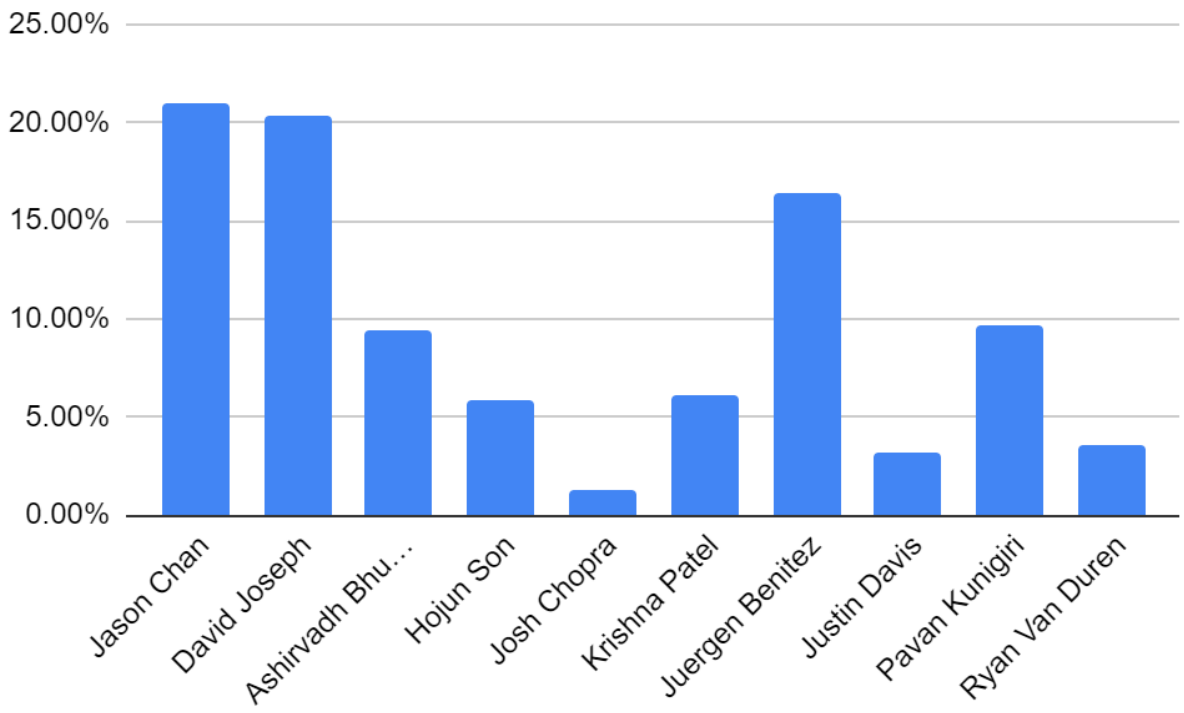
WHY WHINE JUST DINE

Jason Chan
Ashirvadh Bhupathi
David Joseph
Hojun Son
Josh Chopra
Krishna Patel
Juergen Benitez
Justin Davis
Pavan Kunigiri
Ryan Van Duren

Total Contributions Breakdown:
https://docs.google.com/spreadsheets/d/1xWIfiGjpQ62CyRUJnizeW7X-8L-qAOV8YohhL6ydlmk/edit#gid=0

Name	Reports	Demo + Documentations	Code	Total	Coordination	Total + Coordination	Percentages
Jason Chan	16	26	23	65	"+ 5 points to total for coordination"	70.00	20.97%
David Joseph	15	25	23	63	"+ 5 points to total for coordination"	68.00	20.32%
Ashirvadh Bhupathi	11	9	9	29		29.00	9.35%
Hojun Son	11	5	2	18		18.00	5.81%
Josh Chopra	3	1	0	4		4.00	1.29%
Krishna Patel	8	10	1	19		19.00	6.13%
Juergen Benitez	13	15	23	51		51.00	16.45%
Justin Davis	5	0	5	10		10.00	3.23%
Pavan Kunigiri	13	7	10	30		30.00	9.68%
Ryan Van Duren	5	2	4	11		11.00	3.55%
Total	**100**	**100**	**100**	**300**		**310.00**	



Summary Of Changes

- Report
  - Added ECF + Duration
  - Added Object Constraint Language contracts
  - Added Design Patterns
  - Updated History Of Work

        - Updated Gantt Chart
    - Updated Use Cases (Between report 1 and report 2)
    - Updated Test Case Design
    - Updated UI Design and Implementation
    - Updated/Explained data models (ER diagram)
    - Please see contributions for Report 3 for more information (In History of Work)
- App Development
    - Split groups even further so that 2 people worked together on one problem to focus one task and move on
    - Added Paypal payment method - customers can pay us using paypal or a credit card

1.Customer Statement of Requirements

Actor - Host/Hostess:

As a host, my job is to make sure that customers are as comfortable as possible throughout the entire process of dining, or picking up takeout orders. Being one of the first impressions for customers when they enter the restaurant, it is important that I am able to conduct this process as smoothly as possible. However, especially during the times of day where we have a large number of customers flowing in and out of our restaurant, it can be quite difficult to keep track of everything. Currently, as a host I have to keep track of which tables are free, which tables are clean, which tables are reserved, and the location of all these said tables. Furthermore, as a host, another part of my job consists of responding to customer requests timely and efficiently. However, while doing other parts of my job such as welcoming in guests from the front door as well as seating them, sometimes I might not notice a table trying to get my attention. This then can lead to a bit of paranoia as I would have to constantly check on tables to make sure everything is going smoothly. Thus, it would be awesome if there was a program that allowed me to get notified when a customer requests for my presence at their table, and which table I would then have to attend. Furthermore, I would like the program to be able to keep track of all the tables, so it would be much easier to have the tables cleaned, seat the customers, etc. This would allow me to be much more efficient with my work, as I would spend a lot less time seating customers and attending to their requests.

Application Solution: Our application allows the host to see which tables are open, occupied, dirty, and reserved, as well as their corresponding location within the restaurant. The interface will essentially have a status for the table that is able to be changed by the host when a table is reserved, freed, or occupied. Furthermore, through the customer's ordering interface, they will be able to alert the host when they require their presence. The interface on the host's end will notify him that his presence is requested, and from which table.Furthermore, to make it more efficient, during closing time, the host will be able to clean-reset the table statuses to prepare it for the next day.

Actor - Driver

As a driver my job is to deliver food in a timely manner so the food will be fresh when the customer eats the food. To do this I need to be notified immediately when the food is ready so I

can promptly pick up the food from the kitchen and deliver it to the customer. I would also like detailed information on how to contact the customer in case there is a problem. Information like their address and phone number is necessary for me to do my job in an efficient way.

Application Solution: The application will require the chef to mark when the food is finished cooking. This information will then be sent to the driver so they know when to come pick up the food. Any customer who asks for delivery will need to prove their name. Address and phone number. This way the driver will always be able to contact the customer if any problems arise like for example need any help with directions. The driver will input, using the information from whatever navigation software they are using, how long it will take to deliver the food. The customer will also have the option to leave any additional notes in the app that will notify the driver of any specific instructions like leave the food in front of the door.

Actor - Manager:

As a manager the thing I value the most is efficiency. I'm always looking for opportunities to get rid of wastage of resources. A restaurant like ours has a lot of moving parts which at times can get quite chaotic. I would love to have an app that would make my life easier in keeping track of everything that is going on.

One thing I would love to improve in our business would be delivery. With the rise in Covid cases, there has been a greater demand for customers who eat from home. However, delivery from my point of view can be very expensive. I have to hire extra people to do the deliveries and any mistake that occurs can be very difficult to fix. For example if we mess up an order, normally we can quickly fix it in the restaurant. However, due to the extra time it would take in the case of delivery it would leave the customer extremely unsatisfied and frustrated. I would like the app you create to help minimize these mistakes. The app would need to keep track and give me the status of all orders at different stages of completion. Specifically I would want to know if the chef finished cooking the order and at what time it was delivered.

I manage a lot of employees and with the increase in demand for delivery that number will keep going up. Because of this I would like the app to give me the ability to track when they have clocked in and out, as well as keep track when each employee is scheduled to work in the upcoming week. The app should also automatically calculate how much money each employee has made.

I would also like the app to have bookkeeping software and have general data analysis. These features will help me assess how the day to day business is going as well as let me know what products are performing well or doing poorly. I will then use this information to make decisions on where to best invest my resources. To be more specific, I want the app to track which orders have been extremely popular as well as how much money was gained from them. Foot traffic on different days would also be very useful information.

Application Solution: The application will keep track of all orders done by the customer and relay it directly to the chef. The chef will then indicate whenever an order is finished so either the waiter can come to get the food to the customer or the delivery person can come to deliver the food. The delivery driver will need to mark on the app when the order has been

delivered. The application will give the manager data on how many hours each employee has clocked in any given day as well give them the option of seeing future schedules of each employee. This should make it easier for the manager to manage their employees schedules.

For the data analysis, the app will keep track of the popularity of food and drinks in the menu. This will be defined by the number of times an item is ordered during the entire day and it will be shown in the form of a graph. The y axis will be the number of orders and the x axis will be defined by the hour of the day. The app will give more options such as seeing the number of orders by day of the week , or even by month. The use of graphs will make it easier to see trends

The application will keep track of all financial transactions that occur. The app will then use this information to produce graphs that will make it easier for the manager to see how well the business is doing. With this information the manager will be able to make better informed financial decisions.

Actor - Customer:

As a customer, I expect my food and the service to be perfect. I should be getting what I paid for without any issues. When ordering food I want to know what exactly is in the food incase of allergies and for calorie counting purposes. I really don't want to wait for the waiter to give me menus to order, I just want to explore my options before getting seated. A digital menu would be helpful, saving me the time and hassle of talking to the waiter and there being communication issues. If I had the ability to order ahead of time allowing me to cut down my wait time for my food would be fantastic. Along with ordering on my phone, if there was a way for me to know how long my food will take would be amazing, therefore if I order for take out, I know exactly when my hot food will be ready for me to pick up and enjoy. If I wanted to enjoy a nice sit down dinner at the restaurant instead of a take out order, I do not want to have long wait times and wait for hours. If I could reserve a table right on my phone and know exactly when the table will be ready, I would be able to save time and be able to do what I want until the table is ready instead of just waiting at the restaurant for an hour just to be seated. After being seated, I hate waving down waiters for help. I wish there was a way to have two buttons on my table, one to call over the waiter and another to call over the hostess incase of table issues. This would save me the time of waiting for the waiter/hostess to walk past me, instead they should be notified when I need help and then they will be able to come over to me. Due to the COVID-19 pandemic I do not always feel comfortable going into the restaurant to eat and sometimes I just want the food to come to me. If there was a delivery option right from an app where I can place my order and then track the estimated time of arrival as well as track where my delivery driver is would be very helpful.

Application solution: Our application will offer a digital touch menu with features to ensure the highest customer satisfaction. The menu will display what ingredients are in each dish ensuring all allergy concerns are accounted for. Our application will allow customers to select a table to reserve as well as let customers know about wait time so there is no uncertainty. If ordering for delivery the customer will be able to track their order, when the food is ready and when the food is on the way. They will also be able to track the driver that is delivering our food, where they are and when their food will be delivered. Being able to track your food allows for the customer to be more efficient with their time, allowing them to do

whatever they want instead of just waiting for the driver to come with their food. Our application will also let customers reserve tables from their phones before coming to the restaurant. This helps them manage their time better instead of coming to the restaurant and then waiting an hour for their table after arriving. Reserving a table beforehand and the ability to track when their food will be ready will help customers be more efficient with their time.

Actor - Waiter/Waitress:

I get scolded all the time by the chefs and managers for not being perfect at my job, yet I have to juggle so many things at a time. Despite this, they still expect me to perform all my duties perfectly and efficiently. I have to attend to customers, take their orders, fetch food from the kitchen, deliver orders to their corresponding table, and collect the bills from the customers. I always have to keep checking on orders in the kitchen to see if they are ready, and if i'm even a bit late the chef gets mad at me, while the customers are unhappy their order is taking a bit long to come out. Furthermore, I have to remember which orders are for which table, as well as which tables have been brought their food or not. Furthermore, I have to remember which tables have paid their bills or not, as well as paying attention to any waving hands stuck in my face to get my attention. Something that tells me whether an order is complete in the kitchen, as well as estimated time of completion would save a lot of trips to the kitchen checking if an order is ready. Furthermore, if there was another way for the customer to notify me that they are ready to order or if they need anything else, that would be awesome.

Application solution: This application will allow the customer the option to self-order using a digital menu, or to call the waiter to order. This saves the waiter as they will know when the table is ready to order, as well as some customers ordering for themselves. Furthermore, there will be a button on the customers interface that will alert the waiter. On the waiter's end of the interface, they will be quickly notified that a table requests for them, and which table they are seated at. Furthermore, all the orders will digitally be sent to the kitchen, saving the waiter a trip. The waiter is given an estimated completion time with each order, as well as which table the order is coming from. Once the order is complete, the waiter will be notified and they can then proceed to deliver the food to the corresponding table. Furthermore, the waiter can also see which tables are ready to pay, have paid already, or haven't paid yet.

Actor - Chef:

As a chef, I take pride in the food that I prepare for our customers. However, one thing that troubles me is how messy orders can be especially during busy hours. A lot of times orders come in with special modifications requested, and when multiple orders for the same dish come in but with different modifications, it can be confusing sometimes. Quite frankly it's hard to keep track of everything especially when I am running around the kitchen preparing the food. Furthermore, sometimes the orders that the waiters write down are hardly legible. If I had some way to be able to have these orders more organized, as well as any customizations clear, that would be much appreciated. Another Issue that I have in the kitchen is sometimes the waiters take way too long to come in to get the food. Not only does it take up space on the countertop, and makes things cluttered, but it also causes the food to get cold. I'm a big advocate of serving food while it's hot, and I dislike it when those waiters are so slow to deliver the food. I don't

entirely blame them as they are also very busy, but I would like a way to be able to notify the waiters that an order is ready so they can get it ASAP.

Application Solution: The orders will digitally be displayed in neat order for the chef to read easily. Furthermore, the chef will be able to notify the waiters when the food is ready, as well as an estimated time of when they think food will be ready once they start an order. Furthermore, and modifications to orders will be highlighted so the chef can see it clearly.

Actor - Busboy:

Being a busboy is not always the hardest job but there are always those days that can end up being stressful. The busy days where we have a lot of customers coming in and out are the worst. The hostess is relying on me to let them know when the tables are cleaned and ready for the next party to be seated. If there was an option in the application that shows which tables have been cleared off and cleaned, it would be a more efficient way to let the hostess know when the tables are ready for the next guest to enjoy.

Application Solution: Our application will offer an interactive display which will keep track of every step along the way of a successful dining experience. The busboys will be informed when a customer has left the restaurant, which will let the busboys know to clean those certain tables. Once the busboys clean the table they can update the status of the table on the application which will then let the hostess know that the table is clean and available for the next party to be seated.

2.Glossary of Terms

Busboy

An employee at a restaurant who performs the cleaning tasks in the restaurant. This includes: clearing and cleaning tables, taking dishes and cleaning them in the kitchen, and setting tables for new customers who enter the restaurant. With this app the busboy could be notified when tables need cleaning and is able to communicate with other employees when a table is cleared and a customer could be seated there.

Chef

A chef is a professional cook who has trained in the culinary arts to prepare food for customers of a restaurant. With this app the chef would be able to receive orders and notify other employees of when orders are prepared to be served.

Clock-in Hours

Employees work for a certain amount of hours a day and the Owner/Manager needs to keep track of this information so that employees get paid accordingly. Clock-in hours will be a feature in our app to keep track of how many hours employees have been working for per day.

Cook Time

The cook time is the amount of minutes/seconds required to make a meal for a customer by a chef. With our system, the cook time is managed by someone in the kitchen (in this case the chef) and they provide estimates of times for waiters and customers so they could prepare for their meal.

Customer
A person or organization that buys goods from another person or organization. With our app, the customer can do many things such as place orders in or out of the restaurant depending on what they feel like doing.

Customer Account
For the person buying goods or services and provides them a user account which will be protected by a username and password. Using an account will allow you to order from this restaurant online and will accumulate points for their reward system.

Customer Service
An employee of the restaurant that helps customers out with all sorts of issues whether it be technical (ex. an issue with the app) or physical (ex. order was wrong).

Database
A set of data stored on our network that can be accessed by users such as Owners/Managers.

Delivery
If a customer chooses to not want to eat inside the restaurant and eat in the luxury of their own they can do so. Delivery allows a restaurant to bring a customer their food directly to their location. This type of ordering will usually amount a small additional fee (for travel expenses).

Delivery Driver
An employee working for a restaurant that brings a customer their order directly to them. They will drive their own or company owned cars to bring a customer their food. With our app they can communicate with the customer of when their food has been taken from the restaurant, when they are driving, and when they have arrived at the desired location.

Dine-In
When a customer wants to sit down and eat at the restaurant itself.

Discounts
A discount is when an organization reduces the price for customers. Using the reward system with our app, rewards earned by frequent customers can be used to reduce the price of items.

Dish Popularity
When a meal is enjoyed by many customers, its approval rating up making it a popular dish. With our app, our rating system can produce data that will be sent to Owners/Managers that show which dishes are popular based on how each customer rates each meal and based on

how many customers ordered the meal. This data can help managers push certain dishes forward, adjust prices on certain meals, remove items from the menu, and even modify certain meals.

Employee
An employee is a person who works at an organization, which in this case is a restaurant. These employees vary in jobs done such as: busboy, waiter/host, or chef.

Employee Account
An employee using our app can login to their account with their given username and a created password. This account can see many things based on their occupation. Every employee can see their hours worked, and can clock in. Chefs and Waiters can see when orders have been placed, which tables have placed each order, and what each order contains. Waiters and Busboys can see which tables are occupied, available, or need to be cleaned.

Floor Layout
A floor layout is a diagram of the restaurant. For certain employees using our app, they can see whether a table is available, occupied, or dirty. They could also see which order has come from which table.

Menu
The list of food choices and drinks a customer can order. With our app customers can view the restaurant's menu to pick different options, which will be delivered to the other employees to make and bring the customer that item.

Order
A list of items from the menu that a specific customer or group of people being served has ordered.

Order Queue
The list of orders the chefs must cook. Follows a first in first out order ,so orders that are placed the soonest will get served more soon. The app will show the chef what they need to make.

Owner/Manager
The person who manages employee scheduling, payroll, customer service issues, inventory, and ensures that the restaurant is operating in an efficient manner.

Owner Account
A login for the owner and/or manager that comes with special privileges that other employees don’t have. This can include being able to update the restaurant menu or seing sales analysis.

Payment
The amount of money a customer will need to buy their order. Either through cash or card depending on what customer chooses.

Reservation
When a customer(s) decides to dine-in and schedule a table for themselves ahead of time.

Rating System
Customer feedback on either a menu item or the restaurant. Based on five stars where 1 star is less favorable and 5 stars is exceptional.

Reward System
A system where regular customers can receive special offers by buying items. Requires a user account to claim an offer or reward. The way our reward system works is whenever a customer (with an account) buys an item, they earn a small amount of reward points. Once the customer reaches a certain amount of reward points, they can use this to get a discount off menu items and possibly get items for free if enough reward points are earned.

Table Status
A table's status is either available, occupied/reserved, or dirty based on whether no one is at the table, is at a table or when a customer wants a table in advance, or a customer has just left a table which needs to be cleaned for the next customer. Using our app, employees (waiters/hosts and busboys) can view and modify a tables' status and customers can view a tables' status and reserve a table as well.

Take-Out
Take-out is when a customer does not want to sit and eat at the restaurant, and instead of being delivered to a certain location but can directly pick up their food from the store.

Tip
A tip is a small amount of money, separate from the meal amount, that can be included for a waiter for providing an enjoyable experience.

Tracking (Delivery)
Delivery tracking is used to provide a customer with information about where their food is currently. Our app will provide information to customers who order online for when their food is about to be delivered.

User Account
A private location on a network server that stores information about a user's information such as usernames, passwords, names, etc. for the app.

Waiter/Host
The host is an employee at a restaurant that seats customers when they enter the restaurant. Waiters are also employees, however they take orders directly (if the customer chooses not to order through the app) and/or serve customers with the food they order.

3.System Requirements

3.1 Enumerated Functional Requirements

Key: Customer, Manager, Employee

Food Delivery

Key	Label	Priority	Description
	REQ-1	4	The application shall provide takeout, delivery, and dine in options.
	REQ-2	2	The application shall allow for the user to track order delivery.
	REQ-3	4	The application shall allow the user to view the estimated arrival time of a delivery.
	REQ-4	3	The application shall allow for tracking the status (cooking, almost done, complete) of food orders.
	REQ-5	4	The application shall allow the manager to view delivery completion time.
	REQ-6	2	The application shall allow the employee to check delivery status. `
	REQ-7	4	The application shall allow the chef to notify waiters when food is ready to deliver.
	REQ-8	4	The application shall allow the manager to view order completion time.

Account Management

Key	Label	Priority	Description
	REQ-9	2	The application shall allow the registered members of customers to login into their customer account.
	REQ-10	2	The application shall allow the customers who do not have their accounts to create their new account.
	REQ-11	2	The application should allow the customers to change their usernames or passwords.
	REQ-12	3	The application should allow the customers to retrieve their username or password that they have forgotten.

	REQ-13	1	The application shall allow for a rewards system tied to the accounts of customers.
	REQ-14	2	The application shall allow the registered member of manager to login into his/her manager account.
	REQ-15	2	The application shall allow the new manager to create the new manager account.
	REQ-16	2	The application shall allow the manager to delete his/her account.
	REQ-17	3	The application should allow the manager to retrieve the username or password that he/she has forgotten.
	REQ-18	2	The application shall allow the registered members of employees to login into their employee account.
	REQ-19	2	The application shall allow the new employees to create the new employee account.
	REQ-20	3	The application should allow the manager to retrieve the username or password that he/she has forgotten.

Scheduling

Key	Label	Priority	Description
	REQ-21	1	The application should allow the manager to view employee schedules
	REQ-22	2	The application shall allow the manager to view when employees have arrived and left.
	REQ-23	2	The application shall allow the employees to clock in and clock out.
	REQ-24	3	The application should allow the manager to create schedules for employees
	REQ-25	2	The application should allow the manager to view and edit employee payroll
	REQ-26	2	The application should allow the manager to view finances

Table Status

Key	Label	Priority	Description

	REQ-27	3	The application shall allow the user to see which tables are available.
	REQ-28	2	The application shall allow the manager to view the number of available tables and their respective number of seats.
	REQ-29	2	The application shall allow the manager to view the time when a table has paid their receipt.
	REQ-30	1	The application shall allow the manager to view the receipt of different tables.
	REQ-31	5	The application shall allow the employee to view which tables are occupied and empty.
	REQ-32	3	The application shall allow the employee to view which tables are dirty and clean.
	REQ-33	4	The application shall allow the employee to check whether a table has paid their receipt or if they have yet to do so.
	REQ-34	3	The application shall allow for tracking the status (cooking, almost done, complete) of food orders.

Ordering Interface

Key	Label	Priority	Description
	REQ-35	5	The application shall allow customers to order food from a provided menu.
	REQ-36	4	The application shall allow the user to view order start time and estimated completion time.
	REQ-37	2	The application shall suggest popular menu choices
	REQ-38	2	The application shall allow the manager to update the menu by removing or adding menu items.
	REQ-39	5	The application should provide the customer option to pay by debit, credit, or cash

Data Analysis

Key	Label	Priority	Description
	REQ-40	2	The application shall produce informational graphs using data compiled from orders.
	REQ-41	1	The application shall allow the manager fast access to customer reviews.
	REQ-42	2	The application shall provide data on sales and update finance accounts accordingly.

3.2 Enumerated Nonfunctional Requirements

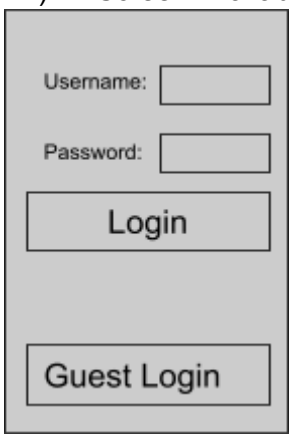
Key: Customer, Manager, Employee

Key	Label	Priority	Description
	REQ-42	5	The application should be easy to use and be user friendly. Any person, even with little to no app experience should be able to use this app without too much trouble. If one does have trouble they should be able to contact customer service easily.
	REQ-43	3	The application should only take a maximum of a few seconds of delay time between each step during the ordering process.
	REQ-44	5	The application should be well phrased so that all customers can easily understand functions of each part of the application.
	REQ-45	4	The application should be able to accept up to 250 total customers without severe system interruption.
	REQ-46	2	The application should display information to customers in high quality of resolution. For example: if their phone has a maximum resolution of 2280 x 1080, the output of our app should match that
	REQ-47	3	The application should open and close almost instantaneously when a user chooses to exit the app.
	REQ-48	2	The application should send the table vacancy or delivery completion information to the manager within a few seconds for efficient table/delivery flow.
	REQ-49	4	The application should send the information about incomplete delivery to the manager within 5-10 minutes when delivery is not completed after the estimated time.

	REQ-50	5	The application must be designed in the way the manager can handle the fast recovery when system error occurs. They can click on a button that then displays customer service information.
	REQ-51	3	Security requirements in place to authenticate users and therefore user privileges (manager has advanced privileges)
	REQ-52	4	The application should allow the employee to receive order information in less than 5-10 seconds after a customer finishes ordering.
	REQ-53	3	The application should display order information or customer complaints with simplicity in the Order lists section, so that the employees can easily view such information.

3.3 User Interface Requirements

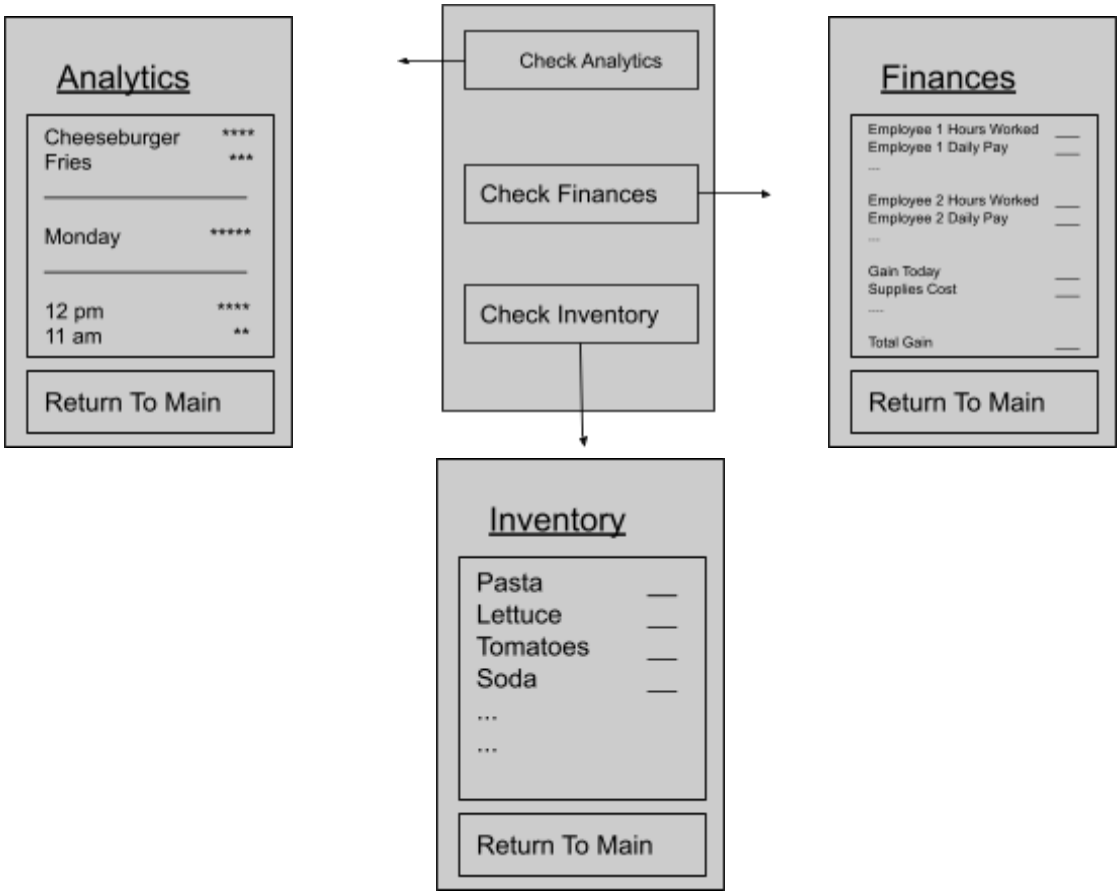
1) Screen Available to all users.Depending on the output, will load into different screens



Key: ALL

Key	Label	Priority	Description
ALL	REQ-54	3	All of the customers, managers, and employees would be able to log in to their account by typing in the username and password that they have created into the text boxes.

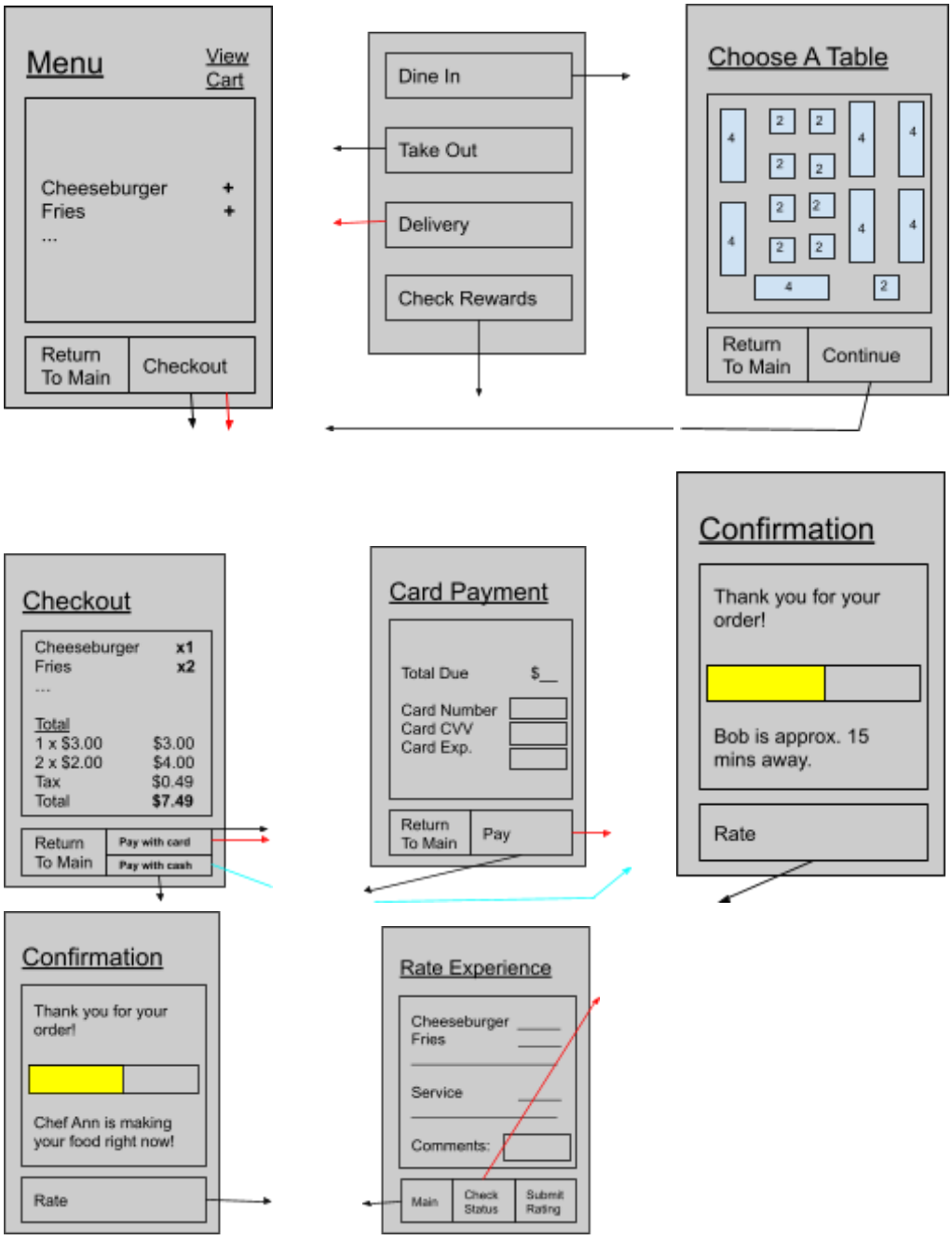
2) Owners Screens



Key: Manager

Key	Label	Priority	Description
	REQ-55	3	On the main screen, the manager would be able to look at analytics, finances, or inventory conditions of the restaurant by clicking on the button of corresponding sections.
	REQ-56	4	In the finance page, the manager would be able to see overall total revenue gained as well as a list of the outcomes and incomes. The manager would be able to return to the main page after finishing looking at information.
	REQ-57	4	In the inventory page, the manager can track the current stock of the ingredients used in the restaurant by real time. The manager would be able to return to the main page after finishing looking at information.
	REQ-58	4	In the analytics page, the manager can track various types of statistics of total number sold for each menu, number of customers visited each day in a week, and number of customers visited for each time in a day.

3) Customers Screens (Difference between logged in user and guest will be reward system visibility)



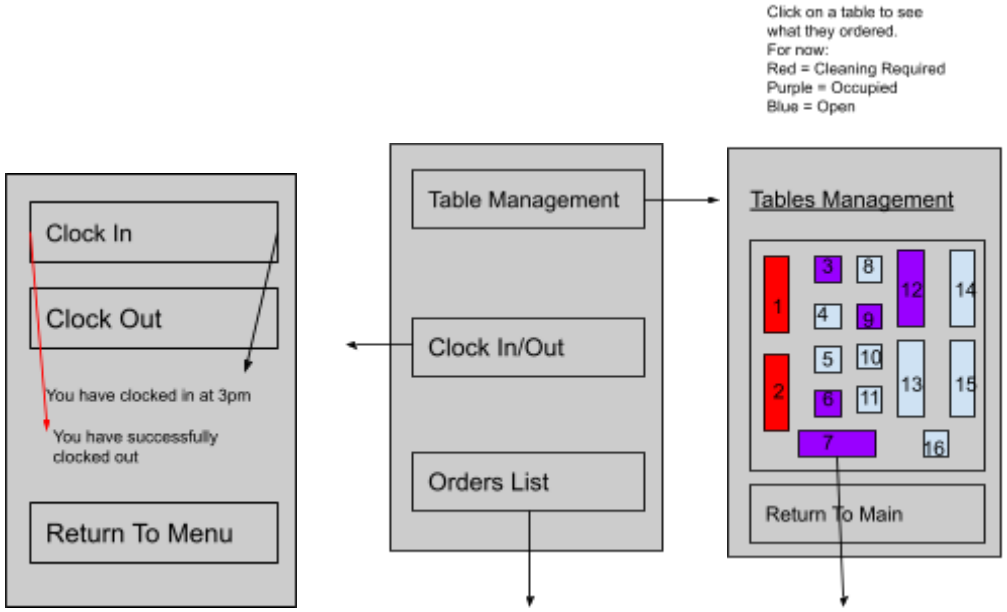
Key: Customer

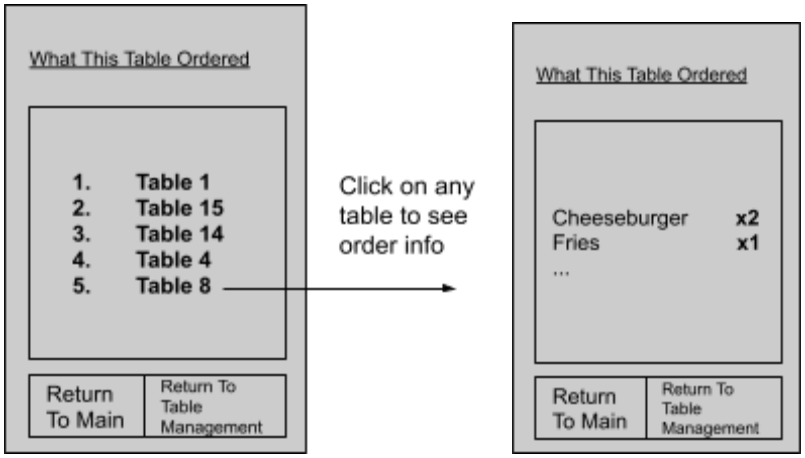
Key	Label	Priority	Description
	REQ-59	3	On the menu, the customer should be able to View the Menu and add Food items to their Cart, as well as access a “Cart” and Checkout screen.

