# Why W8

## Restaurant Automation

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

**Group 4**

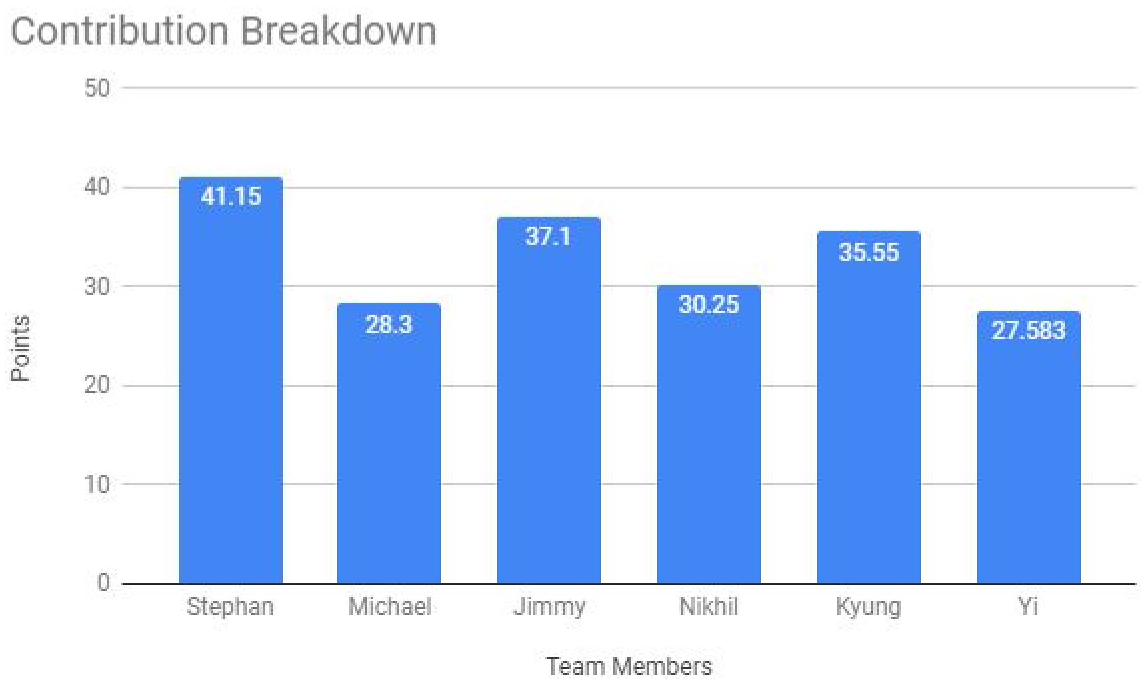
**Full Report**

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

| | Project Category | | Team Member Name | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Stephan | Michael | Jimmy | Nikhil | Kyungsuk | Yi |
| Responsibility Levels | **Summary of Changes** *5 Points* | | | | 100% | | | |
| | **Sec 1. Customer Statement of Requirements** *6 Points* | | | | 100% | | | |
| | **Sec 2. Glossary of Terms** *4 points* | | | 50% | | | 20% | 30% |
| | **Sec 3. System Requirements** | **Functional Requirements** *2 Points* | 100% | | | | | |
| | | **Nonfunctional Requirements** *2 Points* | | | | | 100% | |
| | | **UI Requirements** *2 Points* | | | | 100% | | |
| | **Sec 4. Functional Requirements Specification** | **Stakeholders, Actors, Goals** *2 Points* | 30% | | 50% | | 20% | |
| | | **Use Case Casual Descriptions** *8 Points* | 50% | 40% | | | | 10% |
| | | **Use Case Diagram** *5 Points* | 100% | | | | | |
| | | **Use Case Full Descriptions** *10 Points* | | | | | 100% | |
| | | **System Sequence Diagrams** *5 Points* | | 15% | | 15% | | 70% |
| | **Sec 5. Effort Estimation Using UCP** *4 Points* | | 100% | | | | | |
| | **Sec 6. Domain** | **Concepts** | 20% | 15% | | 15% | 50% | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **Analysis** | ***10 Points*** | | | | | | |
| | | **Associations** ***5 Points*** | | 25% | | | | 75% |
| | | **Attributes** ***5 Points*** | | 25% | 75% | | | |
| | | **Contracts** ***5 Points*** | | | | 50% | | 50% |
| | **Sec 7. Interaction Diagrams** | **UML Diagrams** ***10 Points*** | 66% | 34% | | | | |
| | | **Prose Description of Diagrams** ***10 Points*** | 50% | | | | 50% | |
| | | **Alt. Solution Descriptions** ***10 Points*** | 50% | | | | | 50% |
| | | **Design Patterns** ***10 Points*** | | | 50% | | 50% | |
| | **Sec 8. Class Diagram and Interface Specification** | **Class Diagram & Description** ***5 points*** | | | | | | 100% |
| | | **Signatures & Traceability Matrix** ***5 points*** | | 50% | | | | 50% |
| | | **OCL Contract Specification** ***10 Points*** | | 33% | | 33% | | 33% |
| | **Sec. 9 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. 10 Alg's & Data Structures** ***4 points*** | | | | 70% | 30% | | |
| | **Sec. 11 User Interface** | **Appearance** ***6 points*** | | 40% | | 60% | | |
| | | **Prose Description** ***5 points*** | | | 100% | | | |
| | **Sec. 12 Testing Design** ***12 points*** | | | 50% | | 50% | | |
| | **Sec 13. History of Work** ***5 Points*** | | | | | 100% | | |
| | **Project Management 13 Points** | | 40% | | 40% | | 20% | |



# Table of Contents

## Summary of Changes

- Updated Functional Requirements
- Updated User Statements and diagrams
- Updated more images of User Interface
- Added description to each use case traceability matrix
- Added Estimation using use case points
- Updated Interaction diagrams
- Added new customer portal features within application
- Added Object Constraint Language Contracts
- Added Design Patterns
- Updated History of Work and added Future Work

# 1. Customer Statement of Requirements

## 1.1 Problem Statement

**Actor - Host/Hostess:**

By being the first person the customer sees, after coming through the front door, there is a paramount of importance placed on hospitality that is served by the host. As a host, I have to try to make the customer feel welcome, and have them seated in a calm and collected manner. However, this feat can be more difficult than it sounds when factoring a large influx of customers and limited seating, along with the confusion of people moving about. If there was a program that could keep track of the open seats, and where they were located, I could serve newly entering customers much faster. What would be even better is if the customer had reserved a seat for themselves prior to arrival. That way I wouldn't even have to search for an open seat, and all I would have to do is confirm that they did indeed reserve a seat and guide them to it.

The Why W8 solution: Why W8 allows the customer to select their own table before even stepping foot inside the restaurant. The process is simple, the customer will select their desired time and table. If available, the customer will reserve said table at said time. Upon arrival the customer will simply give their name and the hostess will see when and where to seat them.

**Actor - Manager:**

Running a business is not easy by any means. If there is anything that could help minimize the amount I have to spread myself across tasks, and help me keep the business profitable, then I'm all ears. I was personally thinking about something that can help the restaurant keep track of the flow of customers, and perhaps what they ordered. That way I know when to keep extra staff on hand, and what to order more of in order to keep a stocked inventory.

That being said, there is also the issue of my having to keep tabs on my employee, to make sure that they are working in a timely manner. I hear that you have some kind of table tracker that you might be able to implement. Well, if you could add in something that keeps track of how long it takes the busboy to clean off the tables and how long it takes for the guests to be seated along with being served, that would make my day. Saves me some time from having to look over the camera records to make sure there isn't any major slacking.

However, while all of the aforementioned is nice and dandy, I really don't want to have to juggle around another program. Sometimes I get so lost when I have to sift through so many different programs in order to manage payrolls, inventory, shift management, expenditures, et cetera. If there is any way you lot can combine all of these tools for me in one compact program, I would be eternally grateful.

The Why W8 solution: The overall design of Why W8 keeps track of practically every action made in the restaurant by both the employees and the customers. This makes relaying data back to the management much easier. For example, Why W8 will be able to recognize how many orders were placed, how many dishes are ordered per order, how much of the inventory was used per order, and how many employees are on staff between a given interval of time. With knowledge of this information, Why W8 can easily create charts for management to keep track of the day to day running of the restaurant.

**Actor - Busboy:**

Most people think living the life of a busboy is easy - and it is, for the most part. But, there are some days that just have me turned about, with having so many customers and all. With the hostess asking you for an informative layout of the tables in the restaurant, I was wondering if you could put in some information that shows which tables have to be cleared off and cleaned. You know, that way I could quickly come over and do whatever has to be done to make the place spiffy for the next customer.

The Why W8 solution: Why W8 offers an interactive display which will keep track of every step along the way of a successful dining experience. With that being said, the Busboy(s) will know when a customer has left the restaurant, which will trigger the Busboy(s) to clean the now unoccupied table, once the table is clean the Busboy(s) will be able to set the table as a clean “Available” table.

**Actor - Customer:**

I’m really picky about my food - I prefer knowing that the food I’m about to get is near perfection, at least to most people. Some kind of rating system might help me with decisions regarding just that. In consideration of the foregoing, I also want to know what exactly is in the food, in the case of allergies and for calorie counting purposes. All this talk about food has made me hungry, so I’ll probably call somewhere for food; actually, that gives me another idea! If there was a digital menu, I wouldn’t have to go through the hassle of having to speak with someone and waste time trying to understand them as they try to understand me - my phone isn’t especially good, so this is more of a problem than you might think. And perhaps if I could order ahead of time, that would cut down my waiting to get the food - fantastic! Speaking of waiting, if the menu also displayed how long an item would take to make, then I would know when to arrive at the restaurant to pick up the food when it’s nice and hot. Although, there are also occasions when I like to just eat at the restaurant, but the line goes through the door sometimes! If only there was a way everyone already knew where to sit and could just get to it. What if there was a way to combine some of those ideas! Hey, let’s say I order off of this menu, then maybe I can also reserve a seat for myself, and pay the bill while I’m eating. That way, I already have a seat and food ready to go - presto, time saved.

The Why W8 solution: Why W8 offers a digital touch menu with numerous features to ensure customer satisfaction. An integrated rating system will show the customer what previous customers thought about a specific dish, this will guide the customer into selecting a dish that will help them leaving satisfied. The menu will also display what ingredients are in each dish which will avoid any allergy issues. To remove the uncertainty of a wait time, Why W8 has the time it will take for each dish to be made displayed on the menu. Another great feature Why W8 provides is a table selection option. This option allows the customer to reserve a specific table at a specific time.

**Actor - Chef:**

Day in and day out, I work in the kitchen - being a chef isn't terrible, but boy is it stressful. Having to make sure orders are filled out in a timely fashion and keeping track of dishes that have a few modifications requested to them by the customer really does a number on me sometimes. Not to mention, sometimes what the waiter writes down looks like downright chicken scratch, it all just makes the job harder than it has to be. So, what I would like to get from you guys is something that will help me read incoming orders clearly, with any possible modifications done to them in a different color. That way, I don't have to rely on the waiter's chicken scratch, and I can clearly see what is different about this meal from the standard. Next on my list would be something to help tell the waiter that the food is ready to pick up - that's another one of my pet peeves; I have a fresh meal ready for a customer and it just sits around for 5 minutes, taking up space on the counter for more dishes and getting cold all the while.

Now since that's all settled, there is one more idea that I have, but I'm not sure if it's worth the effort. I was thinking about having the incoming orders laid out in such a way so that I could tell what ingredients are common among them. That way, I could prepare larger batches of whatever it is that the meals require, say sautéed mushrooms for instance. If an order for a burger with mushrooms came in, and then an order of creamy mushroom on fettuccine, I would like to be able to see that they share sautéed mushrooms in common, so I could make more in one go rather than having two sautéing sessions. But, that's just my own little idea - I'm not even 100% sure it would work the way I think it could.

The Why W8 solution: Because Why W8 is a connected application, when the customer places an order it immediately goes into a queue until it is ready to be made. Once the order is ready to be made the Chef will be able to click “Start Order.” This will bring up the recipe for each dish. Once the Chef completes the order they will simply click “Order Complete” which will signal the server to pick up the food.

**<u>Actor - Waiter/Waitress:</u>**

Man do I hate when the chef yells at me - he gets so agitated when I don't pick up stuff from him in the time he wants me to. It's not like I diddle-dally my way around the restaurant. I have customers to attend to as well, taking their orders and, well, waiting on them. I hear you guys are already implementing some sort of order ahead system - that is absolutely fantastic, since it will take a bit of the load off of me. I was just wondering if there was some way you can have me fill out orders digitally, and I would just send them off to the kitchen without even my having to go there. Also, if you guys could have a little list for me indicating which meal is done, and to which table it should go to, that would be fabulous. In fact, if you could have some way of indicating which table has yet to be served, or is in the middle of eating, or is wanting to pay their bill, or ... well, you get the idea. Something that tells me the status of each table, so I know where to run to next that too would be extremely helpful. Maybe with all of these things done, the chef would no longer have a reason to yell at me.

The Why W8 solution: Because of the integration of all components of the app, once the customer orders their food it will be relayed directly to the kitchen eliminating the middleman where things can get lost in translation. Once an order is ready, a notification will alert the server prompting them to deliver the food to the corresponding table. Once the food is delivered the server simple presses "Delivered" for the corresponding table which will remove it from the list.

**<u>What Makes Why W8 Better?</u>**

Taking a look at [1], we see that this group's proposed solution involves a tablet at the front of the restaurant where they input their information. This can cause lines to form as people wait for others to finish in front of them, defeating the purpose of trying to speed up the dining process. Why W8 addresses this problem by allowing customers to download our application directly to their phone and select a table for a specific group size without having to use a tablet. Of course, there will be backup tablets for the special occasions where customers do not own a smartphone.

Further, in [1] the payment is done through the tablet when the customers finish their meals, which we believe can cause some dishonest customers to avoid paying their bill. We address this by having the customer pay on their phone as soon as their order is placed, otherwise, the order does not go through in our system for the chef to cook. We also added QR codes on the tables to allow customers to confirm their table when they enter the restaurant, as well as scan the code when they are leaving the restaurant to let our system know that the table has opened up. If for some reason the customers to do not scan the QR code upon leaving, the waiter will be able to scan it to override the system.

Why W8 also has its own features not seen in other apps, such as the rating and favorites system. Customers are given the option to create an account on the app so that they can keep track of their "favorite" foods for easy re-order and also rate foods that they have tried. This gives all customers, even those who do not have an account, the ability to view the top rated foods as well as the most favorited foods. These appear at the top of the menu when the customers get to the ordering screen.

# 2. Glossary of Terms

**Admin**
The restaurant owner/manager utilizing the reports generated by the application.

**Application (App)**
A software program designed to perform specific functions for a user.

**Busboy**
A busboy is a person who works in the restaurant and catering industry clearing tables, taking dirty dishes to the dishwasher, setting tables. In this report, a busboy would get an alert when the customers finished.

**Chef**
A chef is a trained professional cook who is proficient in food preparation, often focusing on a particular cuisine. In this report, the chef is the person prepares the food after the customers confirm their order in our app.

**Cross-Platform**

An application that can be used on multiple types of devices, such as varying phone models.

**Customer**
A person or organization that buys goods or services from a store or business. In this report, the customer is the person utilizing the application, and not the one requesting the proposed system.

**Customer Account**
A user account to access the application as a customer that allows them to reserve tables and stores their preferred meals, ratings, etcetera.

**Customer Service**
Customer service is the process of ensuring customer satisfaction with a product or service. In this report, customer can contact the restaurant or give some comments or complaint  in the interface of customer service.

**Database**
A structured set of data stored on the Why W8 network, certain parts of which are accessible to certain user accounts.

**Dine-In**
To order and eat food within a restaurant.

**Favorite Rating**
A score that reflect customers' satisfaction. The application shall allow the customer to rate and "favorite" meals. "Favorite" meals will be highlighted in the customer's account for future orders.

**Floor Layout**
Shows all tables in the restaurant along with their respective status for the customer's viewing purposes.

**Guest Account**
An account almost identical to a customer account minus the features of storing information such as favorite meals and personal ratings.

**Host**
Assigns seats to people who come to the restaurant, especially those who wish to not use the app. Changes the status of tables from "Occupied" to "Vacant".

**Menu**
In this report, a menu is a list of food with pictures and favorite rates. Customers can view or order food in the menu.

**Manager**

Manages inventory, payroll, employee list and charts, customer’s bills to override any issues, and the log-in interface to prevent unauthorized access of admin panel.

**Order Queue**
A list of orders that are placed in first in, first out order (FIFO). The orders are sent to the chef’s PC, where the chef can then view and prepare the meals. [6]

**QR Code**
QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix barcode (or two-dimensional barcode. In this report, the QR code is used for customers confirming their table number. [5]

**Rating**
A position or standing of something, such as how popular some meal is amongst customers determined by their feedback and stored on the Why W8 database.

**Reservation**
An arrangement to secure a table ahead of time.

**Restaurant System**
Customers can view all available tables and reserve a specific table at a specific time using the android application.

**Screen**
A specific window displayed on the Why W8 user interface that provides convenient functionality for the user.

**Tablet**
A portable thin computer that utilizes a touchscreen as its primary user interface.

**Take-Out**
Food that is sold at a restaurant and packaged so that the customer can eat it elsewhere.

**Tip**
A small percent of money to be given to the waiter/waitress.

**User Account**
A private location on a network server for an individual who utilizes the application to store information such as their username, password, etcetera.

**User Interface**

The visual aspect of the android application on the user's phone that allows user interaction with the system.

**Waiter**
A waiter is the person who works at a restaurant, attending customers—supplying them with food and drink as requested. In this report, waiters are working on delivering food to tables and serving for the special requirements.

**Walk-in Queue**
A list of table reservations that are placed in first in, first out order (FIFO). These reservations are made on the customer's end of our application, which is on their android phones.

