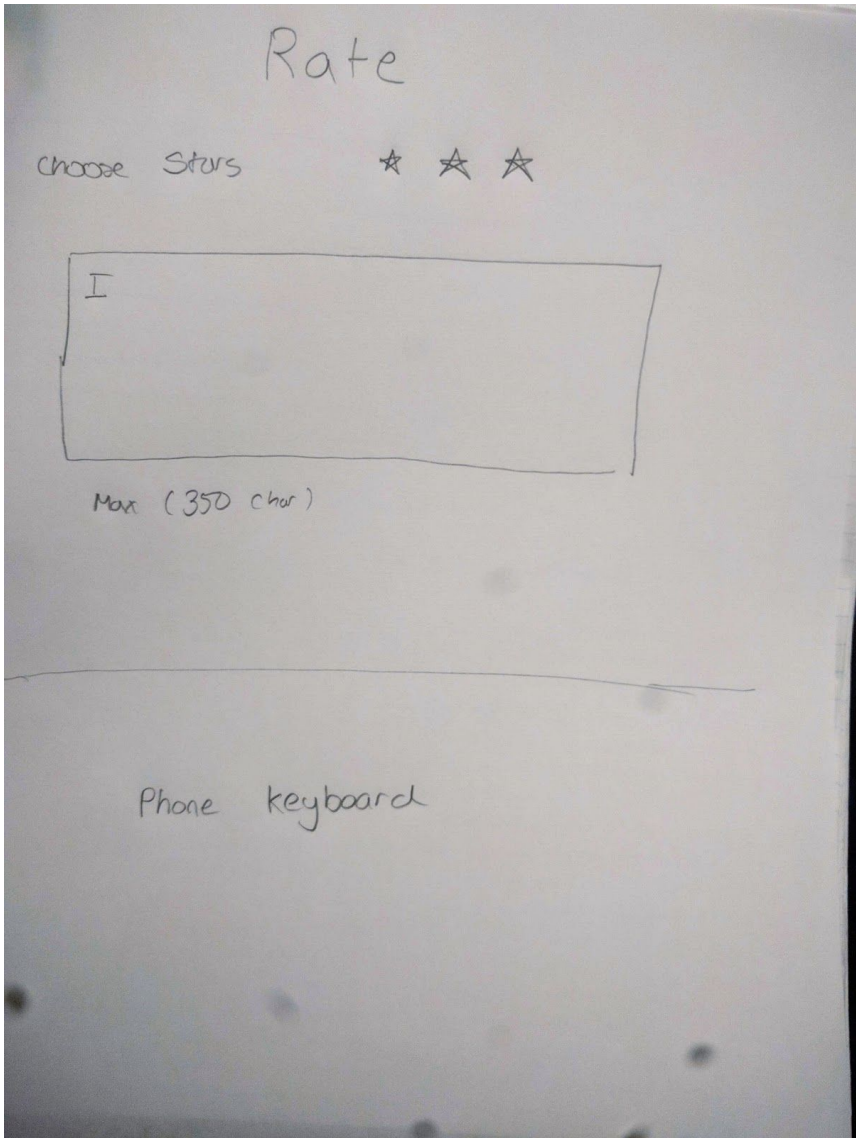
After designing the menu screen, we realized that the customer would benefit more from a simpler easier to access menu. And so on the next iteration we made the UI simpler and increased the sizes of the button as to be more appealing to the user.

Once a category is selected, the following screen appears with the expanded category. This screen contains related foods which are listed as selectable items. Clicking on one will add the item to the bill. The items also contain reviews left by previous customers. There is also a description provided by the admin on what the item contains.

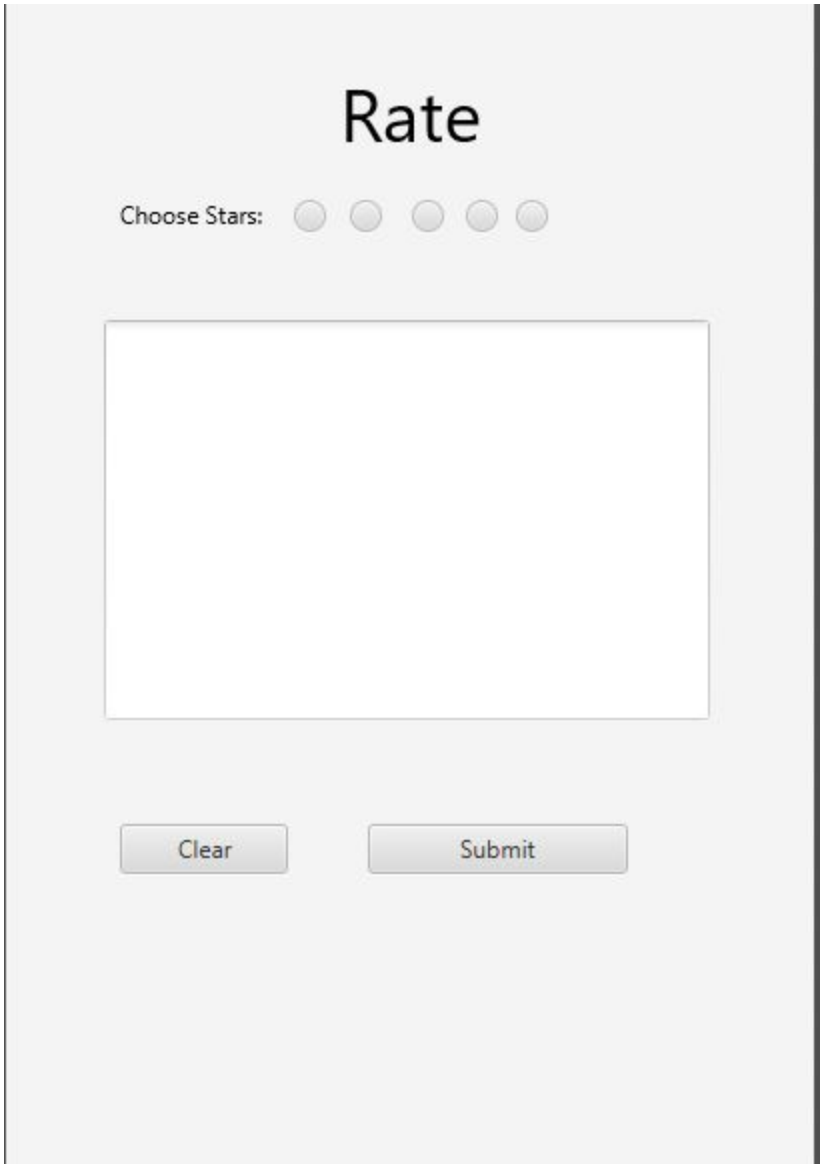
After the order button is selected, the following screen appears. In it the customer can choose to either pay with cash, which will request a waiter to come, or they can also choose to pay with a debit/credit card which will open fields through which the customer can input their card information. The customer can also choose to rate the food which will open a screen with the foods they ordered.