	REQ-60	2	After logging in the Customer has the choice to click on Dine in, Take out, Delivery, and check their rewards.
	REQ-61	4	In the Checkout page the customer can view their selected items, as well as the cost per item and total cost. They can also choose to return to main to cancel it, or pay with card/cash.
	REQ-62	3	On the card payment screen the customer can input their card info to pay.
	REQ-63	3	After selecting dine in, the customer can then choose their table from a table select page, from any available and clean tables.
	REQ-64	3	There are several confirmation pages for various types of customers(Dine in, Delivery, Take out). It should show approximate time it takes for the order to be complete as well as an option to rate the service.
	REQ-65	2	After finishing their meal customers get an option to rate their meal. Customers will also be allowed to give feedback to the chef through the comment section.

4) Employees Screens

Waiter/Host:

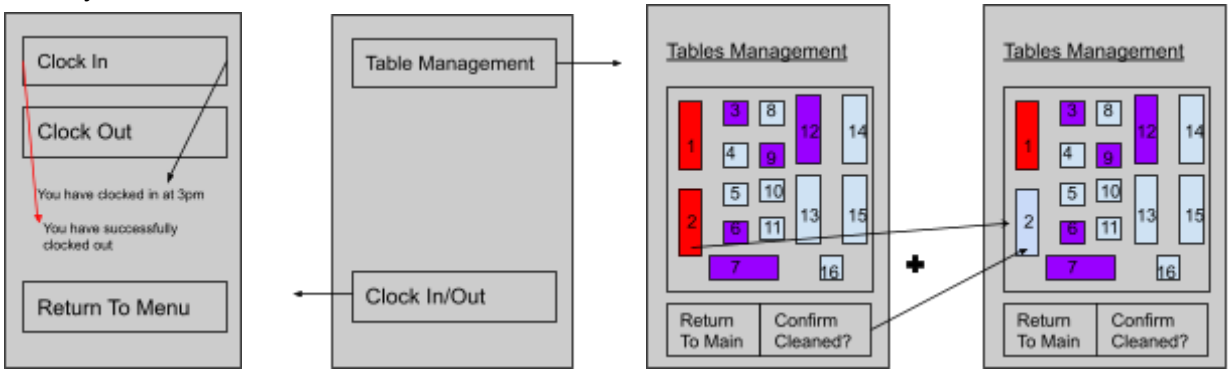




Key: Employee

Key	Label	Priority	Description
	REQ-66	3	The waiter would have the choice to view the table management page, clock in/out page, or list of orders page by clicking on the corresponding buttons.
	REQ-67	3	In the clock in/out page, the waiter would be able to notify their attendance through the clock in button and get off of the restaurant through clicking on the clock out button. The waiter would be able to return to the main menu after clicking the clock in or clock out button.
	REQ-68	4	In the tables management page, the waiter would be able to view the current status for each table being in available, occupied, or dirty condition. The busboy would be able to return to the main page upon viewing the page.
	REQ-69	5	In the orders list page, the waiter can track the tables that customers in such tables have confirmed the order. By clicking each table, the waiter can see the menu that customers in each table have ordered.

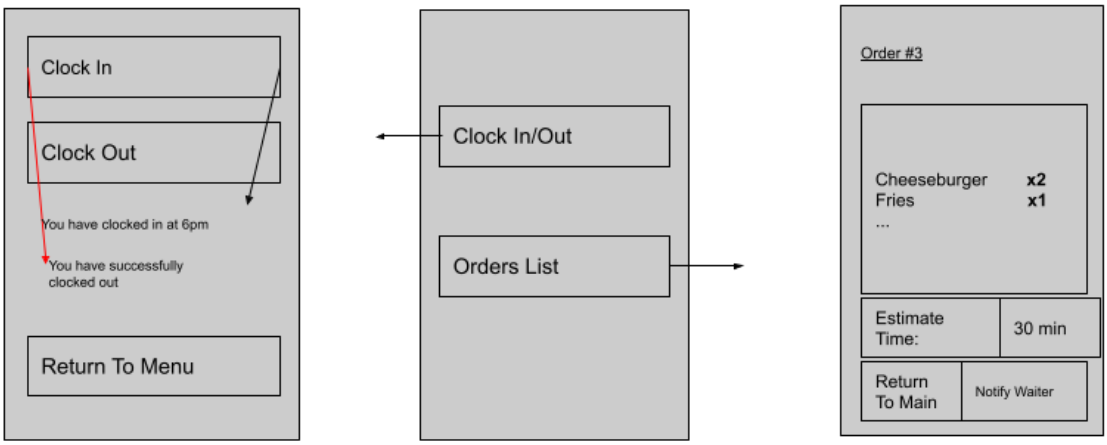
Busboy:



Key: Employee

Key	Label	Priority	Description
	REQ-70	3	The busboy would have the choice to view the table management page or clock in/out page by clicking on the corresponding buttons.
	REQ-71	3	In the clock in/out page, the busboy would be able to notify their attendance through the clock in button and get off of the restaurant through clicking on the clock out button. The busboy would be able to return to the main menu after clicking the clock in or clock out button.
	REQ-72	4	In the tables management page, the busboy would be able to view the current status for each table being in available, occupied, or dirty condition. The busboy would be able to return to the main page or click the certain table to change the status of that table to be in available condition. The status of that table would be turned into an available condition upon clicking the “Confirm Cleaned?” button.

Chef:



Key: Employee

	REQ-73	2	The chef should be able to login with his/her credentials to see the chef main page.
	REQ-74	4	On the menu, the chef has two options, one being able to clock in and clock out for the manager to see how many hours a day the chef has worked, and the other being an orders list for the chef to produce food in its given order for the customer.
	REQ-75	3	After pressing the “Clock In/Out” option on the main screen, the chef should be able to Clock in or clock out based on when he/she arrives and when he/she ends work.

	REQ-76	4	After pressing the “Orders List” option on the main screen, the chef should be able to see what customers have ordered (sorted based on when each order was placed). The chef should also be able to put out estimates (possibly preexisting) and notify waiters when orders are done.

Driver
Key: Employee

Key	Label	Priority	Description
	REQ-77	3	On the menu screen, the driver has the option of looking at the food status or the Delivery/customer information.
	REQ-78	4	On the Food Status screen the driver will see if there is any food that is ready to be picked up from the kitchen for delivery.
	REQ-79	4	On the Delivery/customer information screen the driver will have to input how long it will take them to deliver the food. This screen will also give the driver information about the customer and ways they can contact them.

4.Functional Requirements Specification

4.1 Stakeholders

There will be a number of stakeholders needed for this app to be used efficiently. The stakeholders include: **the Owner and/or Manager, the Waiter/Waitress and the Host, the Busboy, the Chef, the Driver, and the Software Developers.**

The restaurant owner and/or manager can use this app to make their business more efficient by allowing their employees to benefit from an effective app which would allow them to communicate with each other without having to waste time. The customers, who are the most valuable asset to the restaurant, could also benefit from this app due its nature of being simple and effective allowing them to choose how they want to spend their time and money at this restaurant. Lastly, the software developers are the key to a great app for this restaurant. Without them, the app will not be able to run smoothly, or even exist! That being said, software developers will help build the app even further, creating more features based on how the owner and or managers sees fit to make a more efficient method for employees and suitable experience for customers.

4.2 Actors and Goals

Initiating actors

Actor	Role	Goal
Customer (Account)	A customer with an account will be able to choose any of the available methods of ordering food such as dine-in, take-out, or delivery. They could also reserve a table before arriving at the restaurant and place/track orders as well. Lastly, customers with accounts will gain and spend rewards based on what they have purchased previously.	A customer, when using our app, should have a great experience using our and promote more business in the restaurant especially with a reward system. This actor will likely be one who returns many times.
Customer (Guest)	A guest will be able to choose any of the available methods of ordering food such as din-ine, take-out or delivery. They could also reserve a table before arriving.	A customer without an account should have a similar experience to one with an account without earning rewards. This actor will be those trying out this restaurant likely.
Manager	A manager with an account will be able to add and remove items from the menu.	The manager will be able to easily alter the menu using the interface.
Waiter/Waitress/Host	A waiter, waitress, or host should be able to serve cooked foods to the customers who ordered such foods. They should be able to tell customers and busboys when tables become occupied, free, or need cleaning.	Through the application, the waiter can serve foods to the right table so that the miscommunication or situation in which foods are served to wrong tables would not take place. The waiter can welcome new customers to the vacant table with time saved using the application.
Chef	The chef should be able to notify the waiter when an order is ready to be picked up	The goal is to have efficient communication between the chef and the waiter so that the food can be delivered as quickly as possible.
Busboy	A busboy should be able to notify other employees and customers of when a table is cleaned.	The busboy will be able to notify others of when tables open up providing a more fluent system of restaurant automation.
Driver	A driver should be able to tell a customer where they are and	The driver should be able to communicate with a customer

	approx. how much time it will take to reach them.	allowing customers to be satisfied when an order arrives based on ongoing information.

Participating actors

Actor	Role	Goal
Waiter/Waitress/Host	A waiter, waitress, or host should be able to receive orders, track which foods are ordered from each table, and check the vacant tables.	Through the application, the waiter, waitress, or host would be able to look at all of the orders received from each of the tables so that foods can be served to proper tables. The waiter, waitress, or host can also track for the non-dirty vacant tables that customers can be seated on.
Chef	A chef should be able to view customer orders and begin cooking meals depending on when each order was placed.	Through the app, the chef will be given clear and concise instructions about when each item was placed and who ordered the meal to be more productive and efficient.
Busboy	A busboy should be able to check when a table is dirty, occupied or clean.	This function allows busboys to be proactive and instead of waiting around looking for tables to be clean, can be notified when a table needs to be cleaned.
Driver	A driver should be able to check when an order is ready for pickup.	This allows drivers to be notified of when an order is ready to be picked up saving time.
Manager	The manager should be able to view seating statuses and order statuses.	This allows them to oversee the restaurant to make sure everything is running smoothly.

4.3 Use Cases

4.3.1) Casual Description

Food Delivery-

- **Use Case 1(UC1) Types of orders :** The applications will allow a customer to place an order for Dine in, Take out, or Delivery.
  Responds to REQ 1
- **Use Case 2(UC2) Order Status:** The waiters and chefs will be able to communicate to each other when an order is ready to be delivered. The manager can also keep track of completion of orders.
  Responds to REQ 7
- **Use Case 3(UC3) Track Order :** The applications will allow a customer, manager, and employee to view the order status.
  Responds to REQ 2,3,4,5,6,8,36

Scheduling-

- **Use Case 4(UC4) Clock In:** The application will allow for employees to check in when they are starting their shift.
  Responds to REQ 22, 23
- **Use Case 5 (UC5) Clock Out:** The application will allow for employees to check out when they are finished working their shift.
  Responds to REQ 22, 23
- **Use Case 6 (UC6) Employee Schedule:** The application will allow for managers to create and manage weekly schedules for employees for when they have to work.
  Responds to REQ 21, 24
- **Use Case 7 (UC7) Payroll:** The manager will be able to view payrolls for employees as well as be able to pay them.
  Responds to REQ 25
- **Use Case 8(UC8) Finance:** The manger will be able to view the finances of the company and their analytics.
  Responds to REQ 26

Account Management-

- **Use Case 9(UC9) Login :** The applications will allow a manager, customer or employee to login, or change password to their account.
  Responds to REQ 9,11,12,13,14,17,18,20
- **Use Case 10(UC10) Logout :** The applications will allow a manager, customer or employee to logout from their account.
- **Use Case 11(UC11) Register:** Allows a guest to create an account so that he or she can access the application features. The employees can also create their corresponding employee accounts.
  Responds to REQ 10,15,19

- **Use Case 12(UC12) Delete:** Allows a manager to delete an account so that the important restaurant information can be kept secure when managers quit the job or the manager account gets hacked.
  Responds to REQ 16
- **Use Case 13(UC13) Edit:** Any user is able to edit certain information in their account, after having inputted their username and password(login).
  Responds to REQ:38

Order Interface-

- **Use Case 14(UC14) Take Out** : The applications will allow a customer to place and pick up an order.
  Responds to REQ: 35,
- **Use Case 15(UC15) Dine in** : The applications will allow a customer to choose to dine in.
  Responds to REQ:35
- **Use Case 16(UC16) Delivery** : The applications will allow a customer to place a delivery
  Responds to REQ:35
- **Use Case 17(UC17) Payment:** Allows the customer to pay for their meal, after which the chef will begin cooking.
  Responds to REQ: 39
- **Use Case 18(UC18) Rate Food:** This application will allow the customer to rate the quality of the food on a scale from 1(worst) to 5 (best).
  Responds to REQ 37

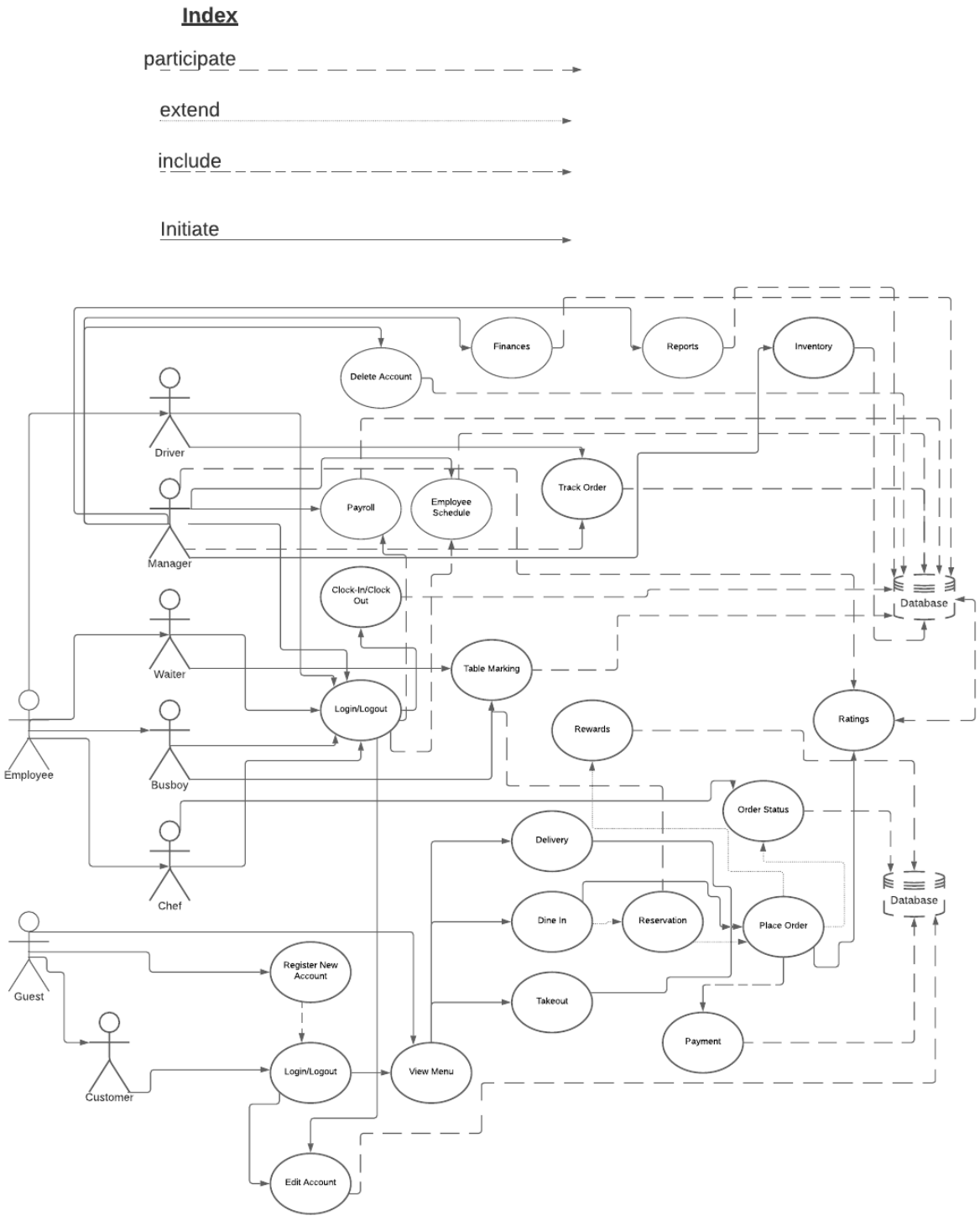
Table Status-

- **Use Case 19(UC19) Reservation :** Allow the customer to reserve a table for a later date or time right from our application without having to call the restaurant.
  Responds to REQ: 27
- **Use Case 20(UC20) Check Tables :** Allows employees to check a table's status like the number of seats at the table, if it's open , if it's clean , and customer payment status.
  Responds to REQ: 28, 29, 30, 31, 32, 33, 34

Data Analysis-

- **Use Case 21(UC21) Reports :** The application will make reports which allows managers to view important information aspects about the restaurant generated from the daily customers in easy to follow tables, charts and graphs.
  Responds to REQ:40,41,42

4.3.2) Use Case Diagram



4.3.3) Traceability Matrix

	W	UC 1	UC 2	UC 3	UC 4	UC 5	UC 6	UC 7	UC 8	UC 9	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16	UC 17	UC 18	UC 19	UC 20	UC 21
REQ1	4	X																				
REQ2	2			X																		
REQ3	4			X																		
REQ4	3			X																		
REQ5	4			X																		
REQ6	2			X																		
REQ7	4		X																			
REQ8	4			X																		
REQ9	2									X												
REQ10	2											X										
REQ11	2									X												
REQ12	3									X												
REQ13	1									X												
REQ14	2									X												
REQ15	2											X										
REQ16	2												X									
REQ17	3									X												
REQ18	2									X												
REQ19	2											X										
REQ20	3									X												
REQ21	1						X															
REQ22	2				X	X																
REQ23	2				X	X																
REQ24	3						X															

REQ25	2							X													
REQ26	2								X												
REQ27	3																		X		
REQ28	2																			X	
REQ29	2																			X	
REQ30	1																			X	
REQ31	5																			X	
REQ32	3																			X	
REQ33	4																			X	
REQ34	3																			X	
REQ35	5														X	X	X				
REQ36	4		X																		
REQ37	2																	X			
REQ38	2													X							
REQ39	5																X				
REQ40	2																				X
REQ41	1																				X
REQ42	2																				X
Total	W	4	8	19	4	4	4	2	2	18	0	6	2	2	5	5	5	5	2	3	20	5

4.3.4) Fully-Dressed Description

UC-9: Login
Related Requirements: REQ-9,REQ-11, REQ-12,REQ- 13,REQ-14,REQ-15,REQ-16,REQ-17,REQ-18,REQ-20
2Initiating Actors: Manager, Employees, Customers
Actor Goals: Grant access to corresponding accounts in the system

Participating actors: database
Preconditions: User already has account created
Postconditions:
Flow of Events for Main Success Scenario: 1. →The user opens the application to view the login/guest user page 2. →The user clicks to login 3. ←The system prompts for account information 4. →The user inputs their account username and password 5. ←The systems the displays the corresponding interface depending on the user
Flow of Events for Alternate Success Scenario 1 (Password reset): 1. →The user opens the application to view the login/guest user page 2. →The user clicks to login 3. ← The system prompts for account information 4. →The user inputs an account username and password 5. ←The system displays that incorrect information was inputted and prompts the user to try again or prompted to change their password 6. →The user inputs their correct account username and password (4 and 5 can loop) or clicks to change their password. 7. ←The systems then displays the corresponding interface depending on the user
Flow of Events for Alternate Success Scenario 2 (Created an account): 1. →The user opens the application to view the login/guest user page 2. →The user clicks to login 3. ←The system prompts for account information 4. →The user clicks forgot password 5. ←The system prompts for email to send password reset info to

UC-21: Reports
Related Requirements: - REQ-40, REQ-41, REQ-42,
Initiating Actor: Manager
Actor's Goal - Allows the manager to view important aspects of a restaurant's data generated daily by customers in the form of tables, charts, and graphs.
Participating Actors: Database
Preconditions: Manager has signed into the app.
Postconditions: Manager is shown information.
Flow of Events for Main Success Scenario:

1. →
   The manager opens the application to the homepage
2. →
   The manager clicks on “Sign-In”. Is prompted with a “Login Page”.
3. ←
   The manager enters his/her username and password. Then they can hit login to enter their account.
4. →
   The manager is brought to their “Manager/Owner Homescreen”. The manager can now click on the “View Analytics”
5. →
   The manager is presented with information regarding restaurant data in the form of tables, charts and graphs.

UC-15: Dine-In
Related Requirements: - REQ-35
Initiating Actors: - Customers
Actor’s Goal: - Allows the customers (Account and Guest) to order the table to have a seat.
Participating Actors: - Waiter/Waitress/Host, Busboy, Database
Preconditions: - Customers have signed into the application.
Postconditions: - The customers are shown choices of the available tables to reserve.
Flow of Events for Main Success Scenario: 1. → The customers open the application to the entry page. 2. ← The application gives options for customers to sign in with their accounts or as guests. 3. → The customers click the “accounts” button. 4. ← The application shows up the spaces for inputs of user-ids and passwords. 5. → The customers can type in their user-ids and passwords in order to successfully sign in. 6. ← The application shows the “Customer Homescreen” with choices of dine-in, take-out,

delivery, or check rewards.

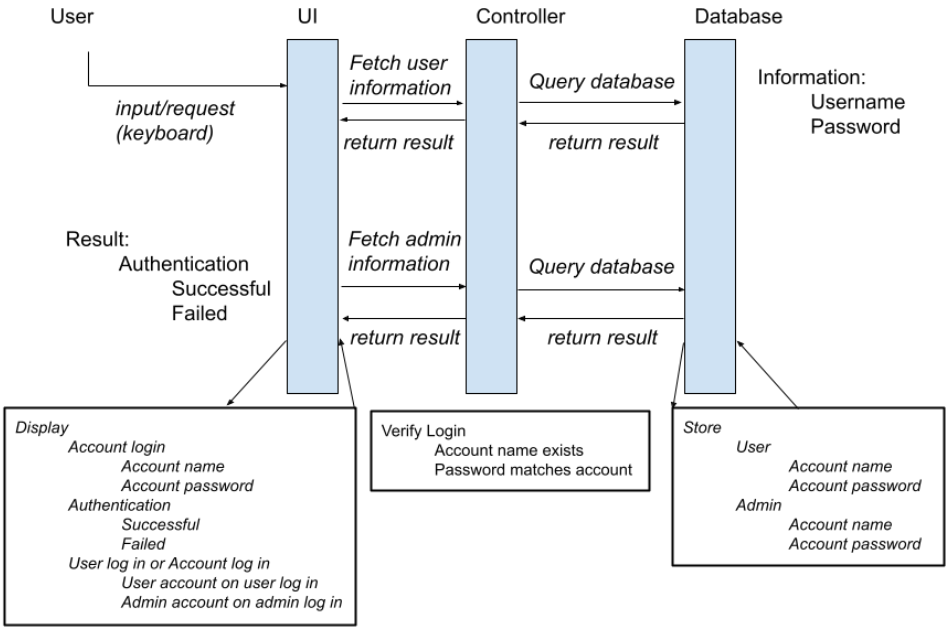
7. →
   The customers click the “dine-in” option.
8. ←
   The application shows the screen of the “Dine-In” section with vacant, available tables shown.
9. →
   The customers can reserve one of the available tables.
10. ←
    The application notifies the successful reservation of the table.

Flow of Events for Alternate Success Scenario:

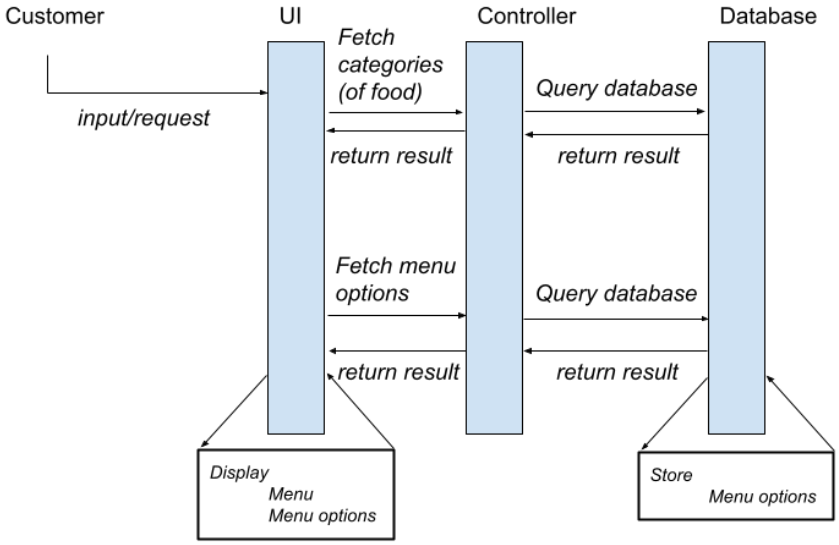
1. →
   The customers open the application to the entry page.
2. ←
   The application gives options for customers to sign in with their accounts or as guests.
3. →
   The customers click the “guest” button.
4. ←
   The application shows the “Customer Homescreen” with choices of dine-in, take-out, delivery, or check rewards.
5. →
   The customers click the “dine-in” option.
6. ←
   The application shows the screen of the “Dine-In” section with vacant, available tables shown.
7. →
   The customers can reserve one of the available tables.
8. ←
   The application notifies the successful reservation of the table.

4.4) System Sequence Diagrams

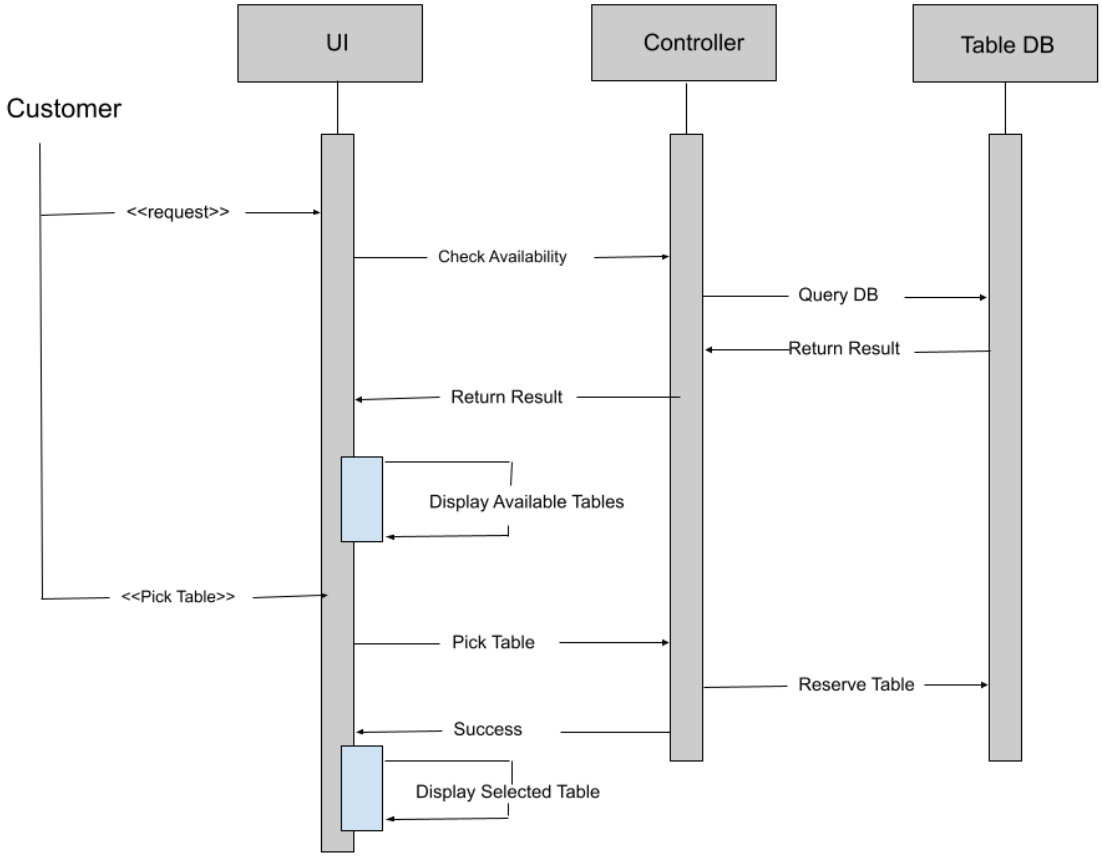
UC-5 Login



UC-6 Menu



UC-10 Table Selection



5.Effort Estimation using Case Points

5.1) Find UAW

Finding UAW

Table 4-1: Actor classification and associated weights.

Actor type	Description of how to recognize the actor type	Weight
Simple	The actor is another system which interacts with our system through a defined application programming interface (API).	1
Average	The actor is a person interacting through a text-based user interface, or another system interacting through a protocol, such as a network communication protocol.	2
Complex	The actor is a person interacting via a graphical user interface.	3

Actor Name	Description	Complexity	Weight

Customer	Customer is interacting via graphical interface	complex	3
Manager	Same as customer	complex	3
Waiter/Waitress	Same as customer	complex	3
Host/Hostess	Same as customer	complex	3
Chef	Same as customer	complex	3
Busboy	Same as customer	complex	3
Driver	Same as customer	complex	3

UAW = 1*(0) + 2(0) + 3(7) = 21

5.2) Finding UUCW

Finding UUCW

Table 4-3: Use case weights based on the number of transactions.

Use case category	Description of how to recognize the use-case category	Weight
Simple	Simple user interface. Up to one participating actor (plus initiating actor). Number of steps for the success scenario: ≤ 3. If presently available, its domain model includes ≤ 3 concepts.	5
Average	Moderate interface design. Two or more participating actors. Number of steps for the success scenario: 4 to 7. If presently available, its domain model includes between 5 and 10 concepts.	10
Complex	Complex user interface or processing. Three or more participating actors. Number of steps for the success scenario: ≥ 7. If available, its domain model includes ≥ 10 concepts.	15

Use Case	Description	Category(Simple,Average,or Complex)	Weight(5,10,15)
UC1:Types of Orders	One initiating actor with more than 3 steps.	Average	10
UC2:Order Status	A couple actors involved. A logged in	Average	10

	user should be able to do this in a couple steps.		
UC3:Track Order	A couple actors involved (customer, manager, and employee). A couple steps for logged in users.	Average	10
UC4:Clock In	One initiating actor and a couple steps.	Simple	5
UC5:Clock Out	Same as Clock in.	Simple	5
UC6:Employee Schedule	1 involved actor, and a couple steps (Login,click employees option, and edit)	Simple	5
UC7:Payroll	Almost the same as UC6	Simple	5
UC8:Finance	1 involved actor(manager) and several steps to check the finances	Average	10
UC9:Login	One initiating actor and a few simple steps.	Simple	5
UC10:Logout	Same as Login	Simple	5
UC11:Register	A couple involved actors (customers or employees) and a few steps (Login, create account, enter info)	Average	10
UC12:Delete	A couple actors involved and several steps after login..	Average	10
UC13:Edit	An initiating actor and a couple steps after the login to make changes to the	Simple	5

	account.		
UC14:Take Out	1 initiating actor, several participating actors.Steps: login choose order, pay, rate.	Average	10
UC15:Dine In	1 initiating actor, 4 participating actors, and 10 steps in the success scenario.	Complex	15
UC16:Delivery	Almost the same as takeout	Average	10
UC17:Payment	One initiating actor and several steps since a user has to make an order first.	Average	10
UC18:Rate Food	Customer rates at the end of their order	Simple	5
UC19:Reservation	1 initiating actor, several participating actors and steps:Login,select Dine In, choose table, make order, and pay.	Average	10
UC20:Check Tables	A couple involved actors (employees) and a few steps to check.	Average	10
UC21:Reports	1 participating actor and a couple steps.	Simple	5

UUCW = 5(9) + 10(11) + 15(1) = 45 + 110 + 15 = 170

5.3) Finding TCF

Finding TCF

Technical factor	Description	Weight	Perceived Complexity	Calculated Factor (weight *

				perceived complexity)
T1	Distributed system (running on multiple machines)	2	Average (3)	2*3 = 6
T2	Performance objectives (are response time and throughput performance critical?)	1	Average (3)	1*3 = 3
T3	End user efficiency	1	Simple (0)	1*0 = 0
T4	Complex internal processing	1	Complex(5)	1*5 = 5
T5	Reusable design or code	1	Average (3)	1*3 = 3
T6	Easy to install	0.5	Simple (0)	0.5*0 = 0
T7	Easy to use	0.5	Simple (0)	0.5*0 = 0
T8	Portable	2	Average (3)	2*3 = 6
T9	Easy to change(add or modify features)	1	Average (3)	1*3 = 3
T10	Concurrent use	1	Simple (0)	1*0 = 0
T11	Special security features	1	Average (3)	1*3 = 3
T12	Provides direct access for third parties (the system will be used from multiple sites in different organizations)	1	Average (3)	1*3 = 3
T13	Special user training facilities are required.	1	Simple (0)	1*0 = 0
			Technical Factor Total	**32**

Constant 1 = 0.6
Constant 2 x Technical Factor Total = 0.01(32)
TCF = Constant-1 + Constant-2 x Technical Factor Total
TCF = 0.6 + 0.01(32) = 0.92.

