WHY WHINE JUST DINE

https://github.com/Jc2375/WHY_W9.git

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

Individual contributions breakdown

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%

Table of Contents

1.Analysis and Domain Modeling

Domain Model Diagram:
The Domain Model Diagram is listed under Part C.

a. Conceptual Model

i.Concept definitions

Responsibility	**Type**	**Concept**
R1: Coordinate activity between customers, chef, waiters, busboys, drivers etc	D	Controller

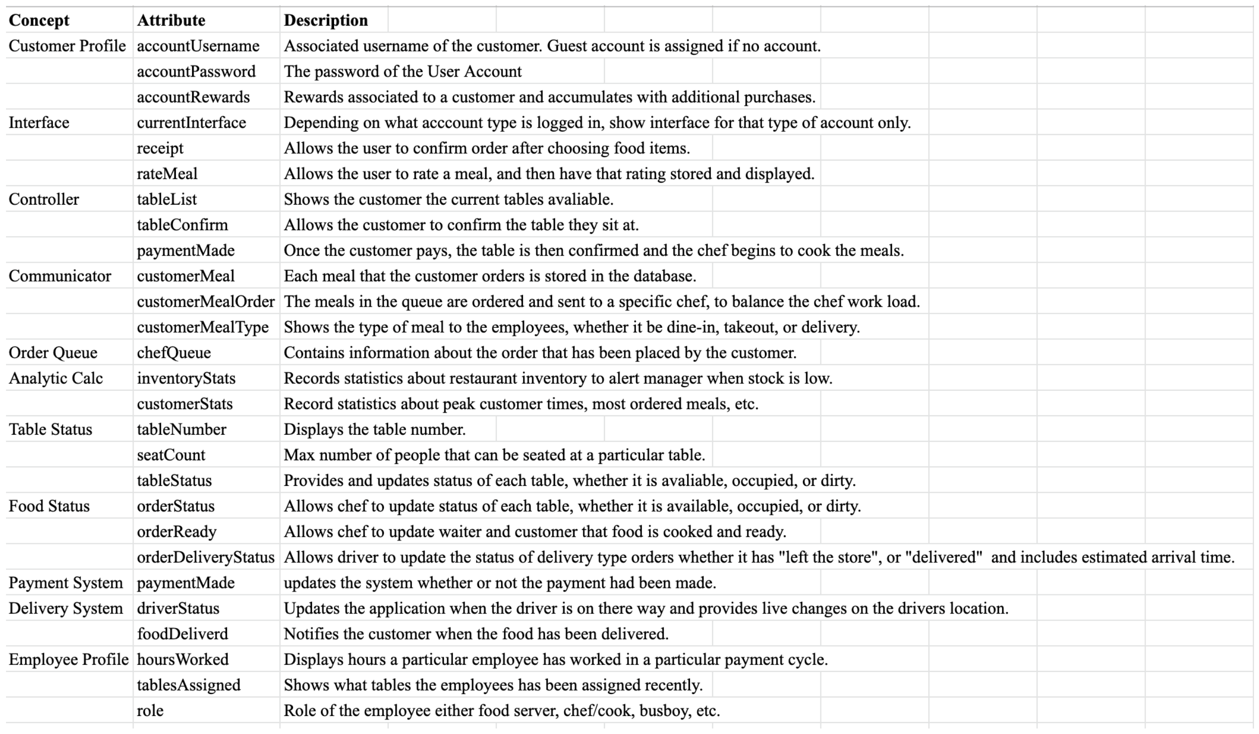
R2: Display a user interface for customers, chef, waiters, busboys, drivers, and managers	D	Interface
R3: Store customer login information and purchase history, and reward points	K	Customer Profile
R4: Store employee login information, employee type, and hours worked	D	Employee Profile
R5: Prompt customer 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

ii.Association definitions

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⇔ Database	Store payment record/receipt into the database	Payment records
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

iii.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.

1. ## iv.Traceability matrix — show how your use cases map to your domain concepts.

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

b. System Operation Contracts

Operation	UC-9 Login

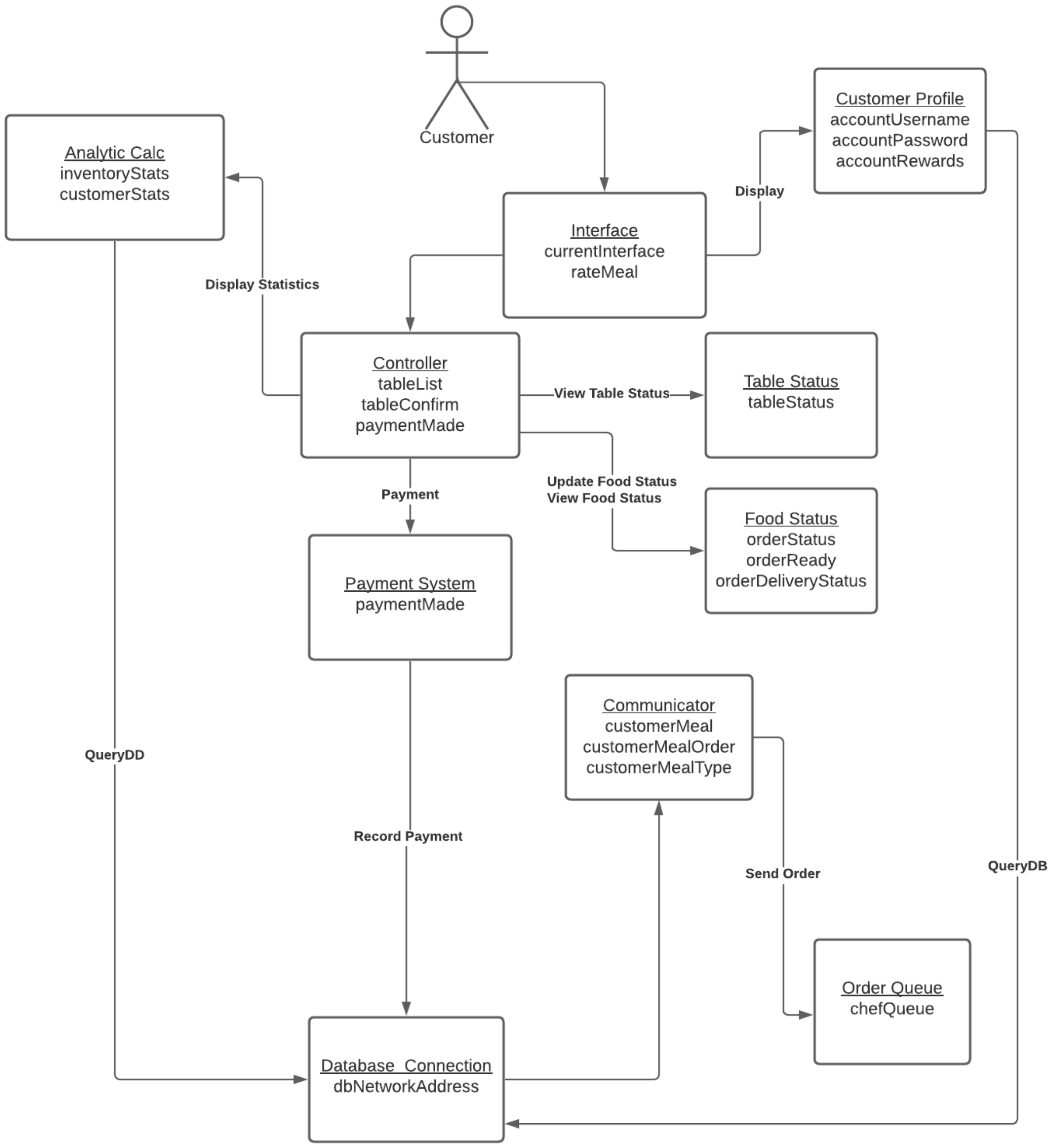
Preconditions	● A user such as a customer or an employee has already created an account. ● The user knows their username and password.
Postconditions	● The customer is now able to order food. ● An employee can access their interface features.

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.

c. Data Model and Persistent Data Storage

2. Data Model



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.

3. 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]
```

```
    }
```

d. Mathematical Model

An explanation of mathematical models is provided in Part 3 of the Report. You will find this in part 4.a.

List(add if you think of any):

Underlined Items = Done, Non-Underlined Items Need To Be Finished

- Reservation
- Delivery Juergen (jason edits)
- Payroll Juergen
- Payment + Rewards
- Analysis: Jason
- Scheduling Juergen/ Justin(wrote method)
- Ordering Juergen
- Employee Clock in and out (Pavan)
- Food Order Status (Pavan)
- Update Menu (Pavan)
- Display Daily Sales/Profits (Pavan)
- Food Inventory (Justin)
- Account Login (Justin)