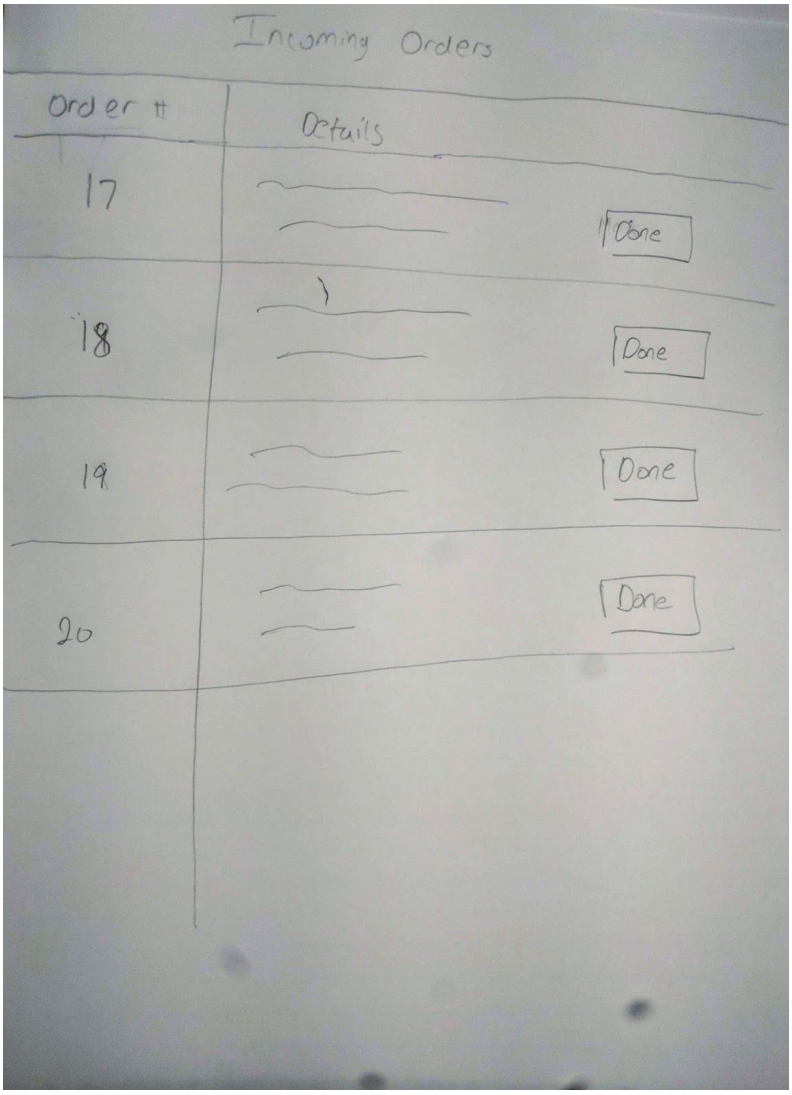
After the rate button is selected, the following screen appears. Here the customer can assign a number of stars to the food and then write their opinion on the textfield shown. Clicking on the textfield will open the phone keyboard and wait for the customer input.



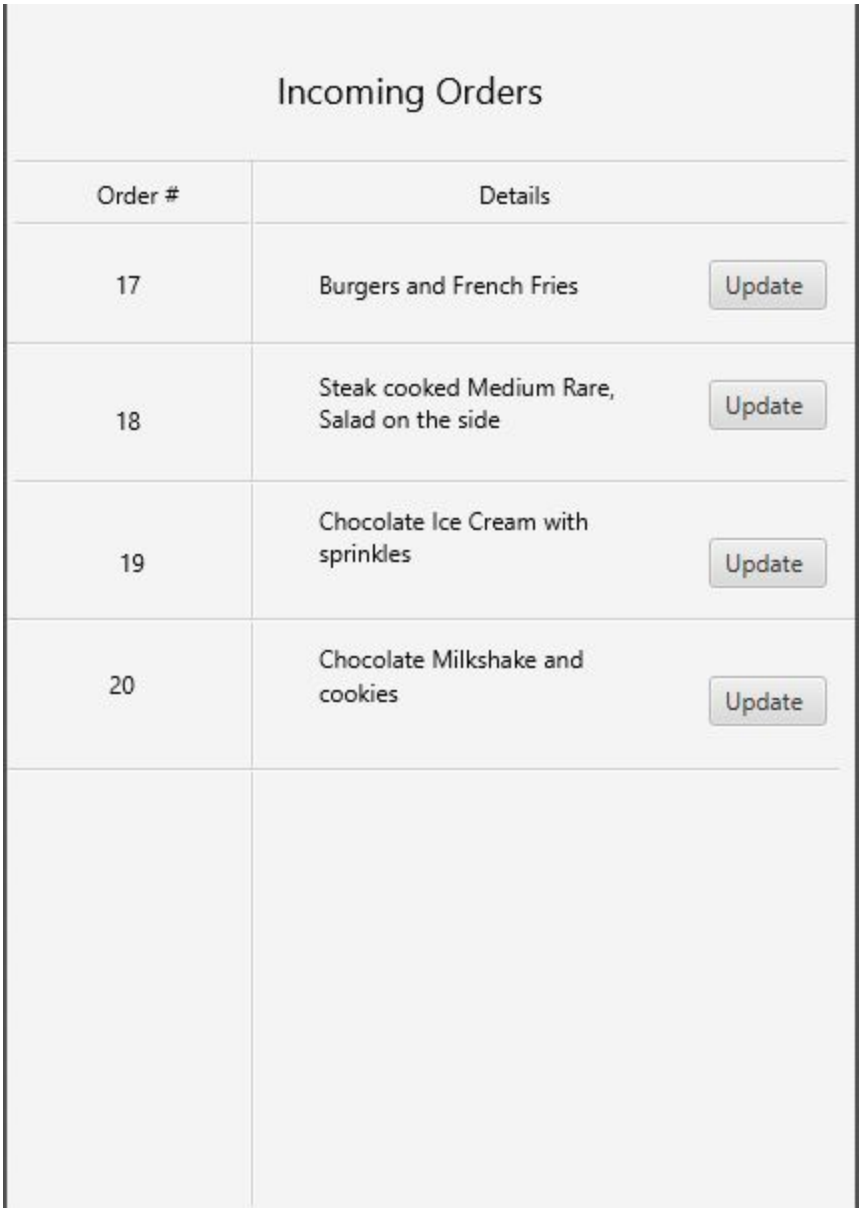
We updated this design for a better experience for the customer. This updated design is shown below.