5.4) Finding ECF

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor(Weight * Perceived Impact
E1	Beginner familiarity with the UML- based development	1.5	1	1.5*1
E2	Some Familiarity with application problem	.5	2	.5*2= 1
E3	Some Knowledge of object-oriented approach	1	2	1*2 =2
E4	Beginner lead analyst	.5	1	.5*1= .5
E5	Stable requirement expected	2	5	2*5= 10
E6	No part-time will be involved	-1	0	-1*0 = 0
E7	Programming Language of average difficulty will be used	-1	3	-1*3 = -3
Environmental Factor Total				7

ECF = 1.4 - 0.03 × Environmental Factor Total = 1.4 - 0.03 × 7 = **1.19**

5.5) Use Case Points

Use Case Points = UUCP * TCF * ECF = 191*0.92*1.19 = 209

5.6) Duration

UCP* PF = 209*28 = 5852 Hours

6.Domain Analysis

6.1) Domain Model

6.1.1) Concept Definitions

Responsibility	**Type**	**Concept**
R1: Coordinate activity between customers, chef, waiters, busboys, drivers etc	D	Controller
R2: Display a user interface for customers, chef, waiters, busboys, drivers, and managers	D	Interface
R3: Store customer login information and purchase history, and reward points	K	Customer Profile
R4: Store employee login information, employee type, and hours worked	D	Employee Profile
R5: Prompt customer to select Dine In / Takeout / Delivery, select table, place order, make payment, etc.	D	Controller
R6: Prevent invalid table selections	D	Table status
R7: Store customer's order in the database	K	Communicator
R8: Queue incoming orders for the chef to complete	K	Order Queue
R9: Record statistics for manager report	D	Analytic Calc
R10: Allow chef to input and update estimated food times	D	Order Status
R11: Manage interactions with the database	K	DB Connection
R12: Handle payment processing	D	Payment System
R13: Change table status when a customer selects a table, reserves a table, checks out, or when a busboy cleans the table	D	Table Status
R14: Prevent invalid table selections	D	Table Status
R15: Display current orders to be served and the table (waiters)	K	Serving
R16: Display current orders to be delivered and the address (drivers)	K	Delivering

R17: Display current tables to be cleared (busboys)	K	Clearing

6.1.2) Association Definition

Concept Pair	**Attribute Description**	**Association Name**
Customer Profile ⇔ Database	Retrieve customer's data from the database	AccessDB
Customer Profile ⇔ Interface	A Display of customer's options	Customer Display
Controller ⇔ Analytic Calc	Display statistics for specific users(different users can see different stats)	Display Stats
Interface ⇔ Controller	Allows the user to interact with the interface(that will depend on the access level of the user). Ex. Customers can interact with the menu, payment options, food ratings, etc.	User Actions
Communicator ⇔ Database	Insert, Delete, or Update data in the database. (Some actions may require admin level permissions)	UpdateDB
Communicator ⇔ Order Queue	Send order and update Order List for the chef	Send Order
Analytic Calc ⇔ Database	Receive data from database for data analysis	AccessDB
Controller ⇔ Order Status	Allow users to update or view the food status. Example, the Chef updates the order as complete, and the customer/waiter can view this status.	Update/View Order Status
Controller ⇔ Table Status	Allows users to view and update table statuses depending on permission level. (Busboys can update as clean, Hosts can update tables as occupied, etc.)	View/Update Table Status
Controller ⇔ Payment System	Allows customers to complete the payment through various options.	Pay
Payment System⇔	Store payment record/receipt into	Payment records

Database	the database	
Interface ⇔ DB Connection	Retrieve the data from the database that the user requests(given appropriate permission levels). Some data may be only available to certain users such as admin.	AccessDB

6.1.3) Attribute Definitions

Concept	Attribute	Description
Customer Profile	accountUsername	Associated username of the customer. Guest account is assigned if no account.
	accountPassword	The password of the User Account
	accountRewards	Rewards associated to a customer and accumulates with additional purchases.
Interface	currentInterface	Depending on what acccount type is logged in, show interface for that type of account only.
	receipt	Allows the user to confirm order after choosing food items.
	rateMeal	Allows the user to rate a meal, and then have that rating stored and displayed.
Controller	tableList	Shows the customer the current tables avaliable.
	tableConfirm	Allows the customer to confirm the table they sit at.
	paymentMade	Once the customer pays, the table is then confirmed and the chef begins to cook the meals.
Communicator	customerMeal	Each meal that the customer orders is stored in the database.
	customerMealOrder	The meals in the queue are ordered and sent to a specific chef, to balance the chef work load.
	customerMealType	Shows the type of meal to the employees, whether it be dine-in, takeout, or delivery.
Order Queue	chefQueue	Contains information about the order that has been placed by the customer.
Analytic Calc	inventoryStats	Records statistics about restaurant inventory to alert manager when stock is low.
	customerStats	Record statistics about peak customer times, most ordered meals, etc.
Table Status	tableNumber	Displays the table number.
	seatCount	Max number of people that can be seated at a particular table.
	tableStatus	Provides and updates status of each table, whether it is avaliable, occupied, or dirty.
Food Status	orderStatus	Allows chef to update status of each table, whether it is available, occupied, or dirty.
	orderReady	Allows chef to update waiter and customer that food is cooked and ready.
	orderDeliveryStatus	Allows driver to update the status of delivery type orders whether it has "left the store", or "delivered" and includes estimated arrival time.
Payment System	paymentMade	updates the system whether or not the payment had been made.
Delivery System	driverStatus	Updates the application when the driver is on there way and provides live changes on the drivers location.
	foodDeliverd	Notifies the customer when the food has been delivered.
Employee Profile	hoursWorked	Displays hours a particular employee has worked in a particular payment cycle.
	tablesAssigned	Shows what tables the employees has been assigned recently.
	role	Role of the employee either food server, chef/cook, busboy, etc.

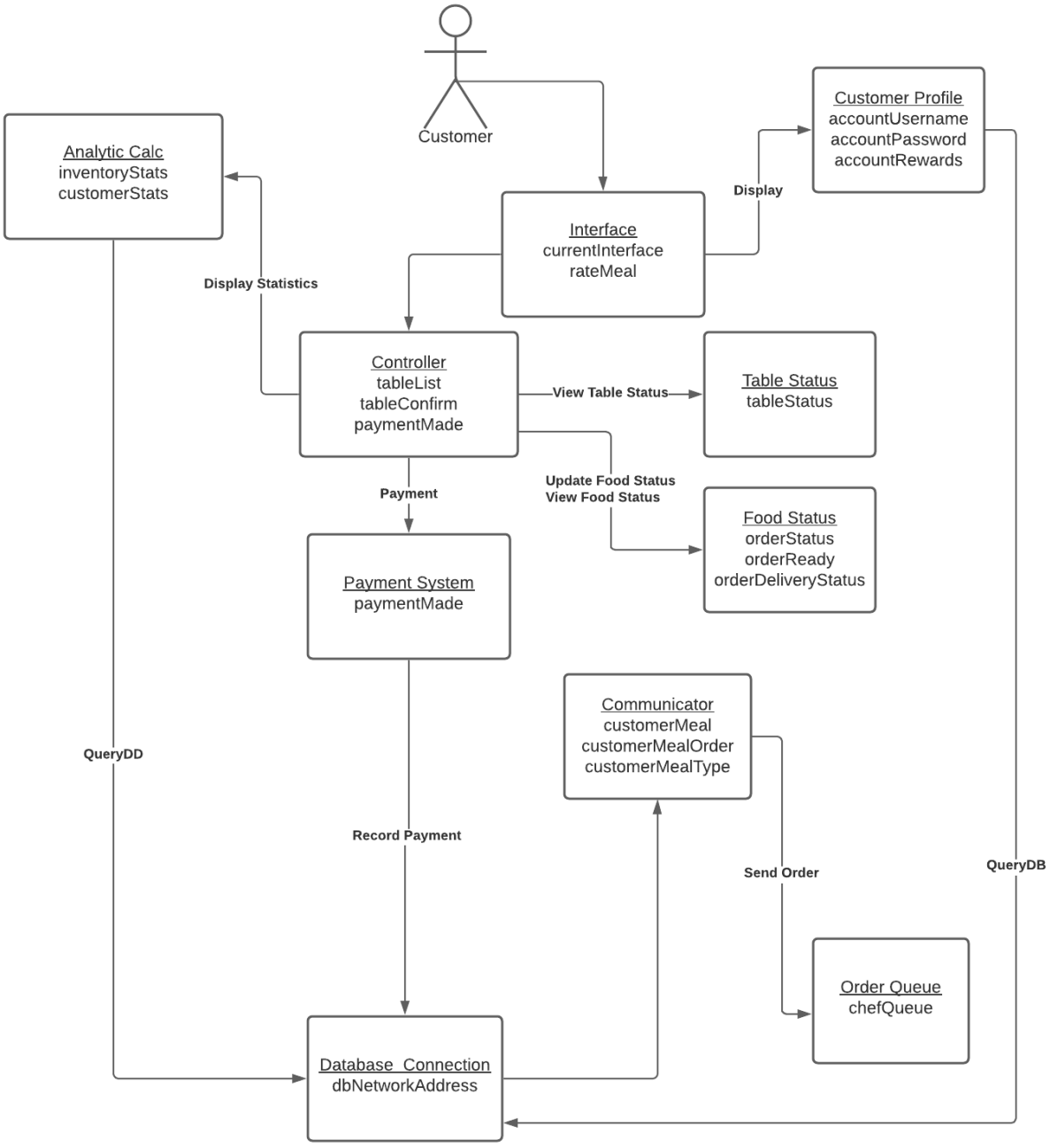
6.1.4) Traceability Matrix

Domain Concepts

Use Case	PW	Customer Profile	interface	Controller	Communicator	Order Queue	Analytic Calc	Table Status	Food STatus	Payment System	Delivery System	Employee Profile
UC-1	10		X	X	X	X	X		X	X	X	

UC-2	10		X	X	X	X		X	X		X	
UC-3	10			X					X		X	
UC-4	5		X	X								X
UC-5	5		X	X								X
UC-6	5			X								X
UC-7	5		X									
UC-8	10		X	X			X					X
UC-9	5	X	X	X								X
UC-10	5	X	X	X								X
UC-11	10	X	X	X						X		X
UC-12	10	X		X								X
UC-13	5	X		X								X
UC-14	10		X	X	X	X			X	X		
UC-15	15		X	X	X	X	X	X	X	X		
UC-16	10		X	X	X	X	X		X	X	X	
UC-17	10			X	X					X		
UC-18	5			X			X					
UC-19	10		X	X	X			X				
UC-20	10		X					X				
UC-21	5	X			X		X					X
Max PW	---	10	15	15	15	15	15	15	15	15	10	10
Total PW	---	40	120	150	80	55	55	45	65	65	40	65

6.1.5) Domain Model Diagram



Explanation: This is the domain model diagram. It lists various features and their relationship between each other. Functions are represented as arrows. The user is placed in association with the rest of the features on the diagram. This is how the network is presented. This diagram is a plan that will help us stay organized.

This data model could be represented using an ER diagram. The primary key is features listed within tables such as Controller, Database Connection, and Order Queue.

6.2) System Operation Contracts

Operation	UC-9 Login
Preconditions	● A user such as a customer or an employee has already created an account. ● The user knows their username and password.
Postconditions	● The customer is now able to order food. ● An employee can access their interface features.

Operation	UC-15 Dine-In
Preconditions	● The customer is logged into the application.
Postconditions	● The customer is shown choices to reserve the table. ● The customer is able to reserve the table using the application. ● The customer is notified of a successful table reservation.

Operation	UC-21 Reports
Preconditions	● The manager has logged into their account. ● There is data stored in the database to make the reports.
Postconditions	● The manager has access to all data models/graphs that they are interested in seeing. ● The manager can see revenues and profits.

6.3) Data Model and Persistent Data Storage

List of persistent objects:

Persistent Objects
Account email
Account password
Account address
Account type
Account (city, name, state, zip code)
Daily Traffic
Expenses
Food (description, image, menuId, name, price)
Food Stats (sales)
Category Menu (Image, Name)
Revenues (quarterly revenue)
Employee Schedule (weekly schedule)
Table Reservations (table #, time)

We are using Firebase to store all this information in a database. The file format we are using is JSON. Below is the database schema and format of the database tables. They are listed in the order they appear in the table above. Please note that we intend to fix the letters that preface the rest of the data entry.

Account email
Account password


```
"User" : {
  "Bo5lGJujlke4WwI7eqUtBRv9Kr22" : {
    "email" : "waiter100@gmail.com",
    "password" : "waiter100password",
    "user_type" : "waiter"
```

```
  },
 }
```

Account address
Account type
Account (city, name, state, zip code)

(Prototype. This code is yet to be implemented.)

```
 "Account" : {
  "cpN4sF9nMKXj3iir5qUlDBq4vZY2" : {
   "address" : "55 Stone St",
   "city" : "New Brunswick",
   "name" : "Ryan",
   "state" : "08901",
   "zipcode" : "NJ",
   “type” : “customer”,
  }
 },
```

Daily Traffic

```
 "DailyTraffic" : {
  "4-2-21" : {
   "10am" : 10,
   "10pm" : 16,
   "11am" : 12,
   "12pm" : 11,
   "1pm" : 5,
   "2pm" : 7,
   "3pm" : 8,
   "4pm" : 15,
   "5pm" : 13,
   "6pm" : 12,
   "7pm" : 25,
   "8pm" : 27,
   "9pm" : 23
  },
 },
```

Expenses

```
 "Expenses" : {
  "2021" : {
    "01" : 200000.12,
    "02" : 100234.12,
    "03" : 99999.99,
    "04" : 89999.99
  }
 },
```

Food (description, image, menuId, name, price)

```
 "Food" : {
  "01" : {
    "Description" : "A bagel with everything on it.",
    "Image" :
"https://www.abeautifulplate.com/wp-content/uploads/2018/05/everything-bagel-sandwich-1-5-6
00x900.jpg",
    "MenuId" : "01",
    "Name" : "Everything Bagel",
    "Price" : "699"
  },
 },
```

Food Stats (sales)

```
 "FoodStats" : {
  "Arnold Palmer Iced Tea (with Rum)" : {
    "01-Sales" : 12,
    "02-Sales" : 34,
    "03-Sales" : 45,
    "04-Sales" : 100
  },
 },
```

Category Menu (Image, Name)

```
 "Menu" : {
  "01" : {
    "Image" : "https://recipe-graphics.grocerywebsite.com/0_GraphicsRecipes/6117_4k.jpg",
    "Name" : "Sandwiches"
  },
 },
```

Revenues (quarterly revenue)

```
"Revenues" : {
  "2021" : {
    "01" : 126126.16,
    "02" : 131333.33,
    "03" : 124125.25,
    "04" : 123000.45
  }
},
```

Employee Schedule (weekly schedule)

```
"Schedules" : {
  "Chefs" : {
    "Chef1" : {
      "1-Monday" : "8am-2pm",
      "2-Tuesday" : "-",
      "3-Wednesday" : "-",
      "4-Thursday" : "-",
      "5-Friday" : "-",
      "6-Saturday" : "-",
      "7-Sunday" : "-"
    }
  }
},
```

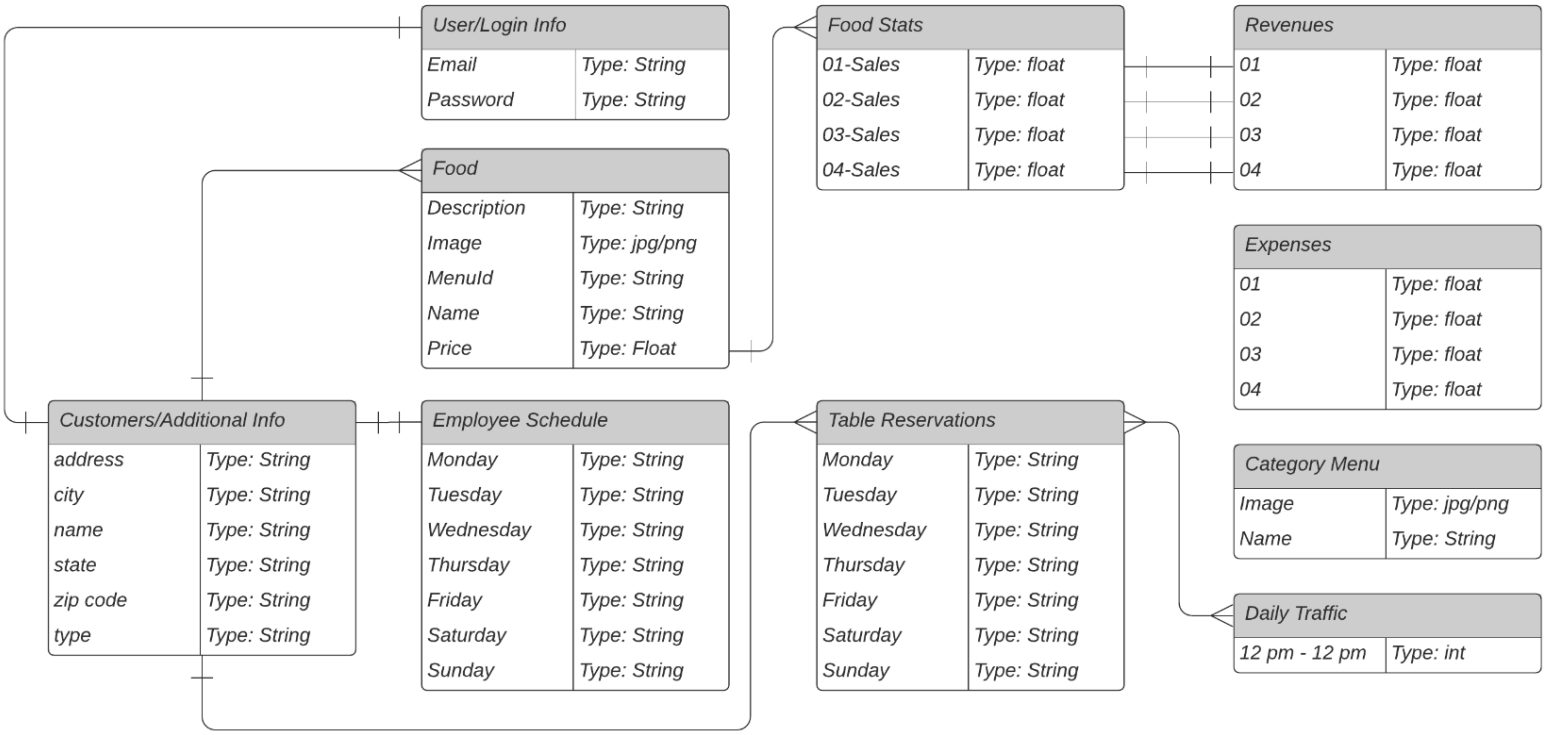
Table Reservations (table #, time)

(Prototype. This code is yet to be implemented.)

```
"Reservations" : {
  "Table1" : {
    "Eric" : {
      "1-Monday" : "11am-12pm",
      "2-Tuesday" : "-",
      "3-Wednesday" : "-",
      "4-Thursday" : "-",
      "5-Friday" : "-",
      "6-Saturday" : "-",
      "7-Sunday" : "-"
```

```
    }
  }
},
```

Database and Data Model ER Diagram (crow's foot)



Explanation of ER Diagram:

The tables above indicate information that must be stored by the application even after a single execution. The lines connecting different tables together indicate the relationship between those tables. Tables can have a one to one, one to many, or many to many relationship. The type of relationship can be determined by the marks on the connecting line. The first type of information that must be kept is the "User/Login" Info information. Upon registration the user will have the option to fill out additional information. The user may be required to fill out this additional information (such as address on the event of delivery) depending on how the user plans on using the system. This information is stored in the "Customers/Additional Info table". Despite containing the words 'Customer' in the table title, this table is meant for all account types. However, those accounts with the type 'customer' are eligible to order food items. One customer can order many foods. That is why there is a one to many relationship between the "Customers/Additional Info" table and the "Food" table. The "Customers/Additional Info" table is also applicable to employees. Every employee has an employee schedule. That is why there is a one to one relationship between the two tables. Normal customers have the benefit of reserving tables. They are able to do this once per reservation. As one customer can reserve many tables throughout the lifespan of their account there is a one to many relationship

between “Customers/Additional Info” and “Table Reservations”. At the end of this database chain, information in “Table Reservations” feeds into the “Daily Traffic” table. Many tables can be reserved and each time they are an update is made in the respective time period slot. This is why these two tables share a many to many relationship. On the other side of the database chain exists the one to many relationship. Every food item contributes to the quarterly sales statistic everytime it is ordered. For the purpose of advanced calculation the “Revenue” table has been included. This table interprets data from the “Food Stats” table in a one-to-one manner and subtracts costs to fill in this information. The last two tables are kept isolated. “Expenses” and “Category Menu” are kept available for the manager to edit.

Overview:

After logging out of the app the system stores persistent data. This will consist of each customer’s ratings. Customers personal information will be stored in the database. Each customer will be required to create a customer username which will store all the ratings under this username. In addition, personnel inventory and other information that the manager needs to know will be stored in the database. For personal inventory, there will be an entry for each food item and the amount of this. As orders are placed the inventory will be updated. This is so when the manager logs in the inventory will be viewable. Customers will also be able to choose which table is available and this will be stored in the database. During busy hours we will have a table that corresponds with customer peak times. Periodically this will be updated and stored in the database. Employee’s salary information and hours will also be stored within a database. Changes to the menu and statistics about the sale of each food item will be stored persistently. The amount of deliveries verse takeout verse dining in will be stored as well to be able to see the popularity of each to weigh where to allocate more resources towards.

6.4) Mathematical Model

```
Reservation(){
        Reservation.Time = getReservationTime();
        Table.Available = total number of tables - tables occupied;
        if(Table.Available = 0){
                Tell customer there are no available reservations
        }
        Table.Number = getTableNumber(); \\ The table customer reserved
        If(Table.status= 0){
                Prompt customer that the table is already occupied

                Else{
                        Table.status = 0
                        // Makes the table unavailable to be reserved by other customers
                }
        }
```

```
}
```

This model checks if there are any available tables for a certain time that the customer wants to reserve. If there are none, the customer is prompted with a message that there are no available reservations for that particular time and they would have to select another time. If there are available tables at that time, the customer chooses a table and its status is checked. If it is not occupied then the system will mark it occupied so that no other customer can reserve it for that time.

```
Delivery(Food Order){
        //Find an available driver
        //might assign a driver a couple different orders
        Driver = getAvailableDriver();
        //Send the driver a notification about a customer's order
        Driver.assign(Food Order);
        //Driver updates order status as delivered or not
        Driver.update(Order1, bool)
}
```

When a customer selects delivery and the chef is done making that order order,there is a list of kitchen orders ready to be delivered that a driver can see on their screen after the chef updates it. A driver can look at the current list of orders that are ready to be delivered and accept an order. Once they accept it, the order is removed from the kitchen list and it will be shown in the drivers current orders. The driver can then mark an order completed to update the system when they finish their oder.

```
Payroll() {
        For each employee in a payroll period:
                getEmployeeHours();
                getHourlyPayRate();
                calculatePay;
                Employee.UpdatePay(monthlyPay)
                UpdateDatabase for later analysis of profits
}
```

The model for payroll is for each employee in the restaurant, the number of hours the employee worked that week and their hourly pay is retrieved from the database to find their weekly pay. Their weekly pay is updated to the database.

```
Payment(Food Order) {
        //Initial Pay 0
        TotalPrice = 0;
        MenuPrice = 0;
        addMenu = 1;
```

```
        while (addMenu == 1) {
                MenuPrice = getMenu();
                TotalPrice = TotalPrice + MenuPrice;
                //Customer selects whether to add more foods. (addMenu = true or false)
                addMenu = addMoreMenu();
        }
        Output the total price to the customer
        //Deduct the amount of the price from customer's account
        if (TotalPrice <= Customer.balance) {
                if (Customer.rewardUse == true) {
                        Customer.rewards = Customer.rewards - TotalPrice;
                }
                Customer.balance = Customer.balance - TotalPrice;
                //If successfully paid, notify the customer of the successful payment
                Customer.payment(true);
                Customer.rewards = Customer.rewards + (0.01)*TotalPrice;
        }
        //If the customer's balance is below the total price, do not process the payment
        else {
                Customer.payment(false);
        }
}
```

This is payment for a food order using customer rewards. The customer rewards is a balance that is registered in the system for a customer. As a customer adds items to their order, the menu price of that item is retrieved and as is used to update the total price. Once finished, the total price is shown to the customer and the system will check if the customer has a large enough rewards balance to pay for the order. If they do, the total price is subtracted from their rewards balance and their rewards balance gets increased a little for spending money at the restaurant. The customer is then notified that their payment has been processed. If the rewards balance is not enough to pay for the order then the order is not processed.

```
Analysis(){
        //add food items in orders to database
        For(int i=0; i< order.length(); i++)
                databasefooditems.update(order.item(i));

        //call graph function to display graphs based off data in database
        graph(fooditemsPerWeek[][]);
}
```

This is a method to graph the data on food orders. At the current moment, the analysis for food items is based on the number of times the item was ordered by month since January.

```
Scheduling(){
        //Scheduling done by weekly basis
```

```
        //Employee Object that has name, availability, full/part time, job position
        //Array of employee objects, sorted by reliability (most reliable/best employee listed first)
        /*Schedule object has array inside called Schedule.day, and day array contains an
        object for each day in week, day[0] is monday, day[6] is sunday. */

        for (int i =0; i<employeeArray.length();i++){
                Employee employee = employeeArray[i];
                for(int i = 0; i<7;i++){ //i = 0 is Monday, i = 1 is Tuesday, …. i = 6 is Sunday
                        if(employee.isAvailable()){
                                Schedule.day[i].add(employee);
                                /*also have to check if enough people scheduled for day
                                before adding anymore employees to day to avoid
                                over scheduling a day */
                        }
                }
                employee.sendNotification(); //texts employee their schedule
        }
}
```

The manager has screens for making and editing the schedules for the employees separately by category. In other words, the manager has a screen to specifically edit chef schedules, driver schedules, and so on. On the screen, the manager will select an employees name and fill out the start time, end time , and day. Once ready, the manager selects an update, and this information is sent to the database. Once done, when an employee logs in , then they can check their schedule and they will be able to see this information.

Ordering(){

```
//create Order Object
Order customerOrder = new customerOrder();

while user is not done ordering{

        if(user clicks add item)
                customerOrder.addItem()
        if (user clicks remove item)
                customerOrder.removeItem(specific item)
        If (user clicks on customize options)
                customerOrder.item.update()
}
```

When ordering, an order object is created. This object gets updated as a user adds more items to their order or removes them.

employeeClock(){

```
Class employee:
```

```
        String clockin;
        String clockout;

        //create an array of class employee
        clock_in(current_time, index){ //index associated with employee id
                //update array at id/index with current time
        }

        clock_out(current_time, index){ //index associated with employee id
                //update array at id/index with current time
        }

        //display all employees clock in and clock out times at end of business day to manager
        //clear array at the start of a new business day
}
```

When an employee logs in, they have an option to clock in and clock out. When they clock in the method retrieves the current time and registers that into the system. The same thing happens for clocking out.

```
foodOrderStatus(){ //track the status of food
        chefUpdate(int customerOrderID, int degreeOfCompletion){ //available to Chef
                //Access database of orders. Default set to 0
                //0 - pending, 1 - preparing, 2 - almost finished, 3 - complete
                //update database
        }
        displayOrderStatus(){
                //displays order status to requesting customer
        }
}
```

For a food order, the chef has the option of updating the status field for this order. Orders have 4 states: pending, preparing, almost finished, and complete. When a customer wants to know the status of their order, the state of the order is retrieved through the database through an order id assigned to that particular order.

```
updateMenu(){ //update menu
                //Privileged users can add or remove items
        shortcutUpdateMenuDatabase(int menuItemID, int action){
                //access menu database and modify menu item to be displayed or removed
                //0 - remove, 1 - add
        }
        addItemToMenuDatabase(int menuItemID, String title, String description){
                //access and update menu database
                //adds new menu item
```

```
        }
        removeItemToMenuDatabase(int menuItemID, String title, String description){
                //access and update menu database
                //removes menu item from database
        }
}
```

When the manager wants to remove an item from the menu, they must provide the menu item and the action remove. This will update the database by removing the item. If they want to add something to the menu, then they have to provide an id, title, and description of the new item (like price) , so that it can be added to the database.

```
displaySale/Profit(){ //calculate total sales/profit over the span of a day and display
        global int daily_cost;

        updateDailyCost(int new_daily_cost){
                daily_cost = new_daily_cost;
        }
        accessReceiptDatabase(String date){ /*each receipt automatically appended to
        database with important details */
        //access receipts within the database for a certain date
        //return database with receipts specific to a date (in table/database format)
        }
        displaySales(String date){
                cutDatabase=accessReceiptDatabase(date);
                int sales;
                countDatabase(){
                        for(i=0;i++;i<length(cutDatabase[“Amount”])){
                        sales += cutDatabase[“Amount”][i];
                }
        return sales;
        }
        displayProfit(int sales){
        int profit;
        return profit=sales-daily_cost;
        }
}
```

In the current implementation profits, sales, and expenses are shown by month. The data is retrieved through an array list to use as the arguments for the graphs, Ideally, when a food order is processed it should update by adding that amount to the revenues for the same month in the database.

```
/*Checks if low inventory of a specific food item,
Food object as input*/
checkInventory(Food food){
        int desiredAmount = food.desired; //checks desired amount before restock
        int amount = food.amount;

        if (amount <= desiredAmount){
                food.needRestock = true;
                return;
        }
        food.needRestock = false;
        return;
}
```

When a food item is added to order, the system checks if there are enough ingredients to use. Once a certain threshold is reached, then a restock is needed and the manager will be notified,

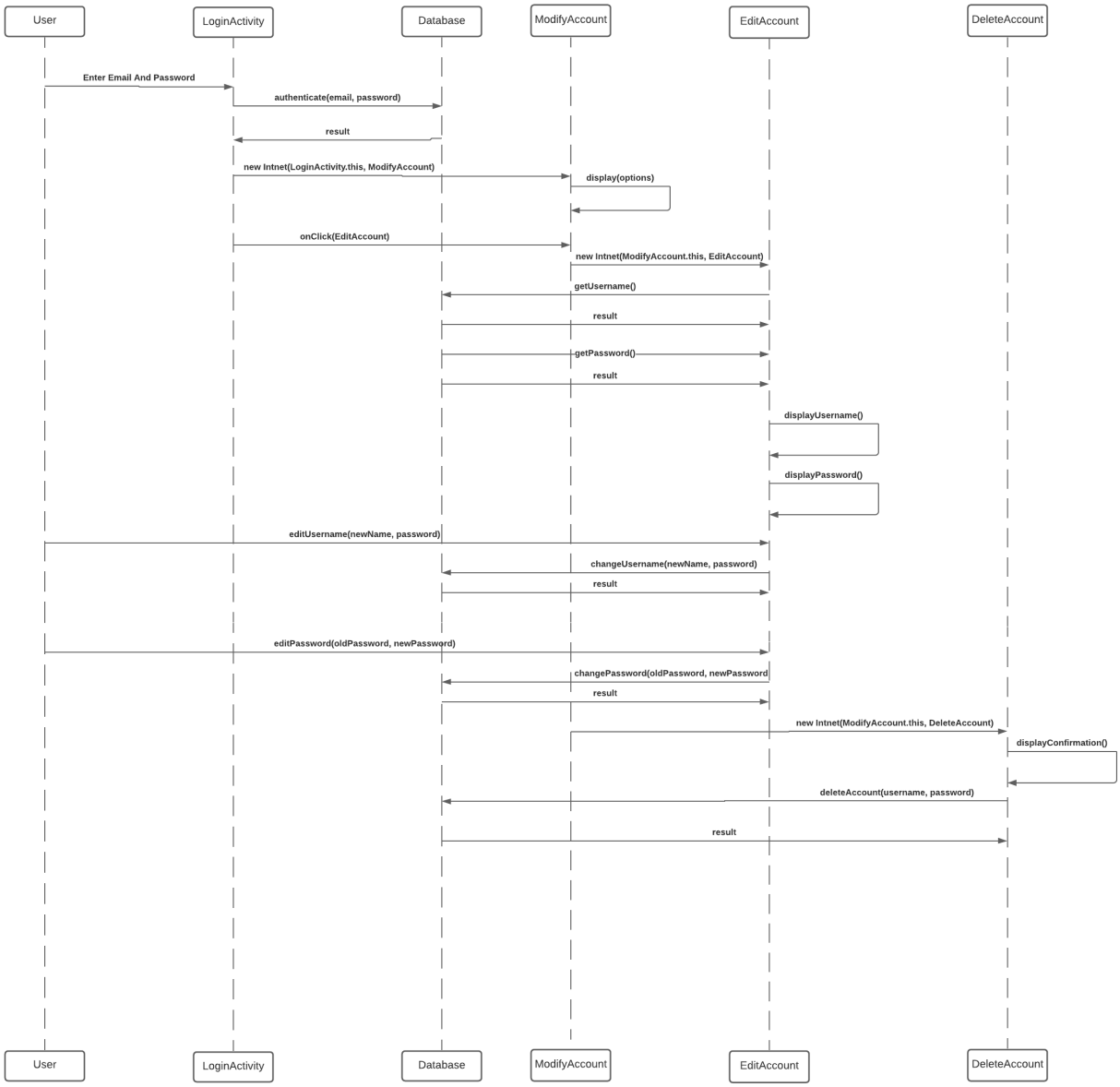
```
logInAccount(){ //Mathematical part consists of the failed login attempts
        String username; // user inputted username
        String password; // user inputted password
        int attempts = 5;
        while(attempts>0){
        if (isUserInDB(username) &&isPassInDB(password)){ /*Checking if username and
        password entered is in database */
                Login();
                return;
        } else {
                display “username or password is incorrect, ” + attempts + “ attempts remaining”;
                attempts --;
        }
        }
        display “out of attempts, want to reset password?”;
        return;
}
```

To login, the app will check the database to make sure that the user already has an account in the system. If an account is registered and the password is wrong, then the attempted number is decreased. After a certain number of times, the app will notify the user for a password reset.

7.Interaction Diagrams

Edit/Delete Account UC

https://lucid.app/lucidchart/invitations/accept/a7c2418f-c352-46b8-a5fa-fa278ff04758

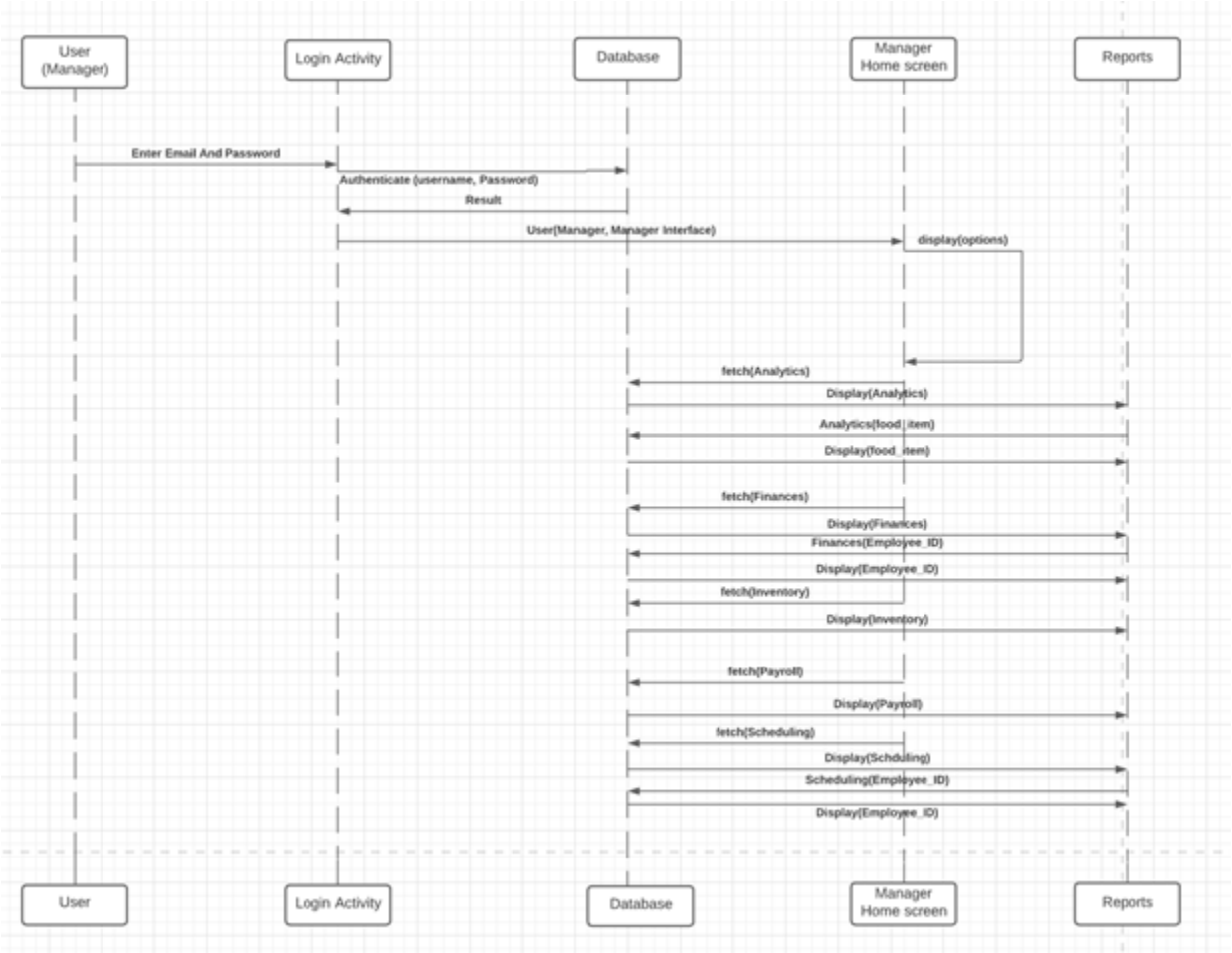


This diagram portrays the interactions between classes and Use Cases 12 and 13, also known as, Delete And Edit. These two use cases can be combined into one interaction diagram since they will be both located in the same area, with the difference being each has their own separate button which functions differently. After logging into the app, the user can go to the “Modify Account” section of their app. Once they are in this section, they will be able to change their username or password, or delete their account completely. Modifying your account will always require your old password as a confirmation before changing any of your account information. Modifying your account will also give you a confirmation before changing your account in any way in case the user has second doubts about it.