```
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

        }
    }


}
```

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)
}
```

Payroll() {

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

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
```

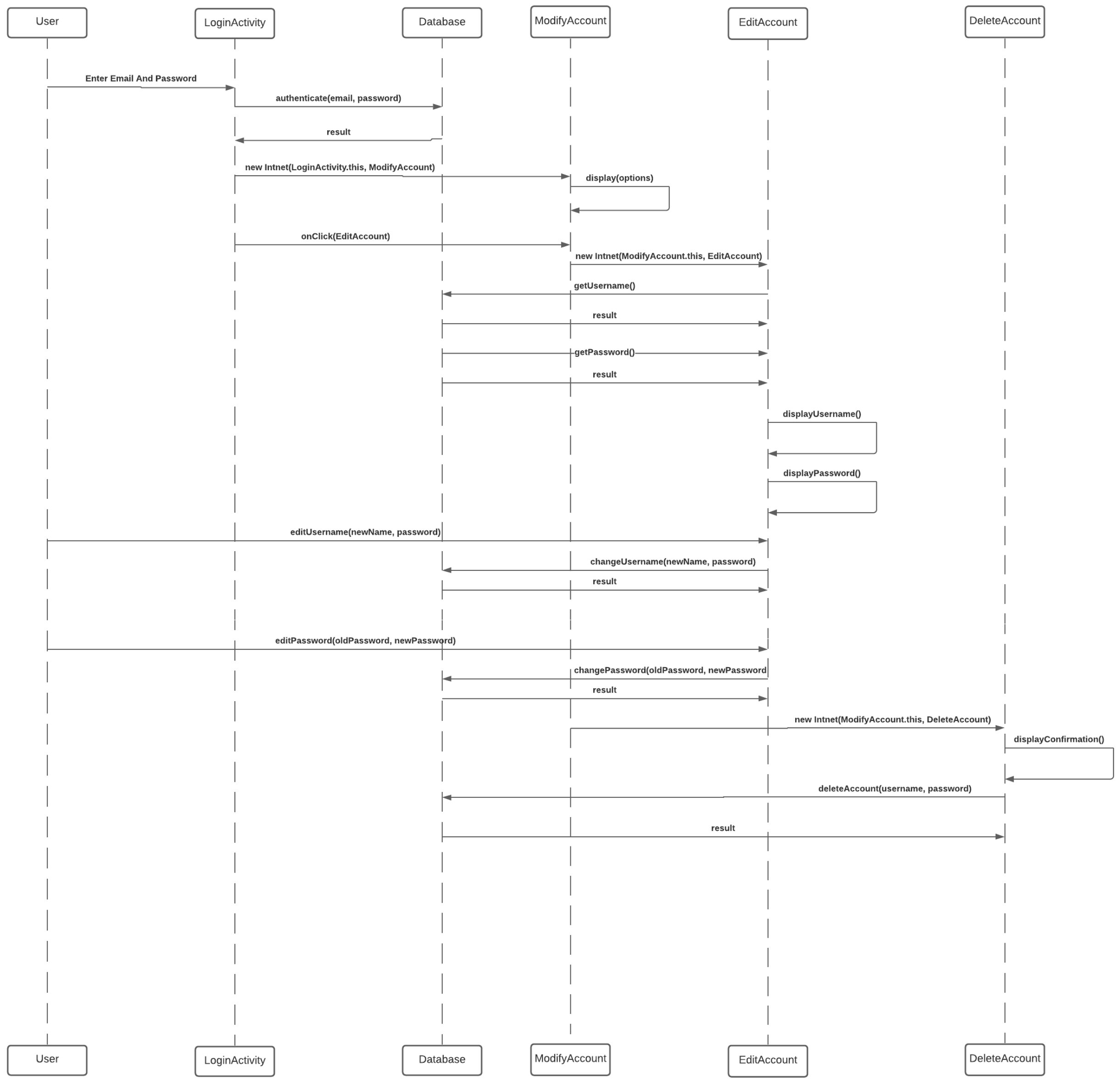
```
        //Deduct the amount of the price from customer’s account
        if (TotalPrice <= Customer.balance) {
                if (Customer.rewardUse == true) {
                        Customer.rewards = Customer.rewards - TotalPrice;
                }
                Customer.balance = Customer.balance - 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);
        }
}

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[][]);
}

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
        }
}
```

```
**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()
}

**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
}

**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
    }
}

**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
        }
}

displayDailySale/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;
        }
}


/*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;
}

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;
}
```

2.Interaction Diagrams

Edit/Delete Account UC

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



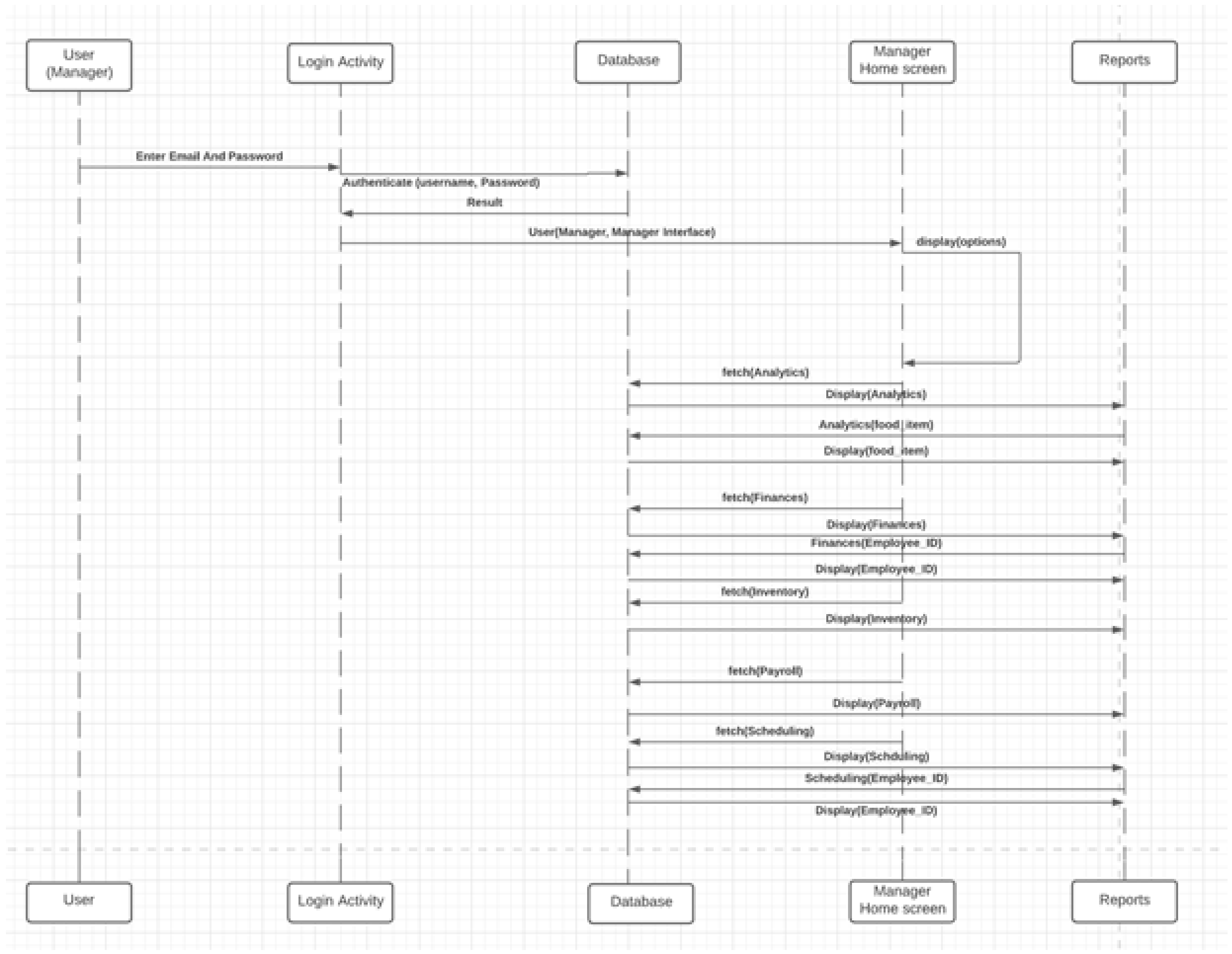
User
LoginActivity
Database
ModifyAccount
EditAccount
DeleteAccount
Enter Email And Password
authenticate(email, password)
result
new Intnet(LoginActivity.this, ModifyAccount)
display(options)
onClick(EditAccount)
new Intnet(ModifyAccount.this, EditAccount)
getUsername()
result
getPassword()
result
displayUsername()
displayPassword()
editUsername(newName, password)
changeUsername(newName, password)
result
editPassword(oldPassword, newPassword)
changePassword(oldPassword, newPassword
result
new Intnet(ModifyAccount.this, DeleteAccount)
displayConfirmation()
deleteAccount(username, password)
result
User
LoginActivity
Database
ModifyAccount
EditAccount
DeleteAccount

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.

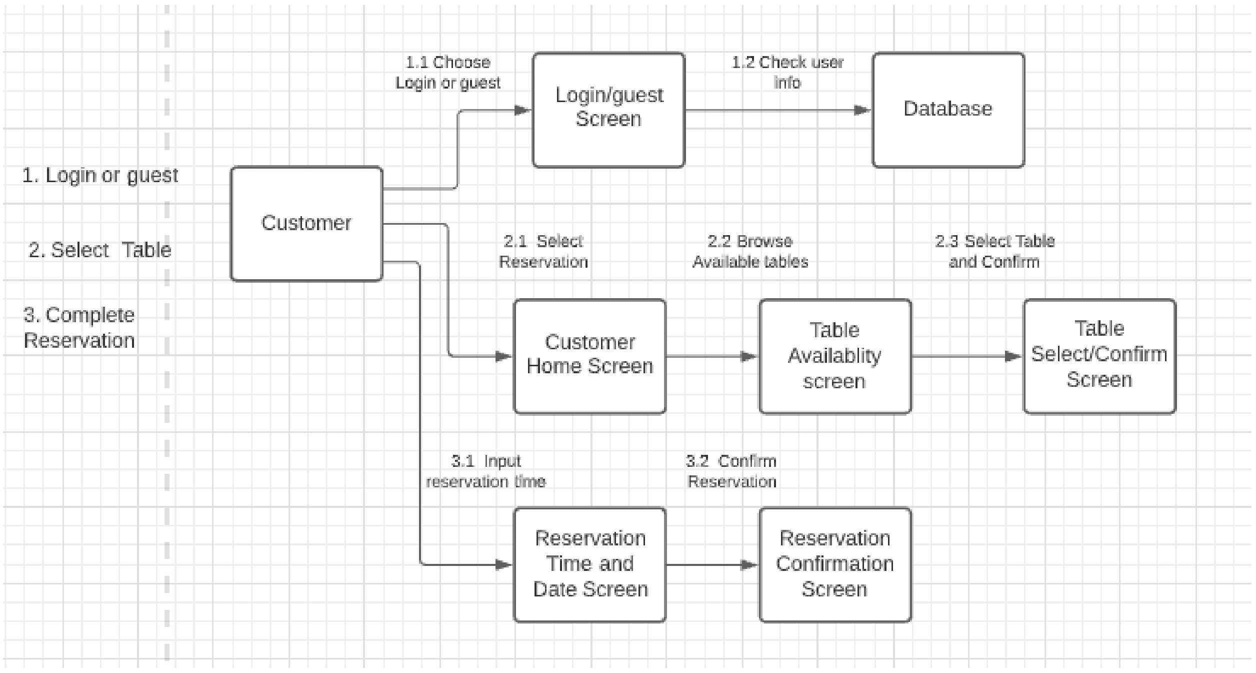
Reports UC

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.

I will do Reservation UC- Jason



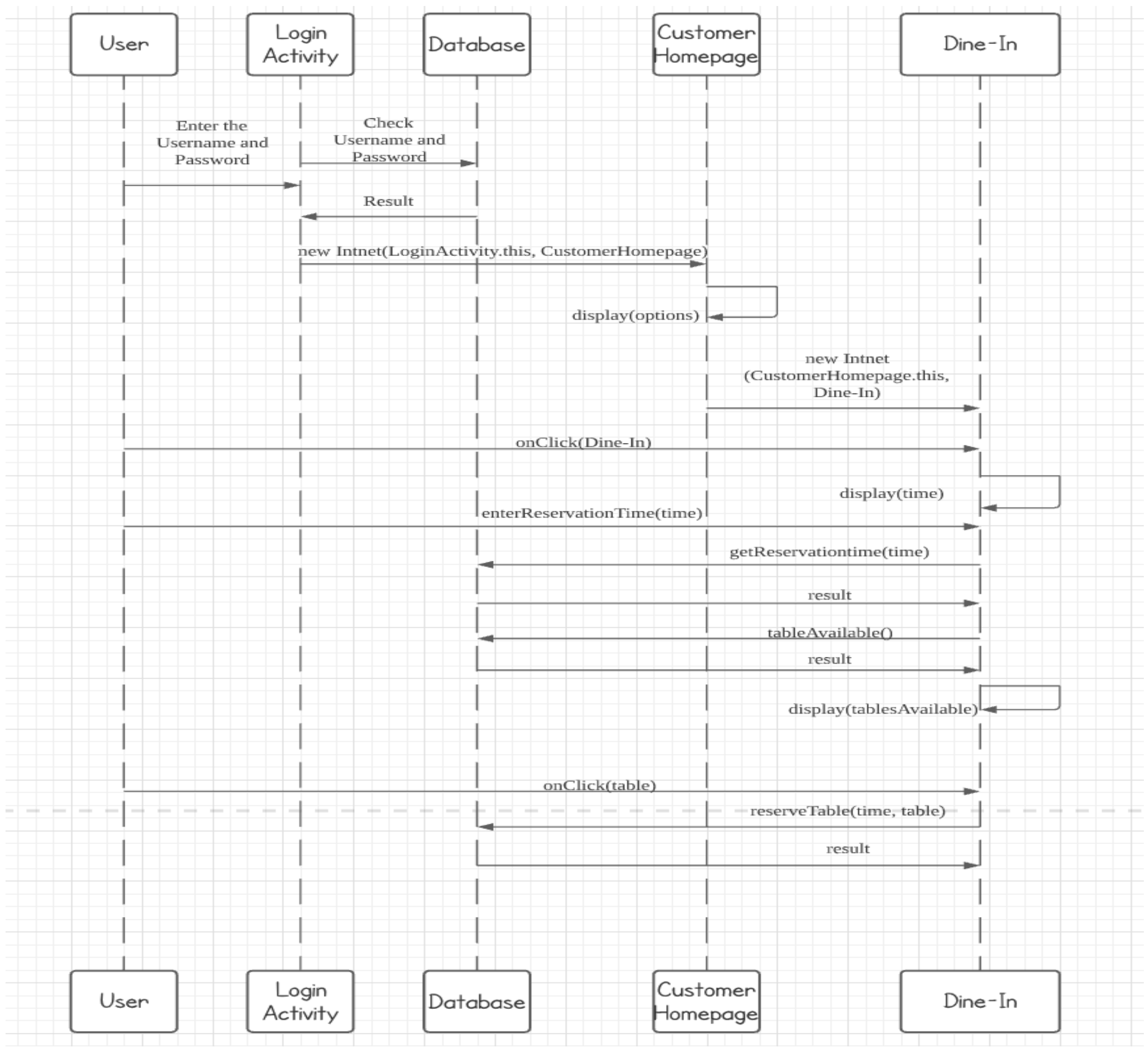
This communication diagram represents the interactions in the Reservation Use Case. As can be seen, there are three main parts to this UC. The user/customer can choose to login to their account or choose to stay as a guest user on the login screen. The interface will then (behind the scenes) access the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

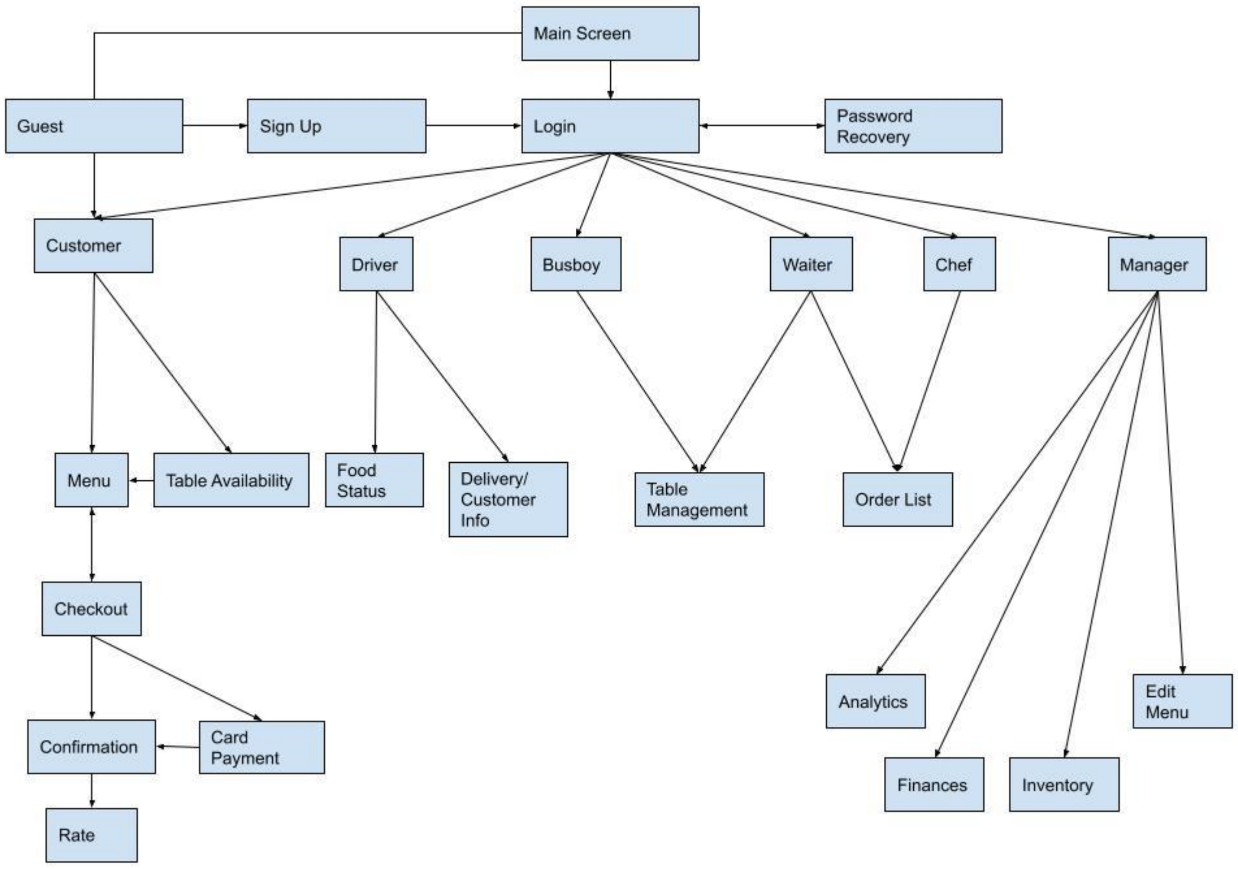
https://lucid.app/lucidchart/e69607b9-ca7d-43d4-bf97-8a77df827879/edit?page=0_0#



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.

3.Class Diagrams

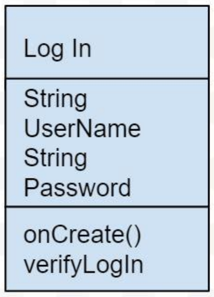
a.Class Diagram



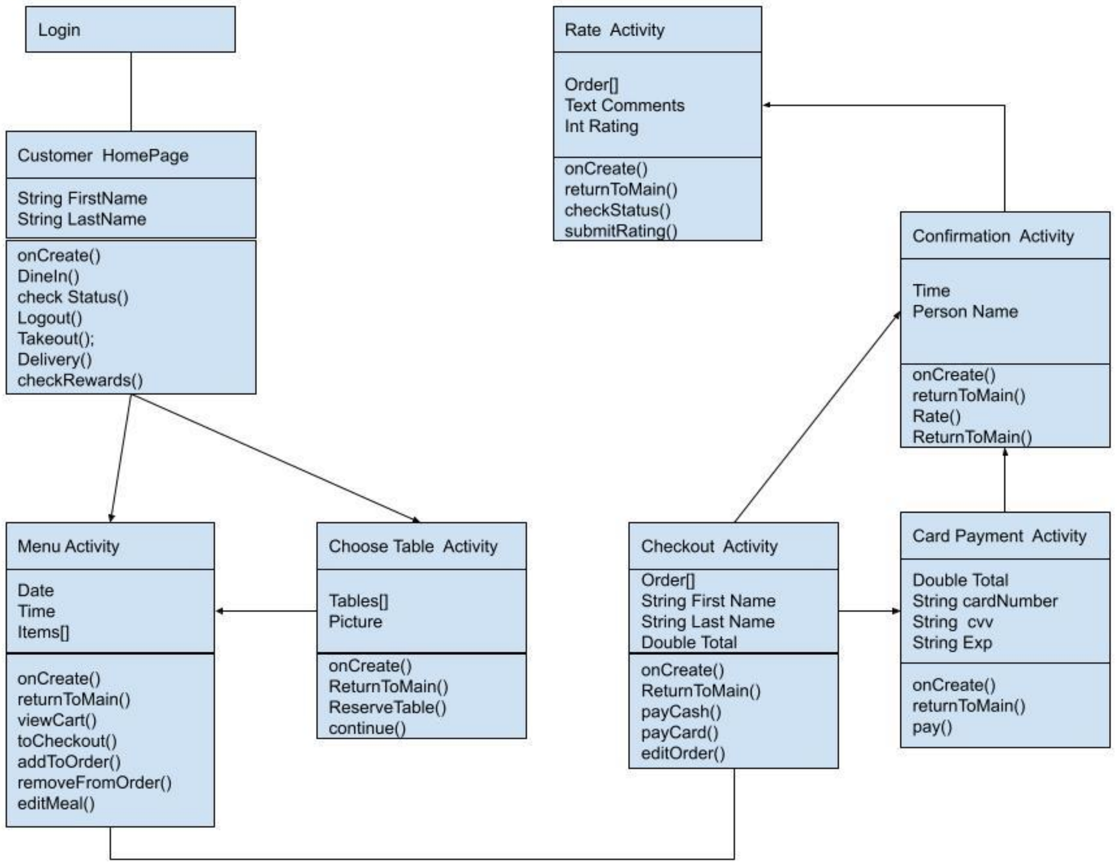
MainScreen - This is the main screen. All users will be prompted to enter through this screen.
Guest - These are for someone who does not want to login. They will not receive benefits associated with more defined roles (i.e. customer, busboy).
Sign Up - Allows for the creation of an account.
Login - user logs in and credentials are checked. If verified, the next screen is brought up.
Password Recovery - Password is recovered using normal procedures common to other password recovery features seen elsewhere.
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.
Chef - chef account. Chef is responsible for viewing orders and preparing them.
Manager - manager account. Manager has the privilege of overseeing all features appropriated to employee accounts. Has special privileges like adding or removing menu items.

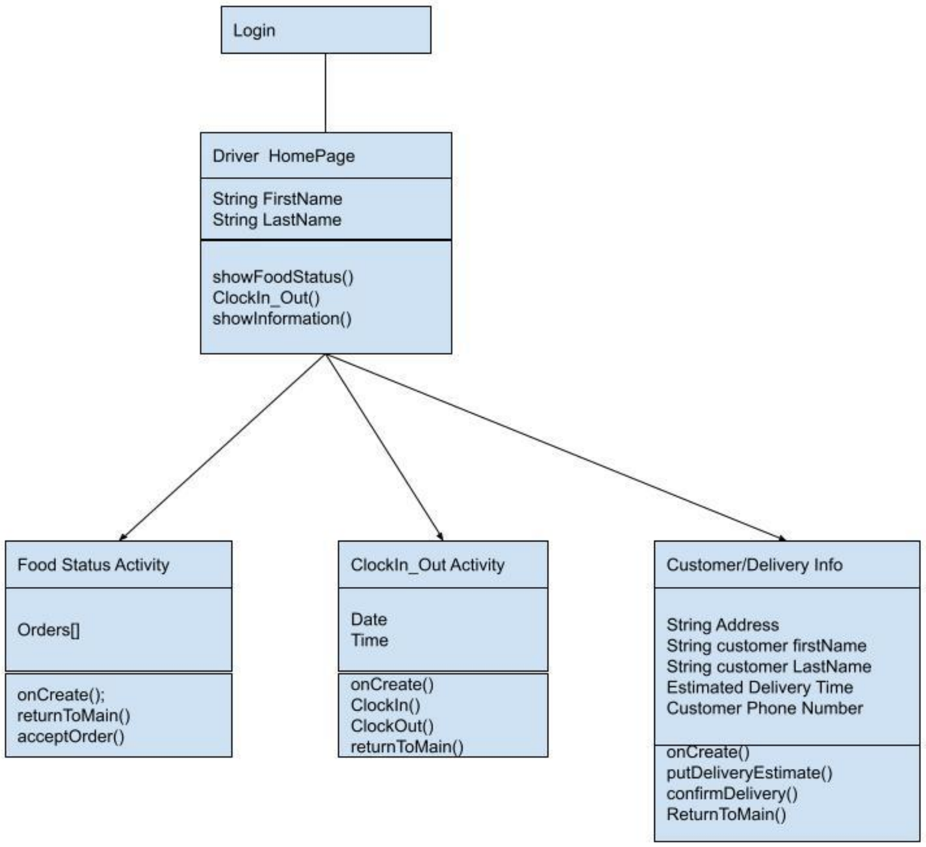
Menu - list of available food items.
Table Availability - list of available tables. For ease of customer seating.
Food Status - Indicates the current status of order.
Delivery/Customer Info - Information for the driver to deliver a customer’s order to the correct person.
Table Management - class available to busboys. Allows for the management of tables (typically cleaning).
Order List - List of orders. A log of orders to be kept and managed.
Checkout - Once a customer has finished, option to checkout. Transaction occurs.
Confirmation - customer can confirm their payment method before checkout.
Card Payment - Provides the option to pay by credit card.
Analytics - Analytics option for manager to view statistics about business.
Edit Menu - option to edit menu (manager) by adding or removing menu items.
Rate - displays price of items.
Finances - displays current finances(i.e. Cost, Sale, Profit).
Inventory - displays current inventory and stock of items.

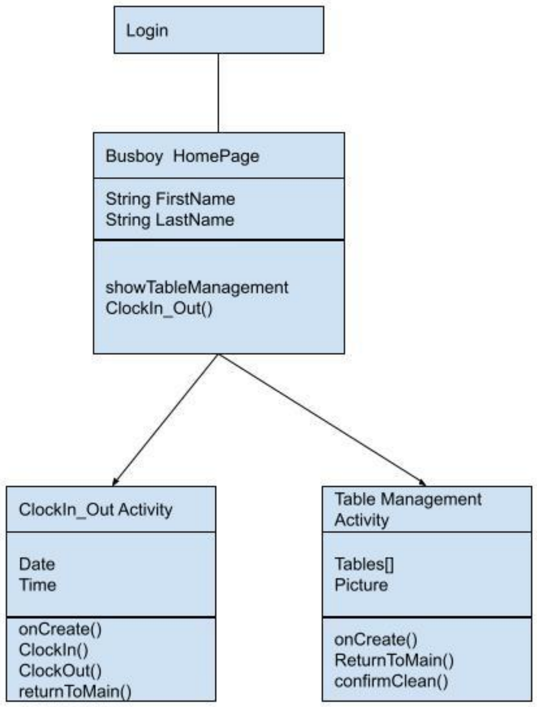
The above class diagram shows the different activities or user screens, which are classes themselves, in our Android Studio application. Depending on what button the user presses on their screen we can call a method within that activity to take the user to another screen.

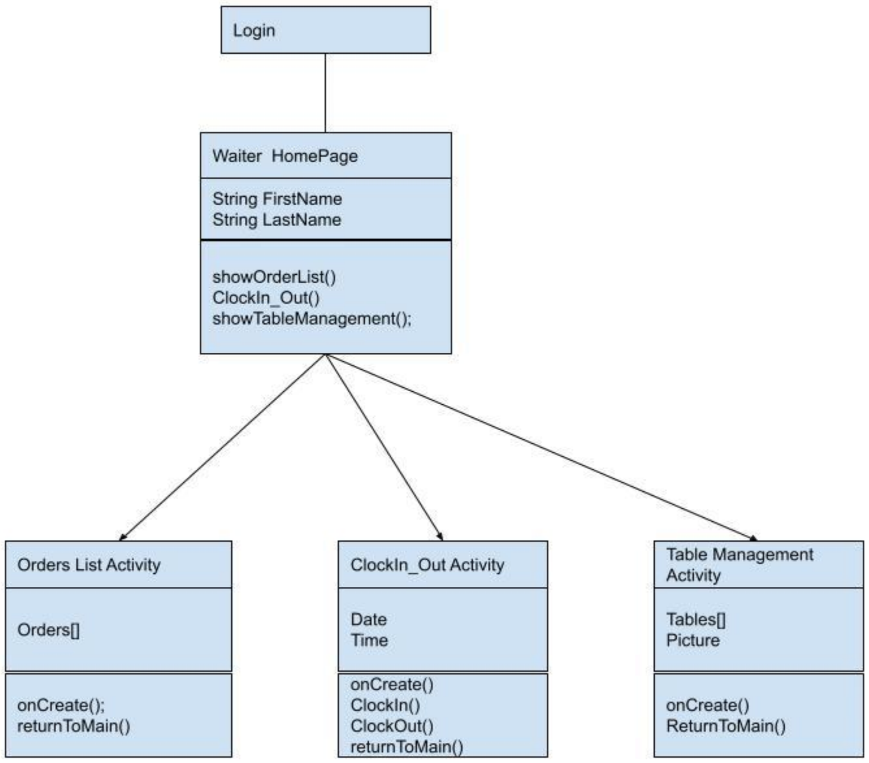


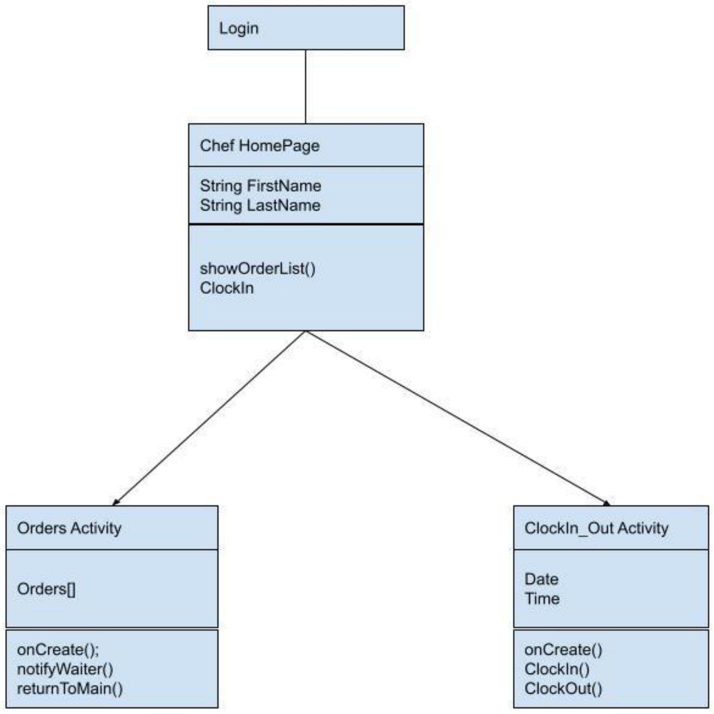
***I believe it is a good idea for the “Log In” class to contain the string UserName and Password. The order of operations starts with creating a “Log In” class before a “Customer” or any variety of Employee class. The idea is for the “Log In” class to acquire the necessary information and forward the information to a “Customer” or any variety of Employee class only after user credentials are verified. For this reason, it is important for the “Log In” class to have a string for both username and password, albeit these strings are only used temporarily.**

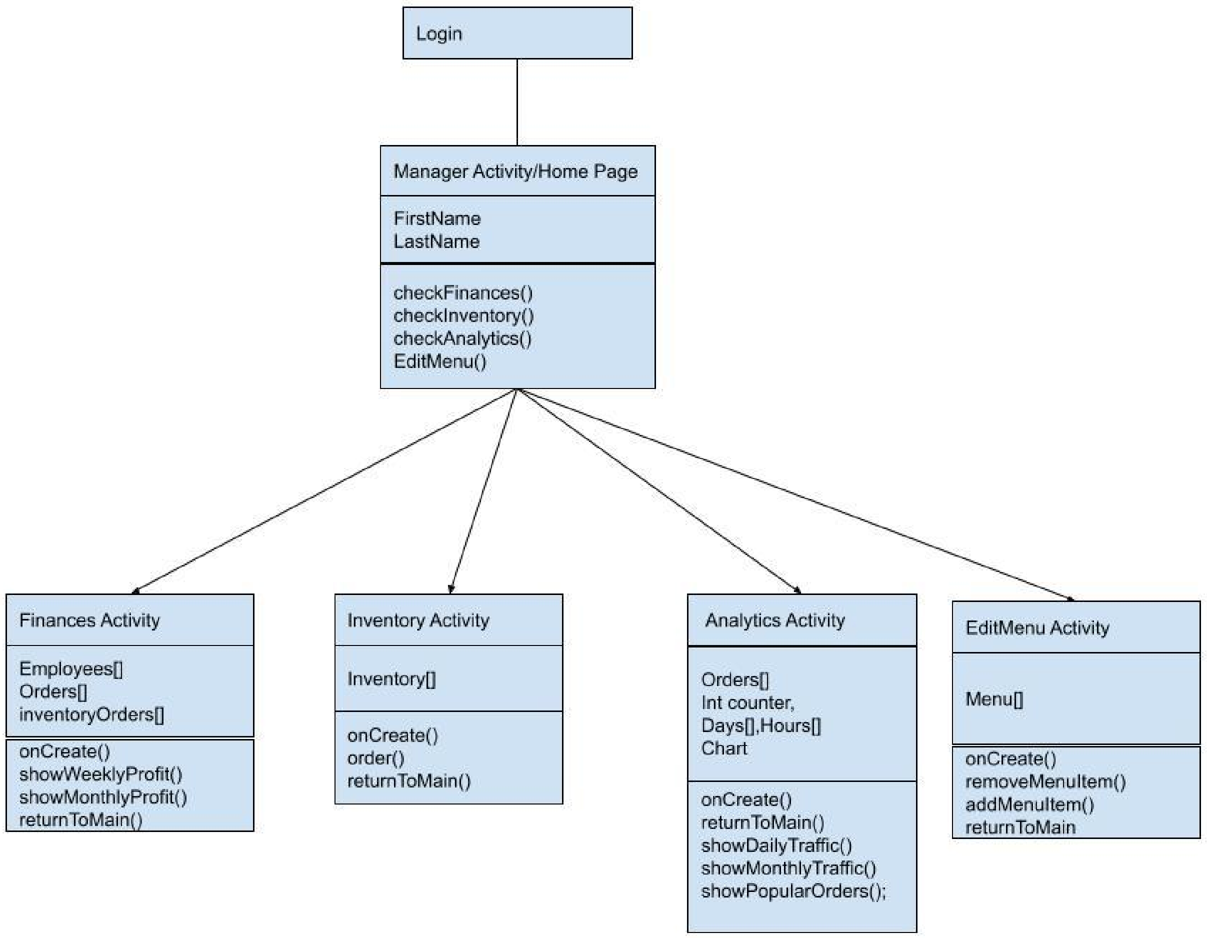

Login
Customer HomePage
String FirstName
String LastName
onCreate()
DineIn()
check Status()
Logout()
Takeout();
Delivery()
checkRewards()
Menu Activity
Date
Time
Items[]
onCreate()
returnToMain()
viewCart()
toCheckout()
addToOrder()
removeFromOrder()
editMeal()
Choose Table Activity
Tables[]
Picture
onCreate()
ReturnToMain()
ReserveTable()
continue()
Rate Activity
Order[]
Text Comments
Int Rating
onCreate()
returnToMain()
checkStatus()
submitRating()
Confirmation Activity
Time
Person Name
onCreate()
returnToMain()
Rate()
ReturnToMain()
Checkout Activity
Order[]
String First Name
String Last Name
Double Total
onCreate()
ReturnToMain()
payCash()
payCard()
editOrder()
Card Payment Activity
Double Total
String cardNumber
String cvv
String Exp
onCreate()
returnToMain()
pay()


Login
Driver HomePage
String FirstName
String LastName
showFoodStatus()
ClockIn_Out()
showInformation()
Food Status Activity
Orders[]
onCreate();
returnToMain()
acceptOrder()
ClockIn_Out Activity
Date
Time
onCreate()
ClockIn()
ClockOut()
returnToMain()
Customer/Delivery Info
String Address
String customer firstName
String customer LastName
Estimated Delivery Time
Customer Phone Number
onCreate()
putDeliveryEstimate()
confirmDelivery()
ReturnToMain()


Login
Busboy HomePage
String FirstName
String LastName
showTableManagement
ClockIn_Out()
ClockIn_Out Activity
Date
Time
onCreate()
ClockIn()
ClockOut()
returnToMain()
Table Management Activity
Tables[]
Picture
onCreate()
ReturnToMain()
confirmClean()


Login
Waiter HomePage
String FirstName
String LastName
showOrderList()
ClockIn_Out()
showTableManagement();
Orders List Activity
Orders[]
onCreate();
returnToMain()
ClockIn_Out Activity
Date
Time
onCreate()
ClockIn()
ClockOut()
returnToMain()
Table Management Activity
Tables[]
Picture
onCreate()
ReturnToMain()


Login
Chef HomePage
String FirstName
String LastName
showOrderList()
ClockIn
Orders Activity
Orders[]
onCreate();
notifyWaiter()
returnToMain()
ClockIn_Out Activity
Date
Time
onCreate()
ClockIn()
ClockOut()



b.Data Types and Operation Signatures

```
Class Login{
        String Username
        String Password
        onCreate()
        verifyLogin()
}
```

Attributes

- String Username: a user field belonging to a specific customer or employee
- String Password: another user field belonging to a specific customer or employee

Methods

- void verifyLogin(): a method that is called when the user presses the SignIn button on screen. It will check the database if that combination of user and password exists so that the appropriate screen will be shown.

```
Class customerProfile{
Fields:
String Name;
String userName
String Password;
String email;
customersOrders: [Order]

Methods:
DineIn();
Takeout();
Delivery();
checkStatus();
checkRewards();
Logout()
}
```

Attributes

- String Name : The customer's Name
- String userName: The customers username for logging in
- String Password : A customers password
- String email : a customer email address
- customersOrders: [Order] : List of orders that the customer has ordered.

Methods

- DineIn() : Takes a user to a screen to select their table when they press the Dine-In button on screen.
- Takeout(): Takes the user to the menu screen so they can begin their order after they press the Takeout button on screen.
- Delivery(): Takes the user to the menu screen so they can begin their order after they press the delivery button on screen.
- checkStatus(): Takes the user to a different screen showing the status of their current orders after they press the status button on the screen.
- checkRewards(): Takes users to a screen where they can see their rewards and specials after they press the check rewards button on the screen.
- Logout(): Logs the user out and takes them back to the login screen.

```
Menu :: {
Items [item]
Date
Time
viewCart()
toCheckout();
addToOrder()
removeFromOrder()
```

editMeal()
}
Attributes
- String Date : The date an order is being made.
- String Time: The time an order is being made.

Methods
- viewCart : After pressing the view cart button on screen, it will show the user a popup or another screen of what is in their order so far.
- toCheckout: After the user presses the checkout button, it will take the user to a different screen where they will be shown their order and choose their method of payment.
- addToOrder(): After the user presses the + button next to something on the menu it will add that item to their cart.
- removeFromOrder():After the user presses the - button on their cart it will remove that item from their cart.
- editMeal():If the user chooses to edit, then they can filter out items they don't want in one of their foods.

Class Table Availability
Attributes
- Table[] : List of tables and their status.

Methods
- reserveTable() : Reserves a table for a customer if available.

Class Checkout:
Attributes
- String FirstName : The customer's first name
- String LastName: The customer's last name.
- Double Total : The cost of a customer's order.

Methods
- editOrder():Allows the customer to go back and edit their current order.
- payCash():Takes the customer to the confirmation screen.
- payCredit():Takes the customer to the card payment screen.

Class CardPayment:
Attributes
- String FirstName : The customer's first name
- String LastName: The customer's last name.
- Double Total: Cost of customers order.
- String CardNumber:Credit card info user field
- String CVV: Credit card info user field
- String EXP:Credit card info user field.

Methods
- pay():A method that will take the user to the confirmation screen.

Class Confirmation:

Attributes
- String FirstName : Customer first name
- String LastName: Customer last name
- Time: An estimate of when the order will be ready.

Methods
- Rate():Takes the customer to a different screen where they can submit their rating to the system.
- returnToMain():Takes the customer to their home screen

Class Rate:
Attributes
- Order[]:Order list for customer to rate if they wanted to.
- Text Comments: Customer comment field.
- Int Rating:Customer Rating.

Methods
- submitRating:Customer submit rating to the system after clicking the submit button.
- checkStatus():If a customer doesn't want to rate then they can check the status of their order after clicking the button on screen.
- returnToMain(): If the customer doesn't want to wait they can return to their home screen.

```
Class Chef {
String FirstName
String LastName
Methods:
clockIn_Out()
showOrderList();
}
```
Attributes
- String FirstName : The chef's first name
- String LastName: The chef's last name.

Methods
- clockIn_Out(): Takes an employee to the clock in and clock out screen when they press the button on the screen.
- showOrderList(): Take the chef to a screen where it will show them a queue of current orders they have to make.

```
Class Orders(Activity){
Order[]
onCreate()
notifyWaiter()
returnToMain()
}
```
Attributes
- Order[] : A queue of orders that will be displayed on the chefs screen.

Methods

- notifyWaiter(): When the chef presses this button it will alert the waiter an order is ready to be served.

Class Busboy {
First Name
Last Name

Methods:
clockIn_Out()
TableManagement();
}

Attributes

- String FirstName : The busboy’s first name
- String LastName: The busboy’s last name.

Methods

- clockIn_Out(): Takes the employee to the clock in and clock out screen when they press the button on the screen.
- showOrderList(): Takes the busboy to the table management screen

Class TableMangemnt(Busboy){
Tables[];
confirmClean();
}

Attributes

- Tables[] : A list of tables and their current status.

Methods

- confirmClean() : A method that will update the systems status on a table.

Class Waiter {
String FirstName
String LastName
Methods:
clockIn();
ClockOut();
checkOrders();
TableManagement();
}

Attributes

- String FirstName : The waiter’s first name
- String LastName: The waiter’s last name.

Methods

- clockIn_Out(): Takes an employee to the clock in and clock out screen when they press the button on the screen.

- showOrderList(): Take the waiter to a screen where it will show them a list of orders ready to be served.
- TableManagement():Takes the waiter to the table management screen.

```
Class Manager{
String First Name
String Last Name
Methods:
checkFinances();
checkAnalytics();
checkInventory();
EditMenu();
}
```

Attributes

- String FirstName : The manager’s first name
- String LastName: The manager's last name.

Methods

- checkFinances(): Takes the manager to the finances screen.
- checkAnalytics(): Takes the manager to the analytics screen.
- checkInventory():Takes the manager to the inventory screen.
- EditMenu():Takes the manager to the edit menu screen where they can remove or add items to the menu.

```
Class Driver{
String First Name
String Last Name
Methods:
showFoodStatus()
showInformation();
}
```

Attributes

- String FirstName : The driver’s first name
- String LastName: The driver’s last name.

Methods

- showFoodStatus():Takes the driver to a screen where they can see a list of orders ready to be picked up from the kitchen.
- showInformation(): Take the driver to a screen where they 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.

```
classFoodStatus{
Orders[];
acceptOrder();
}
```

Attributes

- Orders[]:A list of orders the driver can see to pick up from the kitchen.

Methods

- acceptOrder():When the driver clicks on the screen it will make them responsible for delivering the food. Their information screen is updated.

```
Class DeliveryInfo{
Orders[]
String Address;
String firstName;
String LastName;
Time
Phone Number
putDeiliveryEstimate()
confirmDelivery()
}
```

Attributes

- String Address:Customer delivery info for driver.
- String firstName:Customer delivery info for driver.
- String LastName:Customer delivery info for driver
- Time-Time estimate for delivery
- Phone Number:Customer delivery info for driver.
- Orders[] : A list of orders that the driver has accepted for delivery.

Methods

- putDeliveryEstimate():Lets driver input the delivery estimate to the customer
- confirmDelivery():Notifies the system an order has been successfully delivered and the drivers list of orders is updated.

```
Class Order :: {
Fields:
customerProfile Customer;
items  [Item];
Date date;
Double cost;

Methods:
cancelOrder();
```

```
          editOrder();
          }


          Class Item {
          Fields:
          String name
          Int quantity
          Double price
          Double rating
          String type(Meal,Individual Order,Drink,etc)


          Methods
          addToOrder();
          removeFromOrder();
          filterItem();
          }
Class Table{
          Fields:
          Int seats;
          Boolean taken;
          Double receipt;

          Methods:
          reserveTable();
          unreserveTable();
          pay();
          }
```

c.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
ManagerEditEmployee		x									x
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 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
- <u>Controller</u>
    - 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
- <u>Communicator</u>
    - Manager - communicator allows manager to contact employees
    - Driver - communicator allows driver to contact customer
    - Waiter - communicator allows waiter to contact chef
- <u>Order Queue</u>
    - 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.
- <u>Analytic Calc</u>
    - 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
- <u>Table Status</u>
    - 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.

--- PART 3--

4. Algorithms and Data Structures

a.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.

- <u>Food Order Status (Pavan)</u>
    - 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.
- <u>Update Menu (Pavan)</u>
    - 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.
- <u>Display Daily Sales/Profits (Pavan)</u>
    - 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.

b.Data Structures

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.

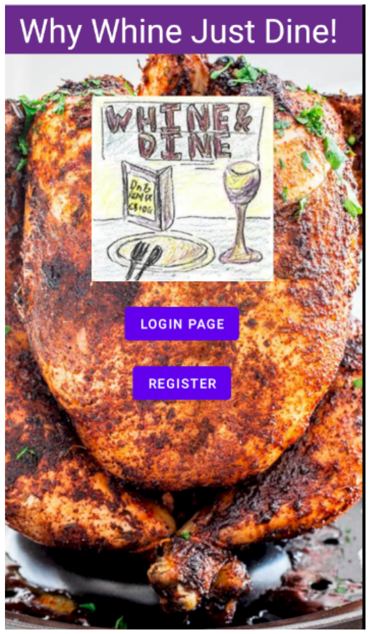
The primary takeaway is that arrays and databases have a consistent read and write time and are both flexible enough to handle adding or removing items without much difficulty.

c.Concurrency

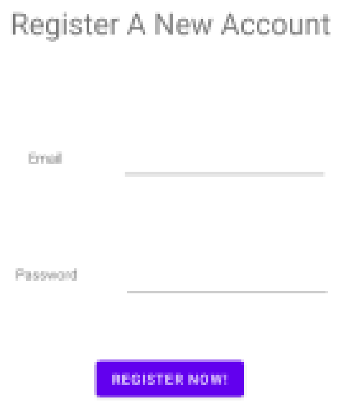
Due to the fact mutual exclusion happens when data is modified in some way, sync automatically occurs and our database requires the use of multiple threads.

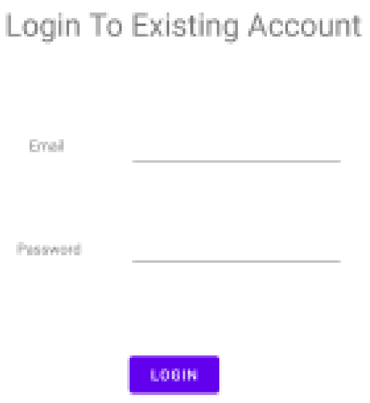
5.User Interface Design and Implementation

Register/Login



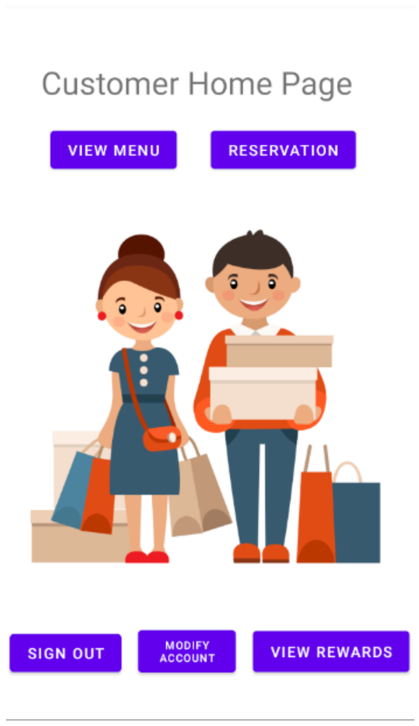
As can be seen, this is the home screen of our application, “Why Whine Just Dine”. When the user opens the application, this will be the first thing that they will see. On the screen, the user can see two buttons, one that brings them to the login page, and another to the register page.

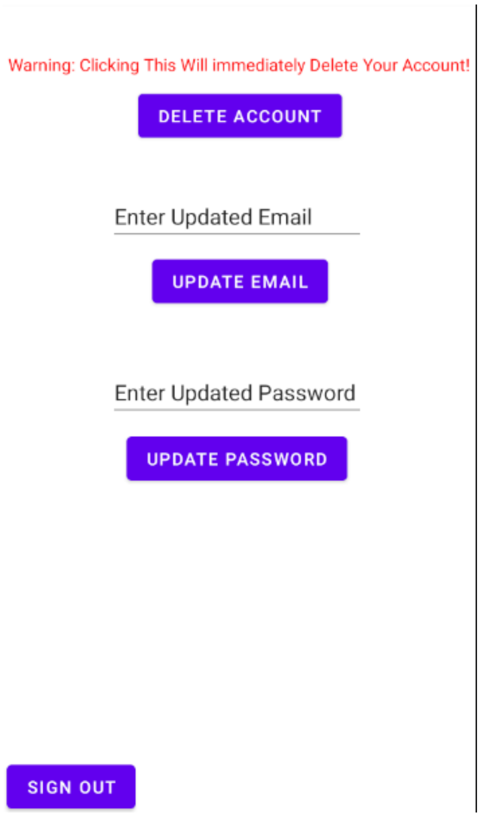




As seen above, there is a registration page and a login page. If one doesn't have an account yet, then the user can register a new account with an email and a password. There is a specified minimum length for both the email and password, so if the inputs are invalid it will prompt the user to input again.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

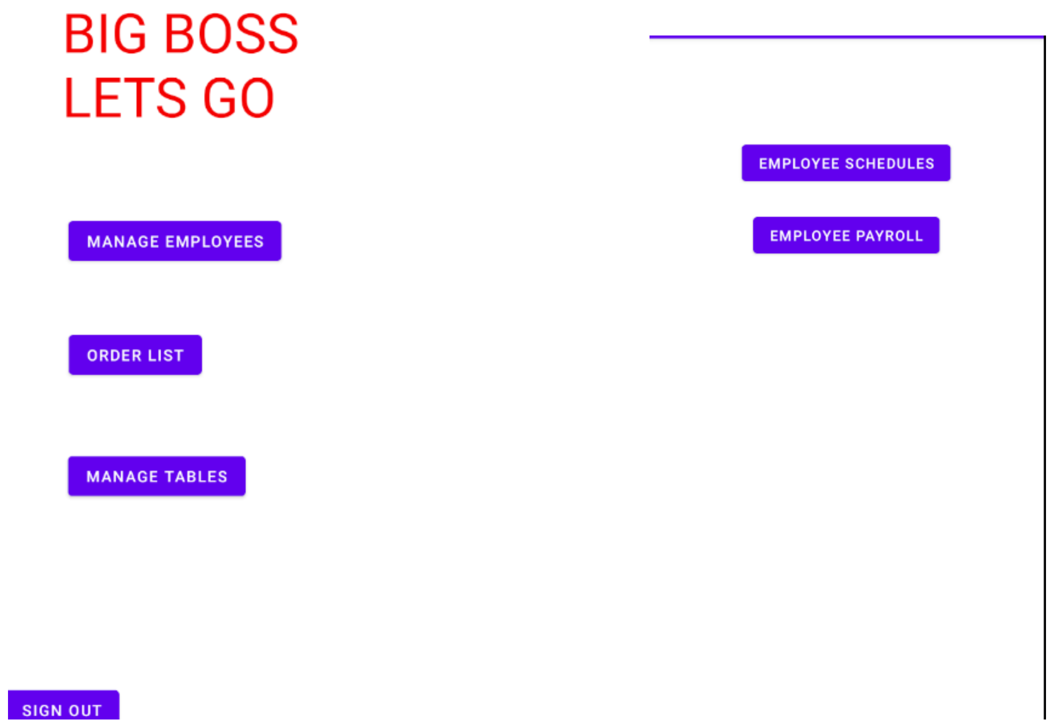
Customer Interface:



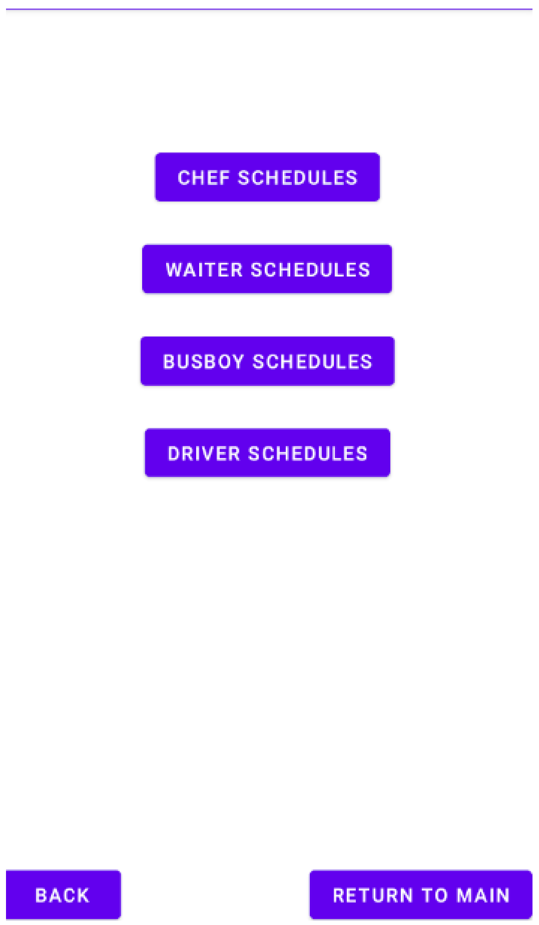


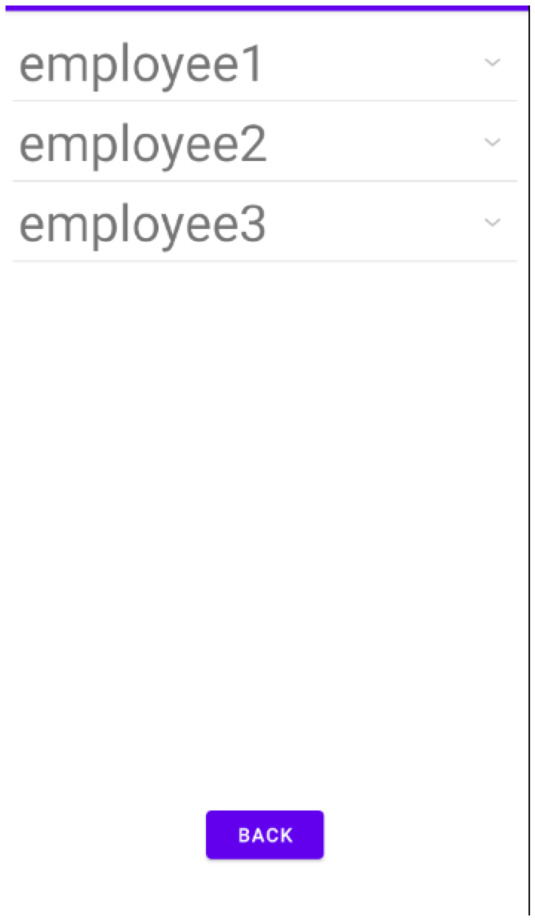
Above you can see the customer home page. As can be seen, there are 5 buttons. The first button “View Menu” will bring you to a menu page where the customer can then order food. The “Reservation” button will bring the customer to a page that allows them to reserve a table in advance. The modify account 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.

Manager Interface:

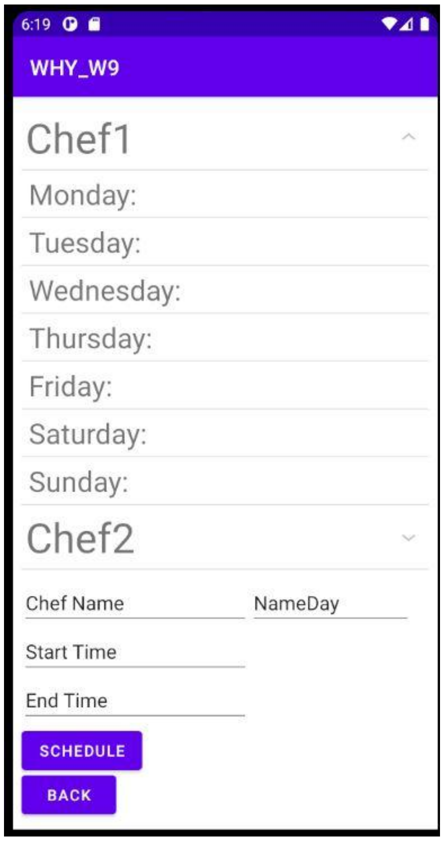


As can be seen on the screen above, the manager home page has a few buttons. The first button “Manage Employee” will bring the user to the employee management page where the user can then click on buttons to go to employee schedules and payroll. The second button on the home page is an order list, that 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.





If the manager clicks on the manage employees button, they will be brought to this screen, where they can filter the employees by their job in the restaurant. Thus they can check the schedules of the respective employees based off the occupation, and the screen they will be brought to will allow them to see information of the employee if they click the drop down arrow. As can bee seen in the chef schedules below, there is a full day of the week displayed, as well as the start and end times for that day with the chef's name.


6:19

WHY_W9

Chef1

Monday:

Tuesday:

Wednesday:

Thursday:

Friday:

Saturday:

Sunday:

Chef2

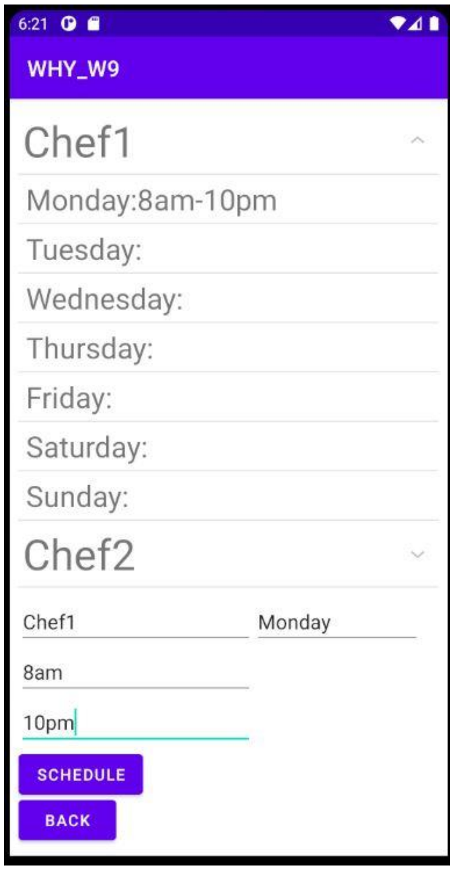
Chef Name NameDay

Start Time

End Time

SCHEDULE

BACK


6:21

WHY_W9

Chef1

Monday:8am-10pm

Tuesday:

Wednesday:

Thursday:

Friday:

Saturday:

Sunday:

Chef2

Chef1 Monday

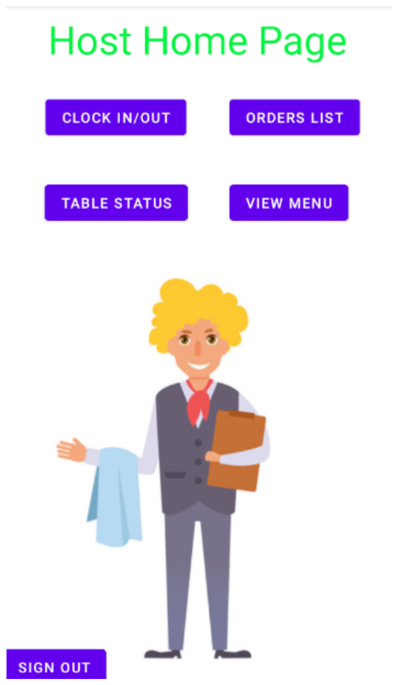
8am

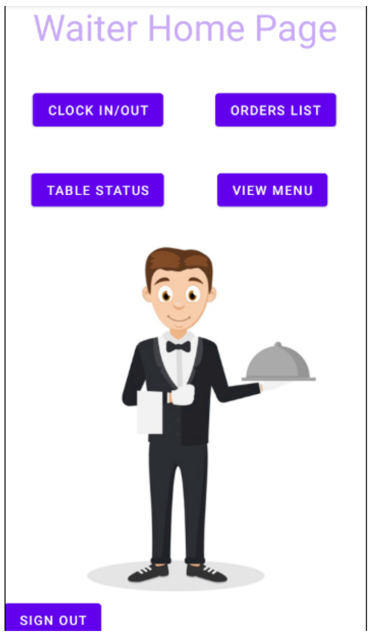
10pm

SCHEDULE

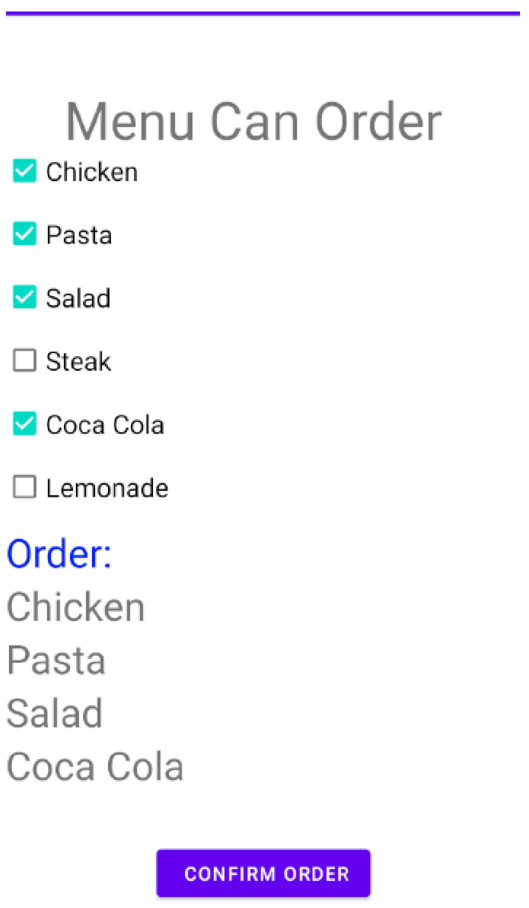
BACK

Employee Interface:`





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 tolog 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.

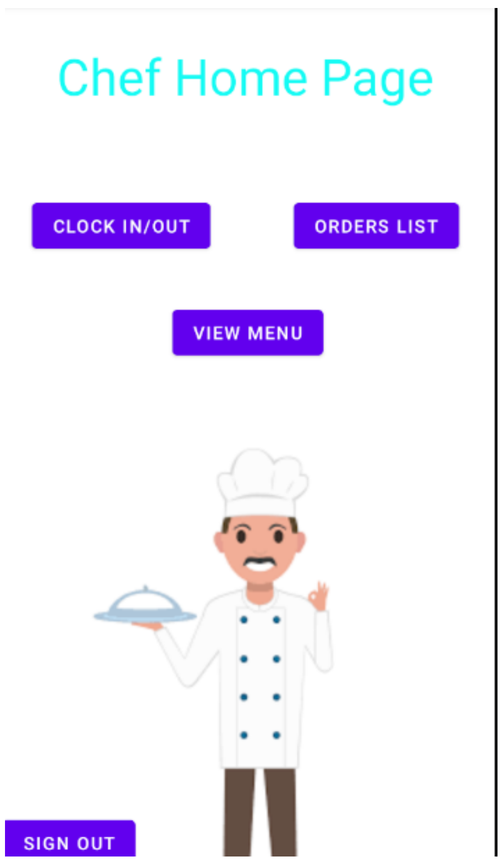


Menu Can Order

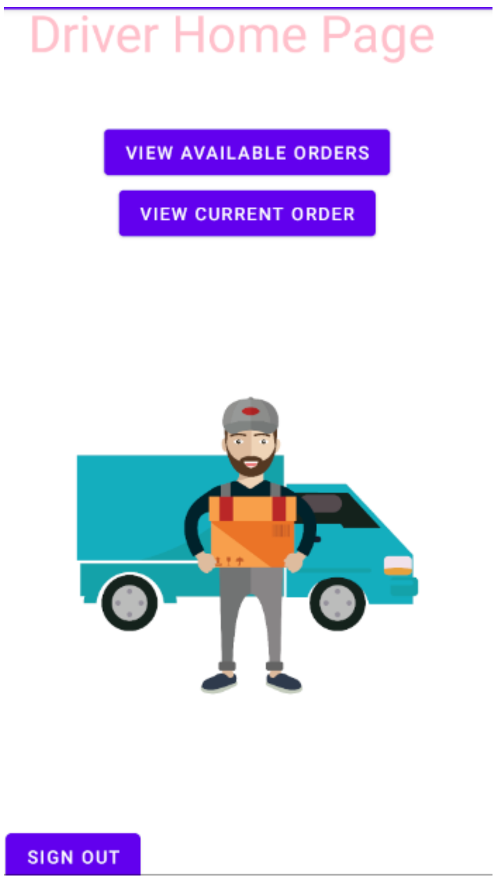
- [✓] Chicken
- [✓] Pasta
- [✓] Salad
- [ ] Steak
- [✓] Coca Cola
- [ ] Lemonade

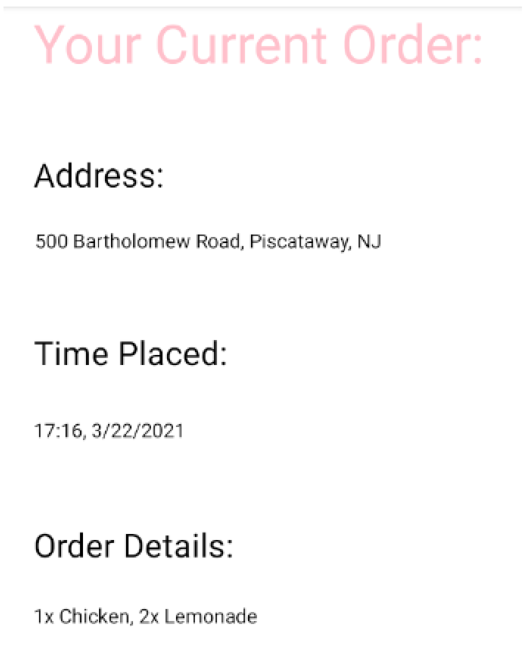
Order:

Chicken
Pasta
Salad
Coca Cola



The screen above is a simple menu that one can order from. As can be seen the user can check off the items they want, and the order will be displayed. Further implementation of this page with a database will be done in the future.




Your Current Order:

Address:

500 Bartholomew Road, Piscataway, NJ

Time Placed:

17:16, 3/22/2021

Order Details:

1x Chicken, 2x Lemonade

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.

6.Design of Tests

a.Test Cases

<u>The following are the test cases to be used for unit testing</u>:

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. Step 2: The user (an employee) enters a correct combination of their email and password into the login screen and selects login. Step 3: The user (a manager) enters a correct combination of their email and password into the login screen and selects login. Step 4: The user enters an incorrect 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. The app accepts the combination and allows the user to login. The app should bring the user to the employee main page. The app accepts the combination and allows the user to login. The app should bring the user to the manager main page. The app should not accept the combination and allows the user to attempt to login again and notifying an error message.	The app brings the user to the correct screen. The app brings the user to the correct screen. The app brings the user to the correct screen.

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.	(as of now) Clicking dine-in brings you to the payment screen and is located after selecting your items.

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. Step 2: The user will add multiple items from the menu to their list. Step 3: The user will delete a few items from the list.	The Takeout menu will appear. Items added will be added (with price and quantity) to the list. Items will be removed (with price and quantity) from the list.	The menu will appear and then can choose takeout as an option for eating. Follows Expected Result 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 Step 2: The user should select a Table from the Available tables list. Step 3: The user should then input the date the table is to be reserved for in the appropriate input text box. Step 4: The user should then input the time the table is to be reserved for in the appropriate input text box.	A screen that shows available tables as well as location within the restaurant pops up. A new screen pops up that has times where the table is unable to be reserved shown, as well as two input boxes. If the date is an invalid input, then produce an error message and prompt another input. If the time is an invalid input, then produce an error message and prompt another input. Else, proceed to the confirmation	Follows Expected Result Have not gotten this far Have not gotten this far Have not gotten this far

Step 5: The user should then confirm reservation, when a confirmation screen pops up.	screen. Correctly update the database for table availability with according time and date.	Have not gotten this far

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 the menu option to view the menu	A menu interface will pop up, allowing the user to view food items on the menu, as well as additional info like price/calories	Follows expected result 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.	Have not gotten this far
Step 2: Although it will bring up	The user will be able to see all	Have not gotten this far

the current table status screen, users can click on the “specific” button.	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.	
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 gotten this far

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.	The user should see a screen where they have 3 dining options.	User 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.	Have not gotten this far
Step 3: A chef account logs in and checks their order list.	The chef should be able to see the recently placed order.	Have not gotten this far
Step 4: 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.	Have not gotten this far

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.	Have not gotten this far
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.	Have not gotten this far

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 their account	The manager should see a list of options such as check analytics.	Manager can login to accounts and check certain options.

Step 2: Manager clicks check analytics.	The manager should see the number of times an item was sold in a chart.	Have not gotten this far

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. Step 2: The user will enter a used or incorrectly formatted email with a password that matches the requirements. Step 3; The user will enter an unused correctly formatted email with a password that does not meet the requirements.	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. The system will notify the user that the email is incorrectly formatted or already used and allow them to retype it. The system will notify the user that the password does not meet the requirements and allows them to retype it.	Follows expected result Follows expected result 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

b.Test Coverage:

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, and unlocked. When the user enters the login page, the user is in the locked mode. 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.

c.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.

7.Project Management

b.Project Coordination and Progress Report

The following use cases have been implemented:
Use Case 6 Employee Schedule
Use Case 7 Payroll
Use Case 9 Login
Use Case 10 Logout
Use Case 11 Register

c.Plan of Work

Coding milestones plan
February- early March : Implement the login and clock in/out interfaces. Each of the subgroups should implement at least one important feature and then test and debug as needed.

Customer: Implement dine-in and/or takeout along with payment.

Manager: Implement analytics. Start with finances or inventory features.

Employee:
Chef:Implement orders feature
Waiter:Implement orders list feature
Busboy and/or waiter: Implement table management feature

d. Breakdown of Responsibilities

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

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

March 23(date of first demo)

March 23 - Early/Mid April: Implement the other important features and improve upon previous features as needed. Hopefully, by this point some of the following have already been started.
Customer:Implement delivery and rewards system.
Manager: Finances and inventory related features.
Employee:Implement the driver interface

Week of April 20 (date of second demo):

April 20-May7th: Each subgroup should make any last changes and/or implement any remaining features.More testing, debugging.

8.References

https://www.ece.rutgers.edu/~marsic/Teaching/SE1/report2.html
https://www.ece.rutgers.edu/~marsic/Teaching/SE1/demo1.html

Duplicates From Report 1:
http://eceweb1.rutgers.edu/~marsic/books/SE/projects/Restaurant/2018f-g4-report3.pdf

https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2019-g13-report3.pdf