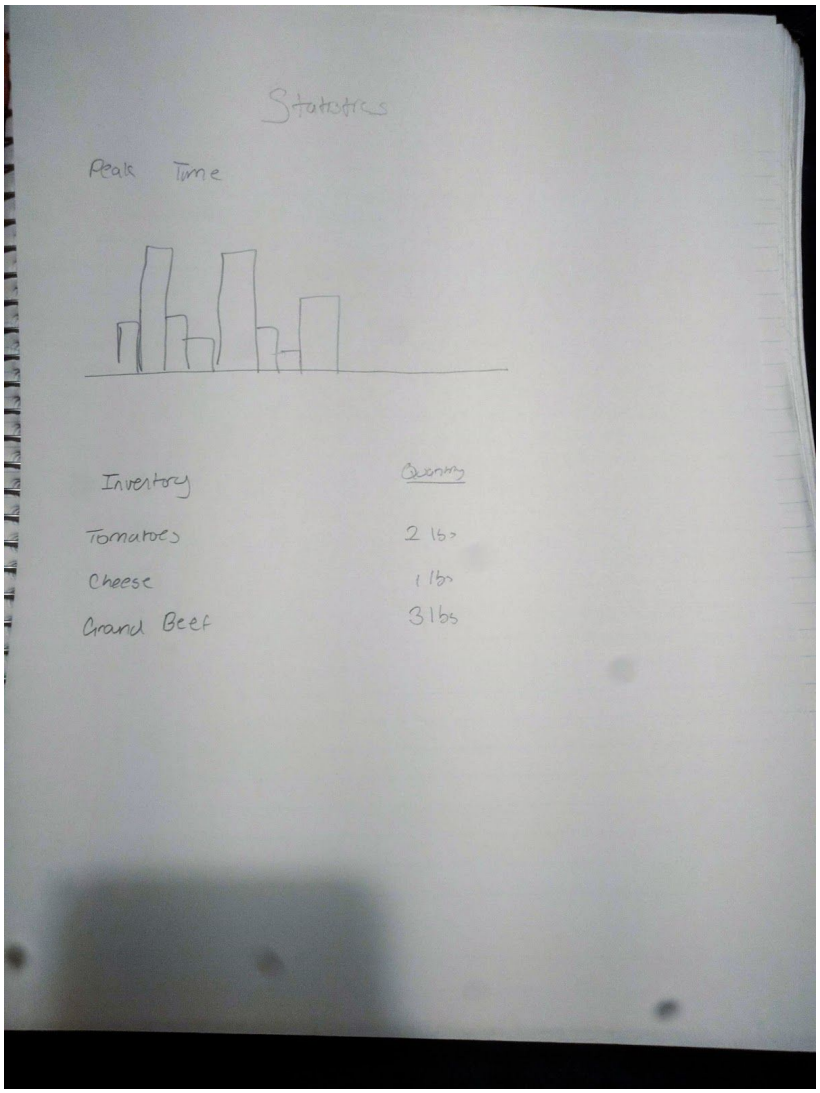
The chefs will see the screen below. Here a queue is displayed filled with orders from the customers as well as details on said orders. Based on the times at which the customers submitted their orders and the estimated preparation times of said orders, the application will display the optimal order in which to prepare the meals, though the chef can rearrange the queue if they so choose.



