

Robo Ramsay

Software Engineering

Team 6

5/2/2021

Github repository:

https://github.com/agc234/roboramsey_ui

Project Page:

https://dpmon1.github.io/Robo_Ramsey

Akira Brown, Esteban Salazar, Alan Chacko, Suryansh Singh, Adrian Mah, Parth Vora, Shrey Joshi, Raghav Gopalakrishnan, Kunj Desai, Ray Chau

Individual Contributions

| Topic | Akira | Esteban | Alan | Suryansh | Adrian | Parth | Shrey | Raghav | Kunj | Ray |
|---|-------|---------|------|----------|--------|-------|-------|--------|------|-----|
| Summary of Changes | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Customer Statement of Requirements | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| System Requirements | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Functional Requirements | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Effort Estimation using Use Case Points | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Domain Analysis | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Interaction Diagrams | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Class Diagram and Interface Specification | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| System Architecture and System Design | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Algorithms and Data Structures | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| User Interface Design and Implementation | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| Design of Tests | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |
| History of | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Work, Current Status, and Future Work | | | | | | | | | | |
| References | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% | 10% |

Summary of Changes

1. Changed older database PostgreSQL to Firebase. Updated the Stripe section. Added in an Expo section.
2. Added multiple terms and definitions to the glossary. Added images from our app for extra readability.
3. Updated many requirements to be more in line with current application style. New additions include 1, 2, 3, 5, 6, 14, 16, 30, 33, 34, 35, 37, 38.
4. Added a few more use cases to account for the new requirements and features.
5. Changed requirements for UC-1 to 12 in Effort Estimate. Changed UC-1 to 12 in Effort Estimate. Added UC 13 to 21 in Effort Estimate.
6.
 - 1) Updated Responsibilities, adding more. Remapped use cases to association definitions in the domain model.
 - 2) Changed persistent storage model from 4 relational SQL db to a non-relational noSQL db.
7. Redone for UC-1, 2, 10, 18, 19
8. Added key variable names to our class diagram and data types. Removed menu item as a domain concept from traceability matrix. Added multiple Design Patterns
9.
 - 1) There are now 2 main subsystems instead of 6
 - Menu, Accounts, Alerts, Orders, Inventory, and Statistics have been removed
 - We are now using 2 subsystems, Main and Accounts
 - 2) We have changed our architecture style
 - We have gone from a microservice architecture to a monolithic architecture
 - 3) We have changed how we map our subsystems to hardware. Now, much of it is in the cloud, meaning the hardware used is easily changeable.
 - 4) We have simply mentioned the network protocols Firebase uses. Previously the plan was to create a personally hosted HTTP server, but then we decided to switch to Firebase and use their API to contact their server. The requests are still HTTP requests.
10.
 - 1) Added the algorithm on how to calculate meal preparation time as well as the data structures needed in 10.2 order to implement this new algorithm.
 - 2) Updated database type from SQL to NoSQL, added Stack for the Dish allocation
11. User Interface Design and Implementation adjusted to match current code
12.
 - 1) UC-13, UC-10, UC-11, UC-7, and UC-3 have all been removed for the final demo

- 2) Updated TC-5, TC-6, TC-7, TC-8, and TC-9 from Section 12.1 and likewise updated Section 12.2 to reflect those changes
- 13. New Section Entirely
- 14. Added a new reference for firebase

Table of Contents

| | |
|---|-----------|
| Individual Contributions | 1 |
| Summary of Changes | 3 |
| Table of Contents | 5 |
| Section 1: Customer Problem Statement | 7 |
| 1.1: Problem Statements | 9 |
| 1.2: Decomposition into Sub-problems | 12 |
| Section 2: Glossary of Terms | 17 |
| Section 3: System Requirements | 20 |
| 3.1: Business Goals | 20 |
| 3.2: Enumerated Functional Requirements | 21 |
| 3.3: Enumerated Nonfunctional Requirements | 23 |
| 3.4: User Interface Requirements | 25 |
| Section 4: Functional Requirements Specification | 37 |
| 4.1: Stakeholders | 37 |
| 4.2: Actors and Goals | 37 |
| 4.3: Use Cases | 38 |
| i. Casual Description | 38 |
| ii. Use Case Diagram | 40 |
| iii. Traceability Matrix | 41 |
| iv. Fully-Dressed Description | 44 |
| 4.4: System Sequence Diagrams | 49 |
| Section 5: Effort Estimation using Use Case Points | 54 |
| Section 6: Domain Analysis | 62 |
| 6.1: Domain Model | 62 |
| Conceptual Model Diagram: | 62 |
| i. Concept Definitions | 63 |
| ii. Association Definitions: | 65 |
| iii. Attribute Definitions: | 66 |
| iv. Traceability Matrix: | 67 |
| 6.2: System Operation Contracts | 69 |
| 6.3: Data Model and Persistent Data Storage | 71 |
| 6.4: Mathematical Model | 72 |
| Section 7: Interaction Diagrams | 73 |
| Section 8: Class Diagram and Interface Specification | 80 |

| | |
|---|------------|
| 8.1: Class Diagram - see attached pdf for clearer image | 80 |
| 8.2: Data Types and Operation Signatures | 81 |
| 8.3: Traceability Matrix | 84 |
| 8.4: Design Patterns | 86 |
| 8.5: Object Constraint Language (OCL) Contracts | 87 |
| Section 9: System Architecture and System Design | 88 |
| 9.1: Identifying Subsystems | 88 |
| 9.2: Architecture Styles | 89 |
| 9.3: Mapping Subsystems to Hardware | 89 |
| 9.4: Connectors and Network Protocols | 89 |
| 9.5: Global Control Flow | 89 |
| i. Execution orderliness | 89 |
| ii. Time dependency | 90 |
| 9.6: Hardware Requirements | 90 |
| Section 10: Algorithms and Data Structures | 91 |
| 10.1: Algorithms | 91 |
| 10.2: Data Structures | 92 |
| Section 11: User Interface Design and Implementation | 93 |
| Section 12: Design of Tests | 95 |
| 12.1: Design of Tests | 95 |
| 12.2: Test Coverage | 96 |
| 12.3: Integration Testing | 97 |
| Section 13: History of Work, Current Status, and Future Work | 98 |
| History of Work | 98 |
| Current Status | 100 |
| Future Work | 100 |
| Section 14: References | 101 |

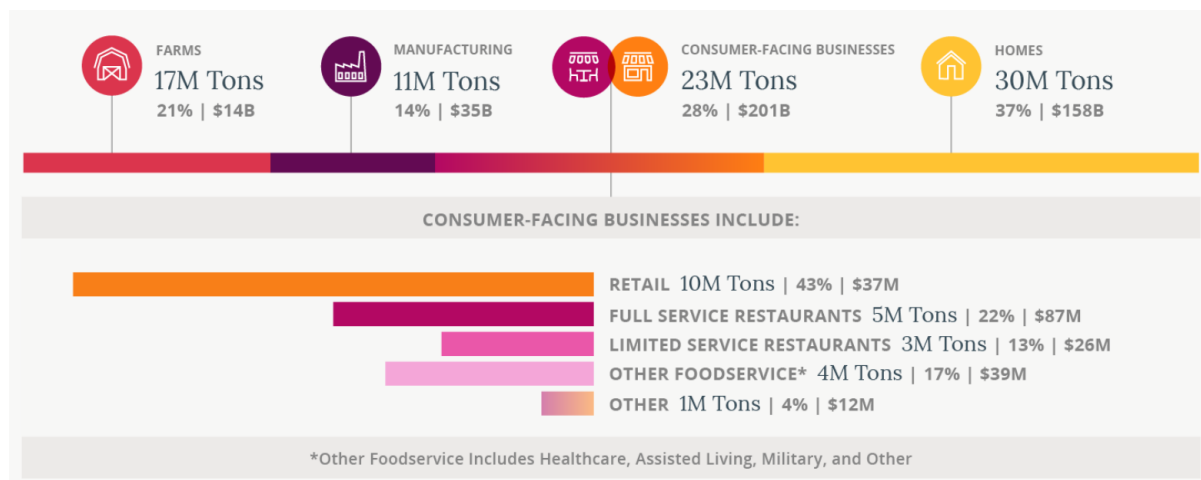
Section 1: Customer Problem Statement

Problem Diagnosis

Most restaurants have leftover food by the day's end, and sometimes there are menu items that have not been ordered even once throughout the day. The Green Restaurant Association found that the average restaurant may produce up to 100,000 pounds of food waste in a year. Ingredients that could have been consumed are instead disposed of because they expire. This results in most of the food waste ending up in landfills, where it rots or ferments into greenhouse gases. Additionally, this wastage also contributes to the proliferation of chemicals or undesirable bacteria that are a danger to public health and the environment if left unchecked. The food waste also represents lost revenue. Money that could have been spent to make revenue-generating dishes is instead spent on unused ingredients for less popular dishes. It is even possible that state regulations issue fines for food waste violations.

Furthermore, items that are less popular are often kept on a restaurant's menu, resulting in funds being wasted on buying ingredients for these items. These funds could have been instead allocated to ingredients for more popular items. An application that could automatically identify these less popular items would be desirable.

Another problem: There tends to be a disconnect between customers and restaurants on what dishes are available, especially when dining in. A method of being able to convey this information almost instantaneously would help streamline the link between customer, restaurant, and menu item so that the customer will not get their hopes up on ordering something that is unavailable.



Source: *Food waste challenge*. (n.d.). Food Waste Recycling Analysis, Reduce Food Waste & Food Recovery - ReFED.

<https://refed.com/food-waste/the-challenge/?sort=economic-value-per-ton>

All these issues could be avoided if there was proper analysis of the restaurant's clientele and the market for ingredients. This can be easily done by a computer with simple sorting algorithms and the use of a database, and then graphed to help the manager make better informed choices.

1.1: Problem Statements

Restaurant Customer Statement

Way back before COVID dining restrictions were put in place, I loved going out to eat at different restaurants with my friends. It was usually an enjoyable time, but I found that I often did not know what their recommendations were, especially if they do not signal any popular dishes on the menu. Due to that, I often take too long looking at every section of the menu, and my friends often tease me about it but I just never know what to get! This only gets worse when the restaurant has such a large menu that is separated by multiple pages with little to no pictures. This can lead me to ordering something and then not liking how it tastes, which makes me feel bad because growing up I was told to not waste food, and even now I don't like wasting food because it's my wallet that suffers! When I eventually order what I want, the waiter sometimes comes back telling me that due to specific ingredients having run out, the specific dish I wanted can not be made. This becomes a big hassle as the waiter offers me something to replace it that would not be the same or if I wanted to order something different. What's worse is if some of my friends and I order the same dish, we can get greeted by the waiter by them mentioning how there is not enough ingredients to fulfill our order, which some of my friends would go "How do you not know that there isn't enough of [the dish's name or ingredients to make it]?!". This makes the dining experience less enjoyable because the information is not readily available right away upon ordering, because at least we'd be able to quickly order something different and have all the food come at roughly the same time, and not have one or more people wait even longer.

I also live in a part of the United States where COVID has been having declining numbers of cases for a while, so my friends and I have been going out to restaurants occasionally, but following guidelines of using masks. However, I am wary of going to restaurants now because of any potential exposure through the waiter coming to our table even when they have a mask on, since I have family members that can be at risk. I feel like if anything were to help with any of these issues I am experiencing, it would be a virtual or online version of a restaurant's menu that lets me order food to my table when dining in. Not only would it solve my woes of potentially getting COVID and spreading it to family members, but it can probably do so much more! For example, I tend to have trouble ordering something so maybe if that same menu can list recommendations or include images of the actual dishes that can be made, it would save me some trouble and time. Also, with a menu being through an electronic medium, it could also let my friends or I know what can't be made due to the restaurant running out of something instantly.

Restaurant Chef Statement

I enjoyed working with my team in the kitchen, it was lively, with a bit of casual banter. As a team of cooks, good communication was of the utmost importance. We would communicate what orders we were working on as well as keeping check of stock so that we could notify the

manager later on so that they can order more, it's never a good thing to run out of ingredients during a busy day. Ever since COVID struck, our restaurant restricted the amount of kitchen staff allowed in the building at any given time. This generally made work go slightly slower since more people makes prep work and cooking easier especially if there are multiple stations to be monitored. Due to the restrictions, there are times I couldn't be in the kitchen and as a result, some of my fellow yet less experienced cooks would either underestimate or overestimate the current stock of ingredients. This leads to running out of ingredients while we still have to put out orders, or ending up with too much of an ingredient that we have come in fresh. If we keep a very high amount of fresh ingredients in the fridge and constantly put in more stock of it, eventually we could have the whole bunch of fruits or vegetables wilting or rotting. Leaving perfectly edible ingredients in the fridge or freezer until it goes bad wastes the restaurant's resources and storage space; efficiency and freshness are the virtues of a chef, not being able to have both sucks. It's a big pain when in the middle of work, we receive an order for a dish that we later find out that we don't have enough ingredients to fulfill the recipe's measurements. Even worse is the dilemma of "Do we try to finish it with what we have and have the waiter bring it out? Or do we call in the waiter so that they share the news with the customers?". Both situations can end up badly especially if the customers in question are not in the mood to be civil. There have been times where they demand a free dessert or have that specific dish free on the check, and that never ends well with the manager.

A way to potentially fix the issue of inventory management would be to have a virtual or electronic list of it, an ingredient tracker one could call it. This would greatly improve efficiency between different shifts of work, and solves the potential issue of less experienced cooks miscounting or misjudging the amount of ingredients left. Not only will it help with inventory management, the process of the manager buying ingredients to maintain stock will be easier. Not only that, but we could have another part of the system that works with the ingredient tracker to feature dishes and the measurements required to make them, so that stock could be updated as we are making the dishes that are being ordered.

Restaurant Manager Statement

As a manager working in a restaurant setting, the roles and responsibilities that I handle include purchasing items, keeping inventory, and making sure all components of the restaurant are functioning efficiently. Sometimes I have cooks telling me that we need more of an ingredient, but too often giving a higher estimate than what would be needed, which leads to wasting funds as well as wasting perfectly good ingredients. Using an application that would tell me when the kitchen is running low on certain items and when they need to be ordered, I would be able to ensure that the customers are given the restaurant experience they should be expecting while also keeping the kitchen in working order. I would also be able to track usage of each ingredient in each dish to maximize product output with less waste. This helps me make necessary cuts to certain dishes and re-evaluate which dishes might be a better fit for the restaurant.

The biggest issue a customer runs into at a restaurant is the wait times. Having an application track the time an order is placed, the time it takes for a chef to prepare the dish, and the time it takes for the wait staff to deliver the dish, customers will be able to have a very accurate picture of how long it should take for their food to arrive. I can also pinpoint where the staff is working the most efficiently and where they might be able to improve. For example, if I see that the same dish takes 20 minutes longer to prepare during the day vs. night, I am able to see whether it might be an issue of staffing, lack of prepared ingredients, etc. This ensures that I am tending to the staff's needs, so that in turn, they can tend to the customers' needs.

Restaurant Waiter Statement

The biggest responsibility I have as a waiter is to provide excellent service to the customers by keeping them happy and satisfied. My job description includes taking customer orders, delivering food and beverages, and making menu recommendations.

As I take customer orders, I run into problems with stock and inventory all the time, and it's a terrible situation to be in. The customer orders an item that they see on the menu. I go into the kitchen to convey the message. The kitchen tells me that they've run out of stock to make the item. I, as the messenger, have to take that news and explain it to the customer as best I can. Hearing something like that wouldn't make anyone happy, so as a waiter I have already failed in one of my biggest responsibilities: keeping customers happy and satisfied. I'm put in an embarrassing situation, the customer is disappointed, and the restaurant risks losing business. The smart menu fixes this problem by "greying out" (making unavailable for selection) items on the menu that do not have enough ingredients for an order to be filled. As a waiter, I know that I won't ever have to be put in a situation where a customer might order an item that isn't available at that moment, which helps me do my job better and keeps the customer happy.

The restaurant welcomes in new customers almost on a daily basis, most of which have very little idea on what's hot on the menu. Popular menu items can consistently change, and it's often difficult for me to keep track of what most customers are ordering, especially when there are multiple waiters. Beginner waiters at the restaurant also have very little experience, and may not be able to answer customer questions and give them perfect recommendations. Noting down how many orders of a menu item was done each day with pen and paper can get messy and difficult to handle for both the waiters and the manager. A smart menu system where the menu item orders are easily counted and automatically arranged according to the popularity of the dish easily fixes all of these issues. Communication between all of the waiters and the manager in this regard doesn't need to be a major priority anymore, saving valuable time for the restaurant. Novice waiters don't need to be embarrassed when asked for recommendations anymore. Overall, the restaurant is put in a much better position.

1.2: Decomposition into Sub-problems

Based on our problem statement, the two overarching problems we are trying to solve are decreasing food waste and increasing the efficiency of restaurants. These two problems can be further decomposed into sub-problems.

Food Waste

The first problem we are trying to solve is decreasing the amount of food/ingredients that is wasted by restaurants. This problem can further be divided into subproblems, each of which is a potential source of food waste. The first subproblem relates to the fact that dishes that are unpopular take up valuable space in the ingredients storage. These ingredients are wasted and must be resupplied in case the customer orders the dish, sometimes perpetuating the wasting of ingredients. Additionally, the second subproblem is that without knowledge of the inventory, ingredients are also wasted from popular dishes when management is unaware of the exact amount of ingredients currently in possession. These two subproblems will be directly addressed by features in our application.

Increasing Efficiency

In addition to decreasing food waste, the other problem that we are trying to solve is increasing the efficiency of the restaurant. Increasing efficiency can be broken down into several subproblems, each of which will be addressed by a particular feature in our application. The first of these subproblems is the fact that currently, customers at a restaurant often do not know when a dish is not available due to lack of ingredients unless explicitly told so by a waiter. This can result in longer wait times for customers. Another subproblem is increasing the speed of ordering. The average wait time in a restaurant is approximately 23 minutes (Full Service Restaurant News, 2013). Reducing this time would be desirable. The third subproblem relating to efficiency is reducing the time required to address ingredient shortages in the restaurant pantry. The fourth subproblem relating to efficiency is allowing the manager to easily have a bird's eye view of all orders in the restaurant to allow them to easily detect any bottlenecks in service. Each of these subproblems will be addressed by a particular feature or functionality in our application.

The Solution

In order to fix the issues revolving around the availability of a certain menu item, we have proposed to introduce a Smart Menu which will end up helping both the restaurant and the customer. The Smart Menu will, like a normal menu, keep note of the daily specials as well as highlight new dishes. Along with this, it will also keep track of the restaurant's ingredient stocks. The Smart Menu will also keep track of the popularity of all the dishes, using it to provide customer recommendations and rearrange the menu to be relevant. Another function will be to provide the manager with insight on long term trends. This will help the restaurant greatly in

figuring out how many of each ingredient they should order, and hence reduce the unnecessary wastage of food. This will save the restaurant a good deal of money as well as make them more efficient.

Such an automated version of the restaurant process will benefit the customers, employees and the management at the same time. The inventory would provide the restaurant staff with live usage data. This will allow the restaurant management to more accurately order quantities and lessen the amount of food wasted at the end of a business period. In addition, the tracking of inventory using an automated software - as opposed to manually keeping inventory - not only helps lessen the employees' workload, but also maintains the data virtually. This allows for our application to use it to predict trends. This way it can better curate menu specials and generally refine the menu. The refined and updated menu will in turn benefit the customer as they will be offered more relevant food that's guaranteed to be in stock.

We can measure these changes using concise metrics based on data gathered from our software. The economic impact of our application can be measured by calculating the difference between the value of the amount of food thrown out prior to our automation and the value of the amount tossed afterwards. In addition, the time saved by the employees who would manually check inventory and track usage at the end of the day could be used elsewhere. We could measure how relevant the menu is by finding the distribution of different plates ordered. Menus with a wider distribution leave dishes at the end that receive nearly no orders indicating it's a poor menu choice. On the other hand, menus with a more even distribution - where all plates get at least some traction - would indicate a more relevant menu. The change in that distribution would be our menu relevance metric, which a business can use to decide whether to replace a dish or set of dishes.

Suppose a customer ordered a meal that requires a particular ingredient and is not currently in the restaurant's inventory. In a restaurant without a Smart Menu, the customer would order the meal and be told several minutes later that the ingredient is not available. The customer would then have to order something else or be offered a replacement ingredient. These minor annoyances could negatively affect the customer's opinion on the restaurant. On the contrary, a restaurant with a Smart Menu would keep a live inventory that provides updates on the availability of these ingredients. Management would be notified about the shortages of certain ingredients well ahead of time and potentially never have customers or employees experience shortages.

Suppose that in another situation, a new customer is unsure of what to order on their first visit. The Smart Menu would provide the customer with an updated list of popular items as well as the availability of items to customers and employees alike. Incoming orders would result in updates to the availability of ingredients, which would in turn result in updates to the smart menu.