# 3. System Requirements

## 3.1 Enumerated Functional Requirements

**Table 3.1** Functional Requirements

| Identifier | Priority | Requirement |
|---|---|---|
| REQ-1 | 5 | The application shall allow customers to choose an open table based on the restaurant's seating chart. Once a table is chosen, customers shall scan a QR Code at the selected table to confirm their seat. |
| REQ-2 | 5 | The application shall provide a menu once the table is confirmed, where the customer places their order. |
| REQ-3 | 5 | The application shall send the order to the chef queue to be cooked once the customer has paid for their meal. |
| REQ-4 | 4 | The application shall provide the option to make a table reservation at the restaurant. |
| REQ-5 | 3 | The application shall show the customer an approximation of the remaining wait time for their meal. |
| REQ-6 | 5 | The application shall alert the waiter when an order has been completed. The waiter will then be able to deliver the order to the designated table. |
| REQ-7 | 4 | The application shall allow the customer to choose a take-out option, where orders are placed and picked up. |
| REQ-8 | 2 | The application shall allow the customer to rate and "favorite" meal options. "Favorite" meals will be highlighted in the customer's account for future orders. |

| | | |
|---|---|---|
| REQ-9 | 1 | The application shall display to the admin food, customer, and inventory reports. Popularity of meals, number of customers, and remaining supplies will be available to view for the admin. |
| REQ-10 | 1 | The application shall allow the customer to view the menu options without choosing a table or placing an order. |
| REQ-11 | 4 | The application shall request the customer to scan the QR code found on the table once they are ready to leave. |
| REQ-12 | 4 | The application shall ask the customer if they would like to leave a tip once they have confirmed they are leaving the restaurant. There will be suggested tip percentages, as well as customer input. |
| REQ-13 | 5 | The application shall alert the busboy when a table is to be cleaned. When the customer scans the QR code at the table signalling that they are leaving the restaurant, the busboy will receive a notification. |
| REQ-14 | 2 | The application shall request a login, or ask the customer to use a guest account to proceed. |

In Section 1.1, we propose the solutions we believe to be the answer to the customer's statement of the problem. As we can see the host/hostess use case is satisfied by REQ-1, as the user is able to select a table on our app even without having to step inside the restaurant. The manager use case is satisfied by REQ-9, as the application gathers the data from the database of user ratings, inventory, favorite foods, etc. and displays detailed reports for the day. The busboy use case is satisfied by REQ-13, as the busboy receives a notification once the customer has left the restaurant that their table is dirty. The most important use case, for the customer, is satisfied by REQ-1, REQ-2, REQ-4, REQ-5, REQ-7, REQ-8, REQ-10, REQ-11, REQ-12 and REQ-13. All of these requirements give the customer the options to use our application to speed up the dining process by using our app for every step of their experience. The chef use case is satisfied by REQ-3, REQ-5, and REQ-6. Our app gives the chef the ability to start an order and give updates on the progress of a customer's meal. Finally, the waiter/waitress use case is satisfied by REQ-3, REQ-6, and REQ-12. The waiter/waitress does not have to write down every order or bring a receipt to the customer as everything is taken care of by our app. Find the acceptance test cases for each of the requirements below.

**Table 3.2** Acceptance Test Cases for Functional Requirements

| Identifier | Acceptance Test Case |
|---|---|
| REQ-1 | If the user chooses to dine-in, the seating chart will be shown. A customer selects an open table, designated by the color green. If a customer selects a green colored table, they will be asked to scan a QR code, otherwise, they will not be sent to the next screen if they choose a red colored table. |
| REQ-2 | Once the table is confirmed, the customer is directed to a QR scanner where they must scan the QR code given on their selected table. If the correct QR code is scanned, the |

| | |
|---|---|
| | menu screen shows where customers can place their order, otherwise, they will not be able to proceed. |
| REQ-3 | After placing their order, the customer will be asked to pay for their meal. If they make a payment, the order will be sent to the chef queue to be cooked, otherwise theri meal will remain on hold until a payment is made. |
| REQ-4 | At the beginning screen, there will be a reservation option. Clicking this will show the seating chart for available tables at certain times. Once a free table is chosen, our database will be updated with the reservation. |
| REQ-5 | After the chef starts cooking a customer’s meal, the status of the meal, from preparation to completion will be displayed for the customer to see. There will also be an approximate time remaining shown for the customer. |
| REQ-6 | When the chef updates the status of a meal as done, the waiter should receive an alert to pick up the meal and bring it to the appropriate table. |
| REQ-7 | At the beginning screen, there will be a take-out option. Choosing this will bring the customer to the order screen where they choose their meal. After receiving payment, the order is sent to the chef and is prepared, giving the customer an estimated time to pickup. |
| REQ-8 | Rating a food on a scale from 1 to 5 adds the rating to the database and updates the average rating for the food item. Favoriting a meal adds a heart next to the food and highlights the food the next time the customer uses the app. |
| REQ-9 | The application gathers data from the current day from the database and produces an accurate report based on that data. |
| REQ-10 | At the beginning screen, there will be a menu option. Choosing this allows the customer to view the menu without having to make an order. |
| REQ-11 | After making payment, the app will ask the customer when they are ready to leave. Once ready, they will be taken to a QR scanner. Scanning the QR code marks the table as dirty for the busboy and free the table in the seating chart. |
| REQ-12 | Scanning the QR code to confirm leaving should bring up a tip screen. The customer can choose to leave a tip by picking the suggested tips, or input their own tip. |
| REQ-13 | Scanning the QR code to confirm leaving should send a notification to the busboy that the table is dirty and should be cleaned. |
| REQ-14 | The application uses a customer account to keep track of ratings and favorite foods, but if the customer does not want an account, they can use a guest account. Picking either will bring them to the same menu order screen. |

## 3.2 Enumerated Nonfunctional Requirements

**Table 3.3** Nonfunctional Requirements [2]

| Identifier | Priority | Requirement |
|---|---|---|
| REQ-15 | 1 | The application should be cross-platform for maximum usability to the general audience |
| REQ-16 | 5 | The application should have a high mean time between failure (MTBF) and high mean time to recovery (MTTR) to ensure reliability |
| REQ-17 | 3 | The application should be intuitive and easy to use |
| REQ-18 | 3 | The application should look aesthetically pleasing and modern |
| REQ-19 | 4 | The application should have minimal transitions from each screen/action |
| REQ-20 | 5 | The application should run as a service to save resources on the user's device |
| REQ-21 | 3 | The application should be designed to handle an overestimated number of users for consistent throughput and availability |
| REQ-22 | 3 | The application should be designed to make updates and fixes easy to implement |

## 3.3 On-Screen Appearance Requirements

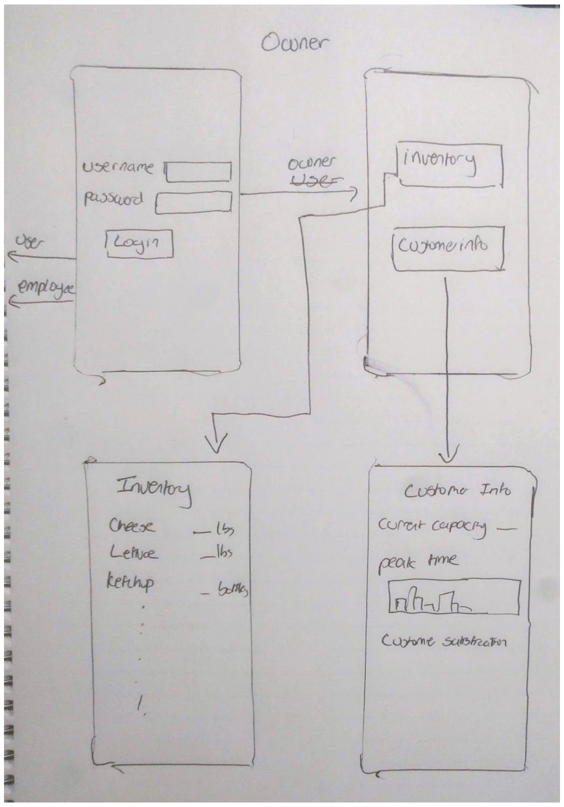


**Figure 3.1** User interface and path for the owner/admin

**Table 3.4** On-Screen Requirements for the Owner/Admin

| Identifier | Priority | Requirement |
|---|---|---|
| REQ-23 | 1 | Two text fields are used to take in values for username and password. A button is used to send the information to the system which determines the authorization of the user. |
| REQ-24 | 1 | On next screen, two buttons are used to determine type of information the owner wants to look at. Clicking on the button takes owner to the page linked with that button. |
| REQ-25 | 3 | On the Inventory screen, there is a list of items in stock which is updated in real time. Items running low are marked accordingly. |
| REQ-26 | 3 | On the Customer info screen, there are statistics such as the current number of people in the restaurant. There is also a graph showing the amount of customers for each hour. |

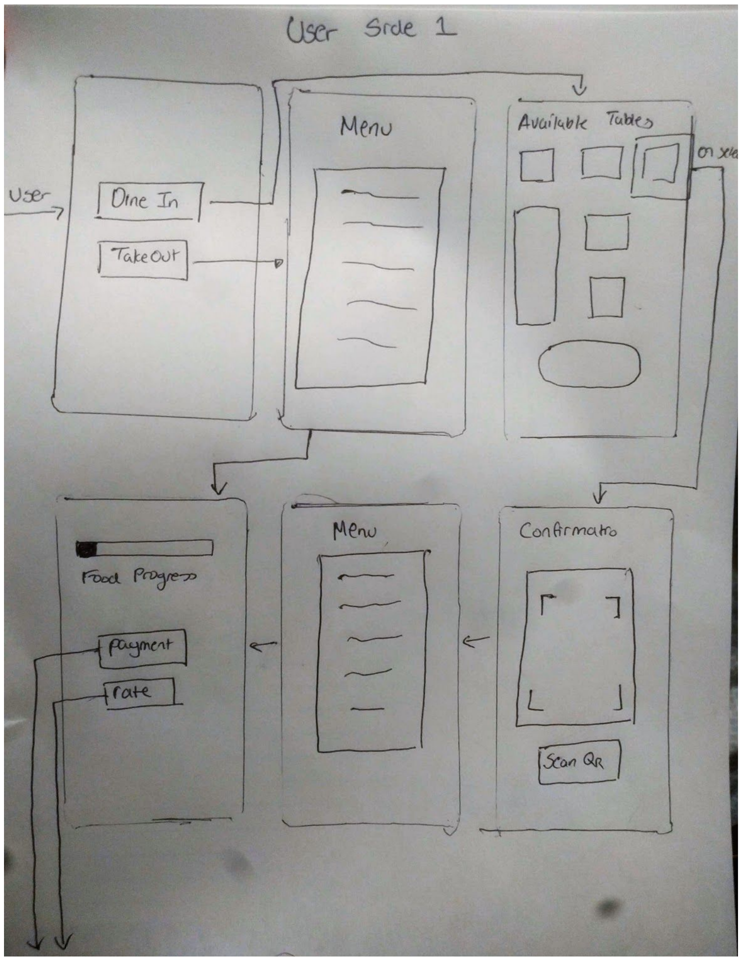


**Figure 3.2** User interface and path for the customer

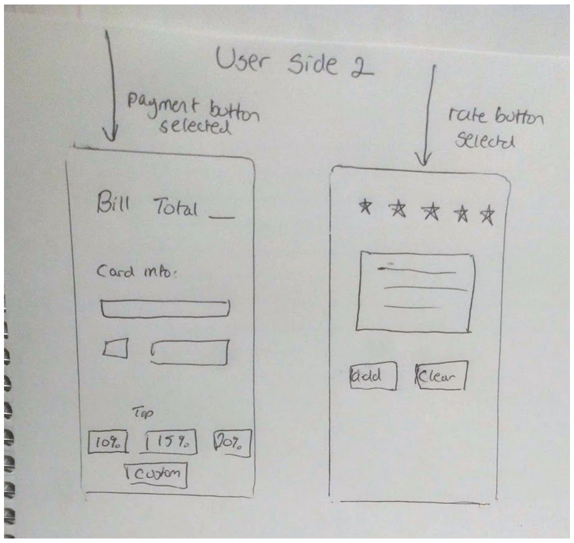


**Figure 3.2 (Cont.)** User interface and path for the customer

**Table 3.5** On-Screen Requirements for the Customer

| Identifier | Priority | Requirement |
|---|---|---|
| REQ-27 | 1 | Two buttons appear for customer to choose to dine in or take out. Button leads to corresponding pages. |
| REQ-28 | 1 | If the takeout button is clicked, the menu page appears for customers to select what they want to order. There is a list of food with specifications which show when an item is clicked. |
| REQ-29 | 1 | If the dine in button is clicked, the available tables appear. This shows the tables available labeled as green. The customer can choose the table they want and sit there. |
| REQ-30 | 2 | Upon table selection, confirmation page appears. This requires customers to scan the qr on the table to prove they are sitting in the correct table. There is an on screen camera and the scan QR button takes the picture. |
| REQ-31 | 1 | After confirmation, user is taken to menu page again as with take out. |
| REQ-32 | 5 | Once food items are selected, this page appears. Contains a food progress bar which is pre-programmed to a timer of expected time. There are also two buttons each linked to corresponding screens. |
| REQ-33 | 2 | If the payment button is selected, payment screen appears. This includes bill total that is printed as text. There are also text fields that take in card information. There are also buttons to add in tips to the total. |
| REQ-34 | 5 | If the rate button is selected, the review page appears. Here customers can leave a review for the food they ate. There is a text field for the customer to write the review on as well as a button to submit the review. |

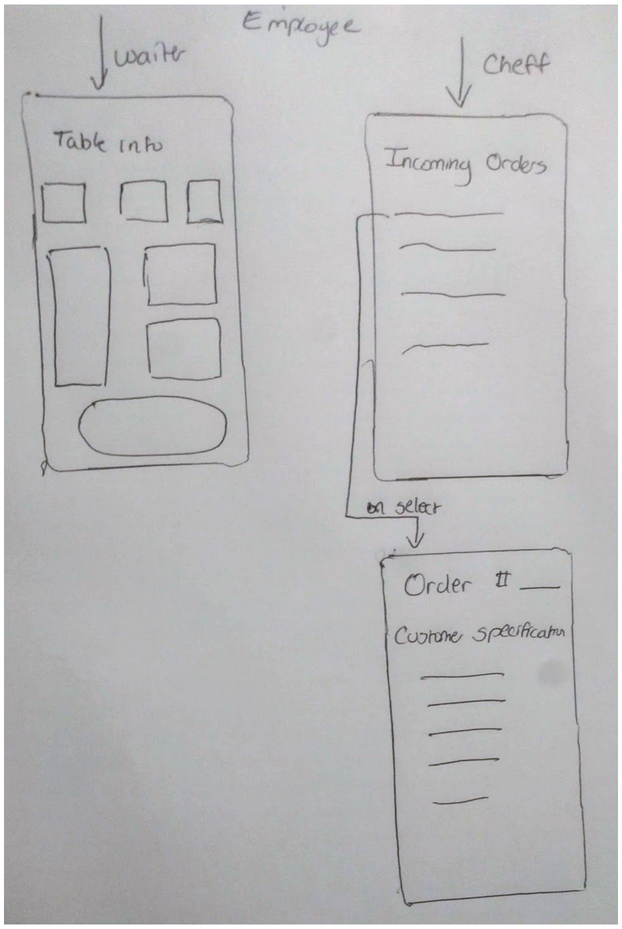


**Figure 3.3** User interface and path for the employee (general)

**Table 3.6** On-Screen Requirements for the Employee (General)

| Identifier | Priority | Requirement |
|---|---|---|
| REQ-35 | 1 | If the employee is a waiter, a screen with table info appears. These tables are marked according to duties the waiter must perform. If the table is dirty, the waiter needs to clean it. If the table has an order attached that is ready the waiter will be informed to deliver the food there. The waiter also has the ability to make tables available. |
| REQ-36 | 1 | If the employee is a chef, this screen with incoming orders appear. It contains a queue of what the customers ordered. The chef can select to get more information which brings up the next screen. |
| REQ-37 | 2 | If an order is expanded, it appears on this screen. This contains text information about the order details which includes customer specification. The chef can press a button to confirm he is currently making that dish. |

# 4. Functional Requirements Specification

## 4.1 Stakeholders

The following are the stakeholders who will have the most interest to design and run the system efficiently.

- ➢ Restaurant owner
- ➢ Employees i.e Manager, Waitress, Busboy, Host, Chef, and other employees
- ➢ Software Developers i.e front-end, back-end, full-stack developers/programmers

Alongside the restaurant owner seeing their business become more efficient, all of the employees simply would benefit from a streamlined process. On top of this, the customer, who is one of the most important aspects of the restaurant world, would make great use of the all-in-one application for their restaurant outings. Lastly, software developers on our team have a great interest in assuring our application runs smoothly and makes the whole restaurant process truly efficient. With this in mind, future software developers will be able to build and add more features in the future with the restaurant owner/manager in mind; specific customizations, niche features, etc.

## 4.2 Actors & Goals

**Table 4.1** Initiating Actors

| Actor | Role | Goal |
|---|---|---|
| Customer | The customer uses the application to either choose to order food for take-out, or dine in and select their table before entering the restaurant. The customer also places their order in the system. | The goal of the customer is to have an excellent experience with minimal wait times for all aspects of the restaurant dining process. |
| Guest | The guest is a special type of customer which does not have an account. The guest has the same role as the customer, without being able to favorite food. | The goal of the guest is to have an excellent experience with minimal wait times for all aspects of the restaurant dining process. |
| Manager | The manager uses the application to review the daily reports associated with their restaurant, including trends, ratings, inventory, customer peak times, etc. | The goal of the manager is to understand the important aspects of their restaurant as easily and quickly as possible. |

**Table 4.2** Participating Actors

| Actor | Role |
|---|---|

| | |
|---|---|
| Chef | The chef waits for the customer to place their order(s) if there are no orders in the queue. Otherwise, the chef begins cooking the top meal in the queue. The chef also updates the status of the customer’s meal so that an estimate for the remaining cooking time can be relayed to the customer. |
| Waiter/ Waitress | The waiter/waitress receives notifications from the database that a customer’s order is ready. Once notified, the waitress retrieves the customer’s meal and brings it to their table. |
| Busboy | The busboy receives notifications from the database that a customer has left their table, thus marking the table dirty. Once notified, the busboy goes to the appropriate table to clean up. |
| Database | The database records a customer’s orders, table selection, favorite foods, and ratings. |

## 4.3 Use Cases

### 4.3.1 Casual Description

The summary use cases are as follows:

UC-1: Dine-in - Allows the customer to dine inside the restaurant.
Derived from requirement REQ-1, REQ-2, REQ-3, REQ-5, REQ-6, REQ-8, REQ-11, REQ-12, and REQ-13.