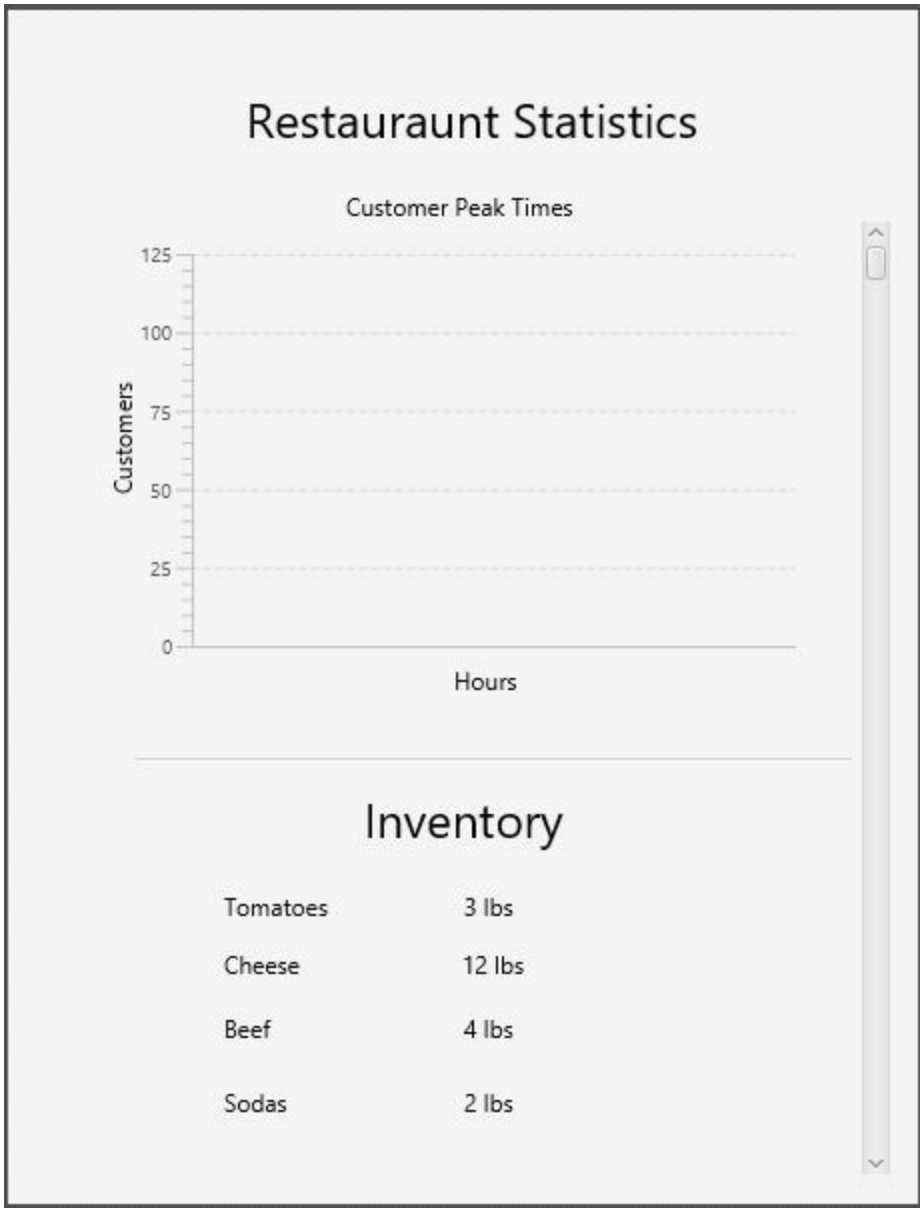
After designing a rough sketch for the incoming orders screen, we made some adjustments to make the Orders better to understand to the chefs. These changes can be seen the screen below.



The manager will see the screen shown below. Here statistics such as peak customer times for the day are shown as a graph. Moreover, inventory is shown along with how much of each item is remaining. This page can be scrolled through to see more information. All information will continuously update as statistics change.



After listening to complaints from the manager, we decided to change how we displayed our statistics. We kept the scrollable page but we added a more defined graphs and changeable stock menus. These adjustments can be seen below.



4.2 User Effort Estimation

Customer Interface:

The customer effort estimation varies greatly depending on the dining option that is chosen by the user, as well as the number of meals chosen. The payment method at the end also will vary depending on the type of customer.

Best Case Scenario (Customer ordering a meal):

- ➢ Customer selects Take Out option. (1 Click)
- ➢ Customer selects one dish to order. (1 Click)
- ➢ Customer selects checkout. (1 Click)
- ➢ Customer selects cash payment. (1 Click)
- ➢ Customer pays cash.
- ➢ Customer selects not to rate the meal and instead closes the app.

Total = 4 Clicks.

Worst Case Scenario (Customer ordering a meal):

- ➢ Customer selects Dine In option. (1 Click)
- ➢ Customer selects a time from the table reservation menu. (1 Click)
- ➢ Customer selects a table. (1 Click)
- ➢ Customer sits down and scans the QR code to confirm they are seated. (1 Click)
- ➢ Customer selects N dishes to order. (N Clicks)
- ➢ Customer selects check out. (1 Click)
- ➢ Customer rates each item they ordered. (N Clicks)
- ➢ Customer closes the app.

Total = 5 + 2*N Clicks.

Employee Interface:

The effort estimation for the employee interface depends on many different variables, but mainly it is based off of customer input.

Best and Worst Case Scenario (Chef):

- ➢ Chef reads menu items that are in queue to be prepared and clicks "Start dish." (1 Click)
- ➢ Chef completes cooking the dish and chooses "Dish complete." (1 Click)

Total = 2 Clicks.

Best Case Scenario (Server):

- ➢ Server sees that all dishes for table are prepared and clicks "Order delivered." (1 Click)
- ➢ Customer pays for meal via credit/debit card and payment is automatically processed.

Total = 1 Click.

Worst Case Scenario (Server):

- ➢ Server sees that all meals for table are prepared, brings them to the table, and then clicks "Order delivered." (1 Click)
- ➢ Server collects cash from customer and selects "Meal Paid." (1 Click)

Total = 2 Clicks.

Manager Interface:

The manager interface receives data from all of the other interfaces and logs them appropriately. The manager will be able to keep track of statistics such as inventory, income, and even the number of customers ordering meals within a tracked certain period of time. All of this information is then displays as graphs/charts for ease of use.

Best and Worst Case Scenario (Manager viewing restaurant statistics):

- ➢ Manager opens app and logs into admin portal. (1 Click)
- ➢ Manager is presented with different options for each statistic viewable, and chooses a menu item to observe customer's satisfaction of the product. (1 + N Clicks)
- ➢ Manager is brought to a graph of the items' sales progress hourly, for the current day.
- ➢ Manager has the option to change period of time of item's sales progress (daily, weekly, etc). (1 Click)
- ➢ Manager closes the app.

Total = 3 Clicks for best case.
Total = 3 + N Clicks for worst case.