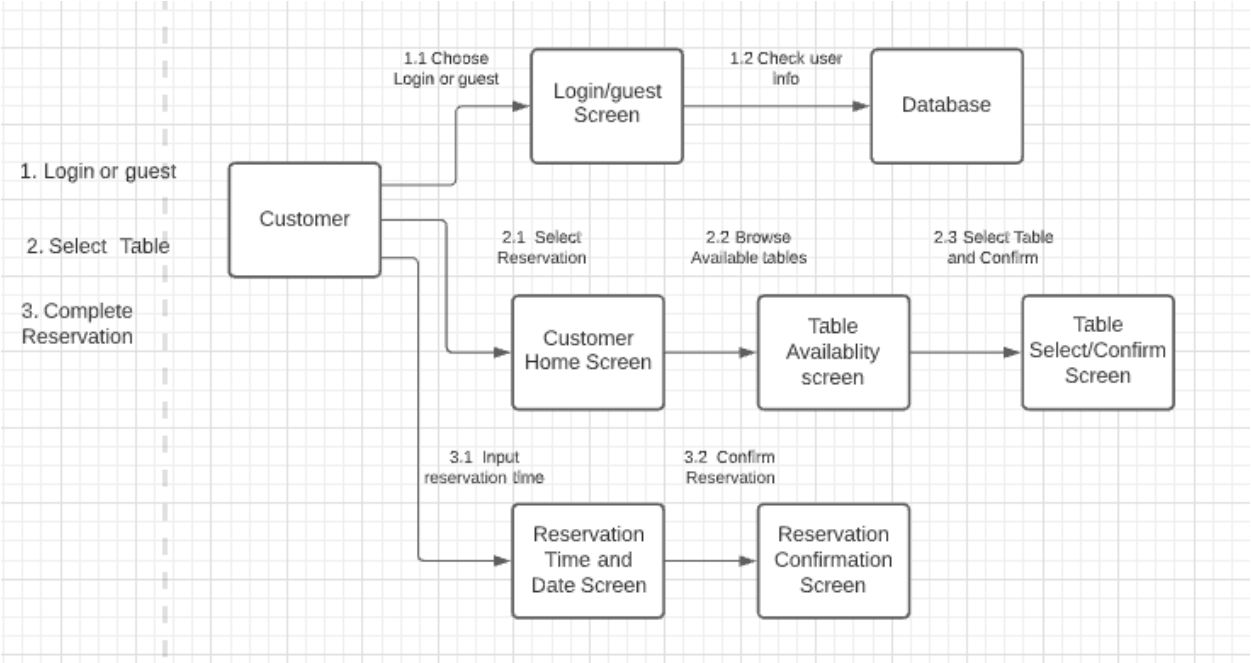
A design principle used in this would be the low coupling and high cohesion principle where each class has a specific and small amount of tasks to accomplish and is well defined. For example, ModifyAccount only handles EditAccount and DeleteAccount, and EditAccount handles only modifications to the username and password while DeleteAccount only handles removing all user data from the database.

Reports UC

https://lucid.app/lucidchart/invitations/accept/0603f786-15c1-4304-99cb-0bac3114bbe2?viewport_loc=-1768%2C-1348%2C3072%2C1512%2C0_0



This interaction diagram shows how the manager may use the app to do various tasks such as scheduling, inventory, analytics, etc. When the manager logins, the app first must authenticate the manager login credentials. After this manager goes to the manager home screen where he/she is presented with various options. The manager can click on Analytics, Finances, Inventory, Payroll, and scheduling.



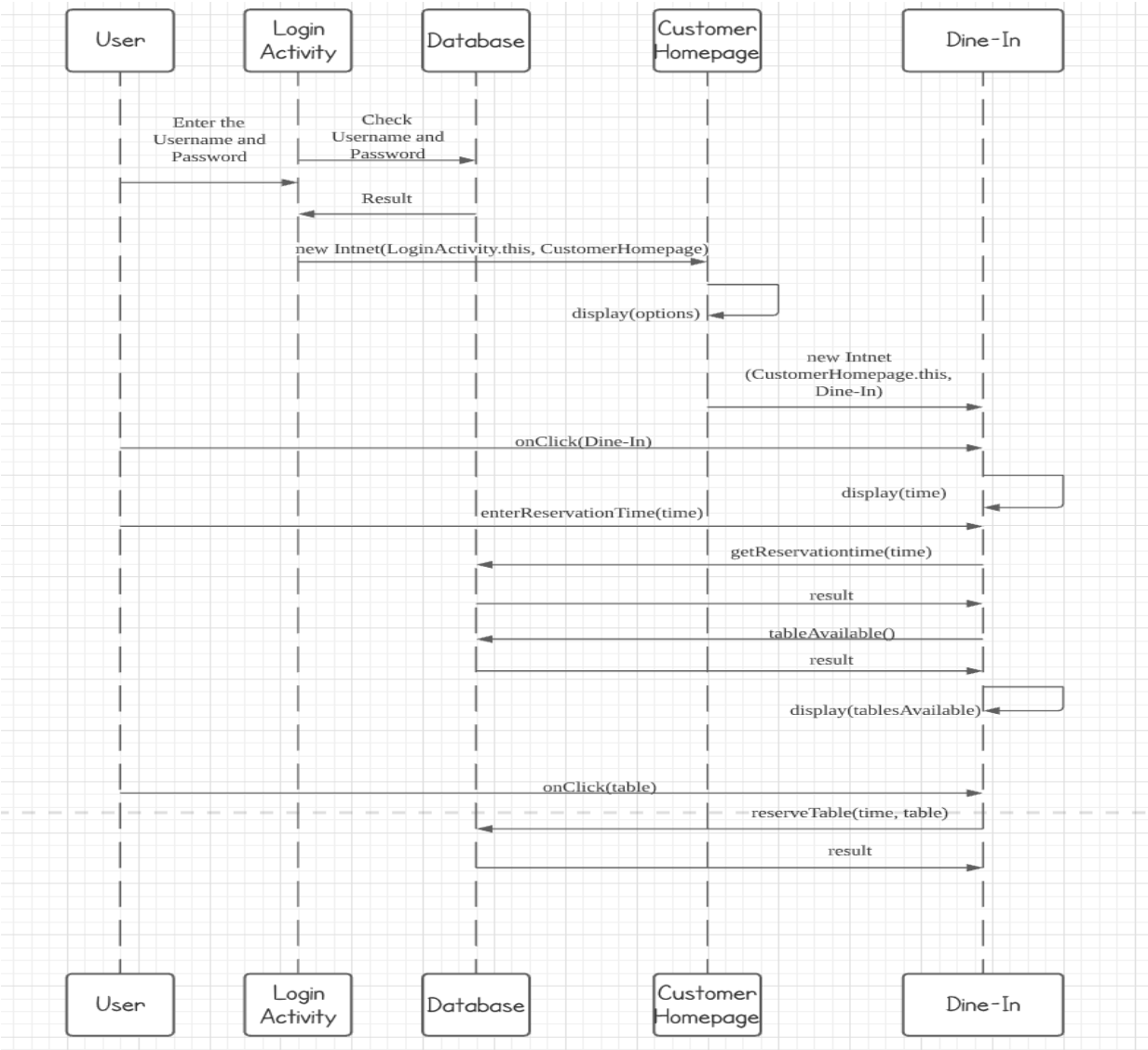
This communication diagram represents the interactions in the Reservation Use Case. As can be seen, there are three main parts to this UC. The user/customer can choose to login to their account or choose to stay as a guest user on the login screen. The interface will then (behind the scenes) access the database to check the user info in the case they input login info.

Once this step has been completed, the customer will then be brought to the customer home screen, where they will then be able to access the table availability screen. They will then be brought to a table select interface where they can select which table they want.

Once they have chosen their table, they will be brought to a screen where they can input the reservation date and time, and then to a confirmation screen to confirm the entire reservation.

Dine-In UC

https://lucid.app/lucidchart/e69607b9-ca7d-43d4-bf97-8a77df827879/edit?page=0_0#



This diagram shows the interaction between classes and Use Case 15, Dine-In. After logging into the customer homepage, the user can select the “Dine-In” option Once the user is in the Dine-In page, the user can select the time preferred to dine-in in the restaurant. Then the availability for each table in the selected time would be displayed. The user can select the table to reserve, and the application will show whether the table is successfully reserved in the selected time or not.

7.1 Design Patterns

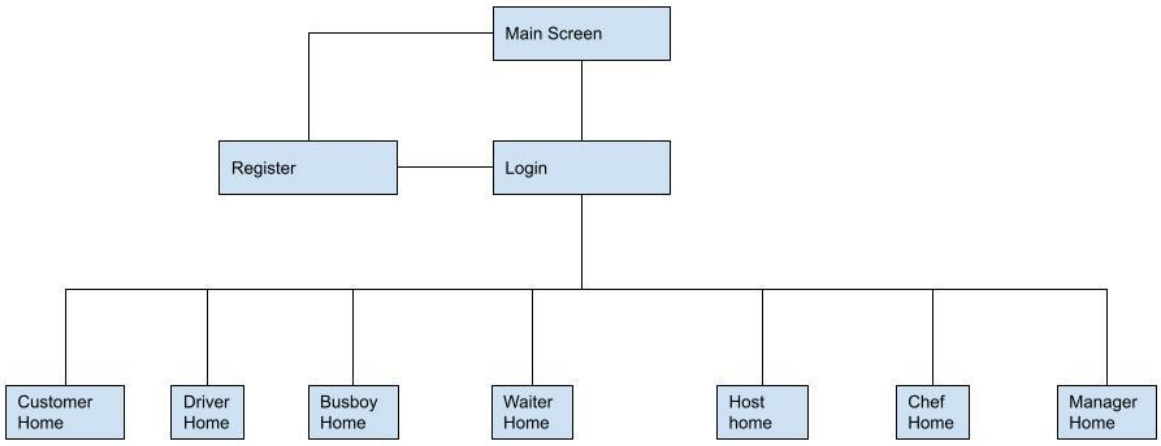
Several design patterns are accounted for in the Why W9 application. First, the command pattern is used in the application in order to separate the preparation of calling parameters from the passing program control. Through the command pattern, the execution part is separated from the parameter preparation section. As business logic is in the execution area, now the business logic has been decoupled from parameter preparation. Therefore, client and

custodian codes can be evolved by different developers independently. As a result, the client only needs to decide when to execute each input of the application. Thus, as the command interface never changes, server changes do not force the client changes.

Secondly, the state pattern is used in order to separate the state-dependent event handling functions from each other. Using the state pattern, new states and new events can be easily implemented without changing any of the current state status. The event-handling object would externalize all of its state-dependent functionality into different state objects. Those state objects look identical to each other as they implement the same interface. Each of the state objects knows the next state, and the method handle for each state would return a following state.

8.Class Diagrams and Interface Specification

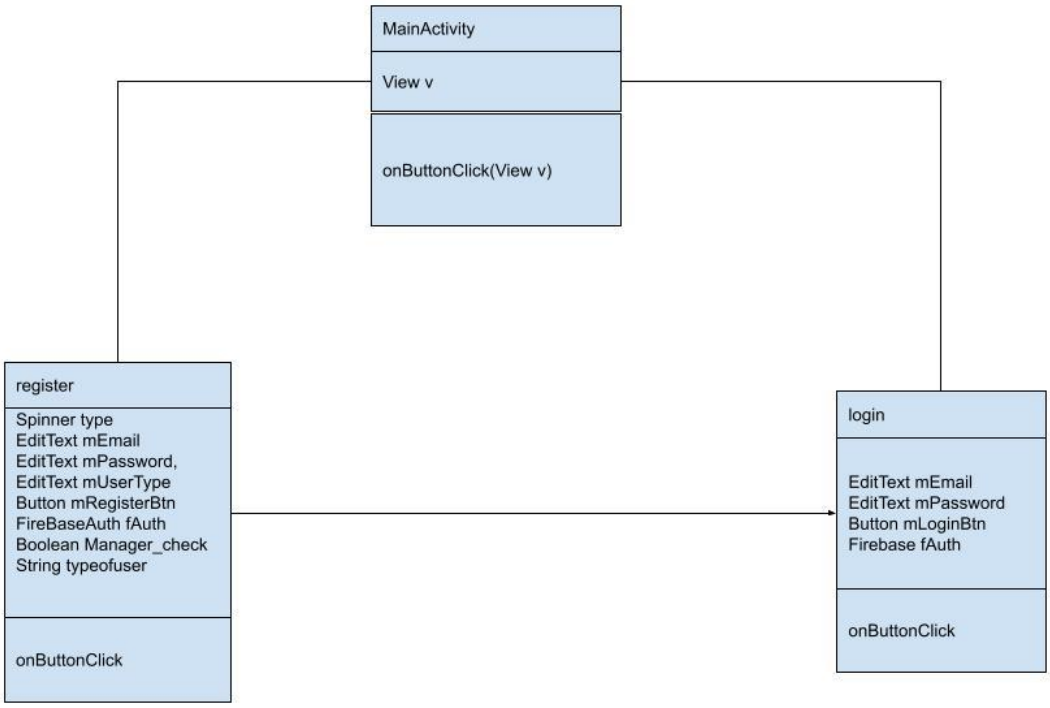
8.1 Class Diagram

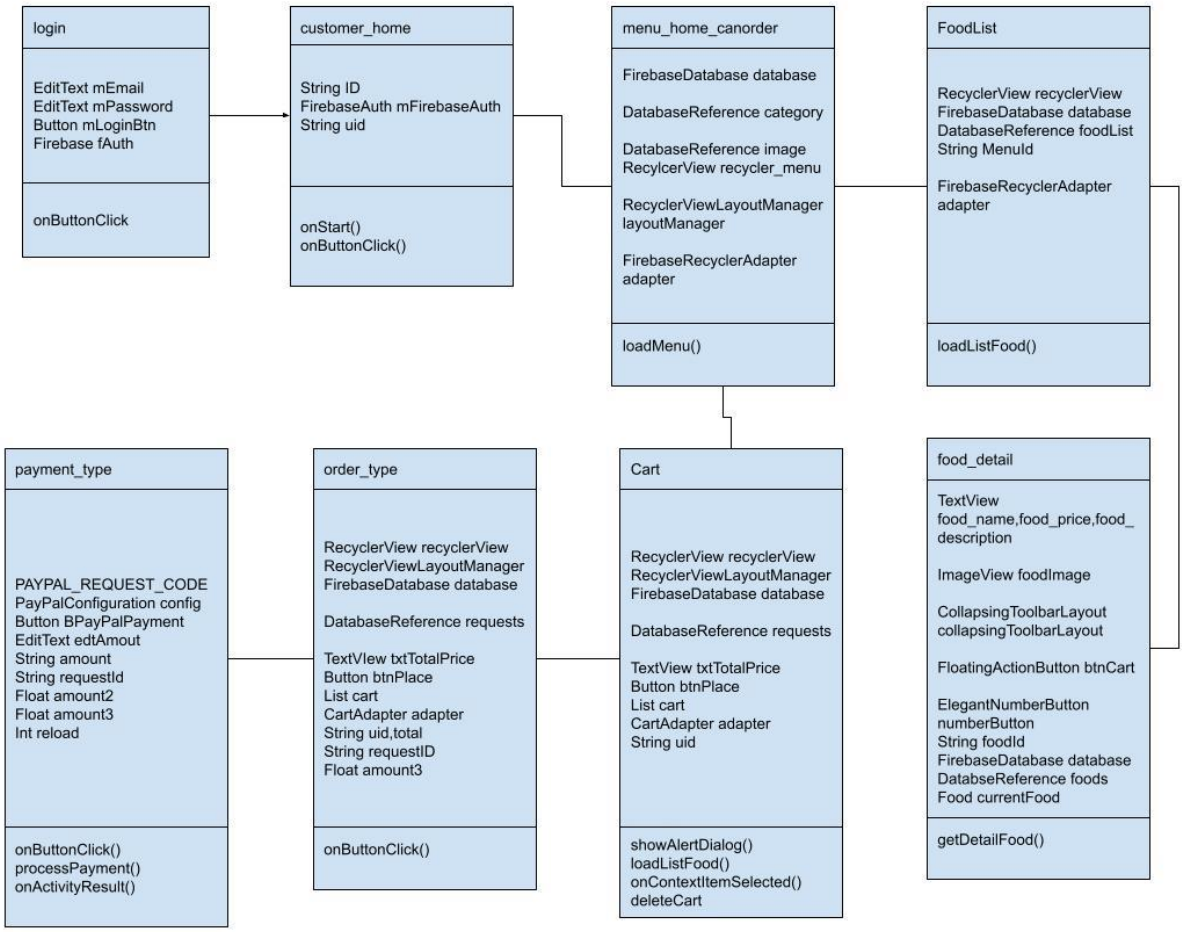


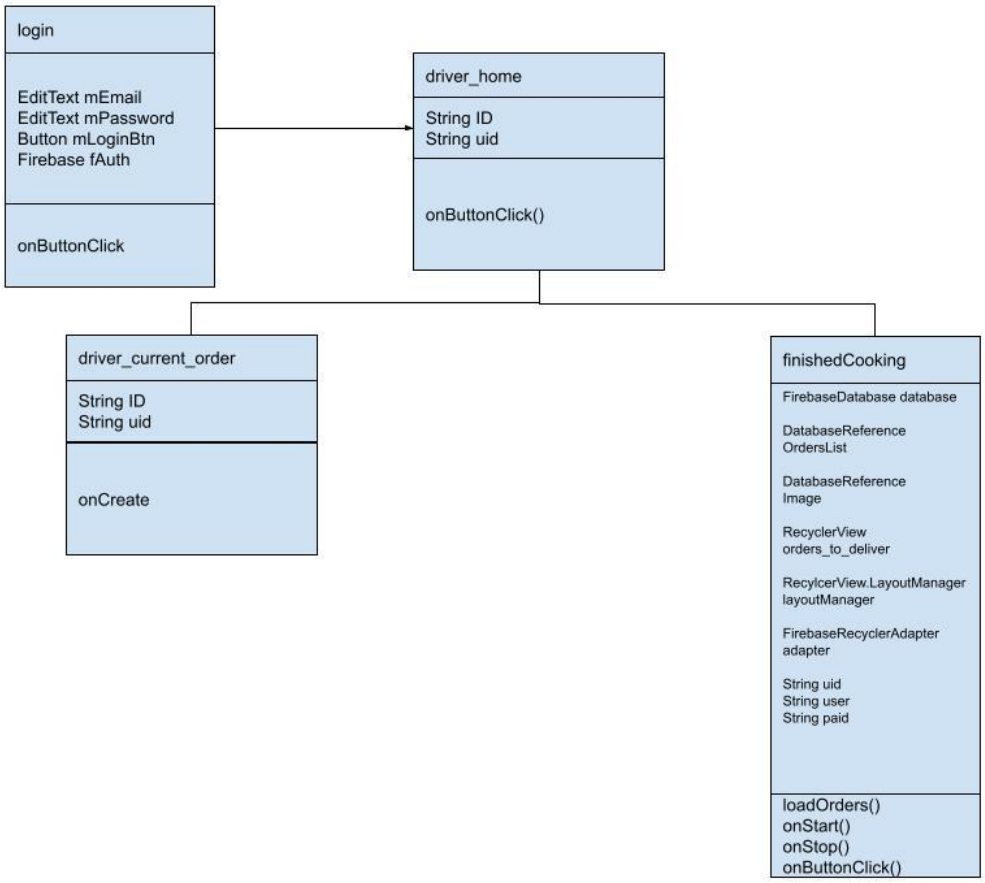
MainScreen - This is the main screen. All users will be prompted to enter through this screen.
Login - User logs in and credentials are checked. If verified, the next screen is brought up.
Customer - customer account. Allows for different privileges.

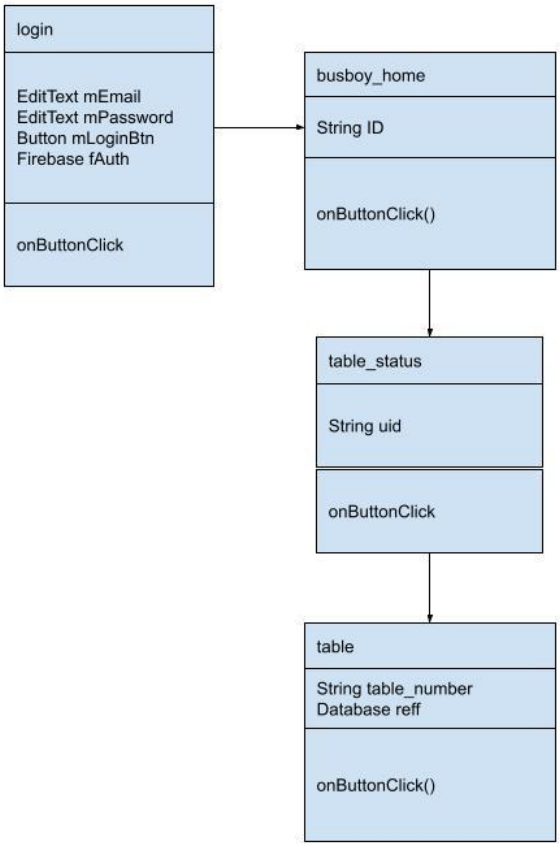
Driver - driver account. Driver is responsible for delivering orders to customers.
Busboy - busboy account. Busboy is responsible for cleaning up tables.
Waiter - waiter account. Waiter is responsible for taking orders from customers who have been seated.
Host-host account.
Chef - chef account. Chef is responsible for viewing orders and preparing them.
Manager - manager account. Manager has the privilege of overseeing all features appropriated

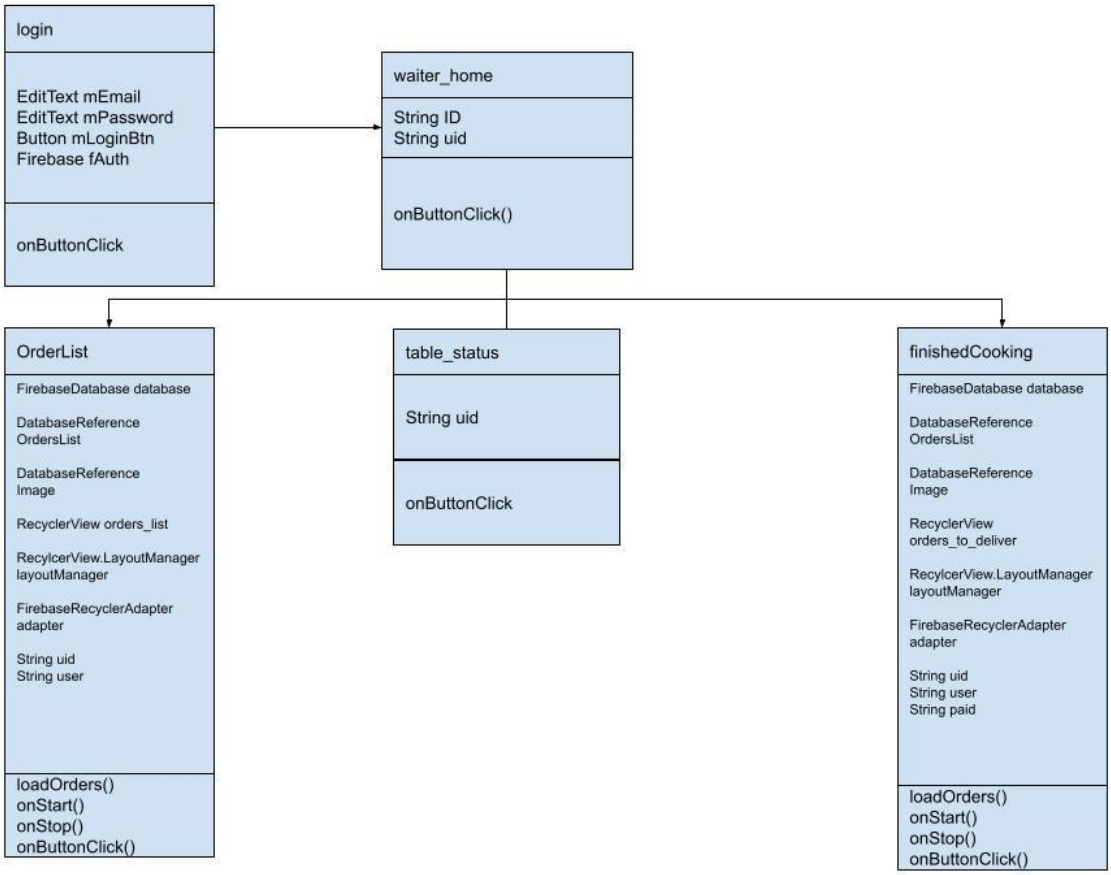
The above class diagram shows the different activities or user screens, which are classes themselves, in our Android Studio application. Depending on what button the user presses on their screen we can call a method within that activity to take the user to another screen.

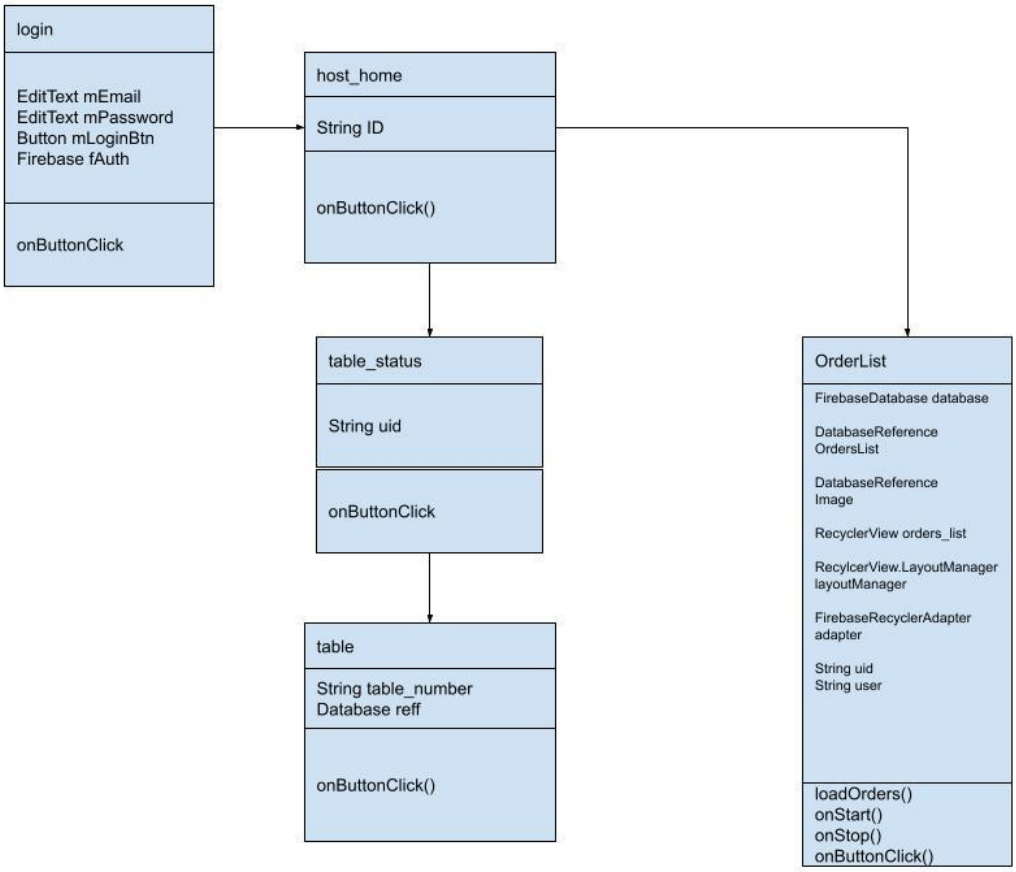


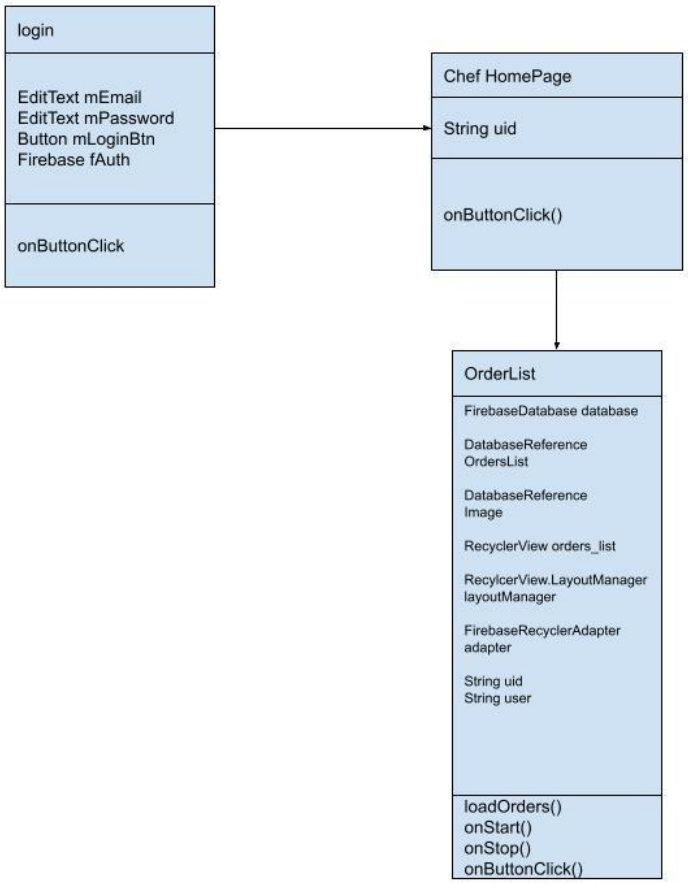

login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
customer_home
String ID
FirebaseAuth mFirebaseAuth
String uid
onStart()
onButtonClick()
menu_home_canorder
FirebaseDatabase database
DatabaseReference category
DatabaseReference image
RecylcerView recycler_menu
RecyclerViewLayoutManager layoutManager
FirebaseRecyclerAdapter adapter
loadMenu()
FoodList
RecyclerView recyclerView
FirebaseDatabase database
DatabaseReference foodList
String MenuId
FirebaseRecyclerAdapter adapter
loadListFood()
payment_type
PAYPAL_REQUEST_CODE
PayPalConfiguration config
Button BPayPalPayment
EditText edtAmout
String amount
String requestId
Float amount2
Float amount3
Int reload
onButtonClick()
processPayment()
onActivityResult()
order_type
RecyclerView recyclerView
RecyclerViewLayoutManager
FirebaseDatabase database
DatabaseReference requests
TextVIew txtTotalPrice
Button btnPlace
List cart
CartAdapter adapter
String uid,total
String requestID
Float amount3
onButtonClick()
Cart
RecyclerView recyclerView
RecyclerViewLayoutManager
FirebaseDatabase database
DatabaseReference requests
TextView txtTotalPrice
Button btnPlace
List cart
CartAdapter adapter
String uid
showAlertDialog()
loadListFood()
onContextItemSelected()
deleteCart
food_detail
TextView food_name,food_price,food_description
ImageView foodImage
CollapsingToolbarLayout collapsingToolbarLayout
FloatingActionButton btnCart
ElegantNumberButton numberButton
String foodId
FirebaseDatabase database
DatabseReference foods
Food currentFood
getDetailFood()

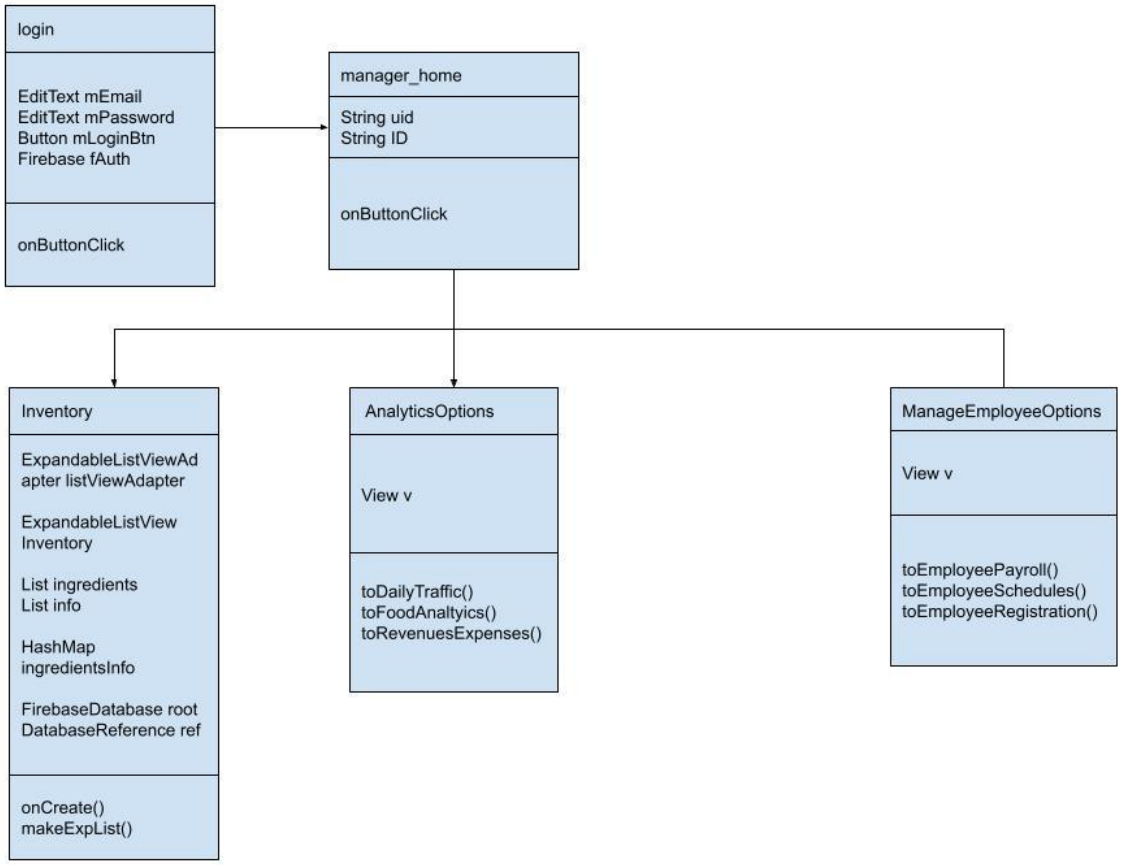

login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
driver_home
String ID
String uid
onButtonClick()
driver_current_order
String ID
String uid
onCreate
finishedCooking
FirebaseDatabase database
DatabaseReference
OrdersList
DatabaseReference
Image
RecyclerView
orders_to_deliver
RecylcerView.LayoutManager
layoutManager
FirebaseRecyclerAdapter
adapter
String uid
String user
String paid
loadOrders()
onStart()
onStop()
onButtonClick()


login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
busboy_home
String ID
onButtonClick()
table_status
String uid
onButtonClick
table
String table_number
Database reff
onButtonClick()


login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
waiter_home
String ID
String uid
onButtonClick()
OrderList
FirebaseDatabase database
DatabaseReference OrdersList
DatabaseReference Image
RecyclerView orders_list
RecylcerView.LayoutManager layoutManager
FirebaseRecyclerAdapter adapter
String uid
String user
loadOrders()
onStart()
onStop()
onButtonClick()
table_status
String uid
onButtonClick
finishedCooking
FirebaseDatabase database
DatabaseReference OrdersList
DatabaseReference Image
RecyclerView orders_to_deliver
RecylcerView.LayoutManager layoutManager
FirebaseRecyclerAdapter adapter
String uid
String user
String paid
loadOrders()
onStart()
onStop()
onButtonClick()


login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
host_home
String ID
onButtonClick()
table_status
String uid
onButtonClick
table
String table_number
Database reff
onButtonClick()
OrderList
FirebaseDatabase database
DatabaseReference OrdersList
DatabaseReference Image
RecyclerView orders_list
RecylcerView.LayoutManager layoutManager
FirebaseRecyclerAdapter adapter
String uid
String user
loadOrders()
onStart()
onStop()
onButtonClick()


login
EditText mEmail
EditText mPassword
Button mLoginBtn
Firebase fAuth
onButtonClick
Chef HomePage
String uid
onButtonClick()
OrderList
FirebaseDatabase database
DatabaseReference OrdersList
DatabaseReference Image
RecyclerView orders_list
RecylcerView.LayoutManager layoutManager
FirebaseRecyclerAdapter adapter
String uid
String user
loadOrders()
onStart()
onStop()
onButtonClick()



8.2 Data Types and Operation Signatures

MainActivity Class

Attributes:

- View v:Used to identify the button on the screen that the user pressed.