UC-2: Take Out - Allows the customer to order food from their home for pick-up.
Derived from requirement REQ-7.

UC-3: Reservation - Allows the customer to reserve a table for a later date or time without having to call the restaurant.
Derived from requirement REQ-4.

UC-4: View Menu - Allows the customer to view the menu (mandatory sub use case, «include» from UC-1: Dine-in and UC-2: Take Out). Extension Point: the customer is able to rate food items. Extension Point: the customer is able to favorite a food item.
Derived from requirement REQ-2, REQ-8, and REQ-10.

UC-5: Table Selection - Allows the customer to select an open table in the seating chart (mandatory sub use case, «include» from UC-1: Dine-in and UC-3: Reservation).
Derived from requirement REQ-1 and REQ-4.

UC-6: QR Scan - Allows the customer to confirm their table by scanning the QR code associated with the table they chose in UC-5 (mandatory sub use case, «include» from UC-1: Dine-in).
Derived from requirement REQ-11

UC-7: Payment - Allows the customer to pay for their meal, after which the chef will begin cooking (mandatory sub use case, «include» from UC-1: Dine-in and UC-2: Take Out).
Derived from requirement REQ-3.

UC-8: Rate Food - Allows the customer to rate food they have eaten at the restaurant on a scale from 1 to 5 (optional sub use case, «extend» UC-4: View Menu).
Derived from requirement REQ-8.

UC-9: Favorite Food - Allows the customer to favorite a food on the menu so that it is highlighted the next time they order at the restaurant (optional sub use case, «extend» UC-4: View Menu).
Derived from requirement REQ-8.

UC-10: Login - Allows the customer to utilize the restaurant options and track their rated and favorited foods (mandatory sub use case, «include» from UC-1: Dine-in).
Derived from requirement REQ-14.

UC-11: Generate Report - Allows the manager to view important aspects about the restaurant generated from the daily customers in easy to follow charts and graphs.
Derived from requirement REQ-9.

UC-12: Register - Allows a guest to create an account so that he or she can access the application features.
Derived from requirement REQ-14.

UC-13: Estimate Time - Allows the chef to estimate the remaining cooking time for a customer’s meal.
Derived from requirement REQ-5.

It is important to note how the waiter/waitress and busboy are not given their own use cases because they are not initiating actors. They only participate in the actions which the customers initiate, and the database relays notifications to them. Dismissing notifications is not shown here for simplicity.

## 4.3.2 Use Case Diagram

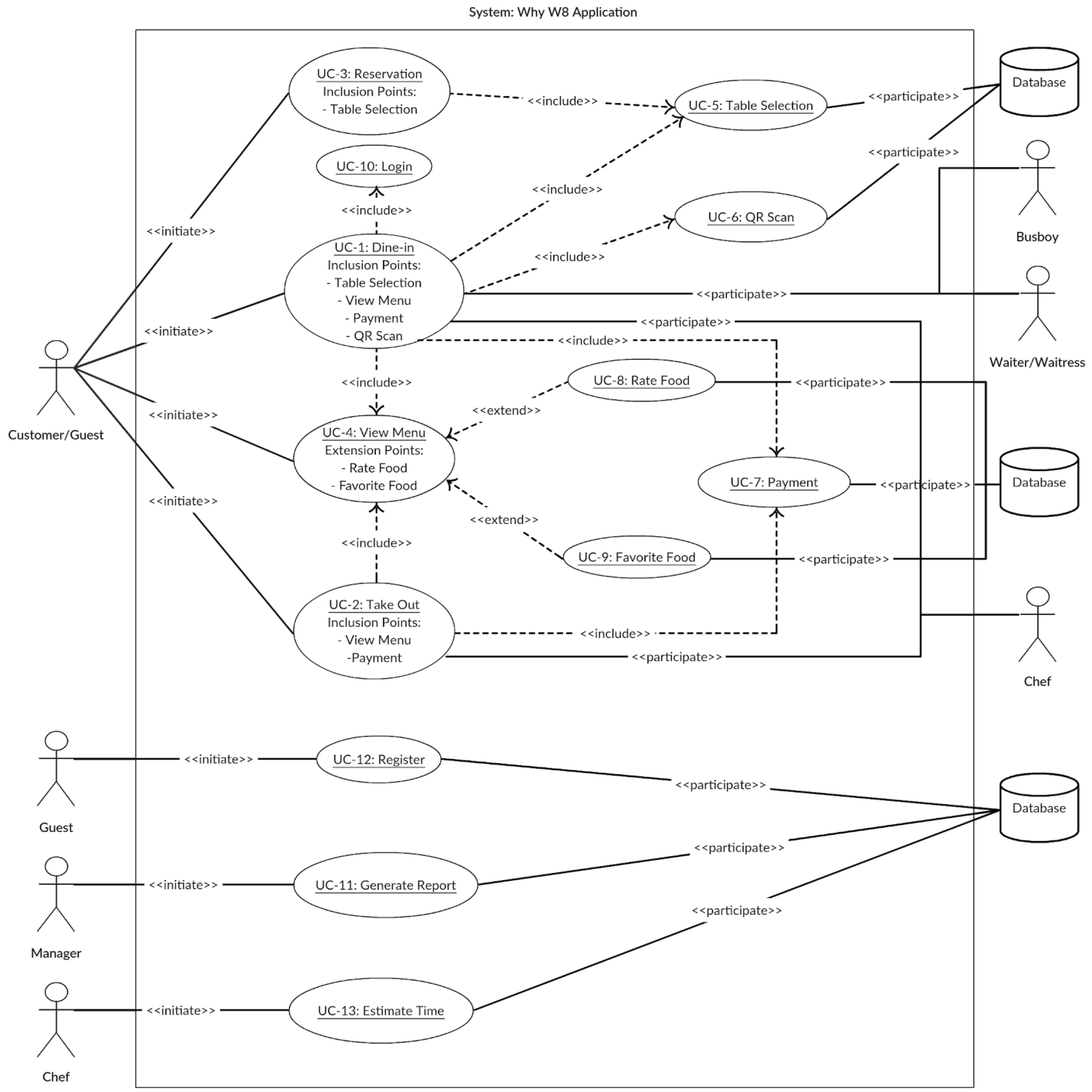


**Figure 4.1** Use Case Diagram for the Why W8 application

## 4.3.3 Traceability Matrix

| Req't | PW | UC-1 | UC-2 | UC-3 | UC-4 | UC-5 | UC-6 | UC-7 | UC-8 | UC-9 | UC-10 | UC-11 | UC-12 | UC-13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REQ-1 | 5 | X | | | | X | | | | | | | | |
| REQ-2 | 5 | X | | | X | | | | | | | | | |
| REQ-3 | 5 | X | | | | | | X | | | | | | |
| REQ-4 | 4 | | | X | | X | | | | | | | | |
| REQ-5 | 3 | X | | | | | | | | | | | | X |
| REQ-6 | 5 | X | | | | | | | | | | | | |
| REQ-7 | 4 | | X | | | | | | | | | | | |
| REQ-8 | 2 | X | | | X | | | | X | X | | | | |
| REQ-9 | 1 | | | | | | | | | | | X | | |
| REQ-10 | 1 | | | | X | | | | | | | | | |
| REQ-11 | 4 | X | | | | | X | | | | | | | |
| REQ-12 | 4 | X | | | | | | | | | | | | |
| REQ-13 | 5 | X | | | | | | | | | | | | |
| REQ-14 | 2 | | | | | | | | | | X | | X | |
| Max PW | | 5 | 4 | 4 | 5 | 5 | 4 | 5 | 2 | 2 | 2 | 1 | 2 | 3 |
| Total PW | | 38 | 4 | 4 | 8 | 10 | 4 | 5 | 2 | 2 | 2 | 1 | 2 | 3 |

**Figure 4.2** Traceability matrix mapping the system requirements to use cases

### 4.3.4 Fully-Dressed Description

| **UC-1 : Dine-in** |
|---|
| **Related Requirements:**<br>● REQ-1, REQ-2, REQ-3, REQ-5, REQ-6, REQ-8, REQ-11, REQ-12, REQ-13<br>**Initiating Actor:**<br>● Customer, Guest<br>**Actor's Goal:**<br>● To choose to dine-in at the restaurant<br>**Participating Actors:**<br>● Waiter/waitress, busboy, database |

| |
|---|
| **Preconditions:**<br>● User has application loaded<br>**Postconditions:**<br>● User is shown available tables |
| **Flow of Events for Main Success Scenario:**<br>1. → The customer opens the application to the homepage for login/guest user<br>2. → The customer clicks login<br>3. ← The system prompts for account information<br>4. → The customer inputs account ID/PW<br>5. ← The system displays option for dine-in, take out, reservation, view menu<br>6. → The customer clicks dine-in<br>**Flow of Events for Alternate Success Scenario:**<br>1. → The customer opens the application to the homepage for login/guest user<br>2. → The customer clicks guest user<br>3. ← The system displays option for dine-in, take out, reservation, view menu<br>4. → The customer clicks dine-in |

| **UC-5 : Table Selection** |
|---|
| **Related Requirements:**<br>● REQ-1, REQ-4<br>**Initiating Actor:**<br>● Customer, Guest<br>**Actor's Goal:**<br>● To select an open table for dining in<br>**Participating Actors:**<br>● Database<br>**Preconditions:**<br>● User has application loaded<br>● User has chosen to dine-in<br>**Postconditions:**<br>● User is prompted to confirm table |
| **Flow of Events for Main Success Scenario:**<br>1. → The customer opens the application to the homepage for login/guest user<br>2. → The customer clicks login<br>3. ← The system prompts for account information<br>4. → The customer inputs account ID/PW<br>5. ← The system displays option for dine-in, take out, reservation, view menu<br>6. → The customer clicks dine-in<br>7. ← The system checks database to find available tables<br>8. ← The system displays available tables for customer<br>9. → The customer clicks an available table<br>10. ← The system confirms table, updates database |

**Flow of Events for Alternate Success Scenario:**

1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks guest user
3. ← The system displays option for dine-in, take out, reservation, view menu
4. → The customer clicks dine-in
5. ← The system checks database to find available tables
6. ← The system displays available tables for customer
7. → The customer clicks an available table
8. ← The system confirms table, updates database

**UC-4 : View Menu**

**Related Requirements:**

- REQ-2, REQ-8, REQ-10

**Initiating Actor:**

- Customer, Guest

**Actor's Goal:**

- To view the menu of food/drinks

**Participating Actors:**

- Database

**Preconditions:**

- User has application loaded

**Postconditions:**

- N/A

**Flow of Events for Main Success Scenario:**

1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks login
3. ← The system prompts for account information
4. → The customer inputs account ID/PW
5. ← The system displays option for dine-in, take out, reservation, view menu
6. → The customer clicks view menu
7. ← The system displays the menu

**Flow of Events for Alternate Success Scenario:**

1. → The customer opens the application to the homepage for login/guest user
2. → The customer clicks guest user
3. ← The system displays option for dine-in, take out, reservation, view menu
4. → The customer clicks view menu
5. ← The system displays the menu

## 4.4 System Sequence Diagrams

UC-1: Dine-in is the most important use case, however it contains a combination of other use cases through inclusions and extensions, so it is not demonstrated here due to the system sequence diagrams showing the flow of the system components. Since UC-1: Dine-in includes other use cases, we show those important use cases instead, as they show the flow of the components in our system.