5. Domain Analysis

5.1 Domain Model

5.1.1 Concept Definitions

Table 5.1 Concept Definitions

Responsibility	Type	Concept
R1: Coordinate activity between the customer, chef, waiter, busboy, etc.	D	Controller
R2: Keep the record of customer accounts with favorites and rated foods	K	Customer Profile
R3: Display the options for the customers, waiters, managers, chef, etc.	D	Interface
R4: Prompt the customer to select a table, scan the QR code, make a payment, etc.	D	Controller
R5: Store the customer order in the database	K	Communicator
R6: Queue incoming orders for the chef to complete	K	Order Queue
R7: Select order and send to a specific chef for cooking	D	Communicator
R8: Manage interactions with the database	K	DB Connection
R9: Record daily statistics for manager report	D	Analytic Calc
R10: Prevent invalid table selections	D	Table Status
R11: Allow chef to update estimated food time	D	Food Status
R12: Input to receive payment type and amount	D	Payment System
R13: Display change of table status when customer reserves or leaves and when busboy cleans	D	Table Status
R14: Display customer favorites and high rated foods at top of user profile for easy access	D	Interface
R15: Conclude the failed negotiations for selecting invalid options	D	Communicator

5.1.2 Association Definitions

Table 5.2 Association Definitions

Concept Pair	**Association Description**	**Association Name**
Customer Profile ⇔ DB Connection	Fetch customer's data from the database	QueryDB
Customer Profile ⇔ Interface	Display customer's option	Display
Interface ⇔ Controller	Allow the user to interact with the option the application (select a table, scan QR code, estimate food time etc...)	User Action
Communicator ⇔ DB Connection	Inject or modify data in the database	UpdateDB
Communicator ⇔ Order Queue	Send order and queue for the chef	Send Order
Controller ⇔ Analytic Calc	Display statistic for specific user	Display Statistic
Analytic Calc ⇔ DB Connection	Extract data from database for calculation	QueryDB
Controller ⇔ Food Status	Allow user to update or view the food status	Update Food Status View Food Status
Controller ⇔ Table Status	Allow user to view table status	View Table Status
Controller ⇔ Payment System	Allow user to complete the payment	Pay
Payment System ⇔ DB Connection	Store payment record to the database	Record Payment
Interface ⇔ DB Connection	Get the data from the database for the user	QueryDB

5.1.3 Attribute Definitions

Table 5.3 Attribute Definitions

Concept	Attribute	Description
Customer Profile	accountUsername	Associated username of the customer. Guest account is assigned if no account.
	accountPassword	Password of user account.
	accountFavorites	Contains all the reviews and favorites left by the customer.
Interface	currentInterface	Depending on what account type is logged in, show interface for that type of account only.
	rateMeal	Allows the user to rate a meal, and then have that rating stored and displayed.
Controller	tableList	Shows the customer the current tables available.
	tableConfirm	Allows the customer to confirm the table they sit at via scanning the QR code.
	paymentMade	Once the customer pays, the table is then confirmed and the chef begins to cook the meals.
Communicator	customerMeal	Each meal that the customer orders is stored in the database.
	customerMealOrder	The meals in the queue are ordered and sent to a specific chef, to balance the chef work load.
Order Queue	chefQueue	Contains information about the order that has been placed by the customer.

Analytic Calc	inventoryStats	Records statistics about restaurant inventory to alert manager when stock is low (daily, weekly, monthly).
	customerStats	Records statistics about peak customer times, most ordered meals, etc.
Table Status	tableStatus	Provides and updates status of each table, whether it is available, occupied, or dirty.
Food Status	orderStatus	Allows chef to update the status on currently being cooked meals, i.e time remaining.
	orderReady	Allows chef to update waiter and customer that food is cooked and ready.
Payment System	paymentMade	Updates the system whether or not the payment has been made.

5.1.4 Traceability Matrix

Table 5.4 Traceability matrix mapping the use cases to domain concepts

Use Case	PW	Domain Concepts								
		Customer Profile	Interface	Controller	Communicator	Order Queue	Analytic Calc	Table Status	Food Status	Payment System
UC-1	38	X	X				X			
UC-2	4	X	X				X			
UC-3	4	X	X	X				X		
UC-4	8	X	X							
UC-5	10	X	X	X				X		
UC-6	4	X		X				X		X
UC-7	5	X	X	X		X			X	X
UC-8	2	X	X		X		X			
UC-9	2	X	X		X					
UC-10	2	X	X							
UC-11	1		X		X		X	X	X	
UC-12	2		X							
UC-13	3		X		X	X	X		X	X
Max PW		38	38	10	3	5	3	10	5	5
Total PW		79	81	23	8	8	48	19	9	12

