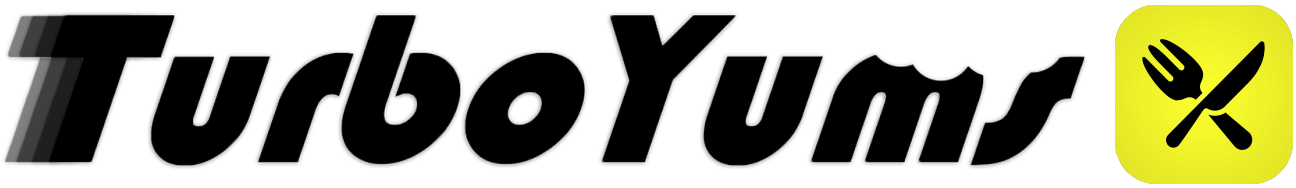


Restaurant Automation Software Suite

https://turboyums.github.io/SEWebsite/

Group 13

Third Report

Michelle Curreri
Suvranil Ghosh
Ziad Mallah
Anthony Merheb
Roshni Shah
Holly Smith
Hersh Shrivastava
Brandon Tong
Dante Torello

Table of Contents

Individual Contributions Breakdown

Topics	Michelle	Suvranil	Ziad	Anthony	Roshni	Holly	Hersh	Brandon	Dante
Individual Contributions Breakdown	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Summary of Changes	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Customer Statement of Responsibilities	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Glossary of Terms	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
System Requirements	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Functional Requirements Specification	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Effort Estimation using Use Case Points	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Domain Analysis	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Interaction Diagrams	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
System Architecture and System Design	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Algorithms and Data Structures	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
History of Work, Current Status, and Future Work	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
References	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Editing	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%
Organization	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%	11.11%

Summary of Changes

- Customer Statement of Requirements
- On-Screen Requirements Updated
- Use Case Diagram Revised
- Edited Fully Dressed Use Case Descriptions
- Updated Interface Requirements
- Design of Tests Revised
- Updated Test Cases
- Updated Table of Contents
- Added to Glossary of Terms
- Updated Hardware Requirements
- Updated Algorithms and Data Structures
- Updated Database Schema
- Updated System Requirement
- Updated Data Types and Operation Signatures and removed classes, classes removed:
    - Reservation
    - Chef
    - Busboy
    - Server
    - Manager
    - Address
- Updated Class Diagram
- Updated Interface Designs and Descriptions
- Included OCL
- Updated Description of Diagrams

1 Customer Statement of Requirements

1.1 Problem Statement

Chef:

Working in a kitchen can be strenuous and it can be difficult to keep track of everything that needs to be done. Orders need to be filled in a reasonable amount of time, while simultaneously ensuring that any special dish accommodations are accounted for. This needs to be done with accuracy because it can be a disaster if an order is made incorrectly or even worse, a person with a food allergy could get hurt as a result of a waiter's poor handwriting. We want to get our orders out with accuracy and speed to make sure we have a happy, returning customer. When dishes are complete and the server does not pick it up for a few minutes, it can be a pain because the food gets cold and it takes up space on the counter that could be used otherwise.

It would be a great help if I had something that clearly displays all incoming orders. This can save me more time in the kitchen and make the cooking process more efficient. Finally, I would love a way to signal to the servers that the order is finished and ready to be taken to the table to reduce the time the food sits on the counter.

How will TurboYums help me? The application will generate a queue for the chef to follow so that they are aware of what dishes they should be making and provide them with a general idea of how old the order is so that they may stay on track. The application will also allow chefs to ping waiters when a certain order is complete so that they know when to come and pick the meals up so that they may be delivered to the tables.

Host/Hostess:

Whenever I am welcoming customers into our restaurant, I aim to make the start of their experience as flawless as possible. As parties of multiple people arrive, they tend to have different requests and requirements based on how many people they have and where they want to sit. Sometimes you can have a party of 8 looking for a round table in a corner, where as other times you can get a family of 4 wanting a booth. I love to find them their preferred seating, but sometimes there just isn't an available spot, and we aren't really sure how long it will be for an appropriate spot to open up.

Finding an available table for guests can sometimes be a challenge, especially when I'm unsure of which tables are ready to go. When I'm welcoming customers in, it would be great if there was a way to easily keep track of which tables are currently cleaned and empty and which are occupied, without having to go through endless reams of paper. At the same time, it'd be nice to keep track of how long each table has already been occupied. This will be a big help in providing guests with accurate estimations on when a valid seating arrangement may become available.

How will TurboYums help me? Using a handy tablet/phone I can keep on me, I can mark tables as occupied, vacant, clean, etc. as the day goes on. Whenever I seat new guests, I can mark the table as occupied. Once the guests leave, that table can be marked as vacant, notifying a busboy that the table is ready to be cleaned. Once the busboy is finished, they can mark it as cleaned which lets me know that the table is ready to be used once again.

Customer (Restaurant Patron):

I love going out to eat, but I don't love how slow things can be. When I'm seated, it'd be really great if I could place an order for drinks or appetizers without waiting for a waiter to get to me, especially if the restaurant is incredibly busy. This could really cut down on time spent waiting. If I have a special request immediately after my initial order or need to notify the waiter for anything, it can be rough getting someone's attention. Waiting for a server can be a major inconvenience while dining out. It's usually very difficult to get their attention unless they walk right past you, and if no one ever walks by, you might be waiting extended periods of time before anyone notices you. It can also be difficult to split a bill while out with friends, often times it doesn't get mentioned until the very end of the outing, and it becomes an awkward experience determining with the server what the best way to split the bill is. Some people prefer to split it by itemizing and some by simply splitting the total bill at the end. When this hasn't been properly planned it can be difficult to figure out how to fix it.

How will TurboYums help me? A tablet on the table with a menu that my party could use to place orders immediately after being seated would be a great idea. It would also be a great way to have more interactive menus that could give me more information on each item, since it can take a while to call a server over to my table. The ability to simply press a button in an app to call the server over to my table would be an easy solution to this problem. It would also be fantastic if I could even order on the app!

Server:

Working as a server is an extremely demanding job. Servers have to run around the restaurant taking customers orders, keeping track of which table ordered what, and returning orders. Servers have to interact with chefs/cooks and get food to people right away so the food does not get cold. They also have to enter checks when a customer or group is ready to pay. Since there are many tables in this restaurant, it is somewhat difficult to connect a certain order to a certain table, and also to determine whether a table needs to be cleaned and set up for the next customer. Unfortunately, sometimes a server may have to send a dish back if the customer does not like it or finds something wrong.

How will TurboYums help me? Having an application that could help take care of customer's orders so that they can take their time would be delightful. It would be amazing if the software would be able to send the orders directly to the kitchen and remember which table and which person ordered each item. Sometimes customers need help however I'm not currently in the same area as them due to other job responsibilities. A way for my customers to ping me whenever they need my help would be greatly appreciated! I would also love to be able to receive notifications from the chef once orders are finished getting prepared so the food isn't standing out for so long.