So, we can build an app that will incorporate all these individual solutions. It can be a convenient tool to gather, analyse and understand the relevant data points, as well as provide an interactive menu. It will keep track of ingredients, and thus assist with minimizing waste. Large

restaurant chains already have apps which are used to order food, review orders, and collect data. Our application will allow smaller restaurants and restaurant chains to easily achieve similar granular functionality, along with better utility towards the restaurant staff.

Plan of Work

The app will consist of four main business functionalities. (Note: some of these functionalities are groupings of highly related functionalities.)

1. Ordering: The customer will have the ability to select and order food from the menu, pay for it, and receive a receipt. Upon ordering, they will be able to view the status of each dish (not prepared, being prepared, prepared) in their order and will receive a notification when dish status changes. This feature can be used alone.
2. Dish analysis: The manager will be able to view the popularity of each dish. Popularity is measured as the amount of times a dish is ordered within a particular time interval that can be selected by the user - week, month, 3 months, or year. The customer will receive recommendations based on which dishes are the most popular. This feature cannot be used alone and has to work in combination with 1.
3. Ingredient tracking: The manager will be able to input the quantities of each ingredient available into an ingredient management system. This system will update the amount of each ingredient available based on food cooked throughout the day. This will be done by communicating with the ordering and smart menu features. The manager will also be able to add new ingredients, search for ingredients based on keywords, and filter the ingredients list based on availability and type. The manager will also be notified when there is not enough of a particular ingredient to cook a dish. This feature can be used alone.
4. Smart Menu: The menu of the restaurant will be a “smart menu” that updates according to ingredient availability. Items that do not have enough ingredients to make at least 6 of them will be “greyed out” (unavailable for selection) on the menu. The menu will also include sort, search, and recommendation features to allow customers to quickly select a dish to order. This feature cannot be used alone and has to work in combination with 3.

The following sub-teams will develop each functionality:

1. Ordering process - (Esteban, Akira)
2. Dish analysis - (Alan, Suryansh)
3. Ingredient tracking and search - (Adrian, Parth, Shrey)
4. Smart menu - (Raghav, Kunj, Ray)

In order to implement the project, the following features will be reused from the TurboYums app produced by a previous team:

1. Customer Interface - We will reuse the order, payment, and receipt functionalities for the customer. To this we will be adding an order status feature that allows the customer to view and be notified of changes in the status of each dish. We will also add a “smart menu” functionality that “grays out” dishes that are not available and recommends dishes based on popularity.
2. Manager and Waiter Interfaces - We will reuse the order-queue feature that allows the manager and waiters to view incoming orders as a queue. For the manager, we will reuse the menu editing (add, remove, and edit item) feature. To this we will add an ingredient tracking system that lets the manager keep track of and search for ingredients and updates based on food cooked throughout the day. The manager will also be able to view the popularity of each dish and be notified when there are not enough ingredients to produce a certain dish.
3. Chef Interface - We will reuse the order-queue feature for the chef. To this we will add the ability for the chef to update the status of each item in the queue of incoming orders as they cook. The chef will also be able to view and edit the ingredient-tracking system.

The project itself will be software-only. We won't need any specialised data or specific hardware. We will only need some computer that meets the minimum requirements (suitable processing power, RAM/ROM, OS and a network connection to user devices) to run the server application, and each user will need a suitable device as well for the user application.

We will be using JavaScript mainly for the application. We will be using a few libraries and frameworks in this application:

1. **Firebase:** It is a Backend-as-a-System (BaaS) managed by Google Cloud Platform. That means that we only have to work on the frontend for the most part; Firebase can handle things like user authentication (like login, register, logout), data storage (As a NoSQL database - we can feed in data of any shape and it can store it), hosting and deployment. To access Firebase services, we use their API.
2. **Express:** This is a framework for javascript that was built specifically to facilitate development of applications that use Node.js. It is used to create and manage servers. Using this, we will make code that allows us to host a server on the machine that runs the application.
3. **React Native:** This is a cross-platform mobile application framework. It incorporates the React.js library. It is built to allow supported platforms (mobile phones - both iOS and Android, tablets, desktops, etc.) to render the same React code easily. Using this, we will code an application that resides on the user's device, which the user can use to interact

with the server. It allows us to separately create the pages and the components inside the pages.

4. Expo: This is an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React. More specifically, it is a third party library and compiler. It can convert any code written with JavaScript and React Native into software that works on Android, iOS, and the web.
5. Node.js: This is a JavaScript Runtime Environment. This means that Node.js is software that Javascript can use to run. This is common in JavaScript applications. Using this, we will run our code.
6. Stripe API: Given enough time, we may use this API to process customer payments. This is one of many vendor APIs that a given business can use to accept payments. This is not explicitly implemented as various businesses may already have purchasing systems in place.

Team Description and Strengths:

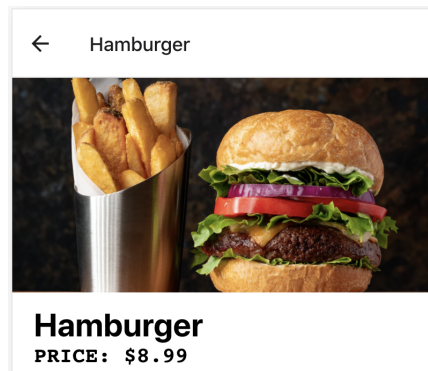
1. Akira Brown - JS
2. Esteban Salazar - JS
3. Alan Chacko - JS, Node.js/Express, React Native, SQL
4. Suryansh Singh - JS, SQL
5. Adrian Mah - JS
6. Parth Vora - JS
7. Shrey Joshi - JS, React Native, AWS, SQL
8. Raghav Gopalakrishnan - React Native, AWS, JS,
9. Kunj Desai - JS
10. Ray Chau - JS

Section 2: Glossary of Terms

Menu - A list of food available to be prepared, cooked, and presented, including pictures and ratings usually in the form of a hand-held paper

Smart Menu - Menu available on phone/website that automatically “greys out” dishes that are unavailable for selection by communicating with the ingredient tracker.

Dish - An item on the smart menu.



Grey Out - To make an item unavailable for selection on the Smart Menu

Dish Description - Any food or substance, excluding water, that is combined to make a particular dish

Description:

Hamburger with lettuce, tomato, and mayo

Chef - Makes food that is requested by the customers.

Customer - Someone that comes to the restaurant as a guest to be waited on and served food.

Manager - Supervises restaurant to make sure everything is going smoothly. The person who is ultimately responsible for both the customer experience and the restaurant inventory.

Waiter - Delivers food to customers. Can take orders if the customer does not have a phone.

Order Viewer Screen - Where customers can see the properties of their order. Allows for deletion of dishes, shows ETA, and the total of the order.

←

OrderViewer

ETA: 57 minutes

Subtotal: \$48.96

Total: \$52.39

| ORDERED | | |
|-------------|---------|-------|
| Item | Price | Quant |
| Chili Mu... | \$12.99 | 1 |
| Spaghatt... | \$11.99 | 3 |

Dish status - The status of each dish within an order (not prepared, being prepared, prepared).

PENDING

Order queue - A feature that displays a queue that shows the orders and corresponding order status of all customers in a restaurant.

Ingredient tracker - A feature that lets the manager/chef enter the amount of ingredients required for a particular recipe, update the quantity of each ingredient available, and search for ingredients. This tracker updates the amount of ingredients available throughout the day as food is ordered.

Availability- Whether or not a dish can be made depending on the ingredients in stock.

Dish popularity - The number of times a dish is ordered within a particular time interval (week, month, three months, or year).

Menu Editor - A feature that allows the user to add, remove, or edit items on the menu. Also displays the popularity of each dish.

Guest Account - An account where a customer can place an order without having to register their name, email, and password. Each device has a unique user ID so that the restaurant knows where to serve the order.

LOGIN

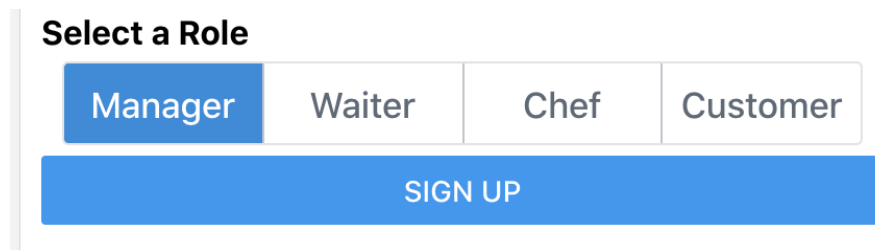
REGISTER

GUEST LOGIN

Manager Account - An account that has access to the order queue, ingredient tracker, and menu editor.

Chef Account - An account that has access to order queue and can change the status of each dish within an order and ingredient tracker.

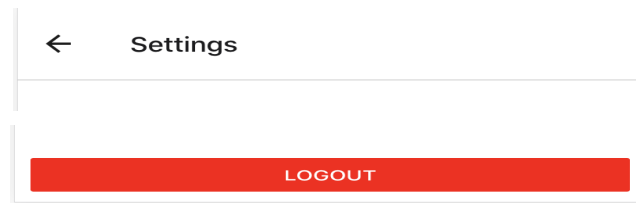
Customer/Waiter Account - An account that can order food, pay for the order, and view the status of each dish within an order.



The interface shows a vertical grey bar on the left. To its right, the text "Select a Role" is displayed. Below this text are four buttons: "Manager" (highlighted in blue), "Waiter", "Chef", and "Customer". Below these buttons is a wide blue button labeled "SIGN UP".

Restaurant Automation - Process of automating restaurant tasks and functionalities that would otherwise require employees to do.

Settings - A page where users can adjust account properties and design, currently also allows for a user to log out of their account.

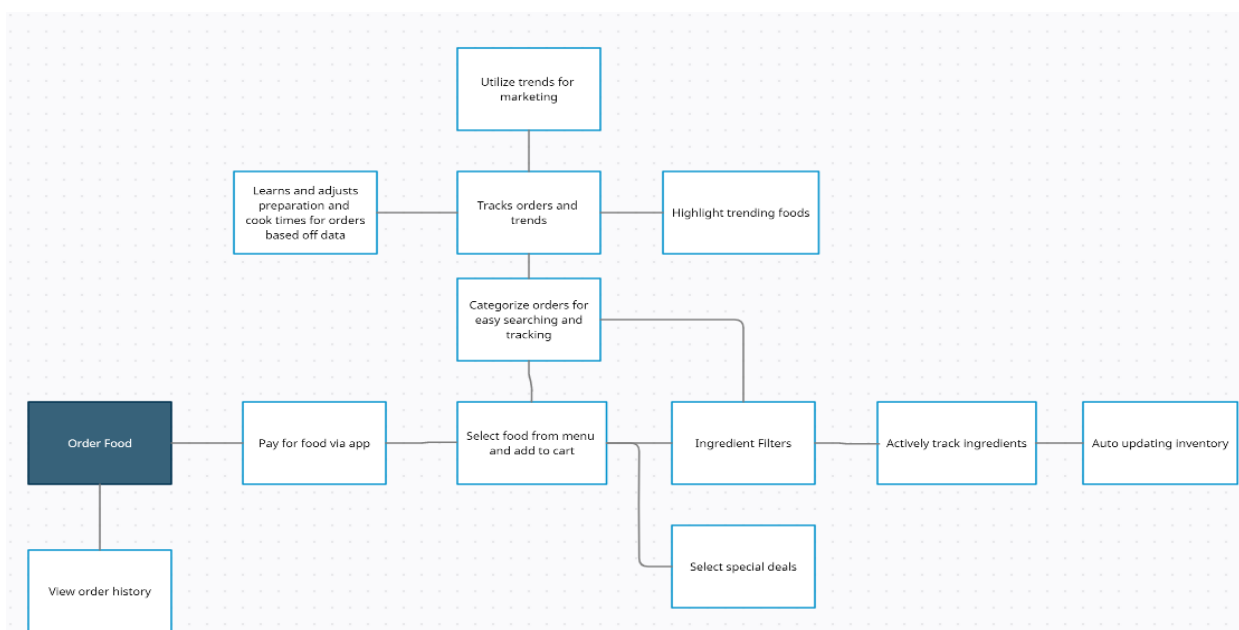


The settings page layout includes a header with a left-pointing arrow and the word "Settings". Below the header is a horizontal line. At the bottom of the page is a red button labeled "LOGOUT".

Section 3: System Requirements

3.1: Business Goals

In order to successfully implement our proposed system to automate restaurants the team would have to first create a “smart menu” which would link the automated process from the consumer all the way to management. This smart menu would, at the least, allow patrons to browse, order, and pay from the smart menu system. From there, the team gets the goal of giving the smart menu the ability to maintain a database of the restaurant’s inventory. This database would then need to be able to be edited by manual input or changed automatically based on orders placed so that we can have the information to keep inventory. We also want to be able to keep track of the estimated quantity of ingredients used as well as how much of a dish is used in an order. With the database of ingredients patrons should also be able to filter the menu by ingredient in various ways such as “preferred” or “allergic”. After the implementation of the database has been successfully completed, the smart menu can begin tracking trends based on orders to find what people like to order, when they order it, how they want it cooked, and how many ingredients are used by the restaurant in one period of time. With this data tracking, the smart menu can then be used as a marketing tool which can boost specials on the menu, help target ads, and provide a “popular foods” page to help new customers order with ease. Once an order is placed, our database will be able to provide the user with the estimated time until arrival based on time delivered of previously similar orders. We also plan to make our application more versatile for different restaurants and their customers, so customers can either have one person order for the table, or we would develop a feature where customers of a single party can send their items to a single person of their party to place the order. We believe customers would appreciate this feature since it does not require verbal communication, so they can continue to socialize while deciding on what to order. The customer can also choose to have the items arrive as soon as they are done or to wait until the entire order is finished before the waiter sends it to the table. Tree diagram:



3.2: Enumerated Functional Requirements

Note: Priority 5 is highest, 1 is lowest.

A. Ordering

| Identifier | Priority | Requirement | Status |
|------------|----------|--|----------|
| REQ-1 | 4 | The application's customer interface will have a way to collect and edit a list of items and quantities, and then place the list as an order. | Complete |
| REQ-2 | 4 | The application's customer interface will have a screen to show the collected list of items and quantities. | Complete |
| REQ-3 | 4 | The application's customer interface will have a screen to list all their current and past orders and their statuses. | Future |
| REQ-4 | 4 | The application's customer interface will have a screen to display more details about the current order and the statuses of all the individual items in the order. | Future |
| REQ-5 | 2 | The application's customer interface will have a way to edit items in the current order, but only if the items are not already cooking. | Future |
| REQ-6 | 3 | The application's customer interface will have a screen to pay for the order. | Future |
| REQ-7 | 3 | The application's waiter interface will have a screen that lists all orders. | Future |
| REQ-8 | 3 | The application's waiter interface will have a screen that displays more information regarding orders and allows editing orders. | Future |
| REQ-9 | 3 | The application's waiter interface should allow them to order on behalf of customers | Future |
| REQ-10 | 4 | The application's manager interface will have a screen that lists all orders. | Future |
| REQ-11 | 4 | The application's manager interface will have a screen that displays more information regarding orders and allows editing orders. | Future |

| | | | |
|--------|---|---|----------|
| REQ-12 | 4 | The application's chef interface will have a screen that lists all orders. | Future |
| REQ-13 | 4 | The application's chef interface will have a screen that displays more information regarding orders and allows editing orders and order status. | Future |
| REQ-14 | 1 | Individual items in an order should have statuses. | Complete |

B. Smart Menu (Note: These are functionalities that extend the menu beyond simply displaying and allowing selection of items, i.e. a dumb menu)

| Identifier | Priority | Requirement | Status |
|------------|----------|---|----------|
| REQ-15 | 5 | The application's customer interface will have a screen that lists available dishes and has options to sort, search and filter. | Complete |
| REQ-16 | 5 | The application's customer interface will have a screen that displays more information regarding dishes. | Complete |
| REQ-17 | 4 | The application's customer interface will not allow customers to select items that are not available or possible to make. | Future |
| REQ-18 | 3 | The application's customer interface should have a category for popular dishes in the menu. (as measured by order frequency) | Future |
| REQ-19 | 2 | The application's customer interface should be able to request a waiter. | Future |
| REQ-20 | 4 | The application's waiter interface should display all the screens of a customer's menu. | Complete |
| REQ-21 | 4 | The application's chef interface should display all the screens of a customer's menu. | Complete |
| REQ-22 | 4 | The application's manager interface should display all the screens of a customer's menu. | Complete |
| REQ-23 | 5 | The application's manager interface should allow them to edit the digital menu. | Future |
| REQ-24 | 2 | The application's manager interface should display data on popularity and trends. | Future |

C. Ingredient Tracking (Inventory)

| Identifier | Priority | Requirement | Status |
|------------|----------|--|--------|
| REQ-25 | 4 | The application's manager interface will display an alert if it's not possible to make a dish due to insufficient ingredients. | Future |
| REQ-26 | 3 | The application's manager interface will display an alert if a kitchen ingredient is below a certain threshold. | Future |
| REQ-27 | 4 | The application's manager interface should allow them to view and edit the ingredient count manually. | Future |
| REQ-28 | 4 | The application's chef interface should allow them to view and edit the ingredient count manually. | Future |
| REQ-29 | 5 | The application should maintain and allow the editing of an inventory of the quantity of each ingredient available. | Future |
| REQ-30 | 3 | The application should update the inventory based on initial and/or calculated estimated usage whenever a dish is made. | Future |

D. Miscellaneous

| Identifier | Priority | Requirement | Status |
|------------|----------|--|----------|
| REQ-31 | 5 | The application's chef interface should allow them to summon a specific waiter or manager. | Future |
| REQ-32 | 4 | The application should log usage data. | Future |
| REQ-33 | 4 | The application will have a screen to log in to an account. | Complete |
| REQ-34 | 3 | The application will have a way to register an account. | Complete |
| REQ-35 | 3 | The application will have a way to log in via a guest account. | Complete |
| REQ-36 | 5 | The application will restrict certain screens to those users based on account type | Future |
| REQ-37 | 2 | The application will include a setting for large text. | Future |
| REQ-38 | 3 | The application's waiter interface will display alerts from Customers, Chefs and Managers. | Future |



3.3: Enumerated Nonfunctional Requirements

Note: Priority 5 is highest, 1 is lowest.


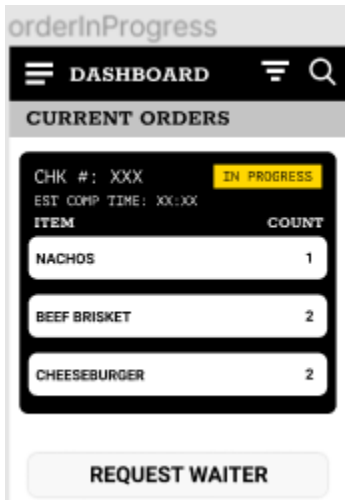
| Identifier | Priority | Requirement |
|------------|----------|---|
| REQ-39 | 4 | The application interface should be capable of functioning on older devices. |
| REQ-40 | 4 | The application should authenticate all orders using the OAuth authentication protocol and also implement AES encryption in between and within subsystems. |
| REQ-41 | 3 | The application should be accessible from any device with at least 1 mbps internet access. |
| REQ-42 | 2 | The application should be able to efficiently use varying levels of processing and/or storage, to service varying levels of loads, and hence be able to scale in the cloud. |
| REQ-43 | 4 | The application should be accessible enough to satisfy WCAG 2. |
| REQ-44 | 3 | The application should provide a large text option and be tested with common text-to-speech providers to assist visually impaired users. |
| REQ-45 | 4 | The application should be responsive, being able to respond to user input within 50 milliseconds or less. |
| REQ-46 | 5 | The application should be able to provide a user with an easy and non-labor-intensive way of getting their job done, within 5 clicks. |
| REQ-47 | 3 | The application should be aesthetically pleasing with eye-catching graphics without being overwhelming. |

3.4: User Interface Requirements

Note: Priority 5 is highest, 1 is lowest.

