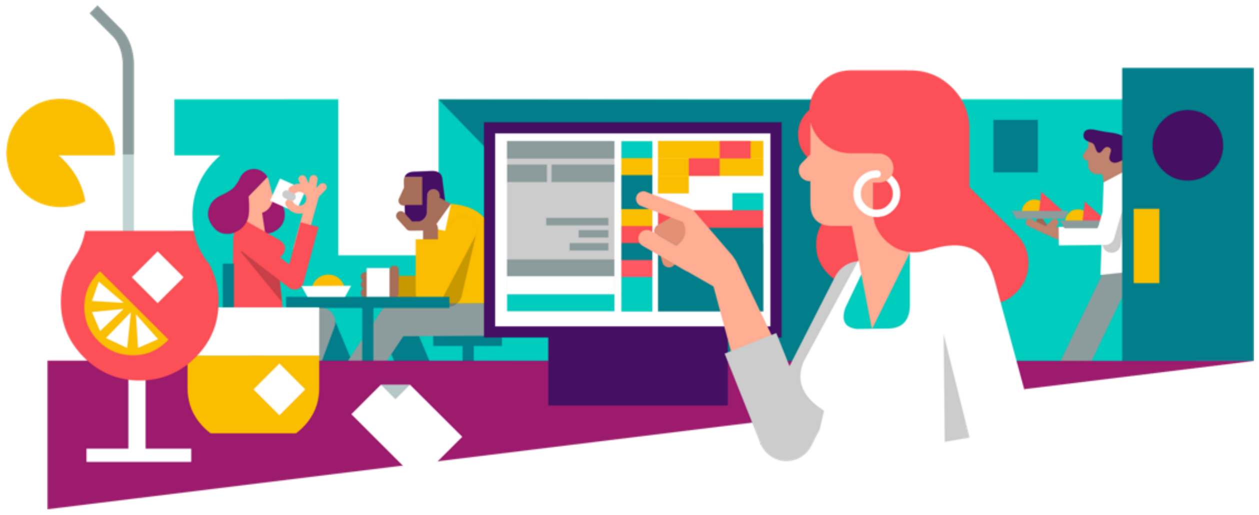


Robo Ramsay

(Name pending)

Software Engineering

—

Team 6
3/12/2021

Github repository:
https://github.com/Dpmon1/Robo_Ramsey

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
Conceptual Model	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
System Operation Contracts	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Data Model and Persistent Data Storage	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Mathematical Model	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Interaction Diagrams	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Class Diagram	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Data Types and Operation Signatures	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Traceability Matrix (Class Diagram and Interface Specification)	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Project Management	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%

Table of Contents

Section 1: Analysis and Domain Modeling

1.1: Conceptual Model

i. Concept Definitions:

Responsibility #1	Concept
R.1 Lists and facilitates ordering available dishes	Menu 1
R.2 Customer requests a waiter	Communication 1
R.3 Customer Interface to display current order status	Orders 1
R.4 Waiter interface to display all orders	Orders 2
R.5 Waiter interface to display customer's menu	Menu 2
R.6 Waiter can order for customer	Menu 2
R.7 Manager can view and edit the status of all orders	Orders 3
R.8 Chef can view and edit status of all orders	Orders 3
R.9 Estimate meal arrival time	Orders 1
R.10 Remove option to select unavailable items	Menu 1
R.11 Display list of popular dishes	Menu 1
R.12 Manager can edit menu	Menu 3
R.13 Manager can view data on popularity and trends	Data log 2
R.14 Manager will be alerted of unavailable dishes due to insufficient ingredients.	Inventory control 2
R.15 Manager will be alerted if ingredient is below a threshold	Inventory control 2
R.16 Manager can edit and view ingredient count	Inventory control 3
R.17 Chef can edit and view ingredient count	Inventory control 3
R.18 Allows editing of inventory of ingredients	Inventory control 3
R.19 Chef can request a specific waiter or manager	Communication 2
R.20 Application logs usage data	Data log 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:

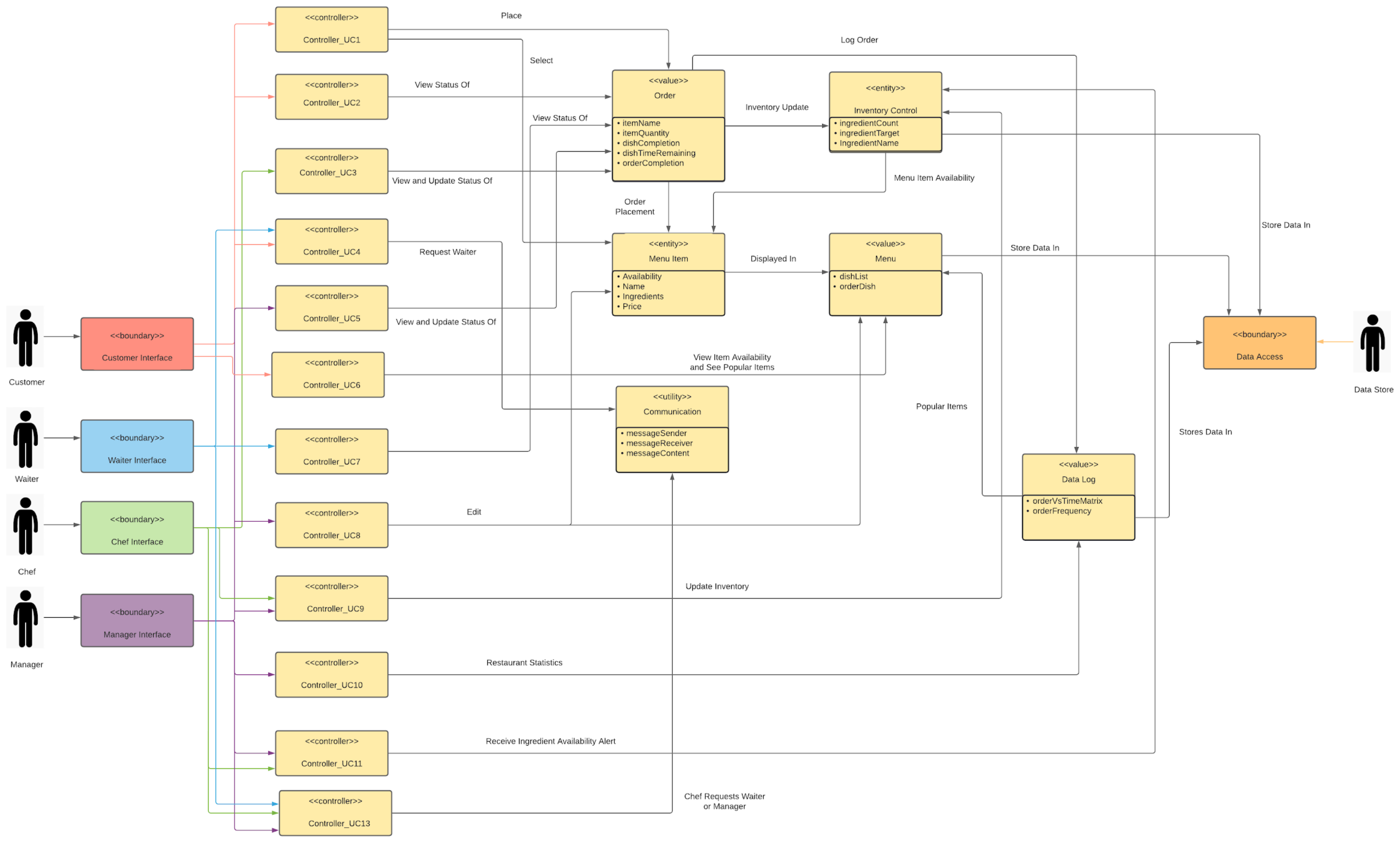
Concept	Attribute	Definition
Menu	dishList	Allows customers to view items on the menu
	orderDish	Allows customers to order dishes through the menu
Orders	dishCompletion	The status of the progress towards the dish's completion
	dishTimeRemaining	The expected time remaining for dish's completion
	orderCompletion	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
10			X		X	X
11				X		X
12				X		