Methods:

- onButtonClick():Once we identify the button user pressed, this will take the user to the register or login screen.

Register Class

Attributes:

- Spinner type : Visible only to the manager when registering an employee to the app.
- EditText mEmail : email of user.
- EditText mPassword : password of user.
- EditText mUserType : Type of employee field visible only to the manager.
- Button mRegisterBtn : Button used to register once every field is filled.
- FireBaseAuth fAuth : For database authorization.
- Boolean Manager_check: Boolean to check if manager is registering an employee.
- String typeofuser - type of user on the screen.

Methods:

- onButtonClick():Register new user to the database with the specified account type.

Login Class

Attributes:

- EditText mEmail:Users email.
- EditText mPassword:Users password.
- Button mLoginBtn:Button to click when user finishes typing their credentials.
- Firebase fAuth: Instance of firebase to authorize accounts.

Methods:

- onButtonClick():Login user (customer or employee) to their home screen.

customer_home Class

Attributes:

- String ID:customer id in the database
- FirebaseAuth mFirebaseAuth:authentication
- String uid:customer id

Methods:

- onButtonClick():Method that identifies the button that the customer pressed on the screen and then shows the customer the correct screen based on the button. Will take the customer to view the menu, their cart, and other customer features.

driver_home Class

Attributes:

- String ID:driver id in the database
- String uid:driver id to get schedule of driver in the database.

Methods:

- onButtonClick():Method that identifies the button that the driver pressed on the screen and then shows the driver the correct screen based on the button. Will either take the driver to the screens that show the available orders, current order information , their schedule, clock in/out, etc.

busboy_home Class
Attributes:

- String ID:busboy id in the database

Methods:

- onButtonClick():Method that identifies the button that the busboy pressed on the screen and then shows the busboy the correct screen based on the button. Will either take the busboy to the screens that show the table status, schedule, etc.

waiter_home Class
Attributes:

- String ID: waiter id in the database

Methods:

- onButtonClick():Method that identifies the button that the waiter pressed on the screen and then shows the waiter the correct screen based on the button. Will either take the waiter to the screens that show the orders list, table status, schedule, etc.

host_home Class
Attributes:

- String ID: host id in the database

Methods:

- onButtonClick():Method that identifies the button that the host pressed on the screen and then shows the host the correct screen based on the button. Will either take the host to the screens that show the orders list, table status, schedule, etc.

chef_home Class
Attributes:

- String ID: chef id in the database

Methods:

- onButtonClick():Method that identifies the button that the chef pressed on the screen and then shows the chef the correct screen based on the button. Will either take the chef to the screens that show the orders list, menu, etc.

manager_home Class
Attributes:

- String ID: manager id in the database

Methods:

- onButtonClick():Method that identifies the button that the manager pressed on the screen and then shows the manager the correct screen based on the button. Will either take the manager to the screens that show the inventory, analytics options, employee management, etc.

8.3 Traceability Matrix

Domain Concepts

Classes	Customer Profile	Interface	Controller	Communicator	Order Queue	Analytic Calc	Table Status	Food Status	Payment System	Delivery System	Employee Profile
Customer	x	x	x					x	x	x	
Manager		x	x	x		x	x				x
Chef		x	x					x			x
Driver		x	x	x						x	x
Busboy		x	x				x				x
Waiter		x	x	x			x	x	x		x
Menu	x	x	x		x				x	x	x
MenuAdapater											
Reservation Time	x		x		x						x
Payment	x		x			x			x		
Table Availability		x	x				x				x
LoginServerRequest			x								
LoginActivity											
MainScreen	x	x									x
ManagerStatistics						x					x
ManagerEd		x									x

itEmployee											
ManagerEditItem		x									x
ManagerProfitView						x					x
ManagerInventoryView						x					x

- Customer Profile
    - Customer - Customer Profile labels login as customer and provides relevant information
    - Menu - Customer Profile creates instance of menu suitable for customer view
    - ReservationTime - Customer Profile displays ReservationTime for customer
    - Payment - Customer Profile allows option for Payment.
    - MainScreen - Customer Profile creates an instance of MainScreen suitable for customer view.
- Interface
    - Customer - Customer has access to interface
    - Manager - Manager has access to interface
    - Chef - Chef has access to interface
    - Driver - Driver has access to interface
    - Busboy - Busboy has access to interface
    - Waiter - Waiter has access to interface
    - Menu - Waiter has access to interface
    - Table Availability - Table Availability updated through interface
    - MainScreen - MainScreen is displayed as an navigable interface
    - ManagerEditEmployee - Interface to provide editing capabilities
    - ManagerEditItem - Interface to provide editing capabilities
- Controller
    - Customer - controller allows relevant data provided by the customer to pass
    - Manager - controller allows relevant data provided by the manager to pass
    - Chef - controller allows relevant data provided by the chef to pass
    - Driver - controller allows relevant data provided by the driver to pass
    - Busboy - controller allows relevant data provided by the busboy to pass
    - Waiter - controller allows relevant data provided by the waiter to pass
    - Menu - controller allows for data such as selection to pass
    - ReservationTime - controller requires reservation time to be set
    - Payment - controller requires payment to be set
    - TableAvailability - allows table availability to be set
    - LoginServerRequest - allows entered and requested data to pass
- Communicator
    - Manager - communicator allows manager to contact employees

    - Driver - communicator allows driver to contact customer
    - Waiter - communicator allows waiter to contact chef
- Order Queue
    - Menu - Order queue places menu items ordered in the order received
    - ReservationTime - Order Queue places set reservation time among other reservation times. This is to manage the order of reservation with respect to time.
- Analytic Calc
    - Manager - Analytic Calc is used for this class to display various class specific functions
    - Payment - Analytic Calc is used to calculate cost and verify payment
    - ManagerStatistics - Analytic Calc is used to provide viewable statistics for the manager
    - ManagerProfitView - Analytic calc is used to calculate profits
    - ManagerInventoryView - Analytic calc is used to take inventory
- Table Status
    - Manager - Table Status is a domain concept available to managers. Available because of the administrator role.
    - Busboy - Table status is a domain concept available to busboys. Notified of what table requires their attention.
    - Waiter - Table status is a domain concept available to waiters. Notified of what table requires their attention.
- Food Status
    - Customer - Status of food displayed to customer
    - Chef - Status of food updated by chef
    - Waiter - Status of food displayed to waiter (to reassure customer)
- Payment System
    - Customer - Payment System available to customers once order is delivered and on every subsequent order
    - Waiter - Payment system also available to waiters to keep track of and assist customers through experience.
    - Menu - Payment system located in menu
    - Payment - Payment class located within payment system domain concept
- Delivery System
    - Customer - Delivery progress and status displayed to customer as per Delivery System
    - Driver - Driver updates class and customer notified of update
    - Menu - delivery system located on menu for ease of access
- Employee Profile
    - Manager - access to employee specific profile with maximum privilege
    - Chef - access to employee specific profile with chef relevant privilege
    - Driver - access to employee specific profile with driver relevant privilege
    - Busboy - access to employee specific profile with busboy relevant privilege
    - Waiter - access to employee specific profile with waiter relevant privilege
    - Menu - available to those underneath employee profile

- ReservationTime - available to those underneath employee profile and with specific privileges
- TableAvailability - Requires specific privilege. One of those being an employee. Busboy, Waiter, and Manager have access to this feature.
- MainScreen - all profiles share this in common
- ManagerStatistics - manager feature
- ManagerEditEmployee - manager feature
- ManagerEditItem - manager feature
- ManagerProfitView - manager feature
- ManagerInventoryView - manager feature

*Bottom five listed classes are all manager features. Included because they are a part of the employee profile but restricted only to the manager field.

8.4 Object Constraint Language (OCL)

MainScreen:
Context MainScreen::clickBusboy
Invariant: privateKey, username ,busboy
Pre-conditional: findViewById(R.id.busboyButton)
Pre-conditional: Intent intent = new Intent(MainScreen.this, Busboy.class)
Post-conditional: MainScreen.Busboy.startActivity()

Context MainScreen::clickCustomer
Invariant: privateKey, username, customer
Pre-conditional: findViewById(R.id.customerButton)
Pre-conditional: Intent intent = new Intent(MainScreen.this, ThreeDiningOptions.class)
Post-conditional: MainScreen.Busboy.startActivity()

Context MainScreen::clickWaiter
Invariant: privateKey, username, waiter
Pre-conditional: findViewById(R.id.waiterButton)
Pre-conditional: Intent intent = new Intent(MainScreen.this, Waiter.class)

Post-conditional: MainScreen.Busboy.startActivity()

Context MainScreen::clickChef
Invariant: privateKey, username, chef
Pre-conditional: findViewById(R.id.chefButton)

Pre-conditional: Intent intent = new Intent(MainScreen.this, Chef.class)
Post-conditional: MainScreen.Busboy.startActivity()

Context MainScreen::clickManager
Invariant: privateKey, username, manager
Pre-conditional: findViewById(R.id.managerButton)
Pre-conditional: Intent intent = new Intent(MainScreen.this, ManagerOptions.class)
Post-conditional: MainScreen.Busboy.startActivity()

LoginActivity:
Context: LoginActivity::passwordChecked:boolean
Invariant: auth, loginButton, signupLink,
Pre-conditional: mailId = emailText.getText().toString()
Pre-conditional: pass = passwordText.getText().toString()
Post-conditional: return isValid

SignupActivity:
Context: SignupActivity::signup
Invariant: progressDialog
Pre-conditional: name = nameText.getText().toString();.
Pre-conditional: address = addressText.getText().toString()
Pre-conditional: email = emailText.getText().toString()
Pre-conditional: mobile = phoneText.getText().toString()
Pre-conditional: password = passwordText.getText().toString()
Post-conditional: onSignUpSuccess()
Post-conditional: onSignUpFailed()

Context: SignupActivity::onSignUpSuccess
Invariant: signupButton
Pre-conditional: If sign up success
Post-conditional: setResult(RESULT_OK, null)
Context: SignupActivity::onSignUpFailed
Invariant: signupButton
Pre-conditional: If sign up failed
Post-conditional: Toast.makeText(getBaseContext(), "MainScreen Failed",
Toast.LENGTH_LONG).show()
Context: SignupActivity::checked:boolean
Invariant: name, address, email, mobile, password, reEnterPassword

Pre-conditional: name.isEmpty() || name.length() < 3
Pre-conditional: address.isEmpty()
Pre-conditional: email.isEmpty() || !Patterns.EMAIL_ADDRESS.matcher(email).matches()
Pre-conditional: mobile.isEmpty() || mobile.length()!=10
Pre-conditional: password.isEmpty() || password.length() < 4 || password.length() > 10
Pre-conditional: reEnterPassword.isEmpty() || reEnterPassword.length() < 8 ||
reEnterPassword.length() > 12 || !(reEnterPassword.equals(password))
Post-conditional: return isValid

9.System Architecture and System Design

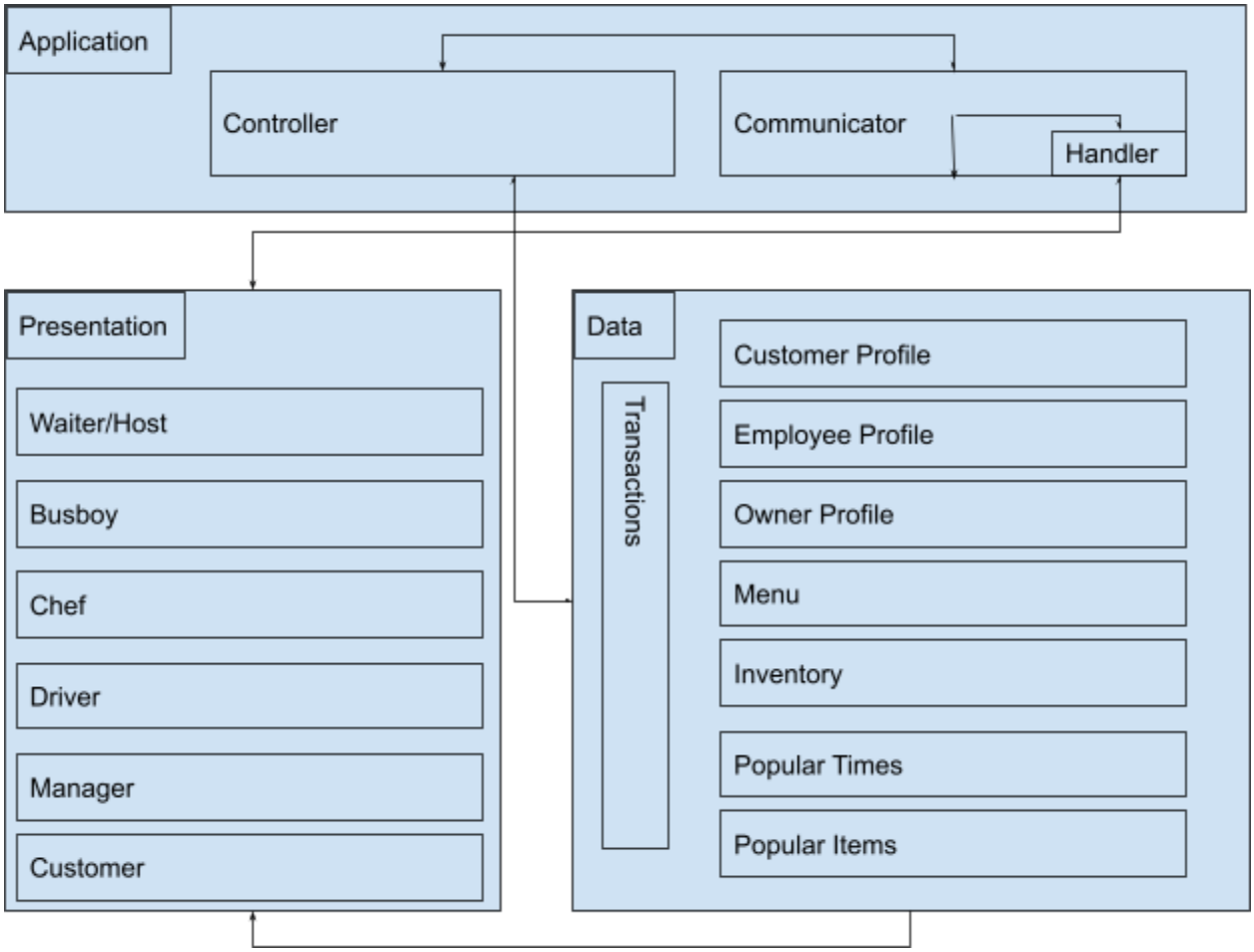
9.1) Architectural Styles

Our application will have three layers and consists of a Data Layer, a Presentation Layer, and an Application Layer. This 3 layered architecture allows the Data layer to be run in the background providing resources alongside with the Application layer to produce what is seen by the user in the Presentation layer allowing a front-end/back-end situation.

This type of architectural style is the most practical for our application since each layer it will be running alongside our application's code. Our application will run using Java and build/edited through Android Studio which allows us to design the graphical user interface directly through this program. The application will be using a database, which will be called through and API which will allow users to get information as soon as it is edited to help get quick and accurate data.

Our application will also use a client/server architectural style which will be used to update our database frequently. This architecture will allow us to return information whenever a client requests to the server. The server, in our case, would be the database with additional application logic. Our database will be used to keep track of menu items, transactions, popularity, location (delivery and open tables), and employee/manager/customer profiles (and their permissions within the app).

9.2) Identifying Subsystems



The subsystem for our application is shown above, and consists of three layers which is the Application Layer, the Presentation Layer, and the Data Layer.

Data Layer: Uses the database and contains information about the Profiles of each user, the menu items, the inventory, the transactions that occur throughout the day, and lastly the popular food items and customer visit times.
Presentation Layer: Consists of the different users such as Waiter/Host, Busboy, Chef, Driver, Manager, and Customers. This layer takes information from the database and displays it to the user's screen depending on the type of user they are.
Application Layer: This layer contains a Controller, which takes tasks between packages and will use the Communicator to provide permissions between the packages and use the handler to load the events as per instruction. This layer also deals with business logic, and will contain objects that are able to implement the business functions. The use cases will all be implemented through this way in the application layer.

9.3) Mapping Subsystems to Hardware

No, all our subsystems can be run on the same server. However, there will be a downloadable mobile application where customers can order delivery, takeout, make reservations, look at reviews, etc. There will also be a version of the application for the manager/employees side, as they will obviously have a different interface.

9.4) Persistent Data Storage

After logging out of the app the system stores persistent data. This will consist of each customer's ratings. Customers personal information will be stored in the database. Each customer will be required to create a customer username which will store all the ratings under this username. In addition, personnel inventory and other information that the manager needs to know will be stored in the database. For personal inventory, there will be an entry for each food item and the amount of this. As orders are placed the inventory will be updated. This is so when the manager logs in the inventory will be viewable. Customers will also be able to choose which table is available and this will be stored in the database. During busy hours we will have a table that corresponds with customer peak times. Periodically this will be updated and stored in the database. Employee's salary information and hours will also be stored within a database. Changes to the menu and statistics about the sale of each food item will be stored persistently. The amount of deliveries verse takeout verse dining in will be stored as well to be able to see the popularity of each to weigh where to allocate more resources towards.

```
customerprofile :: {
name:String
username: String
customersorders: [Order]
}

Order :: {
customer: Customer
items : [Item]
date:Date
}

Item :: {
name: String
quantity: Number
price: Number
rating: Double
}

Menu :: {
```

items: [item]
}

9.5) Network Protocol

The network protocol to be used for our purposes is normal sockets. The reason we are going in this direction of transmitting information is because all the backend information processing will be done on the server computer. The application will mainly serve as little more than a graphical frontend for the end user. The front end of the application will handle performing the requests. The user interacts with the application and when they perform a task, a request will be sent to a server where the information will be processed. For example, if the user orders food after clicking confirm, the order will be sent to the server which will respond back with information including estimated wait time, which is necessary for the client to display to the user.

9.6) Global Control Flow

Execution Orderliness
The application is event-driven and the user has full control over the application's use. From the menu, the application will branch to show different features the application provides as per the user's command.

Time Dependency
The system is dependent on time. All applications are synced to real time in order to keep track of when menu orders (inside the restaurant) and delivery times (outside the restaurant) are placed. The system also inputs the current time to estimate when certain features will be completed. These features include estimated delivered time as well as estimating when a table will be empty- to name a few.

Concurrency
Data is maintained in sync as requests are sent to the database. The database will update with every request and send relevant information to be displayed on the app interface.

9.7) Hardware Requirements

This software will run on smartphones and tablets with a touch screen running an Android operating system. The targeted minimum version is API 23: Marshmallow which would allow the program to run on 62.6% of devices. Devices will require at least 1MB of storage space to install the application. The minimum resolution required to display the application is 640 x 480 pixels. The minimum bandwidth required to access the server and database is 56 kbps. The minimum RAM required to run the application is 1GB. The software is intended to run on mobile devices with a touchscreen interface, so there is no desktop version of the application. Both smartphones and tablets will use the same version of the application.

10.Algorithms and Data Structures

10.1) Algorithms

List of Algorithms:

- Reservation (Jason)
    - For reservations, there will be a number displayed that shows how many tables are available. Thus, we need a simple algorithm that subtracts the total number of tables by the current number of tables occupied. Furthermore, this simple algorithm will be contained with another one, that contains a bunch of conditionals. For example, if there are no tables available, then it will have to notify the customer there are no available reservations for that time. If this is not the case, and if the table is not already occupied, it will reserve the table number for the customer.
- Delivery (Juergen)
  The process of delivery can be broken down into:
  The customer completes their order for delivery and then the chef will be given an order to complete.
  Once the chef completes the order the system will send this order information to the driver interface.
  A driver can see any orders to deliver and then they can accept an order.
  Once the order is completed, the driver will change the status to completed.
- Payroll (Juergen)
  The algorithm to calculate payroll is for every employee that worked during a pay period, then we find their hourly pay and the number of hours they worked that period and multiply them.
- Payment - Rewards: (Jason)
    - An important algorithm for food order, is an algorithm that allows the customer to add food items to their order. Every time the customer selects a food item, it will search the item's price and add it to the total bill.
    - For payment, the customer will have multiple choices of payment, an each will have its own algorithm to deal with the transaction.However, the main theme will hold still, if the customer is able to fully pay the bill, while taking note of any rewards or discounts, then the payment will be marked as successful. However,

in the case that it does not successfully pass through the payment algorithm, then an error screen will be brought up, and the payment will temporarily be marked as unsuccessful until it is paid.

- Analysis: (Jason)
    - Regarding analysis, there are a few things that will require algorithms to function properly. To find popular food items, you will need a simple algorithm that tallies up the amount of times an item on the menu has been ordered that week. Then, another sorting algorithm will be implemented to find the food items with the highest counts that week. There can also be algorithms to help with bookkeeping. There can be arithmetic algorithms that add up checks for the day, and calculates the profit by subtracting the costs of the day. Furthermore, when it comes to paying employees monthly or weekly, there can be algorithms that keep track of how much money is left as net profit.
- Scheduling (Juergen)
  To schedule an employee, a manager needs to specify a start time, end time, and employee name. We then need to implement an algorithm that will show the manager who is working around those times , to avoid any overscheduling. The manager can then confirm and the employee will be updated with their schedule.
- Ordering (Juergen)
  To order, a list is created. Everytime the user adds something to their order on screen, we update the list with their order. When a user changes their mind and wants to remove an item from their order we will need to search the list and delete it.
- Employee Clock in and out (Pavan)
    - For each employee, a class is created that contains the time an employee has clocked in and clocked out for the day. These are represented as strings within the class. An interface allows an employee to clock in and this time is recorded in the string “clockin”. Specifically, based on user interaction the current time and index of the employee (employee id) is passed into the function clock_in which updates an array of class employee. The index of the array is accessed (as it represents the employee id) and is updated by changing the “clockin” time to what was passed into the function. Likewise a similar process is completed to update “clockout” time. This module will also display all employee clock in and clock out times to the manager at the end of the business day. This module runs per business day and the array is cleared at the start of a new business day.
- Food Order Status (Pavan)
    - This algorithm tracks the status of food and gives control of updates to the chef. The chef passes in information such as the id of the customer order and the degree of completion. This information is then used to access and update the order status. Degrees of completion range from 0-pending, 1-preparing, 2-almost completed, 3-complete. Upon update, the order status will automatically be displayed to the customer so they are notified of a change in status.
- Update Menu (Pavan)
    - Accessing and updating the menu database is handled by privileged users. Only these users have access to this algorithm. The module updateMenu comes with

several key functions. A manager needs to take advantage of a fast operation that can quickly remove and add stored items in realtime in case of shortages or unpredictable changes. This is why one key function is the shortcutUpdateMenuDatabase. This allows a user to access the menu item and either remove or add an item. The other two key functions are adding and removing an item. This is handled by passing information such as menuItemID, the title of the item, and a helpful description to help customers decide. Based on what function is used either an item will be added or removed from the menu.

- Display Daily Sales/Profits (Pavan)
    - This algorithm calculates the total sales/profits over the span of an entire day. Daily cost in an integer that will also be displayed. This is so revenue can be calculated as well. The way this algorithm is designed to work is in several ways. Daily cost can be updated manually. A receipt database is kept and maintained to store receipts throughout the day. Receipts are appended to this database. A function called displaySales cuts the database so as to access information within the day exclusively. Then, the algorithm counts the receipt in the day block and tallies the total dollar amount. Then sales are returned. Lastly, a displayProfit option is chosen to subtract cost from sales and display revenue. This works by calling the function with the total number of sales and making daily cost a global variable.
- Food Inventory (Hojun)
    - The food inventory class is created so that the manager can track the status for inventory of each food item in order to prevent shortage of certain menus. The desired amount of inventory is set for each food item, and if the inventory left for a food item is below its desired amount, then the system notifies that a food item is running short and that it must be supplied more.
- Account Login (Hojun)
    - This algorithm allows the user to log-in to his/her own account by typing in valid username and password. If the username and password matches with the information stored in the database, then the system allows the user to access his/her account. If the username or password does not match, the system lets the user re-enter username and password. If the user enters invalid username or password for 5 times, the system blocks the reattempt to enter username or password for a security reason, but instead it leads the user to reset the username/password.

Our system primarily uses arrays and databases. Criteria involved in choosing the proper database involves reliability and flexibility. In terms of reliability, users can expect a consistent look up time of O(n). At the same time, arrays and databases tend to be flexible because adding to a database/array results in only a change of O(1) look up time.

While listing certain information on our application interface arrays are helpful. It is easy to extract the needed information and display menus, receipts, tables, and items ordered. This goes for both arrays and databases.

The primary takeaway is that arrays and databases have a consistent read and write time and are both flexible enough to handle adding or removing items without much difficulty.

The app can serve multiple users concurrently by using a realtime database so that no information is lost and is updated to the system. Each user also has their own different accounts with information associated with each account.

There are a number of different features in the app that need users to be served concurrently. For table reservations, each table has a status field of occupied or unoccupied that is updated in the realtime database. When a customer wants to reserve a table for a specific time, this must be checked to avoid conflicts. Table reservations also have a “first come first serve” scheme where if two users try to reserve a table around the same time then the user who clicked reserve first should get it by possibly introducing a small delay.

When customers make orders,the information of the order is sent to the database where the chef can get updates on this information so they can make the orders and update other employees like drivers or waiters who would need to know when a food order is done. This way no information is lost.

Another example is employee scheduling. When a particular employee logs in they should be able to check their schedule by getting that information from the database. There are many more other examples.

11. User Interface Design and Implementation

Register/Login Interface:

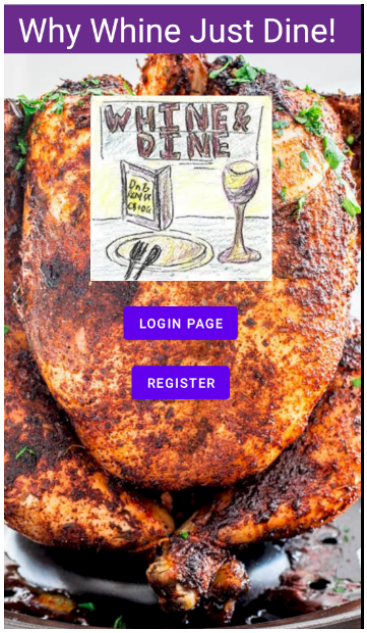


Fig. 1, Main Screen

As can be seen, this is the home screen of our application, “Why Whine Just Dine”. When the user opens the application, this will be the first thing that they will see. On the screen, the user can see two buttons, one that brings them to the login page, and another to the register page.

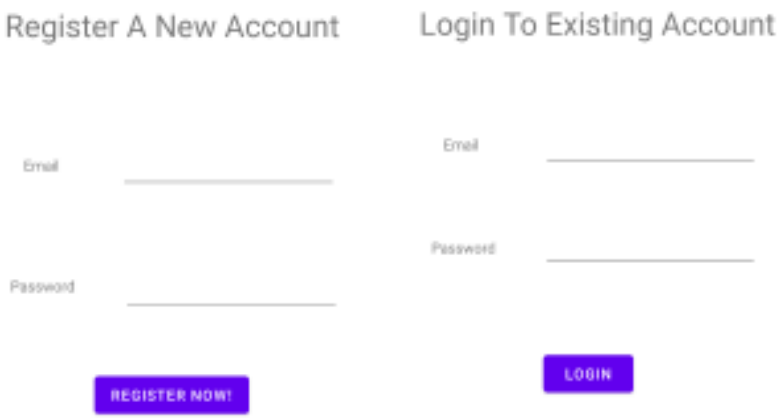


Fig. 2, Register/Login

As seen above, there is a registration page and a login page. If one doesn’t have an account yet, then the user can register a new account with an email and a password. There is a specified minimum length for both the email and password, so if the inputs are invalid it will prompt the user to input again. The email and password will be sent to a database and stored there.

For the login page, it can be seen that the user can enter an email and address to login. The entered email and password will then be sent to the database where it will check and authenticate the user. Once this is done, depending on the type of user, (ie. employee, manager, or customer), they will be sent to a respective home page.

Customer Interface:

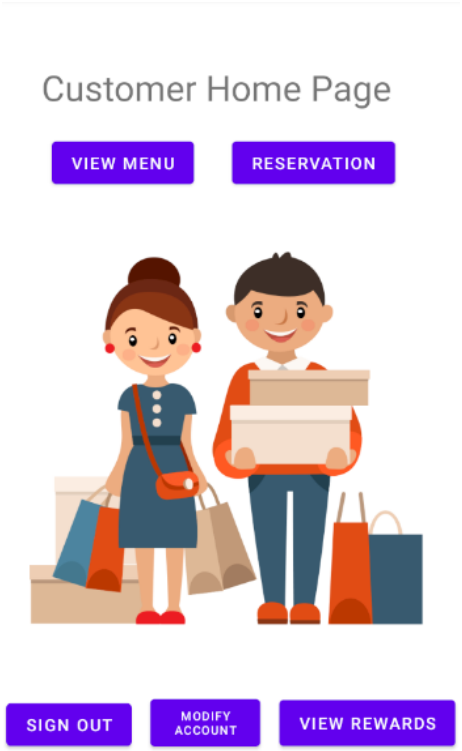


Fig. 1, Customer Home Page

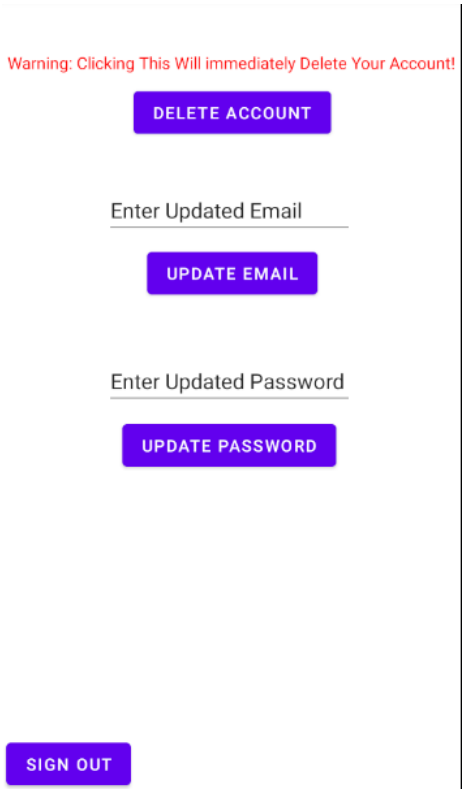


Fig. 2, Account Deletion

Above you can see the customer home page. As can be seen, there are 5 buttons. The first button “View Menu” will bring you to a menu page where the customer can then order food. The “Reservation” button will bring the customer to a page that allows them to reserve a table in advance. The modify account button will bring the user to a screen as shown above on the top right. As can be seen, one can delete their account, update their email, and/or update their password. If one deletes their account, the account will be wiped from the database and cannot be retrieved. There is also a signout button on both pages to allow the customer to exit after they have completed what they want to on the app.

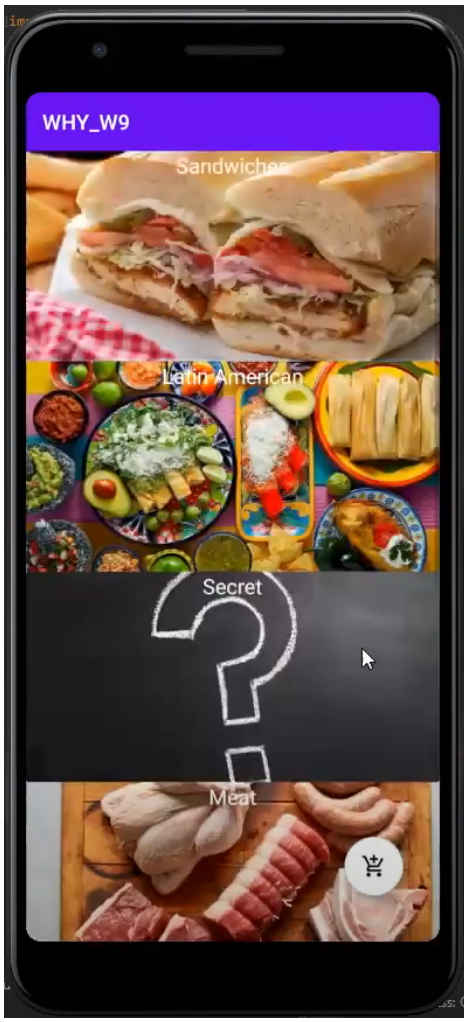


Fig. 3, Category Menu

In Fig. 3, we showcase what our menu looks like. Our menu is scrollable. There are many categories listed. Clicking on a category will lead to Fig. 4.

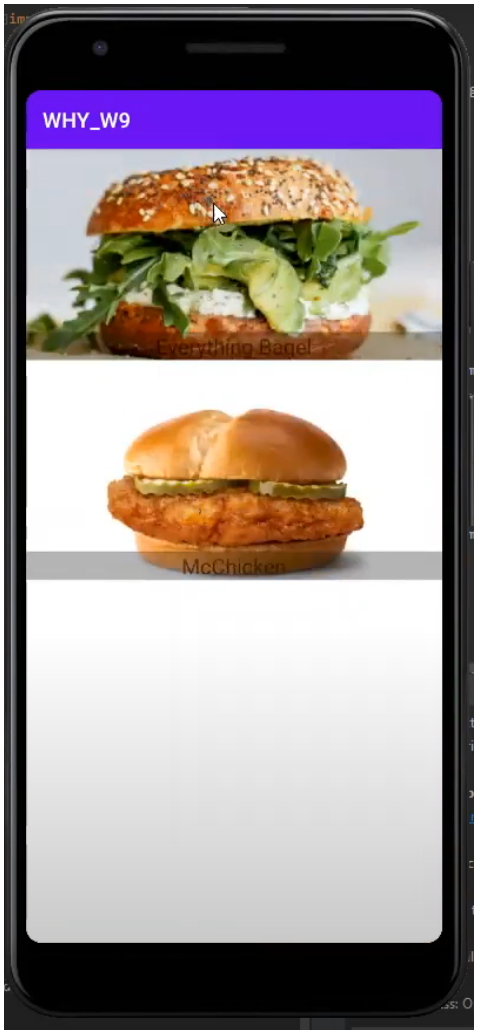


Fig. 4, Items Listed

In Fig. 4, the items underneath the selected category in Fig. 3 are listed. The user can then select an item. This will lead to Fig. 5.

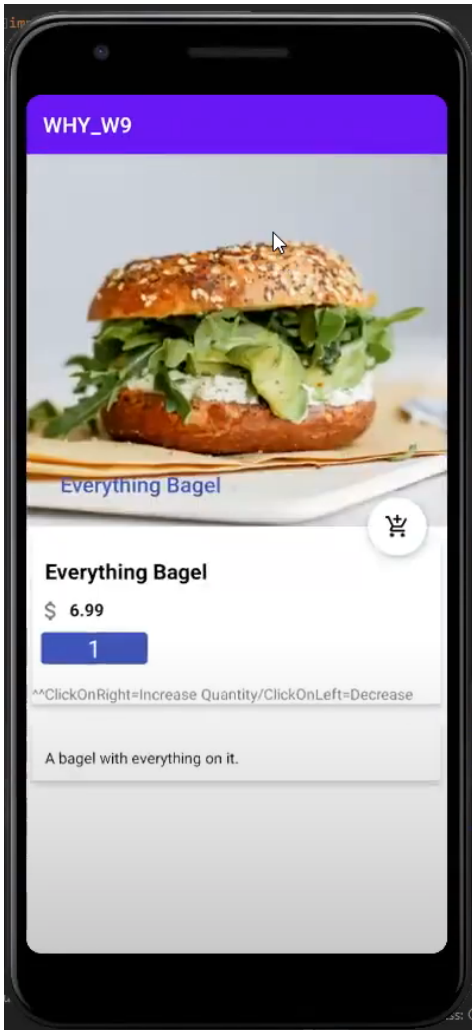


Fig. 5, Item Description

In Fig. 5, we showcase details describing the selected image. We have a picture, the name of the item, the price, as well as a description. The user can select the amount of the item they would like to add to cart. Users add to cart by clicking the cart icon.