5.1.5 Domain Model Diagram

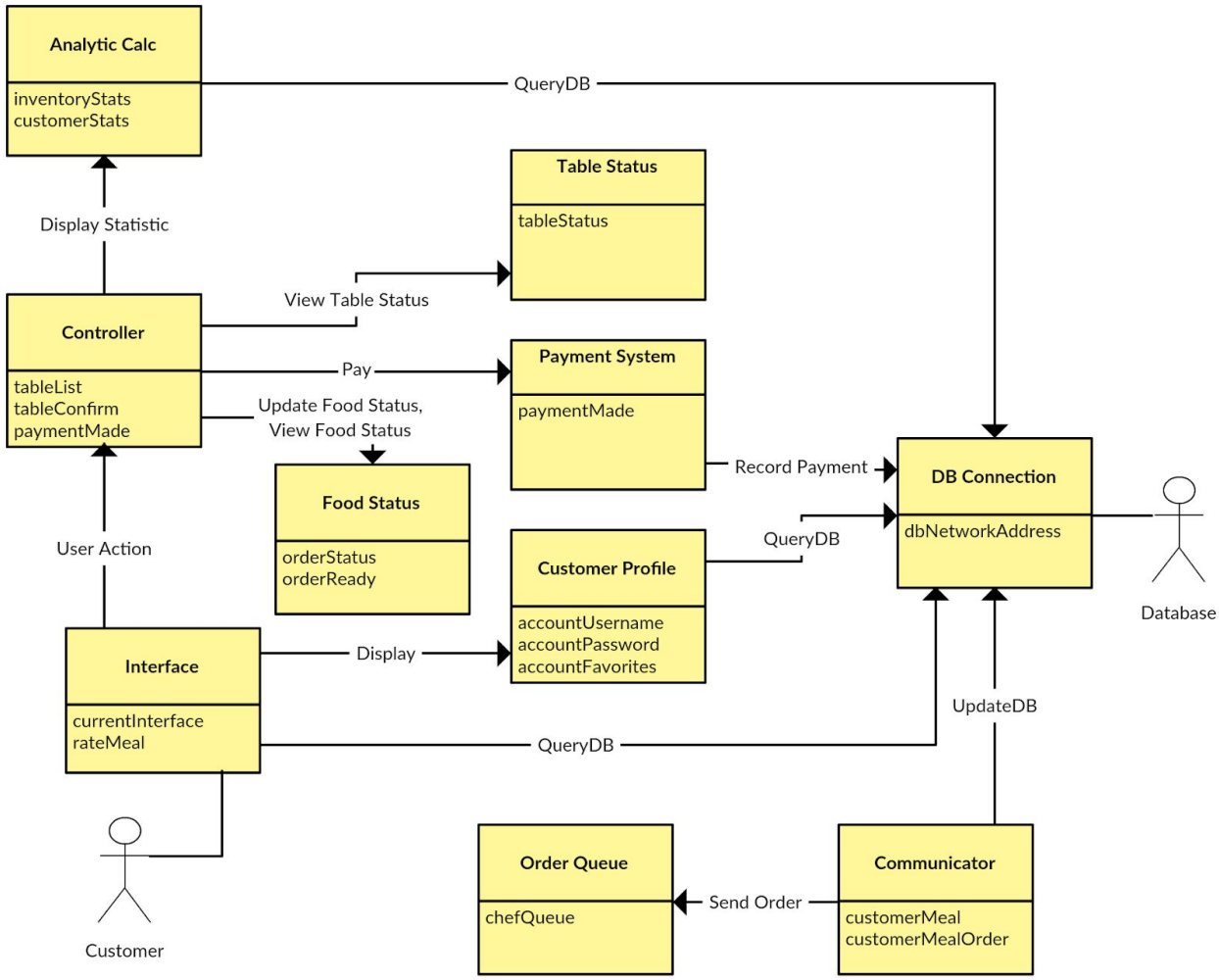


Figure 5.1 Domain Model Diagram

5.2 System Operation Contracts

Name:	Dine in
Responsibilities:	To dine inside the restaurant
Use Case:	UC - 1
Exception:	None
Preconditions:	Customer needs to be in the restaurant physically
Postcondition:	Customer will occupy one of the available table

Name:	Take out
Responsibilities:	To dine outside of the restaurant
Use Case:	UC - 2
Exception:	None
Preconditions:	Customer needs to order the food thru the application
Postcondition:	Food is packed and ready to go

Name:	Reservation
Responsibilities:	To reserve a table
Use Case:	UC - 3
Exception:	None
Preconditions:	None
Postcondition:	A table is reserved for a later date / time

Name:	View Menu
Responsibilities:	Display the menu to the user
Use Case:	UC - 4

Exception:	None
Preconditions:	None
Postcondition:	None

Name:	Table Selection
Responsibilities:	To select a table
Use Case:	UC - 5
Exception:	None
Preconditions:	Table must be available
Postcondition:	Table is marked as reserved in the database

Name:	QR Scan
Responsibilities:	To select an open table
Use Case:	UC -6
Exception:	None
Preconditions:	Table must be available
Postcondition:	Table will be display as occupy

Name:	Payment
Responsibilities:	To pay for the meal
Use Case:	UC -7
Exception:	None
Preconditions:	Order a meal
Postcondition:	Payment is completed

Name:	Rate Food
Responsibilities:	To rate the food

Use Case:	UC -8
Exception:	None
Preconditions:	Food must be ordered by the customer
Postcondition:	Store the rating into database

Name:	Favorite Food
Responsibilities:	To favorite the food
Use Case:	UC -9
Exception:	None
Preconditions:	Food must appear in the menu
Postcondition:	Favorited food will be highlighted

Name:	Login
Responsibilities:	Login
Use Case:	UC -10
Exception:	None
Preconditions:	User must register an account before
Postcondition:	Login to the user's page

Name:	Generate Report
Responsibilities:	Generate report
Use Case:	UC -11
Exception:	None
Preconditions:	Data must be available in the database
Postcondition:	Display the graph and chart

Name:	Register

Responsibilities:	Create an account
Use Case:	UC -12
Exception:	None
Preconditions:	Account has not existed in the database
Postcondition:	Account is registered

Name:	Estimate Time
Responsibilities:	Estimate the remaining cooking time
Use Case:	UC -13
Exception:	None
Preconditions:	Food starts preparing by the chef
Postcondition:	Display the remaining cooking time

5.3 Mathematical Model

Table Designation Algorithm

The algorithm that allows customers to either reserve or immediately sit at a table of their choosing.