V. See Attached PDF for clearer Image



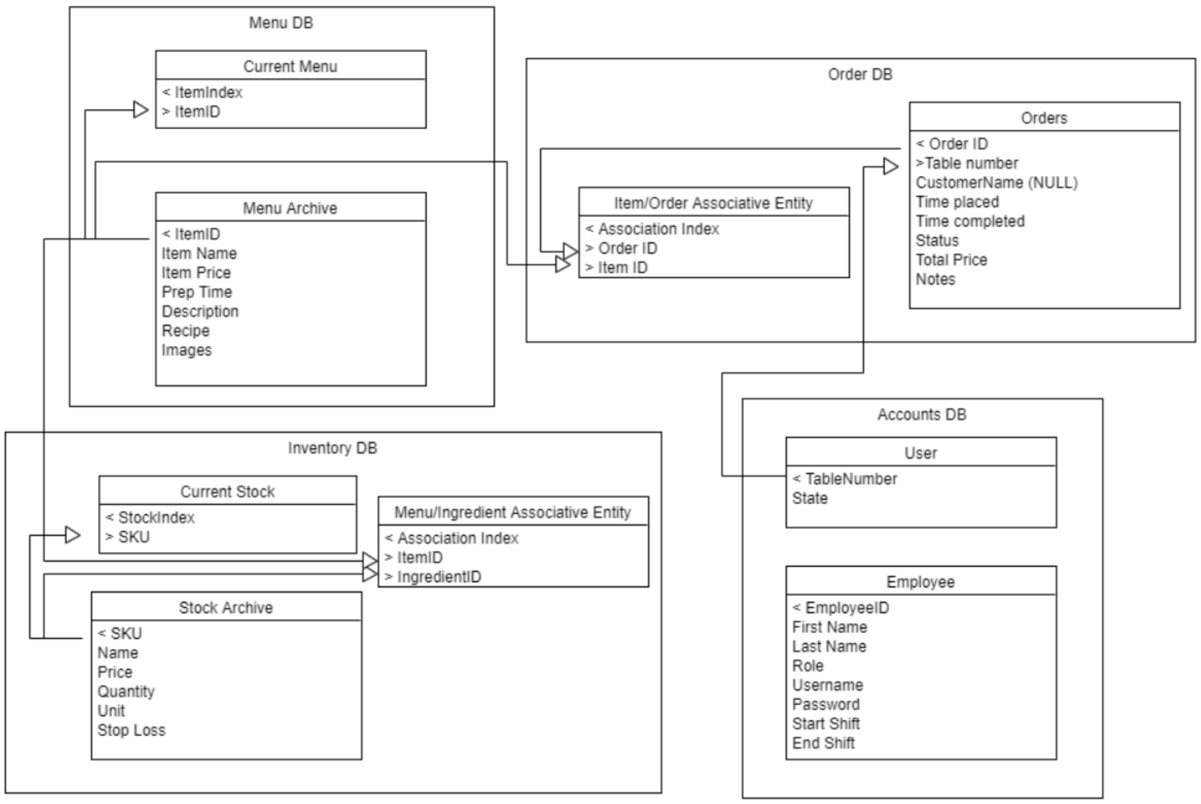
1.2: System Operation Contracts

Operation:	Place Order
Use Case:	UC-1
Precondition:	● There is at least one item in the order ● The user customer does not order something that is unavailable on the Menu
PostCondition:	● The order is sent to Order Queue ● The customer is presented the Order Progress tab

Operation:	Edit Kitchen Stock
Use Case:	UC-9
Precondition:	● There is at least one ingredient in the stock list ● Users must change at least one value or the menu won't update
PostCondition:	● Kitchen stock is correctly updated and fully accurate ● Changes are immediately shown through all smartmenu devices so that the chef and manager are always updated

Operation:	Edit Menu
Use Case:	UC-8
Precondition:	● There is at least one dish on the menu ● The user must change at least one component of a dish to submit changes to the menu
PostCondition:	● The new menu will be accessible by customers of the restaurant with the edits applied ● The old version of the menu will be inaccessible.

1.3: Data Model and Persistent Data Storage



This project will contain 4 separate databases: Menu, Order, Accounts, Inventory. We can containerize each database using Docker, allowing for a mix of database paradigms. Menu DB will likely use a document DB like MongoDB, as it will speed up reads (meaning the Current Menu table might be represented as a flag in Menu Archive). The other 3 will use a relational database allowing for fast writes and joins, such as PostgreSQL.