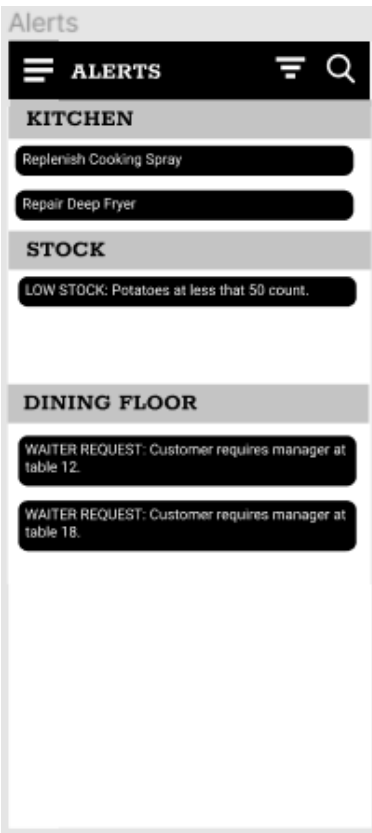
| Identifier | Priority | Requirement | Image |
|------------|----------|--|---|
| REQ-48 | 5 | The customer interface should have a menu that lists all the items that the restaurant offers. |  |
| REQ-49 | 2 | The customer interface should obscure items that are no longer available. |  |

| | | | |
|--------|---|---|---|
| REQ-50 | 3 | The customer interface should have a section in which the user can filter the results from | <div><div>filterOverlay</div><div><div>FILTER BY</div><div>NAME</div><div>RECENTLY ADDED</div><div>LOWEST PRICE</div><div>HIGHEST PRICE</div><div>CATEGORY</div><div>INGREDIENT</div><div><div>+</div><div>-</div></div></div></div> |
| REQ-51 | 5 | The customer interface should have an order page listing the customer's selections and permitting payment | <div><div>OrderConfirmation</div><div><div>← CHECKOUT</div><div><div><div></div><div>ITEM NAME QUANTITY \$XX.XX</div></div><div><div></div><div>ITEM NAME QUANTITY \$XX.XX</div></div><div><div></div><div>ITEM NAME QUANTITY \$XX.XX</div></div><div><div></div><div>ITEM NAME QUANTITY \$XX.XX</div></div></div><div><div>SUBTOTAL: \$XX.XX TAX: \$X.XX TOTAL: \$XX.XX</div><div>CONFIRM ORDER</div></div></div></div> |

| | | | |
|--------|---|---|--|
| REQ-52 | 4 | In the customer interface, there should be a popular item list which lists items that help customers see which items are popular according to the analysis on the popularity of food items. |  |
| REQ-53 | 3 | In the customer interface there should be a tab where a customer can request an additional waiter, utensils, and how long it will take their order, etc. |  |


| | | | | | | | | | | | | | | | |
|-----------|-------|--|--|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|
| REQ-54 | 5 | The manager interface should visualize the popularity and trends of the items. (The X image represents a chart) | <div><div>Statistics</div><div><div>≡</div>STATISTICS</div><div>ITEM NAME</div><div></div><div><table><tr><td>STAT NAME</td><td>VALUE</td></tr><tr><td>STAT NAME</td><td>VALUE</td></tr><tr><td>STAT NAME</td><td>VALUE</td></tr><tr><td>STAT NAME</td><td>VALUE</td></tr><tr><td>STAT NAME</td><td>VALUE</td></tr><tr><td>STAT NAME</td><td>VALUE</td></tr></table></div></div> | STAT NAME | VALUE | STAT NAME | VALUE | STAT NAME | VALUE | STAT NAME | VALUE | STAT NAME | VALUE | STAT NAME | VALUE |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| STAT NAME | VALUE | | | | | | | | | | | | | | |
| REQ-55 | 5 | The manager interface should be able to view and edit the menu. | <div><div>Menu</div><div><div>≡</div>MENU EDITOR<div>≡</div><div>🔍</div></div><div>EDIT CATEGORIES</div><div>POPULAR</div><div><div></div>ITEM NAME PRICE: \$XX.XX</div><div>BREAKFAST</div><div><div></div>ITEM NAME PRICE: \$XX.XX</div><div><div></div>ITEM NAME PRICE: \$XX.XX</div><div><div></div>ITEM NAME PRICE: \$XX.XX</div><div>LUNCH</div><div><div></div>ITEM NAME PRICE: \$XX.XX<div>+</div></div></div> | | | | | | | | | | | | |

| REQ-56 | 4 | The manager interface should be able to view and edit ingredient count. | <div><div>Stock Count</div><div><div><div>STOCK COUNT</div><div></div></div><table><tr><th>ITEM</th><th>VALUE</th><th>UNIT</th></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr></table><div>EDIT VALUES</div></div></div> | ITEM | VALUE | UNIT | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS |
|-------------|---------|--|--|------|-------|-------|------------|---------|---------|-------------|---------|--------|------------|---------|---------|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|
| ITEM | VALUE | UNIT | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REQ-57 | 4 | The manager interface should be able to view and edit past and present orders. | <div><div>editOrder</div><div><div><div>EDIT ORDER</div><div><div>CHK #: XXX TIME CREATED: XX:XX:XX SUBTOTAL: \$XX.XX TOTAL: \$XX.XX</div><div><div>SUBSTATS</div><div><div>STAT</div><div>VALUE</div></div><div><div>NAME</div><div>JOHN SMITH</div></div><div><div>TABLE</div><div>XXX</div></div><div><div>TIME ELAPSED</div><div>XX:XX:XX</div></div><div><div>+</div><div>-</div></div></div><div><div>ORDERED</div><table><tr><th>ITEM</th><th>COUNT</th><th>PRICE</th></tr><tr><td>MINI PIE</td><td>2</td><td>\$12.99</td></tr><tr><td>PIZZA SLICE</td><td>1</td><td>\$9.99</td></tr><tr><td>SUBTOTAL</td><td>2</td><td>\$13.98</td></tr></table></div></div></div><div><div>Order Queue</div><div><div>ORDER QUEUE</div><div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>CHK #: XXX CUSTOMER NAME TABLE: XXX SUBTL: \$XX.XX TIME: XX:XX:XX TOTAL: \$XX.XX</div><div>+</div></div></div></div></div></div> | ITEM | COUNT | PRICE | MINI PIE | 2 | \$12.99 | PIZZA SLICE | 1 | \$9.99 | SUBTOTAL | 2 | \$13.98 | | | | | | | | | | | | | | | | | | | | | | | | |
| ITEM | COUNT | PRICE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MINI PIE | 2 | \$12.99 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PIZZA SLICE | 1 | \$9.99 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SUBTOTAL | 2 | \$13.98 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |


| | | | |
|--------|---|---|--|
| REQ-58 | 5 | The manager interface should be alerted if any ingredient is below a certain threshold or insufficient to make a dish that uses it. |  |
|--------|---|---|--|

| | | | |
|--------|---|--|--|
| REQ-59 | 4 | The chef interface should be able to view and edit current orders. | <div><div>Order Queue</div><div><div><div>≡</div><div>ORDER QUEUE</div><div>≡</div><div>Q</div></div><div><div>CHK #: XXX</div><div>ACTIVE TIME: XX:XX:XX</div><div><div>IN PROGRESS</div></div><div><div>ITEM</div><div>COUNT</div><div>NACHOS</div><div>1</div></div><div><div>BEEF BRISKET</div><div>2</div></div><div><div>CHEESEBURGER</div><div>2</div></div></div><div><div>CHK #: XXX</div><div>ACTIVE TIME: XX:XX:XX</div><div><div>PENDING</div></div><div><div>ITEM</div><div>COUNT</div><div>APPLE PIE</div><div>1</div></div><div><div>BROWNIE</div><div>1</div></div></div><div><div>CHK #: XXX</div><div>ACTIVE TIME: XX:XX:XX</div><div><div>PENDING</div></div><div><div>ITEM</div><div>COUNT</div><div>CHIKEN SANDWICH</div><div>2</div></div><div><div>PIZZA SLICE</div><div>2</div></div><div><div>⚙</div></div></div></div></div> |
|--------|---|--|--|

| REQ-60 | 4 | The chef interface should be able to view and edit ingredient count. | <div><div>Inventory</div><div><div><div>≡</div><div>INVENTORY</div><div>≡</div><div>Q</div></div><table><tr><th>ITEM</th><th>COUNT</th><th>UNIT</th></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr><tr><td>INGREDIENT</td><td>XXXX.XX</td><td>LBS</td></tr></table><div>REQUEST RESTOCK</div><div>edit buttons</div><div><div>✓</div><div>✗</div></div></div></div> | ITEM | COUNT | UNIT | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS | INGREDIENT | XXXX.XX | LBS |
|------------|---------|--|---|------|-------|------|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|------------|---------|-----|
| ITEM | COUNT | UNIT | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INGREDIENT | XXXX.XX | LBS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | |
|--------|---|--|---|
| REQ-61 | 4 | The chef interface should be alerted if any ingredient is below a certain threshold or insufficient to make a dish that uses it. |  A mobile application interface titled 'Alerts' with a dark header bar containing a menu icon, the word 'ALERTS', and a search icon. Below the header, there are two sections: 'KITCHEN' and 'INVENTORY'. The 'KITCHEN' section contains two alert buttons: 'Replenish Cooking Spray' and 'Repair Deep Fryer'. The 'INVENTORY' section contains two alert buttons: 'LOW STOCK: Potatoes at less than 50 count.' and 'LOW STOCK: Insufficient Apples for 5 Apple Pie'. |
|--------|---|--|---|

| | | | |
|--------|---|---|---|
| REQ-62 | 4 | The chef interface should be able to summon a specific waiter or manager. | <div><div>Request</div><div><div>≡</div><div>REQUEST</div><div>≡ Q</div></div><div><div>JEB NOITRA</div><div>M</div></div><div><div>SALLY BASKIN</div><div>W</div></div><div><div>ZAKI SADLER</div><div>C</div></div><div><div>JOHN MOLINA</div><div>W</div></div><div><div>KURT VAUGHAN</div><div>W</div></div><div><div>KENZO EWING</div><div>J</div></div><div><div>JAYLA BURNS</div><div>C</div></div><div><div>RUBY TURNER</div><div>C</div></div><div><div>requestOverlay</div><div>REQUEST</div></div></div> |
|--------|---|---|---|

| | | | |
|--------|---|---|---|
| REQ-63 | 3 | The waiter interface should display the orders made and the progress on each order. |  |
| REQ-64 | 3 | The waiter interface should display alerts from chefs or customers |  |

| | | | |
|--------|---|--|---|
| REQ-65 | 3 | The waiter interface should display the menu as the customer sees it | <div><div>ItemViewer</div><div></div><div>ITEM NAME PRICE: \$XX.XX</div><div>CATEGORIES FAVORITES</div><div>DESCRIPTION Lorem Ipsum</div></div> <div><div>Menu</div><div><div>← SELECT ITEMS</div><div>≡</div><div>Q</div></div><div>POPULAR</div><div> ITEM NAME PRICE: \$XX.XX</div><div>BREAKFAST</div><div> ITEM NAME PRICE: \$XX.XX</div><div> ITEM NAME PRICE: \$XX.XX</div><div> ITEM NAME PRICE: \$XX.XX</div><div>LUNCH</div><div> ITEM NAME PRICE: \$XX.XX</div><div> ITEM NAME</div></div> |
|--------|---|--|---|

Section 4: Functional Requirements Specification

4.1: Stakeholders

1. Restaurant Owner - Have an interest in improving worker efficiency so that better service will enable better word-of-mouth which in turn can cause an increase in revenue.
2. Managers - Keeping track of operations and maintaining quality will allow smoother operation between the wait staff and the kitchen.

4.2: Actors and Goals

Definitions: Item status (not prepared, being prepared, prepared)

- i. Customers: Initiating, Order food, satisfactory dining experience, dish attribute comprehension (attributes include: description, allergens/ingredients)
- ii. Waiters: Initiating, Deliver food, order food if the customer cannot, customer satisfaction, order tracking, keeps track of complete dishes (async).
- iii. Chef: Initiating, Cook food, track ingredient stock, minimize waste, order tracking (only output no input)
- iv. Managers: Initiating, Track ingredient stock, minimize waste, item trend tracking,
- v. Application: Participating, Restaurant automation

4.3: Use Cases

i. Casual Description

Ordering

- UC-1: A customer will be able to use the system to place an order, from the available dishes, through their phone in addition to directly via the waiter by requesting they come to their table. **REQ-1, REQ-2, REQ-3, REQ-17, REQ-19, REQ-33, REQ-40.**
- UC-2: Customers will be able to see, collect, remove, or edit quantities of items in their shopping cart or order. They will be able to add notes as well. **REQ-1, REQ-5, REQ-17.**
- UC-3: Waiters will be able to see all relevant information and be able to create and edit orders on behalf of the customer. **REQ-7, REQ-8, REQ-9.**
- UC-4: Managers will be able to see all relevant information, and the ability to view and edit orders, prices and descriptions. **REQ-10, REQ-11, REQ-23, REQ-27.**
- UC-5: Chefs will be able to see all relevant information and the ability to view and edit ingredients, wait time, and availability. **REQ-13, REQ-28, REQ-29.**
- UC-6: After an order is placed, users should be able to view progress of their order from pending to delivered. This will correspond to **REQ-3, REQ-4, REQ-14.**
- UC-7: The chef should be able to view and update the status of each item in the current order queue. This will correspond to **REQ-13.**
- UC-8: The customer should be able to request a waiter to come to their table, as well as request extra utensils, or other miscellaneous requests through the app. This will correspond to **REQ-19, REQ-38.**
- UC-9: The manager will be able to view and edit the current orders placed, as well as archived orders placed. This will correspond to **REQ-10, REQ-11, REQ-39.**

Smart Menu

- UC-10: All users will be able to see all available items and recommended items (based on the popularity of dishes), which will be kept up to date by the manager and by the application automatically. They should also see other relevant information regarding the dish. This will correspond to **REQ-17, REQ-18, REQ-24, REQ-32, REQ-42.**
- UC-11: A waiter will be able to see the current status of any order placed and track it at any time. This will correspond to **REQ-7, REQ-14.**
- UC-12: The manager will be able to see all available items and ingredients while being able to edit them. The manager can hide items that should not be seen by customers or add new ones. This will correspond to **REQ-23, REQ-27, REQ-29, .**

Ingredient Tracking

- UC-13: The manager and chef will be able to view and edit the ingredient stock count and units it is measured in. This will correspond to **REQ-27, REQ-28, REQ-29.**

UC-14: The manager will be able to view statistics regarding the preparation of all meals including: ingredients used, most popular dishes, and most common time a meal is ordered. This will correspond to **REQ-24, REQ-32**.

UC-15: The manager and chef should receive an alert when a dish cannot be made due to insufficient stock and when an ingredient is below a certain threshold. This will correspond to **REQ-25, REQ-26, REQ-30**.

Miscellaneous

UC-16: The chef should be able to summon a specific waiter or manager. This will correspond to **REQ-31, REQ-38**.

UC-17: The stakeholders of the app will be able to see logged user data. **REQ-32**.

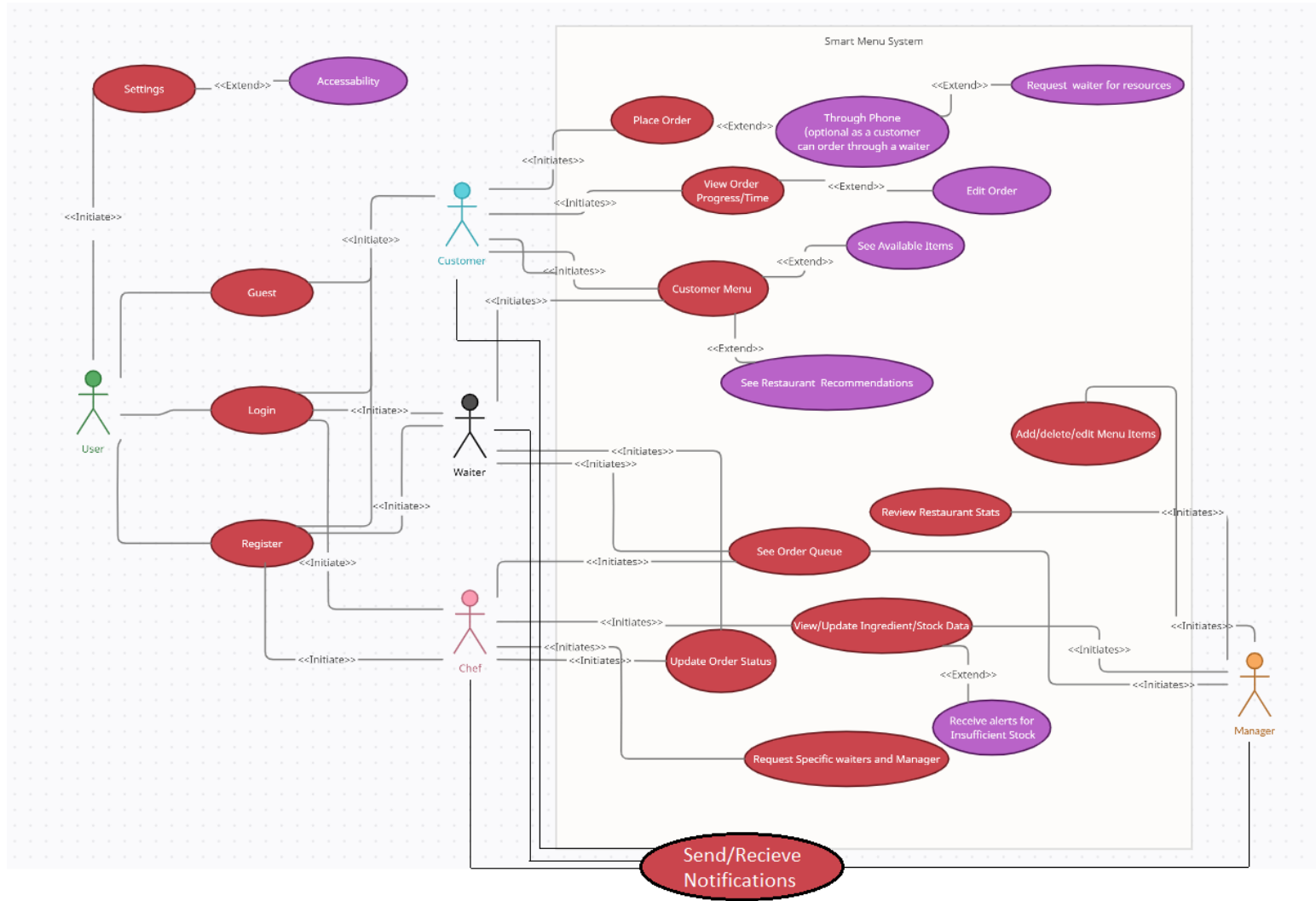
UC-18: It will be possible to create an account, log in to an account, or log in to a guest account on most devices. **REQ-33, REQ-34, REQ-35, REQ-36, REQ-39, REQ-41, REQ-43, REQ-46**.

UC-19: Users will be able to access screens that they have permission to and will not see screens they aren't permitted to see. **REQ-20, REQ-21, REQ-22, REQ-36**.

UC-20: Users will be capable of enlarging text for accessibility. **REQ-37**.