Managers:

Managing a restaurant and a full staff of employees is no easy task, there are many different things that I have to do throughout my day and it can be a little overwhelming at times. It can be difficult to keep track of which employees are on duty, as well as figuring out payroll for every pay period. Another issue with employees is keeping track of them, like whether or not they made it to their shift on time or are in the building and calculating how much to compensate the employees for their time. Any way to reduce the amount of effort that I have to put into any of these tasks would be

immensely helpful. In addition to these issues, a restaurant application must be easily editable and user friendly. I should be able to update the menu with ease to ensure that it matches the items that we are actually serving. Floor plans are also constantly changing, for instance when a larger party comes in, so I must be able to edit the floorplan on the fly to ensure that customers and waiters are able to use a floor plan that matches with the restaurant.
How will TurboYums help me? TurboYums can serve as an employee portal, allowing employees to clock in and out, and will calculate the proper amount to provide to them for compensation. Having this feature will make my life as a manager so much easier, since I won't have to do any of the employee logging or pay calculations - instead, the application will do them for me. The application will also verify the employees location based on IP address or GPS location so that employees can only clock in while they are on site. This way, I will know whether an employee is attempting to clock-in to work before they even arrive to their shift. In addition to this, there are very simple interfaces that will allow you to edit your menu options and quickly readjust the floor plan.

Busboy:
Working as a busboy can be a very time consuming job. At times it's hard to keep track of all the tables currently being used at the restaurant and whether or not they are vacant for me to clean.

How will TurboYums help me? Having an easy way to look at the current status of tables would be a huge advantage. Being able to see which tables are ready to be cleaned and prepared at any given time would be a huge help in efficiently getting stuff ready. Once I'm done, I'd love to be able to update the status of the table to "ready" so it can be used for incoming customers.

2 Glossary of Terms

Application - A software designed to perform a group of coordinated functions, tasks or activities for the benefit of the user.

Busboy - Clears tables, take dirty dishes to the dishwasher and set up the tables to be occupied again.

Chef - Makes food that is requested by the customers.

Cohesion- refers to the notion of a module level “togetherness” viewed at the system abstraction level

Containers - a virtualized isolated environment that allows the existence of multiple user-space instances separate from each other

Cross Platform- An application that can be used on multiple types of devices

Customer - Someone that comes to the restaurant as a guest to be waited on and served food.

Database- A structured set of data stored in TurboYums network where data is accessible to certain user accounts

Docker - Docker is a computer program that performs operating system level virtualization. It runs software packages called containers.

Employee Portal - Allows employees to clock into work from our system.

Filter - To remove meals that have certain ingredients that are unwanted by customers

Floor Layout - Shows how tables in the restaurant are placed and the tables status.

Host/Hostess - Greets incoming customers and assigns seats to each customer.

Menu - A list of food available to be prepared, cooked, and presented, including pictures and ratings.

Manager - Supervises all staff on board and makes sure that everything is in working order.

Manager Account - A user account that allows access to the data collected by the application as well as having extra features that allows management of employees and the restaurant.

MySQL - An open-source relational database management system

NodeJS - A Javascript Runtime environment that allows Javascript execution outside of a browser.

Object-Oriented - Allows systems to be modeled as a set of objects which can be controlled and manipulated in a modular manner

Operating System - System software on which the application will run.

Queue - A first in, first out way of handling orders to ensure that food arrives in order that it is requested.

Rating - A position or standing of something determined by a customer's feedback.

React Native - Allows the use of Javascript and React to develop mobile applications

Reservation - An arrangement made in advance to secure a table.

Restaurant Automation - Makes a restaurant flow more efficiently and more easily. Eliminates certain tasks that servers would normally have to do. Uses devices with preloaded software that manages several tasks which helps eliminate many of the required tasks that are normally done via employees.

Screen- A specific window displayed on TurboYums user interface that provides convenient functionality for the user.

Sequelize - A library in Javascript that makes it easy to manage an SQL database.

Server - Takes orders from customers and delivers food from the kitchen to the customer's table.

SQL - A domain-specific programming language used for designing and managing relational databases.~

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

Tip - A sum of money given to waiter/waitress in addition to the base price.

Translation - To translate from one language into another language

User Interface - The visual aspect of the software that allows user interaction.

3 System Requirements

3.1 Enumerated Functional Requirements

Priority 5 is more vital than Priority 1

Identifier:	Priority:	Requirement:
REQ-1	4	The application will allow customers to select an open table based on an interactive seating chart, after the user has logged in. Once selected, the table is marked as “Occupied”.
REQ-2	5	The application will present an interactive graphical menu once the customer is seated at the table.
REQ-3	5	The application will provide employees with an Employee Portal.
REQ-4	5	The application will keep track of all employee hours for payroll based on clock-in / clock-out reports .
REQ-5	3	The application will notify the busboy when a customer has paid their bill to indicate that they are leaving, so the busboy can clean the table.
**REQ-6	3	The application will allow for customers to conjoin two tables together if desired.
*REQ-7	4	The application will allow for customers to split the bill.
*REQ-8	3	The application will provide the option to make a reservation at the restaurant ahead of time.
REQ-9	4	The application will notify and send the incoming orders to the chef, adding it to the end of the queue in order of time placed.
REQ-10	3	The application will allow a “Take-Out” option where the take-out orders are placed and taken out.
REQ-11	4	The application will allow the customer to sign-up to be a part of a Rewards program.
REQ-12	4	The application should add points earned every time a customer makes a purchase.
REQ-13	4	The application should deduct points and apply them appropriately to the total bill when the customer decides to use them.
REQ-14	4	The application will notify the server when a customer’s food is ready, so they can deliver it to the designated table.
*REQ-15	3	The application will allow the option for the menu to be translated, if needed.

REQ-16	5	The application will allow customers to filter menu items according to dietary restrictions and notify the chef when the order is sent.
*REQ-17	2	The application will allow the customer to rate and leave comments on the order at the end of the meal.
REQ-18	3	The application will allow for servers to mark tables as “Occupied”.
REQ-19	4	The application will allow the customer to alert the server, using a button on the application.
REQ-20	3	The application will allow a table to be marked as “Ready” once the busboy has cleaned it.
REQ-21	5	The application will allow the customer to pay on the spot with a credit-card, if desired.
REQ-22	1	The application will give the option of receipt choice (i.e. paper, e-mail, none).

Modified Functional Requirements

Identifier	Requirement	Comments
**REQ-6	The application will allow for customers to conjoin two tables together if desired.	Requirement changed from being able to move tables around freely to conjoining specific tables.
*REQ-7	The application will allow for customers to split the bill.	This is something that will be part of future work. Additional work on this application may implement these features.
*REQ-8	The application will provide the option to make a reservation at the restaurant ahead of time.	This is something that will be part of future work. Additional work on this application may implement these features.
*REQ-15	The application will allow the option for the menu to be translated, if needed.	This is something that will be part of future work. Additional work on this application may implement these features.
*REQ-17	The application will allow the customer to rate and leave comments on the order at the end of the meal.	This is something that will be part of future work. Additional work on this application may implement these features.

(*) This requirement will not be implemented by demo 2, and is available for future development.

(**) This requirement has been modified since the previous report.