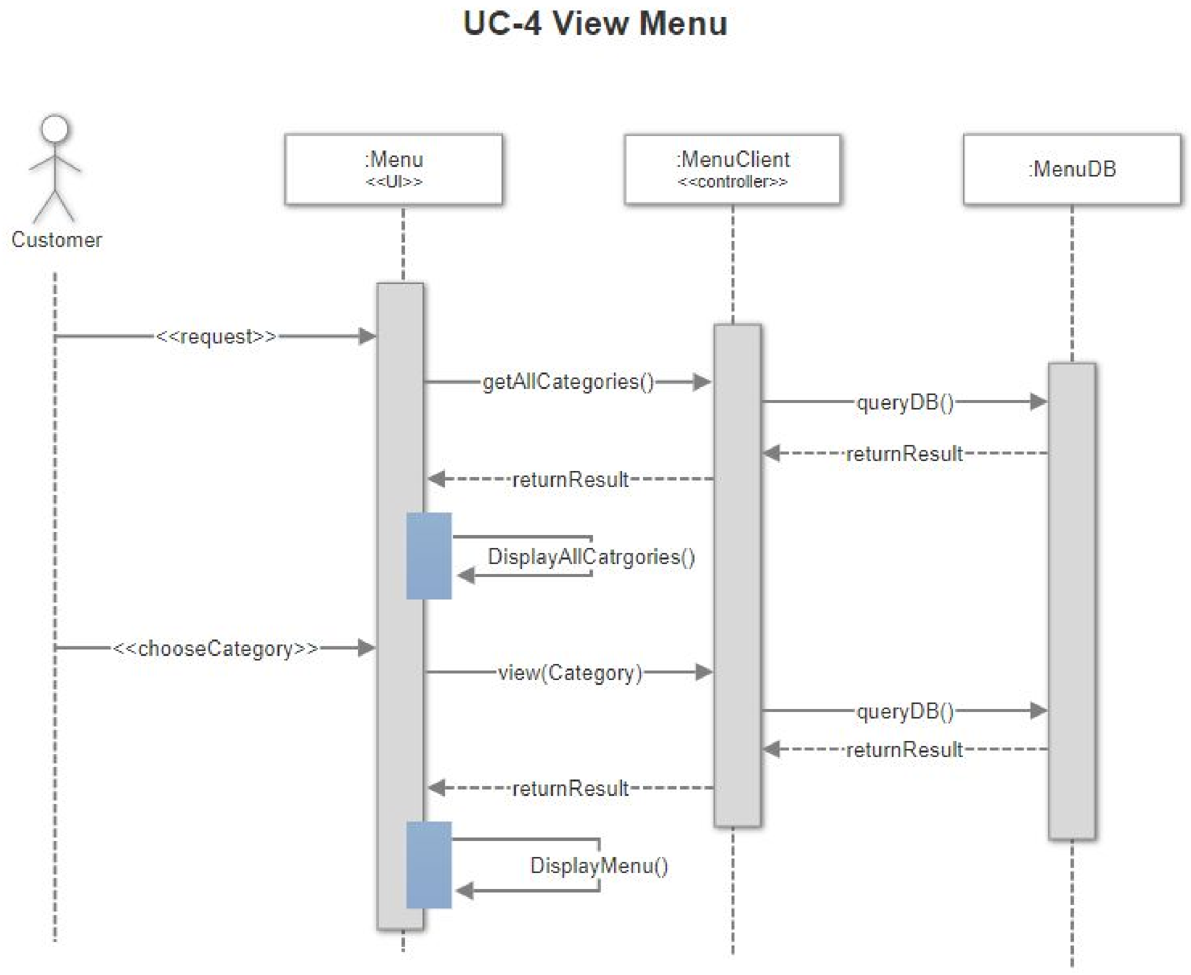


**Figure 4.2:** UC-4 View Menu

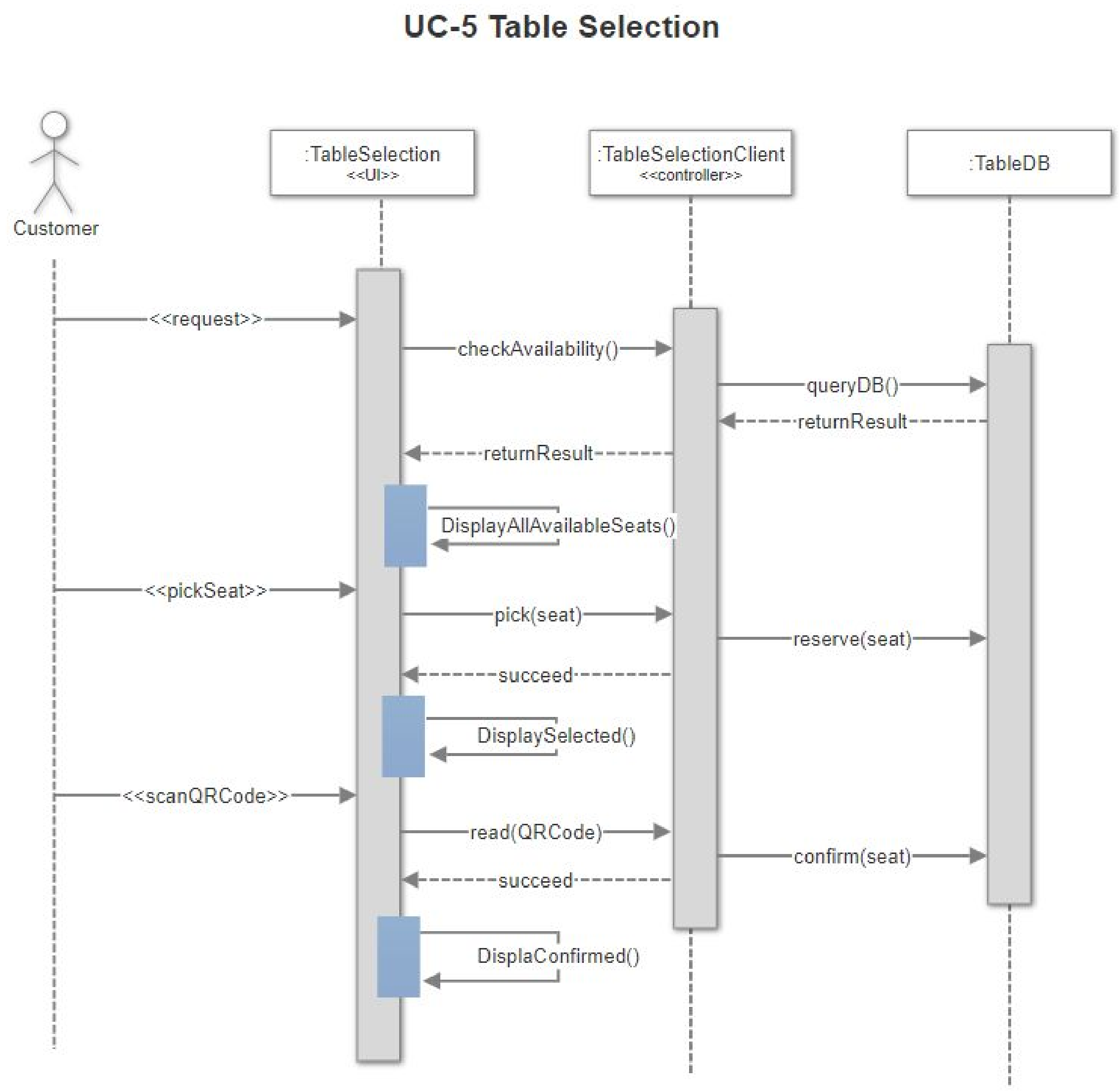


**Figure 4.3:** UC-5 Table Selection

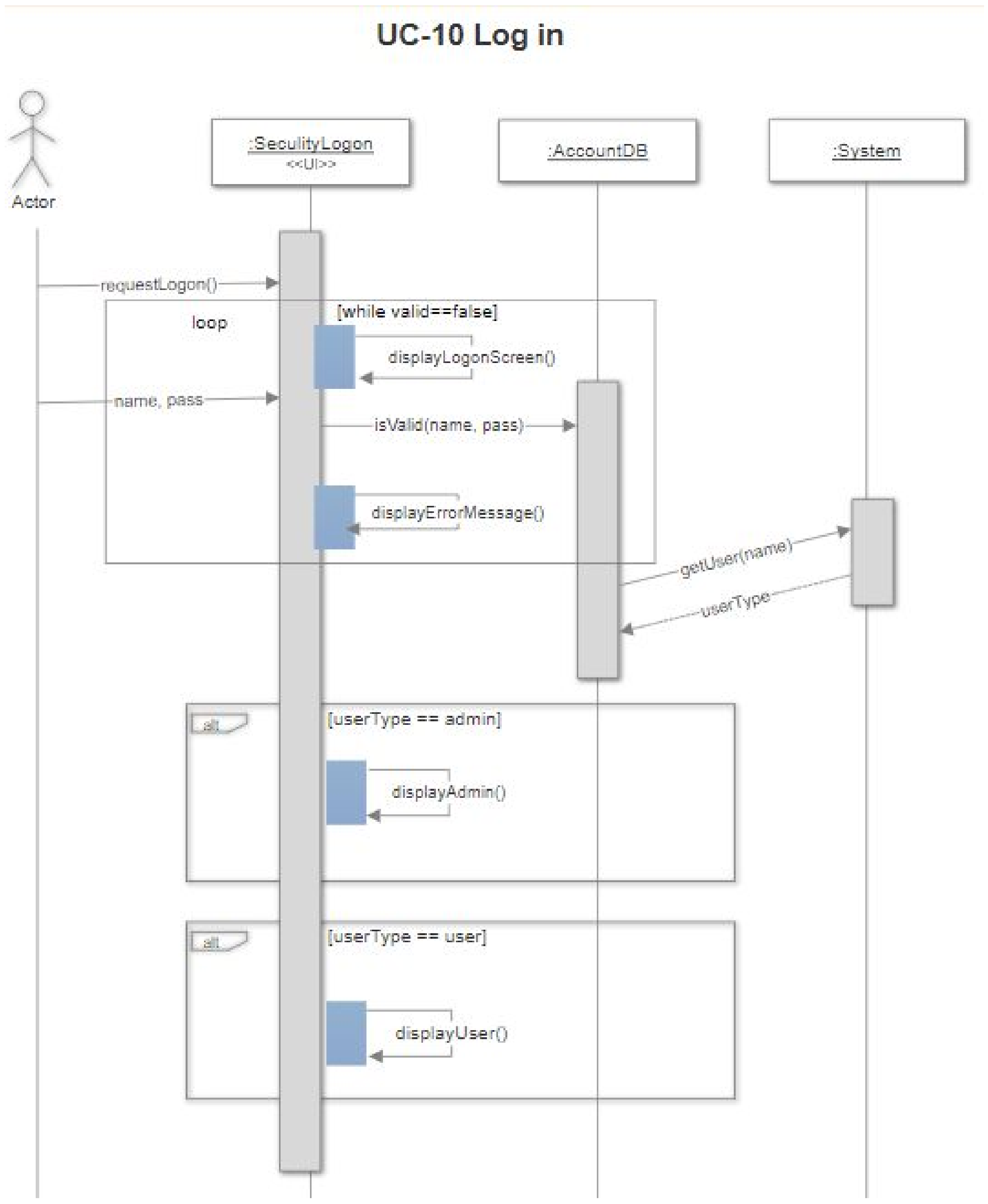


**Figure 4.4:** UC-10 Log In

# 5. Effort Estimation Using Use Case Points

## 5.1 Unadjusted Actor Weight

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

| Actor name | Description of Relevant Characteristics | Complexity | Weight |
|---|---|---|---|
| Customer | Customer is interacting with the system via a graphical user interface. | Complex | 3 |
| Guest | Same as Customer. | Complex | 3 |
| Manager | Same as Customer. | Complex | 3 |
| Chef | Same as Customer. | Complex | 3 |
| Waiter/Waitress | Same as Customer. | Complex | 3 |
| Busboy | Same as Customer. | Complex | 3 |
| Database | Database is another system interacting through a protocol. | Average | 2 |

UAW(Why W8) = 1 × Average + 6 × Complex = 1 × 2 + 6 × 3 = 20

## 5.2 Unadjusted Use Case Weight

| Use case category | Description of how to recognize the use-case category | Weight |
|---|---|---|
| Simple | Simple user interface. | 5 |

| | | |
|---|---|---|
| | Up to one participating actor (plus initiating actor).<br>Number of steps for the success scenario: ≤ 3.<br>If presently available, its domain model includes ≤ 3 concepts. | |
| Average | Moderate interface design.<br>Two or more participating actors.<br>Number of steps for the success scenario: 4 to 7.<br>If presently available, its domain model includes between 5 and 10 concepts. | 10 |
| Complex | Complex user interface or processing.<br>Three or more participating actors.<br>Number of steps for the success scenario: ≥ 7.<br>If available, its domain model includes ≥ 10 concepts. | 15 |

| Use Case | Description | Category | Weight |
|---|---|---|---|
| UC-1: Dine-in | Simple user interface. 6 steps for the main success scenario. Two participating actors (Customer, Database). | Average | 10 |
| UC-2: Take Out | Simple user interface. 6 steps for the main success scenario. Two participating actors (Customer, Database). | Average | 10 |
| UC-3: Reservation | Simple user interface. 6 steps for the main success scenario. Two participating actors (Customer, Database). | Average | 10 |
| UC-4: View Menu | Simple user interface. 7 steps for the main success scenario. Two participating actors (Customer, Database). | Average | 10 |
| UC-5: Table Selection | Simple user interface. 10 steps for the main success scenario. Two participating actors (Customer, Database). | Average | 10 |
| UC-6: QR Scan | Simple user interface. 1 step for the main success scenario. Two participating actors (Customer, Database). | Simple | 5 |
| UC-7: Payment | Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer, Database). | Simple | 5 |
| UC-8: Rate Food | Simple user interface. 1 step for the main success | Simple | 5 |

| | | | |
|---|---|---|---|
| | scenario. Two participating actors (Customer, Database). | | |
| UC-9: Favorite Food | Simple user interface. 1 step for the main success scenario. Two participating actors (Customer, Database). | Simple | 5 |
| UC-10: Login | Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer, Database). | Simple | 5 |
| UC-11: Generate Report | Simple user interface. 9 steps for the main success scenario. Two participating actors (Manager, Database). | Average | 10 |
| UC-12: Register | Simple user interface. 4 steps for the main success scenario. Two participating actors (Guest, Database). | Simple | 5 |
| UC-13: Estimate Time | Simple user interface. 2 steps for the main success scenario. Two participating actors (Customer, Database). | Simple | 5 |

UUCW(Why W8) = 7 × Simple  6 × Average = 7 × 5 + 6 × 10 = 95

## 5.3 Technical Complexity Factors

| **Technical factor** | **Description** | **Weight** | **Perceived Complexity** | **Calculated Factor (Weight × Perceived Complexity)** |
|---|---|---|---|---|
| T1 | Users expect good performance but nothing exceptional | 1 | 3 | 1×3 = 3 |
| T2 | End-user expects efficiency but there are no exceptional demands | 1 | 3 | 1×3 = 3 |
| T3 | Internal processing is relatively simple | 1 | 1 | 1×1 = 1 |
| T4 | No requirement for reusability | 1 | 0 | 1×0 = 0 |
| T5 | Ease of use is very important | 0.5 | 5 | 0.5×5 = 2.5 |

| | | | | |
|---|---|---|---|---|
| T6 | No portability concerns beyond a desire to keep database vendor options open | 2 | 2 | 2×2 = 4 |
| T7 | Easy to change required (different restaurants) | 1 | 3 | 1×3 = 3 |
| T8 | Concurrent use is required | 1 | 4 | 1×4 = 4 |
| T9 | No direct access for third parties | 1 | 0 | 1×0 = 0 |
| T10 | No unique training needs | 1 | 0 | 1×0 = 0 |
| | | | | Technical Factor Total: 20.5 |

TCF = 0.6 + 0.01 × Technical Factor Total = 0.6 + 0.01 × 20.5 = 0.805

## 5.4 Environmental Complexity Factors

| Environmental factor | Description | Weight | Perceived Impact | Calculated Factor (Weight× Perceived Impact) |
|---|---|---|---|---|
| E1 | Beginner familiarity with the UML-based development | 1.5 | 1 | 1.5×1 = 1.5 |
| E2 | Some familiarity with application problem | 0.5 | 2 | 0.5×2 = 1 |
| E3 | Some knowledge of object-oriented approach | 1 | 2 | 1×2 = 2 |
| E4 | Beginner lead analyst | 0.5 | 1 | 0.5×1 = 0.5 |
| E5 | Stable requirements expected | 2 | 5 | 2×5 = 5 |
| E6 | No part-time staff will be involved | -1 | 0 | -1×0 = 0 |
| E7 | Programming language of average difficulty will be used | -1 | 3 | -1×3 = -3 |

| | |
|---|---|
| Environmental Factor Total: | 7 |

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

## 5.5 Use Case Points

UCP = UUCP × TCF × ECF

From the above calculations, the UCP variables have the following values:

UUCP = UAW + UUCW = 20 + 95 = 115
TCF = 0.805
ECF = 1.19

For the case study, the final UCP is the following:

UCP = 115 × 0.805 × 1.19 = 110.16 or 110 use case points.

## 5.6 Duration

Duration = UCP × PF = 110 x 28 = 3,080 hours

# 6. Domain Analysis

## 6.1 Domain Model

### 6.1.1 Concept Definitions

**Table 6.1** Concept Definitions

| Responsibility | Type | Concept |
|---|---|---|
| R1: Coordinate activity between the customer, chef, waiter, busboy, etc. | D | Controller |
| R2: Keep the record of customer accounts with favorites and rated foods | K | Customer Profile |
| R3: Display the options for the customers, waiters, managers, chef, etc. | D | Interface |

| | | |
|---|---|---|
| R4: Prompt the customer to select a table, scan the QR code, make a payment, etc. | D | Controller |
| R5: Store the customer order in the database | K | Communicator |
| R6: Queue incoming orders for the chef to complete | K | Order Queue |
| R7: Select order and send to a specific chef for cooking | D | Communicator |
| R8: Manage interactions with the database | K | DB Connection |
| R9: Record daily statistics for manager report | D | Analytic Calc |
| R10: Prevent invalid table selections | D | Table Status |
| R11: Allow chef to update estimated food time | D | Food Status |
| R12: Input to receive payment type and amount | D | Payment System |
| R13: Display change of table status when customer reserves or leaves and when busboy cleans | D | Table Status |
| R14: Display customer favorites and high rated foods at top of user profile for easy access | D | Interface |
| R15: Conclude the failed negotiations for selecting invalid options | D | Communicator |

### 6.1.2 Association Definitions

**Table 6.2** Association Definitions

| **Concept Pair** | **Association Description** | **Association Name** |
|---|---|---|
| Customer Profile ⇔ DB Connection | Fetch customer's data from the database | QueryDB |
| Customer Profile ⇔ Interface | Display customer's option | Display |
| Interface ⇔ Controller | Allow the user to interact with the option the application ( select a table, scan QR code, estimate food time etc…) | User Action |
| Communicator ⇔ DB Connection | Inject or modify data in the database | UpdateDB |
| Communicator ⇔ Order Queue | Send order and queue for the chef | Send Order |

| | | |
|---|---|---|
| Controller ⇔ Analytic Calc | Display statistic for specific user | Display Statistic |
| Analytic Calc ⇔ DB Connection | Extract data from database for calculation | QueryDB |
| Controller ⇔ Food Status | Allow user to update or view the food status | Update Food Status<br>View Food Status |
| Controller ⇔ Table Status | Allow user to view table status | View Table Status |
| Controller ⇔ Payment System | Allow user to complete the payment | Pay |
| Payment System ⇔ DB Connection | Store payment record to the database | Record Payment |
| Interface ⇔ DB Connection | Get the data from the database for the user | QueryDB |

## 6.1.3 Attribute Definitions

**Table 6.3** Attribute Definitions

| Concept | Attribute | Description |
|---|---|---|
| Customer Profile | accountUsername | Associated username of the customer. Guest account is assigned if no account. |
| | accountPassword | Password of user account. |
| | accountFavorites | Contains all the reviews and favorites left by the customer. |
| Interface | currentInterface | Depending on what account type is logged in, show interface for that type of account only. |
| | rateMeal | Allows the user to rate a meal, and then have that rating stored and displayed. |
| Controller | tableList | Shows the customer the current tables available. |
| | tableConfirm | Allows the customer to confirm the table they sit at via scanning |

| | | |
|---|---|---|
| | | the QR code. |
| | 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. |
| 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 (daily, weekly, monthly). |
| | customerStats | Records statistics about peak customer times, most ordered meals, etc. |
| Table Status | tableStatus | Provides and updates status of each table, whether it is available, occupied, or dirty. |
| Food Status | orderStatus | Allows chef to update the status on currently being cooked meals, i.e time remaining. |
| | orderReady | Allows chef to update waiter and customer that food is cooked and ready. |
| Payment System | paymentMade | Updates the system whether or not the payment has been made. |

## 6.1.4 Traceability Matrix

**Table 6.4** Traceability matrix mapping the use cases to domain concepts

| Use Case | PW | Domain Concepts | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Customer Profile | Interface | Controller | Communicator | Order Queue | Analytic Calc | Table Status | Food Status | Payment System |
| UC-1 | 38 | X | X | | | | X | | | |
| UC-2 | 4 | X | X | | | | X | | | |
| UC-3 | 4 | X | X | X | | | | X | | |
| UC-4 | 8 | X | X | | | | | | | |
| UC-5 | 10 | X | X | X | | | | X | | |
| UC-6 | 4 | X | | X | | | | X | | X |
| UC-7 | 5 | X | X | X | | X | | | X | X |
| UC-8 | 2 | X | X | | X | | X | | | |
| UC-9 | 2 | X | X | | X | | | | | |
| UC-10 | 2 | X | X | | | | | | | |
| UC-11 | 1 | | X | | X | | X | X | X | |
| UC-12 | 2 | | X | | | | | | | |
| UC-13 | 3 | | X | | X | X | X | | X | X |
| Max PW | | 38 | 38 | 10 | 3 | 5 | 3 | 10 | 5 | 5 |
| Total PW | | 79 | 81 | 23 | 8 | 8 | 48 | 19 | 9 | 12 |