UC-21: Waiters, chefs and managers can receive notifications, and customers, chefs and managers can send notifications. **REQ-38**.

ii. Use Case Diagram



iii. Traceability Matrix

REQ = Requirement

UC = use case

| | UC | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|-----|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|
| REQ | Weight | | | | | | | | | | | | | | | | | | | | | |
| 1 | 4 | X | X | | | | | | | | | | | | | | | | | | | |
| 2 | 4 | X | | | | | | | | | | | | | | | | | | | | |
| 3 | 4 | X | | | | | X | | | | | | | | | | | | | | | |
| 4 | 4 | | | | | | X | | | | | | | | | | | | | | | |
| 5 | 2 | | X | | | | | | | | | | | | | | | | | | | |
| 6 | 3 | | | | | | | | | | | | | | | | | | | | | |
| 7 | 3 | | | X | | | | | | | | X | | | | | | | | | | |
| 8 | 3 | | | X | | | | | | | | | | | | | | | | | | |
| 9 | 3 | | | X | | | | | | | | | | | | | | | | | | |
| 10 | 4 | | | | X | | | | | X | | | | | | | | | | | | |
| 11 | 4 | | | | X | | | | | X | | | | | | | | | | | | |
| 12 | 4 | | | | | | | | | | | | | | | | | | | | | |
| 13 | 4 | | | | | X | | X | | | | | | | | | | | | | | |
| 14 | 1 | | | | | | X | | | | | X | | | | | | | | | | |
| 15 | 5 | | | | | | | | | | | | | | | | | | | | | |
| 16 | 5 | | | | | | | | | | | | | | | | | | | | | |
| 17 | 4 | X | X | | | | | | | | X | | | | | | | | | | | |
| 18 | 3 | | | | | | | | | | X | | | | | | | | | | | |
| 19 | 2 | X | | | | | | | X | | | | | | | | | | | | | |
| 20 | 4 | | | | | | | | | | | | | | | | | | | X | | |

| | | | | | | | | | | | | | | | | | | | | | |
|----|---|---|--|--|---|---|--|--|---|---|---|---|---|---|---|---|---|--|---|---|---|
| 21 | 4 | | | | | | | | | | | | | | | | | | X | | |
| 22 | 4 | | | | | | | | | | | | | | | | | | X | | |
| 23 | 5 | | | | X | | | | | | | X | | | | | | | | | |
| 24 | 2 | | | | | | | | | X | | | | X | | | | | | | |
| 25 | 4 | | | | | | | | | | | | | | X | | | | | | |
| 26 | 3 | | | | | | | | | | | | | | X | | | | | | |
| 27 | 4 | | | | X | | | | | | | | X | X | | | | | | | |
| 28 | 4 | | | | | X | | | | | | | X | | | | | | | | |
| 29 | 5 | | | | | X | | | | | | | X | X | | | | | | | |
| 30 | 3 | | | | | | | | | | | | | | | X | | | | | |
| 31 | 5 | | | | | | | | | | | | | | | X | | | | | |
| 32 | 4 | | | | | | | | | X | | | | X | | | X | | | | |
| 33 | 4 | X | | | | | | | | | | | | | | | | | X | | |
| 34 | 3 | | | | | | | | | | | | | | | | | | X | | |
| 35 | 3 | | | | | | | | | | | | | | | | | | X | | |
| 36 | 5 | | | | | | | | | | | | | | | | | | X | X | |
| 37 | 2 | | | | | | | | | | | | | | | | | | | X | |
| 38 | 3 | | | | | | | | X | | | | | | | X | | | | | X |
| 39 | 4 | | | | | | | | | X | | | | | | | | | X | | |
| 40 | 4 | X | | | | | | | | | | | | | | | | | | | |
| 41 | 3 | | | | | | | | | | | | | | | | | | X | | |
| 42 | 2 | | | | | | | | | | X | | | | | | | | | | |
| 43 | 4 | | | | | | | | | | | | | | | | | | X | | |
| 44 | 3 | | | | | | | | | | | | | | | | | | | | |
| 45 | 4 | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | |
|----|---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|---|--|--|--|
| 46 | 5 | | | | | | | | | | | | | | | | | X | | | |
| 47 | 3 | | | | | | | | | | | | | | | | | | | | |
| 48 | 5 | | | | | | | | | | | | | | | | | | | | |
| 49 | 2 | | | | | | | | | | | | | | | | | | | | |
| 50 | 3 | | | | | | | | | | | | | | | | | | | | |
| 51 | 5 | | | | | | | | | | | | | | | | | | | | |
| 52 | 4 | | | | | | | | | | | | | | | | | | | | |
| 53 | 3 | | | | | | | | | | | | | | | | | | | | |
| 54 | 5 | | | | | | | | | | | | | | | | | | | | |
| 55 | 5 | | | | | | | | | | | | | | | | | | | | |
| 56 | 4 | | | | | | | | | | | | | | | | | | | | |
| 57 | 4 | | | | | | | | | | | | | | | | | | | | |
| 58 | 5 | | | | | | | | | | | | | | | | | | | | |
| 59 | 4 | | | | | | | | | | | | | | | | | | | | |
| 60 | 4 | | | | | | | | | | | | | | | | | | | | |
| 61 | 4 | | | | | | | | | | | | | | | | | | | | |
| 62 | 4 | | | | | | | | | | | | | | | | | | | | |
| 63 | 3 | | | | | | | | | | | | | | | | | | | | |
| 64 | 3 | | | | | | | | | | | | | | | | | | | | |
| 65 | 3 | | | | | | | | | | | | | | | | | | | | |

iv. Fully-Dressed Description

| |
|--|
| Use Case UC-1: Place Order |
| <p>Related Requirements: REQ-1, REQ-2, REQ-3, REQ-17, REQ-19, REQ-33, REQ-40.</p> <p>Initiating Actor: Any of: Customer, Waiter</p> <p>Actor's Goal: To place an order to be sent to the kitchen to be prepared for the customer.</p> <p>Participating Actors: None</p> <p>Preconditions:</p> <ol style="list-style-type: none">1. Customer is logged in, either as a guest user or to their own account.2. There is at least one item in the order3. All items in the order are available <p>Minimal Guarantees: The order will not get sent to Order Queue if systems fails.</p> <p>Success Guarantees: Order will be sent to Order Queue.</p> <p>Flow of Events for Main Success Scenario:</p> <ol style="list-style-type: none">1. Customer selects the Smart menu option.2. Customer selects the items they would like to order.3. Customer selects menu item "Place Order"4. Customer receives order confirmation message. |

Use Case UC-2: Shopping Cart

Related Requirements: **REQ-1, REQ-5, REQ-17.**

Initiating Actor: Any of: Customer, Waiter

Actor's Goal: To create and curate a list of items that they wish to later order..

Participating Actors: None

Preconditions:

1. Customer is logged in, either as a guest user or to their own account.

Minimal Guarantees: The existing list will not be added to/edited.

Success Guarantees: The edit is made to the list.

Flow of Events for Main Success Scenario:

1. Customer selects the Smart menu option.
2. Customer adds items to the cart from the menu.
 - a. If needed, customer edits the quantity of an item.
 - b. If needed, customer removes an item from the cart.
3. Customer clicks on Place order.

Use Case UC-10: Item List & Item Viewer

Related Requirements: **REQ-17, REQ-18, REQ-24, REQ-32, REQ-42.**

Initiating Actor: Customer

Actor's Goal: To see a list of available items and see more information about the items.

Participating Actors: None

Preconditions: 1. Customer is logged in, either as a guest user or to their own account.

Minimal Guarantees: The customer will see a list of all available items

Success Guarantees: The customer will see a list of available items, and also a list of popular items. Clicking on an item will show further details about it.

Flow of Events for Main Success Scenario:

1. Customer selects the Smart Menu option.
2. Customer examines the displayed list and comprehends the given information.
 - a. Should further details about an item be required, customer selects a specific item and is shown more information.

Use Case UC-18: Register & Login

Related Requirements: **REQ-33, REQ-34, REQ-35, REQ-36, REQ-39, REQ-41, REQ-43, REQ-46.**

Initiating Actor: Customer, Waiter, Chef, Manager

Actor's Goal: To track their activities, and uniquely identify themselves.

Participating Actors: None

Preconditions: None

Minimal Guarantees: The user's recent activities are tracked on a temporary account. The user is uniquely identified.

Success Guarantees: All of the user's activities, recent and past, are tracked. The user is uniquely identified.

Flow of Events for Main Success Scenario:

1. User selects the desired option
 - a. If Register is selected, user is prompted to input data which is used to generate a new account.
 - b. If Login is selected, user is prompted to input data which is used to authenticate and link their session to their user account.
 - c. If Guest Login is selected, the user continues directly to the next step with no more effort
2. User is taken to the menu screen.

Use Case UC-19: Access Permissions

Related Requirements: **REQ-20, REQ-21, REQ-22, REQ-36.**

Initiating Actor: Customer, Waiter, Chef, Manager

Actor's Goal: To access screens they are permitted to see but not see screens they are not permitted to see..

Participating Actors: None

Preconditions: 1. User is logged in, either as a guest user or to their own account.

Minimal Guarantees: The user will see only the common pages which are permitted for everyone.

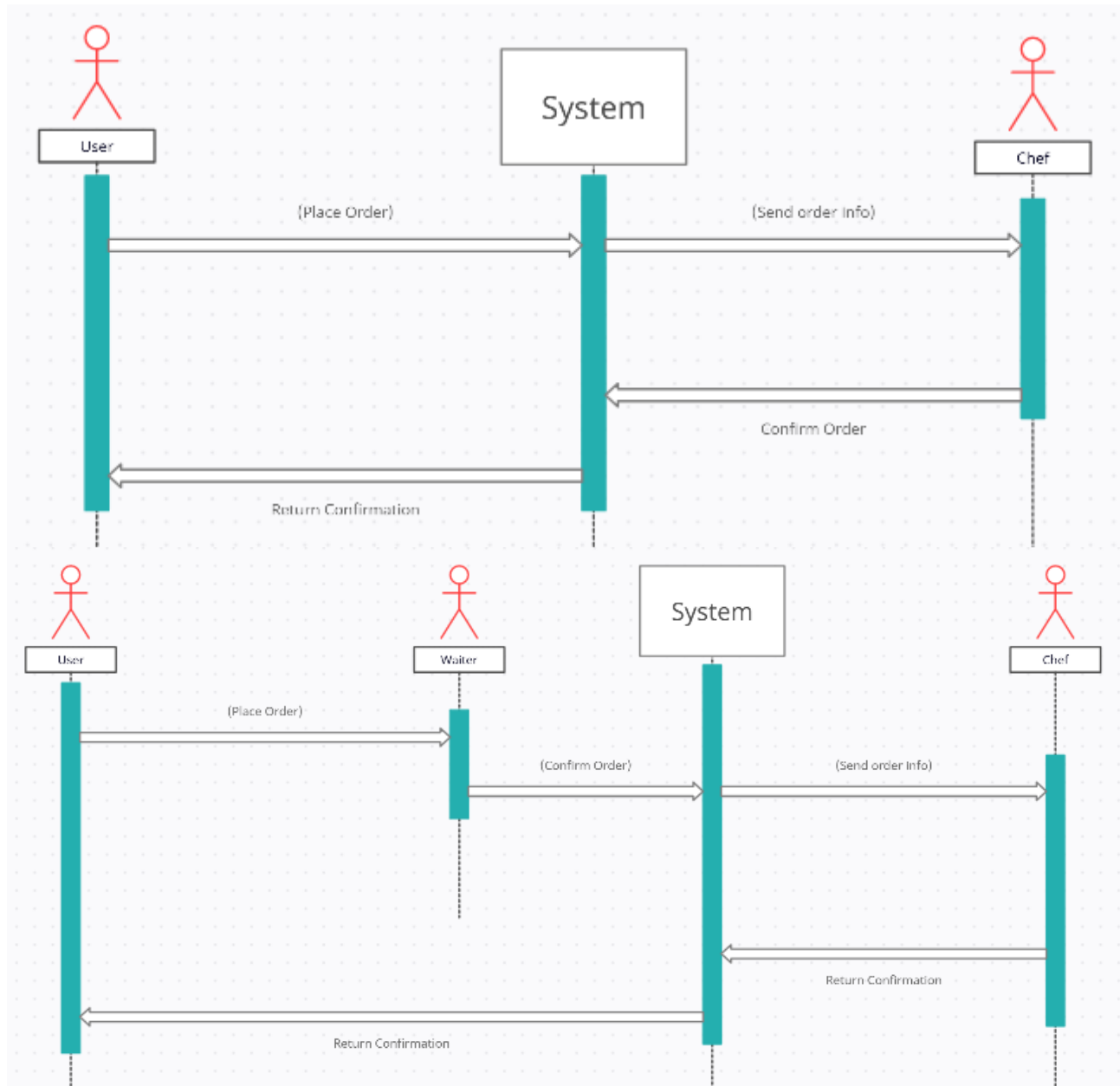
Success Guarantees: The user will see the common pages and also the unique pages which they are permitted to access.

Flow of Events for Main Success Scenario:

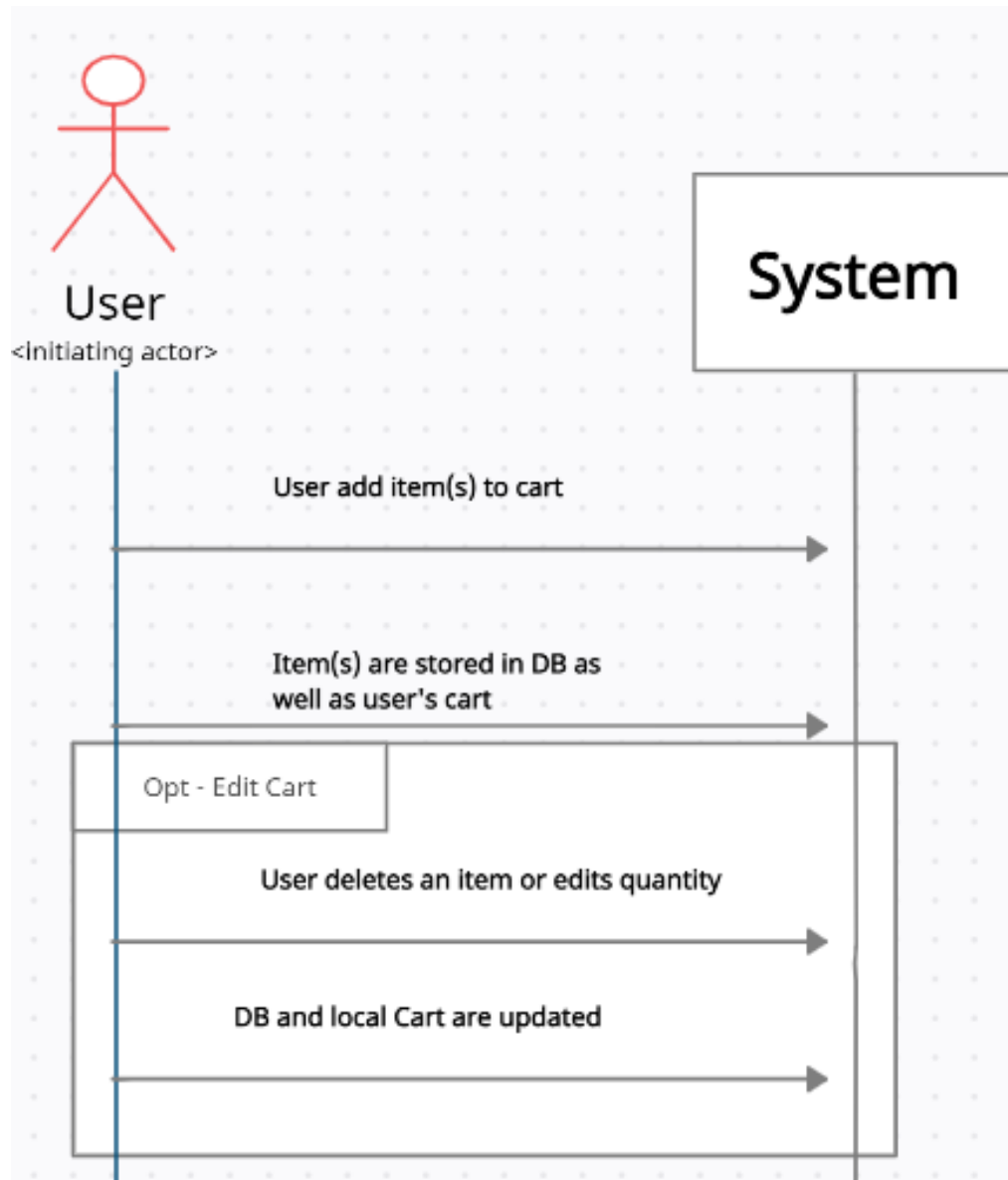
1. User opens the app.
2. User may navigate around to certain screens.
 - a. Common screens: User is always able to navigate there.
 - b. Unique screens: User may navigate here if they have permission to see this screen.