Menu DB:
The Menu database will two tables:
- Current Menu: Holds foreign keys linked to menu items in the Menu Archive table.
- Menu Archive: Stores details for past and present menu items in the restaurant.

Order DB:
The Order database has two tables:
- Orders: Stores historical data on all orders made at the restaurant with 1 Order Id primary key and 1 Table Number foreign key (links order with location in restaurant).
- Item/Order Associative Entity: maps all many-to-many relationships between orders and menu items. Stores primary keys of both Menu Archive and Order ID tables as foreign keys.

Inventory DB:
The Inventory DB has 3 tables:

- Stock Archive: contains details on all ingredients stored, and a SKU as the primary key.
- Current Stock: points to items currently in stock
- Menu/Ingredient Associative Entity: Maps all many-to-many relationships between a multiple menu item and common ingredients.

Accounts DB:
The Accounts DB has 2 tables:

- User: A User at a dine-in restaurant refers to a specific table that an order is associated with. Contains a field called status that tracks whether a table is occupied or not.
- Employee: Contains details about restaurant employees. Used for account creation and deletion.

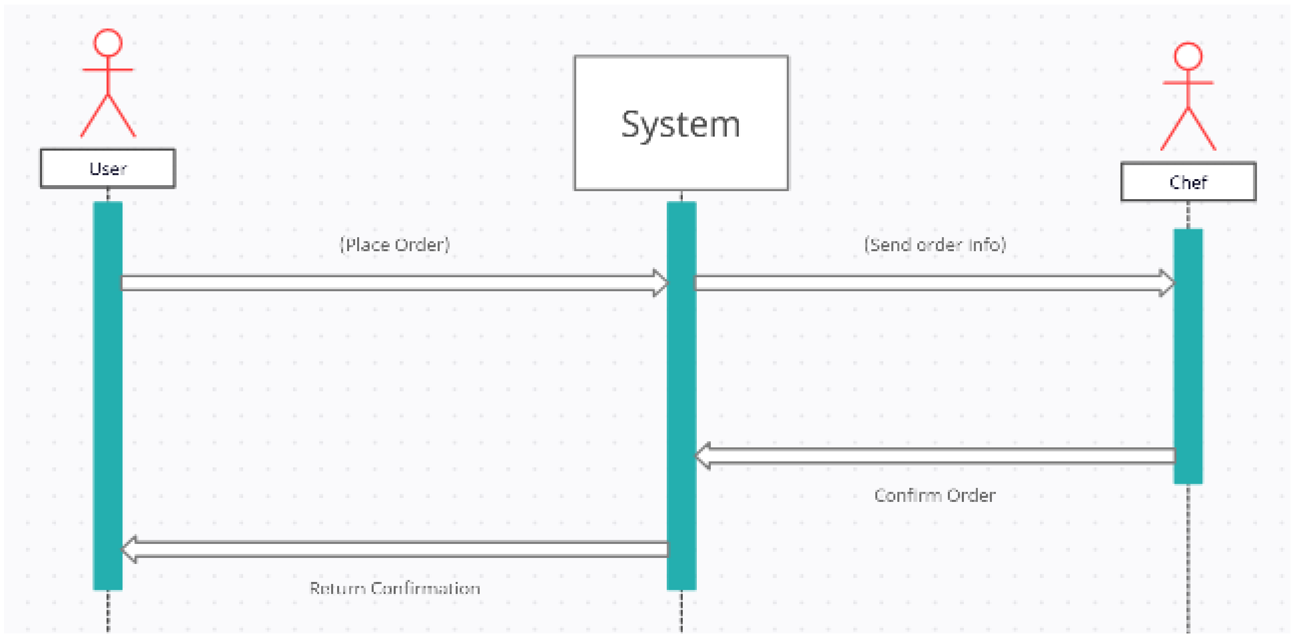
1.4: Mathematical Model

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-11. 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.