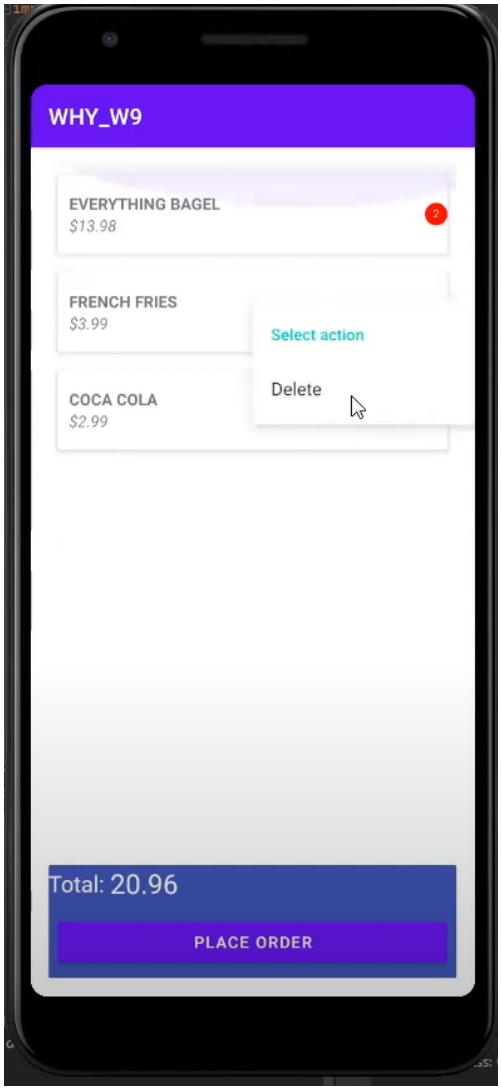


Fig. 6, Cart

Fig. 6 shows what the cart looks like. Items added to cart will appear here. The amount of each item will also appear here. The total price is calculated and displayed at the bottom. The user can choose to delete an item by long pressing on the item and selecting delete. Once the user is certain of their order they can then place the order by clicking the “PLACE ORDER” button.

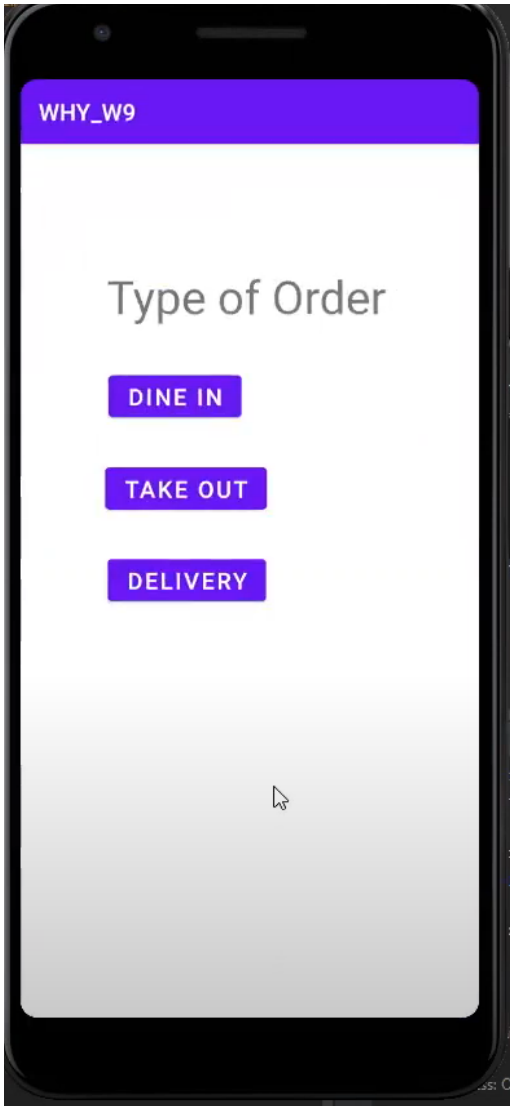


Fig. 7, Type of Order

In Fig. 7, users choose the type of order they prefer.

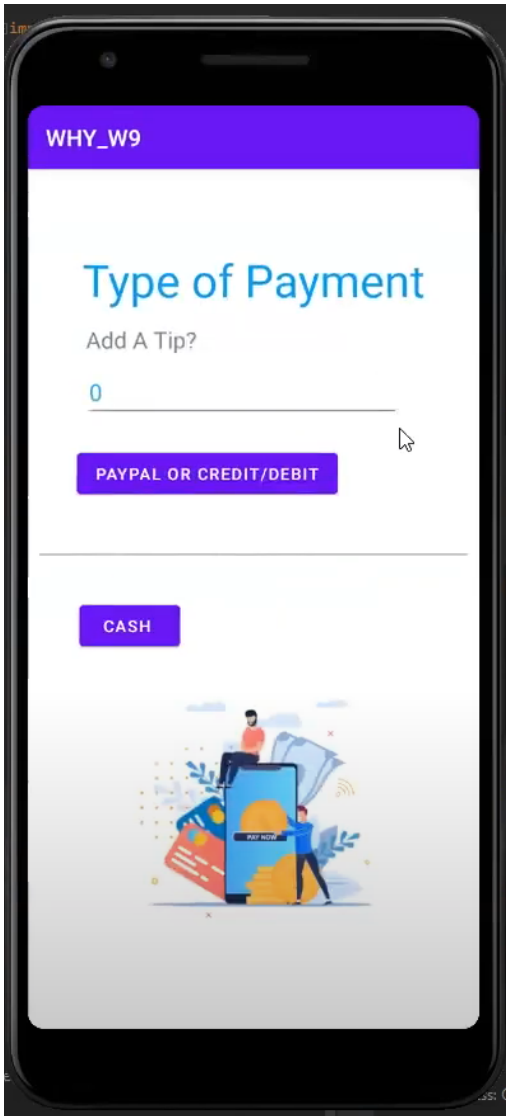


Fig. 8, Dine In and Takeout option

Selecting the “Dine In” and “Takeout” buttons in Fig. 7 brings the user to this screen. The user can choose to add a tip, use paypal, credit, and debit card options, or pay with cash. Selecting the “CASH” button sends the order request to firebase. The order type, such as takeout or dine in is information that will be stored in the order request.

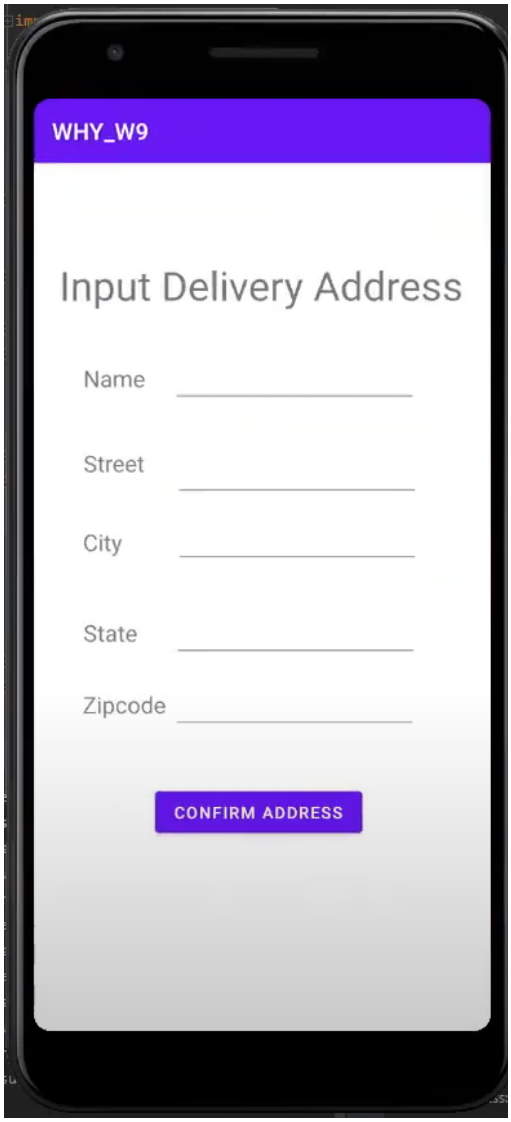


Fig. 9, Delivery option

In Fig. 9, the user is prompted to enter in information on identification and delivery. Once the address is confirmed, we are brought to the page in Fig. 10.

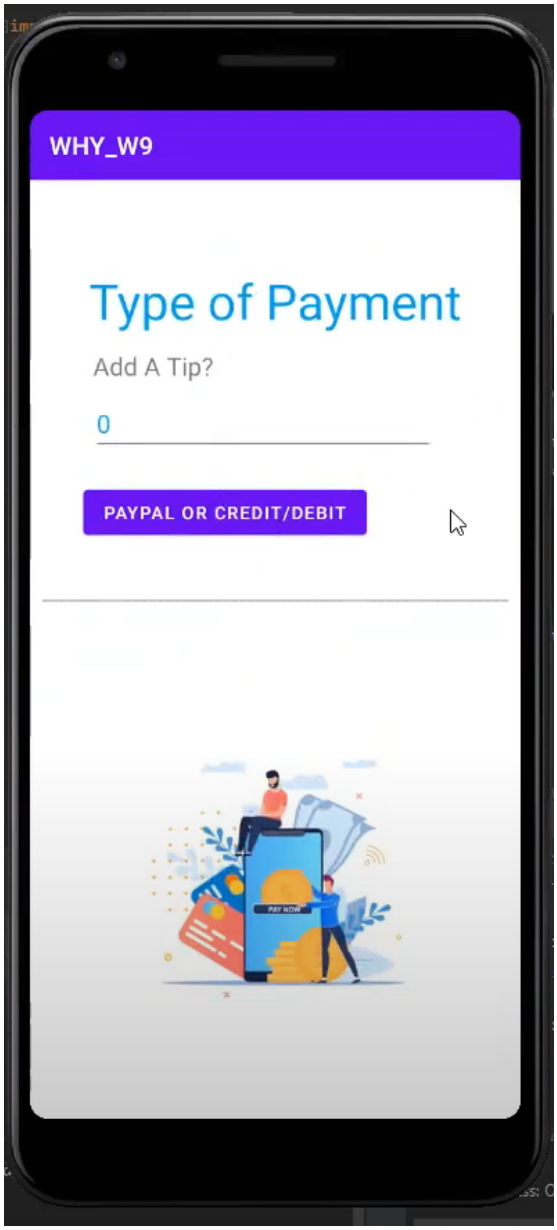


Fig. 10, Payment modified

Fig. 10 is a modified layout of the payment page. Since the option selected was delivery, users are required to pay through paypal, credit, or debit.

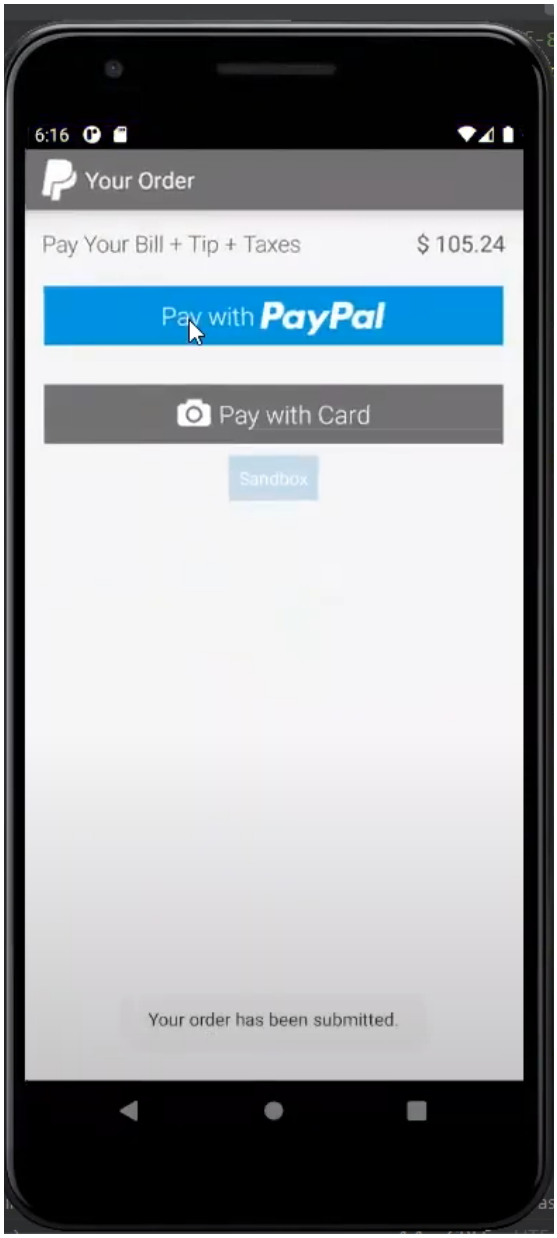


Fig. 11, Paypal

Selecting the paypal option brings up the paypal screen you can see in Fig. 11. At the top right, users can see their total. This is a sum of the bill, tip, and taxes. Selecting the pay with PayPal option brings us to Fig. 12. Selecting the pay with card option brings us to Fig. 13.

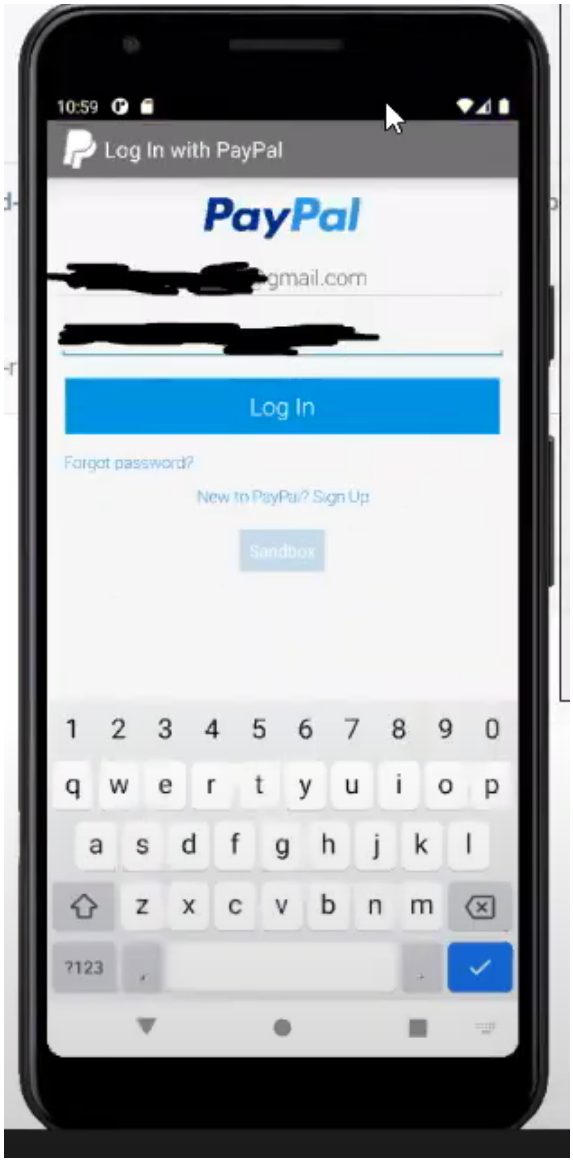


Fig. 12, Pay with Paypal

Users can proceed to login to their PayPal on this page and pay.

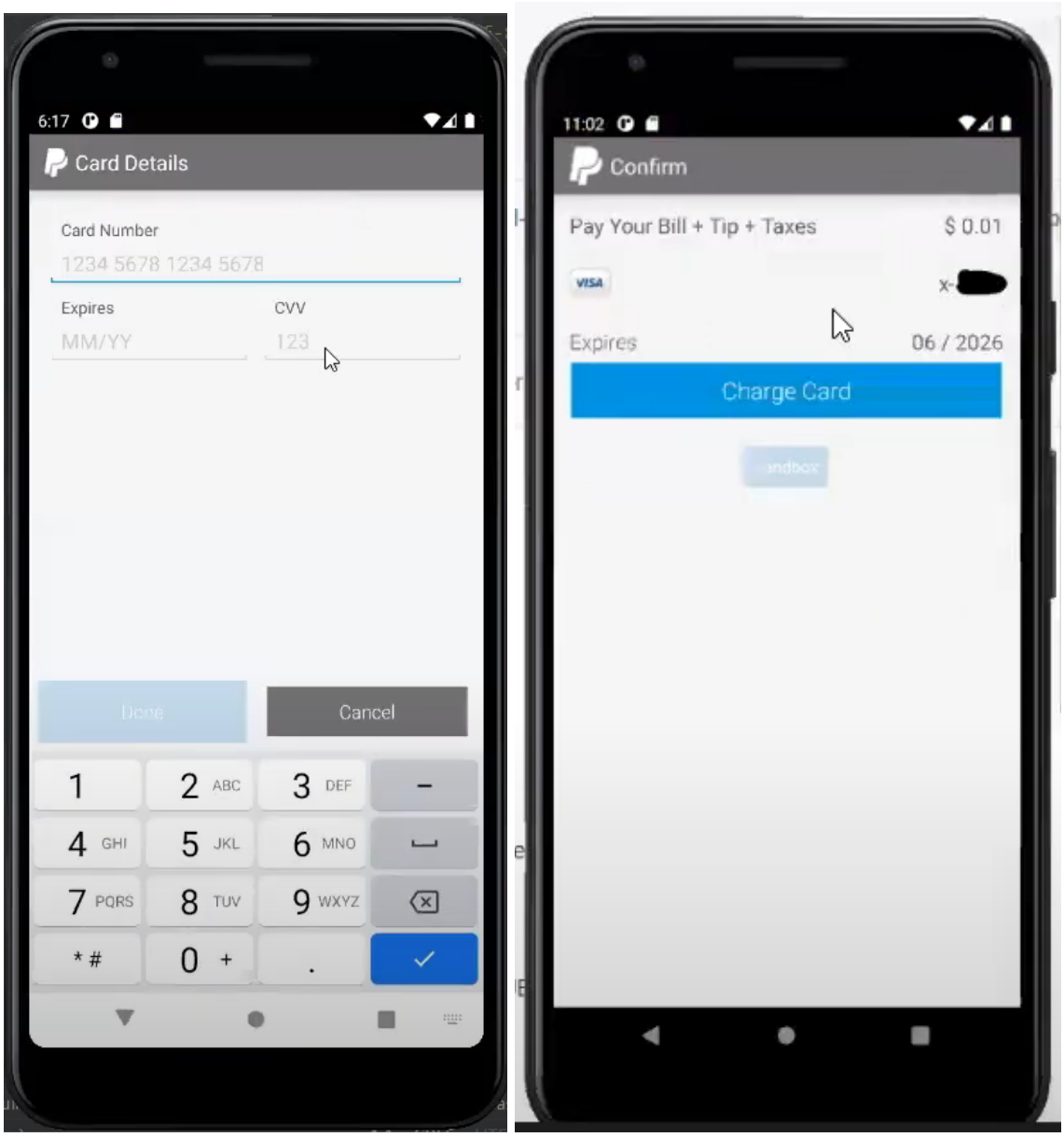


Fig. 12, Pay with Card

Users can proceed to enter their credit card information on this page and pay. Once finished, the user is brought to the screen on the right. One the card has been confirmed, payment will be processed. This usually takes a few seconds.

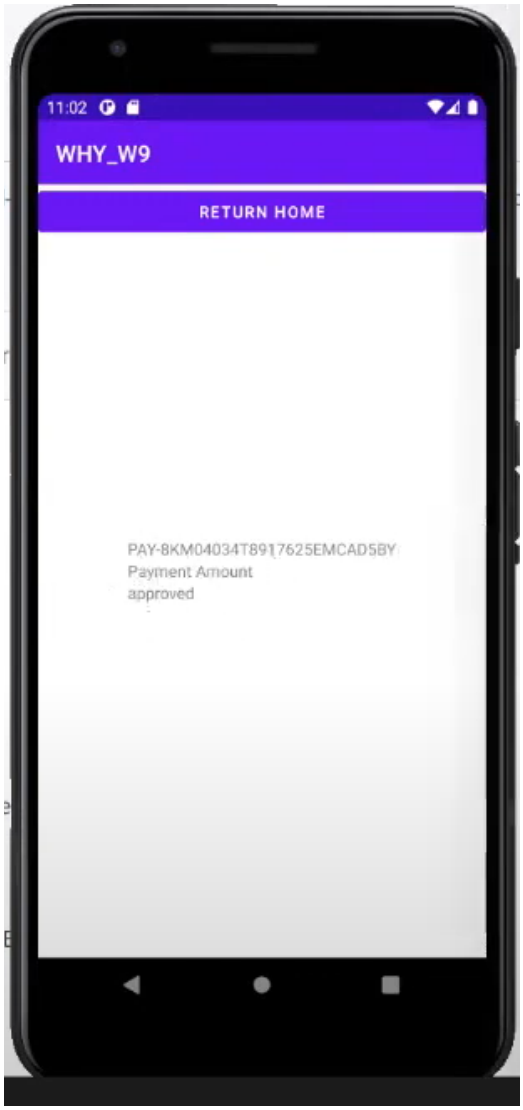


Fig. 13, Payment Details

In Fig. 13, we can see the payment details page. This is the screen that appears after your credit card has been processed. From top to bottom, there is listed a payment id, a payment amount, and information on whether or not the payment was approved.

Manager Interface:

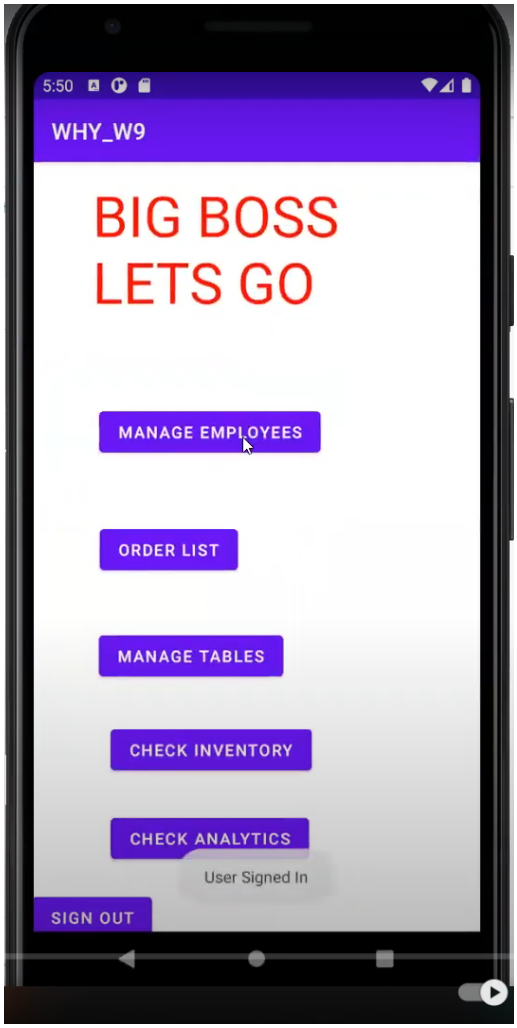


Fig.1, Manager Home

As can be seen on the screen above, the manager home page has a few buttons. The first button “Manage Employee” will bring the user to the employee management page where the user can then click on buttons to go to employee schedules and payroll. The second button on the home page is an order list, that will bring the user to a screen that shows a list of all the current orders in the restaurant. The manager can also manage the tables, which will bring them to a screen to be able to check and update the status of the tables in the restaurant.

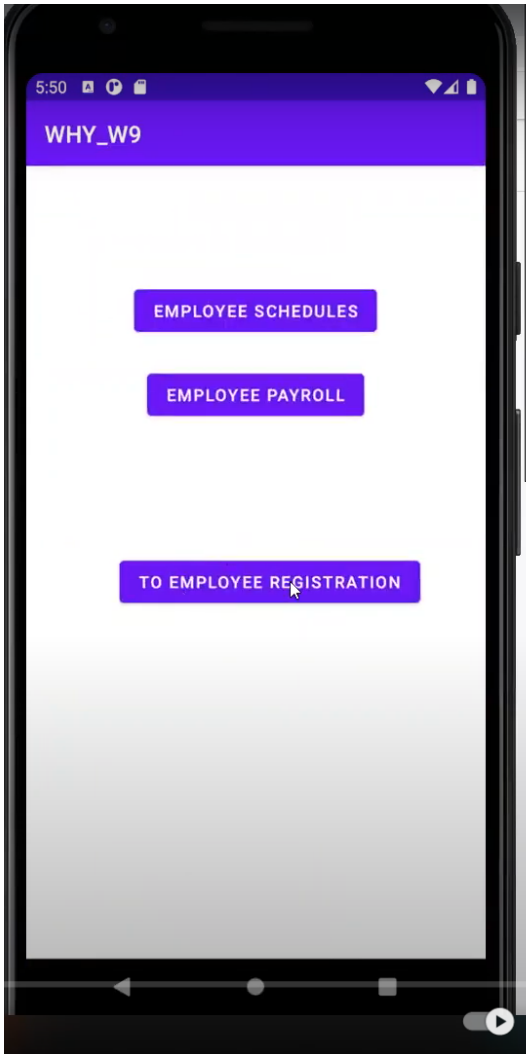


Fig.2, Manage Employees

Clicking the to employee registration button in Fig. 2 leads to Fig. 3 as seen below.

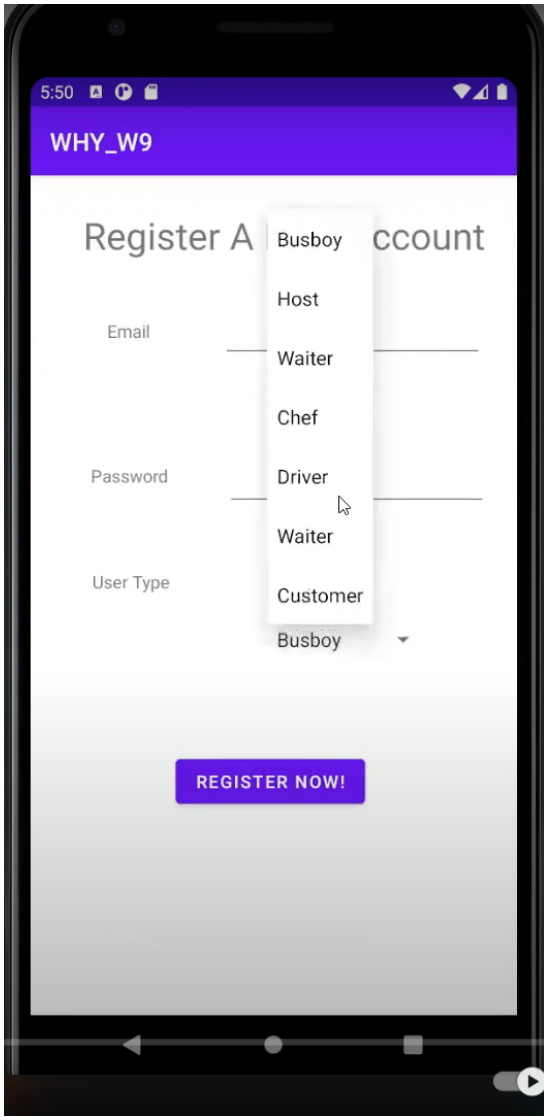


Fig. 3, Registering a new employee account

In Fig. 3, the manager has access to employee registration. He can choose what type of account he would like to create as well the email and password. After creating the account, he can then tell his employee what their email and password is so that they can login from the login screen.

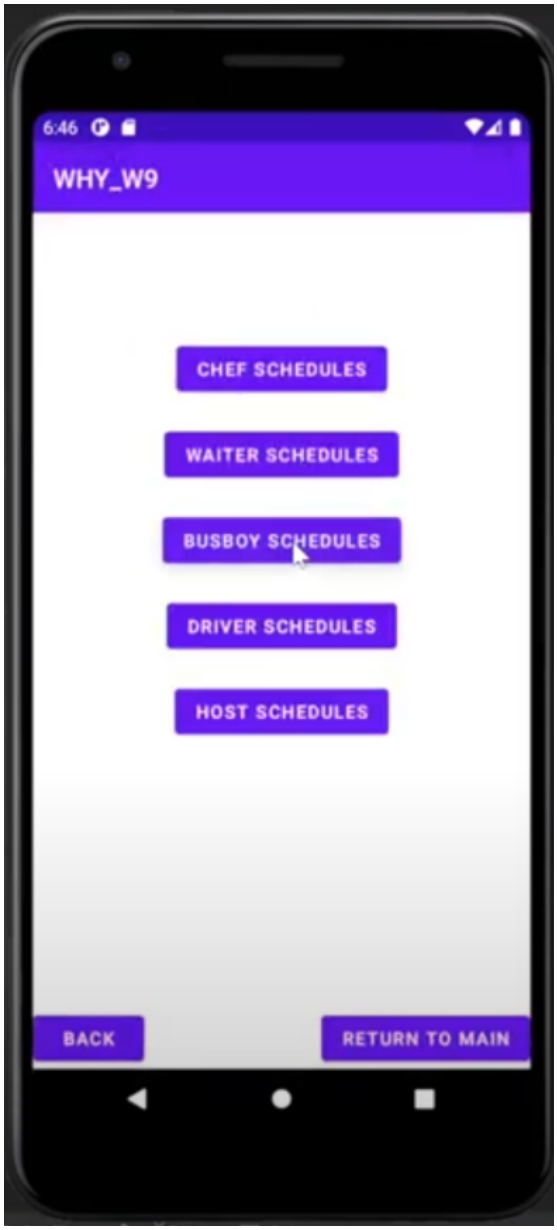


Fig. 4, Employee Schedules Overview

Modifications have been made to employee schedules since Demo 1. Fig. 4, shows us the new look of the employee schedules overview page. Updated changes are now stored in the realtime database. Selecting one of the employee schedule buttons leads us to the screen in Fig. 5.

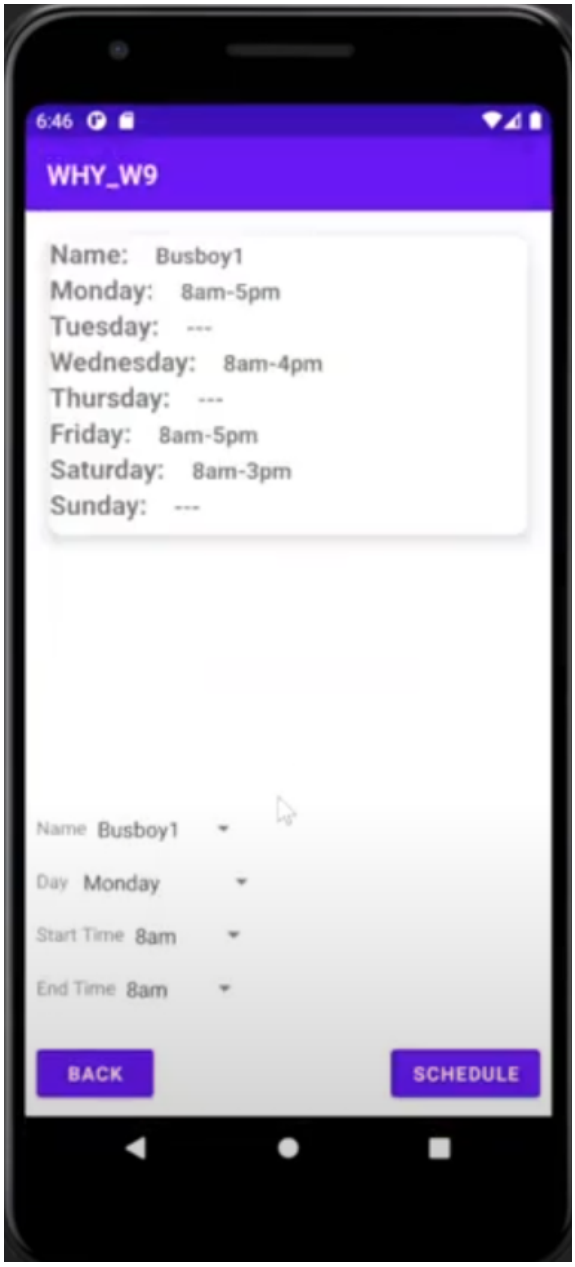


Fig. 5, Employee Schedule

In Fig. 5, we see the layout of the employee schedule for a busboy. The user can update the schedule using the configuration at the bottom of the screen. After the desired update has been selected, selecting the “SCHEDULE” button displays the change not only in the top of the screen but also in the realtime database.

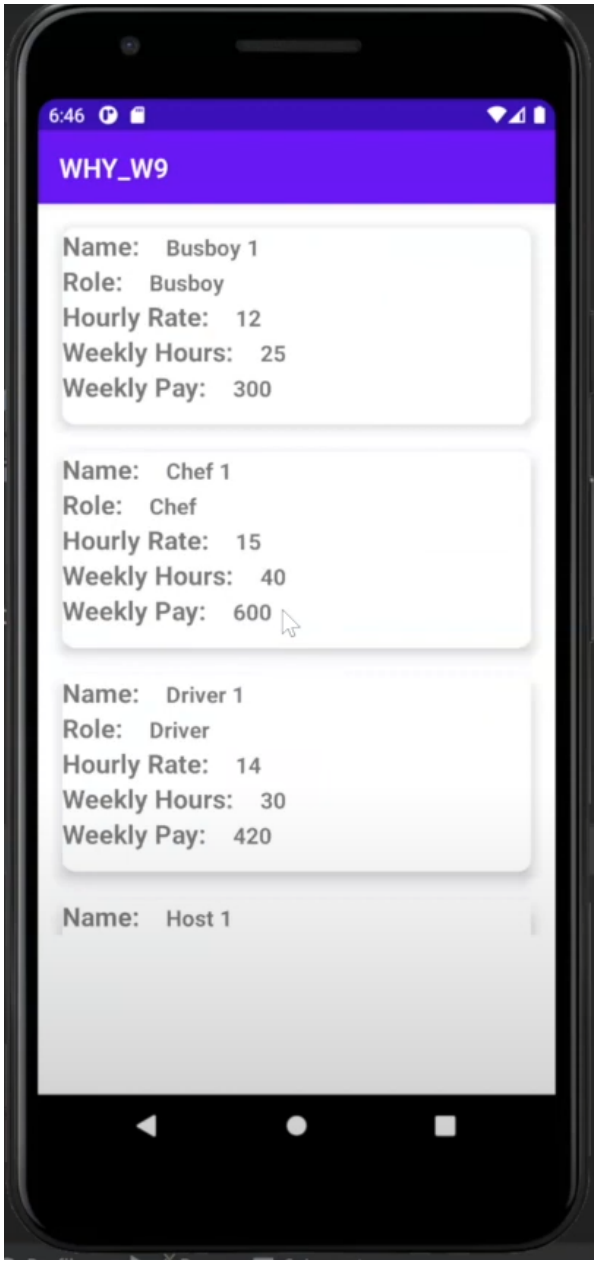


Fig 6, Employee Payroll

This screen can be brought up by selecting the “EMPLOYEE PAYROLL” option in Fig. 2. This screen displays information about the Name, Role, Hourly Rate, Weekly Hours, and Weekly Pay of every employee in the system. This information is in sync with the firebase database.