4.4: System Sequence Diagrams

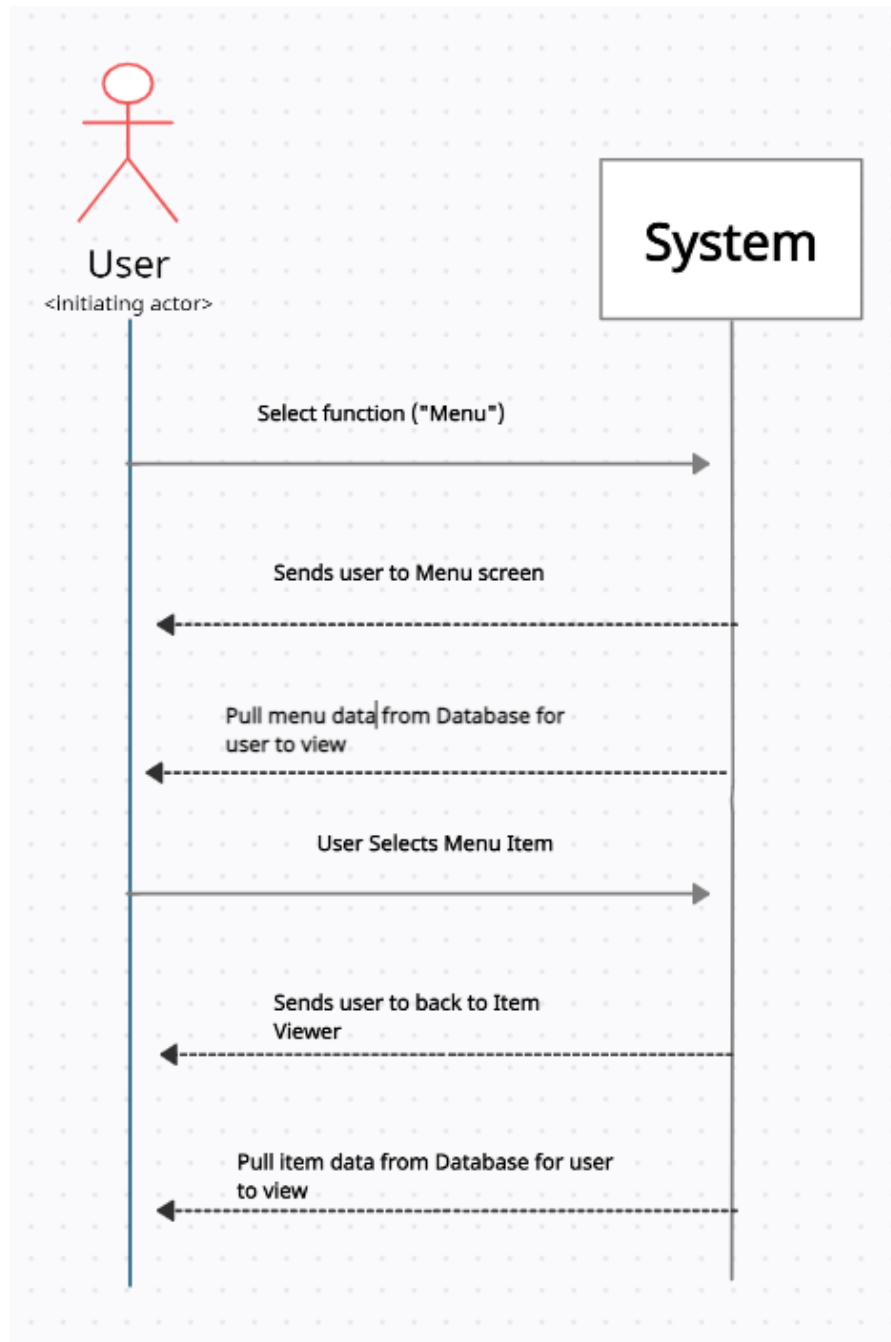
UC-1: Place Order



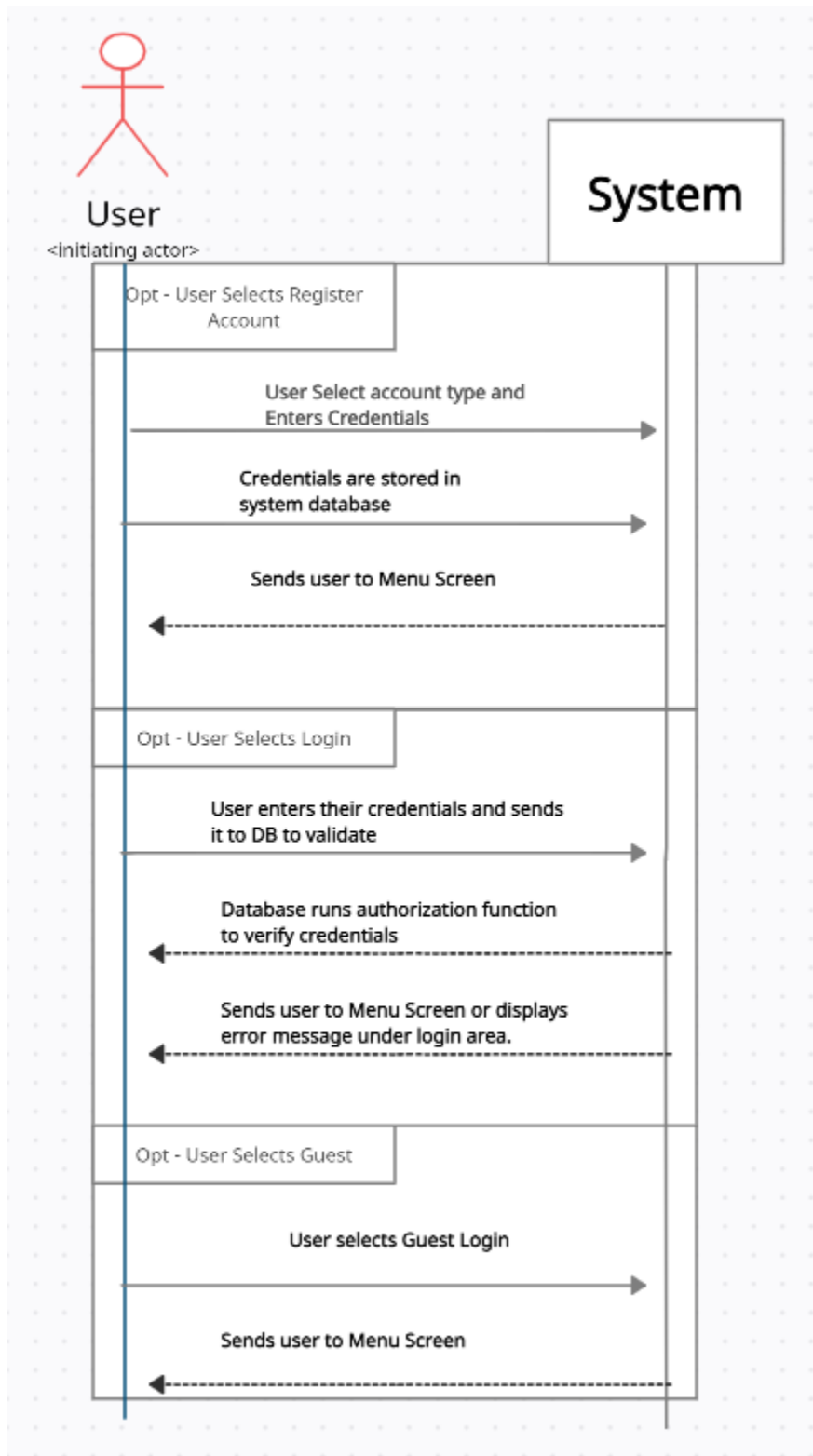
UC-2: Shopping Cart



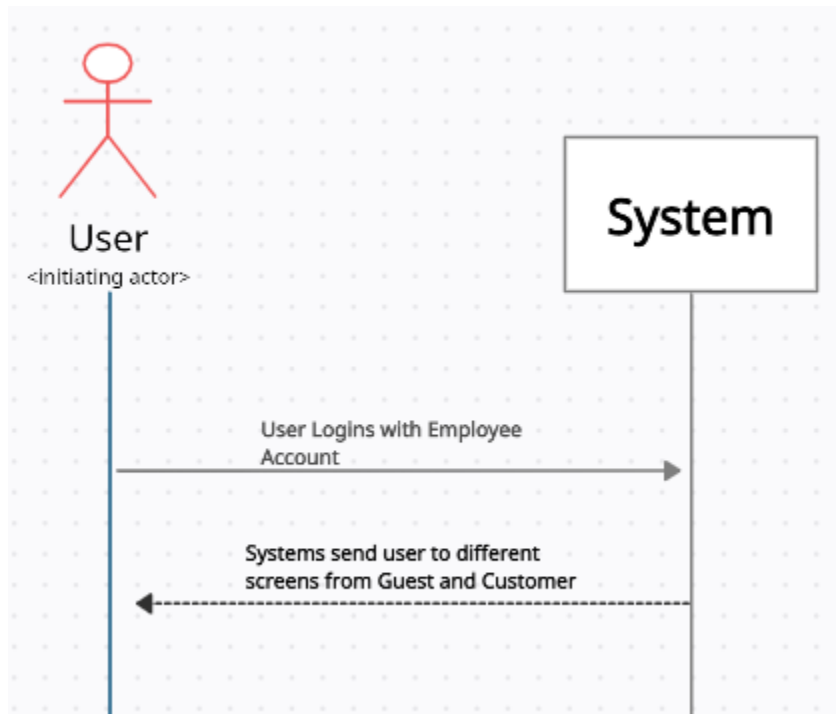
UC-10: Item List and Viewer



UC-18: Register and Login



UC-19: Permission-Locked Screens



Section 5: Effort Estimation using Use Case Points

| Use Case | Navigation | Clerical Data Entry |
|----------|---|--|
| UC-1 | <p>REQ-1, REQ-2, REQ-3, REQ-17, REQ-19, REQ-33, REQ-40.</p> <ol style="list-style-type: none"> 1. Navigate to the menu <ol style="list-style-type: none"> a. Login > Menu 2. Select items to order 3. Press order overlay button. 4. Confirm order at redirected checkout. <p>Total clicks/presses: 5</p> <ol style="list-style-type: none"> 1. Customers can also request a waiter from the application. <ol style="list-style-type: none"> a. Login > Menu > Request waiter 2. Order items through the waiter. <p>Total clicks/presses: 3</p> | <ol style="list-style-type: none"> 1. Enter First Name 2. Enter Last Name 3. Enter Card Number 4. Enter Card Exp. Date 5. Enter Card CVV 6. Select “Pay Now” <p>Total clicks/presses: 1 Total entries: 5</p> |
| UC-2 | <p>REQ-1, REQ-5, REQ-17.</p> <ol style="list-style-type: none"> 1. Navigate to the order screen before checkout, the shopping cart will display current items. <ol style="list-style-type: none"> a. Login > Orders 2. Press edit order. 3. Swipe left to delete order or edit quantities of items. <p>Total clicks/presses: 4</p> | <ol style="list-style-type: none"> 1. Enter Number of Items 2. Select “Confirm” <p>Total clicks/presses: 1 Total entries: 1</p> |
| UC-3 | <p>REQ-7, REQ-8, REQ-9</p> | <ol style="list-style-type: none"> 1. Enter Number of Items 2. Select “Confirm” |

| | | |
|------|---|---|
| | <ol style="list-style-type: none"> Waiters can navigate to the menu. <ol style="list-style-type: none"> Login > Menu Select items to order. Press order overlay button. <p>Total number of clicks/presses: 4</p> <ol style="list-style-type: none"> Waiters can navigate to the shopping cart. <ol style="list-style-type: none"> Login > Orders Press edit order. Swipe left to delete order or edit quantities of items. <p>Total number of clicks/presses: 4</p> | <p>Total clicks/presses: 1 Total entries: 1</p> |
| UC-4 | <p>REQ-10, REQ-11, REQ-23, REQ-27.</p> <ol style="list-style-type: none"> Managers can navigate to the order queue to view orders. <ol style="list-style-type: none"> Login > Order Queue Press edit order. Swipe left to delete order or edit quantities of items. <p>Total number of clicks/presses: 4</p> <ol style="list-style-type: none"> Managers can navigate to the menu to view prices and descriptions. <ol style="list-style-type: none"> Login > Menu Press edit item. Prices and descriptions can be changed. <p>Total number of clicks/presses: 3</p> | <ol style="list-style-type: none"> Enter Number of Items Enter Price Enter Description Select “Confirm” <p>Total clicks/presses: 1 Total entries: 3</p> |
| UC-5 | <p>REQ-13, REQ-28, REQ-29.</p> | <ol style="list-style-type: none"> Enter Wait Time Enter Ingredients Check/Uncheck availability box |

| | | |
|------|---|--|
| | <ol style="list-style-type: none"> 1. Chefs can navigate to the menu to view item details (wait time, ingredients, and availability). <ol style="list-style-type: none"> a. Login > Menu 2. Press edit item. 3. Wait time, ingredients, and availability can be changed. <p>Total number of clicks/presses: 3</p> | <ol style="list-style-type: none"> 4. Select “Confirm” <p>Total clicks/presses: 2 Total entries: 2</p> |
| UC-6 | <p>REQ-3, REQ-4, REQ-14</p> <ol style="list-style-type: none"> 1. Navigate to the order that was placed. <ol style="list-style-type: none"> a. Login > Orders 2. The page will show a progress bar and an estimated delivery time. <p>Total clicks/presses: 2</p> | |
| UC-7 | <p>REQ-13.</p> <ol style="list-style-type: none"> 1. Chef can navigate to the order queue to view current item orders. <ol style="list-style-type: none"> a. Login > Order Queue 2. Press edit order. 3. Order status can be updated. <p>Total clicks/presses: 3</p> | <ol style="list-style-type: none"> 1. Enter Order Status 2. Select “Confirm” <p>Total clicks/presses: 1 Total entries: 1</p> |
| UC-8 | <p>REQ-19, REQ-38.</p> <ol style="list-style-type: none"> 1. Customers can request a waiter from the application. <ol style="list-style-type: none"> a. Login > Menu > Request Waiter <p>Total clicks/presses: 3</p> <ol style="list-style-type: none"> 1. Navigate to the menu. <ol style="list-style-type: none"> a. Login > Menu > Other Requests | |

| | | |
|-------|---|--|
| | <p>2. A variety of requests will be shown</p> <p>Total clicks/presses: 3</p> | |
| UC-9 | <p>REQ-10, REQ-11, REQ-39</p> <ol style="list-style-type: none"> Managers can navigate to the order queue to view orders. <ul style="list-style-type: none"> Login > Order Queue Press edit order. Swipe left to delete order or edit quantities of items. The orders can also be archived from this screen. <p>Total number of clicks/presses: 3</p> | |
| UC-10 | <p>REQ-17, REQ-18, REQ-24, REQ-32, REQ-42.</p> <ol style="list-style-type: none"> Navigate to the menu <ul style="list-style-type: none"> Login > Menu The user will see all available dishes as well as recommended dishes and all relevant information regarding the dish. <p>Total clicks/presses: 2</p> | |
| UC-11 | <p>REQ-7, REQ-14.</p> <ol style="list-style-type: none"> Waiter can navigate to the order queue. <ul style="list-style-type: none"> Login > Order queue The order queue will show the current status of any orders placed. <p>Total number of clicks/presses: 2</p> | |

| | | |
|-------|--|--|
| | | |
| UC-12 | <p>REQ-23, REQ-27, REQ-29.</p> <ol style="list-style-type: none"> 1. Manager can navigate to the inventory to view all available items and ingredients <ol style="list-style-type: none"> a. Login > Menu > Inventory 2. Press edit inventory. 3. Manager can hide, add, and edit items. <p>Total number of clicks/presses: 4</p> | |
| UC-13 | <p>REQ-27, REQ-28, REQ-29.</p> <ol style="list-style-type: none"> 1. Manager and chef can navigate to the inventory <ol style="list-style-type: none"> a. Login > Menu > Inventory 2. Press edit inventory 3. Manager and chef can edit ingredient stock count and units it is measured in. <p>Total number of clicks/presses: 4</p> | <ol style="list-style-type: none"> 1. Enter Ingredient Stock Count 2. Enter Units 3. Select “Confirm” <p>Total clicks/presses: 1 Total entries: 2</p> |
| UC-14 | <p>REQ-24, REQ-32.</p> <ol style="list-style-type: none"> 1. Manager can navigate to the statistics <ol style="list-style-type: none"> a. Login > Statistics 2. From the statistics page the manager can view ingredients used, popular dishes, and most common time a meal is ordered. | |

| | | |
|-------|--|--|
| | Total number of clicks/presses: 2 | |
| UC-15 | <p>REQ-25, REQ-26, REQ-30.</p> <ol style="list-style-type: none"> 1. Manager and chef can navigate to the alerts <ol style="list-style-type: none"> a. Login > Alerts 2. In the alerts tab, the manager and chef can receive automatic notifications when a ingredient stock is below a certain threshold <p>Total number of clicks/presses: 4</p> | |
| UC-16 | <p>REQ-31, REQ-38.</p> <ol style="list-style-type: none"> 1. Chef can request a waiter the same way a customer can <ol style="list-style-type: none"> a. Login > Menu > Request Waiter <p>Total number of clicks/presses: 3</p> <ol style="list-style-type: none"> 1. Chef can also request a manager <ol style="list-style-type: none"> a. Login > Menu > Request Manager <p>Total number of clicks/presses: 3</p> | |
| UC-17 | <p>REQ-32.</p> <ol style="list-style-type: none"> 1. Navigate to the settings <ol style="list-style-type: none"> a. Login > Settings 2. Press the logs button 3. Stakeholders can view the logged user data <p>Total number of clicks/presses: 3</p> | |

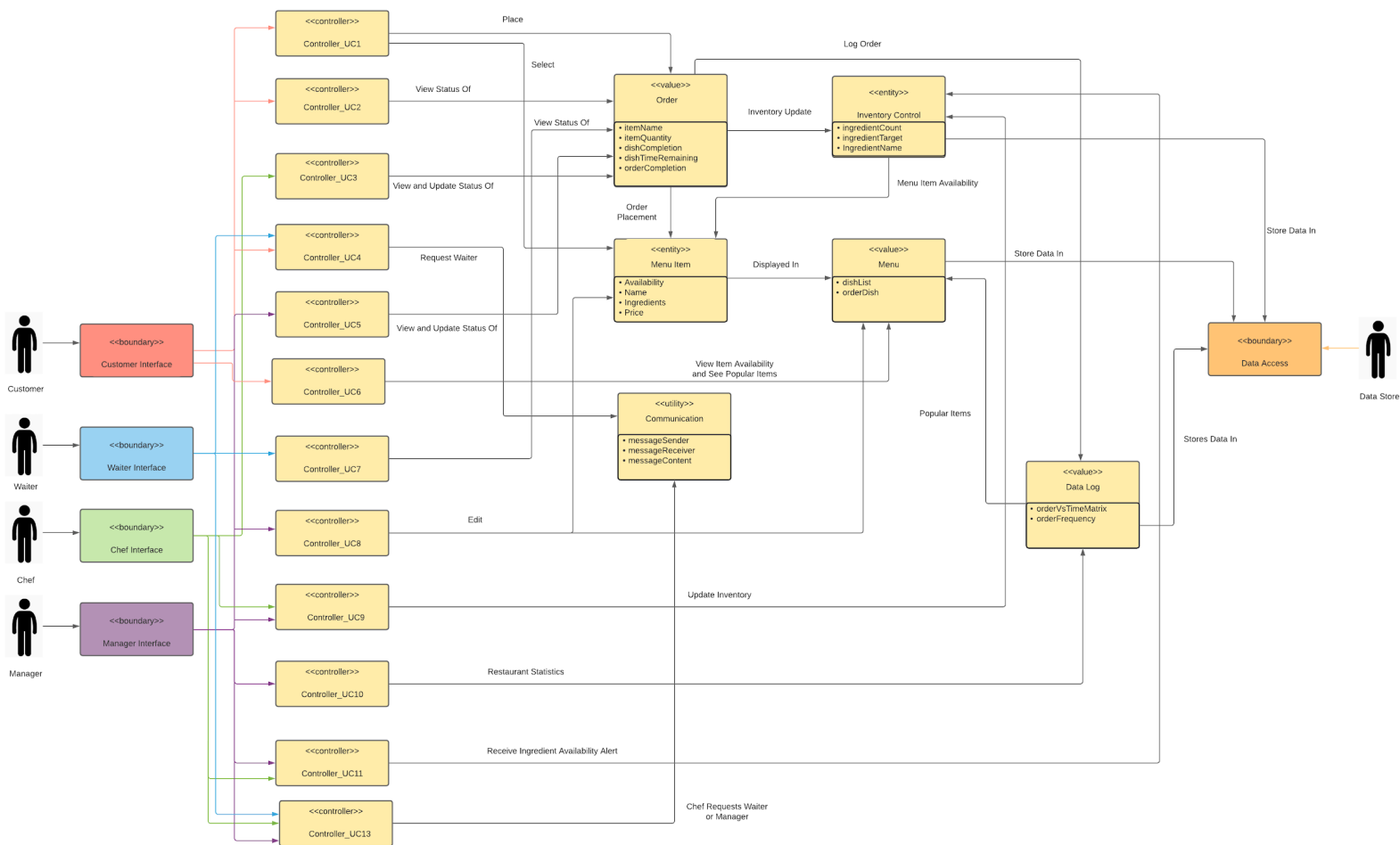
| | | |
|-------|--|--|
| UC-18 | <p>REQ-33, REQ-34, REQ-35, REQ-36, REQ-39, REQ-41, REQ-43, REQ-46.</p> <ol style="list-style-type: none"> 1. Register for an account, login to an account, or sign in to a guest account on the login screen. <ol style="list-style-type: none"> a. Open application > Login <p>Total number of clicks/presses: 1</p> | <ol style="list-style-type: none"> 1. Enter Name 2. Enter Email 3. Enter Password 4. Select Register/Login |
| UC-19 | <p>REQ-20, REQ-21, REQ-22, REQ-36.</p> <ol style="list-style-type: none"> 1. Users will be able to access screens that they have permission to and will not see screens they aren't permitted to see. | |
| UC-20 | <p>REQ-37.</p> <ol style="list-style-type: none"> 1. Navigate to the settings <ol style="list-style-type: none"> a. Login > Settings 2. The user can adjust the size of the text for accessibility <p>Total number of clicks/presses: 2</p> | |
| UC-21 | <p>REQ-38.</p> <ol style="list-style-type: none"> 1. Waiters, chefs, and managers can navigate to the alerts <ol style="list-style-type: none"> a. Login > Alerts 2. Alerts can be viewed here, separately for the waiters, chefs, and managers. | |

| | | |
|--|---|--|
| | <p>Total number of clicks/presses: 2</p> <ol style="list-style-type: none"> 1. Customers, chefs, and managers can navigate can navigate to the requests tab <ol style="list-style-type: none"> a. Login > Menu > Requests 2. Requests can be sent here to the waiter (from the customer, chef, and manager) or to the manager (from the chef) and chef (from the manager). <p>Total number of clicks/presses: 3</p> | |
|--|---|--|

Section 6: Domain Analysis

6.1: Domain Model

Conceptual Model Diagram:



i. Concept Definitions

| Responsibility # | Concept |
|---|-----------------------------|
| R.1 Customer interface lists and facilitates ordering available dishes | Menu 1 |
| R.2 Show items in customer's current order | Orders 1 |
| R.3 Customer can view their current and past orders | Orders 1 |
| R.4 Customer interface will have a screen that displays current order and status of individual items in greater detail | Orders 1 |
| R.5 Customer can edit items in the current order if they are not cooking | Orders 1 |
| R.6 Customer can pay for order | Orders 1 |
| R.7 Waiter can view all orders | Orders 1 |
| R.8 Waiter interface will have screen that displays orders in greater detail and allows editing of orders | Orders 1 |
| R.9 Waiter can order on behalf of customer | Orders 1 |
| R.10 Manager can view all orders | Orders 1 |
| R.11 Manager interface will have screen that displays orders in greater detail and allows editing of orders | Menu 1 |
| R.12 Chef can view all orders | Orders 1 |
| R.13 Chef interface will have screen that displays orders in greater detail and allows editing of orders and order status | Orders 1 |
| R.14 Individual items in an order should have statuses. | Orders 1 |
| R.15 Customer interface will display dishes and allow searching and sorting dishes | Menu 1 |
| R.16 Customer interface will have screen with greater detail about each dish | Menu 1 |
| R.17 Customer cannot select items that are not available | Menu 1, Inventory Control 1 |
| R.18 Customer can view popular dishes in menu | Data Log 1 |
| R.19 Customer can request a waiter | Communication 1 |
| R.20 Waiter can view customer menu | Menu 1 |

| | |
|---|---------------------|
| R.21 Chef can view customer menu | Menu 1 |
| R.22 Manager can view customer menu | Menu 1 |
| R.23 Manager can edit customer menu | Menu 1 |
| R.24 Manager can view dish popularity and trends | Data log 1 |
| R.25 Manager will be alerted if dish has insufficient ingredients | Inventory Control 1 |
| R.26 Manager will be alerted if ingredient below certain threshold | Inventory Control 1 |
| R.27 Manager can view and manually edit ingredient count | Inventory Control 1 |
| R.28 Chef can view and edit ingredient count | Inventory Control 1 |
| R.29 Application should maintain and allow editing of ingredient inventory | Inventory Control 1 |
| R.30 Ingredient inventory should be updated whenever a dish is made | Inventory Control 1 |
| R.31 Chef can summon waiter or manager | Communication 1 |
| R.32 The application will log usage data | Data log 1 |
| R.33 Login Screen | Data log 1 |
| R.34 Application can create and register account | Data log 1 |
| R.35 Application will allow users to log in via guest account | Data log 1 |
| R.36 Application will restrict screens available to users based on account type | Data log 1 |
| R.37 Large text setting | Data log 1 |
| R.38 Waiter can view alerts from customers, chefs, and managers | Communication 1 |

ii. Association Definitions:

| Concept 1 | Concept 2 | Description | Name |
|---------------------|---------------------|---|------------------------|
| Menu 1 | Data Log 1 | Takes data from logs to display popular items for customer menu | Popular Items |
| Menu 1 | Orders 1 | Customer can select items in the Menu and Order them | Order Placement |
| Menu 1 | Inventory Control 1 | Takes data from inventory to determine whether item will be on menu | Menu item availability |
| Data Log 2 | Menu 1 | Takes data from the logs to display Popularity and Trends for manager | Restaurant Statistics |
| Orders 1 | Inventory Control 1 | If an order is completed, the respective ingredient quantities in the inventory will be updated | Inventory Update |
| Inventory Control 1 | Data Log 1 | Inventory control will use data logs to match ingredient counts | Inventory History |
| Communication 1 | Orders 3 | Chef can request waiter to bring the completed dish to the customer | Dish Completion |

iii. Attribute Definitions:

| Concept | Attribute | Definition |
|-------------------|-------------------|--|
| Menu | dishList | Allows customers to view items on the menu |
| | orderDish | Allows customers to order dishes through the menu |
| Orders | dishCompletion | The status of the progress towards the dish's completion |
| | dishTimeRemaining | The expected time remaining for dish's completion |
| | orderCompletion | Confirms the completion of the order, removing it from the order queue |
| Communication | messageSender | Allows the user to send a signal to other application user |
| | messageReceiver | Allows the user to receive a signal from other application user |
| | messageContent | The content of the signal |
| Data Log | orderVsTimeMatrix | Tracks the time taken to complete an order |
| | orderFrequency | Tracks the amount of times a dish is order |
| Inventory Control | ingredientCount | Tracks the amount of ingredients left in the inventory |
| | ingredientTarget | Threshold at which an alert will be issued for ingredient restock |

iv. Traceability Matrix:

| | Domain Concept | Menu | Orders | Communication | Data Log | Inventory Control |
|--------------------|----------------|------|--------|---------------|----------|-------------------|
| UC | Weight | | | | | |
| 1 | | X | | X | | |
| 2 | | | X | | | |
| 3 | | | X | | | |
| 4 | | | | X | | |
| 5 | | | X | | | |
| 6 | | X | | | X | |
| 7 | | | X | | | |
| 8 | | X | | | | |
| 9 | | | X | | X | |
| 10 | | X | | | | |
| 11 | | | X | | | |
| 12 | | X | | | | X |
| 13 | | | | | | X |
| 14 | | | | | X | |
| 15 | | | X | X | | X |
| 16 | | | | X | | |
| 17 | | | | | X | |
| 18 | | | | | X | |
| 19 | | | | | X | |
| 20 | | X | | | | |
| 21 | | | | X | | |

Reference to Matrix Above:

1. A customer will view the menu component, handling both the view and actual ordering of the menu item. Based on the order placement, a signal in the communication component will be sent out to the chef and waiter accordingly.
2. A customer will be able to view their order and make adjustments as part of the order component.
3. A waiter will also be able to view customer orders and make adjustments as part of the order component.
4. Managers will be able to view all relevant signals from waiter to chef, customer to waiter, customer to chef, etc. as part of communication.
5. Chefs will be able to view all relevant signals from waiter to chef, customer to waiter, customer to chef, etc. as part of communication.
6. Customers will be able to view their order via the menu component and information regarding time and other statistics through the data log component.
7. The chef will be able to view orders and make any needed change as part of the order component.
8. The customer will be able to call the waiter through the app in the menu component.
9. The manager will be able to view all orders and statistics - current and archived. As part of the data log and orders component.
10. Users will see all items in the up to date menu component, as well as other dish information, which is kept up to date by the manager.
11. The waiter will be able to see all orders placed via the orders component.
12. The manager can view stock count and which menus appear on the menu as part of the menu and inventory control components.
13. The manager will be able to edit stock count as part of the inventory control component
14. The manager will be able to view all relevant statistics as part of the data log component
15. The manager and chef will receive signals when a dish cannot be made due to low stock via the communication component.
16. The chef will be able to summon a waiter/manager using the communication component.
17. The stakeholders can view all data in the data log component.
18. Users will be able to create accounts, stored in the data log component.
19. Users will only have specified permissions (permissions stored in the data log component).
20. Users can make adjustments to their settings such as text size via the data log component.
21. Waiters, chefs, and managers can send and receive notifications as part of the communications component.

6.2: System Operation Contracts

| | |
|----------------|--|
| Operation: | Place Order |
| Use Case: | UC-1 |
| Precondition: | <ul style="list-style-type: none">• There is at least one item in the order• The user customer does not order something that is unavailable on the Menu |
| PostCondition: | <ul style="list-style-type: none">• The order is sent to Order Queue• The customer is presented the Order Progress tab |

| | |
|----------------|---|
| Operation: | Edit Order before Placement |
| Use Case: | UC-2 |
| Precondition: | <ul style="list-style-type: none">• There is at least one item in the order to remove• There is at least one item whose quantity can be adjusted• There is at least one item where notes can be added |
| PostCondition: | <ul style="list-style-type: none">• The order is updated in the cart• The updated order is visible to the end user |

| | |
|----------------|---|
| Operation: | View Available Items and Popular Dishes |
| Use Case: | UC-10 |
| Precondition: | <ul style="list-style-type: none">• There is at least one item available• There is at least one item that is specified as a popular dish |
| PostCondition: | <ul style="list-style-type: none">• The menu items are visible to the end user• The popular items are visible to the end user |

| | |
|---------------|---|
| Operation: | Create and Log into Account |
| Use Case: | UC-18 |
| Precondition: | <ul style="list-style-type: none">• If logging in, there must be at least one account belonging to the user• If signing up, there must be one valid unused email address to sign |

| | |
|----------------|---|
| | up with. |
| PostCondition: | <ul style="list-style-type: none"> • The user is able to log in and/or create an account |

| | |
|----------------|--|
| Operation: | View only accessible screens |
| Use Case: | UC-19 |
| Precondition: | <ul style="list-style-type: none"> • There is at least one permission level for users • At least one site has a permission accessible for users to see |
| PostCondition: | <ul style="list-style-type: none"> • The user will be able to see all sites they have permission for. |

6.3: Data Model and Persistent Data Storage

RoboRamsay will utilize Firestore, a non-relational document database which stores data in JSON format on the Google Cloud. This is a single database, containing several tables (called collections) and entries (called documents which are analogous to JS objects). We use the provided Firestore methods to add, modify, delete, and search for data. Note there is no accounts table, as user authentication and sessions are handled by the Firebase Authentication service.

Below there are 4 tables:

- Items: stores all menu items
- Users: stores relevant user data for constructing the UI
- OrderQueue: table accessible only by staff, stores all orders made by all signed-in users
- Inventory: stores ingredients

items:

```
uid: {
  itemAvailability,
  itemDes,
  itemDesMini,
  itemImage,
  itemMetaData: [{
    contains: [{
      uid,
      name
    }],
    type,
  }],
  itemName,
  itemPrepTime,
  itemPrice
}
```

users: [

```
uid: {
  currentOrders: [{
    eta,
    key,
    name,
    price,
    quant
  }],
  role
}
```

]

orderQueue: [

```
uid: {
  eta,
  Key,
  name,
  quant,
  status
}
```



```
]
inventory: [
  uid: {
    sku,
    name,
    price,
    quantity,
    unit,
    stopLoss,
  }
]
```

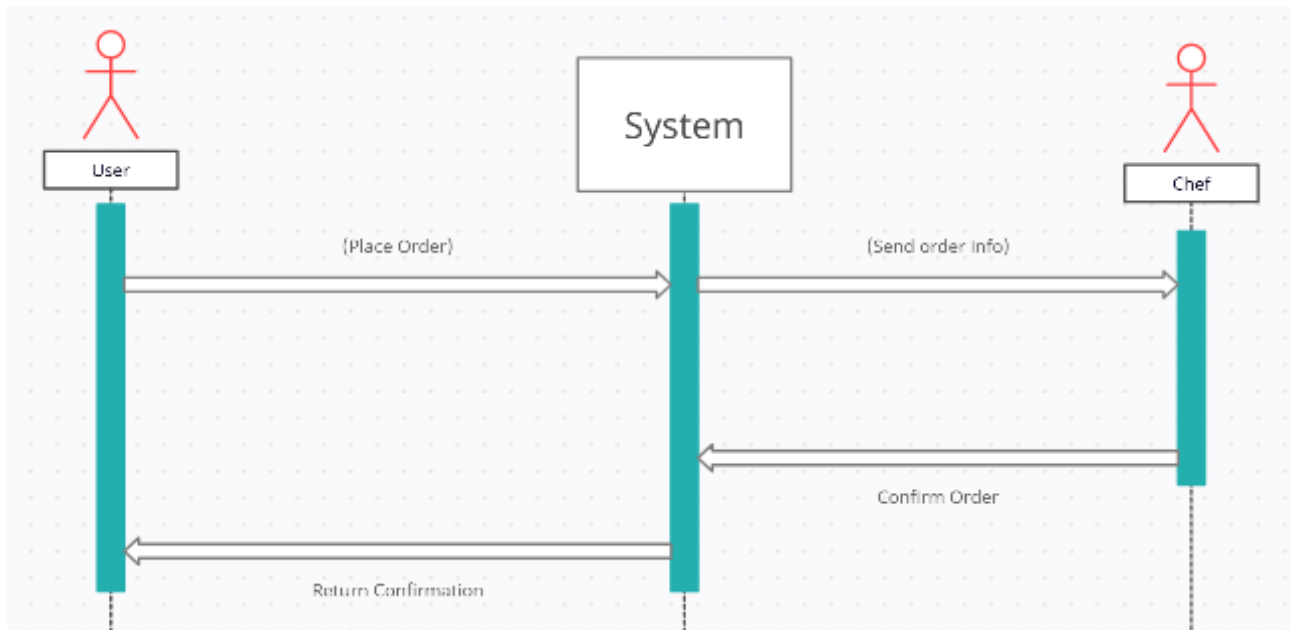
6.4: Mathematical Model

We will use a deterministic model to aid in determining the popularity of dishes to feature in the “Popular” category of the smart menu to fulfill REQ-18. It will consider each dish and the number of times they have been ordered within a given time period, usually a week of time. This needs info such as the number of orders per menu item. We can either use the raw number of dish orders or use the total number of orders to gain the percentage or frequency of the ordering rate of a dish in order to determine the popularity of dishes.

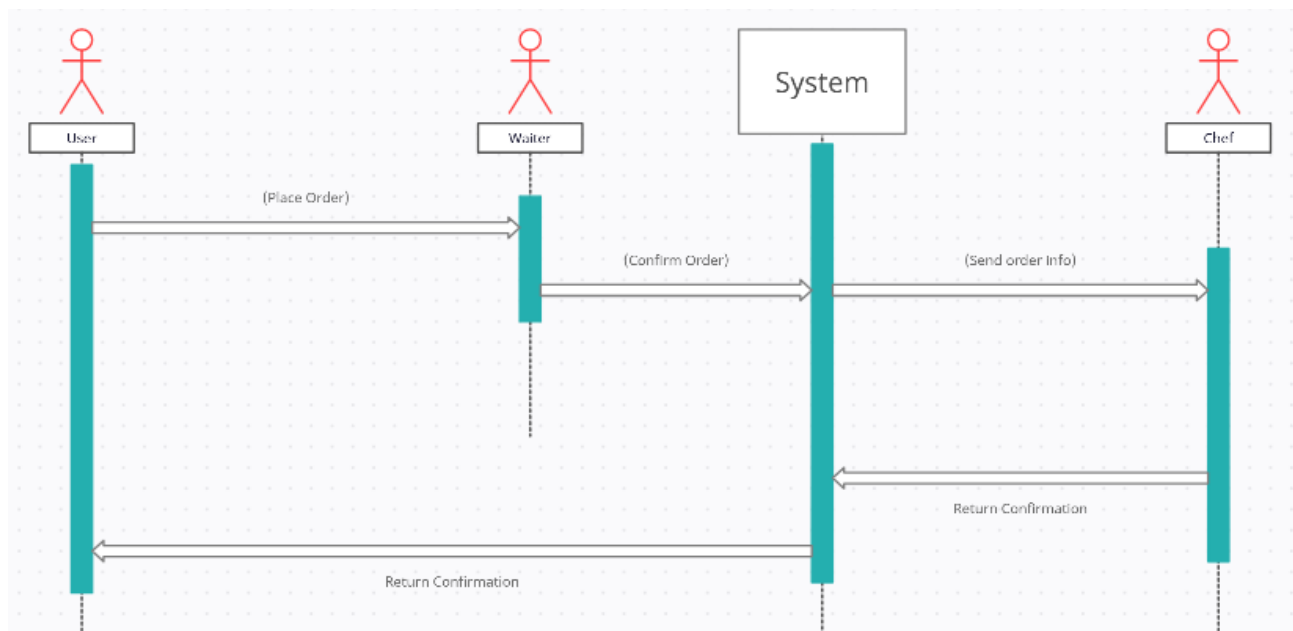
Section 7: Interaction Diagrams

UC-1: Place Order

Customer Orders Through Own Device

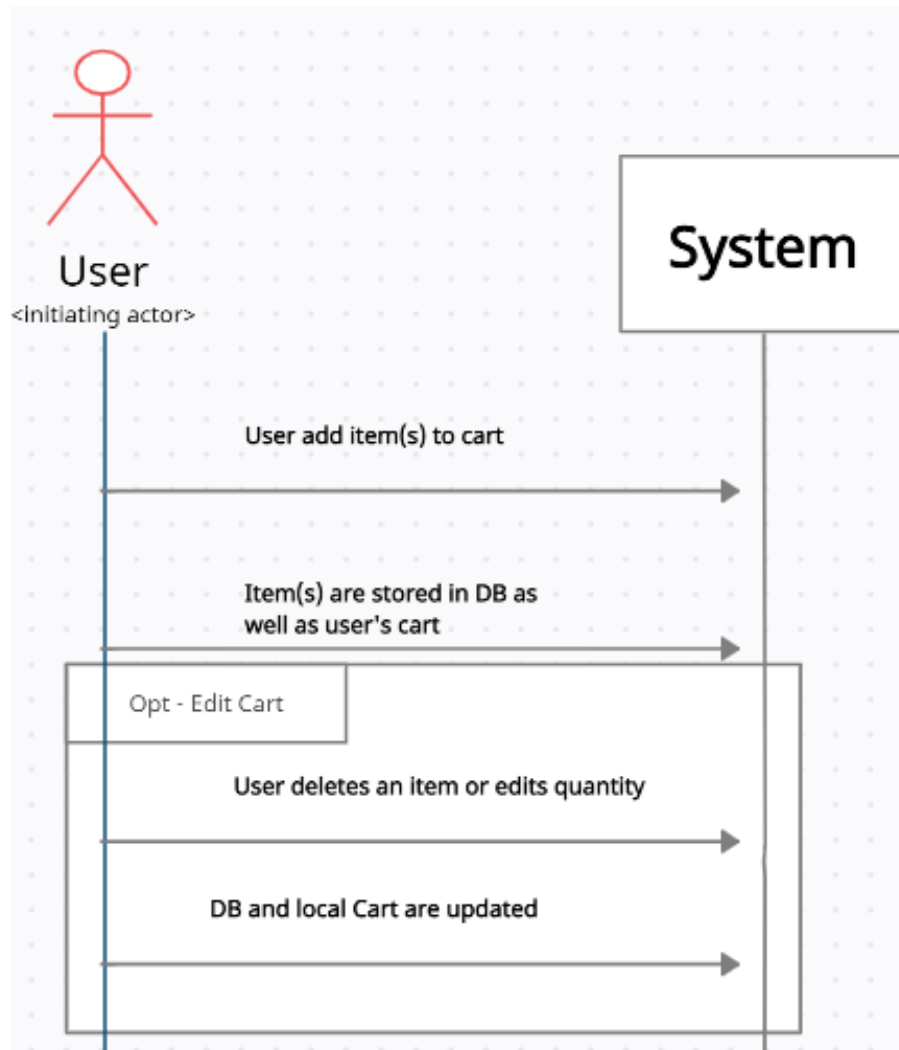


Order Through Waiter



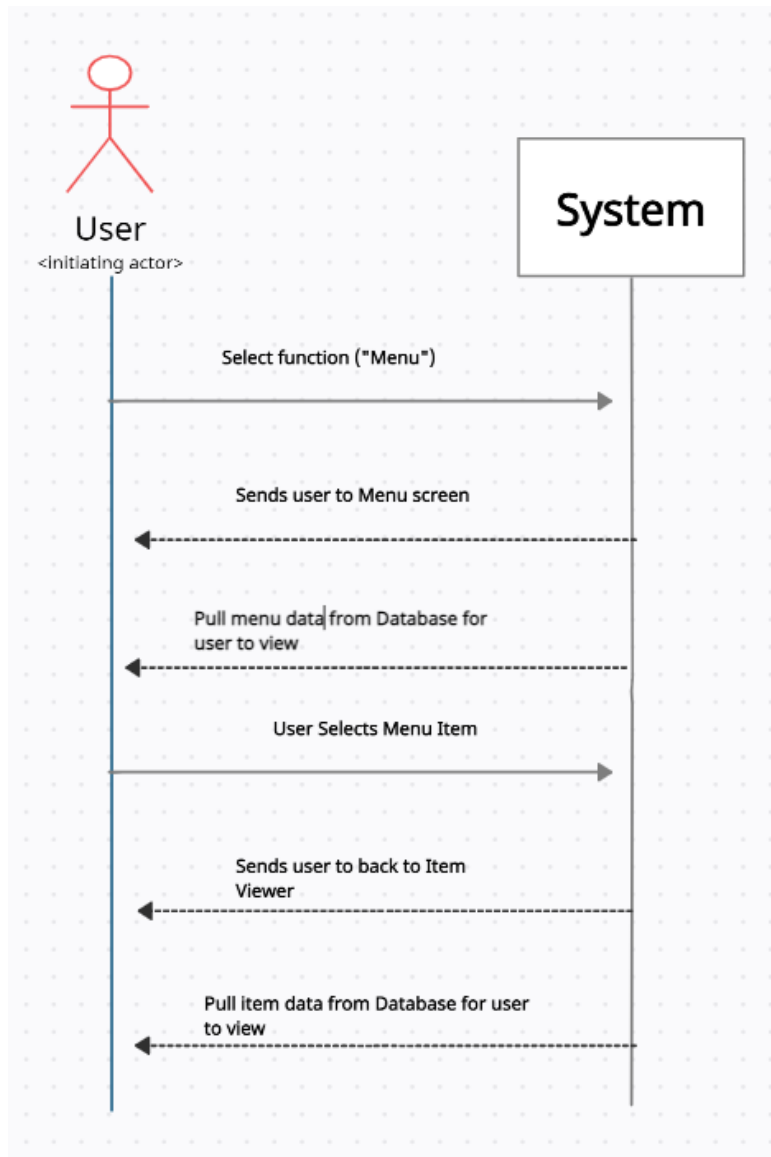
The above sequence diagrams display some of the potential paths that could be taken when placing an order with Robo Ramsey Software. The first diagram shows the user placing the order on their device using our app. The second diagram shows the waiter placing the order on behalf of the customer if it was requested by the customer. The customer would tell the waiter what items they would like to order, and the waiter would place the order through their device if the customer prefers that method. A customer may prefer the second method if they do not have a device or if they prefer a traditional dining experience. The expert doer was most used for the interfaces and the databases because these objects are the only ones that can successfully perform the tasks that they are undergoing. The interface must serve as a bridge between the user and the software, each class is designed to perform their specific tasks. Also a cohesion principle is adopted so that orders' object can be viewed on different interfaces for, waiter, chef, and customer.

UC-2: Shopping Cart



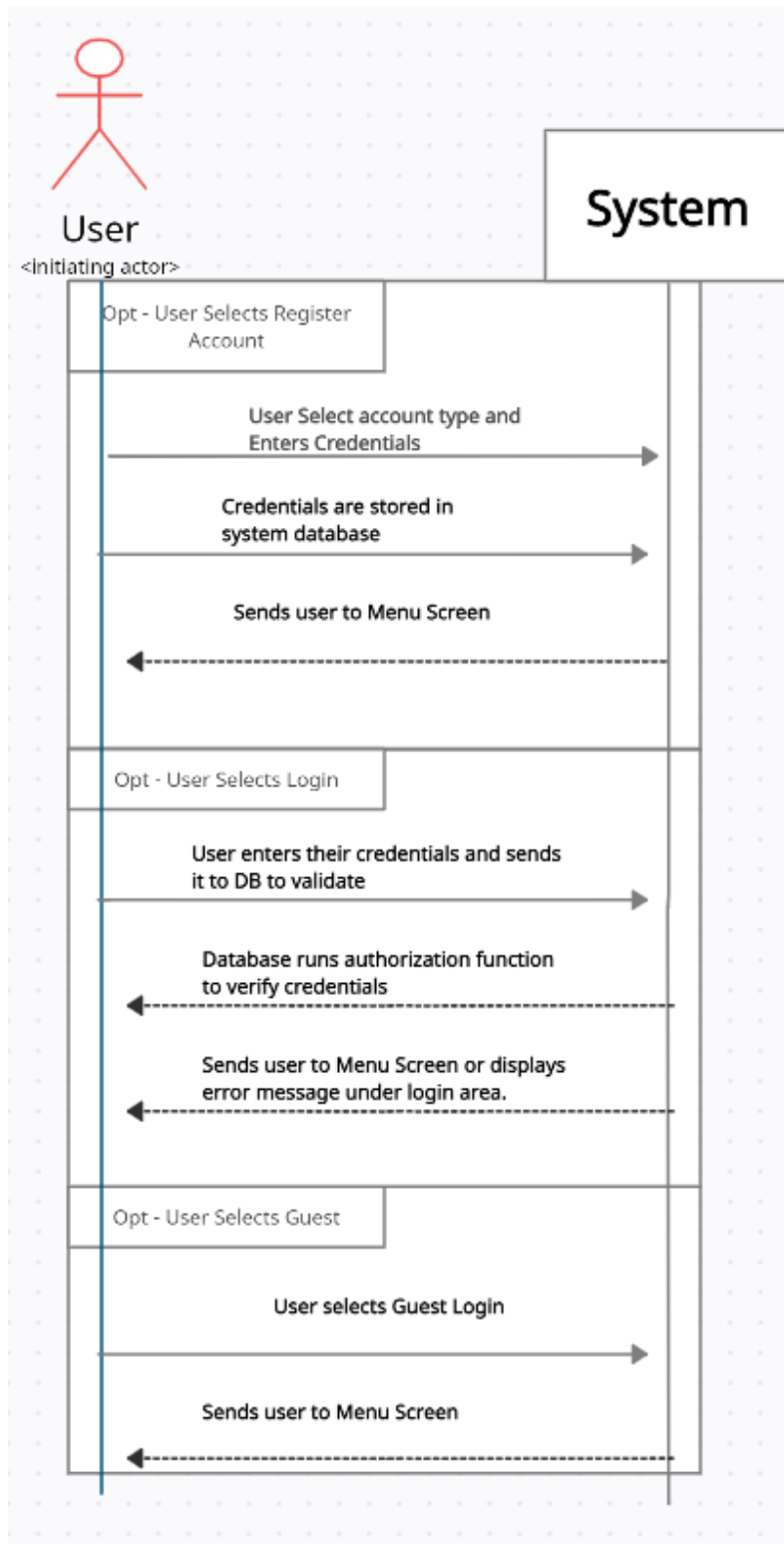
The above sequence diagram are the tasks required for the user to successfully add and edit the items in their cart. After the user has navigated to the Menu interface and has selected an item from the menu, they will be taken to the item viewer screen, then they will be able to add however many items they want to their cart. Once they confirm the items their cart is stored on the system's database. To edit the amount or delete items the user can swipe or tap the subtract or add icon to remove or add items in the cart, which will update the database.

UC-10: Smart Menu and Item Viewer



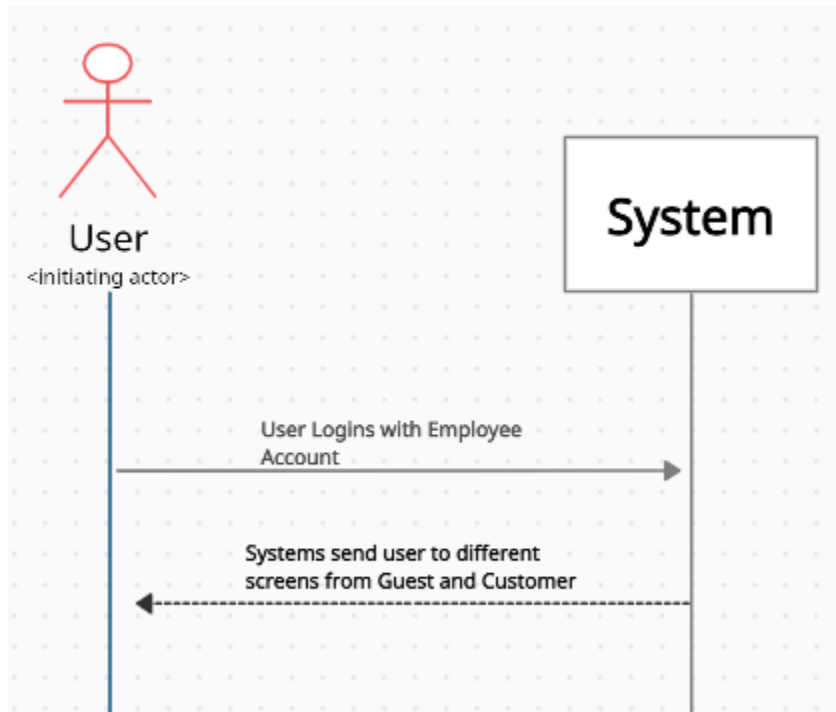
The above sequence diagrams are the simple tasks required to view a smart menu and item viewer. When the user selects the menu tab on the bottom of their screen, and our system takes them to the menu screen, which pulls the information of the menu items from the database, including when the user selects an item, that item information is pulled from the database to present on the user interface.

UC-18: Register & Login



The above sequence diagram shows how our system handles registering and authentication users who want to login on our application. When a user wants to register, they can choose between customer, manager, waiter or chef. When the user provides the adequate information, our system will store the information on our database with a label pertaining to the account type the user registered with. When a user wishes to login, they can simply enter their email address and password they used to register and they will be brought to the menu screen no matter what account type they logged in with. The way our system verifies the login information is that it will first verify if the email address exists in our database, they verify if the password entered matches the password store in the database linked to the email. The appropriate error message will be displayed if either the email does not exist or the password is incorrect. Selecting guest users simply directs the user to the menu screen as a customer.

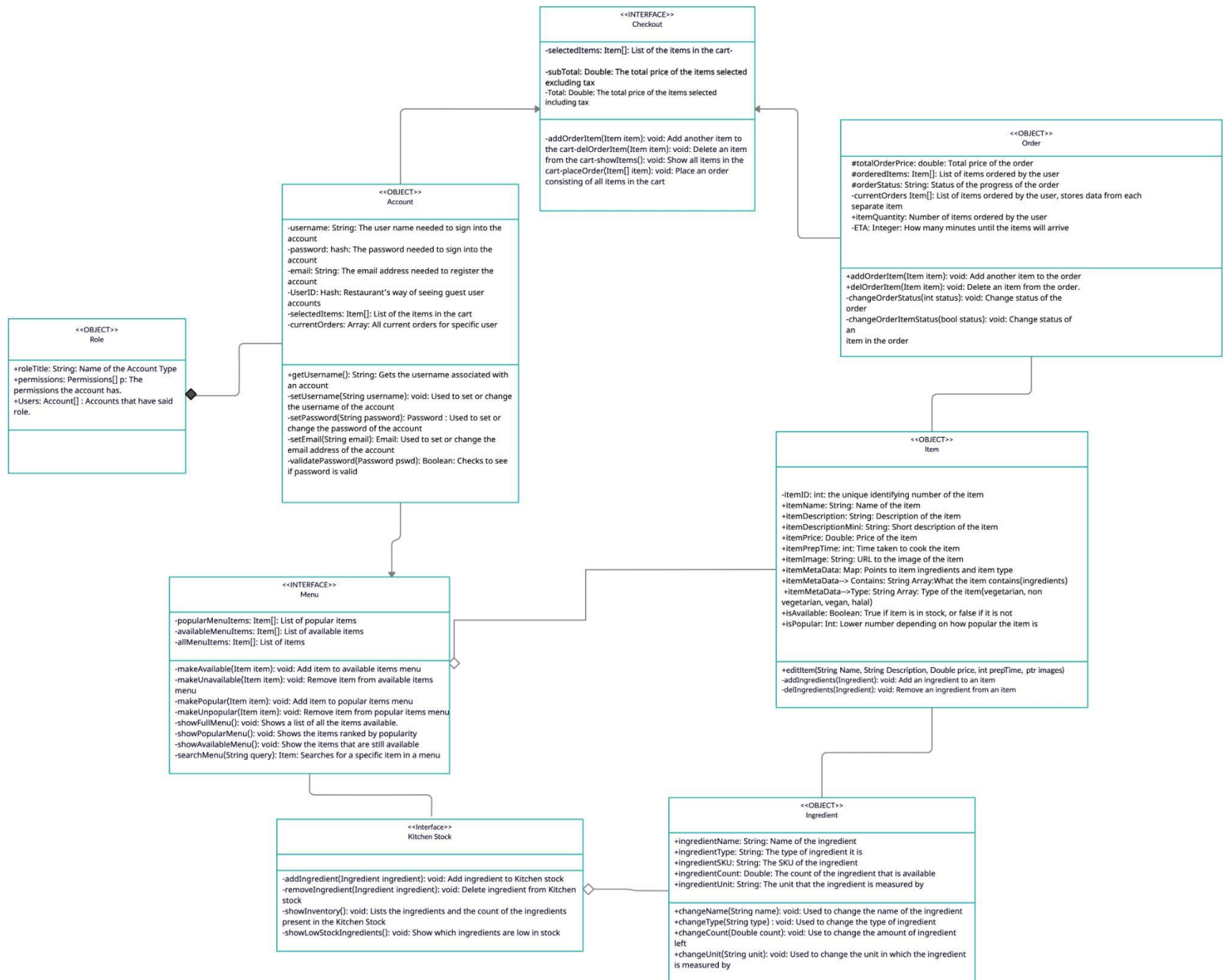
UC-19: Register & Login



When a user is logged in as a manager, waiter, or chef, our system returns additional interface components that will be visible and functional depending on the user's authentication. Some of these components can direct the user to others screens not viewable by guest users. The rationale for this is that it takes less storage on our remote server if we use components rather than creating new screens for the different users.

Section 8: Class Diagram and Interface Specification

8.1: Class Diagram - see attached pdf for clearer image



8.2: Data Types and Operation Signatures

(Interface) Menu: Shows, lists and edits list of menu items

1. -popularMenuItems: Item[]: List of popular items
2. -availableMenuItems: Item[]: List of available items
3. -allMenuItems: Item[]: List of items
4. -makeAvailable(Item item): void: Add item to available items menu
5. -makeUnavailable(Item item): void: Remove item from available items menu
6. -makePopular(Item item): void: Add item to popular items menu
7. -makeUnpopular(Item item): void: Remove item from popular items menu
8. -showFullMenu(): void: Shows a list of all the items available
9. -showPopularMenu(): void: Shows the items ranked by popularity
10. -showAvailableMenu(): void: Show the items that are still available
11. -searchMenu(String query): Item: Searches for a specific item in a menu

(Interface) Checkout: Lists selected items and enables placing orders

1. -selectedItems: Item[]: List of the items in the cart
2. -subTotal: Double: The total price of the items selected excluding tax
3. -Total: Double: The total price of the items selected including tax
4. -addOrderItem(Item item): void: Add another item to the cart
5. -delOrderItem(Item item): void: Delete an item from the cart
6. -showItems(): void: Show all items in the cart
7. -placeOrder(Item[] item): void: Place an order consisting of all items in the cart

(Object) Order: Stores and tracks an order

1. -totalOrderPrice: Double: Total price of the order
2. -currentItems[]: List of items ordered by the user, stores data from each separate item
3. +itemQuantity: Number of items ordered by the user
4. -ETA: Integer: How many minutes until the items will arrive
5. -orderStatus: String: Status of the progress of the order
6. -addOrderItem(Item item): void: Add another item to the order
7. -delOrderItem(Item item): void: Delete an item from the order
8. -changeOrderStatus(int status): void: Change status of the order
9. -changeOrderItemStatus(bool status): void: Change status of an item in the order

(Interface) Kitchen Stock: Records and tracks inventory of ingredients

1. -addIngredient(Ingredient ingredient): void: Add ingredient to Kitchen stock
2. -removeIngredient(Ingredient ingredient): void: Delete ingredient from Kitchen stock
3. -showInventory(): void: Lists the ingredients and the count of the ingredients present in the Kitchen Stock

4. -showLowStockIngredients(): void: Show which ingredients are low in stock

(Object) Item: Stores data about a dish

1. -itemID: int: the unique identifying number of the item
2. +itemName: String: Name of the item
3. +itemDescription: String: Description of the item
4. +itemDescriptionMini: String: Short description of the item
5. +itemPrice: Double: Price of the item
6. +itemPrepTime: int: Time taken to cook the item
7. +itemImage: String: URL to the image of the item
8. +itemMetaData: Map: Points to item ingredients and item type
9. +itemMetaData--> Contains: String Array: What the item contains(ingredients)
10. +itemMetaData-->Type: String Array: Type of the item(vegetarian, non vegetarian, vegan, halal)
11. +isAvailable: Boolean: True if item is in stock, or false if it is not
12. +isPopular: Int: Lower number depending on how popular the item is
13. +editItem(String Name, String Description, Double price, int prepTime, ptr images)
14. -addIngredients(Ingredient): void: Add an ingredient to an item
15. -delIngredients(Ingredient): void: Remove an ingredient from an item

(Object) Ingredient: Stores data about an ingredient

1. +ingredientName: String: Name of the ingredient
2. +ingredientType: String: The type of ingredient it is
3. +ingredientSKU: String: The SKU of the ingredient
4. +ingredientCount: Double: The count of the ingredient that is available
5. +ingredientUnit: String: The unit that the ingredient is measured by
6. +changeName(String name): void: Used to change the name of the ingredient
7. +changeType(String type) : void: Used to change the type of ingredient
8. +changeCount(Double count): void: Use to change the amount of ingredient left
9. +changeUnit(String unit): void: Used to change the unit in which the ingredient is measured by

(Object) Account: Stores data about a user and facilitates logging in

1. -username: String: The user name needed to sign into the account
2. -password: hash: The password needed to sign into the account
3. -email: String: The email address needed to register the account
4. -UserID: Hash: Restaurant's way of seeing guest user accounts
5. -selectedItems: Item[]: List of the items in the cart
6. -currentOrders: Array: All current orders for specific user
7. +getUsername(): String: Gets the username associated with an account

8. -setUsername(String username): void: Used to set or change the username of the account
9. -setPassword(String password): Password : Used to set or change the password of the account
10. -setEmail(String email): Email: Used to set or change the email address of the account
11. -validatePassword(Password pswd): Boolean: Checks to see if password is valid

(Object) Role: Stores and tracks permission data

1. +roleTitle: String: Name of the Account Type
2. +permissions: Permissions[] p: The permissions the account has
3. +Users: Account[] : Accounts that have said role

8.3: Traceability Matrix

| Domain Concepts | Software Classes | | | | | | | |
|-------------------|------------------|------|----------|-------|---------------|------------|---------|------|
| | Menu | Item | Checkout | Order | Kitchen Stock | Ingredient | Account | Role |
| Order | | | X | X | X | | X | X |
| Communication | | | | | | | X | X |
| Inventory Control | | | | | X | X | | X |
| Menu | X | X | | | | | | X |
| Data Log | X | | | X | | | | X |

Order:

- Checkout: The customer can use the checkout after placing the order
- Kitchen Stock: Placing an order will update the kitchen stock according to the items ordered and the quantity of each
- Accounts: Customers can place and view the status of orders, waiters can view the status of orders, managers and chefs can view and update order status
- Order: Lists the price of the order, as well as quantity and names of items being ordered
- Role: Customers and waiters have the ability to place an order

Communication:

- Accounts: The manager, chef, and waiter can communicate with each other through their accounts. Customers can request a waiter to come to their table
- Role: Each user has the ability to communicate with each other except the customer

Inventory Control:

- Kitchen Stock: Lists the current number of ingredients in stock, as well as a target for how many of each is needed
- Ingredient: Name of ingredients are listed
- Role: Chefs and managers can update the inventory of ingredients, customers and waiters do not have access

Menu:

- Menu: Shows items that the restaurant provides
- Item: A singular item within the menu, multiple will be listed
- Role: Chefs, managers, waiters, and customers can all see the menu. Manager will be able to manually edit the menu

Data Log:

- Menu: Menu will feature popular items based on order frequency
- Order: Placing an order will adjust the order frequency and by extension, popularity
- Role: Customers and waiters place orders which affects order frequency and popularity. Manager will be able to view the statistics from the data log

8.4: Design Patterns

Builder Pattern: Useful for the creation of menu items and ingredients separately, then build an object from the menu item information and ingredients. This pattern is only relevant to UC-12 and UC-13 which refer to the creation of the ingredients and menu items. The advantages of this approach are the versatility of representing the objects created on various interfaces, for example, the order dashboard and menu item edit screen, which would contain the same objects, but display different information from that object.

Singleton Pattern: There may only be one Menu and only one Inventory (aka Kitchen Stock) object. More cannot be created.

Decorator Pattern: Roles are attached to Accounts in the form of a Decorator.

Front Controller Pattern: App.js is a front controller. It answers all requests.

Module Pattern: We group everything into modules, and put modules into modules. views contains user and auth, and also the submodules main and accounts.

Publish/subscribe: Whenever the Menu is updated, currently running client side applications are notified. When there is a change in the shopping cart or in an order, the relevant clients are notified. When there is a change in the availability of a Menu Item due to Ingredient quantity, the Menu Item is so updated.

Event-Based Asynchronous: We have many parts of our application logic work as asynchronous functions with callback.

8.5: Object Constraint Language (OCL) Contracts

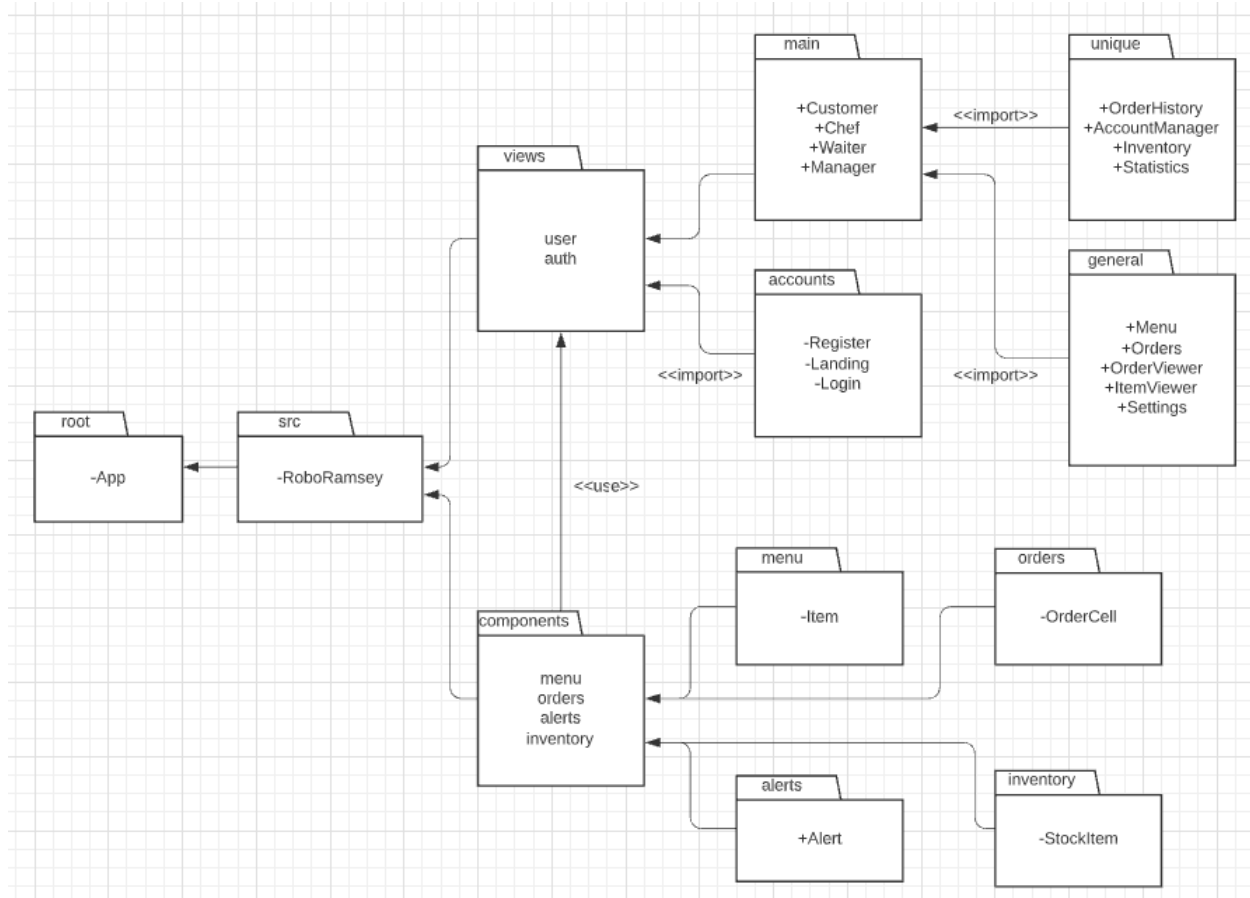
```
context Menu inv
    Menu = Item[]
context Menu::popularItems(Item[]):Item[]
    Pre: Item.itemPopularity > 5
    Post: Item[]
context Menu::availableMenuItems(Item[]):Item[]
    Pre: Item.itemAvailability = true
    Post: Item[]
context Menu::allMenuItems():Item[]
    Post: Item[]
context Menu::makeAvailable(Item)
    Pre: Item.itemAvailability = false
    Post: Item.itemAvailability = true
context Checkout inv
    Items[] > 1
context Checkout::totalPrice(Items[]):Integer
    Pre: Items[]
    Post: total = sum(Item.itemPrice)
context Checkout::addOrderItem(Item):Void
    Pre: Item
context Checkout::deleteOrderItem(Item):Void
    Pre: Item

context Order inv
    Orders[] > 1
context Order::totalOrderPrice(Items[]):Integer
    Pre: Items[]
    Post: total = sum(Item.itemPrice)
context Order::addOrderItem(Item):Void
    Pre: Item
context Order::deleteOrderItem(Item):Void
    Pre: Item

context KitchenStock inv
    Stock > 1
context KitchenStock::addIngredient(Ingredient):Void
    Pre: Ingredient
context KitchenStock::removeIngredient(Ingredient):Void
    Pre: Ingredient
```


Section 9: System Architecture and System Design

9.1: Identifying Subsystems



There are 2 major subsystems:

User: Contains the software (called controllers) that prepare the frontend and backend for each of the 4 users.