3.2 Enumerated Nonfunctional Requirements

Identifier:	**Priority:**	**Requirement:**
*REQ-23	5	The application should be compatible with iOS and Android operating systems.
REQ-24	3	The application should be easy for the typical person to use.
REQ-25	3	The application should be aesthetically pleasing.
REQ-26	2	The application should be able to startup and shut down quickly.
REQ-27	3	The application should have smooth transitions from page to page.
REQ-28	4	The application should support a functioning restaurant.

Modified Nonfunctional Requirements

Identifier	Requirement	Comments
*REQ-23	After logging in or continuing as a guest, and selecting the “Dine-in” option, the customer will be presented with a floor layout that designates the available tables as white and the unavailable tables as grey, and will allow the customer to select an available table to sit at.	We have removed this requirement and reassigned the identifier to represent cross platform compatibility.

(*) This requirement will not be implemented by demo 2, and is available for future development.

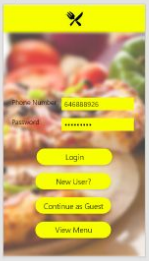
(**) This requirement has been modified since the previous report.

3.3 On-Screen Appearance Requirements

Customer Interface

Splash

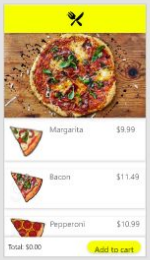
Login



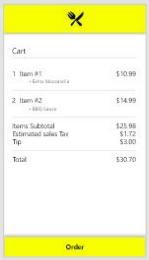
Dine-In/Checkout

Tables

Menu



Summary

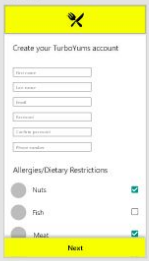


Order Progress

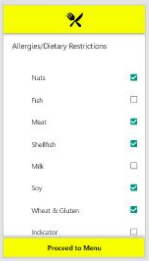
Payment

Receipt/Feedback

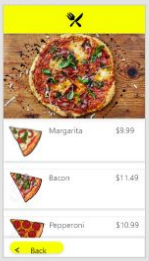
SignUp



Allergies

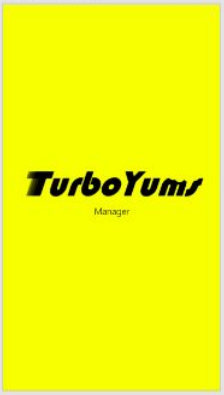


View Menu

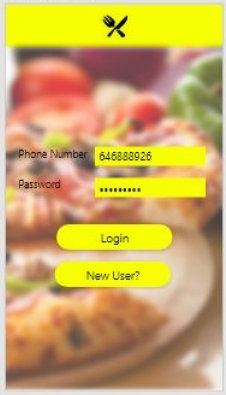


Manager Interface

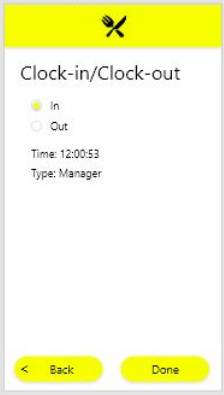
Splash_Manager



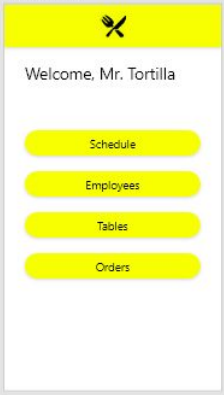
Login_Manager



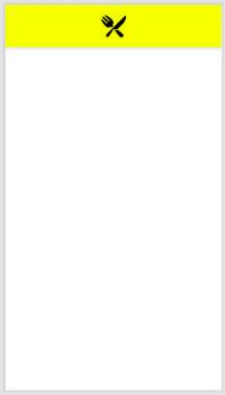
Clock-in/Clock-out



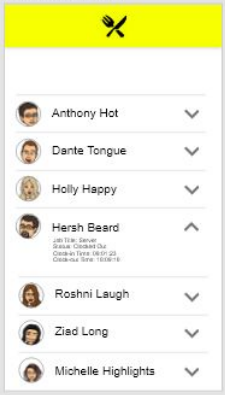
Manager_List



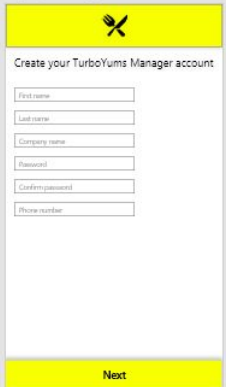
Schedule_Manager



Employees_manager

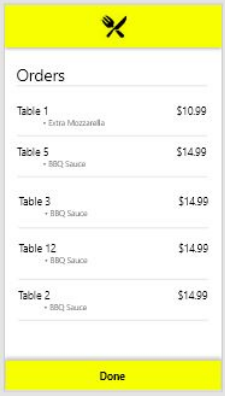


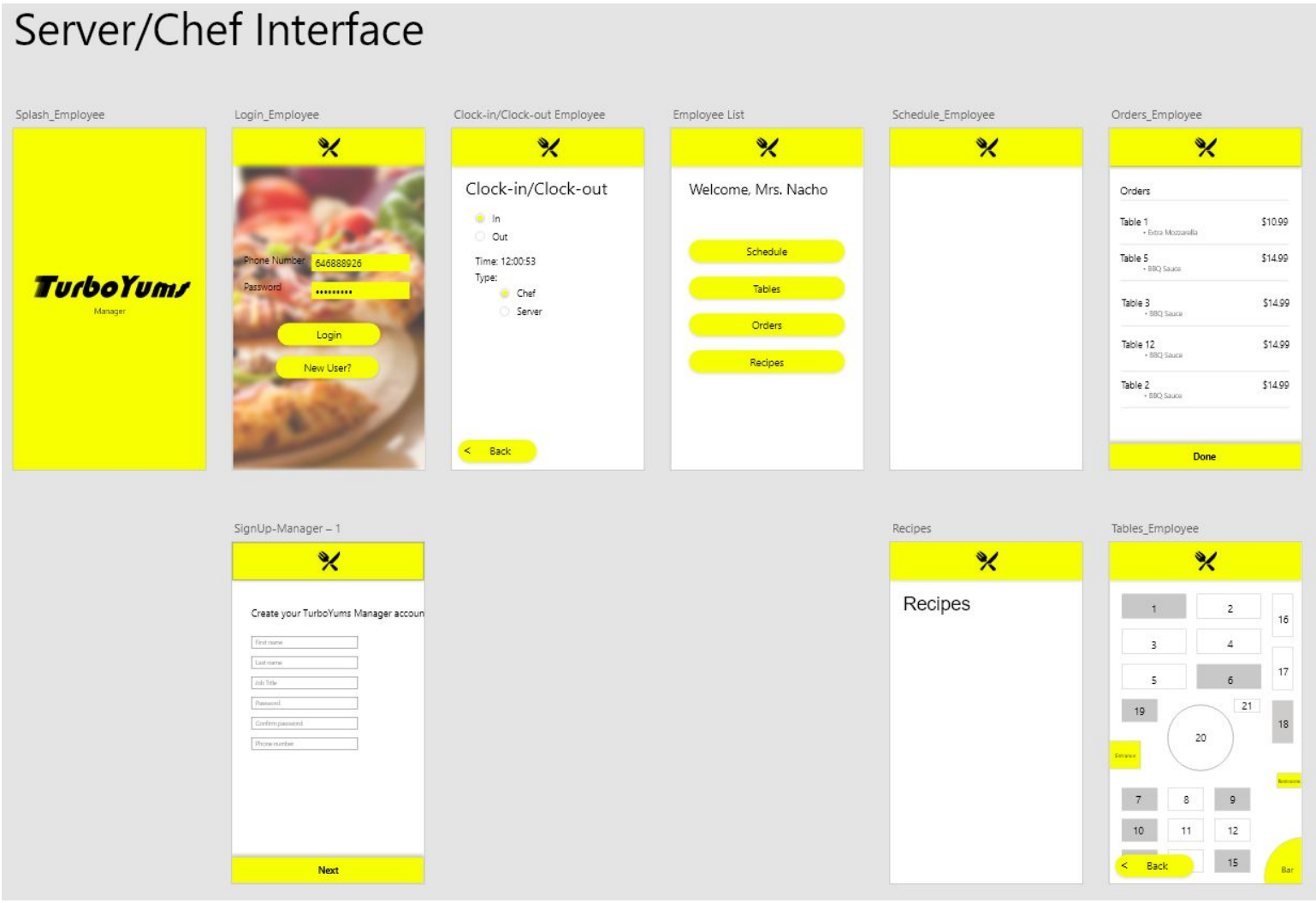
SignUp-Manager



Tables_Manager

Orders_Manager





Customer Interface Requirements

Identifier:	Priority:	Requirement:
CSREQ-1	4	Customer gets to choose among “Log In”, “Continue As Guest”, and “Sign Up”.
CSREQ-2	4	Clicking on the “Login” button takes the customer to the Login Page where he/she inputs her account username and password. If entered correctly, the user can next see the Tables page.
CSREQ-3	5	Clicking on the “Continue As Guest” button takes customer to a page where he/she gets to choose between Dine-In and Take-Out.
*CSREQ-4	5	Clicking on the “View Menu” button as guest takes the customer to a Menu Page **without** “Add to Cart” option to prevent checking out without either logging in or continuing as a guest.
CSREQ-5	4	Clicking on Dine-In takes the customer to the Tables page so that he/she can choose a

		table to sit at.
CSREQ-6	5	Clicking on a ready table (colored in green) takes the customer to the Menu Exclusion page where he/she can filter out the menu according to his/her dietary restrictions/allergies. The user is also allowed to skip this page simply by pressing on Continue. Clicking on an occupied or dirty table (colored in red or coral) is forbidden.
CSREQ-7	4	Clicking on the Take-Out button will take the customer straight to the Menu page.
CSREQ-8	5	After the confirmation of the customers Dietary Restriction/Allergies, the customer can head to the Menu page which is already filtered out.
CSREQ-9	3	After selection from the menu, the customer gets a summary/overview of her selections in the summary page and he/she can proceed to Progress page.
CSREQ-10	1	In the Progress Page, the customer can see the progress of his/her order and Order More or Checkout/Pay.
CSREQ-11	5	Clicking on "Payment" takes customer to the Payment page where he/she can either pay using a Credit/Debit Card or Cash.
CSREQ-12	2	After payment, the Receipt/Feedback page appears where the customer can choose how they want their receipt or not and rate his/her experience of ordering via the app.
**CSREQ-13	3	Clicking on the Home page, the customer can go back to the dine-in/take-out page where she can place another order - let's say, for takeout.
CSREQ-14	5	Clicking on the "Sign Up" button takes the customer to the Sign Up screen where he/she is able to input the required information and then use the account to log in.

Modified Customer Interface Requirements

Identifier	Requirement	Comments