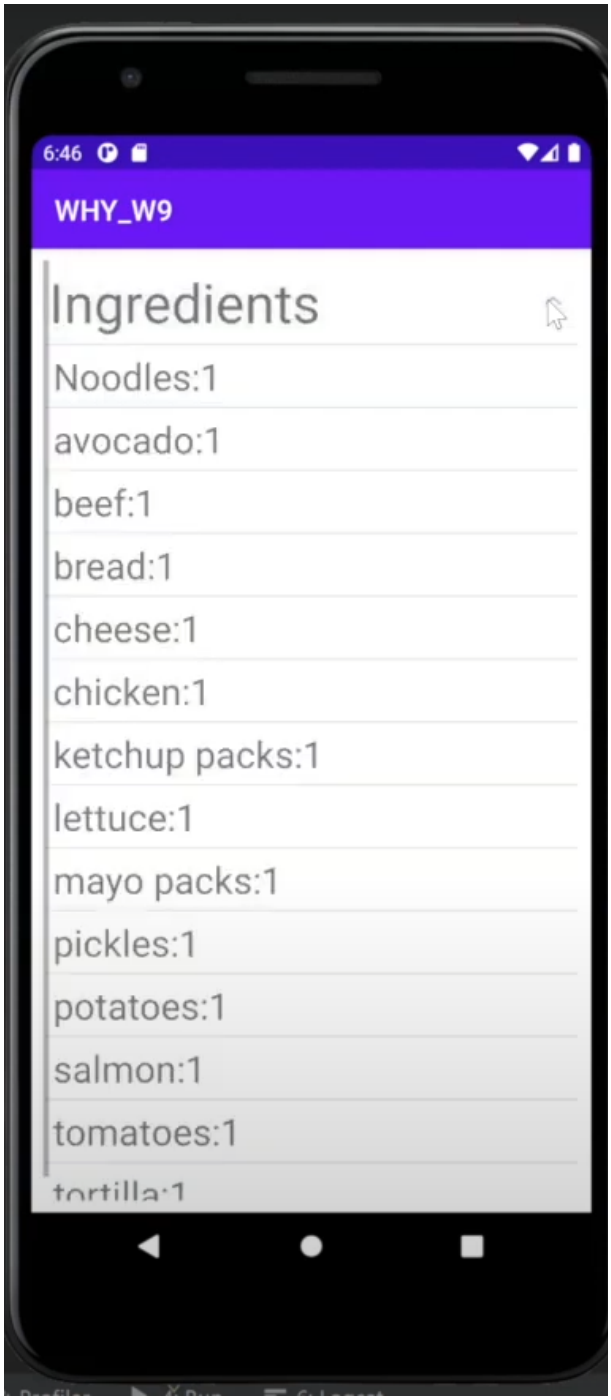
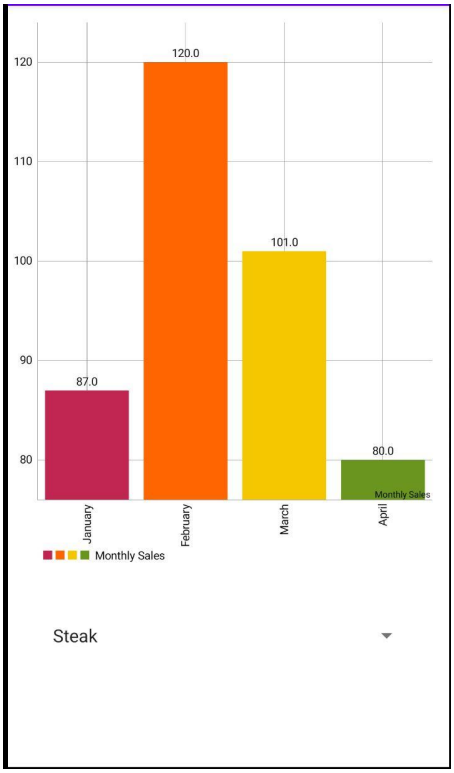
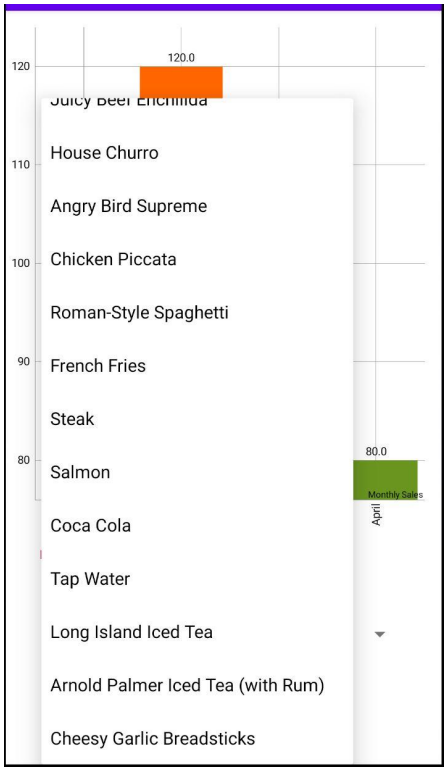


Fig. 7, Check Inventory

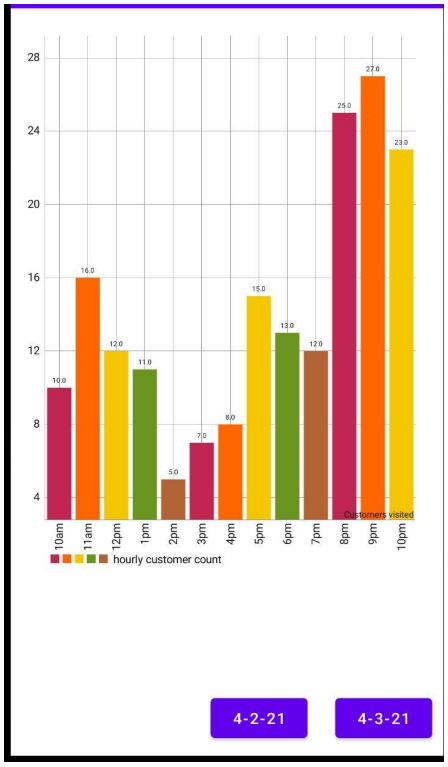
In Fig. 7 we see the screen that appears when a user presses the “CHECK INVENTORY” button in Fig. 1, Manager Home. What appears is a list of ingredient items that the restaurant would need in order to prepare menu items.

Manager Analytics Screen:

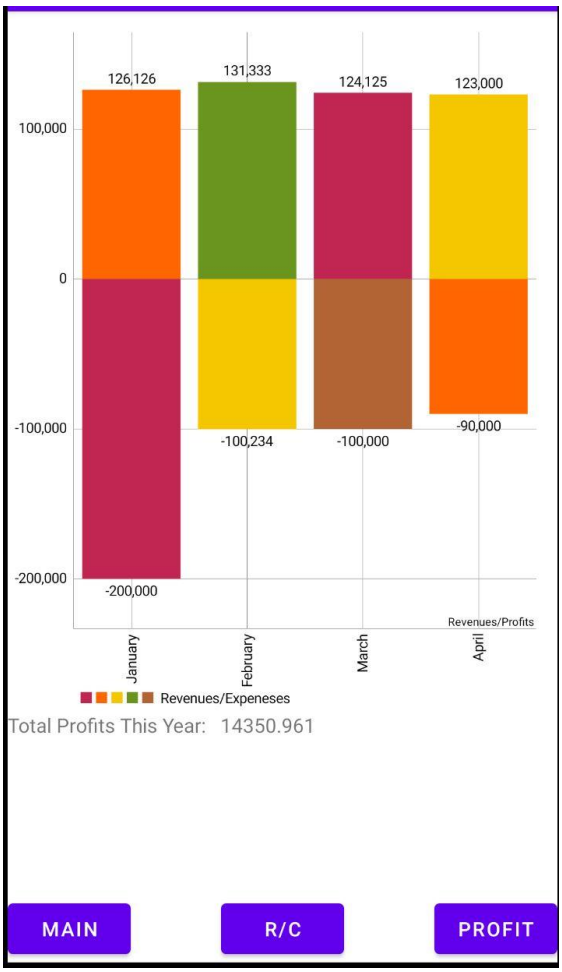


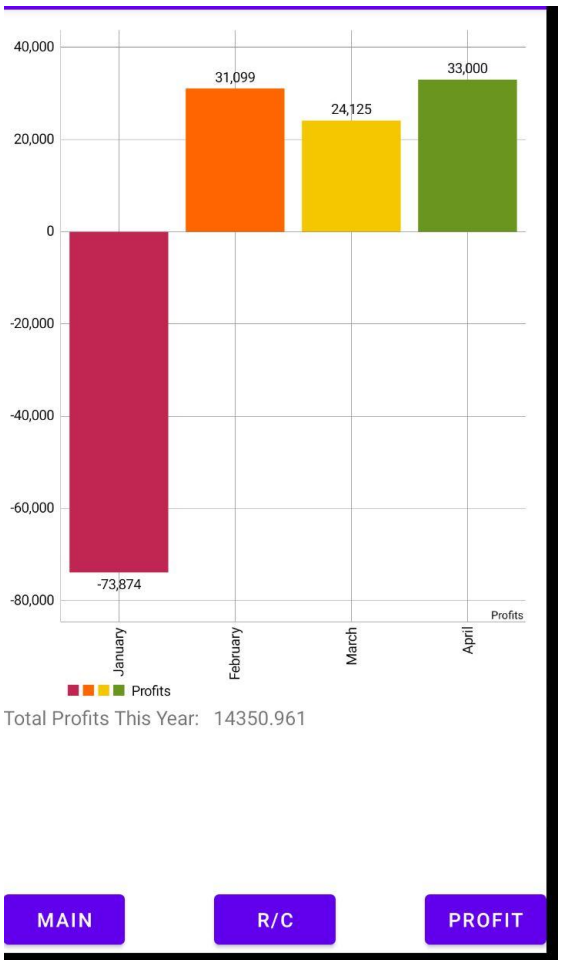


Food Analytics Option is selected and it shows the number of times a particular item was ordered since January.



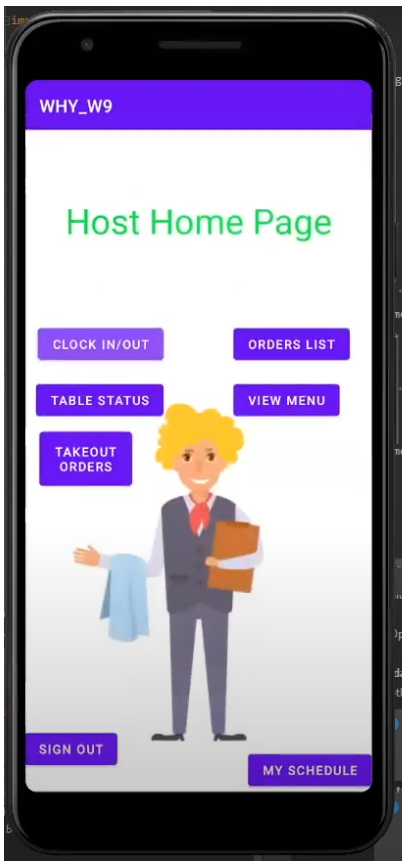
Here, there is data for the daily hourly traffic for two different days using data from firebase.

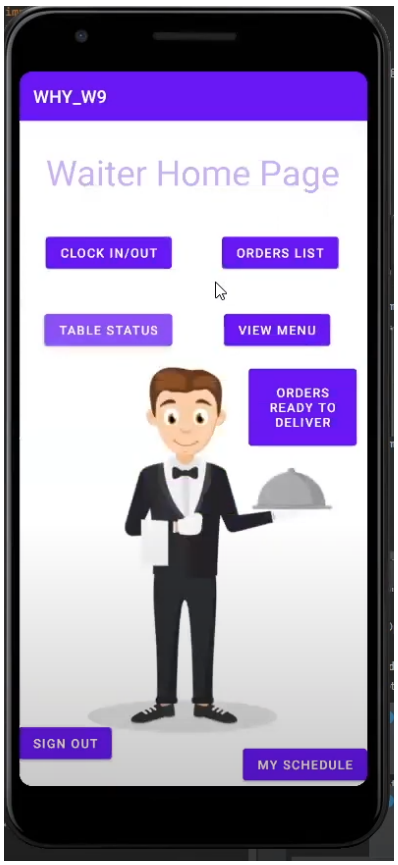


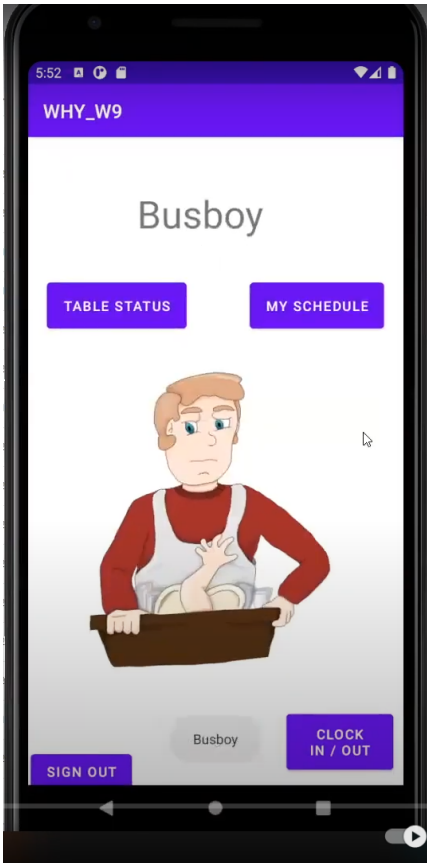


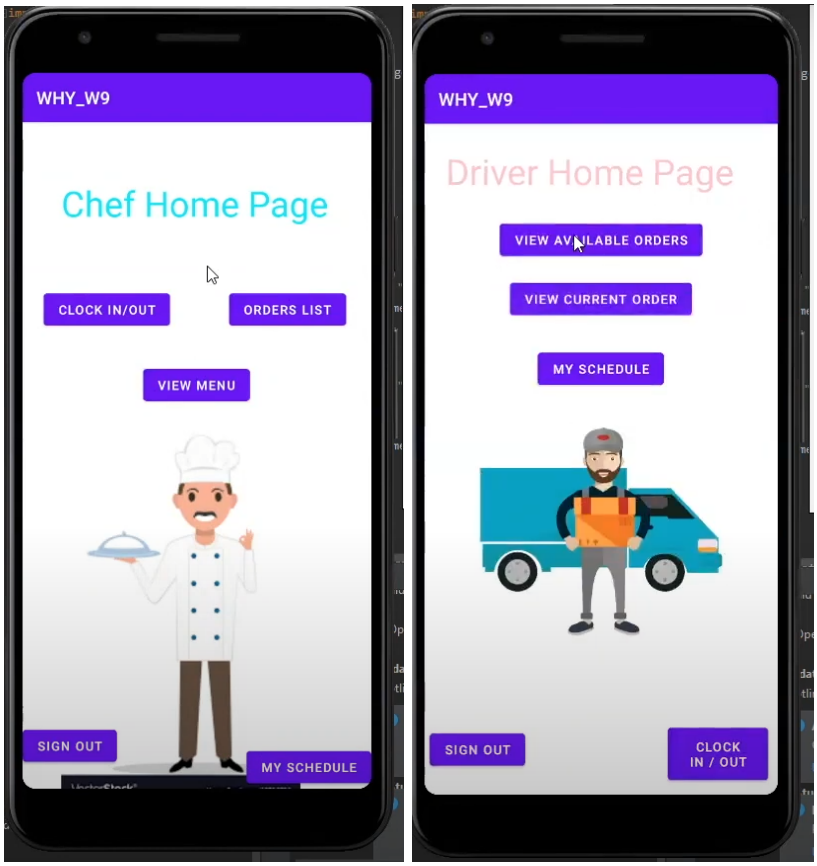
Choosing operations info we can see detailed revenues and cost numbers on the same graph. Or the profits only can be shown with the total profits since January.

Employee Interfaces (Specific functions will be listed in other sections):



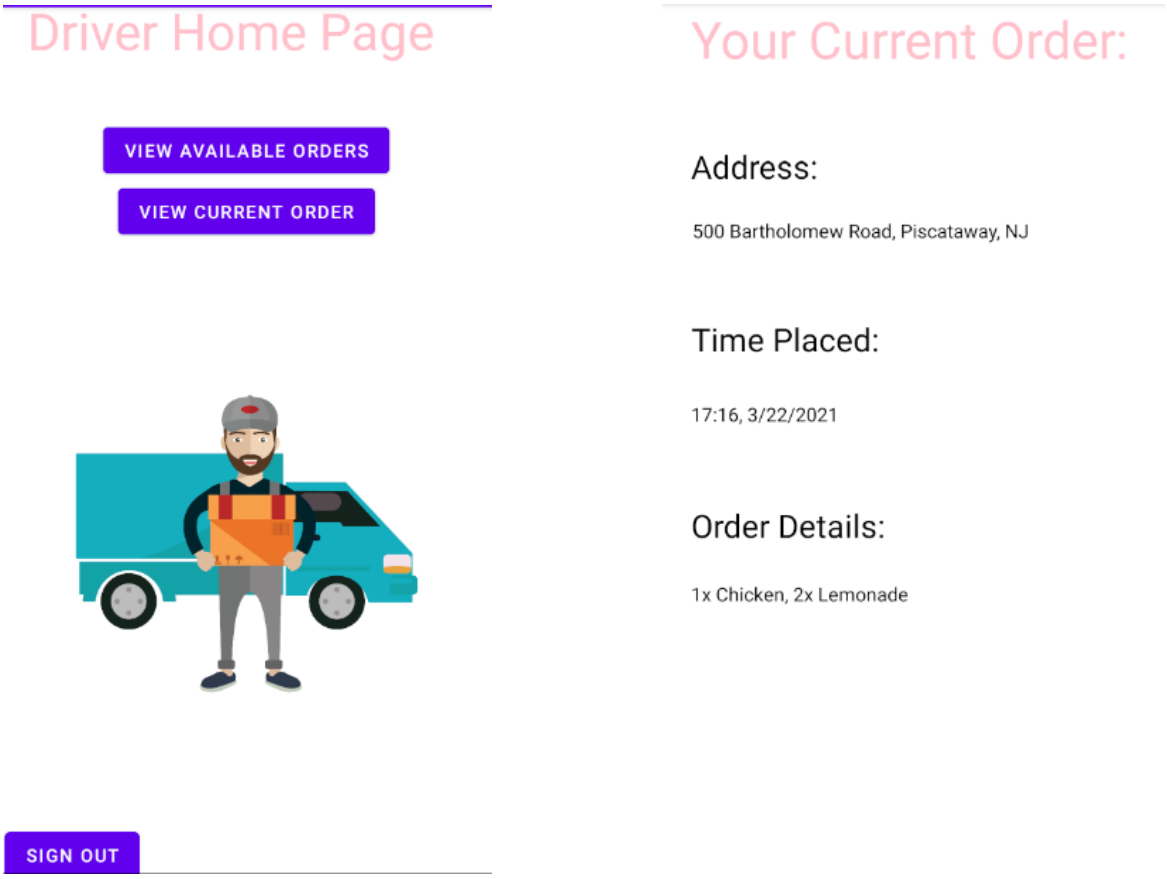






In the following employee interfaces, they also have similar buttons that will allow them to access certain screens relating to their jobs. For example, the host can view the menu and list or orders, as well as access the table status page when seating customers. Furthermore every employee has a clock in and out button that allows them to log when they get on and off of work. The waiter has the same buttons as the host, but will have different interactions on those pages. For example, the waiter can actually order from the menu if the customer isn't using the app, as well as update the list of orders when they have finished delivering an order.

Driver Interface:



As can be seen above, the driver has access to two buttons. He can see a list of available orders he can choose from, as well as the current order they may or may not have already accepted. An example of the current order screen is shown in the top right. The address will be displayed, as well as the time the order has been placed and the order details.

Order Integration with Employees Interface:

Fig. 1, Employee Side - Orders List

Selecting Orders List from the Chef Home Page or an Employee Home Page brings a list of orders. This can be viewed in Fig. 1. The uid is what is listed for every order. Here, we can see multiple orders have been placed with the same customer uid. Clicking on an order brings us to Fig. 2.

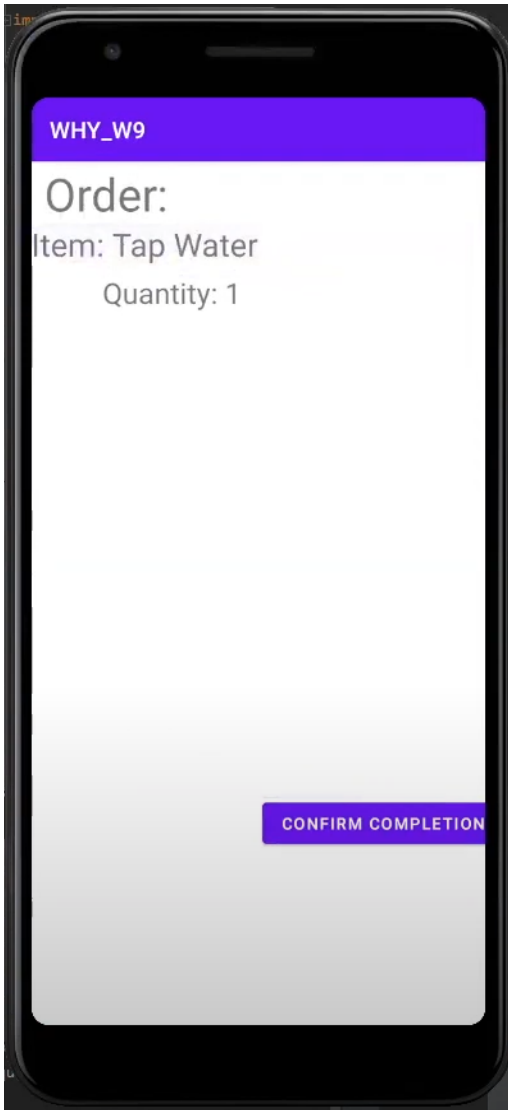


Fig. 2, Employee Side - Orders Details

In Fig. 2, we see the details that are provided per order. We see the item name and the quantity. This information is pulled from firebase. Once an employee, in this case the chef, confirms order completion, the order disappears from firebase. Fig. 1, which originally displayed six orders, now will display five. However, the order has now been marked for delivery and will reappear in firebase in a different part of the database. The waiter can now access this order similar to the chef, as seen in Fig. 1. The details page however, appears a little differently. This can be seen in Fig. 3.

Fig. 1 and Fig. 2 also showcase what Orders List and Orders Details appears like for the manager. The manager views a list of current orders as can be seen in Fig. 1. Selecting any one of these orders brings us to the page in Fig. 2. From here, the manager can select the “CONFIRM COMPLETION” button. This works in much the same way as the previously mentioned Chef Account.

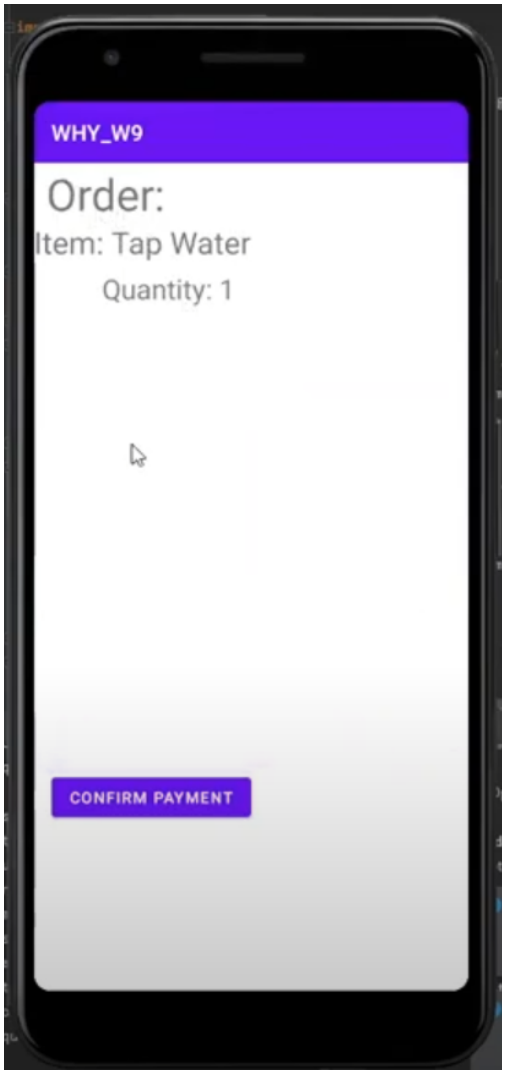


Fig. 3, Employee Side - Orders Details (Waiter and Driver)

Fig. 3 showcases what Orders Details appears like for waiters. The waiter selects the “CONFIRM PAYMENT” button to ensure the order has been paid for. Once selected, we will then proceed to Fig. 2 in order to confirm completion. Once the “CONFIRM COMPLETION” button is selected, the order is deleted from the completeOrders section of the database. Now, refreshing the Orders List on Waiter’s Side will show us one less order.

Fig. 3 also showcases what Orders Details appears like for drivers. The design is as follows: Orders marked for delivery by the customer will be sent to the chef. The chef completes the customer’s order. On completion the order moves from the requests section of the database to the compleOrders section of the database. Aside from these differences, Driver Side Orders are handled exactly like Waiter Side Orders.

Fig. 3 also showcases what Orders Details appears like for the host. The design is as follows: Orders marked for takeout by the customer will be sent to the chef. The chef completes

the customer's order. On completion the order moves from the requests section of the database to the compleOrders section of the database. Aside from these differences, Waiter Side Orders are handled exactly like Waiter Side Orders and Driver Side Orders.

Host Interface:

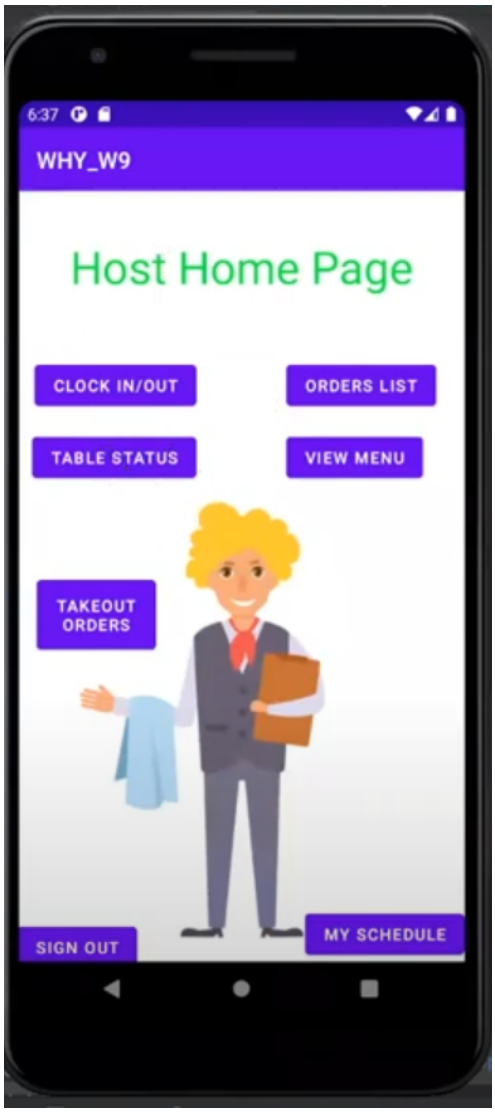


Fig. 1, Host Home Page

In Fig. 1 we can see the Host Home Page. Selecting the "TABLE STATUS" button brings us to Fig. 2.

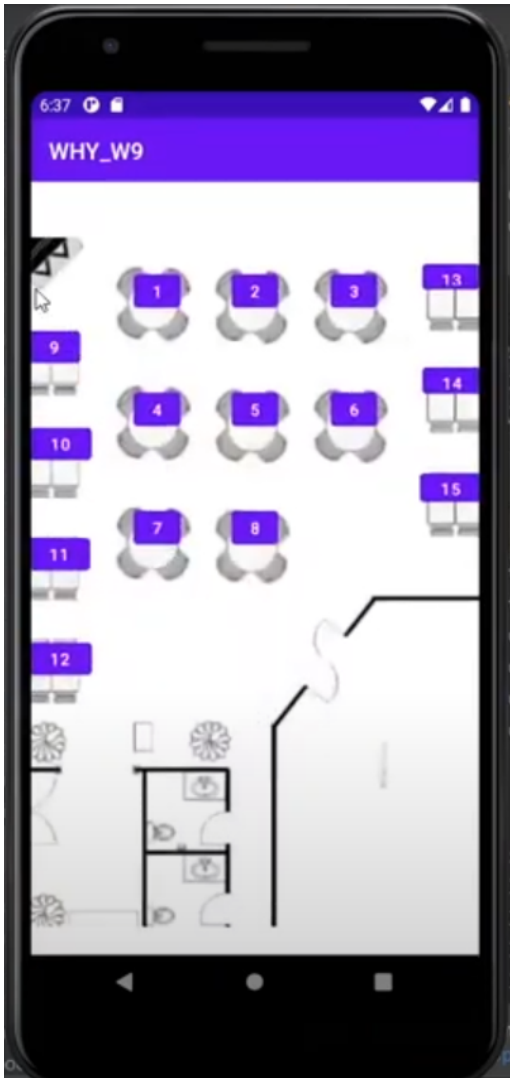


Fig. 2, Table Status Overview

Fig. 2 showcases an overview of the table status. Selecting a table brings the user to Fig. 3, Table Details.

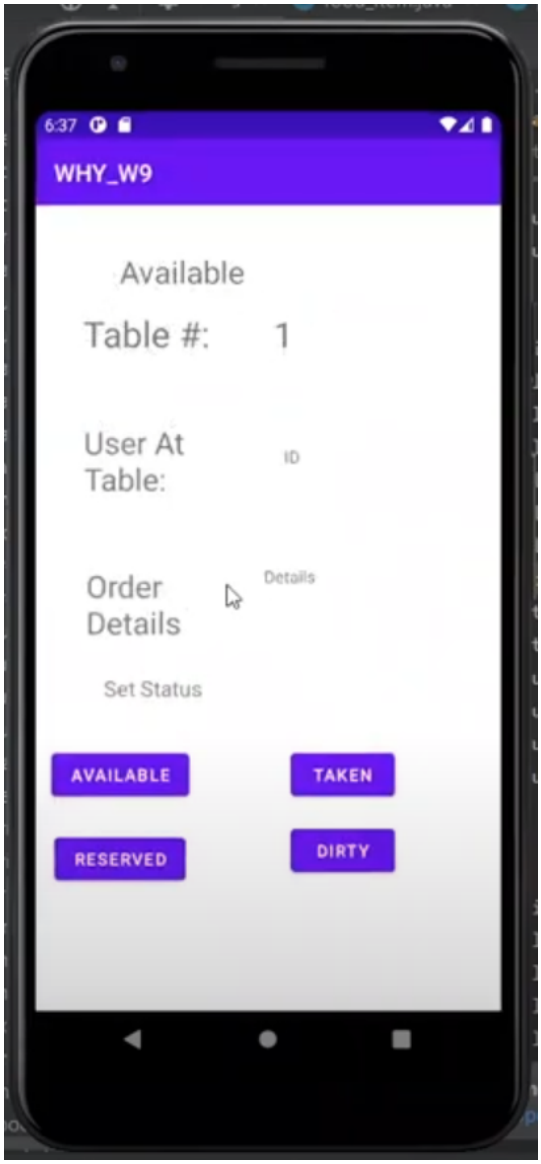


Fig. 3, Table Details

Fig. 3 showcases the details of the table selected. The user can see whether or not the table is available, the table number, which user is at the table, and details of the order. However, the user at table and order details features still need to be implemented. The host can set the status of the table to any of the options listed towards the bottom of the screen. Updates are stored in the realtime database.

Employee Schedule Interface - Employee Side:

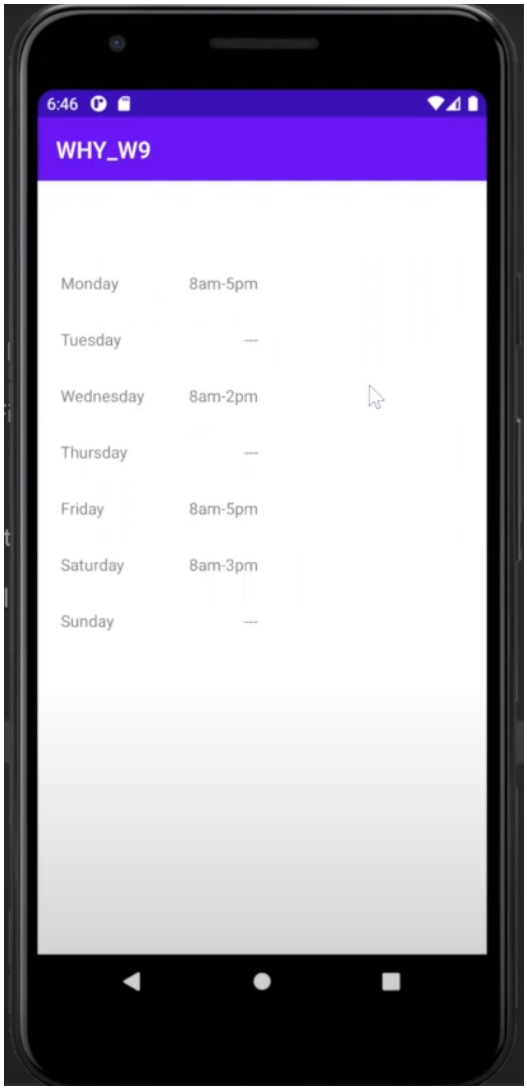


Fig. 1, Employee Schedule - Employee Side

Employees can log in to their account and select the “Employee Schedule” button on their respective Home Page. This will bring them to the employee schedule page that we see in Fig 1.

12. Design Of Tests

12.1) Unit Testing (16 Use Cases Covered)

The following are the test cases to be used for unit testing:

TC - 1: Tests login functionality and accuracy

Test Case Identifier: TC-1
Use Case Tested: UC-9

Pass Criteria: If the user can login with a correct email and password combination.
Fail Criteria: If the user can login with an incorrect email and password combination.

Input Data: Email, Password

Test Procedure	Expected Result	Actual Result
Step 1: The user (a customer) enters a correct combination of their email and password into the login screen and selects login.	The app accepts the combination and allows the user to login. The app should bring the user to the customer main page.	Follows expected result. The app brings the user to the correct screen.
Step 2: The user (an employee) enters a correct combination of their email and password into the login screen and selects login.	The app accepts the combination and allows the user to login. The app should bring the user to the employee main page.	Follows expected result. The app brings the user to the correct screen.
Step 3: The user (a manager) enters a correct combination of their email and password into the login screen and selects login.	The app accepts the combination and allows the user to login. The app should bring the user to the manager main page.	Follows expected result. The app brings the user to the correct screen.
Step 4: The user enters their correct email and purposely enters an incorrect password.	The app should not accept the combination and allows the user to attempt to login again and notifying an error message.	Follows expected result. The app allows the user to attempt to login again and notifies the user with an error message.
Step 5: The user enters an email that they did not use to create an account along with the correct password associated with their real account.	The app should not accept the combination and allows the user to attempt to login again and notifying an error message.	Follows expected result. The app allows the user to attempt to login again and notifies the user with an error message.

TC - 2: Tests user's ability to select dine in

Test Case Identifier: TC-2
Use Case Tested: UC-15

Pass Criteria: The user will click the Dine-In button and show all tables to choose.
Fail Criteria: The user will click on the Dine-In button and the wrong screen appears or if no

screen appears when the button is clicked. Input Data: Dine-In Button Selection		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user (a customer) clicks on the Dine-In button from the main customer screen.	The user will be able to see a screen which shows all tables to choose from to dine in.	Clicking dine-in brings you to the payment screen and is located after selecting your items. See Table Status for this expected result.