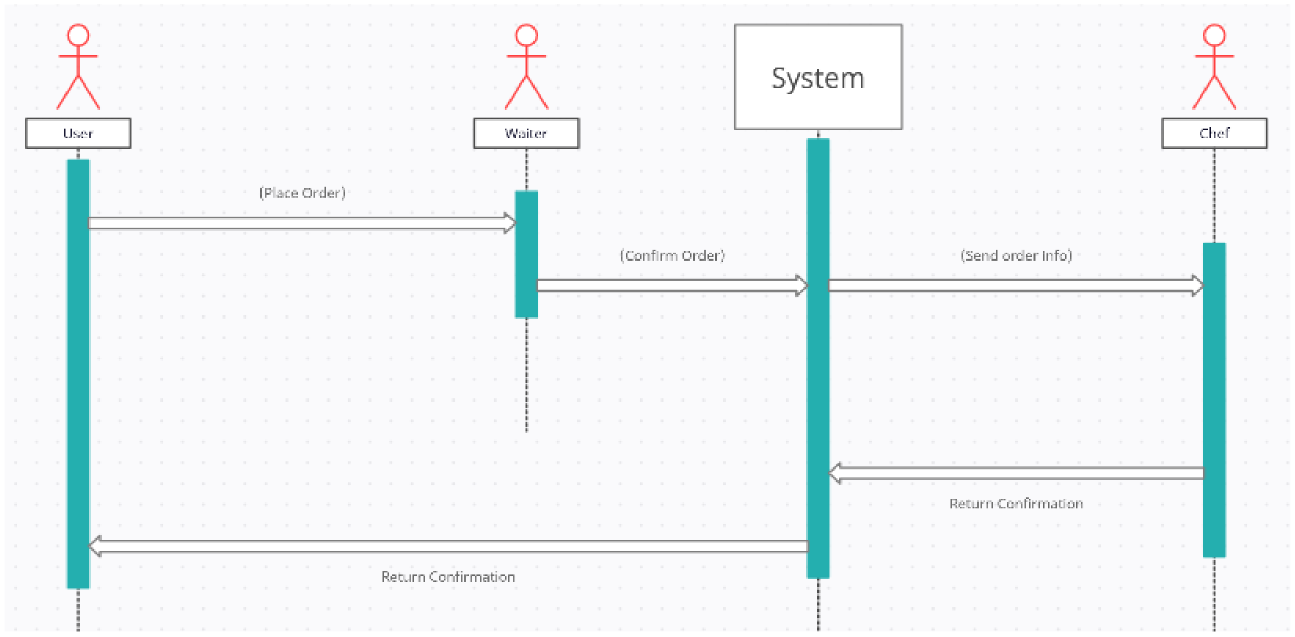
1.5: Interaction Diagrams

UC-1: Place Order

Customer Orders Through Own Device

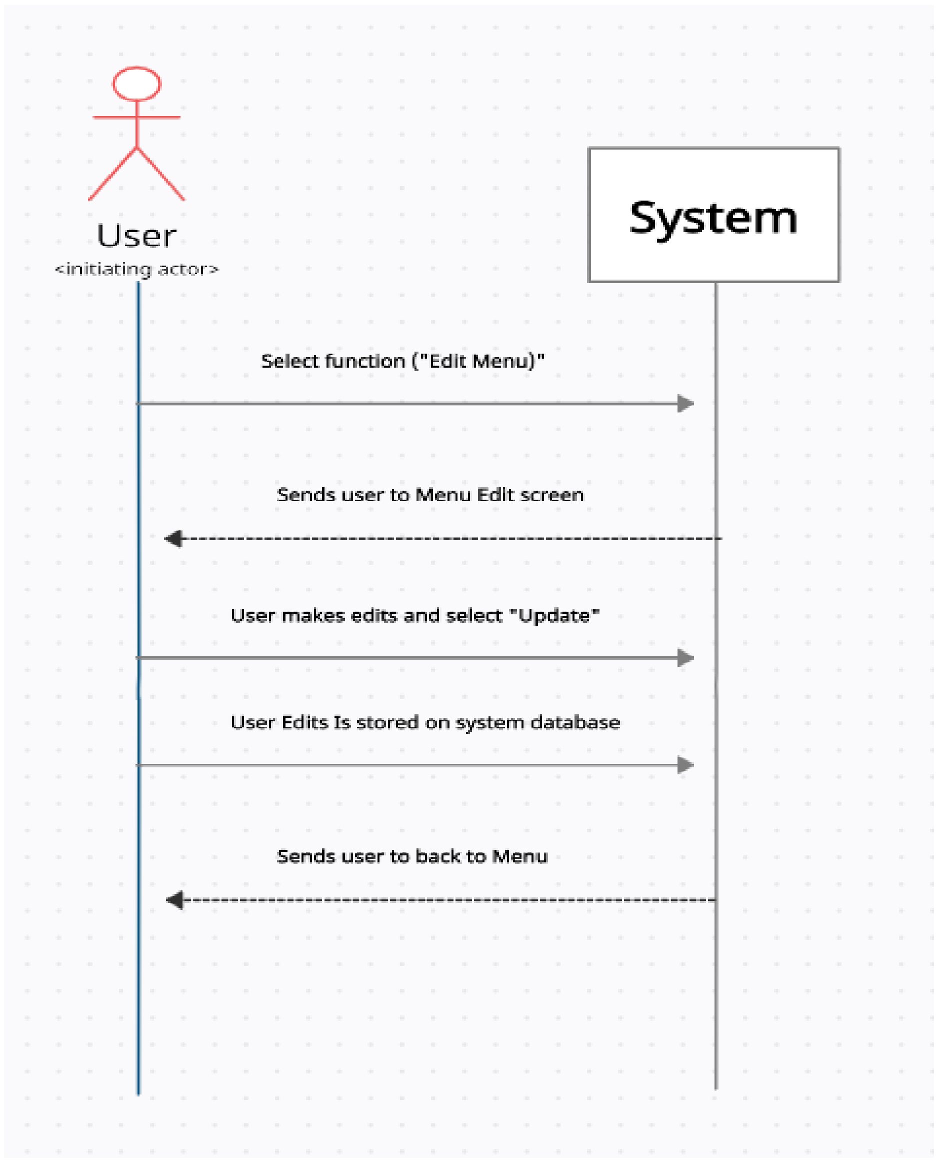


Order Through Waiter



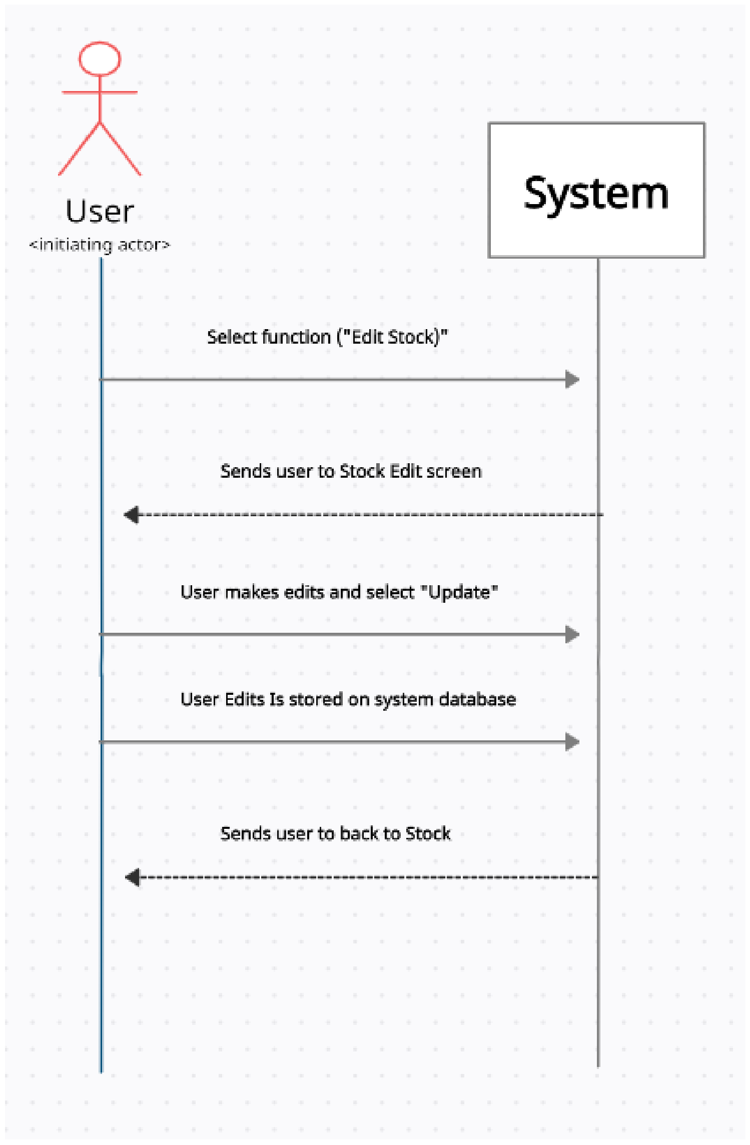
The above sequence diagrams display some of the potential paths that could be taken when placing an order with Robo Ramsey Software. The first diagram shows the user placing the order on their device using our app. The second diagram shows the waiter placing the order on behalf of the customer if it 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-8: Edit Menu