*CSREQ-4	Clicking on the "View Menu" button as guest takes the customer to a Menu Page **without** "Add to Cart" option to prevent checking out without either logging in or continuing as a guest.	We changed the interface flow of events. There is no longer a view menu button. (removed).
**CSREQ-13	Clicking on the Home page, the customer can go back to the dine-in/take-out page where she can place another order - let's say, for takeout.	Home button is not implemented, the customer simply presses back to return to that page

(*) This requirement will not be implemented by demo 2, and is available for future development.

(**) This requirement has been modified since the previous report.

Manager Interface Requirements

Identifier:	**Priority:**	**Requirement:**
MSREQ-1	5	Owner/Manager logs in with his/her username and password.
**MSREQ-2	4	The default page of the Manager Interface holds options of Employee Info, Orders, Tables, Menu, and Logout.
**MSREQ-3	5	The bar at the bottom of the screen will have options to navigate to the following pages: Employees Info, Orders, Tables, Menu, and Logout.
MSREQ-4	5	Employee Info takes the owner to view a list of all his employees and their information.
MSREQ-5	4	Clicking on Orders takes the owner to the Orders page where he/she can see a list of active/past orders and their statuses.
MSREQ-6	4	Logout takes the owner back to the Login Page.
***MSREQ-7	5	Menu takes the manager to either view the current menu and/or update the current menu.

(*) This requirement will not be implemented by demo 2, and is available for future development.

(**) This requirement has been modified since the previous report.

(***) This requirement was added to our manager interface.

Modified Manager Interface Requirements

Identifier	Requirement	Comments
**MSREQ-2	Default page of Tables appears after login to show which tables are occupied (grey), unclean (red) or ready to go (white). The manager will also be allowed to click on the edit button on the top of this screen, in order to keep the table layout updated as needed.	We have changed this so that default screen allows options to navigate to: Employee Info, Orders, Tables, and Logout.

**MSREQ-3	The bar at the bottom of the screen will have options to navigate to the following pages: Employees Info, Orders, Tables, Menu, and Logout.	Navigation bar was not implemented, we instead use central menus to navigate between the different pages

Employee Interface Requirements

Identifier:	**Priority:**	**Requirement:**
ESREQ-1	5	Employee logs in with his/her username and password.
ESREQ-2	5	The Employee clocks in and the time, location of clock-in, IP address, and current clocking status gets logged into the server under the employee's account.
ESREQ-3	4	Default page of Tables appears after login to show which tables are ready (green), occupied (red), or dirty (coral).
**ESREQ-4	4	The bar at the bottom of the screen will have options to navigate to the following pages: Orders, Tables, Clock Out and Logout. These pages are the same as the one's under the Owner/Manager mode except for the Tables page and the Menu option for which the employee wouldn't have the authorization to edit table layout or update the menu.
ESREQ-5	4	The Employee gets to Clock Out. This is only possible if the employee has previously clocked in. Otherwise, the employee will be prompted with an appropriate error message.

Modified Employee Interface Requirements

Identifier	Requirement	Comments
**ESREQ-4	The bar at the bottom of the screen will have options to navigate to the following pages: Orders, Tables, Clock Out and Logout. These pages are the same as the one's under the Owner/Manager mode except for the Tables page and the Menu option for which the employee wouldn't have the authorization to edit table layout or update the menu.	Navigation bar was not implemented, we instead use central menus to navigate between the different pages

4. Functional Requirements Specification

4.1 Stakeholders

There are many stakeholders who have an interest in this system, and ultimately its success:

i. Restaurant Owners - Have a direct interest in using the system as it will help optimize efficiency in the restaurant and provide better service for patrons.
ii. Employees - Have an interest in the system since it will result in a more streamlined process while working making their job much easier.
iii. Restaurant Visitors - The system will result in an enriched dining experience for the restaurant visitors, as they will be interacting with and using the system.
iv. Developers - Have an interest in working to design and implement solutions to create and improve the system.