TC - 3: Tests users ability to takeout food from the menu

Test Case Identifier: TC-3 **Use Case Tested: UC-14** Pass Criteria: When the Takeout button is pressed, it brings up the takeout menu and add items to the list. Fail Criteria: When the Takeout button is pressed, it will not bring up the menu or will not add items. Input Data: Take-Out Button Selection		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user (a customer) clicks on the Takeout button from the main customer screen.	The Takeout menu will appear.	The menu will appear and then can choose takeout as an option for eating. Process is flipped, but works just as expected.
Step 2: The user will add multiple items from the menu to their list.	Items added will be added (with price and quantity) to the list.	Follows Expected Result
Step 3: The user will delete a few items from the list.	Items will be removed (with price and quantity) from the list.	Follows Expected Result

TC - 4: Tests user ability to reserve a table

Test Case Identifier: TC-4 **Use Case Tested: UC19** Pass Fail Criteria: User has to successfully be able to reserve a specific table at their choice

of date/time, given table availability Input Data: A date, time, and specific table selection		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user should click on Reservation Option that shows up in the menu screen	A screen that shows available tables as well as location within the restaurant pops up.	Follows Expected Result
Step 2: The user should select a Table from the Available tables list.	A new screen pops up that has times where the table is unable to be reserved shown, as well as two input boxes.	A layout with the arrangement of tables and table numbers appears. Users can select a table to view table status.
Step 3: The user should then input the date the table is to be reserved for in the appropriate input text box.	If the date is an invalid input, then produce an error message and prompt another input.	Have not implemented date functionality. Currently only works in real-time.
Step 4: The user should then input the time the table is to be reserved for in the appropriate input text box.	If the time is an invalid input, then produce an error message and prompt another input. Else, proceed to the confirmation screen.	Have not implemented time functionality.Currently only works in real-time.
Step 5: The user should then confirm reservation, when a confirmation screen pops up.	Correctly update the database for table availability with according time and date.	Users are able to set table status. Table status options are Available, Taken, Reserved, and Dirty. Database is updated with correct and current status.

TC - 5: Tests ability to order from menu

Test Case Identifier: TC-5 **Use Case Tested: UC15, UC16, UC17** Pass Fail Criteria: A user has to be able to successfully choose items on the menu and add to their order Input Data: Select items from menu as well as any customizations, and have it add to your order.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user can click on	A menu interface will pop up,	Follows expected result

the menu option to view the menu	allowing the user to view food items on the menu, as well as additional info like price/calories	minus the additional information like calories.
Step 2: The user can view different parts of the menu, and choose items they wish to order	The user can navigate the menu to see different parts like lunch, drinks, etc. Every time they select an item it will add to their order and they get an updated total bill.	Follows expected result
Step 3: The user can complete and confirm their order on a confirmation order screen that displays their total order	The user will be brought to a confirmation page that has their total order, and allows them to confirm it. Once this is done it brings them to the payment screen.	Follows expected result

TC - 6: Tests ability for employees to view table status

Test Case Identifier: TC-6 **Use Case Tested: UC-20** Pass Fail Criteria: Employees should be able to check the current table statuses, or of a certain time and date Input Data: time and date		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: Click on table status option	This will bring up the current table status screen.	Follows expected result.
Step 2: Although it will bring up the current table status screen, users can click on the “specific” button.	The user will be able to see all the current table statuses, and if they click on the specific button, input text boxes will appear that will allow the user to input a date and time.	No specific feature has been implemented. Viewing table status is done by selecting a table individually.
Step 3: Users can input specific time and date in the according input textboxes.	If they wish to check the status for a specific time, they will be directed to another screen after they input the data. THis screen will show the table statuses according to the time and date imputed.	Have not implemented time and date functionality. Currently only works in real-time.

TC - 7: Tests user ability to select delivery

Test Case Identifier: TC7
Use Case Tested: UC16

Pass Fail Criteria: Once the user selects their order and pays for their food, they should be able to get an update on their order status that will tell them how long it will take for their food to arrive.

Input Data: An order.

Test Procedure	Expected Result	Actual Result
Step 1: A user logs in with their account email and password.	The user should see a screen where they have 3 dining options.	Follows expected result. Users can select menu, add items, then choose their dining options.
Step 2 : The user selects delivery, places their order, and pays for their order.	The user shall receive a confirmation.	Follows expected result.
Step 3: A chef account logs in and checks their order list.	The chef should be able to see the recently placed order.	Follows expected result.
Step 4:The chef updates the order by making it ready for delivery.	The order should be removed from the chefs order list.	Follows expected result.
Step %: A driver logs in and checks the list of orders ready for delivery.	Once the chef completes the order the driver should be able to see this order and accept it to take the responsibility of delivering it.	Follows expected result. Once a customer has placed an order and a chef has completed that order the driver is able to check the list of completed orders and choose one to deliver.

TC - 8: Tests payment is implemented correctly and that the order is confirmed

Test Case Identifier:TC8
Use Case Tested: UC17

Pass Fail Criteria: If the user enters a card number, cvv , and expiration date of which all are valid lengths, then the user's order is confirmed and the next screen is shown.

Input Data: A numeric number that is 13-19 digits long for the card number, a 3 digit cvv number, and a 4 digit expiration date.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: User enters a 16 digit card number, 2 digit cvv, and a 4 digit expiration date.	The app will stay at the same screen and tell the user that the card information is invalid because the cvv should be 3 digits.	Follows expected result.
Step 2: User enters a 16 digit card number, 3 digit cvv, and a 4 digit expiration date.	The user shall be taken to the confirmation screen.	Follows expected result.

TC - 9: Tests users ability to rate food

Test Case Identifier: TC9 **Use Case Tested: UC18** Pass Fail Criteria: When a customer submitted a rating it should be shown in the application under a reviews section. Input Data: Some text giving an opinion on the food as well as an integer rating from 1-5.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1:User enters 5 star rating for a burger. Then clicks submit.	Confirm submission and the data will go to the database and manager.	Have not gotten this far.
Step 2: A manager logs in and checks the ratings.	The manager should see a recent rating submission.	Have not gotten this far.

TC-10: Tests managers ability to view reports regarding restaurant

Test Case Identifier:TC10 **Use Case Tested: UC21** Pass Fail Criteria: Once the manager logs in they should be able to check different stats on the restaurant such as meal popularity. Input Data: None		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: Manager logs in to	The manager should see a list	Follows expected result.

their account	of options such as check analytics.	The manager is able to select Check Analytics to view Food Analytics, Daily Traffic, and Operations Info.
Step 2: Manager clicks check analytics.	The manager should see the number of times an item was sold in a chart.	Follows expected result. The manager can view the number of times an item was sold in bar graph view per month.

TC-11: Tests users ability to register a new account

Test Case Identifier: TC-11 **Use Case Tested: UC-11** Pass Criteria: A user will be able to create a new account. Fail Criteria: A user will fail to create an account. Input Data: Email, Password.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user will enter their email (unused and real) and password (with requirements) and hit submit.	The system will allow the user to create an account and notify the user. The user will be able to use the app's additional features.	Follows expected result.
Step 2: The user will enter a used or incorrectly formatted email with a password that matches the requirements.	The system will notify the user that the email is incorrectly formatted or already used and allow them to retype it.	Follows expected result.
Step 3; The user will enter an unused correctly formatted email with a password that does not meet the requirements.	The system will notify the user that the password does not meet the requirements and allows them to retype it.	Follows expected result.

TC-12: Tests estimation time for food arrival

Test Case Identifier: TC12 **Use Case Tested: UC3** Pass Fail Criteria: Once an order has been placed, the user will be provided an estimated time for food arrival.

Input Data: Type of order, and food items ordered.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: User enters their order type	The app will be brought to the menu where the user can then choose what items to order	Follows expected result.
Step 2: User selects their order and confirms it	The user shall be taken to the confirmation screen.	Follows expected result.
Step 3: The user will then confirm the order	The user will then be brought to a screen that displays an estimated time of arrival calculated based off the order type, the food items in the order, as well as the amount of orders queued in the kitchen.	Have not gotten this far.

TC-13: Tests users ability to edit/delete account

Test Case Identifier: TC-13 **Use Case Tested: UC-12, UC-13** Pass Criteria: The user will be able to delete or edit their account successfully Fail Criteria: The user will not be able to delete their account or edit any of their account information. Input Data: Password.		
Test Procedure	**Expected Result**	**Actual Result**
Step 1: The user will try and change their password.	The system will notify the user of a successful password change	Follows expected result.
Step 2: The user will try and change their username.	The system will notify the user of a successful username change.	Follows expected result.
Step 3: The user will try to delete their account.	The system will notify the user of a successful deletion of the account and will return the user to the start screen.	Follows expected result.

All of the test cases specified above would be tested as such test cases are important for successful operation of the application and therefore must be prioritized in testing. Condition coverage is used to allow every condition to output with true or false outcomes at least once. For example, during the payment process, the condition would turn true if the payment has been successfully verified, and the condition would turn false if the payment has been attempted from an invalid card or a card with low balance. State-based testing method is also used for the testing of expected states and actual states for the test cases. For example, during the login process, four states would be used: locked, accepting, blocked, if the user enters the correct username and password, the user would be sent to unlocked mode. If the user enters the invalid username or password, the user would be sent to accepting mode, giving the user another chance to enter the correct username and password. If the user enters the invalid username or password five times, then the user would be sent to block mode and would need to create a new username or password. In addition, statement coverage is used for each statement to be executed at least once as all of the test cases would be tested and all statements would be executed at least once by one of the test cases.and unlocked. When the user enters the login page, the user is in the locked mode.

12.2) Integration Testing

Our group has decided to use the bottom-up integration method from the horizontal integration testing for our system. Using the bottom-up integration, the testings are first proceeded on the bottom-level modules, which consist of the subcomponents of the entire system, and kept proceeded in stages up to the top-level module, which consist of the complete system. Since our system consists of many subgroups, it is necessary that such subgroups are well-organized so that the complete system does not become complex. If other methods besides the bottom-up integration method are attempted to be used, then the implementation would be mixed up between each subgroup and the debugging or recovery process would become more complicated. Thus, it is important to break down the whole system into subgroups and begin the implementation and testing from those lower-level modules in order to form the simplicity. As the specific information from each subgroup would be organized, it would be easier to organize the higher-level modules based on the information shown from the lower-level modules. This bottom-up integration method allows the implementation to be organized and thus allows easier debugging and faster recovery when the error occurs in the system as it is easier to track the part that caused the error in the system.

The example of how the bottom-up integration testing will be conducted is explained here. The implementation of the login system would be tested to make sure that when the user enters the username and the password correctly, the user is sent to the main page of the application. In the login subcomponent, the username and the password must be able to be correctly inputted and verified. Then, the interaction between the login subcomponent

(lower-level module) and homepage component (higher-level module) must be successful in order for the user to be directed to the main homepage successfully.

13.History of Work

Gantt Chart:
https://docs.google.com/spreadsheets/d/1Rx3MXlCmaH2qySpwvPafgbbn4g_-N3drkd9JR5_Ikc4/edit#gid=0

Total Contributions Breakdown:
https://docs.google.com/spreadsheets/d/1xWIfiGjpQ62CyRUJnizeW7X-8L-qAOV8YohhL6ydlmk/edit#gid=0

Contributions for Report 1:

Jason Chan:

- **Problem Statement- 33%**
  - Statements for Host, Waiter, Chef
- **Decomposition into Subproblems- 90%**
  - Almost all if not all, as well as changes based on Prof's and TA;s concerns.
- **User Interface Requirements- 5%**
  - Created Diagrams for "Chef", as well as table of requirements
  - Helped edit requirements for other employee subgroups
- **Project Management- 23%**
  - Recorded the work already completed in the Gantt Table, as well as help plan the roadmap for Report 2
- **Communication- 11%**
  - Helped start and lead meetings
- **Actors and Goals- 25%**
  - Helped fill out descriptions for some of the actors and goals
- **Traceability Matrix- 25%**
  - Helped create matrix and fill out the data
- **Fully Dressed Description- 33%**
  - Wrote description for UC-9
- **User Effort Estimation- 50%**
  - Wrote user scenario 2.
- **Feedback2 edits- 14%**
  - Met up with group to discuss and edit Report based off Feedback#2
- **Mapping Subsystems to Hardware- 100%**
  - Nothing much, just a small statement.
- **Project size estimation- 20%**
  - Wrote the complexities for TCF

    - Filled out description and complexity for various actors for UUCW
- **Feedback edit 3**

David Joseph

- **Glossary Of Terms - 70%**
    - All the definitions from “Android” to “Menu” and “Software Engineer” to “Waiter/Host”
    - Also added definitions for “Rewards/Discounts”, “Cook Time”, and “Dish Popularity”
- **User Interface Requirements - 85%**
    - Created diagrams for the “Login screen”, “Owners Screens”, and “Customers Screens”
    - Also created diagrams for “Waiter/Host” and “Busboy” under the employees section.
    - Worked on the Chef REQ table
- **Project Management - 1%**
    - Helped a little with creating the sheet, adding and cleaning up rows/columns.
- **Communication - 11%**
    - Helped find times to meet with most of us. Consistently meets in weekly meetings
- **Stakeholders - 100%**
- **Actors and Goals - 70%**
    - Finished most, but not all of them.
- **Traceability Matrix - 25%**
    - Helped create matrix and fill out data
- **Fully Dressed Description - 33%**
    - Worked on UC-#9
- **Feedback #2 - 14%**
    - Worked on #3
- **Identifying Subsystems - 100%**
- **Architecture Styles - 100%**
- **Feedback edit 3 - Case diagram chart - 100%**

Ashirvadh Bhupathi

- **Problem Statement - 33%**
    - Statement for Manager and Driver
- **User Interface Requirements - 5%**
    - Created diagrams for the drivers
    - Did the UI requirements that pertain to the drivers
- **Project Management - 23%**
    - Planned the work needed to be done on Report 1 part 2 and Part 3 using the Gantt Table
- **UI Diagrams - 50%**

    - Worked on UC-1 and completed UC-3
- **Fully Dressed Description**
    - Fleshed out some alternatives success scenarios
- **Project size estimation based on use case points - 25%**
    - Completed the Unadjusted weight

Juergen Benitez:

- Made edits to Manager Section of **Problem Statement**
- **Glossary of Terms**- 30%
- **Enumerated Functional Requirements**-20%
- **Project Management(Coding plan**)-30%
- **Use cases** -33%
- **Traceability Matrix** -25%
- **User Effort Estimation** -50%
- **Feedback 2 Edits** -14%
- Worked on **Project Size Estimation-40%**

Ryan Van Duren

- **Decomposition into subgroups - 100%**
    - Edited “Customer” subgroup to answer the questions posed by the TA
- **Business Goals - 100%**
    - Rewrote the Business Goals section to accurately reflect the changes made to the project.
- **System Sequence Diagrams - 50%**
    - Created for UC-10 and created UC-6 with Pavan.
- **Hardware Requirements - 100%**
    - Wrote the Hardware requirements section
- **Updated customer_reservation.xml - 100%**
    - Completely changed the customer reservation UI to match our plans for reservations going forward
- **Bug fixes and updates to several home pages - 50%**
    - Tested and made bug fixes to busboy, host, and driver home pages
- **Employee Registration - 10%**
    - Updated the manager UI for creating employee accounts
- **Readme - 100%**
    - Updated README.txt

Hojun Son

- **Enumerated Nonfunctional Requirements - 100%**
    - Created a table of nonfunctional requirements with priority and description for each of goals.
- **User Interface Requirements - 5%**
    - Created tables for Manager, Waiter, and Busboy.
- **Actors and Goals - 5%**

    - Helped fill out the actors and goals description.
- **Traceability Matrix- 25%**
    - Helped fill out the traceability matrix.
- **Fully Dressed Description - 33%**
    - Helped fill out a description of UC-2.
- **Feedback 2 Edits - 14%**
    - Helped edit the report based on feedback #2.
- **Project Size Estimation - 15%**
    - Calculated each technical factor value and the TCF value.

Pavan Kunigiri

- **Enumerated Functional Requirements - 80%**
    - Created Key and Table. Wrote down the description for most of the requirements. Ranked requirements for priority.
- **Project Management - 23%**
    - Recommended Gantt Chart template used and edited document with necessary information.
- **Decomposition into Subproblems - 5%**
    - Added detail to subproblems to address problems TA addressed in his email.
- **Feedback 2 - 14%**
    - Edited requirements.
- **System Sequence Diagrams - 50%**
    - Created diagram for UC-6. Created diagram for UC-5 with Ryan.
- **Global Control Flow - 100%**

Krishna Patel

- **Problem Statement - 33%**
    - Statement for Customer and Busboy
- **Cover Page - 10 %**
- **Communication - 10%**
- **Use Cases - 33%**
- **Diagram - 100%**
- **Feedback 2 edits - 14%**
- **Network Protocol - 50%**

Josh Chopra

- **Cover Page - 90%**
- **Use Cases - 33%**
- **Network Protocol - 50 %**
- **Feedback 2 edits - 14%**

Justin Davis

- **Communication - 11%**
- **Software Choices**

    - I developed the idea to use Android and Java programming language
- **Research**
    - Helped research past projects in determining what others have done to improve upon on our own
- **UI Diagrams - 50%**
    - Did half of UC-1 and all of UC-2
- **Fully Dressed Descriptions - 5%**
    - Added an alternate success scenario for UC-1

Contributions for Report 2:

Jason Chan:
- Association Definition- 100%
- Traceability Matrix- 100%
- Mathematical Model- 10%
- Interaction Diagrams- 25%
- Algorithms- 23%
- User Interface Design and Implementation- 85%
- Design of tests- 30%

David Joseph
- Data Model - 100%
- Attribute Definition - 10%
    - Added 3 new terms
- Interaction Diagrams - 25%
    - Edit/Delete Use Cases
- Concurrency - 100%
- User Interface Design and Implementation - 15%
- Design of tests - 40%

Ashirvadh Bhupathi
- Mathematical model - 30%
- Interaction Diagram - 25%
    - Manager Reports UC
- Proofreading - 50%
- Breakdown of Responsibilities - 100%

Juergen Benitez:
- System Operations Contracts - 67%
- Mathematical Model- 30%
- Class Diagram-100%
- 3b - 100%
- Algorithms (a couple)
- Design of Test-30%

- User Interface Design(2 pics)

Ryan Van Duren
- Concept Definition - 100%
- Proofreading - 50%

Hojun Son
- System Operation Contracts - 33%
- Mathematical Model - 10%
- Interaction Diagram - 25%
    - Dine-In UC
- Algorithms - 15%
    - Food Inventory, Account Login
- Test Coverage - 100%
- Integration Testing - 100%

Pavan Kunigiri
- Mathematical Model: - 30%
    - Employee Clock in and out, Food Order Status, Update Menu, Display Daily Sales/Profits
- Part 2: Traceability Matrix: 100%
    - Provided matrix as well description below
- Part 2: Feedback #2: 80%
- Part 3: Algorithms: 31%
- Part 3: Data Structures: 100%
- Part 3: Feedback #3: 100%

Krishna Patel
- Attribute Definition - 90%

Josh Chopra
- Persistent Data Storage - 90%

Justin Davis
- Mathematical Model: -20%
    - Fixed formatting on all, and did Inventory and Account Creation
- Persistent Data Storage - 10%

Contributions for Demo 1:

Contributions (github ID)	Code (Up to Demo 1)	Video for Demo 1
Jason Chan(jc2375)	55%	22%

David Joseph(drj63)	27%	22%
Ashirvadh Bhupathi	0%	6%
Hojun Son	0%	6%
Josh Chopra	0%	0%
Krishna Patel(ksp115)	1%	7% + 9% (editing) = 16%
Juergen Benitez (jbz8)	17%	22%
Justin Davis	0%	0%
Pavan Kunigiri	0%	6%
Ryan Van Duren	0%	0%

****Code Contributions were calculated referencing contributions page of Github Repository**

****Note all our code was created from scratch and DID NOT copy any of 2018/2019 sample code**

- **Subproblem 1: Customer**

- Ordering Interface
    - Customers should be able to order food from the menu, and customize their orders. Jason (Base Menu Complete, Cart Integrations and Orders In Progress)
    - Account login system where they can get rewards/discounts for coming frequently. (guest system also will be implemented) David (Login/Register complete, Rewards in progress)
    - Takeout/Delivery/Dine in options Jason (Implementation needed)
- Food tracking interface(Simplified Version)
    - Able to see when their order is being started as well as cook time, and estimated time of completion (Implementation needed)
- Delivery tracking interface(Simplified Version)
    - Able to track when the order has left the restaurant, and estimated time of arrival. David (in progress)
- Data Analysis(recommended food items)
    - Suggested food items based off popular menu choices (implementation needed)
- Table Occupancy interface(Availability of Table)
    - Able to see how many tables are available with X amount of seats per table. Jason (In Progress)

- **Subproblem 2: Manager**
    - Delivery/food tracking interface(Based off completion)
        - Manager can see when an order has completed and when a delivery has been completed (implementation needed)
        - Has admin options if he wants to see more details that the waiter and chef see.
            - Add or remove items from the menu Jason (Implementation needed)
            - Estimated food completion time Jason (Implementation needed)
            - Whether an order has been started (implementation needed)
    - Table Occupancy interface(shared)
        - Able to see how many tables are available with X amount of seats per table. Jason (in progress)
        - (Admin): can update table status to any status. Jason (completed)
    - Finance tracking(Full Version)
        - Keeps track of all the finances throughout the day, as well as bookkeeping Juergen (Implementation needed)
        - Able to see when a table has paid and how much their order was/receipts. Juergen (Implementation needed)
    - Data Analysis (Full Version)

        - Data:Food ordered by time, throughout the day/ Food ordered by day of the week. Juergen (Implementation needed)
            - A bar graph will be produced.
            - Divided by time of day and day of the week
        - Data: Number of customers at various times throughout the day/week Jeurgen (Implementation needed)
    - Clocking hours (Full Version) Juergen (Base model completed, further integration in Progress)
        - Able to see which employee has clocked in at what time and how long they have been working, as well as normal shift hours.

- **Subproblem 3: Employee (Waiter/Chef/Driver/Busboy)**
    - Table Occupancy interface(shared)
        - Able to see which tables are occupied/not, which tables are dirty and clean. (implementation needed)
        - Waiters: Can update table status as dirty (implementation needed)
        - Host: Can update table status as occupied (implementation needed)
        - Busboy: Can update table status as Clean (implementation needed)
    - Food tracking interface(Full version)
        - Able to track all the food orders(whether they have started cooking, almost done, complete, etc.) David (in progress)
    - Delivery tracking interface(Full Version)
        - Able to see whether the food has been delivered and track delivery David (in progress)
    - Finance tracking(Whether table has paid or not)
        - Tracks the bills, and whether a table has paid or not David (in progress)

*All functionalities will need to be used in combination with another. Examples of functionalities that will be in combination with another would be: food tracking, payments, data from customer orders, clock in hours, table availability, and surveys/data.

JAVA AND XML FILES

busboy_home: Krishna - David - Jason
busboy_tablestatus: Krishna - Jason
BusboySchedule: Juergen
Chef_home: Jason - David
ChefSchedules: Juergen
customer_home: Jason - David
Customer_reservation: Jason
Delievery_adress: Jason

Driver_availailbe_orders: David
Driver_current_order: David
Driver_home: David
DriverSchedules: Juergen
EmployeePayroll: Juergen
EmployeeSchedules: Juergen
ExpandableListView Adapter: Juergen
host_home: Jason - David
Login: Jason - David
MainActivity: Jason - David
ManageEmployeeOptions: Juergen
manager_home : Jason - David
Menu_home_canorder: Jason
Modify_account: David
order_type: Jason
Payment_type: Jason
Register: Jason - David
Waiter_home: Jason - David
WaiterSchedules: Juergen

Demo Documentation Contributions:
Presentation slides- Krishna, Pavan, Ashirvadh, Hojun, Josh
User Documentation- Juergen- 75%, Krishna- 25%
Brochure - David -100%
Technical Documentation - Juergen - 100%
Code - Jason, David, Juergen, Krishna
Unit Testing - Jason - 100%
Integration Testing - Jason -70% Krishna - 30%
Data Collection - David- 100%

Jason is in charge of coordination. The integration testing will be done by Jason, Juergen, and David.

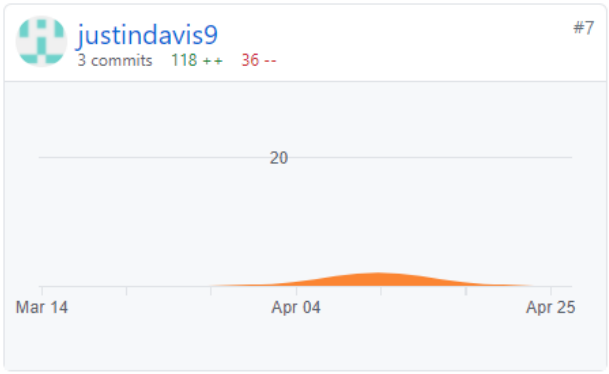
Contributions for Demo 2:

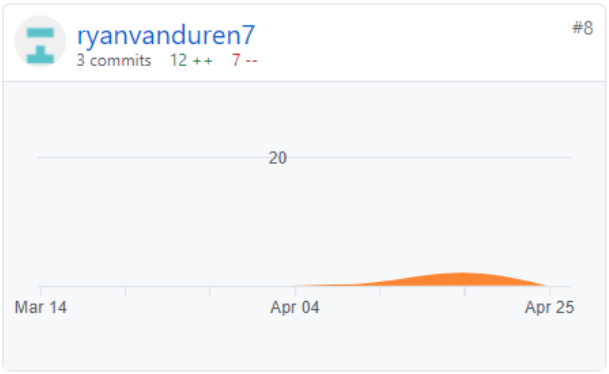
Contributions (github ID)	Code (Demo 1 to Demo 2)	Video for Demo 2
Jason Chan(jc2375)	20.5% (2420/11797)	55%
David Joseph(drj63)	19.8% (2336/11797)	13%
Ashirvadh Bhupathi(Ash4394)	8.6% (1010/11797)	10%
Hojun Son	0% (0/11797)	0%

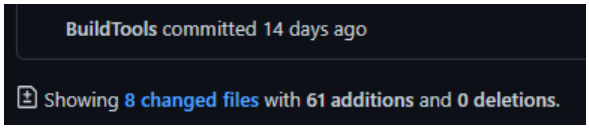
Josh Chopra	0% (0/11797)	0%
Krishna Patel(ksp115)	0% (0/11797)	Video editing
Juergen Benitez (jbz8)	35.6% (4195/11797)	16%
Justin Davis	3.7% (433/11797)	0%
Pavan Kunigiri (pavkun)	6.4% (760/11797)	6%
Ryan Van Duren (rlv31 & ryanvanduren7)	5.5% (643/11797)	0%

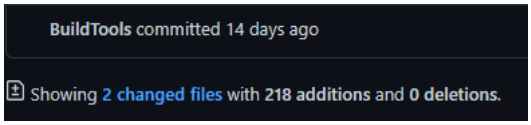
****Code Contributions were calculated referencing contributions page of Github Repository. Calculated by adding the number of added lines with the number of deleted lines since Demo 1. This sum is then placed over the total number of added lines and deleted lines since Demo 1. This gives us the percentage found in Column 2.**
****Note all our code was created from scratch and DID NOT copy any of 2018/2019 sample code**


Jc2375
56 commits 5,382 ++ 875 --
#1
20
Mar 14
Apr 04
Apr 25
drj63
52 commits 3,128 ++ 1,081 --
#2
20
Mar 14
Apr 04
Apr 25
jbz8
18 commits 4,589 ++ 830 --
#3
20
Mar 14
Apr 04
Apr 25
rlv31
8 commits 305 ++ 319 --
#4
20
Mar 14
Apr 04
Apr 25
Ash4394
5 commits 893 ++ 117 --
#5
20
Mar 14
Apr 04
Apr 25
pavkun
4 commits 718 ++ 42 --
#6
20
Mar 14
Apr 04
Apr 25









- **Subproblem 1: Customer**
  - Ordering Interface
    - Customers should be able to order food from the menu, and customize their orders. Jason (Revamped Menu Complete, Cart Integrations and Orders Integration Complete). Pavan (Cart Interface Complete, UI Complete)
      - Customer should be able to select from a category of foods Jason (Complete)
      - Customer should be able to select a menu item bringing them to a screen with details of food Jason (Complete)
      - Customer should be able to add and delete item from cart Pavan (Complete)
      - Displaying numbers of each item ordered Pavan (Complete)
      - Tallying and displaying cart total on cart update Pavan (Complete)
      - Customer should be able to increase or decrease the quantity of the item they wish to add to cart Jason (Complete)

        - Customers should be able to view carts and see their order. Pavan (Complete)
    - Account login system where they can get rewards/discounts for coming frequently. (guest system also will be implemented) Ashirvadh Finished, David initially (Login/Register complete. rewards in progress)
    - Takeout/Delivery/Dine in options
        - Customer should be able to select one of the three options Jason (Complete)
            - Payment integration(paypal and credit) David (Finished)
            - Order paid status(for confirm payment integration) Jason (Complete)
        - Waiter should be able to see list of dine in orders
            - Be able to confirm payment and orders Jason (Complete)
        - Host should be able to see list of takeout orders
            - Be able to confirm payment and order Jason (Complete)
        - Driver should be able to see list of delivery orders
            - Be able to confirm payment and order Jason (Complete)
- Delivery tracking interface(Simplified Version)
    - Able to track when the order has left the restaurant, and estimated time of arrival.
- Data Analysis(recommended food items)
    - Suggested food items based off popular menu choices Jurgen(Data analysis complete)
- Table Occupancy interface(Availability of Table)
    - Able to see how many tables are available with X amount of seats per table. David + Ashirvadh Bhupathi (finished)
- Customer Reservation
    - Allow customers to select a number of guests and a reservation time to reserve a table for the future. Ryan + Ashirvadh (in progress)

- **Subproblem 2: Manager**
    - Delivery/food tracking interface(Based off completion)
        - Manager can see when an order has completed and when a delivery has been completed Jason (Complete)
        - Has admin options if he wants to see more details that the waiter and chef see.
            - Add or remove items from the menu Jason (Complete)

        - Whether an order has been paid for or complete Jason (Complete)
    - Table Occupancy interface(shared)
        - Able to see how many tables are available with X amount of seats per table. David + Ash (finished)
        - (Admin): can update table status to any status. David + Ash (finished)
    - Finance tracking(Full Version)
        - Keeps track of all the finances throughout the day, as well as bookkeeping Juergen (Complete)
        - Able to see when a table has paid and how much their order was/receipts. Juergen (Complete)
    - Data Analysis (Full Version)
        - Data:Food ordered by time, throughout the day/ Food ordered by day of the week. Juergen (Complete)
            - A bar graph will be produced.
            - Divided by time of day and day of the week
        - Data: Number of customers at various times throughout the day/week Jeurgen (Complete)
    - Clocking hours (Full Version) Juergen (Complete)
        - Able to see which employee has clocked in at what time and how long they have been working, as well as normal shift hours.
    - Employee Registration
        - Manager can register employees Ashirvadh + Ryan (Complete)

- **Subproblem 3: Employee (Waiter/Chef/Driver/Busboy)**
    - Table Occupancy interface(shared)
        - Able to see which tables are occupied/not, which tables are dirty and clean. David + Ashirvadh (finished)
        - Waiters: Can update table status as dirty David + Ashirvadh (finished)
        - Host: Can update table status as occupied David + Ashirvadh (finished)
        - Busboy: Can update table status as Clean David + Ashirvadh (finished)
    - Finance tracking(Whether table has paid or not)
        - Tracks the bills, and whether a table has paid or not David + Jason (Finished)

*All functionalities will need to be used in combination with another. Examples of functionalities that will be in combination with another would be: food tracking, payments, data from customer orders, clock in hours, table availability, and surveys/data.

JAVA AND XML FILES (and other files)

AnalyticsOptions- Juergen
busboy_home: Juergen- Jason - Ryan
BusboySchedule: Juergen
Cart: Pavan
CartAdapter: Pavan
Category Jason
Chef_home: Jason
ChefSchedules:Juergen
Clockinout:Justin
Common: Pavan
CreateEmployee: Ashirvadh
Customers: David
Customer_home: Jason - David
Customer_reservation: Ryan
DailyTraffic Juergen
Database: Pavan
Delivery_address: David
Driver_availaible_orders:Jason David
Driver_current_order:Jason David
Driver_home: David Ryan
DriverSchedules: Juergen
EmployeePayroll: Juergen
EmployeeSchedules: Juergen
finishedCooking Jason
finishCookingDetails Jason
Food Jason + Pavan
Food_detail Jason + Pavan
Food_item Jason
FoodAnalytics Juergen
FoodList Jason
FoodviewHolder Jason
Global David
host_home: Jason - David- Ryan
HostSchedules- Juergen
Inventory-Juergen
Login: Ashirvadh Ryan
ManageEmployeeOptions: Ashirvadh Ryan
manager_home : Jason Ryan
Menu_home_canorder: Jason + Pavan
Menu_home_canorder2:Jason
Menu_home_canorder3:Jason
MenuViewHolder Jason
Myschedule Juergen
OperationsInformation Juergen

Order Jason + Pavan
Order_detail Jason
order_type: Jason
OrderDetailViewHolder Jason
OrderToDeliverDetailViewHolder Jason
OrdertoDeliverViewHolder Jason
OrderViewHolder Jason
Payment_type: David
PaymentDetails: David
PayrollModel Juergen
PayrollRecAdapter Juergen
RecyclerAdapter Jason
Register:David
Request Jason + Pavan
ScheduleModel Juergen
ScheduleRecAdapter Juergen
Table: David
Table_parameters Ashirvadh
Table_status: David + Ashirvadh
User_Parameters Ashirvadh
Waiter_home: Jason - David
WaiterSchedules: Juergen

EatItDB.db: Pavan
Readme: Ryan
David is in charge of coordination.

Contributions for Report 3:

Ryan Van Duren:

- Worked on user login code
- Documentation
    - Fixed table of contents 100%
- Updated Gantt Chart for report 3- 100%
- References Section - 100%
    - Collected references from group members and converted previous references to MLA format

Pavan Kunigiri:

- Added to section 1.c. (Data Model and Persistent Data Storage) from Report 2 in Report 3
    - Described data model using an ER diagram and explained the diagram as per part 1 of feedback for Full Report #2
- Updated Gantt Chart

    - Updated as per part 9 of feedback for Full Report #2.
- Updated User Interface Design and Implementation section from showcasing Demo 1 UI Design and Implementation to showcasing the UI Design and Implementation demonstrated in Demo 2.
    - Provided screenshots of the UI covered in Demo 2.
    - Provided paragraphs to supplement images in explaining design functionality and implementation.
    - Labeled figures.
    - Updated as per part 5 of feedback for Full Report #2.
- Updated Design of Tests section.
    - Fixed formatting.
    - Updated actual results section of tables with current results post Demo 2.
    - Updated as per part 7 of feedback for Full Report #2.

David Joseph:
- Summary of changes - 100%
- Helped with formatting of entire report 3

Krishna Patel:
- Documentation
    - Helped with formatting of entire report 3

Juergen Benitez
- Worked on feedback edits such as class diagrams, mathematical model descriptions, and how the app serves multiple users concurrently in the data structures and algorithms section.

Plan of Work

Gantt Chart (FULL)
https://docs.google.com/spreadsheets/d/1Rx3MXlCmaH2qySpwvPafgbbn4g_-N3drkd9JR5_Ikc4/edit#gid=0

14.References

Data Modeling: Conceptual vs Logical vs Physical Data Model, Visual Paradigm, online.visual-paradigm.com/knowledge/visual-modeling/conceptual-vs-logical-vs-physical-data-model/.

Dimitrovski, Stephan, et al. “Why W8 Restaurant Automation.” *eceweb1*, Rutgers University,eceweb1.rutgers.edu/~marsic/books/SE/projects/Restaurant/2018f-g4-report3.pdf.

EDMT Dev. “Making Order Food Application (Android + Firebase).” *YouTube*, YouTube, 27 Aug. 2017, www.youtube.com/playlist?list=PLaoF-xhnnrRW4lXuIhNLhgVuYkIlF852V.

Marsic, Ivan. “Software Engineering Project: Restaurant Automation.” *Software Engineering Project-Restaurant Automation*, Rutgers University, eceweb1.rutgers.edu/~marsic/books/SE/projects/Restaurant/.

Marsic, Ivan. “Software Engineering.” *Ece*, Rutgers University, www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf.