Why W8

Restaurant Automation

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

Group 4

Report 2 - Full Report

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

Individual Contributions Breakdown

	Project Category		Team Member Name					
			Stephan	Michael	Jimmy	Nikhil	Kyungsuk	Yi
Responsibility Levels	Sec 1. Interaction Diagrams	UML Diagrams *10 Points*	66%	34%				
		Prose Description of Diagrams *10 Points*	50%				50%	
		Alt. Solution Descriptions *10 Points*	50%					50%
	Sec 2. Class Diagram and Interface Specification	Class Diagram & Description *5 points*						100%
		Signatures & Traceability Matrix *5 points*		50%				50%
	Sec. 3 System Architecture & Design	Styles *5 points*			40%		60%	
		Package Diagram *2 points*				100%		
		Map Hardware *2 points*	50%				50%	
		Database *3 points*			20%	80%		
		Other *3 points*	25%	25%	25%		25%	
	Sec. 4 Alg's & Data Structures *4 points*				70%	30%		
	Sec. 5 User Interface	Appearance *6 points*		40%		60%		
		Prose Description *5 points*			100%			

	Sec. 6 Testing Design ***12 points***			50%		50%		
	Sec. 7 Project Management	**Document Merge** ***11 points***	40%		40%		20%	
		Project Coord./Progress ***5 points***	50%				50%	
		Plan of Work ***2 points***			100%			

Table of Contents

1. Interaction Diagrams

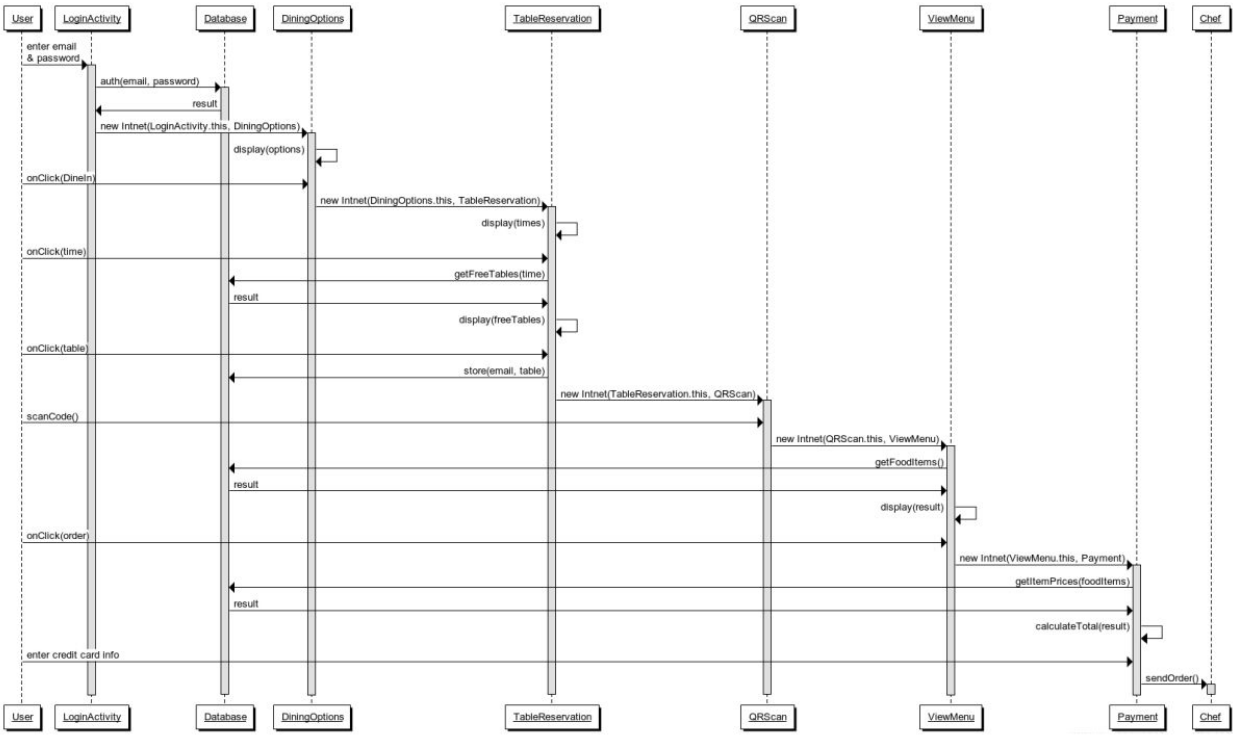


Figure 1 UC-1: Dine-In

This diagram demonstrates the interactions between classes for UC-1: Dine-In. After logging in to our app, the customer is given a dining options screen, where they would choose to “Dine-In”. This takes us to the table reservation screen where the user selects the time for reservation and then is shown the free tables available in the restaurant. After selecting their table, the customer is asked to scan the QR code for the table they have selected. Once scanned, The customer may view the menu and select all food items desired for their meal. Clicking on the order button brings the customer to the payment screen, where they enter their credit card information. Once paid for, the customer’s order is sent to the Chef in order to be cooked.

The design principles employed in the process of assigning responsibilities to objects were the expert doer principle and high cohesion principle. The expert doer principle is employed because each class is an expert for specific functions. For example, the TableReservation object only handles selecting a time and table for the customer, and once done, it pass on responsibility to the QRScan object to handle scanning the QR code located on a table. The high cohesion principle is used because each class only handles computations for its specific functionality, and does not attempt to handle more than needed. This goes hand in hand with the expert doer principle. Finally, the low coupling principle is not used here because it conflicts with our expert doer and high cohesion principles. In order to employ the low coupling principle, we would need to reduce the amount of communication we have in the interactions

between objects. We opted for more communication and less computations for each object in order to reduce the amount of work each class would be responsible for.

The alternative solutions considered for UC-1: Dine-In consisted of payment for the customer's meal as one of the last steps for the process. We decided that this would not be a favorable idea since some customers might be motivated to attempt to get a free meal by placing an order and not paying. Originally, the customer would order their meal, and the chef would immediately start cooking it. Once the customer was ready to leave the restaurant, they would then be asked to pay for their meal. We decided that in order to fix this issue, we would have the customer pay for their meal right after hitting the order button. This way, the order does not get sent to the chef before payment is received and we know that the customer won't try to get out of paying their bill.

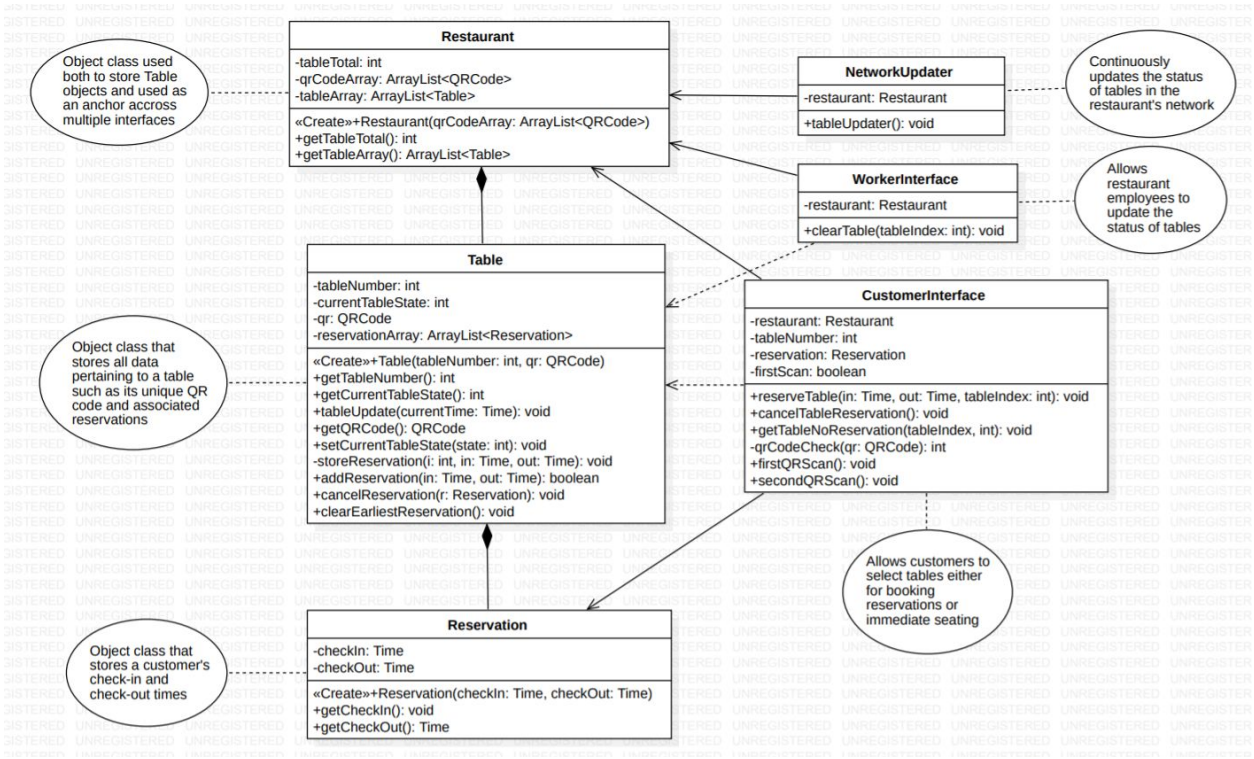


Figure 2 UC-5: Table Selection

This diagram demonstrates the interactions between classes for UC-5: Table Selection. After logging in to our app, the customer is given a dining options screen, where they would choose to "Dine-In". This takes us to the table selection screen where the customer is asked to pick a time to make a table reservation, and then select the table they would like to sit at, provided that the table is not taken. Once the table selection process is completed, the customer is asked to scan the QR code provided on the table where they chose to sit, confirming their reservation.

The design principles employed in the process of assigning responsibilities to objects were the expert doer principle and high cohesion principle. The expert doer principle and the high cohesion principle are employed for the same reasons as in UC-1: Dine-In.

The alternative solutions considered for UC-5: Table Selection consisted of only asking the user to select a table without a time for reservation. We realized that customers might want to make a reservation for other days or for more than a couple hours away from their planned meal. We fixed this by first asking the customer the day/time they would like to make a reservation for and then ask them to choose a free table. The database holds all the information about free tables and will show the customer in real-time the tables which are taken or untaken for that day/time.

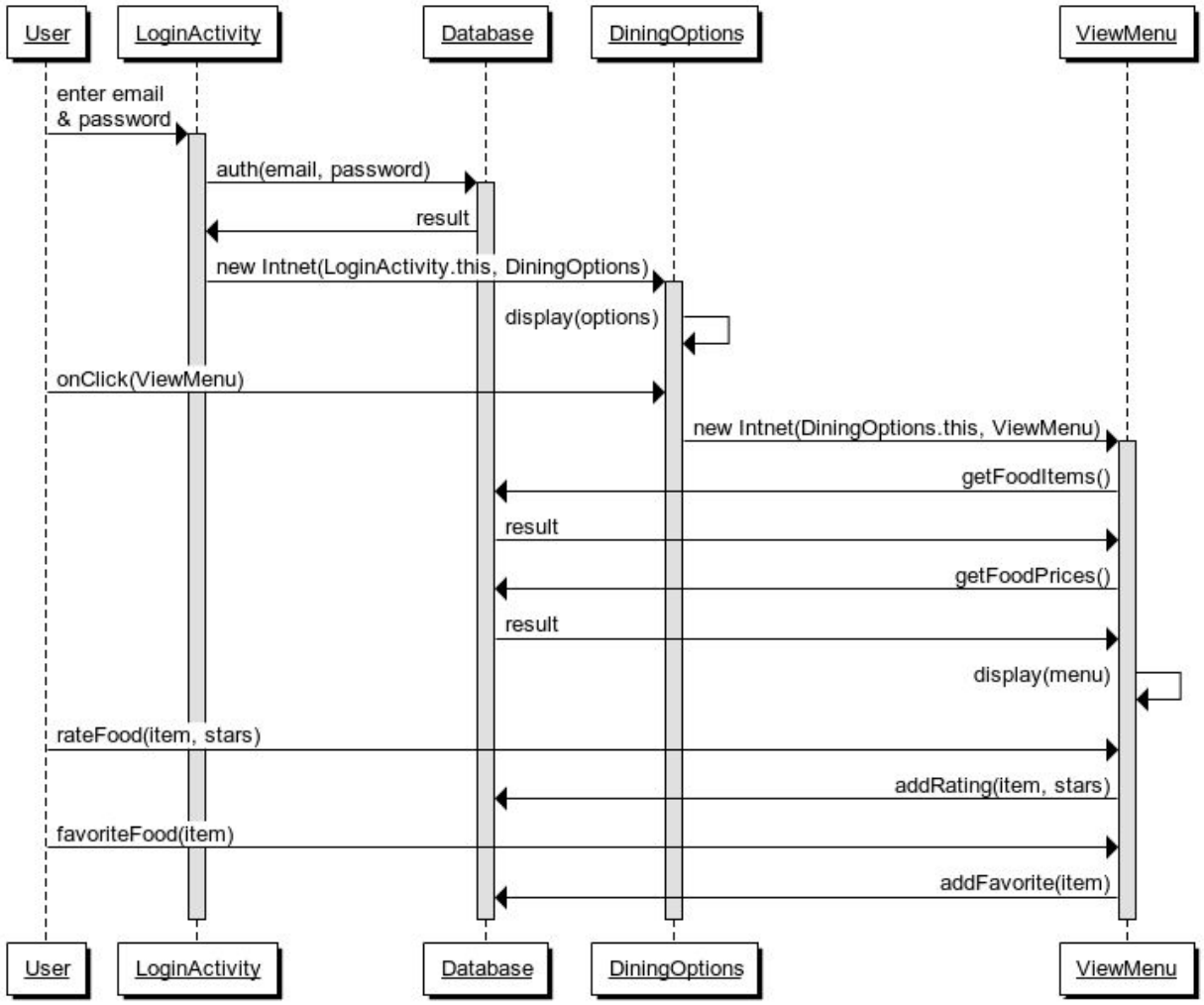


Figure 3 UC-4: View Menu

This diagram demonstrates the interactions between classes for UC-4: View Menu. After logging in to our app, the customer is given a dining options screen, where they would choose to “View Menu”. This takes us to the menu screen where we can see all of the food items available at the restaurant. From this screen, the customer is able to rate a food item by clicking the number of stars (1 to 5 stars) and also favorite a food item by clicking on the heart icon next to it. This screen is also available when the

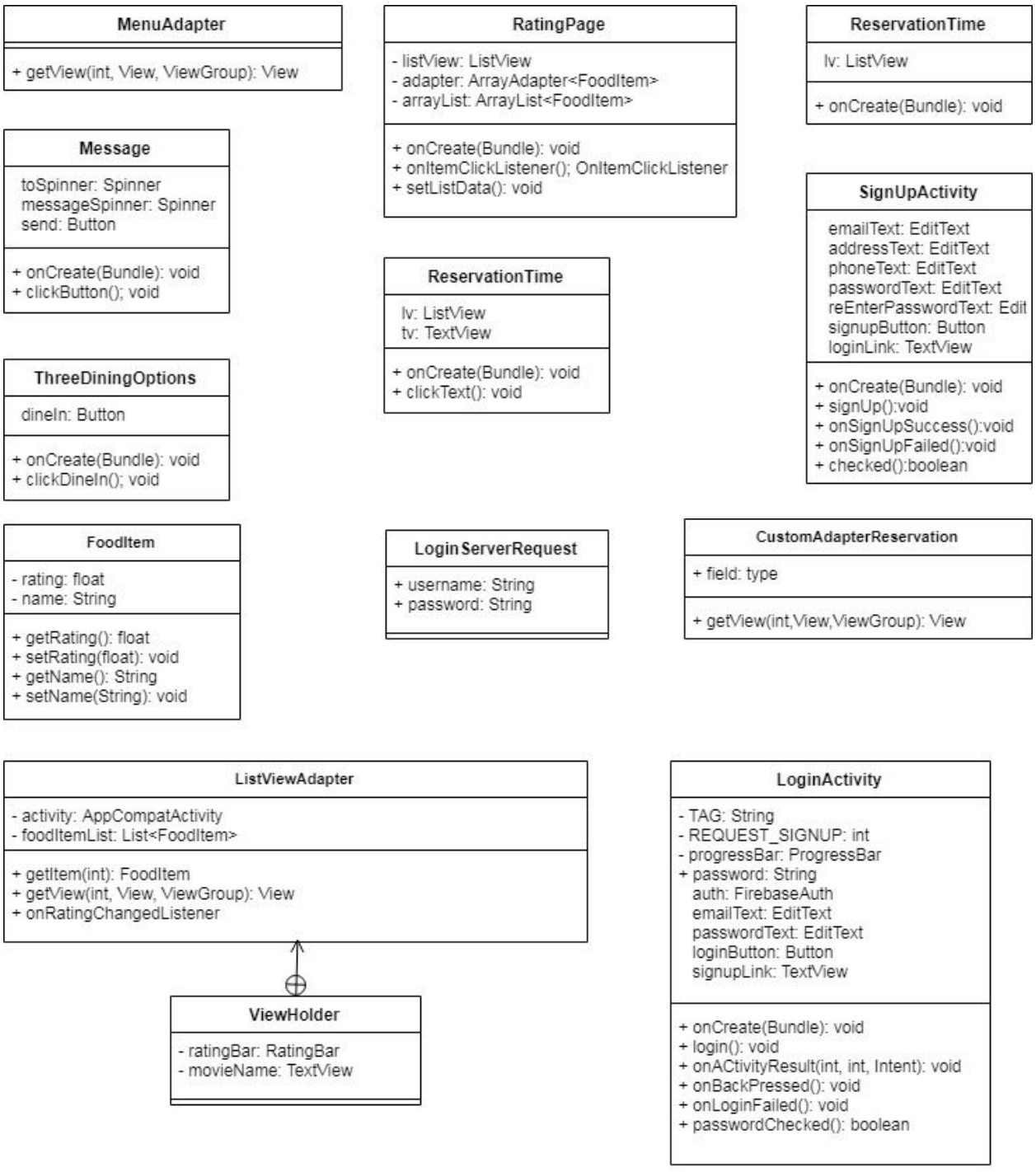
customer picks the “Dine-In” option after selecting their table. The user is able to select the food items desired for their meal.

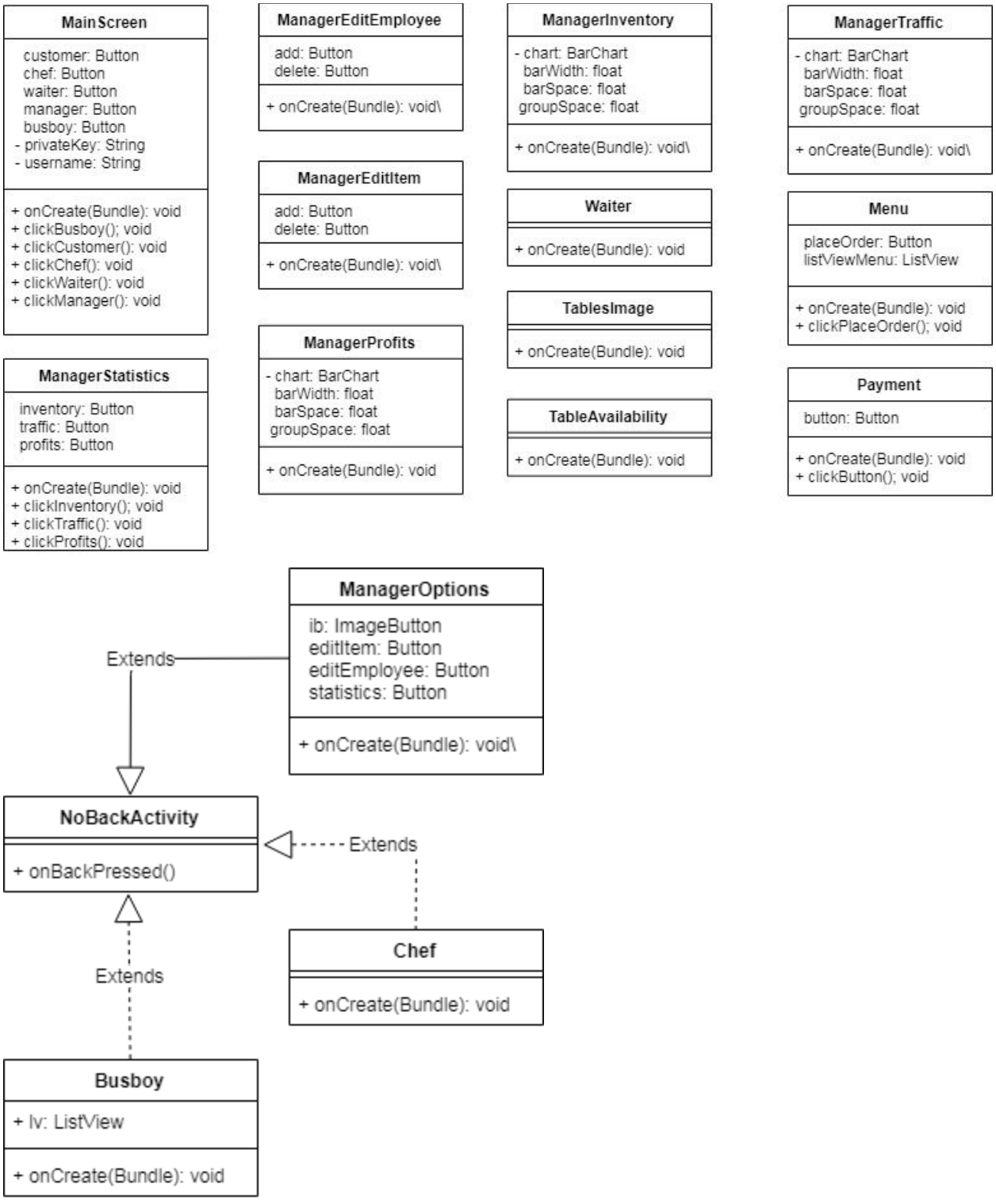
The design principles employed in the process of assigning responsibilities to objects were the expert doer principle and high cohesion principle. The expert doer principle and the high cohesion principle are employed for the same reasons as in UC-1: Dine-In.

The alternative solutions considered for UC-1: Dine-In consisted of only allowing the user to rate foods after having purchased them. We realized that this would require the menu screen to be brought up again at the end of the customer’s meal, at which point the customer may not be interested in rating the food items. Our solution was to allow the customer to interact with the menu as they are ordering, allowing them to rate or favorite foods they have eaten before.

2. Class Diagram and Interface Specification

2.1 Class Diagram




MainScreen
customer: Button
chef: Button
waiter: Button
manager: Button
busboy: Button
- privateKey: String
- username: String
+ onCreate(Bundle): void
+ clickBusboy(); void
+ clickCustomer(): void
+ clickChef(): void
+ clickWaiter(): void
+ clickManager(): void
ManagerEditEmployee
add: Button
delete: Button
+ onCreate(Bundle): void\
ManagerInventory
- chart: BarChart
barWidth: float
barSpace: float
groupSpace: float
+ onCreate(Bundle): void\
ManagerTraffic
- chart: BarChart
barWidth: float
barSpace: float
groupSpace: float
+ onCreate(Bundle): void\
ManagerEditItem
add: Button
delete: Button
+ onCreate(Bundle): void\
Waiter
+ onCreate(Bundle): void
Menu
placeOrder: Button
listViewMenu: ListView
+ onCreate(Bundle): void
+ clickPlaceOrder(); void
ManagerStatistics
inventory: Button
traffic: Button
profits: Button
+ onCreate(Bundle): void
+ clickInventory(); void
+ clickTraffic(): void
+ clickProfits(): void
ManagerProfits
- chart: BarChart
barWidth: float
barSpace: float
groupSpace: float
+ onCreate(Bundle): void
TablesImage
+ onCreate(Bundle): void
Payment
button: Button
+ onCreate(Bundle): void
+ clickButton(); void
TableAvailability
+ onCreate(Bundle): void
ManagerOptions
ib: ImageButton
editItem: Button
editEmployee: Button
statistics: Button
+ onCreate(Bundle): void\
Extends
NoBackActivity
+ onBackPressed()
Extends
Chef
+ onCreate(Bundle): void
Extends
Busboy
+ lv: ListView
+ onCreate(Bundle): void

2.2 Data Types and Operation Signatures

NoBackActivity:

Attribute:	Type:

Method:	Return Type:
onBackPressed()	void

BusBoy:

Attribute:	Type:
lv	ListView

Method:	Return Type:
onCreate(Bundle)	void

MangerOptions:

Attribute:	Type:
ib	ImageButton
editItem	Button
editEmployee	Button
statistics	Button

Method:	Return Type:
onCreate(Bundle)	void

Chef:

Attribute:	Type:

Method:	Return Type:

onCreate(Bundle)	void

MainScreen:

Attribute:	Type:
customer	Button
chef	Button
waiter	Button
manager	Button
busboy	Button
privateKey	String
username	String

Method:	Return Type:
onCreate(Bundle)	void
clickBusboy()	void
clickCustomer()	void
clickChef()	void
clickWaiter()	void
clickManager()	void

ManagerEditEmployee:

Attribute:	Type:
add	Button
delete	Button

Method:	Return Type:

onCreate(Bundle)	void

ManagerInventory:

Attribute:	Type:
chart	BarChart
barWidth	float
barSpace	float
groupSpace	float

Method:	Return Type:
onCreate(Bundle)	void

ManagerTraffic:

Attribute:	Type:
chart	BarChart
barWidth	float
barSpace	float
groupSpace	float

Method:	Return Type:
onCreate(Bundle)	void

ManagerEditItem:

Attribute:	Type:
add	Button
delete	Button

Method:	Return Type:
onCreate(Bundle)	void

Waiter:

Attribute:	Type:

Method:	Return Type:
onCreate(Bundle)	void

Menu:

Attribute:	Type:
placeOrder	Button
listViewMenu	ListView

Method:	Return Type:
onCreate(Bundle)	void
clickPlaceOrder()	void

ManagerProfits:

Attribute:	Type:
chart	BarChart
barWidth	float
barSpace	float
groupSpace	float

Method:	Return Type:
onCreate(Bundle)	void

TableImage:

Attribute:	Type:

Method:	Return Type:
onCreate(Bundle)	void

TableAvailability:

Attribute:	Type:

Method:	Return Type:
onCreate(Bundle)	void

Payment:

Attribute:	Type:
button	Button

Method:	Return Type:
onCreate(Bundle)	void
clickButton()	void

MenuAdapter:

Attribute:	Type:

Method:	Return Type:
getView(int, View,ViewGroup)	View

Message:

Attribute:	Type:
toSpinner	Spinner
messageSpinner	Spinner
send	Button

Method:	Return Type:
onCreate(Bundle)	void
clickButton()	void

ThreeDiningOptions:

Attribute:	Type:
dineIn	Button

Method:	Return Type:
onCreate(Bundle)	void
clickDineIn	void

FoodItem:

Attribute:	Type:
rating	float
name	String

Method:	Return Type:
getRating()	float
setRating(float)	void
getName()	String

setName(String)	void

ListViewAdapter:

Attribute:	Type:
activity	AppCompatActivity
foodItemList	List<FoodItem>

Method:	Return Type:
getItem(int)	FoodItem
getView(int,View,ViewGroup)	View
onRatingChangedListener	void

ViewHolder:

Attribute:	Type:
ratingBar	RatingBar
movieName	TextView

Method:	Return Type:

RatingPage:

Attribute:	Type:
listView	ListView
adapter	ArrayAdapter<FoodItem>
arrayList	ArrayList<FoodItem>

Method:	Return Type:

onCreate(Bundle)	void
onItemClickListener()	OnItemClickListener
setListData()	void

ReservationTime:

Attribute:	Type:
lv	ListView
tv	TextView

Method:	Return Type:
onCreate(Bundle)	void
clickText()	void

LoginServerRequest:

Attribute:	Type:
username	String
password	String

Method:	Return Type:

ReservationTime:

Attribute:	Type:
lv	ListView

Method:	Return Type:
onCreate(Bundle)	void

SignUpActivity:

Attribute:	Type:
emailText	EditText
addressText	EditText
phoneText	EditText
passwordText	EditText
reEnterPasswordText	EditText
signupButton	Button
loginLink	TextView

Method:	Return Type:
onCreate(Bundle)	void
signUp()	void
onSignUpSuccess()	void
checked()	boolean

CustomAdapterReservation:

Attribute:	Type:
field	type

Method:	Return Type:
getView(int,VIew,ViewGroup)	View

LoginActivity:

Attribute:	Type:
TAG	String

REQUEST_SIGNUP	int
progressBar	ProgressBar
password	String
auth	FirebaseAuth
emailText	EditText
passwordText	EditText
loginButton	Button
signUpLink	TextView

Method:	Return Type:
onCreate(Bundle)	void
login()	void
onActivityResult(int, int, Intent)	void
onBackPressed()	void
onLoginFailed()	void
passwordChecked()	boolean

2.3 Traceability Matrix

Classes	Domain Concepts								
	Customer Profile	Interface	Controller	Communicator	Order Queue	Analytic Calc	Table Status	Food Status	Payment System
MenuAdapter		X							
RatingPage		X							
ReservationTime			X						
Message				X					
SignUpActivity		X							
ThreeDinningOptions			X						
FoodItem								X	
LoginServerRequest				X					
CustomAdapterReservation			X						
ViewHolder		X							
ListViewAdapter		X							
LoginActivity		X							
MainScreen		X							
ManagerStatistics						X			
ManagerEditEmployee				X					
ManagerEditItem				X					
ManagerProfits						X			
ManagerInventory						X			
ManagerOptions		X							
Waiter		X							

TablesImage		X							
TableAvailability							X		
ManagerTraffic						X			
Menu			X						
Payment									X
NoBackActivity			X						
Chef		X							
Busboy		X							

- Customer Profile: the initial design features of this domain concept were implemented into other classes, such as implementing the 'ReservationTime' class and utilizing the 'RatingPage'
- Interface
  - MenuAdapter: Allows users to few menu via interface
  - RatingPage: Allows customers to interact with ratings
  - SignUpActivity: Allows users to interact in creating new accounts
  - ViewHolder: Is shown via interface
  - ListViewAdapter: Can be interacted with through the interface
  - LoginActivity: Can be accessed through the interface
  - MainScreen: Can be interacted with through the interface
  - ManagerOptions: Manager accesses options through interface
  - Waiter: Waiter can interact with their UI via interface
  - TablesImage: Can be seen via interface
  - Chef: Chef can interact with their UI via interface
  - Busboy: Busboy can interact with their UI via interface
- Controller
  - ReservationTime: Controller requires customer to set reservation
  - ThreeDinningOptions: Only permits customers three options upon login
  - CustomAdapterReservation: Allows reservations to be made
  - Menu: Allows customers to pick meals from menu
  - NoBackActivity: Allows users to log into their designated accounts
- Communicator
  - Message: Communicator allows manager to contact employees
  - LoginServerRequest: Allows accounts to acquire information from the server
  - ManagerEditEmployee: Allows manager to modify employee info via communicator
- Order Queue: the initial design features of this domain concept were implemented into other classes, such as in 'FoodItem' to place orders from and in 'Chef' for cooks to view and make said orders
- Analytic Calc

- ManagerStatistics: Calculates statistics in manager account
- ManagerProfits: Calculates profits in manager account
- ManagerInventory: Allows manager to view inventory

- Table Status
  - TableAvailability: No changes were made; all features described in the 'Table Status' concept were implemented into the 'TableAvailability' class
- Food Status
  - FoodItem: No changes were made; all features described in the 'Food Status' concept were implemented into the 'FoodItem' class
- Payment System
  - Payment: No changes were made; all features described in the 'Payment System' concept were implemented into the 'Payment' class

3. System Architecture and System Design

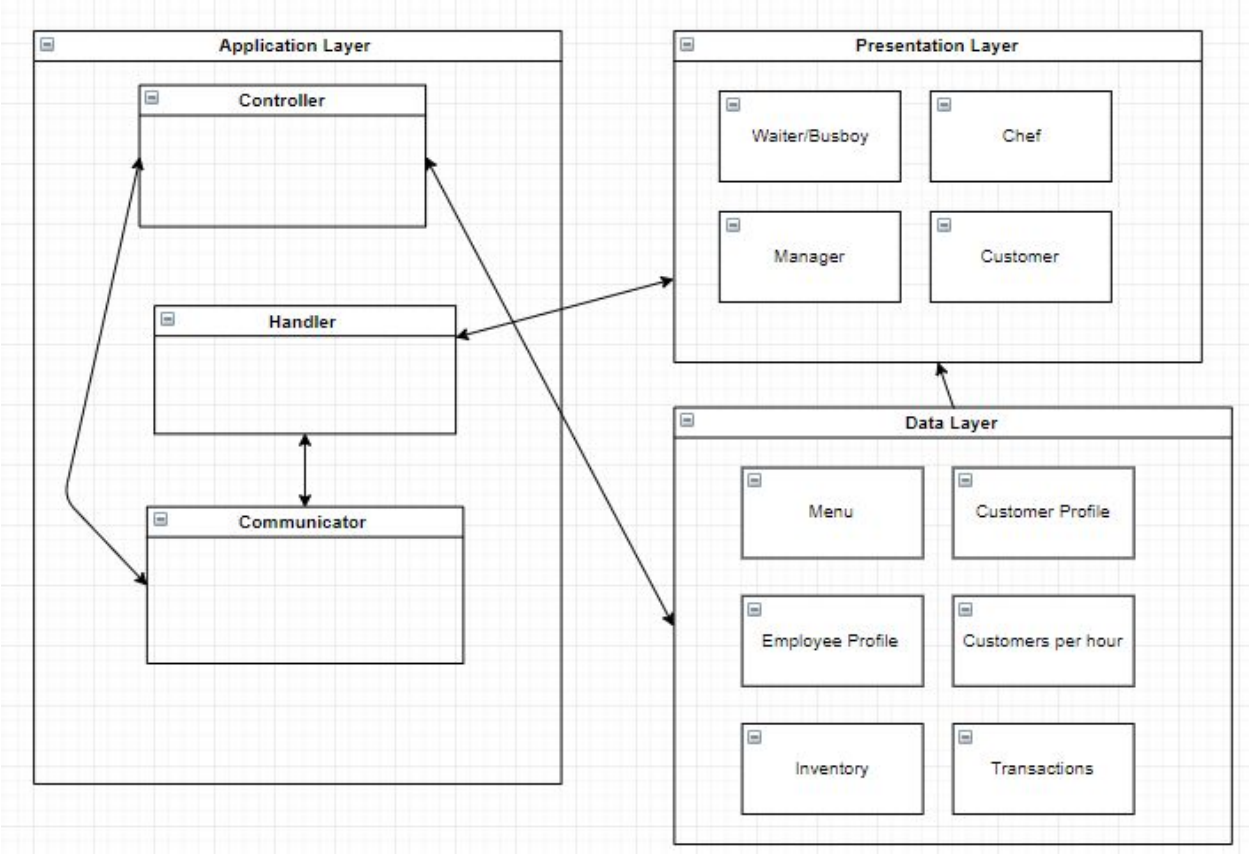
3.1 Architectural Styles

The function of a system architecture is to provide mechanisms and an abstraction of the underlying process of our entire framework. Our system uses a 3-layer architecture consisting of a presentation layer, an application layer, and a data layer. This is analogous to the frontend/backend in website development, where the presentation layer is what is seen by the user, and the application and data layers work behind the scenes.

This scheme is the most logical as the layers are abstracted from each other and will run parallel into the code. The differences in UI design, application code, and database calls are apparent. Our project is constructed through Android Studio, where the UI is designed through the program's graphical interface. The application will be "connected" where different parts of UI are matched with the application code. Within the code, database calls will be made via using an API, which will help pull information real-time in order to aid the user in their day-to-day actions. This structure is easy to follow and implement.

Our application also utilizes a client/server architectural style in which the server is consistently updating the database of our application. This architectural style segregates the system into two applications, where the client makes requests to the server. In our case, the server is a database with application logic represented as stored procedures. The database is responsible for a list of all menu items, their prices, wait times, and even ratings each item. Also, the statistics such as inventory tracking, customer peak times, and total profits are also stored. On top of all of this, the database also holds all information regarding user logins and account permissions, leading to an efficient user experience after logging in.

3.2 Identifying Subsystems



The subsystem shows our three layer system of an application layer, presentation layer, and data layer. The data layer consists of the database which stores information of different user profiles, menu, customer metrics, and transactions. The presentation layer holds the different screens that the user will see depending on their profile. This layer can pull information from the database when displaying the screens. The presentation layer consists of the waiter/busboy, chef, manager, and customer will have their own interface system with their own unique screens. The application layer contains what will call and manage the whole operation. The controller will facilitate the tasks between the packages. It uses the communicator which will handle the authorizations between packages. It also assigns tasks to the handler which is responsible for loading events which it is told to from the communicator.

3.3 Mapping Subsystems to Hardware

Our application works with a native device (Android) and a database server. Our application will utilize a database with a public API and will run as long as it is active. The application can run on various android

devices that meet the minimum operating system requirements as listed in Section 3.7 Hardware Requirements. The Database subsystem will naturally run the database with the Application subsystem interacting with it. To store a user's personal data, the application needs an external database that is hosted on a different server to send the information to the application in real time. The application subsystem will run on Android devices using Android operating systems that are version 4.0 and above since they meet the minimum software requirements to run the application. The server subsystem runs on a external online server called Firebase which hosts all necessary database information for the all the users and can transfer information to the application. Each instance of the client will be on different mobile devices, with each mobile device communicating with the database of the system.

3.4 Persistent Data Storage

The system does store persistent data which need to be accessed after logging out of the app. These data include customer profiles which needs to store each customer's ratings as well as favorites. Customer profile information will be stored in the database provided by Firebase, which also acts as a realtime database with cloud storage. The database will be set up such that each customer username is the key for the database and all ratings and foods will be stored as entries under this name. When a customer logs in, his entry will be pulled from the database and written into a list for ratings and a seperate one for favorites.The restaurant itself will also have a lot of information consisting of inventory and other information which the manager needs to know. All this data will be stored in Firebase as well but in a different format. We will have a table for inventory, customer peak times, and other information which the manager requires. For the inventory, we will have an entry for each item as well as a corresponding amount of that item. This list will be constantly updated as the orders are placed. It will be loaded into the app when the manager logs in and views the inventory. The customer peak times will be stored in the database periodically. Every hour, the app will send information to the database of how many customers confirmed a table for that hour. Each hour will be its own entry. This data will be pulled into the app on manager request as well.

```
Customer :: {
        name:String,
        username:String
        password:String
        orders:[Order]
}

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

Item :: {
        name:String
```

```
        quantity:Number
        price:Number
        rating:Double
}

Menu :: {
        items:[Item]
}
```

3.5 Network Protocol

The network protocol to be used for our purposes is normal sockets. The purpose for going about transmitting information in this manner is that all the backend information processing will be done on the server computer, while the app will mainly serve as little more than a graphical frontend for the end user. The types of messages and message format are thus similarly as simple to reflect this design decision, with type of messages reflecting the type of information requested, and message format being that one entry is sent per line, and if that entry has multiple items they will be separated with delimiters.

3.6 Global Control Flow

Execution Orderness

This project will be an event-driven system. The user will have complete control over how they use the application. There is no linear procedure for the user to take, and they do not even have to use all of the features that the app provides. The user can use the app's interface to use any function at any time.

Time Dependency

The system depends on real time. In order to keep track of the customer traffic, inventory, and profits the system must know when to reset its daily timer as well as when to notify managers of shortages, etc. There is a 24 hour timer for daily resets and a weekly timer to show the progress over a week. Along with this, there are timers within the system for daily tasks. These include the countdown to when an order is prepared by the chef, and when a reservation for a specific table is made for dining in. The customers will be able to view both of these timers in real-time, to be alert of how much time has passed.

Concurrency

As we utilize different threads per request to our database, synchronization is automatically enforced via Firebase because of the fact that there is a level of mutual exclusion that occurs when the data is called or manipulated.

3.7 Hardware Requirements

This software will be run on mobile smartphones with touch screen display and network/WiFi support. The required operating system will be Android, with a minimum and target API 23:

Marshmallow. By targeting API 23, the application is set to run on approximately 62.6% of devices. This platform is accessible and the requirements are met through most Android phones. The amount of space needed to download the application on an Android phone is at least 1MB while the space needed to install the app is .5 MB. The minimum resolution to properly display the images in the application and view them is 640 x 480 pixels. The minimum bandwidth required to access the server and database is 56 kbps. The minimum RAM requirements to display and render the graphics and images of the application is 1 GB. To allow the user to use the QR scanning functionality, the phone must have an inbuilt camera that is at least 1 megapixel.

4. Algorithms and Data Structures

4.1 Algorithms

Customer: The customer has the option of choosing whether he/she wants to dine in, takeout, or simply just view the menu. After this, they can select items from the menu and to maximize efficiency, the items selected are added to an expandable arraylist as they are selected by the customer. If the customer un-selects an item, the item will be removed from the arraylist. This process will be O(n) time and O(n) space. For the customers choosing to dine in, the table selecting algorithm will come into play. For reserved tables, as the time nears the time of reservation selected by the customer, an alert will be given. For the tables that are currently unavailable (taken by others), or dirty from previous customers, the algorithm will only free up the tables for new customer selection after the busboy or waiter chooses the option to do so, on their end of the portal. This would be based off of how much traffic is occurring within the restaurant, but a timing of O(n) would be efficient for most customers.

Manager: The manager end of the application will be quite straightforward. The statistics of the different information stored daily will be pushed to our front end to form visual graphs for ease of use. For this to happen fluently, the database, which has constant updates from the customer's end (for each meal ordered), will be queried for the information whenever the manager enters the portal. Accessing the particular table will be O(1) time. When it comes to waiting for the updated information from the database and outputting it into the proper table/chart, the Big O time will be closer to O(n). This algorithm is still being worked on to help optimize the timings.

Chef: The chef will be able to view all of the orders placed by every customer within their portal. After the customer's order is placed into the arraylist, the chef will have the updated list in real time. When the chef begins cooking a specific meal, he will be able to alert the system (and the customer) that he has begun, with an approximate timer for how much time he needs to finish cooking the meals. The algorithm will be constantly querying for any new items in the arraylist within the loop. When a specific meal is finished cooking, it is removed from the arraylist and the chef is left with only the next meals he must cook. The big O timing for this all is O(n^2).

4.2 Data Structures

The primary data structure we are using is arraylist. This is because it has a simple O(n) look up time which does not add much delay to our performance. More so, this data structure is very flexible. It is very easy to add onto an arraylist as it only takes O(1) time.

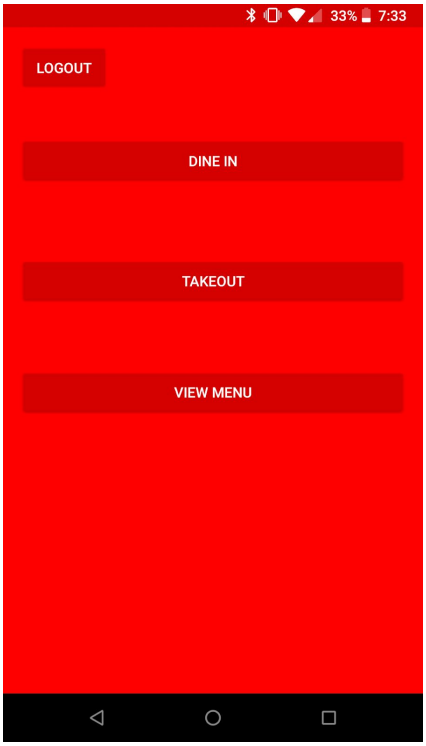
Arraylists are also compatible with listview. This is very important as we are constantly using listviews to display menu, ordered items, and other information. An arraylist can be easily passed into an arrayadapter to be displayed in a listview. Other data structures such as hash tables or linked lists require more work to convert.

We wanted a list type structure also due to the compatibility with the database. Since our database is SQL based, it can be looked at as a list. The values on the tables in our database can be very easily read into lists. It also does not take much effort to write the list back into the database.

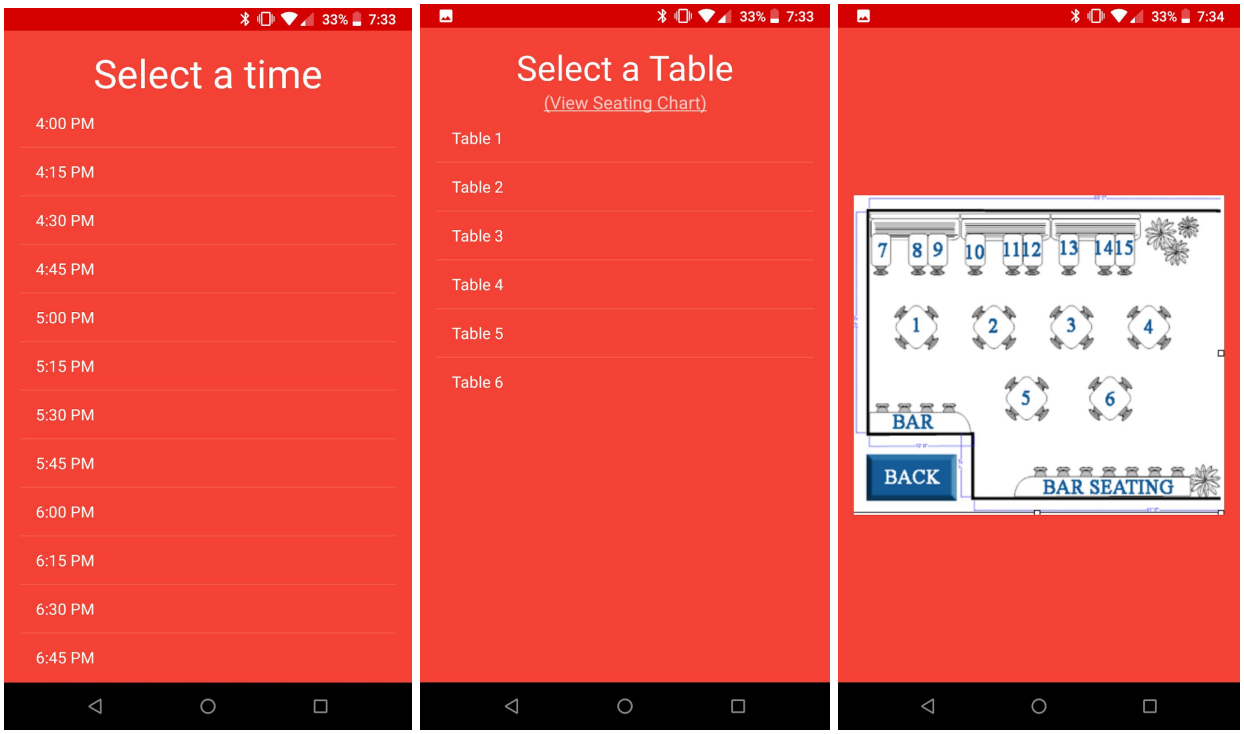
5. User Interface Design and Implementation

Customer Interface:

The customer is first asked to select one of the three options: Dine In, Takeout, and View Menu. If Takeout or View Menu are selected, the user is taken to the main menu, where they can view the entire menu, ratings, and how long each food takes to cook. If Dine In is selected, the user is directed to the Reservation Page.

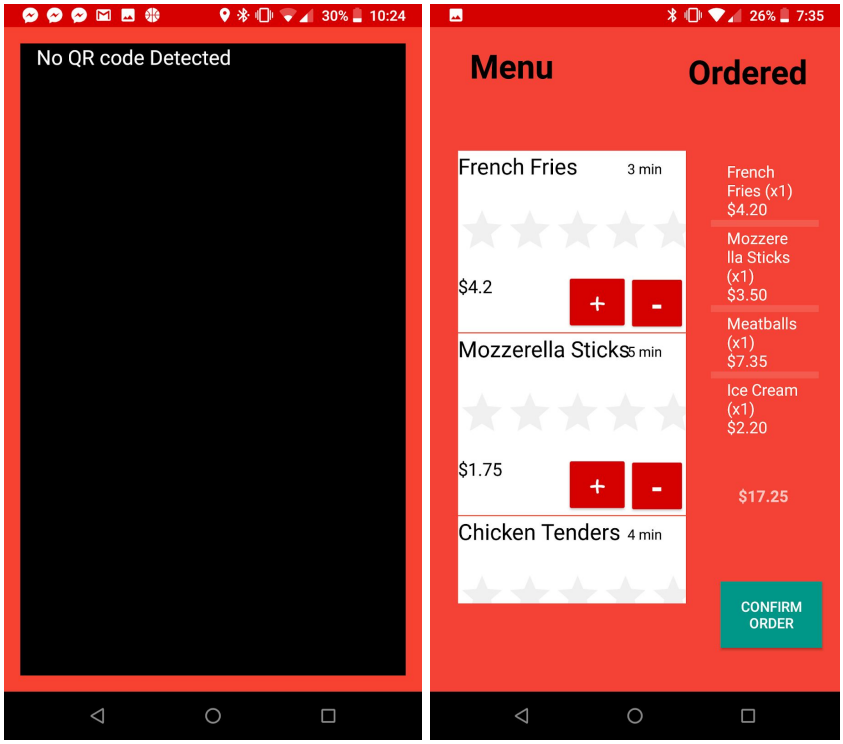


As mentioned, if the customer selects Dine In, they are brought to the Reservation Page. This is an interactive scroll menu that is populated with times in 15 minute intervals. Once the user selects a time, the user is directed to the Table Selection page.



After selecting a time the user is brought to the table selection page, this is an interactive scroll menu that lists all of the tables in the restaurant. If a table is crossed out, this means that the table is unavailable at the current time. For the new customers that are unsure of which tables are which, they can view the table seating chart at the top of the page, which shows an image of an overhead view of the entire restaurant. Once they know which table they want to sit at, they choose the corresponding table number. The table is then reserved for the user at that time selected.

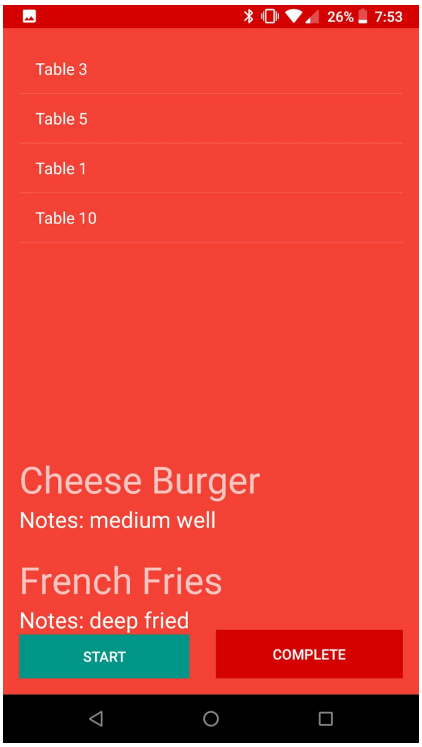
Once the customer sits at the table, they will be prompted to confirm their seat with the in-app QR scanner. The customer will then scan the QR code swiftly, and the table seating will be confirmed. The customer will then be brought to the full menu page.



The full menu page will be yet another interactive scrolling list. Each food item will have a rating, a time it takes to cook, and an adjustable quantity to order. The customer will also be able to hold down on the menu item to input any specific notes to the chef, such as any allergies/requests. Once the customer is satisfied with their order, they can confirm the order is accurate with the real-time receipt being populated on the right side. After that, they can click place order to fully confirm their order. After the order is confirmed, the customer can pay with Credit Card or Cash, and can also rate their meal afterwards, 1-5 stars. These ratings will be inputted into our system and averaged into our menu page for future customers to view.

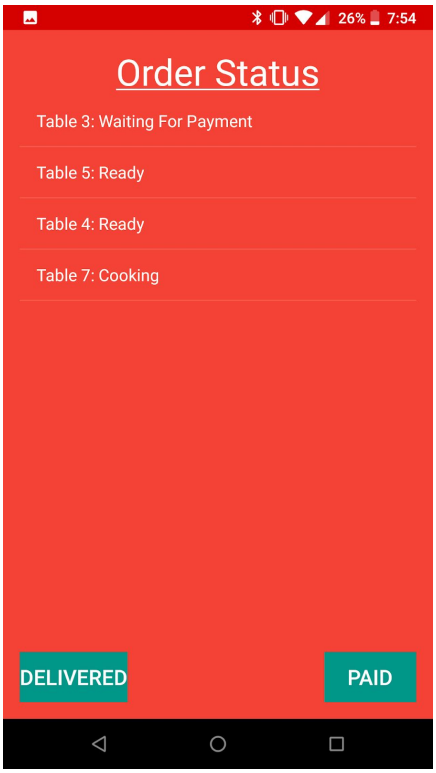
Chef Interface:

The chef's interface will always contain the same page, but will have a lot of information being constantly updated and populated. The table number display shows the specific table's order in a queue that it was received in. Once the table is selected by the chef, the chef can click the Start button to begin working on that specific order. This notifies the customer that their order has commenced cooking. As the timer ticks down, the chef can then pre-maturely select the Complete button to notify the waiter that the order has been finished. The order is then removed from the chef's queue, and the server is able to pick up the order and distribute it to the correct table.



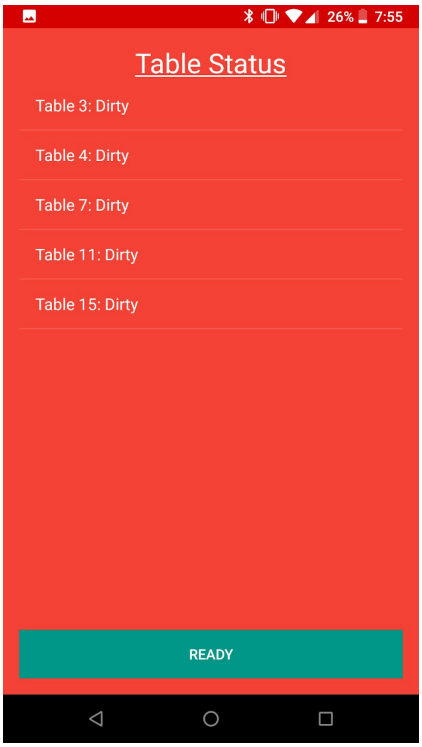
Server Interface:

The server interface is the most simple interface. The server Order Status is derived from the Chef's interface. When a chef starts cooking an order, it appears on the server's interface as "cooking". This lets the server know that the chef has begun cooking that specific order. Once the chef is finished cooking and changes the status to Ready, the server is notified. The server can pick up the food and bring it to the customer flawlessly. After the customer has eaten, paid, and left, the table is removed from the Server's table list, and then a notice is sent to the busboy's interface for cleaning.



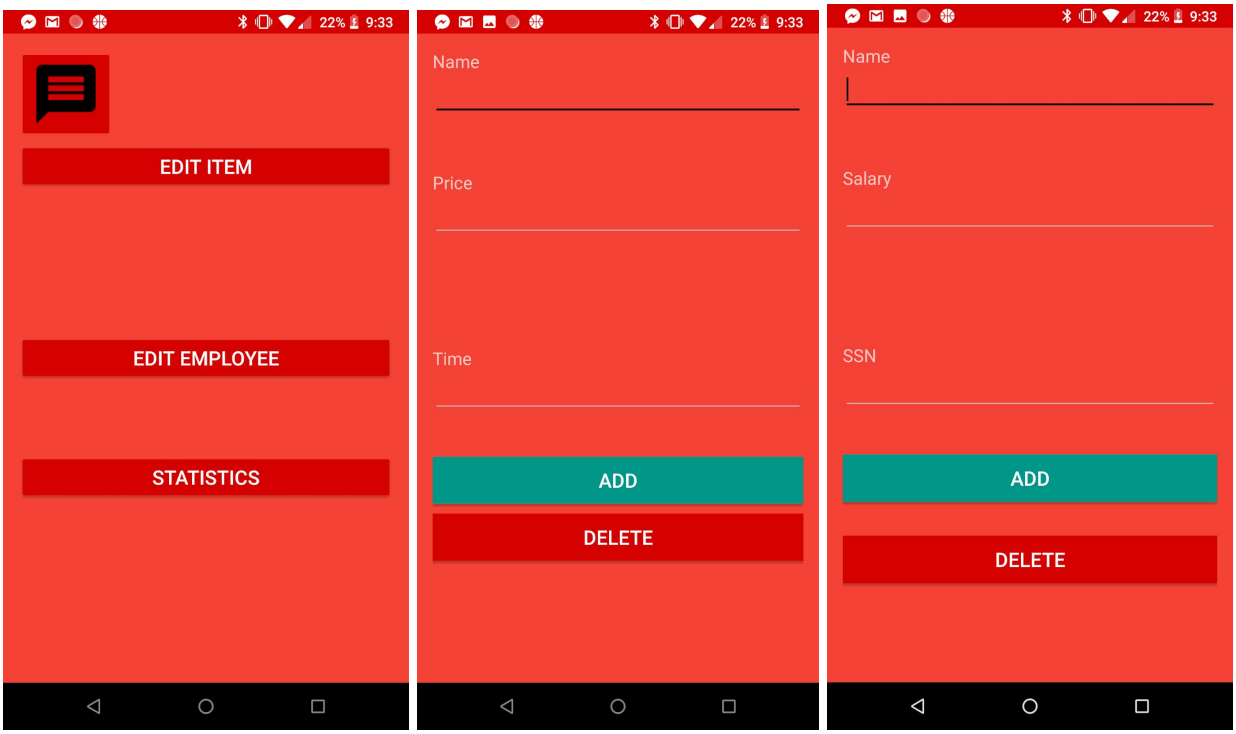
Busboy Interface:

Once the busboy receives the message from the Server that a customer has paid and left, the corresponding table appears on the Table Status list. This notifies the busboy which tables are available for cleaning. After the table has been cleaned and prepped for future customers to use, the busboy simply selects the table that was cleaned and clicks the “Ready” button. This will remove the table from the list and sets the table as available for all future customers.



Manager Interface:

The first options available for the manager are Edit Item, Edit Employee, and Statistics. If the manager chooses to edit an item or an employee, they will be forwarded to an interactive page with text boxes to edit the chosen information. If the statistics page is selected, the manager will be forwarded to the Statistics Selection page.

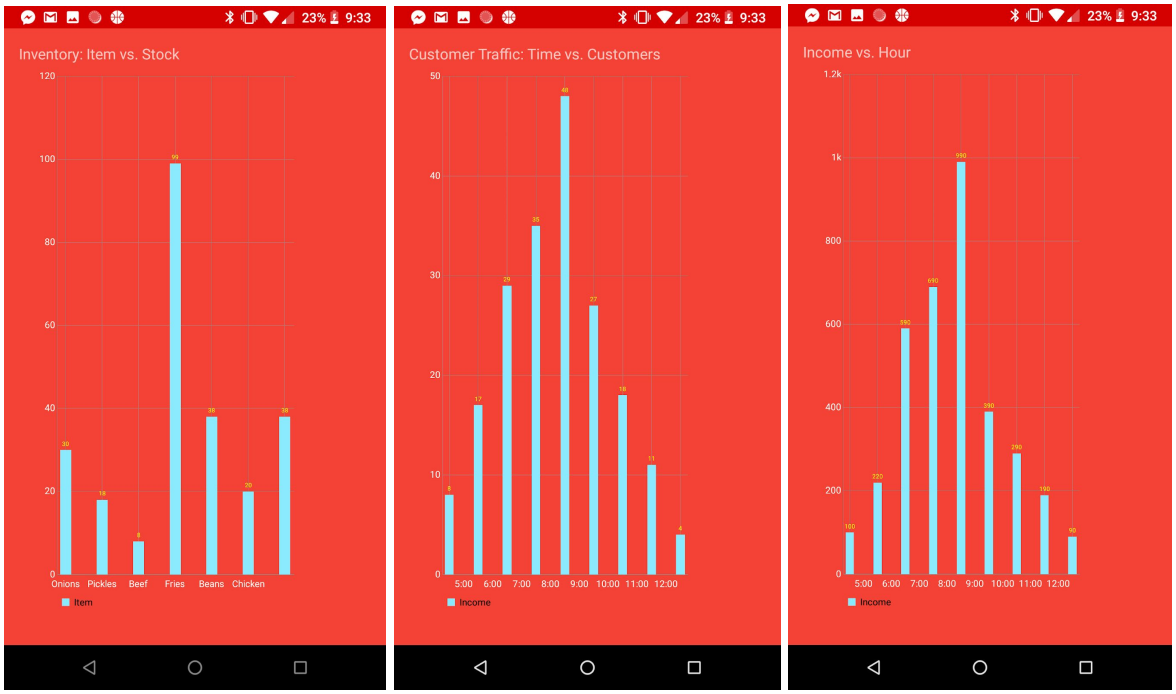


If Edit Item is selected, the manager simply inputs the information for the item and selects Add or Delete. These buttons will query our database for the menu item, and if it is not found after the Add button was selected, it will add that item to the arraylist of our menu.

If the manager selects the Edit Employee button, the manager simply has to input the information for the employee and selects Add or Delete. This will add or remove the employee from the corresponding employee system. This is directly tied to the clocking in and payroll features that we plan to implement in the future. Also, to give waiter, chef, or busboy permissions to a certain account, that employee must be in the system via the manager's portal.

If the manager selects Statistics, they will be forwarded to the Statistics selection page. This is where all of the manager statistics are displayed. The manager can then choose specifically which statistics he or she wants to view.

Inventory, traffic, and profits interfaces:



Within all of these interfaces, if the user wishes to go navigate back to a previous page, the user can simply swipe right on the edge of their screen. Also, with most android phones, there is a dedicated backwards navigate button that they can also use.

Our design has improved a lot since the beginning stages of our initial ideas. From the original GUI drawings to now, there has been a lot of optimizations within the menu, the manager portal, and even the backend algorithms and database. We have streamlined the design to help guide the user through the entire process--which really helps optimize the app and make users want to use it again in the future. This also keeps the user on task which results in a faster transaction time which in the end, benefits the restaurant and customers, alike.

As mentioned, there have been many design alterations from our original GUI drawings to our current layouts. In the initial GUI design, the restaurant owner was only able to view statistics such as inventory and customer info. Now in addition to statistics, our design allows the owner to send messages to their employees and edit employee and menu information. Initially chefs could only view incoming orders, but now they can interact with these orders, being able to set once they've started on a meal and when they've completed a meal. They can also view meals based on what tables the orders came from and what priority the tables have based on time an order was placed and how long each meal from a table will take to complete. The only other type of employee specified in the initial drawings was a waiter, whom could only view the status of tables. Now we have both busboys and servers; a busboy is notified when a table needs to be cleaned and can set a table's status as 'open' upon cleaning a table, and a server is notified when to take out an order and can set an order to 'paid' after it's paid (in the case the payment is in cash) and can set an order as 'delivered' after being delivered to a table. As for the customer user interface, a customer from the main window can view the restaurant's menu in addition to selecting 'Dine In' and 'Takeout'. Also, upon selecting 'Dine In' the customer is asked to select a time in which they would like to dine prior to being taken to the table selection window. Another modification is from the payment window, the customer is automatically taken to the rate meal window as opposed to making the viewing of the rate meal window optional; the customer is not required to rate their meal, however having the rate meal window viewing automated encourages them to make a rating, which in turn helps the restaurant improve customer satisfaction.

The new and improved design rewards regular customers with a faster and more efficient interface, but at the same time, has optimizations to make it easier for the first time users who choose to use our application. Overall, the interface is very simple and user friendly. Each interface was constructed specifically to reduce the amount of user effort. Which, in the end, creates an overall easy to use, all-in-one application that will appeal to many users, employees, and managers across the restaurant industry.

6. Design of Tests

6.1 Unit Testing

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

TC - 1: Tests login functionality and accuracy

TC - 2: Tests user ability to select dine in

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

TC - 4: Tests user ability to reserve table before hand

TC - 5: Tests menu viewability

TC - 6: Tests ability to select an available table

TC - 7: Tests that user sat at the correct table by scanning QR

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

TC - 9: Tests users ability to rate food

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

TC-11: Tests users ability to register for an account

TC-12: Tests estimation time for food arrival

Test-Case Identifier: TC - 1 **Use Case Tested: UC - 10** Pass/Fail Criteria: Test passes if the user is able to login to their account with their combination of email and password. Test fails if the user is able to login to their account with the wrong username or password. Input Data: email, password		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Enter a username and password combination that is valid for customers and select login. Step 2: Enter a username and password combination that is valid for manager and select login. Step 3: Enter a username and password combination that is valid	The app sees that the enter credentials are valid and takes the user to the user's main page. The app sees that the entered credentials are valid for manager and displays the manager main page.	The app displays the customer home page. The app displays the main page for the manager.

for chef and select login. Step 4: Enter a username and password combination that is invalid and select login	The app takes the user to the main page for the chef after confirming that information is valid. The app stays at the main page and displays an error message.	The app displays the main page for the chef App states that the login was not successful.

Test-Case Identifier: TC - 2 **Use Case Tested: UC - 1** Pass/Fail Criteria: The test passes if the available tables screen is shown on button click. The test fails if no screen is loaded or the wrong screen is loaded. Input Data: Button selection		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Click on Dine In button.	The available table screen is loaded for the user to select from.	The available table screen is loaded for the user to select from.

Test-Case Identifier: TC - 3 **Use Case Tested: UC - 2** Pass/Fail Criteria: Test passes if the take out menu button is clicked and it opens up the takeout menu options for the customers to pick from. Customer can then click to add items and click confirm to confirm order. Test will fail if the takeout menu does not open or the items are not added. Input Data: Button click		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Click on takeout button. Step 2: Add item and check that it appears on the ordered list Step 3: Delete item from the list and check ordered list is properly updated Step 4: Confirm purchase and click order.	Takeout menu screen opens. Ordered item appears on ordered list with quantity and price displayed along the side. Items are deducted in quantity and the price is adjusted accordingly. Order is sent to the queue for the chef to select and make.	Takeout menu screen opens Ordered item appears on the list with price and quantity shown. Items are deducted in quantity and the price is adjusted accordingly. This has yet to be implemented

Test-Case Identifier: TC - 4

Use Case Tested: UC - 3 and UC -5

Pass/Fail Criteria: This test will pass if a customer can select an unreserved table. This test will fail if the customer is able to reserve a table that is previously reserved or if the table the customer reserved is not marked as reserved.

Input Data: Button selection

Test Procedure:	**Expected Result:**	**Actual Result:**
Test 1: Select time for which the table is to be reserved.	Only tables that are available during this are shown as available while the rest are marked as taken.	All tables in the restaurant is shown without any distinctions.
Test 2: Select an available table to reserve in advance.	Table is set to be reserved for time the user has chosen.	This feature has yet to be implemented.
Test 3: Select a table that is marked as reserved or taken.	Table cannot be selected by the user.	This feature has yet to be implemented.
Test 4: Select a table for the current time (not reserving table)	Table screen is taken to the QR scan code page to confirm that the user is sitting at the selected table.	The screen with the QR code confirmation is loaded.

Test-Case Identifier: TC - 5

Use Case Tested: UC - 4

Pass/Fail Criteria: This test will pass if menu items are all seen but cannot be selected. The test will fail if the item can be ordered from the view menu or if the menu is not displayed correctly.

Input Data: Button selection

Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Click on View Menu button.	A screen with the full menu appears	A screen with a menu appears
Step 2: Select items on the menu.	No items in the menu are selectable.	Items can be selected and a purchase can be made which links to confirm payment page.

Test-Case Identifier: TC - 6

Use Case Tested: UC - 6

Pass/Fail Criteria: The test will pass if the camera is able to be opened, camera detects the correct QR and then loads the next screen. The test will fail if the camera cannot open or if the QR cannot read the correct QR code or reads the wrong code as correct.

Input Data: Button Selection

Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Load the scan QR page on the app.	The camera opens within the app and is ready to scan the QR code.	If the app is not allowed to use the camera, the user must first allow access and then manually reload the page. Otherwise the camera opens properly.
Step 2: Steady the camera over the QR code of the table the user has selected.	The system recognizes that the QR code corresponds with the table and loads the next screen.	The system recognizes the QR code and makes a Scan QR button visible. Clicking on this button loads the next page.
Step 3: Steady the camera over the QR code of a table that does not correspond with the table that is selected.	The system does not confirm the QR code and prompts the user to go to the correct table.	The system does not confirm the QR code and prompts the user to go to the correct table.

Test-Case Identifier: TC - 7

Use Case Tested: UC - 7

Pass/Fail Criteria: The test will pass if the customer can pay with credit or cash and the order is sent to the chef afterwards. The test will fail if customer cannot pay or if the chef does not receive the order.

Input Data: Button selection and numerical input for credit card

Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Select to pay with cash.	The system continues and someone will come later to retrieve the cash.	The system continues but there is nothing in place to notify a worker to come and collect the cash.

Step 2: Select to pay by card.	The app will provide fields for the person to type credit card information in and enter.	This feature has not been implemented.
Step 3: Select the submit payment button.	The app will send the order the customer has paid for to the chef's queue.	This feature has not been implemented.

Test-Case Identifier: TC - 8

Use Case Tested: UC - 8

Pass/Fail Criteria: The test will pass if the review the user is able to review and that review is stored for the user to use later on. The test will pass if the review does not work properly or if the review is not stored.

Input Data: User Star Selection

Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Load the confirm payment page.	Foods that are ordered should appear along with a star rating system next to each type of food the user has ordered.	Foods that are ordered appear along with a star rating system next to each type of food the user has ordered.
Step 2: Select a number of stars corresponding to what to rate the food.	The appropriate number of stars can be selected out of the maximum number of stars there are.	The stars cannot be selected or changed.
Step 3: Submit the review	The submitted reviews are stored in the user profile.	This feature has not been implemented yet.
Step 4: Load the menu page	The menu should be appear with a star rating system next to each item.	The menu appears with a star rating system next to each item.
Step 5: Select a number of stars corresponding to what to rate the food.	The appropriate number of stars can be selected out of the maximum number of stars there are.	The appropriate number of stars can be selected out of the maximum number of stars there are.
Step 6: Order food which are rated.	The rated objects should be stored in the user profile.	This feature has not been implemented yet.

Could not be implemented

Test-Case Identifier: TC - 9 **Use Case Tested: UC - 9** Pass/Fail Criteria: The test will pass if the user can favorite food items and these food items appear in their profile for the next time they order. If this does not happen then the test fails. Input Data: Button selection		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Load the confirm payment page.	Foods that are ordered should appear along with a favorite button next to each type of food the user has ordered.	
Step 2: Select the favorite button	The food item next to the selected button is shown as favorited.	
Step 3: Submit the review	The food items the user clicked the favorite button on are stored in the user profile.	These features have not been implemented.
Step 4: Load the menu page	The menu should be appear with a favorite button next to each item.	
Step 5: Select a favorite button next to the food item.	The button displays that the food item is favorited.	
Step 6: Select confirm payment page.	The food items the user clicked the favorite button on is selected.	

Test-Case Identifier: TC - 11 **Use Case Tested: UC - 11** Pass/Fail Criteria: The test will pass if the manager can view the reports on clicking view reports button. Test will fail if this does not occur. Input Data: Button selection		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Select category inventory	Display the current inventory in a graph.	An older version of the inventory is displayed.

Step 2: Login as a customer and order an item. Log back in as a manager and select category inventory.	The inventory should be updated with the ordered item using up some inventory.	The same version of the inventory as before is displayed.

Could not implement

Test-Case Identifier: TC - 12 **Use Case Tested: UC - 12** Pass/Fail Criteria: Test passes if the user is able to register account when the user enters a username that is not in use already and password that is at least six characters long. Test fails if the user users a username and password combination that is already in use to register. Input Data: email and password		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Type in username that is already in use and an invalid password Step 2: Type in new email address and valid password	System will not allow the user to register an account. System will ask user to choose a username that not in use or a valid password. System will notify user that a new account has been created to indicate a successful registration; access to user enabled to use other features	These features have not yet been implemented.

Test-Case Identifier: TC - 13 **Use Case Tested: UC - 13** Pass/Fail Criteria: The test passes if the chef is able to update the wait time for food arrival. The test fails if this is not so. Input Data: Wait time		
Test Procedure:	**Expected Result:**	**Actual Result:**
Step 1: Login to Chef account and navigate to the Order Queue.	The Order queue is displayed along with the percentage of each dish the chef has completed.	This has yet to be implemented.

Step 2: Update the status of one of the dishes.	The wait time for the order should now decrease.	

6.2 Integration Testing

- Login Screen to...
    - Manager UI: cannot go back to login screen from this UI
        - Can edit...
            - Employee list: currently no functionality
            - Item list: currently no functionality
    - Chef UI: cannot go back to login screen from this UI
        - Views orders of tables
        - Can set an order as ‘started’; currently no functionality
        - Can set an order as ‘completed’; currently no functionality
    - Busboy UI: cannot go back to login screen from this UI
        - Can view the statuses across all tables
            - Can set tables to ‘ready’ after being cleaned; currently no functionality
    - Server UI: cannot go back to login screen from this UI
        - View order status
            - Can set order as ‘paid’; currently no functionality
            - Can set order as ‘delivered’; currently no functionality
    - Customer UI: can logout and return to login screen
        - View menu
        - Takeout selection: goes straight to ‘Menu’
        - Dine In Selection
            - Select Time; currently no functionality
            - Select Table; stores the desired table and sends the selection to the next window
            - Scan QR Code; will only allow the customer to proceed to the ‘Menu’ screen if they scan the QR Code at their selected table
        - Menu: either accessed via the ‘Dine In’ or ‘Takeout’ options
            - Select food to be ordered
            - After selection, the user can either pay with ‘cash’ or ‘credit’; currently either option leads the user to the ‘Submit Rating’ window
            - Submit Rating: allows user to rate 0-5 stars on what they ordered; currently the user can rate anything on the menu despite whether

or not they ordered it, and currently the inputted ratings are not stored into a database and thus currently have no functionality
- After rating, the user is sent back to the Costumer UI page

Integration Tests:

1. Q: Can a user return to the previous window from any window on the app?
   A: No; neither manager, chef, busboy, nor server accounts can currently log off and return to the login page. As of every other window, yes.

2. Q: After a customer scans the QR code of their selected table, if they accidently go back to the QR scanner page, will the app recall what their selected table was if they rescan the QR code?
   A: Currently yes, however in the future it should be implemented to not allow the customer to return to the QR scanner after having already sat at their table; it may cause glitches in the system

3. Login:
   1. Enter the email of your account
   2. Enter your associated password
   3. Press the 'LOGIN' button

4. Create Account:
   1. From the 'LOGIN' page, press "No account yet? Create one"
   2. Enter your name
   3. Enter your address
   4. Enter your email
   5. Enter your phone number
   6. Enter your password
   7. Re-enter your password for confirmation
   8. Press 'CREATE ACCOUNT'

5. Select 'TAKEOUT' as a Customer:
   1. Logic as a customer (go through integration test 3)
   2. Select 'TAKEOUT'
   3. Press the plus button on each item desired to be ordered the number of times equivalent to the desired quantity of each item
   4. Press 'CONFIRM ORDER'
   5. Either select 'PAY WITH CASH' or 'PAY WITH CREDIT' depending on one's payment preferences

6. Optional: To provide feedback, rate each ordered item from 0-5 stars
7. Press ‘SUBMIT RATING’
8. Select ‘Yes’ when asked “Are you sure?”

6. Select ‘DINE IN’ as a customer:
    1. Login as a customer (go through integration test 3)
    2. Select ‘DINE IN’
    3. Select a desired time
    4. Select a desired table
    5. Select ‘SCAN BARCODE’
    6. Scan the QR code on the desired table
    7. Select ‘HAVE A SEAT!’
    8. Follow steps 3-8 of integration test 5

7. Logout as a Customer:
    1. Login as a customer (go through integration test 3)
    2. Select ‘LOGOUT’

8. View Menu as a Customer:
    1. Login as a customer (go through integration test 3)
    2. Select ‘VIEW MENU’

9. Send Message to Employee as Manager
    1. Login as a manager (go through integration test 3)
    2. Select the message icon
    3. Select what employee to send a message to in the ‘To’ section
    4. Select the pre-generated message to send in the ‘Message’ section
    5. Select ‘SEND’

10. Edit Item as Manager
    1. Login as a manager (go through integration test 3)
    2. Select ‘EDIT ITEM’
    3. Enter the name of the item
    4. Enter the price of the item
    5. Enter the time of the item
    6. Either select ‘ADD’ or ‘DELETE’

11. Edit Employee as Manager
    1. Login as a manager (go through integration test 3)

2. Select ‘EDIT Employee’
3. Enter the name of the employee
4. Enter the salary of the employee
5. Enter the SSN of the employee
6. Either select ‘ADD’ or ‘DELETE’

12. Start Order as Chef
    1. Login as a chef (go through integration test 3)
    2. Select table of order
    3. Select ‘START’

13. Complete Order as Chef
    1. Login as a chef (go through integration test 3)
    2. Select table of order
    3. Select ‘COMPLETE’

14. Clean Table as Busboy
    1. Login as a busboy(go through integration test 3)
    2. Select table to clean
    3. Select ‘READY’

15. Set Status of Order as Server
    1. Login as a server (go through integration test 3)
    2. Select order
    3. Select either ‘PAID’ or ‘DELIVERED’

7. Project Management & Plan of Work

7.1 Merging the Contributions from Individual Team Members

To develop the final copy of the report all team members were independently developing their assigned tasks of the report in one shared Google document for each part of the report. Once the first two parts of each report was completed, they were merged into the full report which was shared by all members in a Google document. The issues we encountered were checking to see if the formatting was consistent throughout the document and the tables were formatted correctly. To format the document we made sure the table of contents accurately matched the page numbers and the headings of the parts of the project. We also reviewed all figure and table names and numbers to verify that everything was in order.

7.2 Project Coordination & Progress Report

UC-1: Dine-In, UC-4: View Menu, UC-5: Table Selection, UC-6: QR Scan, UC-7: Payment, UC-8: Rate Food, UC-10: Login, UC-11: Generate Report, UC-12: Register, and UC-13: Estimate Time, are currently all implemented. However, some use cases were more focused on than others, so although they are functional, they are not fully implemented currently. For example, UC-6: QR Scan if fully functional and works as intended, but UC-13: Estimate Time is implemented in a way such that the chef is able to notify the user of the start time and the end time, but in its current state does not estimate the time beside the time listed in the menu. We are currently working on implementing UC-2: Take Out, UC-3: Reservation, and UC-9: Favorite Food.

7.3 Plan of Work

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

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

Table Alerts - method of alerting busboys that a table has been recently vacated so that they can clean it up.	10/7/18	10/28/18
Account Interface Coordinator - depending on which account logs in (customer, waiter, or manager), take the user to the correct portal for a seamless user experience	10/21/18	11/18/18

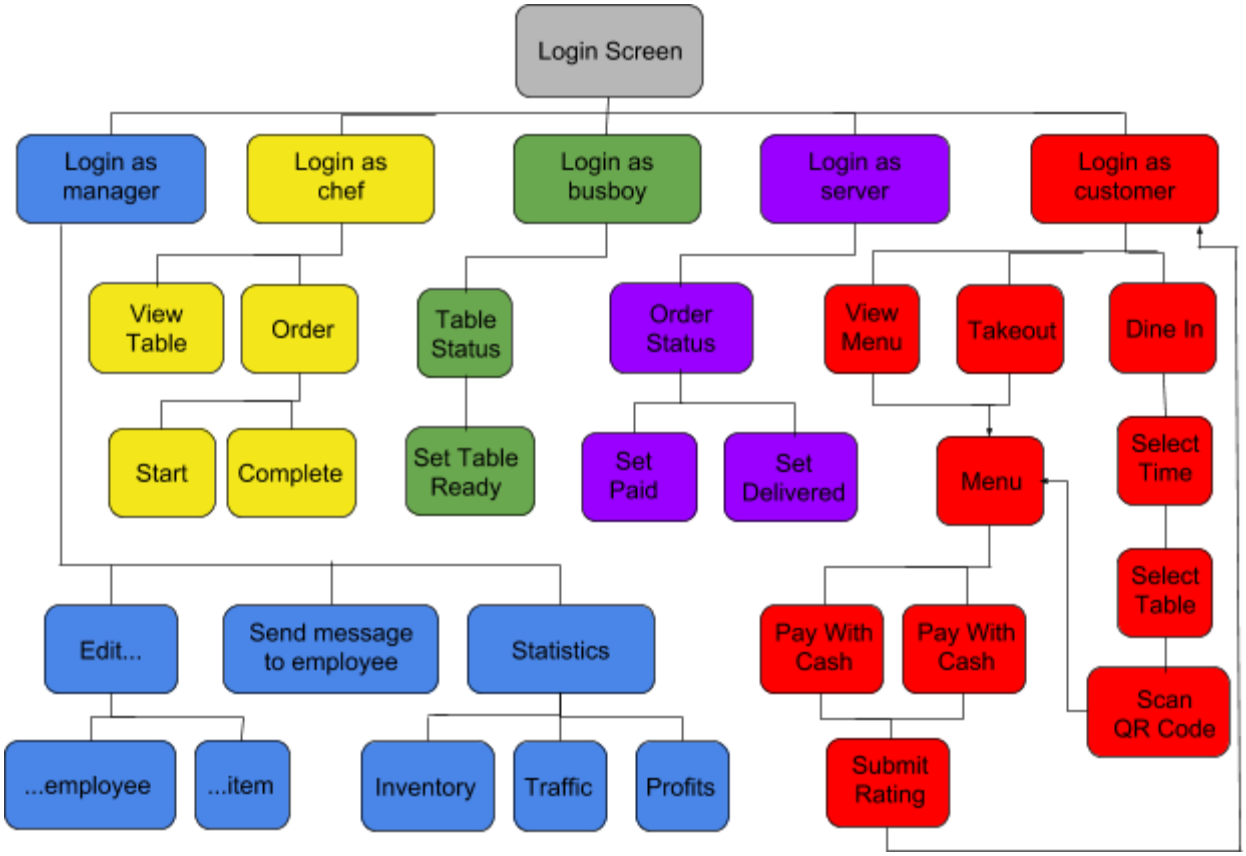
7.4 Breakdown of Responsibilities

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

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

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

8. Cyclomatic Complexity



Number of Nodes: n = 33
Number of Edges: e = 36
Number of Exit Points: p = 1 (Although there seems to be 11 exit points, the user simply cannot progress any further in the application, but can go backwards, so we chose a value of 1 here to represent the end of the application)

Cyclomatic Complexity: V = e - n + p + 1 = 36 - 33 + 1 + 1 = 5

9. References

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

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

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

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

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

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

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