4.2 Actors and Goals

Initiating Actors

Actor	**Role**	**Goal**
Customer	The customer is a restaurant visitor who may choose to dine in or take out food, view the menu, order a meal, eat, and pay for service.	The goal of the customer is to have an excellent dining experience with minimal wait times and smooth service.
Guest	The guest is a customer that does not have an account and chooses to opt out of the features that come with an account, other than that a guest is the same of a customer.	The goal of the guest is to have an excellent dining experience with minimal wait time and smooth service.
Employee	The employee is any type of worker at the restaurant, except for the manager.	The goal of the employee is to provide an excellent dining experience for the customers.
Manager	The manager is the employee who has the additional responsibility of managing all of the needs of the restaurant.	The goal of the manager is to manage employees and scheduling, keeping track of inventory, and monitoring revenue and losses while they also ensure that restaurant customers are accounted for.

Participating Actors

Actor	**Role**
Busboy	The busboy is the employee responsible for cleaning the dishes and tables, and maintaining overall cleanliness of the restaurant. The busboy can see from the database that customers have left their

	table, marking the table as dirty. Once the table is cleaned, it can be marked as clean.
Chef	The chef is the employee responsible for cooking and preparing the food that is ordered by a customer. The chef receives a queue of orders, preparing them in the order they come in. The customer is updated on the status of their order (submitted, preparing, ready).
Database	The database is a system that records a customer's order, table selection, and menu options. It essentially acts as persistent storage for all information that needs to be stored for our application to function.
Host/Hostess	The host/hostess is the employee responsible for greeting incoming customers and assigning seats to them. In the event that the guests have already selected a table, the host/hostess will escort them to the table. The host/hostess can see from the database when a table is marked as clean. If the customer has not yet selected a table, the host can seat him and mark the table as occupied.
Servers	The server is the employee responsible for taking orders from customers and sending them to the kitchen queue, as well as serving the food when it is ready. The waiter/waitress receives notifications that a meal is ready, so that they can pick it up and serve it. They are also notified which table number to serve it to, and get notified when a customer needs additional assistance.

4.3 Use Cases

4.3.1 Casual Description

***UC-1: Reservation** - Allows customers to make reservations online before they come to the restaurant to reserve a table.

Derivations: REQ-8

****UC-2: Payment** - Allows customers to split the bill, choose option of receipt (email, paper) and allow customers to pay on the spot with a credit-card reader (if desired).

Derivations: REQ-5, REQ-7, REQ-12, REQ-13,REQ-21, REQ-22, CSREQ-10, CSREQ-11, CSREQ-12

UC-3: View Menu - Allows customers to view the entire menu with our system.

Derivations: REQ-2, REQ-15, REQ-16, CSREQ-4, CSREQ-8

UC-4: Meal Prep - Allows chefs to be updated on the status and requirements of their orders.

Derivations: REQ-9, REQ-16, ESREQ- 4

***UC-5: Rate Food** - Allows customers to rate their overall restaurant experience.

Derivations: REQ-17, CSREQ-12

****UC-6: Food Filters** - Allows customers to filter out food according to dietary restrictions or preferences.

Derivations: REQ-16, CSREQ-8

UC-7: Clocking In/Out - Allows employees to clock in when they come to work and clock out when they go on break/go home.

Derivations: REQ-3, REQ-4, ESREQ-1, ESREQ-2, ESREQ-4, ESREQ-5, MSREQ-4

UC-8: Serving - Allows waiters/waitresses to keep track of their current orders and customer status.

Derivations: REQ-6, REQ-14,REQ-18, REQ-19, REQ-20, ESREQ-4

UC-9: Placing an Order - Allows servers or customers to send their orders to the kitchen.

Derivations: REQ-9,ESREQ-4, CSREQ-10

UC-10: Table Marking - Allows servers to mark a table as occupied when customers sit down and allows busboys to mark a table as ready once it is cleaned.

Derivations: REQ-1, MSREQ-5, ESREQ-3, ESREQ-4

UC-11: Earning Rewards - When the customer makes a purchase they earn rewards. With a certain amount of rewards the customer can receive discounts or a free drink as an example.

Derivations: REQ-11, REQ-12, REQ-13

UC-12: Redeeming Rewards - When the customer makes a purchase, they can use points earned toward the cost of their meal.

Derivations: REQ-12, REQ-13

UC-13: Take-Out - An option for the customers that allows them to order food from the restaurant for pick up to take home.

Derivations: REQ-10, CSREQ-7

UC- 14: Table Selection - Allows the customer to select a table after either logging in or continuing as a guest if dining in by viewing the floor plan, which will have the status of the tables shown and allowing them to select from all of the open tables.

Derivations: REQ-1, REQ-18, CSREQ-5

UC-15: **Floor Plan Status** - Allows the manager and the servers to change the status of a table from Ready to Occupied to Dirty and back to Ready as desired. Tables conjoined together will change status in unison and have matching Order IDs.

Derivations: REQ-6, MSREQ-2, MSREQ-3, MSREQ-5

UC-16: **Login** - Allows users to login which will dictate which interfaces (Employee or Customer) they will view and the different potential functions that they may access.

Derivations: REQ-3, REQ-11, CSREQ-1, CSREQ-2, CSREQ-3, MSREQ-1, ESREQ-1

UC-17: Create Account - Allows users to create an account so that they may get the benefits of being an account holder.

Derivations: REQ-11, CSREQ-1

***UC-18: Translation** - Allows user to have the option to translate the menu into another language, so they do not need a translator.

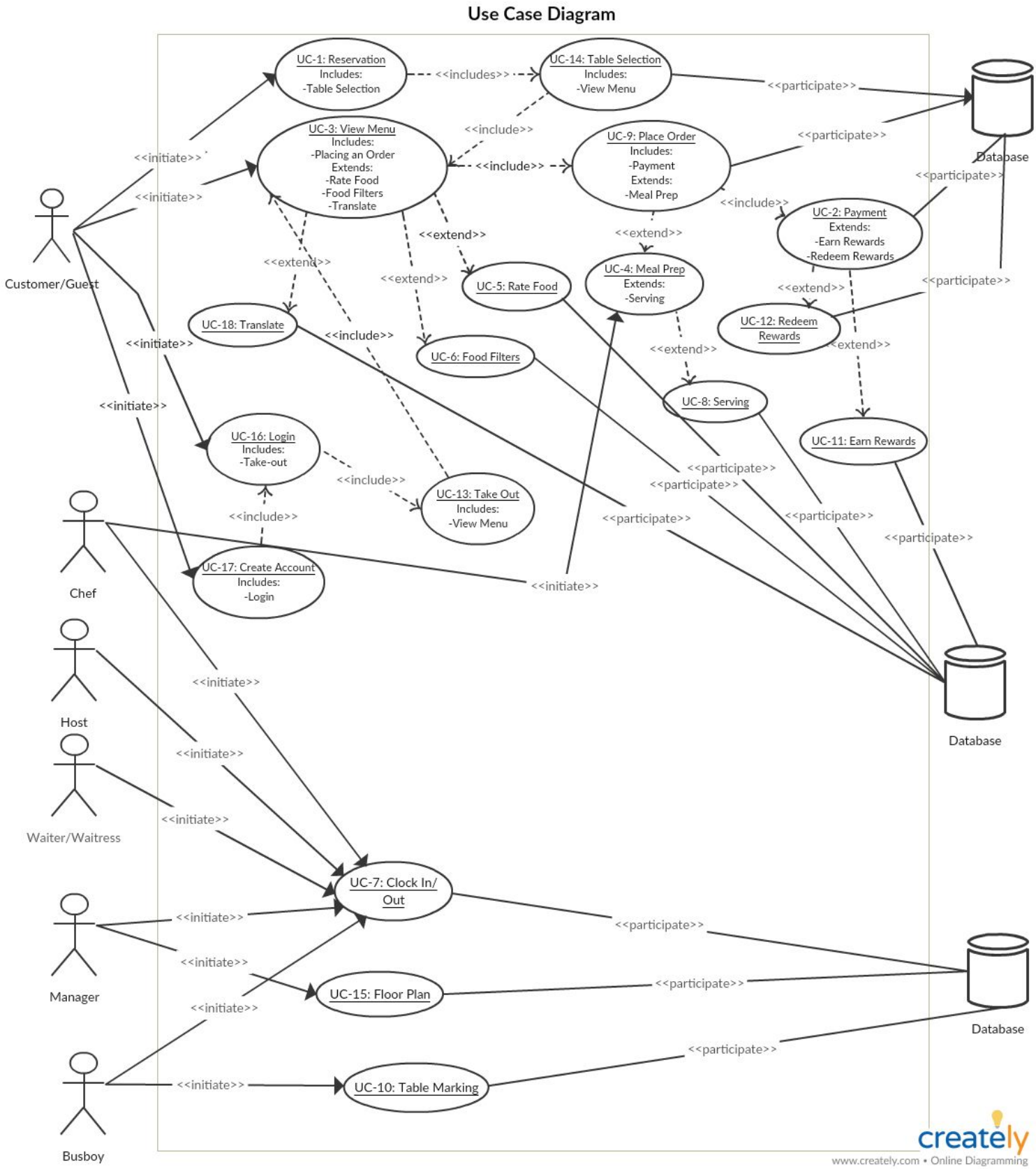
Derivations: REQ-15

****UC-19: Menu Changes** - Allows the manager to easily be able to update and edit the menu.

Derivations: MSREQ-7

Identifier	Requirement	Comments
*UC-1	**Reservation** - Allows customers to make reservations online before they come to the restaurant to reserve a table.	This is something that will be part of future work. Additional work on this application may implement these features.
**UC-2	**Payment** - Allows customers to split the bill, choose option of receipt (email, paper) and allow customers to pay on the spot with a credit-card reader (if desired).	Splitting the bill is not yet being implemented, and also credit card payment is handled with a stored/entered card, not a reader.
*UC-5	**Rate Food** - Allows customers to rate food items on the menu and view their current ratings.	This is something that will be part of future work. Additional work on this application may implement these features.
*UC-18	**Translation** - Allows user to have the option to translate the menu into another language, so they don't need a translator.	Translation was implemented in the backend however, the front end interface for it is left for future work.
**UC-19	**Menu Changes** - Allows the manager to easily be able to update and edit the menu.	Managers will be able to add new items to the menu and edit the existing menu.
**UC-6	**Food Filters -** Allows customers to filter out food according to dietary restrictions or preferences	

4.3.2 Use Case Diagram



In the use case diagram, all the relationships between the Actors, Participating Actors, Use Cases, and the database are displayed. The Actors and Participating Actors are Customer/Guest, Chef, Host, Waiter/Waitress, Manager, Busboy. Some of the use cases are initiated by an actor. Some use cases are extended from other use cases that are related use cases. Use cases are included with use cases that they branch off from. Uses cases communicate with the database by participating.

4.3.3 Traceability Matrix

Req't	PW	1	2	3	4	5	6	7	8	9	10	11	12	3	14	15	16	17	18	19
REQ-1	4										X				X					
REQ-2	5			X																
REQ-3	5							X									X			
REQ-4	5							X												
REQ-5	3		X																	
REQ-6	3								X							X				
REQ-7	4		X																	
REQ-8	3	X																		
REQ-9	4				X					X										
REQ-10	3													X						
REQ-11	4											X					X	X		
REQ-12	4		X									X	X							
REQ-13	4		X									X	X							
REQ-14	4								X											
REQ-15	3			X															X	
REQ-16	5			X	X		X													
REQ-17	2					X														
REQ-18	3								X						X					
REQ-19	4								X											
REQ-20	3								X											
REQ-21	5		X																	
REQ-22	1		X																	
CSREQ-1	4																X	X		
CSREQ-2	4																X			
CSREQ-3	5																X			
CSREQ-4	5			X																
CSREQ-5	4														X					
CSREQ-6	5																			
CSREQ-7	4													X						
CSREQ-8	5			X			X													

CSREQ-9	3																			
CSREQ-10	1		X							X										
CSREQ-11	5		X																	
CSREQ-12	2		X			X														
CSREQ-13	3																			
CSREQ-14	5			X																
MSREQ-1	5																X			
MSREQ-2	4															X				
MSREQ-3	5															X				
MSREQ-4	5							X												
MSREQ-5	4										X					X				
MSREQ-6	4																			
MSREQ-7	5																			X
ESREQ-1	5							X									X			
ESREQ-2	5							X												
ESREQ-3	4										X									
ESREQ-4	4				X			X	X	X	X									
ESREQ-5	4							X												

4.3.4 Fully Dressed Description

UC-2: Payment
Related Requirements: REQ-5, REQ-7, REQ-12, REQ-13, REQ-22, CSREQ-10, CSREQ-11, CSREQ-12
Initiating Actor: Customer
Actor's Goal: To pay for a completed order
Participating Actors: Database Waiter/Waitress
Preconditions: The user has loaded the system The user has logged in as a Customer and has the menu opened
Postconditions: The user is prompted on payment method The user is prompted on receipt option
Flow of Events for Main Success Scenario: 1. → The customer clicks on the "Checkout" button on the menu screen. 2. ← The system displays the payment menu. 3. ← The system displays payment method options. 4. → The customer chooses the "Cash" button in order to pay by cash. 5. ← The system prompts the customer with receipt options. 6. → The customer chooses the "Print Receipt" button. 7. ← The system notifies the waiter/waitress to print the receipt and bring it back to the customer.
Flow of Events for Alternate Success Scenario: 1. → The customer clicks on the "Checkout" button on the menu screen. 2. ← The system displays the payment menu. 3. ← The system displays payment method options. 4. → The customer chooses the "Credit" button in order to pay by credit. 5. ← The system prompts the customer with receipt options. 6. → The customer chooses the "E-Mail Receipt" button. 7. ← The system prompts the user for an email address. 8. → The customer types in an email address and receives a receipt through email.