User
<initiating actor>
System
Select function ("Edit Menu)"
Sends user to Menu Edit screen
User makes edits and select "Update"
User Edits Is stored on system database
Sends user to back to Menu

The above sequence diagram are the tasks required to successfully edit and update the menu for all the staff and customers to view. User opens the menu and navigates to the menu edit interface by selecting the edit option. The user is then able to add or edit menu items with these attributes: Name, description, price, ingredients used and amount used from each ingredient, and an image for the item. Each menu item is an object and each attribute is associated with a variable for that object. One design principle that is used is the High Cohesion Principle. All of the objects do not have many computation responsibilities. The interface simply displays information to the user and prompts them for input. The database will store and return related information for each task. So once the menu is updated, it is updated for all users.

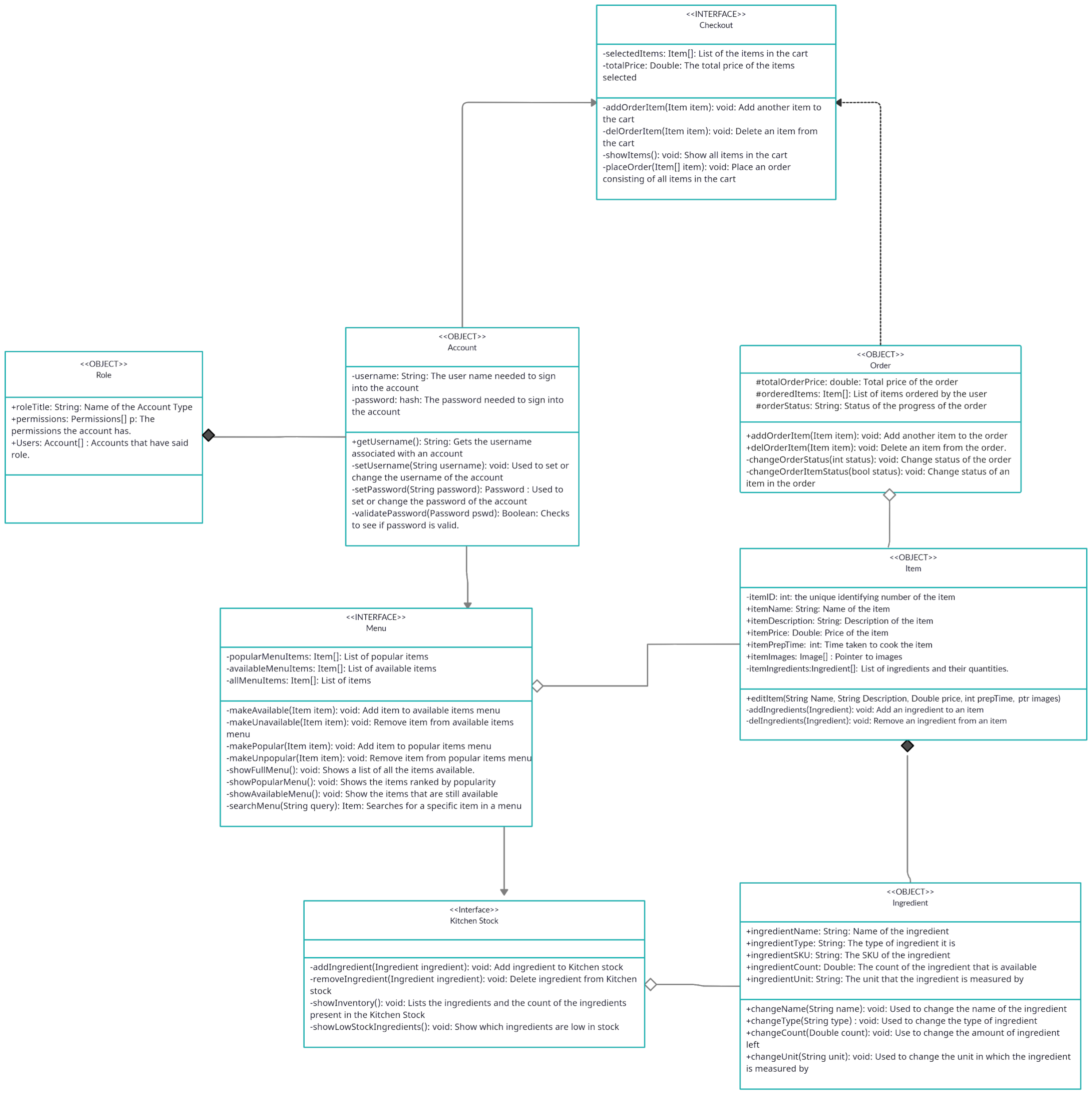
UC-9: Edit Ingredient Stock


User
<initiating actor>
System
Select function ("Edit Stock)"
Sends user to Stock Edit screen
User makes edits and select "Update"
User Edits Is stored on system database
Sends user to back to Stock

The above sequence diagram are the tasks required to successfully edit and update the ingredient stock for all the staff and customers to view. User opens the Kitchen stock, where the ingredients and the count and amount in weight is viewable, and navigates to the stock edit interface by selecting the edit option. The user is then able to add or edit ingredients with these attributes: Name, amount, weight. Each ingredient is an object and each attribute is associated with a variable for that object. One design principle that is used is the High Cohesion Principle. All of the objects do not have many computation responsibilities. The interface simply displays information to the user and prompts them for input. The database will store and return related information for each task.