Pseudo code:

```
Table selection algorithm: //Modeled in Java format

//Reservation Object Class
public class Reservation{
        private Time checkIn;
        private Time checkOut;

        //Object constructor
        public Reservation(Time checkIn, Time checkOut){
                this.checkIn = checkIn;
                this.checkOut = checkOut;
        }

        //Returns check-in time
        public Time getCheckIn(){
                return checkIn;
        }

        //Returns check-out time
        public Time getCheckOut(){
                return checkOut;
        }
}

//Table Object Class
public class Table{
        private int tableNumber;
        private int currentTableState;//0=Empty , 1=Occupied,
                //2=Reservation made for w/in 60 min, 3=Customer late for reservation, 4=Dirty
        private QRCode qr;
        private ArrayList<Reservation> reservationArray = new ArrayList<Reservation>();

        //Object constructor
        public Table(int tableNumber, QRCode qr){
                this.tableNumber = tableNumber;
                this.qr = qr;
                currentTableState = 0;
        }

        //Returns table number
        public int getTableNumber(){
                return tableNumber;
        }

        //Returns table state
        public int getCurrentTableState(){
                return currentTableState;
```

```
	}

	//Updates the current table state based on the time
	public void tableUpdate(Time currentTime){
		if(reservationArray.size() > 0){
			Reservation rEarliest = reservationArray.get(0);
			Time rInEarliest = rEarliest.getCheckIn();
			Time rOutEarliest = rEarliest.getCheckOut();

			if(currentTableState == 0){
				if(currentTime.isLaterThan(rInEarliest)){
					if(currentTime.isEarlierThan(rOutEarliest))
						currentTableState = 1;
				}
				else{
					Time nextHour = currentTime.addHour();
					if(nextHour.isLaterThan(rInEarliest))
						currentTableState = 2;
				}
			}

			if(currentTableState == 2){
				Time lateBy10Min = rInEarliest.add10Minutes();
				if(currentTime.isLaterThan(lateBy10Min)
					currentTableState = 3;
			}

			if(currentTableState == 3){
				if(currentTime.isLaterThan(rOutEarliest){
					currentTableState = 0;
					reservationArray.remove(0);
				}
			}
		}
	}

	//Returns the QR code of the table
	public QRCode getQRCode(){
		return qr;
	}

	//Sets the value of the table state
	public void setCurrentTableState(int state){
		currentTableState = state;
	}

	//Stores new reservations onto the reservation array
	private void storeReservation(int i, Time in, Time out){
		Reservation newRes = new Reservation(in, out);
		reservationArray.add(i, newRes);
	}

	//Checks to see if desired reservation time is available, and if so calls upon
storeReservation()
	public boolean addReservation(Time in, Time out){
		int s = reservationArray.size();
```

```
        if(s == 0){
                storeReservation(0, in, out);
                return true;
        }

        int i;
        for(i = 0; i < s; i++){
                Reservation rCurrent = reservationArray.get(i);
                Time rInCurrent = rCurrent.getCheckIn();

                Time rOutPrev;
                if(i == 0)
                        rOutPrev = null;
                else{
                        Reservation rPrev = reservationArray.get(i-1);
                        rOutPrev = rPrev.getCheckOut();
                }

                if(rInCurrent.isLaterThan(out){
                        if(i == 0){
                                storeReservation(0, in, out);
                                return true;
                        }
                        else if(rOutPrev.isEarlierThan(in){
                                storeReservation(i, in, out);
                                return true;
                        }
                        else
                                return false;
                }
        }

        Reservation rLatest = reservationArray.get(s-1);
        Time rOutLatest = rLatest.getCheckOut();
        if(rOutLatest.isEarlierThan(in){
                Reservation newRes = new Reservation(in, out);
                reservationArray.add(newRes);
                return true;
        }
        else
                return false;
}

//Removes the inputted reservation from the reservation array
public void cancelReservation(Reservation r){
        for(int i = 0; i < reservationArray.size(); i++){
                Reservation rCurrent = reservationArray.get(i);
                if(r.equals(rCurrent)){
                        reservationArray.remove(i);
                        break;
                }
        }
}

//Removes the earliest reservation from the reservation array
```

```
        public void clearEarliestReservation(){
                reservationArray.remove(0);
        }
}

//Restaurant Object Class
public Class Restaurant{
        private int tableTotal;
        private ArrayList<QRCode> qrCodeArray; //A set of given QR codes
        private ArrayList<Table> tableArray;

        //Restaurant constructor
        private Restaurant(ArrayList<QRCode> qrCodeArray){
                this.qrCodeArray = qrCodeArray;
                tableTotal = qrCodeArray.size();

                for(int i = 0; i < tableTotal; i++){
                        QRCode currentQRCode = qrCodeArray.get(i);
                        Table newTable = new Table(i, currentQRCode);
                        tableArray.add(newTable);
                }
        }

        //Returns the total number of tables in the restaurant
        public int getTableTotal(){
                return tableTotal;
        }

        //Returns the array of table objects within the restaurant object
        public ArrayList<Table> getTableArray(){
                return tableArray;
        }
}

//A class in which runs in the restaurant network
public Class NetworkUpdater{
        private Restaurant restaurant;

        //Continuously runs, checking if the state of any tables need to be updated
        public void tableUpdater(){
                int tableTotal = restaurant.getTableTotal();
                Time currentTime;
                Time prevMinuteTime = Time.getTimeOnComputer();

                while(true){
                        currentTime = Time.getTimeOnComputer();
                        currentTimeMinusMinute = currentTime.subtractMinute();

                        if(prevMinuteTime.isEarlierThan(currentTimeMinusMinute)){
                                for(int i = 0; i < tableTotal; i++){
                                        restaurant.getTableArray().get(i).tableUpdate(currentTime);
                                }
                                prevMinuteTime = currentTime;
                        }

                }
```

```
        }
}

//A class in which customers have access to
public Class CustomerInterface{
        private Restaurant restaurant;
        private int tableNumber;
        private Reservation reservation;
        private boolean firstScan = false;

        //Allows a customer to reserve a desired table at a desired time
        public void reserveTable(Time in, Time out, int tableIndex){
                boolean b = restaurant.getTableArray.get(tableIndex).reserveTable(in, out);

                if(b){
                        System.out.println("Table reserved, see you at "+in.toString()+"!");
                        tableNumber = restaurant.getTableArray.get(tableIndex).getTableNumber();
                        reservation = new Reservation(in, out);
                }
                else
                        System.out.println("Sorry, the desired table is unavailable for this time.");
        }

        //Allows a customer to cancel a previously made reservation
        public void cancelTableReservation(){
                restaurant.getTableArray.get(tableNumber).cancelReservation(reservation);
        }

        //Allows a customer whom enteres the restaurant without a reservation to be seated at a desired
table
        public void getTableNoReservation(int tableIndex){
                int state = restaurant.getTableArray.get(tableIndex).getCurrentTableState();
                if(state == 0){
                        tableNumber = restaurant.getTableArray.get(tableIndex).getTableNumber();
                        System.out.println("Please take your seat!");
                }
                else if(state == 3){
                        boolean noOtherTable = true;
                        for(int i = 0; i < restaurant.getTableArray().size(); i++){
                                if(i != tableIndex){
                                        int s = restaurant.getTableArray.get(i).getCurrentTableState();
                                        if(s == 0){
                                                noOtherTable = false;
                                                break;
                                        }
                                }
                        }

                        if(noOtherTable){
                                tableNumber =
restaurant.getTableArray.get(tableIndex).getTableNumber();
                                System.out.println("Please take your seat!");
                        }
                        else
                                System.out.println("Please select one of the other current tables at
this time,\n...
```

```
                                        ...the person who reserved this table is running late.");
                }
        }

        //Returns an integer depending on whichever QR code was scanned and the state of the table the
QR code resides
        private int qrCodeCheck(QRCode qr){
                for(int i = 0; i < restaurant.getTableArray().size(); i++){
                        Table t = restaurant.getTableArray().get(i);
                        if(qr.equals(t.getQRCode()){
                                if(tableNumber == t.getTableNumber()){
                                        if(t.getCurrentTableState == 0 || t.getCurrentTableState == 2
|| t.getCurrentTableState == 3)
                                                return 0;
                                        else
                                                return 1;
                                }
                                else
                                        return 2;
                        }
                }

                return -1;
        }

        //The method that runs when a user scans a QR code prior to being seated
        public void firstQRScan(){
                QRCode currentQR = QRCode.scan();
                int qrResult = qrCodeCheck(currentQR);

                if(qrResult == 0){
                        restaurant.getTableArray().get(tableNumber).setCurrentTableState(1);
                        firstScan = true;
                        System.out.println("Welcome!");
                }
                else if(qrResult == 1)
                        System.out.println("Sorry, your table is currently unavailable.");
                else if(qrResult == 2)
                        System.out.println("This is table #"+t.getTableNumber().toString()+", not table
#"+tableNumber+".");
                else
                        System.out.println("ERROR");

        }

        //The method that runs when a user scans a QR code when they are ready to leave the restaurant
        public void secondQRScan(){
                QRCode currentQR = QRCode.scan();
                int qrResult = qrCodeCheck(currentQR);

                if(qrResult == 0){
                        restaurant.getTableArray().get(tableNumber).setCurrentTableState(4);
                        restaurant.getTableArray().get(tableNumber).clearEarliestReservation();
                        firstScan = false;
                        System.out.println("Thank you, please come again!");
                }
```

```
                else if(qrResult == 2)
                        System.out.println("This is the table you initially sat at.");
                else
                        System.out.println("ERROR");
        }
}

//A class in which workers have access to
public Class workerInterface(){
        private Reservation restaurant;

        //Allows a worker to update the state of a table after being cleaned
        public void clearTable(int tableIndex){
                if(restaurant.getTableArray().get(tableIndex).getCurrentTableState) == 4){
                        restaurant.getTableArray().get(tableIndex).setCurrentTableState(0);

restaurant.getTableArray().get(tableIndex).tableUpdate(Time.getTimeOnComputer());
                }
        }
}
```