UC-3: View Menu
Related Requirements: REQ-2, REQ-15, REQ-16, CSREQ-4, CSREQ-8
Initiating Actor: Customer
Actor's Goal: To view the menu
Participating Actors: Database
Preconditions: The user has loaded the system
Postconditions: The user is shown the available menu
Flow of Events for Main Success Scenario: 1. ← The system gives options for "Login", "Continue As Guest", and "Create Account". 2. → The customer selects to login to their account. 3. ← The system prompts the customer to login with their information. 4. → The customer inputs information for account. 5. ← The system gives options for "Dine-In" or "Take-Out". 6. → The customer selects option, "Dine-In". 7. ← The system allows the customer to select a table from the seating chart. 8. → The customer sits at the table. 9. ← The system provides options to select Dietary Restrictions. 10. → The customer chooses to not filter the menu. 11. ← The system displays the available menu.
Flow of Events for Alternate Success Scenario: 1. ← The system gives options for "Log-In", "Sign-Up", or "Continue as Guest". 2. → The customer selects "Continue as Guest". 3. ← The system gives options for "Dine-In" or "Take-Out". 4. → The customer selects option, "Dine-In". 5. ← The system allows the customer to select a table from the seating chart. 6. → The customer sits at the table. 7. ← The system provides options to select Dietary Restrictions. 8. → The customer chooses to not filter the menu. 9. ← The system displays the available menu.

UC-7: Clocking In/Clocking Out

Related Requirements:
REQ-3, REQ-4, ESREQ-1, ESREQ-2, ESREQ-4, ESREQ-5, MSREQ-4

Initiating Actor:
Employee

Actor's Goal:
To accurately record the amount of time that they spend while on the job and to make sure the employee is actually at the restaurant when they clock-in and out

Participating Actors:
All Employees (i.e. Server, Busboy, etc.)

Preconditions:
The user has downloaded the application

Postconditions:
The user has successfully clocked in/out
The database logs the information for future use

Flow of Events for Main Success Scenario:

1. → The employee opens the application and chooses to login as a regular employee.
2. ← The system prompts the employee for their login information.
3. → The employee logins in with their information.
4. ← The system presents the employee with an Employee Portal.
5. → The employee selects the clock-in button.
6. ← The system keeps track of the employee's clock-in data.
7. ← The system verifies that the employee is actually in the restaurant by checking the IP address and GPS coordinates.
8. → When the employee has completed the shift for the day, the employee will select the clock-out button.
9. ← The system keeps track of the employee's clock-out data.
10. ← The system verifies that the employee is actually in the restaurant by checking the IP address and GPS coordinates.

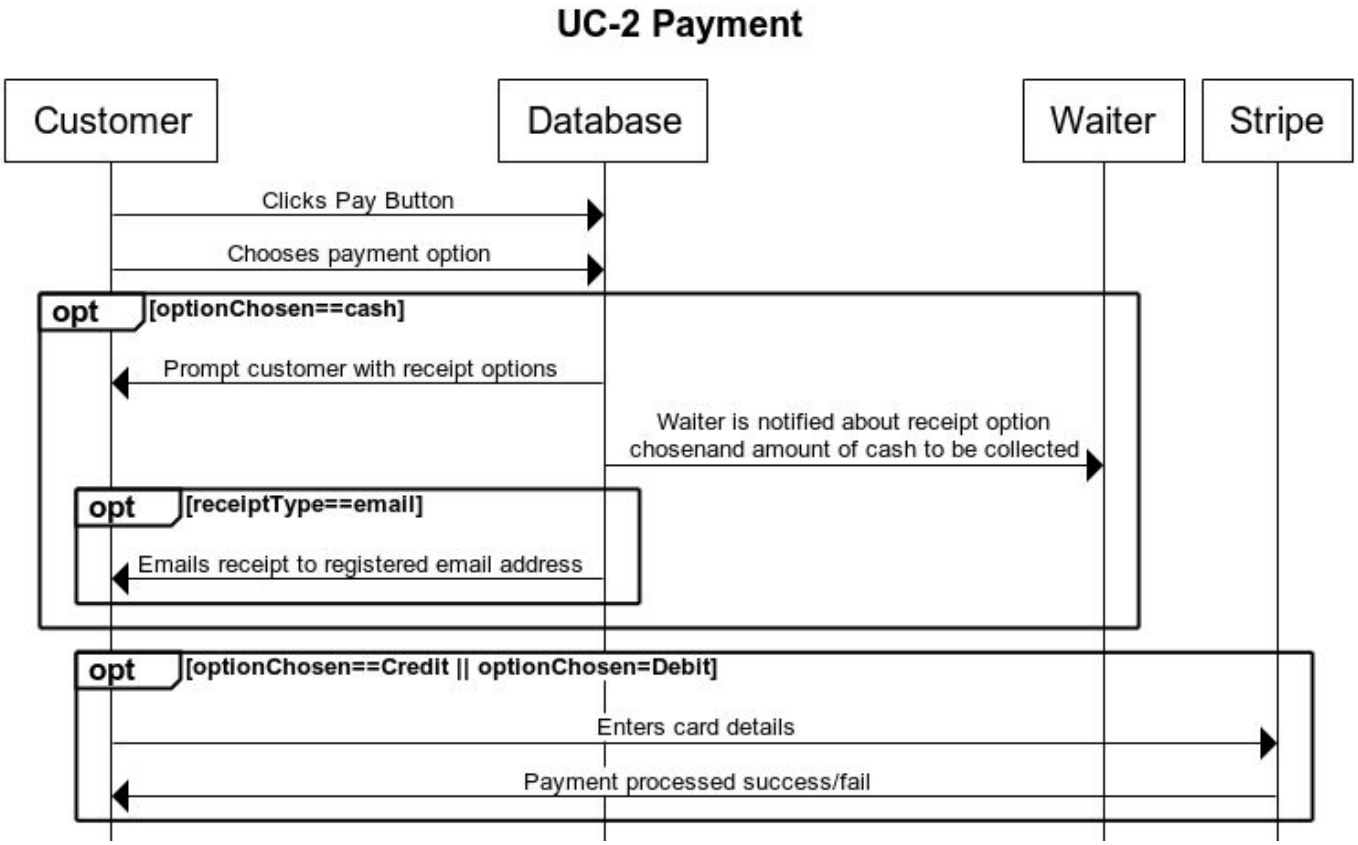
Flow of Events for Alternate Success Scenario:

1. → The employee opens the application and chooses to login as a manager.
2. ← The system prompts the manager for their login information.
3. → The manager logins in with their information.
4. ← The system presents the manager with a manager interface.
5. → The manager selects the clock-in button.
6. ← The system keeps track of the manager's clock-in data.
7. ← The system verifies that the manager is actually in the restaurant by checking the IP address and GPS coordinates.
8. → When the manager has completed the shift for the day, the manager will select the clock-out button.
9. ← The system keeps track of the manager's clock-out data.
10. ← The system verifies that the manager is actually in the restaurant by checking the IP address and GPS