Section 2: Class Diagram and Interface Specification

2.1: Class Diagram - see attached pdf for clearer image



<<INTERFACE>>
Checkout
-selectedItems: Item[]: List of the items in the cart
-totalPrice: Double: The total price of the items selected
-addOrderItem(Item item): void: Add another item to the cart
-delOrderItem(Item item): void: Delete an item from the cart
-showItems(): void: Show all items in the cart
-placeOrder(Item[] item): void: Place an order consisting of all items in the cart
<<OBJECT>>
Role
+roleTitle: String: Name of the Account Type
+permissions: Permissions[] p: The permissions the account has.
+Users: Account[] : Accounts that have said role.
<<OBJECT>>
Account
-username: String: The user name needed to sign into the account
-password: hash: The password needed to sign into the account
+getUsername(): String: Gets the username associated with an account
-setUsername(String username): void: Used to set or change the username of the account
-setPassword(String password): Password : Used to set or change the password of the account
-validatePassword(Password pswd): Boolean: Checks to see if password is valid.
<<OBJECT>>
Order
#totalOrderPrice: double: Total price of the order
#orderedItems: Item[]: List of items ordered by the user
#orderStatus: String: Status of the progress of the order
+addOrderItem(Item item): void: Add another item to the order
+delOrderItem(Item item): void: Delete an item from the order.
-changeOrderStatus(int status): void: Change status of the order
-changeOrderItemStatus(bool status): void: Change status of an item in the order
<<OBJECT>>
Item
-itemID: int: the unique identifying number of the item
+itemName: String: Name of the item
+itemDescription: String: Description of the item
+itemPrice: Double: Price of the item
+itemPrepTime: int: Time taken to cook the item
+itemImages: Image[] : Pointer to images
-itemIngredients:Ingredient[]: List of ingredients and their quantities.
+editItem(String Name, String Description, Double price, int prepTime, ptr images)
-addIngredients(Ingredient): void: Add an ingredient to an item
-delIngredients(Ingredient): void: Remove an ingredient from an item
<<INTERFACE>>
Menu
-popularMenuItems: Item[]: List of popular items
-availableMenuItems: Item[]: List of available items
-allMenuItems: Item[]: List of items
-makeAvailable(Item item): void: Add item to available items menu
-makeUnavailable(Item item): void: Remove item from available items menu
-makePopular(Item item): void: Add item to popular items menu
-makeUnpopular(Item item): void: Remove item from popular items menu
-showFullMenu(): void: Shows a list of all the items available.
-showPopularMenu(): void: Shows the items ranked by popularity
-showAvailableMenu(): void: Show the items that are still available
-searchMenu(String query): Item: Searches for a specific item in a menu
<<Interface>>
Kitchen Stock
-addIngredient(Ingredient ingredient): void: Add ingredient to Kitchen stock
-removeIngredient(Ingredient ingredient): void: Delete ingredient from Kitchen stock
-showInventory(): void: Lists the ingredients and the count of the ingredients present in the Kitchen Stock
-showLowStockIngredients(): void: Show which ingredients are low in stock
<<OBJECT>>
Ingredient
+ingredientName: String: Name of the ingredient
+ingredientType: String: The type of ingredient it is
+ingredientSKU: String: The SKU of the ingredient
+ingredientCount: Double: The count of the ingredient that is available
+ingredientUnit: String: The unit that the ingredient is measured by
+changeName(String name): void: Used to change the name of the ingredient
+changeType(String type) : void: Used to change the type of ingredient
+changeCount(Double count): void: Use to change the amount of ingredient left
+changeUnit(String unit): void: Used to change the unit in which the ingredient is measured by

2.2: 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. -totalPrice: Double: The total price of the items selected
3. -addOrderItem(Item item): void: Add another item to the cart
4. -delOrderItem(Item item): void: Delete an item from the cart
5. -showItems(): void: Show all items in the cart
6. -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. -orderedItems: Item[]: List of items ordered by the user
3. -orderStatus: String: Status of the progress of the order
4. -addOrderItem(Item item): void: Add another item to the order
5. -delOrderItem(Item item): void: Delete an item from the order
6. -changeOrderStatus(int status): void: Change status of the order
7. -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. +itemPrice: Double: Price of the item
5. +itemPrepTime: int: Time taken to cook the item
6. +itemImages: Image[] : Pointer to images
7. -itemIngredients:Ingredient[]: List of ingredients and their quantities.
8. +editItem(String Name, String Description, Double price, int prepTime, ptr images)
9. -addIngredients(Ingredient): void: Add an ingredient to an item
10. -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. +getUsername(): String: Gets the username associated with an account
4. -setUsername(String username): void: Used to set or change the username of the account
5. -setPassword(String password): Password : Used to set or change the password of the account
6. -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

2.3: Traceability Matrix

	Software Classes							
Domain Concepts	Menu	Item	Checkout	Order	Kitchen Stock	Ingredient	Account	Role
Order			X	X	X		X	X
Menu Item	X	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 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

Menu Item:

- Menu: Each menu item will be listed on the menu, with view according to the availability
- Item: Each item on the menu has a name, description, price, and listed ingredients along with their quantities
- Kitchen Stock: Availability of menu items is based on the kitchen's stock of ingredients
- Ingredients: All ingredients are shown to the user for each menu item
- Accounts: Customers can select a specific menu item to view information of, managers can edit menu items
- Role: All actors have the ability to see available menu items. Customers will not see non-available menu items

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

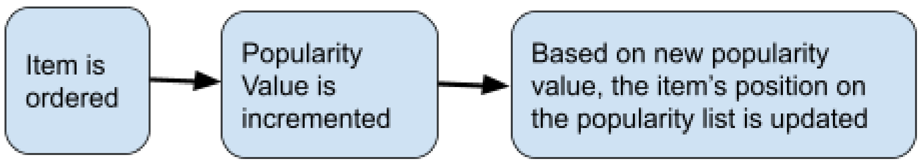
Data Log:

- 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

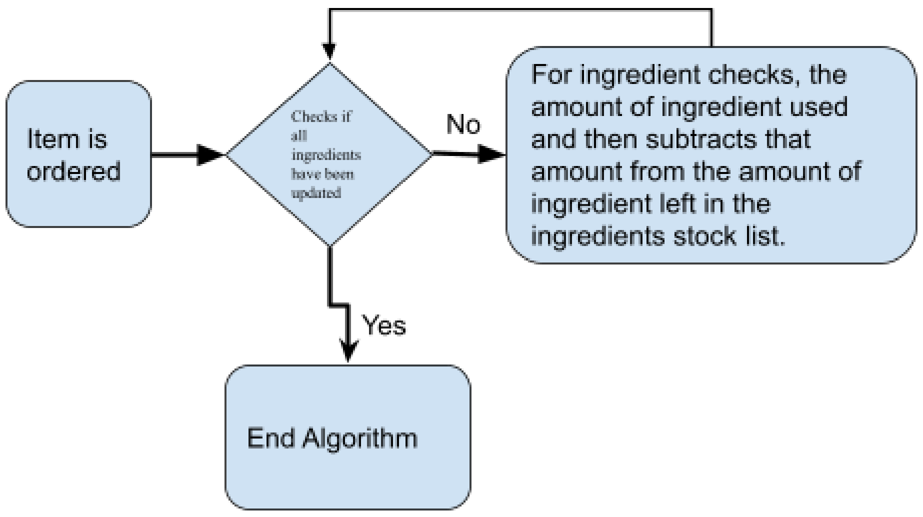
Section 3: Algorithms and Data Structures

3.1: 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. 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.


Item is ordered
Popularity Value is incremented
Based on new popularity value, the item's position on the popularity list is updated

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.


Item is ordered
Checks if all ingredients have been updated
No
For ingredient checks, the amount of ingredient used and then subtracts that amount from the amount of ingredient left in the ingredients stock list.
Yes
End Algorithm

3.2: 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.

Section 4: User Interface Design and Implementation

In the initial screen mock-ups developed in Report #1, very little was modified when implemented. This is because the mock-ups were designed to minimize user effort while maximizing user efficiency. From the 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 a customer perspective, they will have access to a menu and a total checkout screen, which again, will only take 2 clicks per order. From the chef's perspective, they will access almost all of the information made available to the others and the UI was implemented to ensure that access to the application would not interfere with doing their own job.

As far as specifics in implementation, React Native was used for the front end implementation. This allowed us to work our UI design from Report 1 into fruition without losing any convenience for the users. We also used Firestore to code the backend, allowing all of the information in regards to ingredients and menu options to be stored, edited, and retrieved in various parts of the application.

Section 5: Design of Tests

5.1: 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 - 3	Once an order has been placed, the chef user should be able to view the order as well as update the progress of the order.	If the chef user is able to view the order and successfully change its status, then the test is a pass else it would be a fail.
TC - 3	UC - 4	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 - 4	UC - 5	Once an order has been placed, the manager user should be able to view the order as well as update the progress of the order.	If the manager user is able to view the order and successfully change its status, then the test is a pass else it would be a fail.
TC - 5	UC - 6	The user should be able to see all the items on the menu as well as be able to use the popularity filter to filter the items	If the user is able to view all items on the menu and is able to use the popularity filter to sort the items by popularity, then the test is a pass else it would be a fail.
TC - 6	UC - 7	Once an order has been placed, the waiter user should be able to view the order as well as update the progress of the order.	If the waiter user is able to view the order and successfully change its status, then the test is a pass else it would be a fail.
TC - 7	UC - 8	The manager user should be able to view	If the manager user is able

		all the items on the menu and be able to edit the menu	to view the menu and update it without any issues, then the test is a pass else it would be a fail.
TC - 8	UC - 9	The manager user will view the ingredient stock and be able to update it	If the manager user is able to view the ingredient stock and update it successfully, the test is a pass else it would be a fail.
TC - 9	UC - 9	The chef user will view the ingredient stock and be able to update it	If the chef user is able to view the ingredient stock and update it successfully, the test is a pass else it would be a fail.
TC - 10	UC - 10	The manager will view the statistics regarding the popularity of the items	If the statistics are accurately calculated and are able to be viewed by the manager user without any issues, then the test is a pass else it would be a fail
TC - 11	UC - 11	When an ingredient stock is below a certain threshold, an alert is sent out to the chef user and the manager user.	If the alert is sent successfully when the stock is below the threshold, the test is a pass else it would be a fail.
TC - 12	UC - 12	The chef user should be able to request a waiter or the manager.	If the user is able to successfully request a waiter and the manager, then the test is a pass else it would be a fail.

5.2: 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 and filter 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. For the manager, we are testing whether they are able to view and edit the menu as well as the ingredient stock. We are also testing whether the manager is able to view all the current orders that have been placed and is able to check their progress as well as be able to view statistics regarding the popularity of the items on the item. We also are testing whether the manager can also successfully request a waiter. For the waiter, we are testing whether the waiter can also observe and update the progress of a customer's order. For the chef, we are testing whether they can observe the customer's order as well as update the progress as well as be able to view and update the ingredient stocks. We are also testing if the chef is able to request for a waiter as well as the manager. Lastly we are also testing if the chef and manager receive an alert once an ingredient stock falls below a certain threshold.

5.3: 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 6: Project Management and Plan of Work