## 6.1.5 Domain Model Diagram

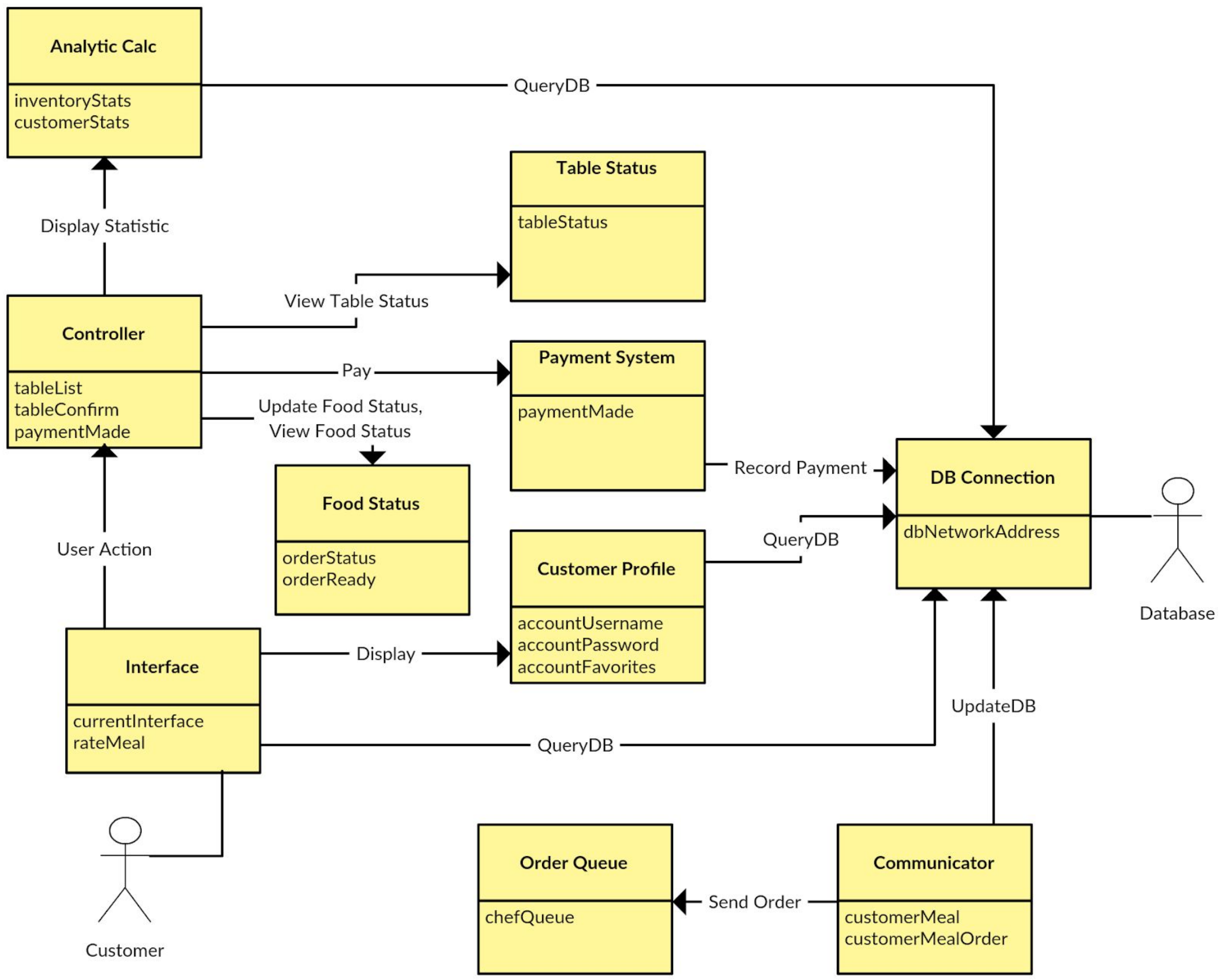


**Figure 6.1** Domain Model Diagram

## 6.2 System Operation Contracts

| | |
|---|---|
| Name: | Dine in |
| Responsibilities: | To dine inside the restaurant |
| Use Case: | UC - 1 |
| Exception: | None |
| Preconditions: | Customer needs to be in the restaurant physically |
| Postcondition: | Customer will occupy one of the available table |

| | |
|---|---|
| Name: | Take out |
| Responsibilities: | To dine outside of the restaurant |
| Use Case: | UC - 2 |
| Exception: | None |
| Preconditions: | Customer needs to order the food thru the application |
| Postcondition: | Food is packed and ready to go |

| | |
|---|---|
| Name: | Reservation |
| Responsibilities: | To reserve a table |
| Use Case: | UC - 3 |
| Exception: | None |
| Preconditions: | None |
| Postcondition: | A table is reserved for a later date / time |

| | |
|---|---|
| Name: | View Menu |
| Responsibilities: | Display the menu to the user |
| Use Case: | UC - 4 |

| | |
|---|---|
| Exception: | None |
| Preconditions: | None |
| Postcondition: | None |

| | |
|---|---|
| Name: | Table Selection |
| Responsibilities: | To select a table |
| Use Case: | UC - 5 |
| Exception: | None |
| Preconditions: | Table must be available |
| Postcondition: | Table is marked as reserved in the database |

| | |
|---|---|
| Name: | QR Scan |
| Responsibilities: | To select an open table |
| Use Case: | UC -6 |
| Exception: | None |
| Preconditions: | Table must be available |
| Postcondition: | Table will be display as occupy |

| | |
|---|---|
| Name: | Payment |
| Responsibilities: | To pay for the meal |
| Use Case: | UC -7 |
| Exception: | None |
| Preconditions: | Order a meal |
| Postcondition: | Payment is completed |

| | |
|---|---|
| Name: | Rate Food |
| Responsibilities: | To rate the food |

| | |
|---|---|
| Use Case: | UC -8 |
| Exception: | None |
| Preconditions: | Food must be ordered by the customer |
| Postcondition: | Store the rating into database |

| | |
|---|---|
| Name: | Favorite Food |
| Responsibilities: | To favorite the food |
| Use Case: | UC -9 |
| Exception: | None |
| Preconditions: | Food must appear in the menu |
| Postcondition: | Favorited food will be highlighted |

| | |
|---|---|
| Name: | Login |
| Responsibilities: | Login |
| Use Case: | UC -10 |
| Exception: | None |
| Preconditions: | User must register an account before |
| Postcondition: | Login to the user’s page |

| | |
|---|---|
| Name: | Generate Report |
| Responsibilities: | Generate report |
| Use Case: | UC -11 |
| Exception: | None |
| Preconditions: | Data must be available in the database |
| Postcondition: | Display the graph and chart |

| | |
|---|---|
| Name: | Register |

| | |
|---|---|
| Responsibilities: | Create an account |
| Use Case: | UC -12 |
| Exception: | None |
| Preconditions: | Account has not existed in the database |
| Postcondition: | Account is registered |

| | |
|---|---|
| Name: | Estimate Time |
| Responsibilities: | Estimate the remaining cooking time |
| Use Case: | UC -13 |
| Exception: | None |
| Preconditions: | Food starts preparing by the chef |
| Postcondition: | Display the remaining cooking time |

## 6.3 Mathematical Model

**Table Designation Algorithm**
The algorithm that allows customers to either reserve or immediately sit at a table of their choosing.

Pseudo code:

```
Table selection algorithm: //Modeled in Java format

//Reservation Object Class
public class Reservation{
        private Time checkIn;
        private Time checkOut;

        //Object constructor
        public Reservation(Time checkIn, Time checkOut){
                this.checkIn = checkIn;
                this.checkOut = checkOut;
        }

        //Returns check-in time
        public Time getCheckIn(){
                return checkIn;
        }

        //Returns check-out time
        public Time getCheckOut(){
                return checkOut;
        }
}

//Table Object Class
public class Table{
        private int tableNumber;
        private int currentTableState;//0=Empty , 1=Occupied,
                //2=Reservation made for w/in 60 min, 3=Customer late for reservation, 4=Dirty
        private QRCode qr;
        private ArrayList<Reservation> reservationArray = new ArrayList<Reservation>();

        //Object constructor
        public Table(int tableNumber, QRCode qr){
                this.tableNumber = tableNumber;
                this.qr = qr;
                currentTableState = 0;
        }

        //Returns table number
        public int getTableNumber(){
                return tableNumber;
        }

        //Returns table state
        public int getCurrentTableState(){
                return currentTableState;
```

```
	}

	//Updates the current table state based on the time
	public void tableUpdate(Time currentTime){
		if(reservationArray.size() > 0){
			Reservation rEarliest = reservationArray.get(0);
			Time rInEarliest = rEarliest.getCheckIn();
			Time rOutEarliest = rEarliest.getCheckOut();

			if(currentTableState == 0){
				if(currentTime.isLaterThan(rInEarliest)){
					if(currentTime.isEarlierThan(rOutEarliest))
						currentTableState = 1;
				}
				else{
					Time nextHour = currentTime.addHour();
					if(nextHour.isLaterThan(rInEarliest))
						currentTableState = 2;
				}
			}

			if(currentTableState == 2){
				Time lateBy10Min = rInEarliest.add10Minutes();
				if(currentTime.isLaterThan(lateBy10Min)
					currentTableState = 3;
			}

			if(currentTableState == 3){
				if(currentTime.isLaterThan(rOutEarliest){
					currentTableState = 0;
					reservationArray.remove(0);
				}
			}
		}
	}

	//Returns the QR code of the table
	public QRCode getQRCode(){
		return qr;
	}

	//Sets the value of the table state
	public void setCurrentTableState(int state){
		currentTableState = state;
	}

	//Stores new reservations onto the reservation array
	private void storeReservation(int i, Time in, Time out){
		Reservation newRes = new Reservation(in, out);
		reservationArray.add(i, newRes);
	}

	//Checks to see if desired reservation time is available, and if so calls upon
storeReservation()
	public boolean addReservation(Time in, Time out){
		int s = reservationArray.size();
		if(s == 0){
			storeReservation(0, in, out);
			return true;
		}

		int i;
		for(i = 0; i < s; i++){
			Reservation rCurrent = reservationArray.get(i);
			Time rInCurrent = rCurrent.getCheckIn();

			Time rOutPrev;
			if(i == 0)
				rOutPrev = null;
			else{
				Reservation rPrev = reservationArray.get(i-1);
				rOutPrev = rPrev.getCheckOut();
			}

			if(rInCurrent.isLaterThan(out){
				if(i == 0){
					storeReservation(0, in, out);
					return true;
				}
				else if(rOutPrev.isEarlierThan(in){
					storeReservation(i, in, out);
					return true;
				}
				else
					return false;
			}
		}

		Reservation rLatest = reservationArray.get(s-1);
		Time rOutLatest = rLatest.getCheckOut();
		if(rOutLatest.isEarlierThan(in){
			Reservation newRes = new Reservation(in, out);
			reservationArray.add(newRes);
			return true;
		}
		else
			return false;
	}

	//Removes the inputted reservation from the reservation array
	public void cancelReservation(Reservation r){
		for(int i = 0; i < reservationArray.size(); i++){
			Reservation rCurrent = reservationArray.get(i);
			if(r.equals(rCurrent)){
				reservationArray.remove(i);
				break;
			}
		}
	}

	//Removes the earliest reservation from the reservation array
```

```
        public void clearEarliestReservation(){
                reservationArray.remove(0);
        }
}

//Restaurant Object Class
public Class Restaurant{
        private int tableTotal;
        private ArrayList<QRCode> qrCodeArray; //A set of given QR codes
        private ArrayList<Table> tableArray;

        //Restaurant constructor
        private Restaurant(ArrayList<QRCode> qrCodeArray){
                this.qrCodeArray = qrCodeArray;
                tableTotal = qrCodeArray.size();

                for(int i = 0; i < tableTotal; i++){
                        QRCode currentQRCode = qrCodeArray.get(i);
                        Table newTable = new Table(i, currentQRCode);
                        tableArray.add(newTable);
                }
        }

        //Returns the total number of tables in the restaurant
        public int getTableTotal(){
                return tableTotal;
        }

        //Returns the array of table objects within the restaurant object
        public ArrayList<Table> getTableArray(){
                return tableArray;
        }
}

//A class in which runs in the restaurant network
public Class NetworkUpdater{
        private Restaurant restaurant;

        //Continuously runs, checking if the state of any tables need to be updated
        public void tableUpdater(){
                int tableTotal = restaurant.getTableTotal();
                Time currentTime;
                Time prevMinuteTime = Time.getTimeOnComputer();

                while(true){
                        currentTime = Time.getTimeOnComputer();
                        currentTimeMinusMinute = currentTime.subtractMinute();

                        if(prevMinuteTime.isEarlierThan(currentTimeMinusMinute)){
                                for(int i = 0; i < tableTotal; i++){
                                        restaurant.getTableArray().get(i).tableUpdate(currentTime);
                                }
                                prevMinuteTime = currentTime;
                        }

                }
        }
}

//A class in which customers have access to
public Class CustomerInterface{
        private Restaurant restaurant;
        private int tableNumber;
        private Reservation reservation;
        private boolean firstScan = false;

        //Allows a customer to reserve a desired table at a desired time
        public void reserveTable(Time in, Time out, int tableIndex){
                boolean b = restaurant.getTableArray.get(tableIndex).reserveTable(in, out);

                if(b){
                        System.out.println("Table reserved, see you at "+in.toString()+"!");
                        tableNumber = restaurant.getTableArray.get(tableIndex).getTableNumber();
                        reservation = new Reservation(in, out);
                }
                else
                        System.out.println("Sorry, the desired table is unavailable for this time.");
        }

        //Allows a customer to cancel a previously made reservation
        public void cancelTableReservation(){
                restaurant.getTableArray.get(tableNumber).cancelReservation(reservation);
        }

        //Allows a customer whom enteres the restaurant without a reservation to be seated at a desired
table
        public void getTableNoReservation(int tableIndex){
                int state = restaurant.getTableArray.get(tableIndex).getCurrentTableState();
                if(state == 0){
                        tableNumber = restaurant.getTableArray.get(tableIndex).getTableNumber();
                        System.out.println("Please take your seat!");
                }
                else if(state == 3){
                        boolean noOtherTable = true;
                        for(int i = 0; i < restaurant.getTableArray().size(); i++){
                                if(i != tableIndex){
                                        int s = restaurant.getTableArray.get(i).getCurrentTableState();
                                        if(s == 0){
                                                noOtherTable = false;
                                                break;
                                        }
                                }
                        }

                        if(noOtherTable){
                                tableNumber =
restaurant.getTableArray.get(tableIndex).getTableNumber();
                                System.out.println("Please take your seat!");
                        }
                        else
                                System.out.println("Please select one of the other current tables at
this time,\n...
```

```
                                        ...the person who reserved this table is running late.");
                }
        }

        //Returns an integer depending on whichever QR code was scanned and the state of the table the 
QR code resides
        private int qrCodeCheck(QRCode qr){
                for(int i = 0; i < restaurant.getTableArray().size(); i++){
                        Table t = restaurant.getTableArray().get(i);
                        if(qr.equals(t.getQRCode()){
                                if(tableNumber == t.getTableNumber()){
                                        if(t.getCurrentTableState == 0 || t.getCurrentTableState == 2 
|| t.getCurrentTableState == 3)
                                                return 0;
                                        else
                                                return 1;
                                }
                                else
                                        return 2;
                        }
                }

                return -1;
        }

        //The method that runs when a user scans a QR code prior to being seated
        public void firstQRScan(){
                QRCode currentQR = QRCode.scan();
                int qrResult = qrCodeCheck(currentQR);

                if(qrResult == 0){
                        restaurant.getTableArray().get(tableNumber).setCurrentTableState(1);
                        firstScan = true;
                        System.out.println("Welcome!");
                }
                else if(qrResult == 1)
                        System.out.println("Sorry, your table is currently unavailable.");
                else if(qrResult == 2)
                        System.out.println("This is table #"+t.getTableNumber().toString()+", not table 
#"+tableNumber+".");
                else
                        System.out.println("ERROR");

        }

        //The method that runs when a user scans a QR code when they are ready to leave the restaurant
        public void secondQRScan(){
                QRCode currentQR = QRCode.scan();
                int qrResult = qrCodeCheck(currentQR);

                if(qrResult == 0){
                        restaurant.getTableArray().get(tableNumber).setCurrentTableState(4);
                        restaurant.getTableArray().get(tableNumber).clearEarliestReservation();
                        firstScan = false;
                        System.out.println("Thank you, please come again!");
                }
                else if(qrResult == 2)
                        System.out.println("This is the table you initially sat at.");
                else
                        System.out.println("ERROR");
        }
}

//A class in which workers have access to
public Class workerInterface(){
        private Reservation restaurant;

        //Allows a worker to update the state of a table after being cleaned
        public void clearTable(int tableIndex){
                if(restaurant.getTableArray().get(tableIndex).getCurrentTableState) == 4){
                        restaurant.getTableArray().get(tableIndex).setCurrentTableState(0);

restaurant.getTableArray().get(tableIndex).tableUpdate(Time.getTimeOnComputer());
                }
        }
}
```

**Order Queue Algorithm**

As the order comes in, it is enqueued onto a queue. It is dequeued onto the Order Queue screen for the chef to interact with. However, only a set amount of orders are shown on this screen so as not to overwhelm the chef. So a certain amount of orders are dequeued from the queue into a list and more dequeued only after an order is confirmed to be done and taken out from the list.

Pseudo code:

```
Order anOrder= new Order()
orderQueue.enqueue(anOrder)
Int displayed=0
while(!orderQueue.isEmpty()){
        if(count<limit){
                display(orderQueue.dequeue())
                displayed++;
        }
        if(isDone()){  //an order is marked
                displayed--;
        }
}
```

## 7. Interaction Diagrams

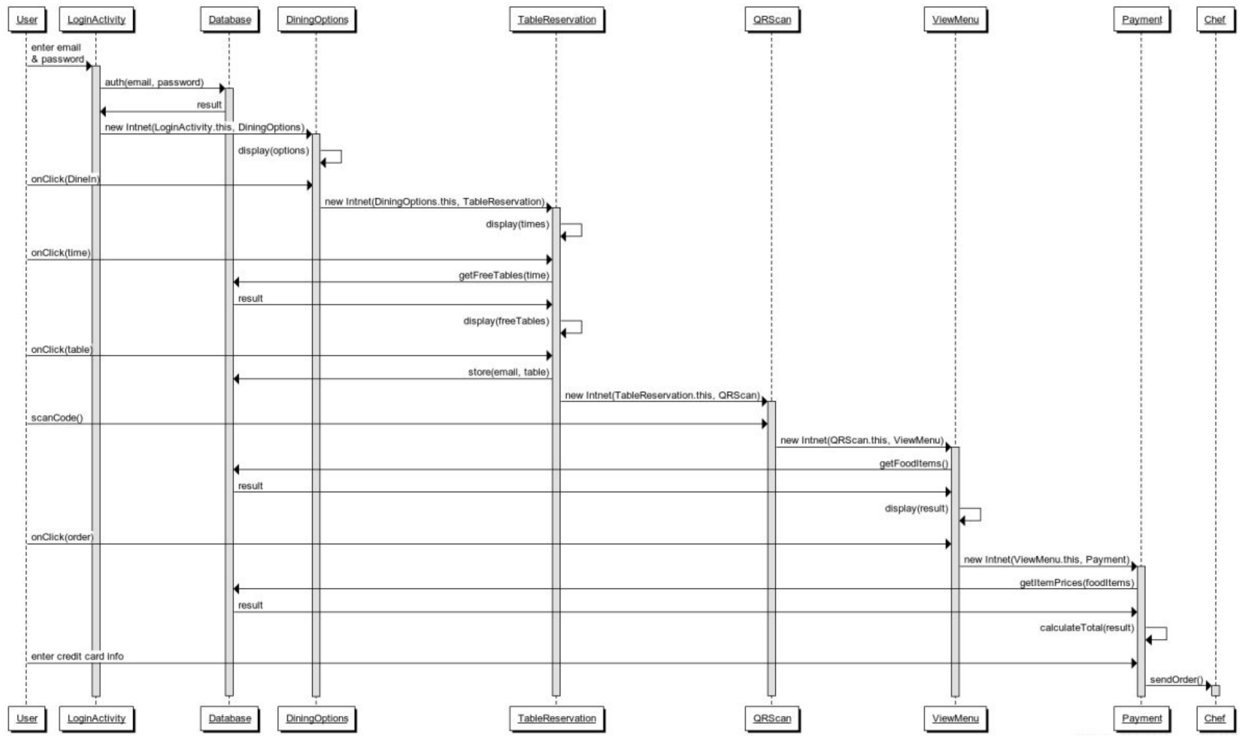


**Figure 7.1** UC-1: Dine-In

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.

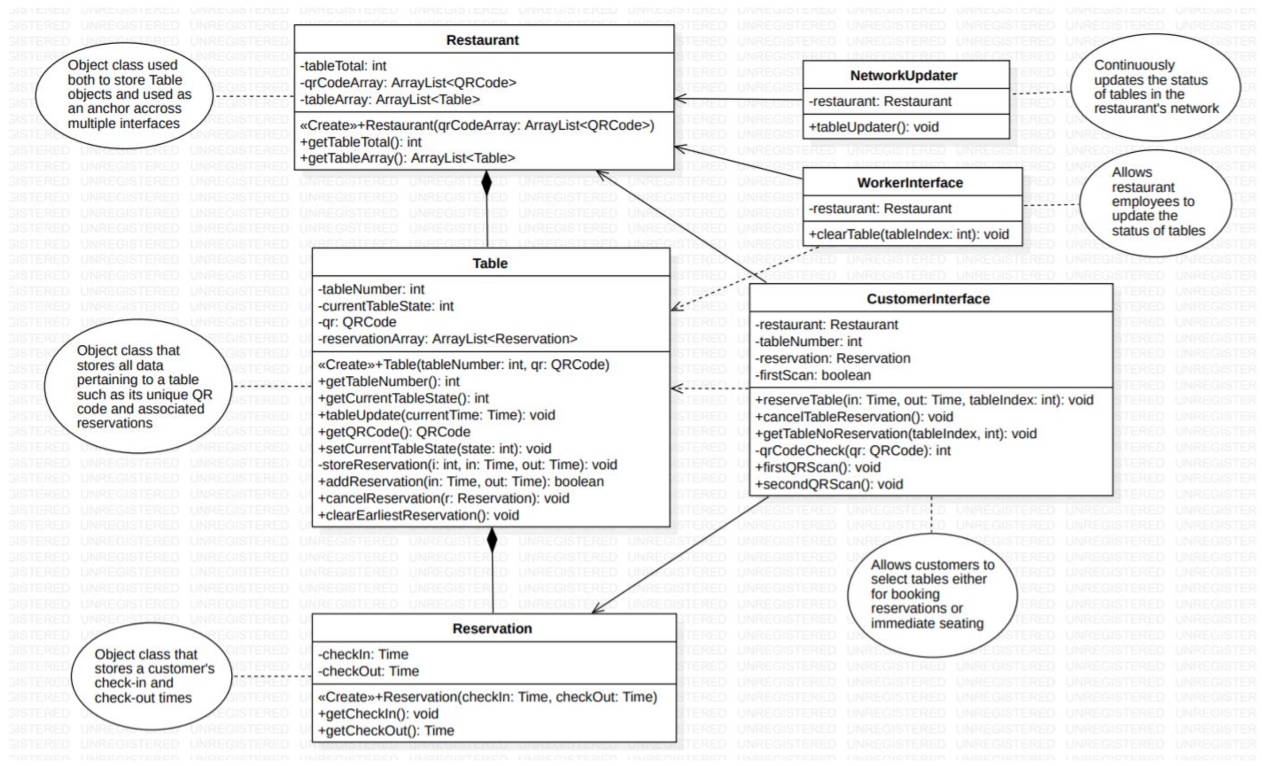

Restaurant
-tableTotal: int
-qrCodeArray: ArrayList<QRCode>
-tableArray: ArrayList<Table>
«Create»+Restaurant(qrCodeArray: ArrayList<QRCode>)
+getTableTotal(): int
+getTableArray(): ArrayList<Table>
Object class used both to store Table objects and used as an anchor accross multiple interfaces
NetworkUpdater
-restaurant: Restaurant
+tableUpdater(): void
Continuously updates the status of tables in the restaurant's network
WorkerInterface
-restaurant: Restaurant
+clearTable(tableIndex: int): void
Allows restaurant employees to update the status of tables
Table
-tableNumber: int
-currentTableState: int
-qr: QRCode
-reservationArray: ArrayList<Reservation>
«Create»+Table(tableNumber: int, qr: QRCode)
+getTableNumber(): int
+getCurrentTableState(): int
+tableUpdate(currentTime: Time): void
+getQRCode(): QRCode
+setCurrentTableState(state: int): void
-storeReservation(i: int, in: Time, out: Time): void
+addReservation(in: Time, out: Time): boolean
+cancelReservation(r: Reservation): void
+clearEarliestReservation(): void
Object class that stores all data pertaining to a table such as its unique QR code and associated reservations
CustomerInterface
-restaurant: Restaurant
-tableNumber: int
-reservation: Reservation
-firstScan: boolean
+reserveTable(in: Time, out: Time, tableIndex: int): void
+cancelTableReservation(): void
+getTableNoReservation(tableIndex, int): void
-qrCodeCheck(qr: QRCode): int
+firstQRScan(): void
+secondQRScan(): void
Allows customers to select tables either for booking reservations or immediate seating
Reservation
-checkIn: Time
-checkOut: Time
«Create»+Reservation(checkIn: Time, checkOut: Time)
+getCheckIn(): void
+getCheckOut(): Time
Object class that stores a customer's check-in and check-out times

**Figure 7.2** 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.

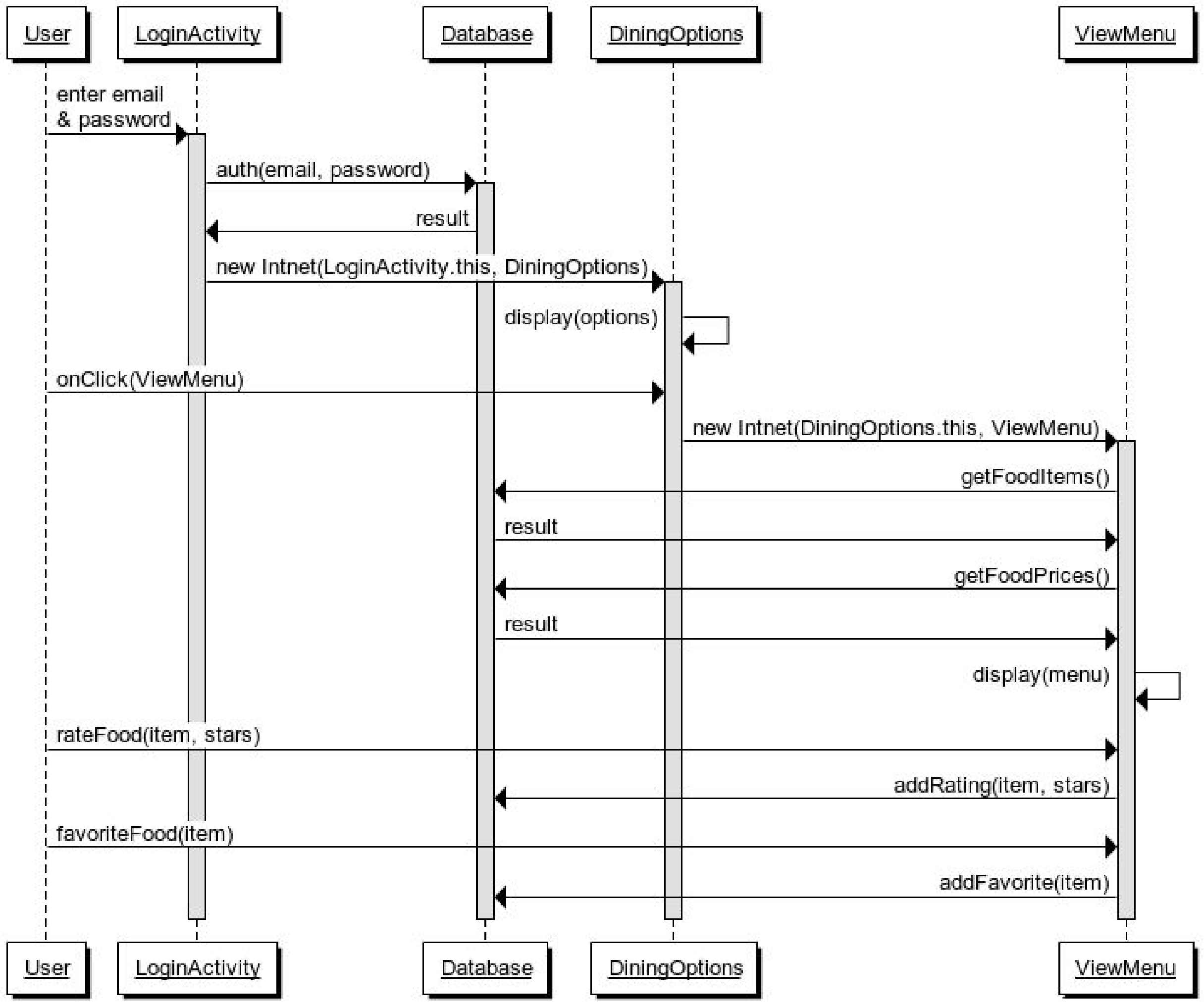


**Figure 7.3** UC-4: View Menu

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.

## 7.1 Design Patterns

Since we are building an Android application with a touch screen user interface that receives inputs/parameters from the user we found that the Command Pattern design pattern facilitated decoupling of a parameters from the business logic that is associated with the code. Specifically, the command pattern lets us localize parameters in a command object and encapsulate them for when the client, which in this case is our database server, may need to utilize the parameters by a different object. Our user interface will be receiving the parameters and then calling the correct methods that will utilize the data.With this pattern, we can log all requests to add inputs from the user and execute them through a command invoker. By having this ability our commands can be reversed/undone with rollback requests from the user whenever a parameter needs to be changed. Although, we used a public interface to create all of the receivers a disadvantage of using this design pattern is that we need to create a list of many small classes that store lists of these commands.

For our project optional features were continuously being added for the user such as a social media platform built inside the app. Since some of these functions were more important than others, we used the decorator pattern to separate the essential functions and the non essential functions. In order to avoid the client code, otherwise known as the caller of the methods or the users of the given classes, from changing every time a feature was added we used the decorator pattern to remove the dependence between the client and feature.

We also utilized the Low Coupling Principle in our application. The Low Coupling Principle requires that objects should not take on too many communication responsibilities. Our design meets the requirement since we have minimized the number of unnecessary interactions between objects/classes to just the necessary ones. Our application utilizes object-oriented design principles that showcase aspects of both the High Cohesion and Low Coupling principles. This allowed our Be Healthy application to operate smoothly, satisfy the user's needs, and create a sophisticated design model for our application.

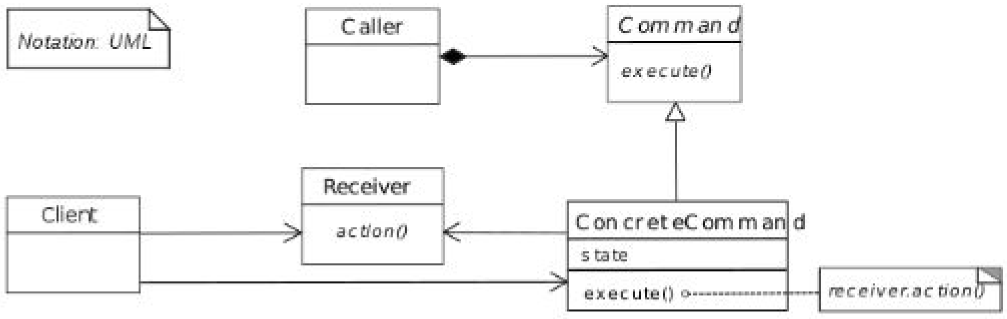
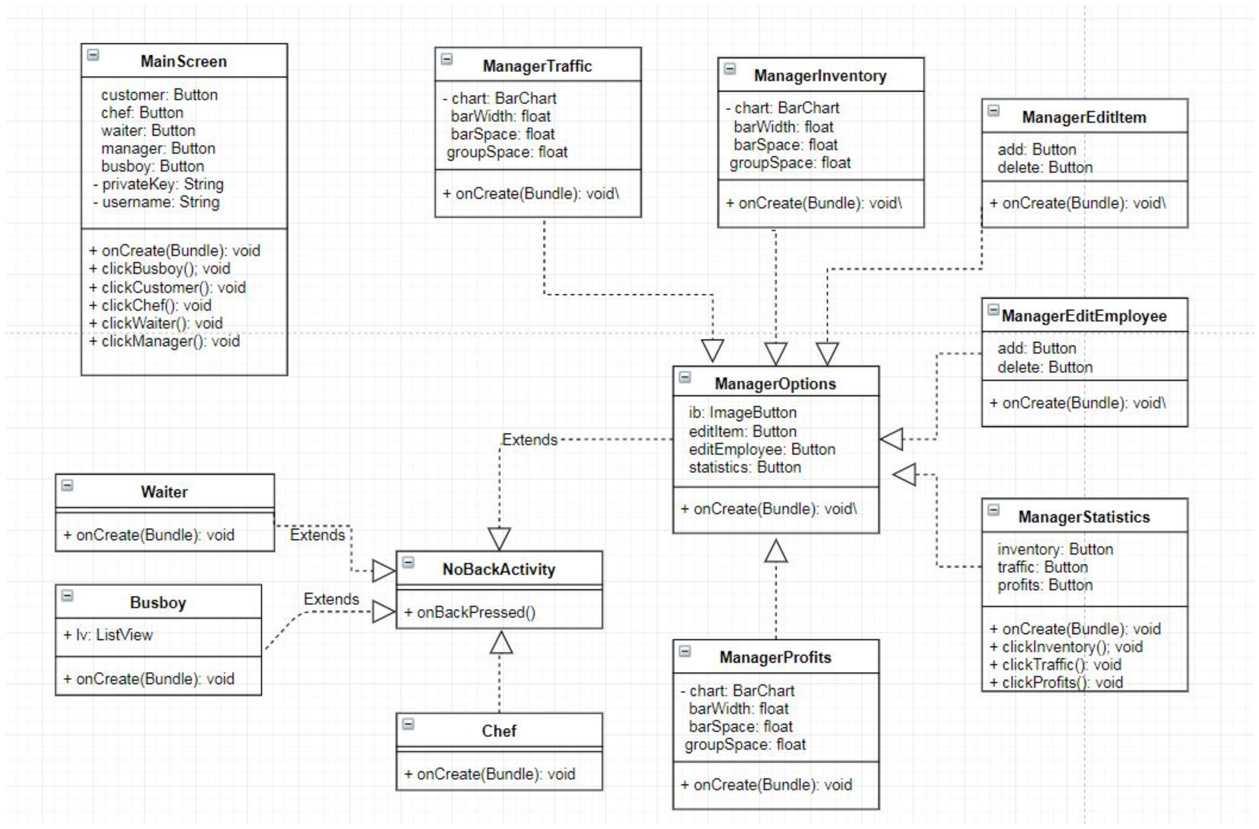
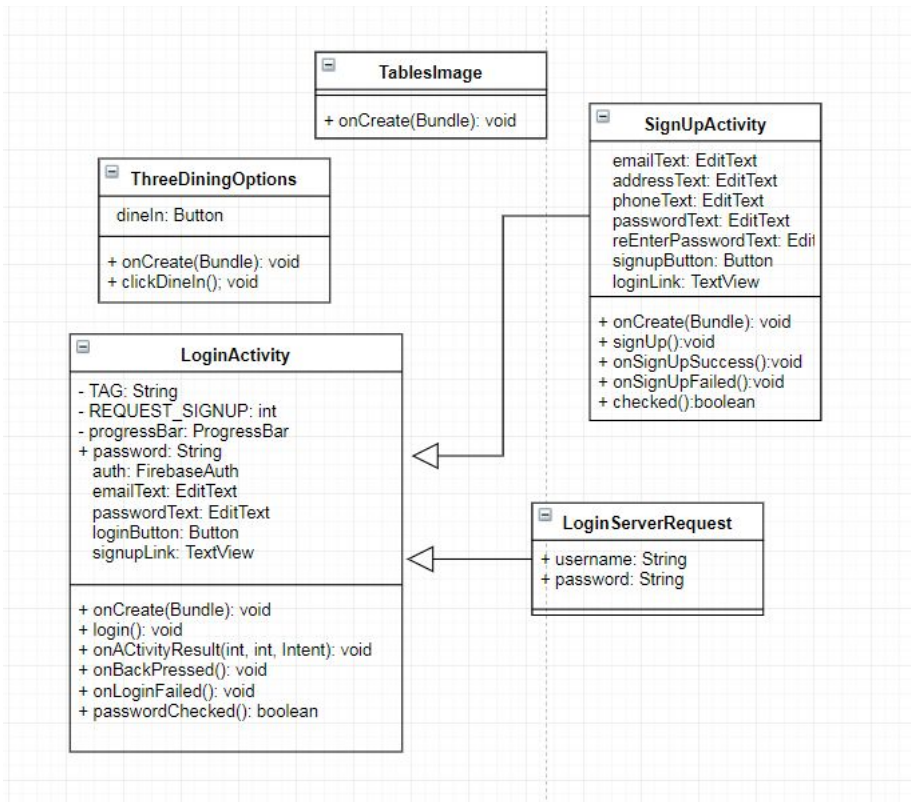


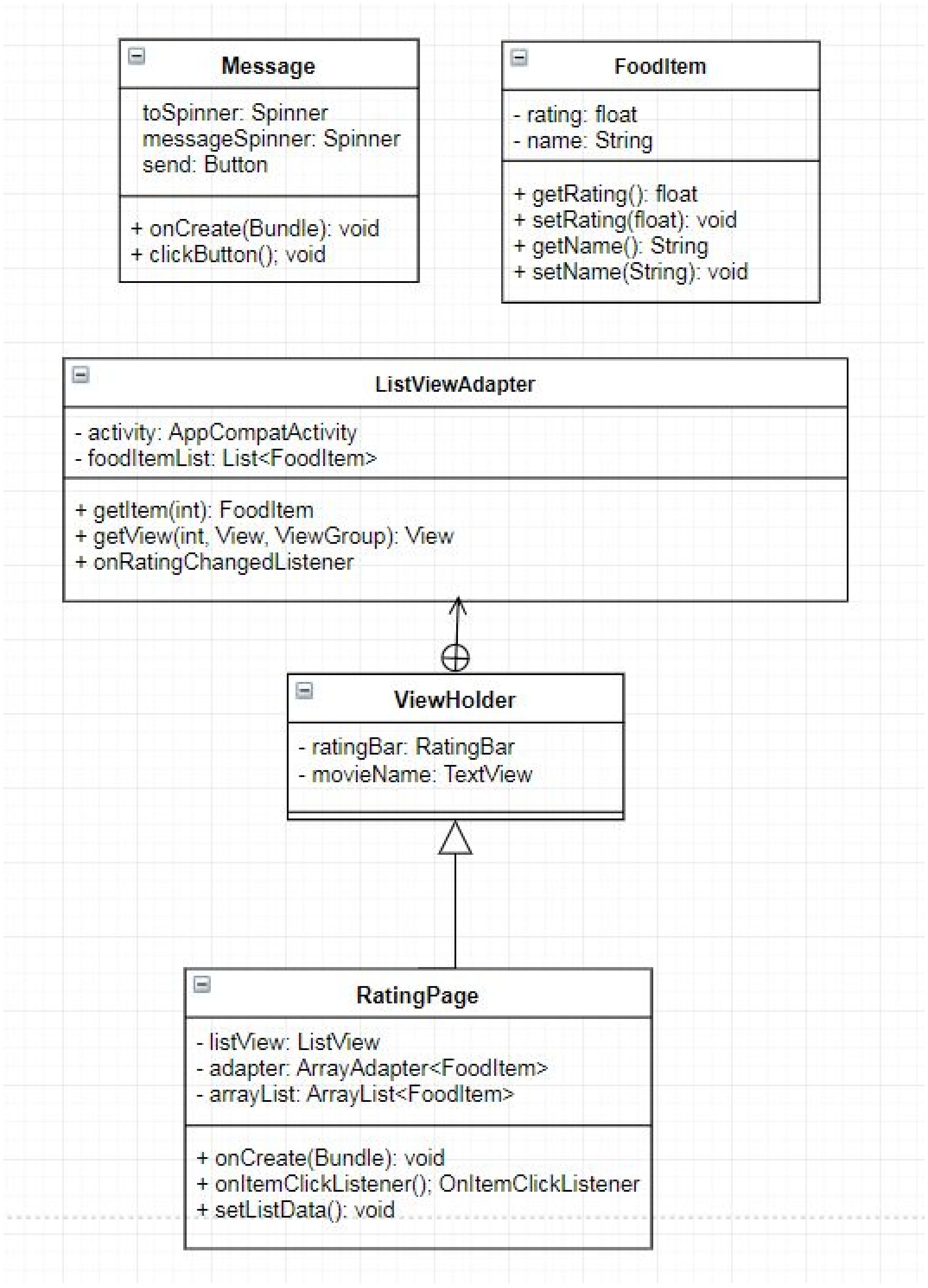
**Figure 7.4:** This is an example of the Command design pattern. For its use, an invoker object manages the different modes for commands without the need for the client to be aware of the different modes as seen from the UML diagram above.

# 8. Class Diagram and Interface Specification

## 8.1 Class Diagram






Message
toSpinner: Spinner
messageSpinner: Spinner
send: Button
+ onCreate(Bundle): void
+ clickButton(); void
FoodItem
- rating: float
- name: String
+ getRating(): float
+ setRating(float): void
+ getName(): String
+ setName(String): void
ListViewAdapter
- activity: AppCompatActivity
- foodItemList: List<FoodItem>
+ getItem(int): FoodItem
+ getView(int, View, ViewGroup): View
+ onRatingChangedListener
ViewHolder
- ratingBar: RatingBar
- movieName: TextView
RatingPage
- listView: ListView
- adapter: ArrayAdapter<FoodItem>
- arrayList: ArrayList<FoodItem>
+ onCreate(Bundle): void
+ onItemClickListener(); OnItemClickListener
+ setListData(): void

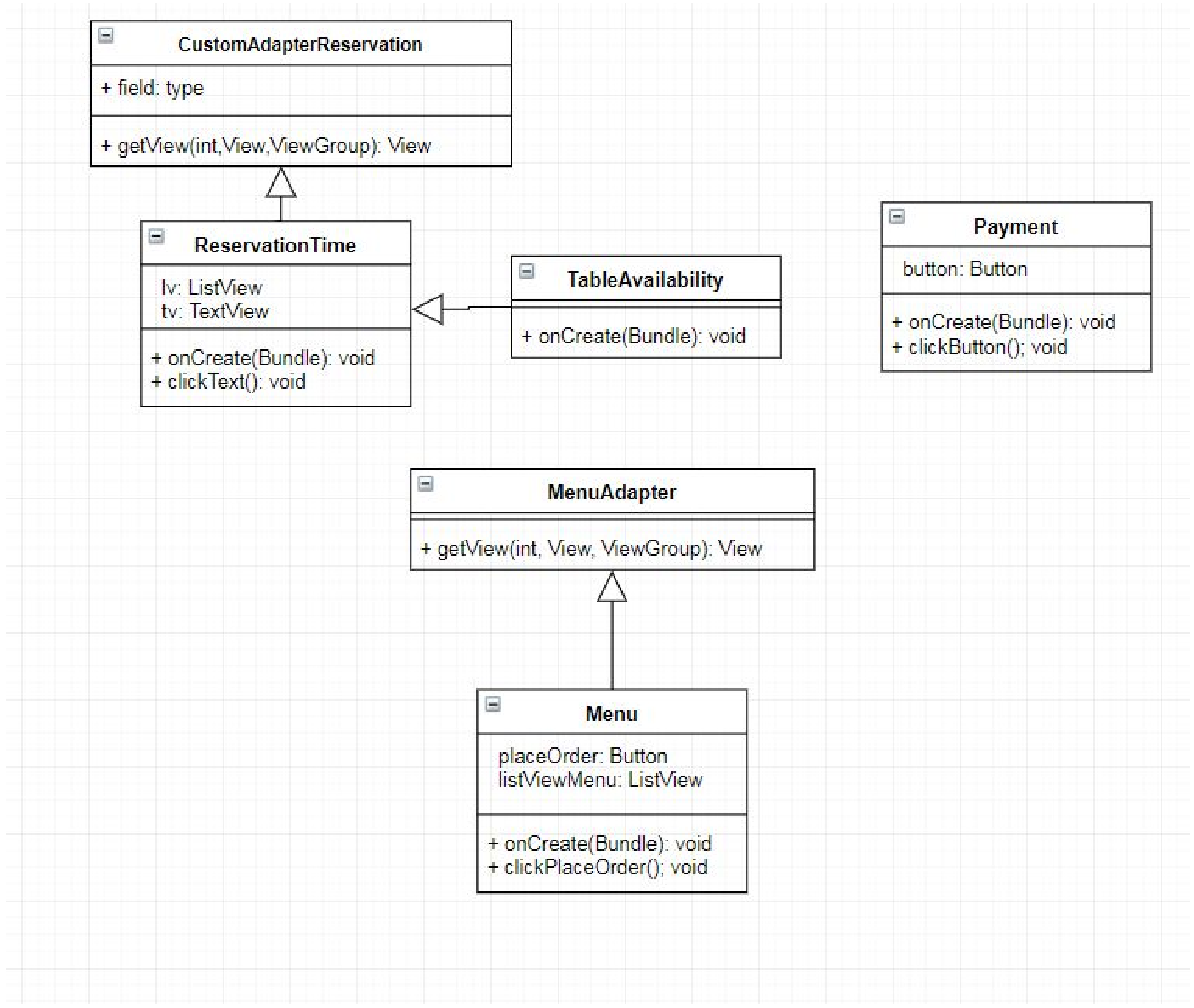


**Figure 8.1:** Class Diagram

## 8.2 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 |

## 8.3 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

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

## 8.4 Object Constraint Language (OCL)

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

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

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

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

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

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

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

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

Context: SignupActivity::onSignUpSuccess
Invariant: signupButton
Pre-conditional: If sign up success
Post-conditional: setResult(RESULT_OK, null)

Context: SignupActivity::onSignUpFailed
Invariant: signupButton
Pre-conditional: If sign up failed
Post-conditional: Toast.makeText(getBaseContext(), "MainScreen Failed", Toast.LENGTH_LONG).show()
Context: SignupActivity::checked:boolean
Invariant: name, address, email, mobile, password, reEnterPassword
Pre-conditional: name.isEmpty() || name.length() < 3
Pre-conditional: address.isEmpty()

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

# 9. System Architecture and System Design

## 9.1 Architectural Styles

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.

## 9.2 Identifying Subsystems

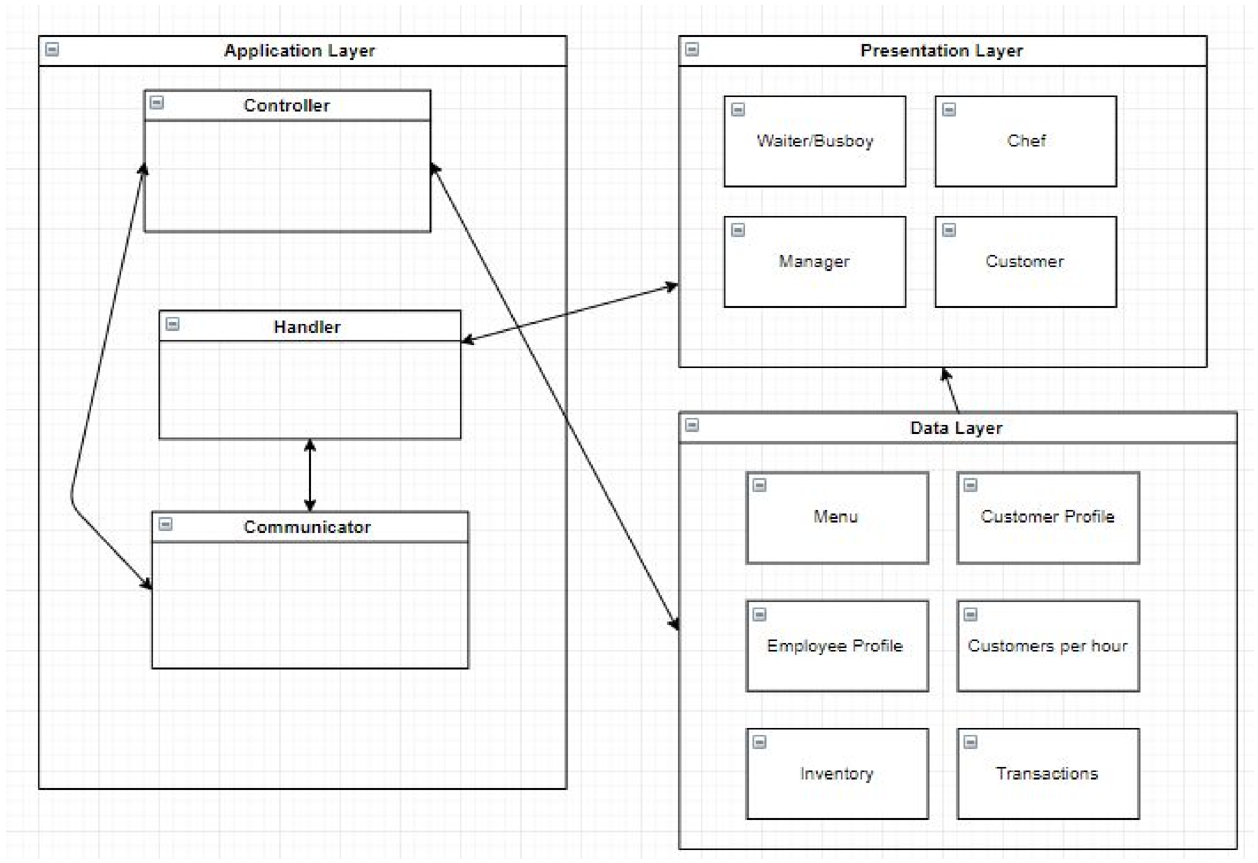


**Figure 9.1:** UML Package Diagram

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.