Order Queue Algorithm

As the order comes in, it is enqueued onto a queue. It is dequeued onto the Order Queue screen for the chef to interact with. However, only a set amount of orders are shown on this screen so as not to overwhelm the chef. So a certain amount of orders are dequeued from the queue into a list and more dequeued only after an order is confirmed to be done and taken out from the list.

Pseudo code:

```
Order anOrder= new Order()
orderQueue.enqueue(anOrder)
Int displayed=0
while(!orderQueue.isEmpty()){
       if(count<limit){
              display(orderQueue.dequeue())
              displayed++;
       }
       if(isDone()){  //an order is marked
              displayed--;
       }
}
```

6. Plan of Work

We will continue to split up the work done evenly amongst the group for the future parts of reports. After finalizing report 1, we will allocate more of our weekly time towards coding our actual application and continuing to be progressive by working on making our app as efficient as possible, while solving each of the niche issues that may appear in the restaurant business.

Functional Feature and Description	Start Date	End Date
Menu - an interactive menu that allows customers to view items in greater detail i.e ingredients, estimated cook time, ratings, etc.	9/23/18	10/21/18
Rating System - collecting information from customers on how well the food was prepared via a 1 to 5 star rating.	10/7/18	10/28/18
Seating Chart - have an interactive chart that shows available seats to customers, and “dirty” seats to busboys.	9/23/18	10/14/18
Payment System - allow customers to pay through the app, and give them the ability to add tip and split the bill.	10/28/18	11/11/18
Reservation List - Customers that wish to dine in at a specific time will be able to choose the time, and will be placed on a reservation list.	10/7/18	10/28/18
Food Wait Time - implementing an active system that updates how long the customer has before their food is finished.	10/7/18	10/28/18
Inventory Tracker - a system that allows the manager to view in real time the current inventory for specific items. Alerts when below a certain threshold will also be a feature.	10/28/18	11/11/18
Table Alerts - method of alerting busboys that a table has been recently vacated so that they can clean it up.	10/7/18	10/28/18
Account Interface Coordinator - depending on which account logs in (customer, waiter, or manager), take the user to the correct portal for a seamless user experience	10/21/18	11/18/18

Plan of Work and Ownership:

Outlined below are the teams, and the proposed work plan over the course of the next few weeks.

TEAM	CODE NAME	MEMBERS
Administration/Management	Team α	Stephan, Nikhil
Employees/Waiters	Team β	Jimmy, Michael
Customers	Team ¢	Kyungsuk, Yi

SHORT TERM PLAN OF WORK	TEAM
Create a communication bridge between manager and company material via a server	Team β , Team ¢
Create an interactive customer order menu	Team α , Team β
Implement the available table seating chart	Team ¢
Administration information and data trends	Team α
Customer reservations via data capture and storage to populate a list	All

7. References

[1] Bandarpalle, Sujay, et al. “Report 3 Part 1: Restaurant Automation.” http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf

[2] “Concepts: Requirements.” *Razor Tie Artery Foundation Announce New Joint Venture Recordings | Razor & Tie*, Rovi Corporation, web.archive.org/web/20180402153505/http://www.upedu.org/process/gcncpt/co_req.htm

[3] “Creately - Online Diagram Editor - Try It Free.” *Creately Blog*, creately.com/app/#

[4] “Go: Implement a FIFO Queue.” *Go: Implement a FIFO Queue | Programming.Guide*, programming.guide/go/implement-fifo-queue.html.

[5] Marsic, Ivan. Software Engineering. New Brunswick: Ivan Marsic, 2012. Print.

[6] “QR Code Features | QR Code.com.” *Razor Tie Artery Foundation Announce New Joint Venture Recordings | Razor & Tie*, Rovi Corporation, web.archive.org/web/20130129064920/http://www.qrcode.com/en/qrfeature.html

[7] “UML 2 Sequence Diagrams: An Agile Introduction.” *UML 2 Sequence Diagrams: An Agile Introduction*, www.agilemodeling.com/artifacts/sequenceDiagram.htm.