Auth: Contains the software (called controllers) that handle user authentication.

The other folders simply pertain to app organization, with:

Root: Stores the main app controller, which handles loading the entire page.

Src: Stores the file which decides which of the app's UI (pages and page components) to load.

Views: Stores all pages in the app. Some can be accessed by all users, whereas others only load for users that are allowed to access the page..

Components: Stores page components, such as the navigation bars, buttons, forms, etc.

9.2: Architecture Styles

We chose to use monolithic architecture, as it synergizes with the development lifecycle used when utilizing Expo and React Native. This means we only need to write one app and we can build three deployable versions across three platforms, web, iOS, and Android.

9.3: Mapping Subsystems to Hardware

Our application will be cross platform and run on mobile or web. Our users may run the app on a mobile web browser or a pc web browser. The server side of our software is handled by Firebase, which means it will be stored on some server owned by the Google Cloud Platform. The location and size of the app is dependent on the workload (i.e. how many orders/customers/users will the software have to track), but they are easily changeable. For the client, it shouldn't take any space and only require an internet connection.

9.4: Connectors and Network Protocols

Our network is handled by Google Cloud service, which likely uses the WebSockets to communicate between devices on its WAN. Our application itself doesn't require server-side code, as we simply use the Firebase API to communicate with its backend service. The API requests themselves are sent using HTTP.

9.5: Global Control Flow

i. Execution orderliness

The application is structured such that there are linear segments that can be initiated in an event-driven manner. For example, for a customer, ordering food is relatively linear, although there will be the option to leave the ordering segment entirely. Logging in is a linear process that can be initiated by an event.

ii. Time dependency

The system is event-driven. All events draw on the real-time clock for logging purposes. Also, all events that have a timeout or a timer measure it in real time. Timeouts are to be user-defined.

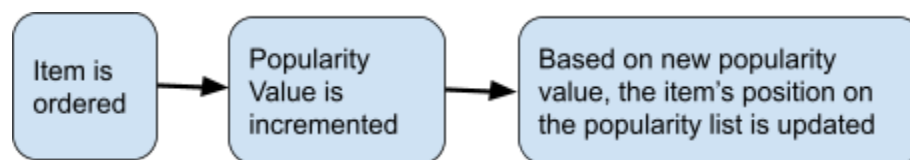
9.6: Hardware Requirements

With the system being software-only, there is no strict need for specific hardware. The software will have to be able to run on mobile devices that connect to the internet in some form, with a minimum required 1 MBps connection speed in order to function. It needs to be able to work on devices as old as the iPhone 6 or the Samsung Galaxy S7, meaning there will be support for the current version of iOS and Android. Devices will need a functioning touchscreen to provide the needed input, as well as a minimum screen resolution of 750 x 1334 pixels. In order to be installed, there must be at least 40 MB of device storage remaining. The device should have a minimum clock speed of 1 GHz. For the server, it requires at minimum 2.4 GHz clock speed as well as memory space capacity of 0.5 GiB.

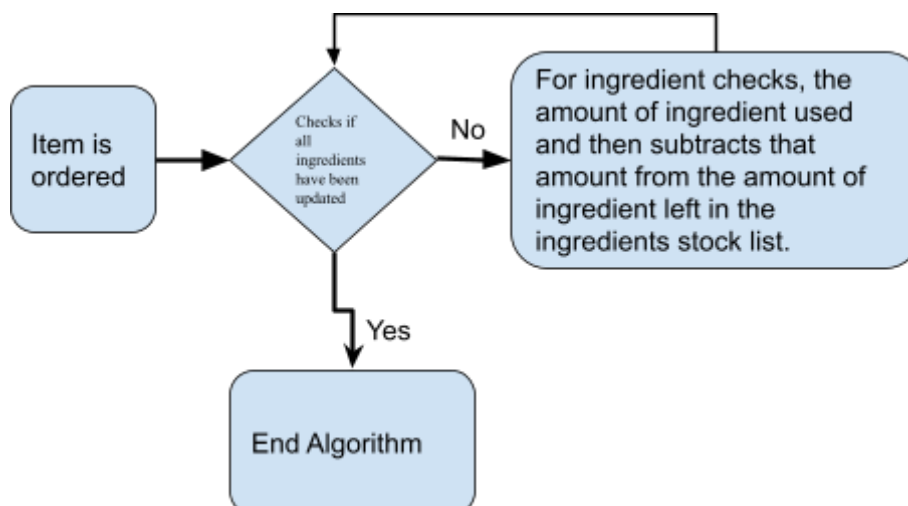
Section 10: Algorithms and Data Structures

10.1: Algorithms

- 1) For the popularity of the items on the menu, each item will contain a popularity value to represent the number of times it has been ordered by the customer during a certain time period. So initially the popularity value will be set to 0, and then when someone orders the item, the popularity value will increment by 1. Then in order to access popularity, the items will be arranged descendingly from the ones with the greatest popularity value to the least popularity value.



- 2) The amount the user is charged is calculated by summing all the prices of the items they have ordered and then multiplying it with the tax rate in order to get the total price they are charged with.
- 3) When an item is ordered, we go through the ingredients it needs in order to prepare it. For each ingredient, we check how much is needed in order to make the item and then subtract that amount from the total ingredient stock in order to update the ingredient stock list.



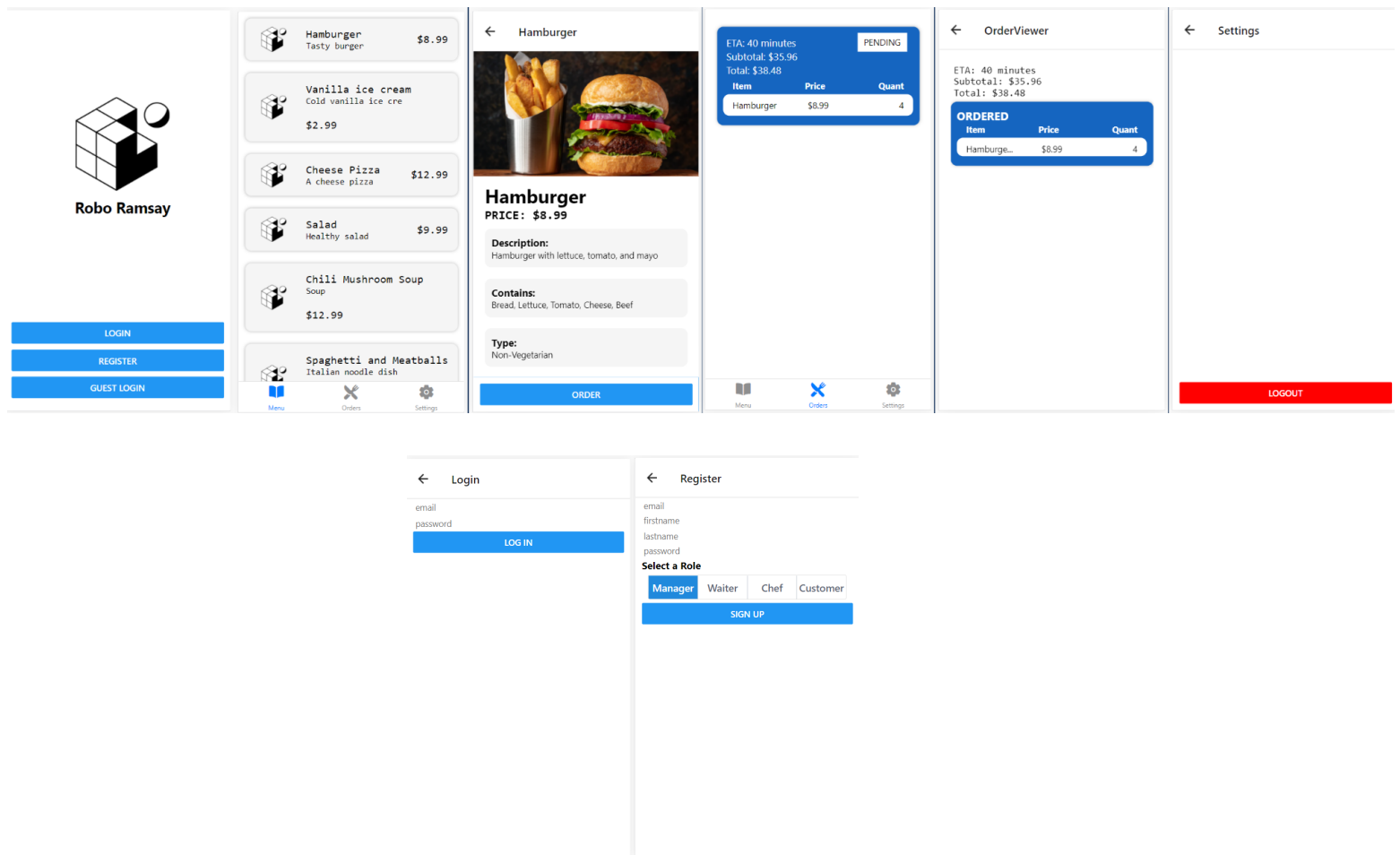
- 4) To calculate the preparation time for a meal, we first check the amount of chefs that are active at a given time. Each chef has a stack that is linked to them and a variable which tracks the amount of time it will take for them to prepare their assigned items. When an

order is placed, the items ordered are arranged from greatest to least. Then we check the time of the items already being prepared by the chefs. We then assign the item from the order to the chef who has the least time assigned to him. We repeat this process with the other items in the order until we are done assigning all the items between the various chefs. Then we check the time between the first assigned item and the last assigned item and use that to calculate the preparation time for an order.

10.2: Data Structures

- 1) We will be using a NoSQL database for storing our data including our orders, ingredient stock list as well as menu items. We chose this data structure as it gives us a better performance than a SQL database as all the data is stored in one database in NoSQL compared to SQL where data is stored in multiple tables. Specifically, we will be using Firebase as it is the easiest and the most cost effective NoSQL databases that are available.
- 2) We will be using a Stack data structure in order to assign items to various chefs.

Section 11: User Interface Design and Implementation



When the customer opens up the app on their system, the intro screen is displayed with the options to log-in using an email and password, register as a new user with email, first name, last name, password, and role. They can also select Guest Login where they can view the menu and order items without having to register or login. Upon selecting Guest Login, they are greeted to the scrollable menu that features every menu item, with a small description and price of the item. If the customer selects any of the menu items, they will be brought to said item's page where it features all details of the item, from ingredients to type of dish, be it vegetarian or non-vegetarian, to estimated prep time. They can then select to order the item and specify the amount that they want. Upon clicking the back button, the customer will be brought back to the menu screen where they can press on the Orders tab that shows a summary of their order, where they can press on their order and provide changes, by swiping to delete the order of an item. There is also a Settings page where relevant options will be shown, as well as the option to logout of the app.

For the other currently unimplemented perspectives, they will follow details in Section 3.4: User Interface Requirements but adjusted to the style of the app as it is currently. For the manager/owner's perspective, we will use manager/owner perspective, users have access to several buttons which when pressed, will display statistics based on the specified button. They will also have the ability to edit the menu and update quantity in less than 3 clicks plus minimal keystrokes (depending on if they need to search for an item). From the chef's perspective, they will access almost all of the information made available to the others and the UI will be implemented to ensure that access to the application would not interfere with doing their own job.

Section 12: Design of Tests

12.1: Design of Tests

| Test Case Identifier | Use Case Tested | Test Procedure | Pass/Fail Criteria |
|----------------------|---------------------|---|--|
| TC - 1 | UC - 1, UC - 2 | The customer user will select items from the menu and place an order. After placing the order, they should be able to check on the progress of his order. | If the user is able to successfully place an order and check his status without any issues, then the test will pass else it would fail. |
| TC - 2 | UC - 8 | The customer user should be able to request a waiter. | If the user is able to successfully request a waiter, then the test is a pass else it would be a fail. |
| TC - 3 | UC - 10 | The user should be able to see all the items on the menu. | If the user is able to view all items on the menu after logging in, then the test is a pass else it would be a fail. |
| TC - 4 | UC - 9 | The manager user should be able to view and edit all the orders placed and all the archived orders as well. | If the manager user is able to view the current and archived orders without any issues, then the test is a pass else it would be a fail. |
| TC - 5 | UC - 18 | The user should be able to create an account. | If the user is able to create an account without any issue, the test is a pass else it would be a fail. |
| TC - 6 | UC - 18 | The user should be able to log into his own account. | If the user is able to log into their pre-existing account without any issue, the test is a pass else it would be a fail. |
| TC - 7 | UC - 18, UC - 19 | The user should be able to view the menu using a guest login. | If the user is able to view the menu using a guest login without any issue, the test is a pass else it would |

| | | | |
|--------|---------|--|---|
| | | | be a fail. |
| TC - 8 | UC - 19 | The user should be able to access screens that they have permission to. | If the user is able to view the screens they have permission to after logging in without any issue, the test is a pass else it would be a fail. |
| TC - 9 | UC - 20 | The user should not be able to see screens they aren't permitted to see. | If the user is able to view the screens they do not have permission to after logging in, the test is a fail else it would be a pass |

12.2: Test Coverage

All of our test cases cover the essential Use Cases that are necessary to the operation of RoboRamsey. The test procedures will be done and it will be seen whether or not they function in accordance to what we were trying to achieve as outlined in our use cases. Our test cases also cover all the types of users that will be using RoboRamsey such as the customers, the managers, the waiters and the chef.

For the customer, we are testing whether or not they are able to view the menu as they desire and then successfully order and then be able to track the progress of their order as well as be able to request for a waiter if they so desire. Also, for the customer, we are making sure that they have no problems creating an account as well as no problems when they log into their account. We also are checking whether or not our customers face any problem when they log into their account and whether they are able to access all the screens they have permission for.

For the managers, waiters and chefs, we are also checking whether they have no problems creating an account as well as no problems when they log into their account. As with the customers, we are also checking whether the staff are able to access all the screens they have permission for.

Lastly, we are also making sure that no one is able to accidentally or purposefully access screens that they do not have permission to access.

12.3: Integration Testing

The Integration Testing strategy that we are using is the Bottom-Up Integration Strategy.

Bottom-Up Integration strategy is an approach where the lower level components are tested first, and then those tested components are used to facilitate the testing of the higher level components. We continue the testing until we are done with testing all the top level components. This approach is extremely useful for identifying bugs and fault localization and it is why we are using this strategy. Since we start with the lower level components and then move on to the higher level components, if we are faced with a fault, it is easy for us to then figure out where exactly it is coming from and the root cause for the fault and thus makes it easier for us when it comes to debugging.

Section 13: History of Work, Current Status, and Future Work

History of Work

January 23 -January 28:

As a group, we listed all the eleven project ideas in a shared spreadsheet and indicated our individual interests in each idea, so that we can select a project to work on before we have to work on the proposal.

January 28 - February 2:

We narrowed down our choices to four, and then decided to work on Restaurant Automation based on project description and some work from previous semesters in the course. We also detailed which programming languages we were interested in which will help in distributing out work between everyone. After that, we worked on the proposal divided into three teams which were responsible for their own parts of the proposal.

February 3 - February 12:

We started working on the first part of Report 1, and incorporated feedback from the proposal such as details for “Dish analysis”, “Ingredient tracking”, hardware, etc. The preliminary GitHub page was also created at the end of this timeframe.

February 13 - February 28:

We worked on the second part of Report 1 during the beginning of this timeframe and decided to be more specific on our organization for how code should be handled. A shared spreadsheet was set up that detailed our project’s initial requirements for code, such as GUI, MenuItems, Accounts, Orders, Inventory. Each member indicated what they’d rather be assigned to. A README file was created (and put on GitHub for easy access) to describe the details of how to utilize GitHub; this was done because a number of members had no experience using Git before, be it on our own computers or via a remote repository. A Figma project was created and worked, which we will later use as a basis for the UI of our application.

Full Report 1 was due at the end of this period, and all members also worked on this. We included changes based on the feedback from the second part of Report 1 that clarifies certain use cases and adjusts the relevant scenarios and system sequence diagrams. Initially we had decided to separate responsibilities of code based on functional requirements for the section on Project Management.

March 1 - March 12:

Upon working on the first part of Report 2, we had decided to use four different databases for each separate aspect of our app, one for Inventory, Orders, Accounts, and Menu.

March 13 - March 26:

Organisation of the group for coding was changed to be more detailed where we split up into groups for Menu, Accounts, Orders, and Inventory, and each was divided into backend and frontend. A secondary repository was created for the UI of our code, which eventually became the main GitHub page for our project, as the frontend and backend were closely mixed. During spring break we started coding our application in preparation for Demo #1, and it was decided that the database we will use will be from Firebase, which has documentation that allows for easy access and edits to the databases involved. With the amount of code accomplished from the limited knowledge of most members along with other classwork, we decided to focus more on the application for the customer rather than the initial expectations of having versions for the chef, manager, and waiter of the restaurant. What we had at the time was individual components that would be combined into specific app pages, but we hadn't gotten that part of implementation yet.

Full Report 2 was also being finalized as it was due on the Tuesday after spring break ended. For Demo #1, everybody would record their parts separately based on what was coded for our application, and would be combined together in one video.

March 27 - April 23:

We attended the Q&A session held for our group between the Professor, TAs, and Graders, and got feedback where a payment feature would be a good feature to have for our app, and we decided to go over the Stripe API and see if we can implement it into our application. We also clarified that Firebase has the required functionality for what our application will be able to do.

Report 3 Part 1 was thrown together with fixes/adjustments to Sections 1 through 14, but excluding 8, 11, 12 for revisions.

April 24 - May 3:

We went through the Q&A where we were asked questions and clarified specific details of our project and discussed what we can potentially work on in the future in terms of implementation. The final report was also worked on after the Q&A session, where we adjusted multiple sections to align with what our code has for use cases and other information regarding the evolution of the project.

Current Status

Currently our application works fully for an unregistered customer (guest), where they can view the menu, details of specific items, and their orders, as well as add items to a shopping cart, and delete items from a shopping cart. We have login and register completed, however other related features are not fully functional yet. We weren't able to implement Stripe API within the timeframe between Demo #1 and Demo #2.

Future Work

- We need to implement Ordering.
- We need to implement Payments.
- We need to implement Notifications.
- We need to implement Roles, and role-based permission locked screens. (Roles will be attached to Accounts)
- We need to implement data collection, and then data analysis. This will be used to implement:
 - The manager's statistics screen
 - Customers' popular items section in item list
 - Optimised per-dish ETA calculation
 - Optimising Dishes' ingredient counts based on usage data
- We need to implement better authentication.
 - We need to research the viability and decide if we want to implement federation (i.e. One can log in with their google or facebook account to our application)

Section 14: References

Apple Inc. (2017, October 30). Ios device compatibility reference. Retrieved February 24, 2021, from <https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html>

Firebase. firebase.google.com/docs/database/rtdb-vs-firestore. Accessed 28 Mar. 2021.

Full Service Restaurant News. (2013, October 8). *Study Released on Average Restaurant Wait Times*. [fsrmagazine.com](https://www.fsrmagazine.com/content/study-released-average-restaurant-wait-times#:~:text=Restaurants%20certainly%20have%20plenty%20of,wait%20longer%20than%2040%20minutes). Retrieved February 9, 2021, from <https://www.fsrmagazine.com/content/study-released-average-restaurant-wait-times#:~:text=Restaurants%20certainly%20have%20plenty%20of,wait%20longer%20than%2040%20minutes>.

Green Restaurant Association. (n.d.). *Waste Reduction and Recycling*. [dinegreen.com](https://www.dinegreen.com/waste). <https://www.dinegreen.com/waste>