6.1: Merging the Contributions from Individual Team Members

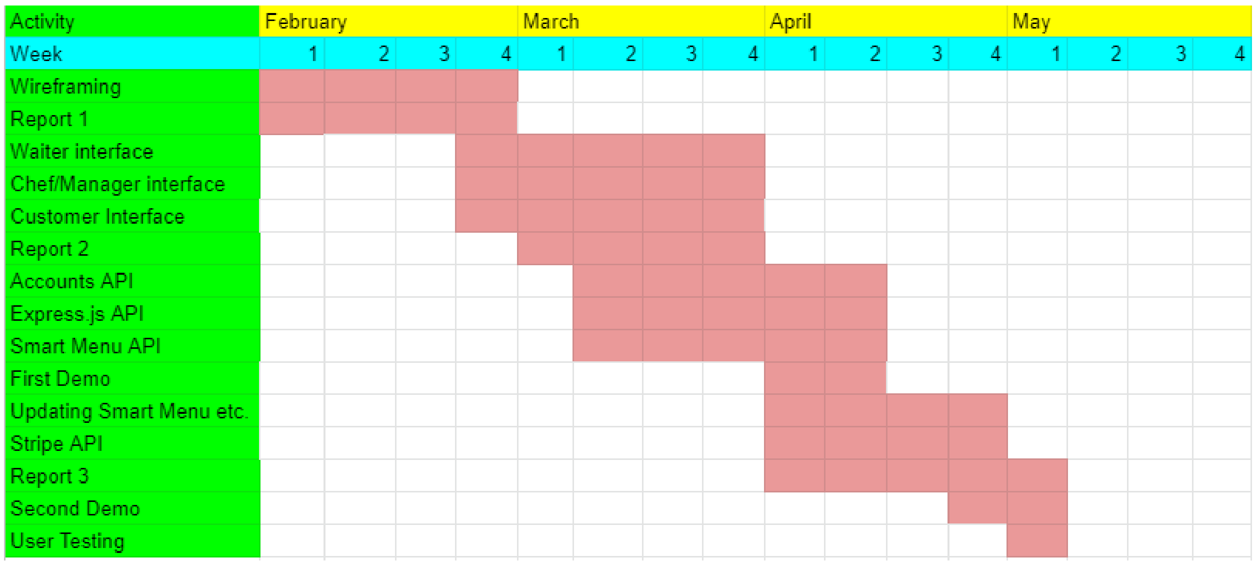
Every member’s work was formatted to belong in either a paragraph, bulleted list, or table format. All work gets placed into sections according to the report format required. There are times that some work does not have everything mentioned that the team agrees upon, so it gets information added to it or is reworded during a team meeting consisting of all members.

6.2: Project Coordination and Progress Report

- Everyone has gotten a baseline understanding of how to use GIT along with GITHUB.
- GITHUB repository does not have any functioning code, rather it has folders for which we will insert our code
- Code will be worked upon shortly

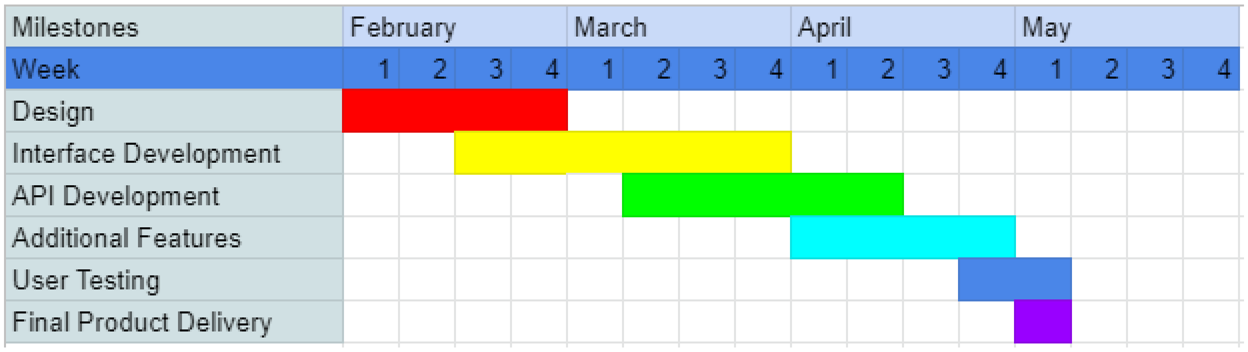
6.3: Plan of Work

Gantt Chart



Activity	February				March				April				May			
Week	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Wireframing	🟥	🟥	🟥	🟥												
Report 1	🟥	🟥	🟥	🟥												
Waiter interface				🟥	🟥	🟥	🟥	🟥								
Chef/Manager interface				🟥	🟥	🟥	🟥	🟥								
Customer Interface				🟥	🟥	🟥	🟥	🟥								
Report 2					🟥	🟥	🟥	🟥								
Accounts API						🟥	🟥	🟥	🟥	🟥						
Express.js API						🟥	🟥	🟥	🟥	🟥						
Smart Menu API						🟥	🟥	🟥	🟥	🟥						
First Demo									🟥	🟥						
Updating Smart Menu etc.									🟥	🟥	🟥	🟥				
Stripe API									🟥	🟥	🟥	🟥				
Report 3									🟥	🟥	🟥	🟥	🟥			
Second Demo												🟥	🟥			
User Testing													🟥			

Project Roadmap



Milestones	February				March				April				May			
Week	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Design	🟥	🟥	🟥	🟥												
Interface Development			🟨	🟨	🟨	🟨	🟨	🟨								
API Development						🟩	🟩	🟩	🟩	🟩						
Additional Features									🟦	🟦	🟦	🟦				
User Testing												🟦	🟦			
Final Product Delivery													🟪			

Product Ownership

Every individual is assigned 2 functional requirements to tackle.

Names/REQ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Raghav											🟦	🟦								
Alan										🟦			🟦							
Esteban			🟦																🟦	
Suryansh	🟦														🟦					
Shrey							🟦	🟦												
Kunj									🟦								🟦			
Adrian					🟦									🟦						
Ray		🟦														🟦				
Akira				🟦														🟦		
Parth						🟦														🟦

6.4 Breakdown of Responsibilities

Section Description: what each team member did so far, is currently doing, will do in the future, including management and coordination activities.

NOTE: requirement assignments/responsibilities subject to change in the future

Raghav: Currently schedules weekly meetings, will implement REQ-11 and REQ-12

Alan: Currently divides labor for each requirement, will implement REQ-10 and REQ-13

Esteban: Currently manages business goals, will implement REQ-3 and REQ-19

Suryansh: Currently manages UI requirements, will implement REQ-1 and REQ-15

Shrey: Currently manages team, will implement REQ-7 and REQ-8

Kunj: Currently manages non functional requirements, will implement REQ-9 and REQ-17

Adrian: Currently manages document formatting, will implement REQ-5 and REQ-14

Ray: Currently manages team communication server, will implement REQ-2 and REQ-16

Akira: Currently manages functional requirements, will implement REQ-4 and REQ-18

Parth: Currently manages references, will implement REQ-6 and REQ-20

References

https://firebase.google.com/docs/database/rtdb-vs-firestore