## 9.3 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.

## 9.4 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]
}
```

## 9.5 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.

## 9.6 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.

## 9.7 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.

# 10. Algorithms and Data Structures

## 10.1 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).

## 10.2 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.

# 11. 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.

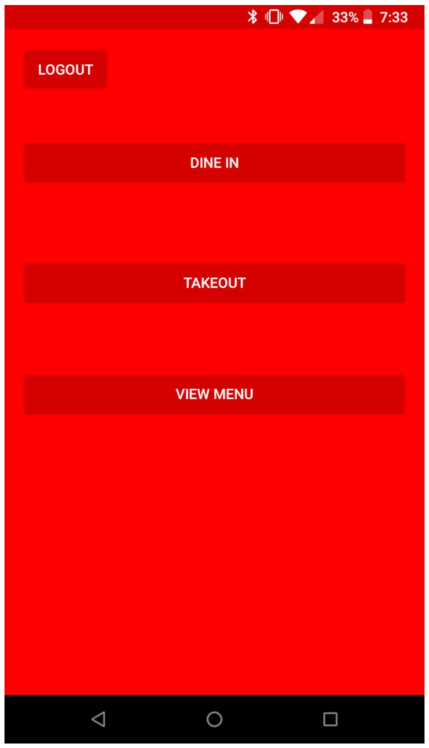


**Figure 11.1:** Dining Options Screen

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.

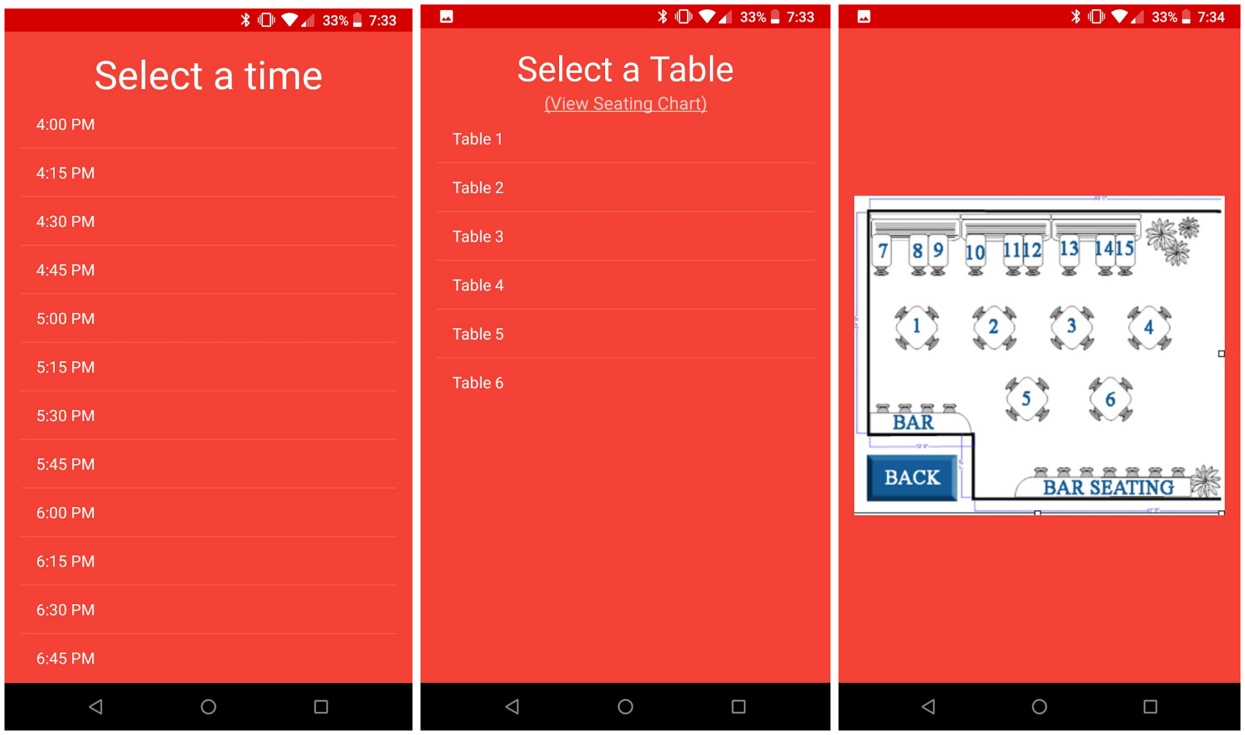


**Figure 11.2:** Table Reservation Screens

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.

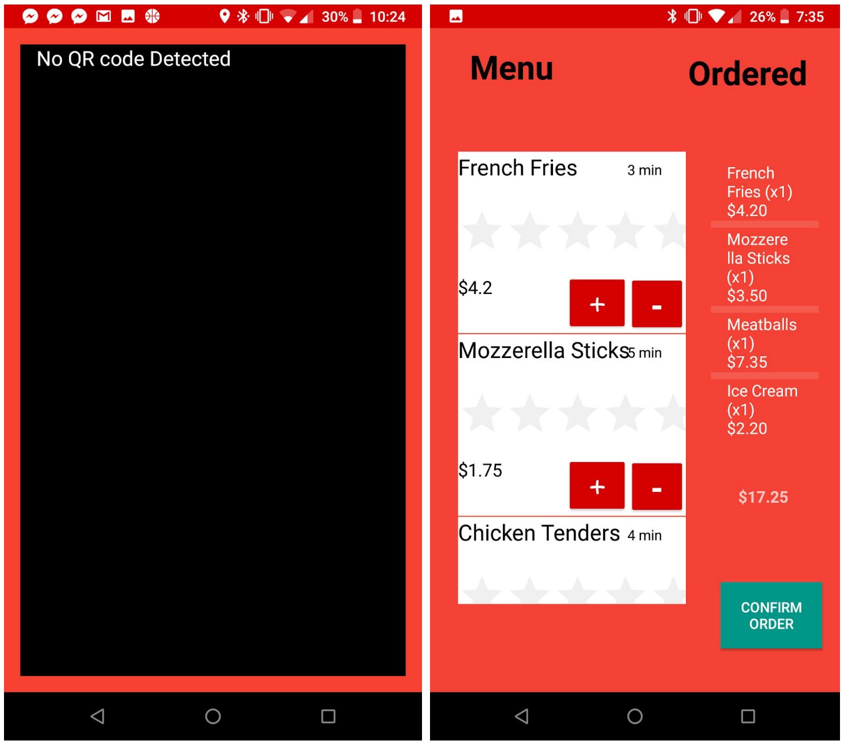


**Figure 11.3:** QR Scan and Menu Screens

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.

<u>Chef Interface:</u>

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.

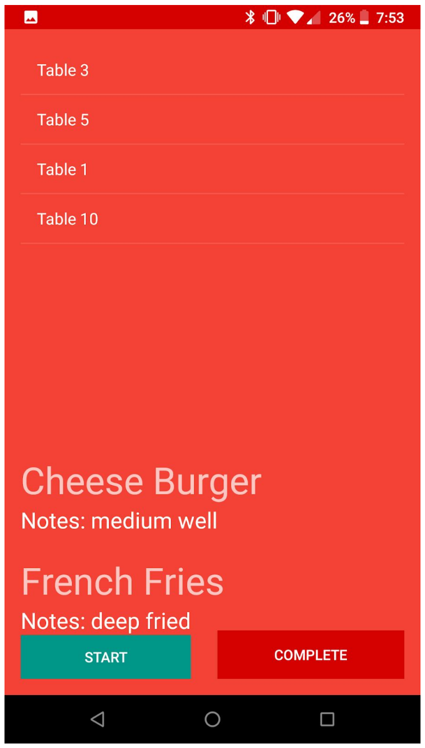


**Figure 11.4:** Chef Queue Screen

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.

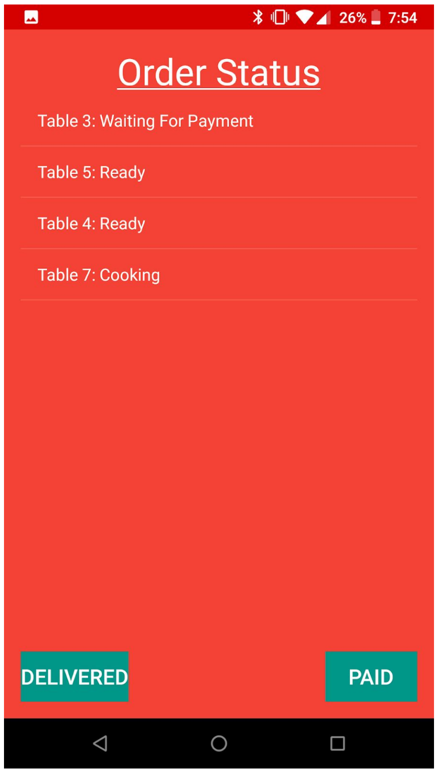

Order Status

Table 3: Waiting For Payment

Table 5: Ready

Table 4: Ready

Table 7: Cooking

**Figure 11.5:** Order Status Screen

<u>Busboy Interface:</u>

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.

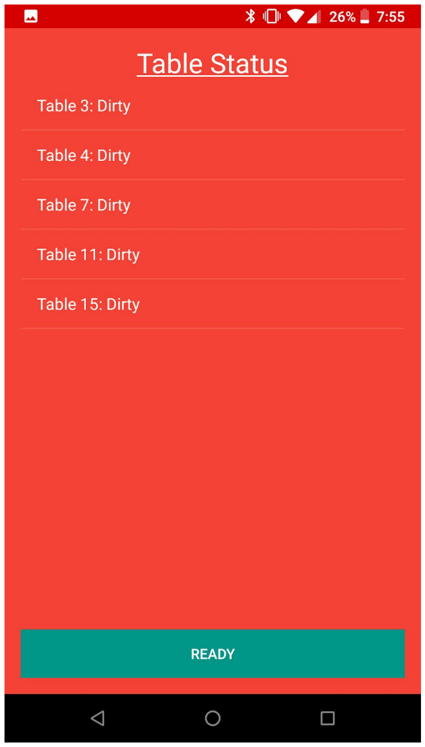


Table 3: Dirty
Table 4: Dirty
Table 7: Dirty
Table 11: Dirty
Table 15: Dirty

**Figure 11.6:** Table Status Screen

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.

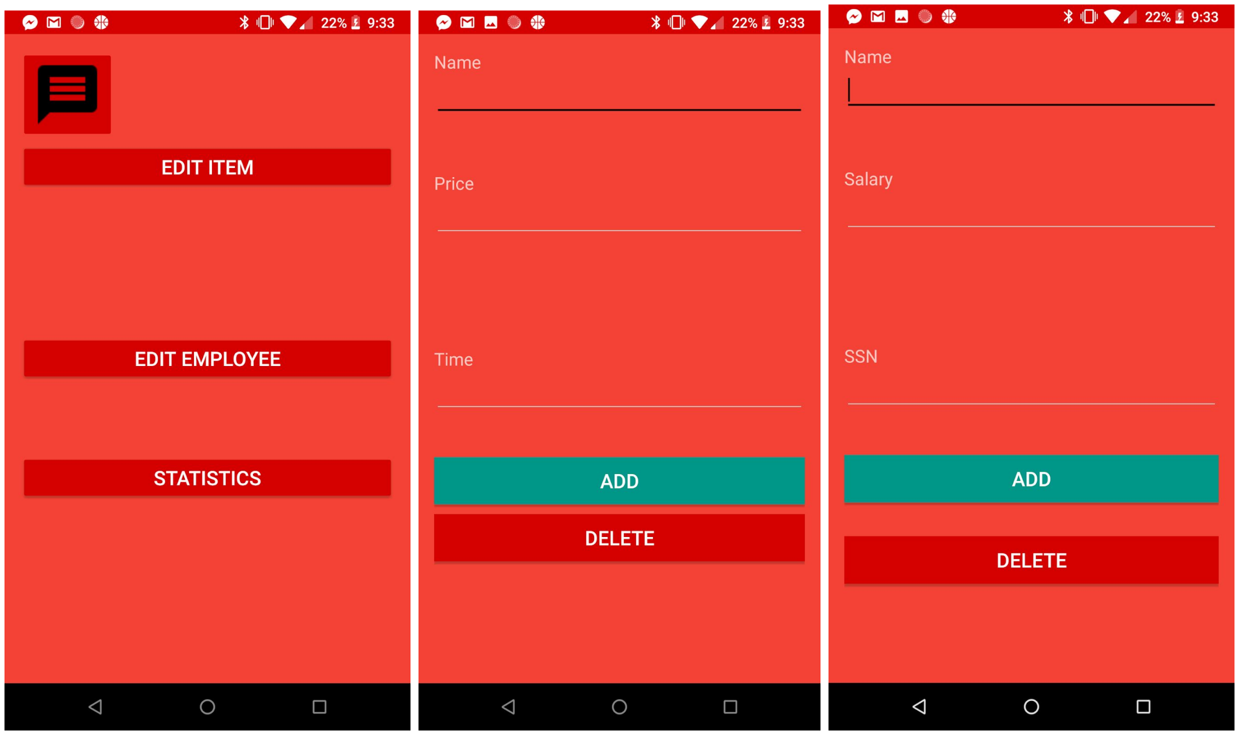


**Figure 11.7:** Manager Edit Item & Employee Screens

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.

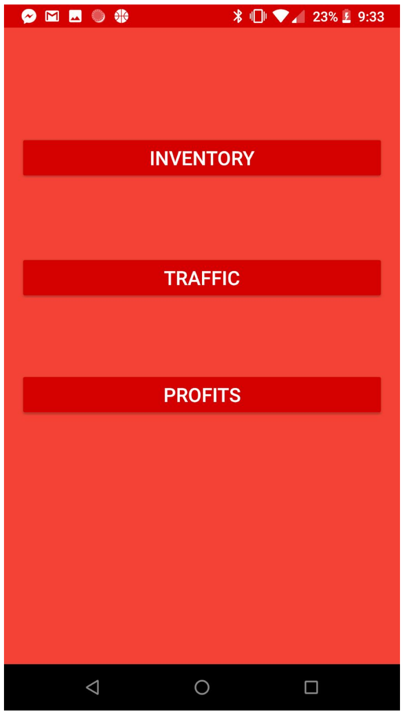


**Figure 11.8:** Manager Options Screen

Inventory, traffic, and profits interfaces:

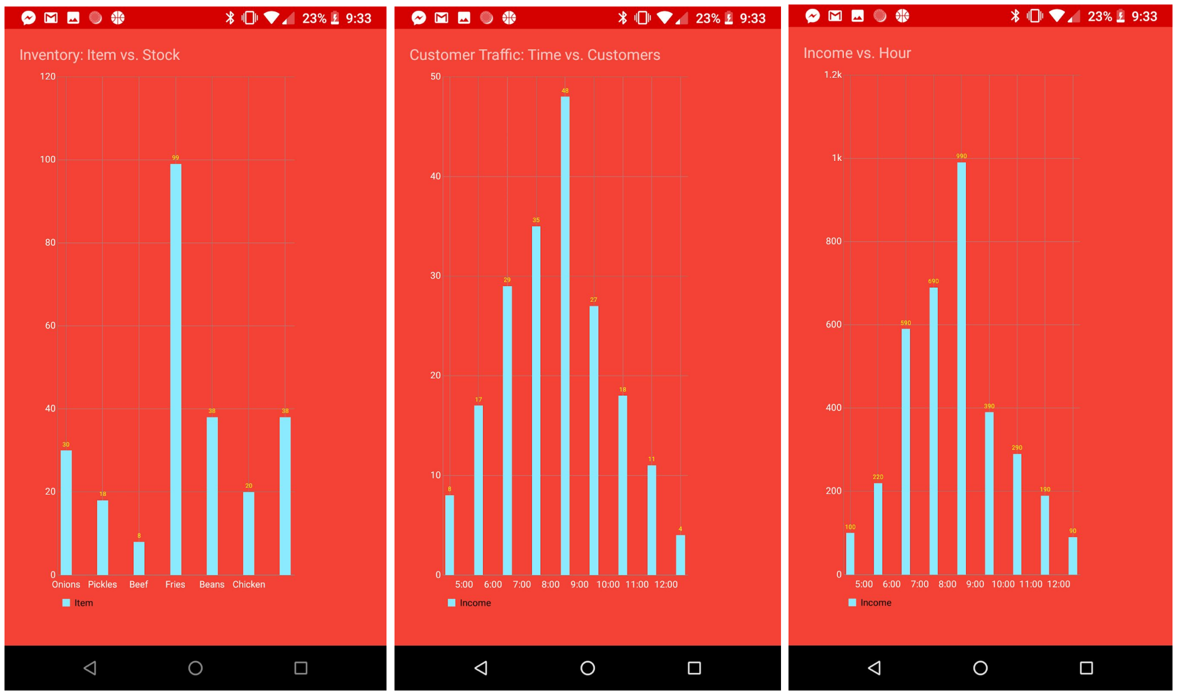


**Figure 11.9:** Statistics Screens

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.

# 12. Design of Tests

## 12.1 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**<br>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.<br>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. | The app sees that the enter credentials are valid and takes the user to the user's main page. | The app displays the customer home page. |
| Step 2: Enter a username and password combination that is valid for manager and select login. | The app sees that the entered credentials are valid for manager and displays the manager main page. | The app displays the main page for the manager. |
| Step 3: Enter a username and password combination that is valid for chef and select login. | The app takes the user to the main page for the chef after confirming that information is valid. | The app displays the main page for the chef |
| Step 4: Enter a username and password combination that is invalid and select login | The app stays at the main page and displays an error message. | App states that the login was not successful. |

| **Test-Case Identifier: TC - 2**<br>**Use Case Tested: UC - 1**<br>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.<br>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**<br>**Use Case Tested: UC - 2**<br>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.<br><br>Step 2: Add item and check that it appears on the ordered list<br><br>Step 3: Delete item from the list and check ordered list is properly updated<br><br>Step 4: Confirm purchase and click order. | Takeout menu screen opens.<br><br>Ordered item appears on ordered list with quantity and price displayed along the side.<br><br>Items are deducted in quantity and the price is adjusted accordingly.<br><br>Order is sent to the queue for the chef to select and make. | Takeout menu screen opens<br><br>Ordered item appears on the list with price and quantity shown.<br><br>Items are deducted in quantity and the price is adjusted accordingly.<br><br>This has yet to be implemented |

| **Test-Case Identifier: TC - 4**<br>**Use Case Tested: UC - 3 and UC -5**<br>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.<br>Input Data: Button selection | | |
|---|---|---|
| **Test Procedure:** | **Expected Result:** | **Actual Result:** |
| Test 1: Select time for which the table is to be reserved.<br><br>Test 2: Select an available table to reserve in advance.<br><br>Test 3: Select a table that is marked as reserved or taken.<br><br>Test 4: Select a table for the current time (not reserving table) | Only tables that are available during this are shown as available while the rest are marked as taken.<br><br>Table is set to be reserved for time the user has chosen.<br><br>Table cannot be selected by the user.<br><br>Table screen is taken to the QR scan code page to confirm that the user is sitting at the selected table. | All tables in the restaurant is shown without any distinctions.<br><br>This feature has yet to be implemented.<br><br>This feature has yet to be implemented.<br><br>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 | | This feature has not been |

| | | |
|---|---|---|
| Step 4: Load the menu page<br><br>Step 5: Select a number of stars corresponding to what to rate the food.<br><br>Step 6: Order food which are rated. | The submitted reviews are stored in the user profile.<br><br>The menu should be appear with a star rating system next to each item.<br><br>The appropriate number of stars can be selected out of the maximum number of stars there are.<br><br>The rated objects should be stored in the user profile. | implemented yet.<br><br>The menu appears with a star rating system next to each item.<br><br>The appropriate number of stars can be selected out of the maximum number of stars there are.<br><br>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.<br><br>Step 2: Select the favorite button<br><br>Step 3: Submit the review<br><br>Step 4: Load the menu page<br><br>Step 5: Select a favorite button next to the food item.<br><br>Step 6: Select confirm payment page. | Foods that are ordered should appear along with a favorite button next to each type of food the user has ordered.<br><br>The food item next to the selected button is shown as favorited.<br><br>The food items the user clicked the favorite button on are stored in the user profile.<br><br>The menu should be appear with a favorite button next to each item.<br><br>The button displays that the food item is favorited.<br><br>The food items the user clicked the favorite button on is selected. | These features have not been implemented. |

| **Test-Case Identifier: TC - 11**<br>**Use Case Tested: UC - 11**<br>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.<br>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**<br>**Use Case Tested: UC - 12**<br>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.<br>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 | 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. | These features have not yet been implemented. |
| Step 2: Type in new email address and 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 | |

**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.<br><br>Step 2: Update the status of one of the dishes. | The Order queue is displayed along with the percentage of each dish the chef has completed.<br><br>The wait time for the order should now decrease. | This has yet to be implemented. |

## 12.2 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’

# 13. History of Work

<u>September 9-18:</u>

We got together in class during this team to form a team to move forward with on a project. Finding a project was no difficulty for us as one of our members had a family friend that worked in the restaurant business. As such, we found it would be fitting if we chose the Restaurant automation idea as this would solve a problem that a member felt especially connected to. And thus, we moved on to writing the proposal. The proposal was written as a collaboration effort over google drive as is the case for most of our reports.

<u>September 19-October 7:</u>

We used the feedback from the proposal to plan our steps during this period. Our main feedback was to make sure to implement novel ideas to add on to our current proposal. And so we spent time on developing new ideas. Also during this time, we met a couple of times to discuss how to allocate who does what for the first report. Everyone contributed their assigned parts throughout this period over the shared drive. A couple of us also got together to begin work on setting up the menu screen in the app.

<u>October 8-31:</u>

During this time, we spent our time setting up the app for the upcoming demo. We split up parts for each person to work on and met every week to make sure everyone was on track. The features we mostly focused on were the menu, rating system, reservation list, and food time. These features were implemented in android studio. We connected the firebase database to handle logins for different users and keep track of inventory. The week leading up to the demo, we met almost every day to make an extra push to complete. On the last day, we merged all the branches and ran final tests to make sure our features worked. We then set up our presentation and finished our preparations.

<u>November 1-11:</u>

Following the demo, we looked into how we can use the feedback to improve on our project. At the same time, we continued to add on the second report. This report was influenced heavily by the demo feedback as well as the feedback we got from the reviews written by our peers in this class. We made the appropriate changes and allocated for people to work on the new sections for this report. We worked remotely on this project and combined our work through google drive once again.

<u>November 12-December 9:</u>

After finishing the second report, we went straight into working on our last report. As soon as the peer reviews our classmates wrote on our report 2 was available, we went to work reading their input. We kept their suggestions in mind while working on our last report. And so, we spent a good deal of time going through and fixing the issues from report 2. Again for this report, we all contributed on our own time into a shared drive. Also during this time, we worked on features suggested to us during our demo. We met various times throughout this period to work together on our app and to make sure we were not falling behind.

<u>December 9-12:</u>

We will be putting our finishing touches on the app during this time. Most of the main features have already been implemented. However, we are pushing ourselves to showcase new features on our app. We will be spending this time making sure that our app functions properly through detailed tests.

## 14. 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.