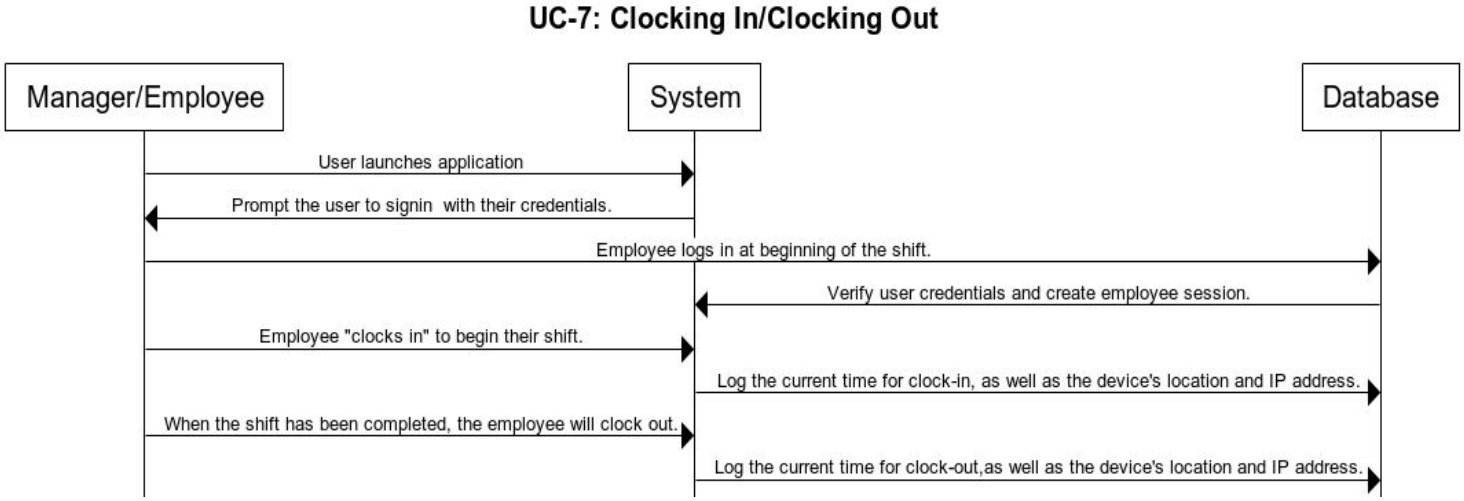
coordinates.

UC-11: Earning Rewards
Related Requirements: REQ-11, REQ-12, REQ-13
Initiating Actor: Customer
Actor's Goal: To receive coupons/discounts based off the amount of purchases at restaurant
Participating Actors: Database
Preconditions: The user has loaded the system The user intends to make a purchase
Postconditions: The user has earned an appropriate amount of points for how much was spent
Flow of Events for Main Success Scenario: 1. ← The system asks the customer at the beginning to login/create account. 2. → The database will retrieve the user's current point balance. 3. ← The customer chooses all the food items and confirms order. 4. → The customer finishes payment. 5. ← The database updates points based on transaction.
Flow of Events for Alternate Success Scenario: 1. ← The system prompts the customer at the beginning to login/create account. 2. → The database will retrieve the user's current point balance. 3. → The customer confirms order. 4. → Customer does not complete payment. 5. ← Database returns that transaction is incomplete, points not added.

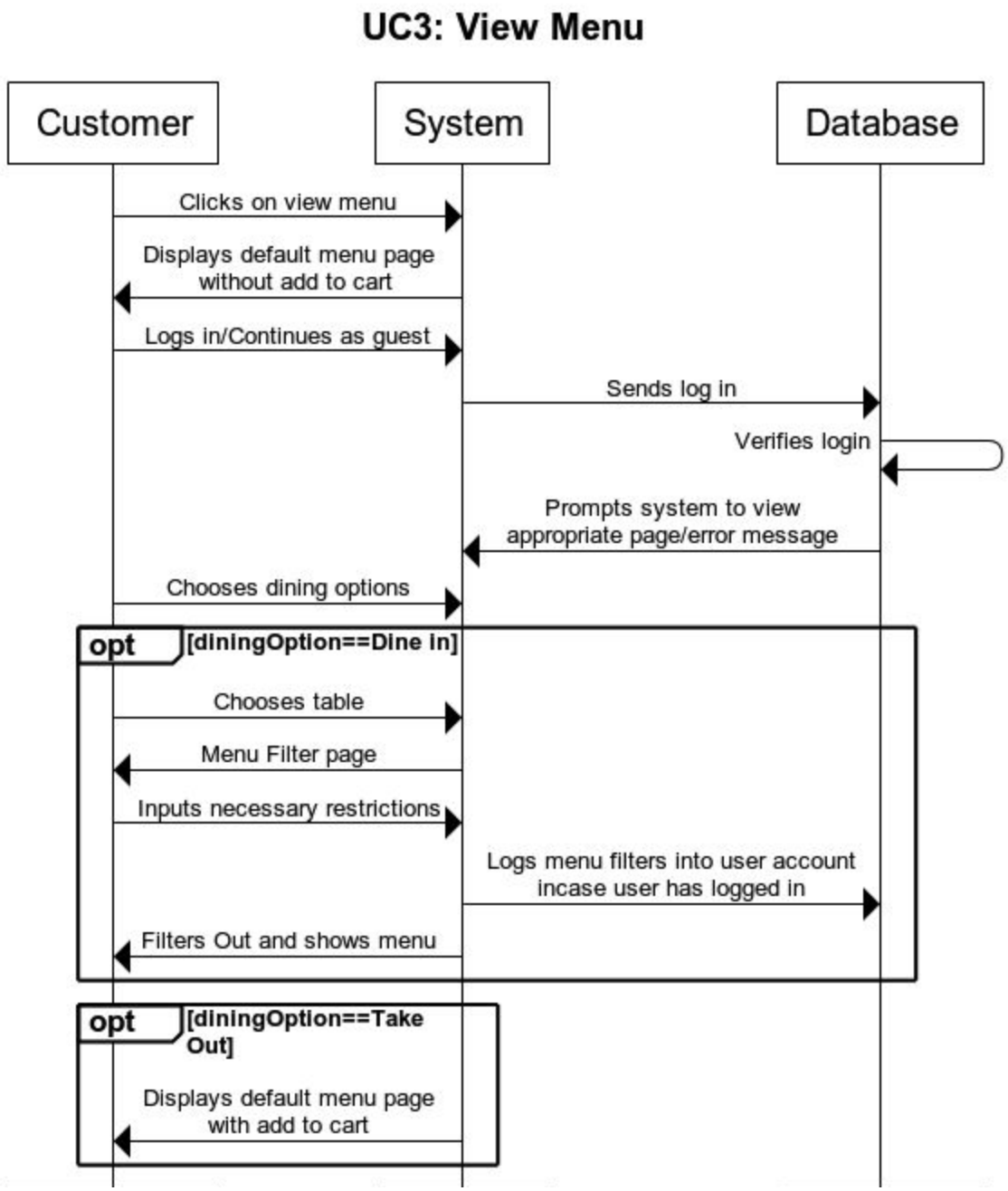
4.4 System Sequence Diagrams



For UC-2: Payment, The customer presses the pay button and chooses their payment option, and the Database record information that it needs to. If payment goes through cash, the waiter is notified and will collect cash. Alternatively, if the option chosen was a card, the information is sent to the Stripe API for processing, Stripe will return information regarding the success of the payment.



For UC-7: Clocking In/Clocking Out, the information recorded when clocking in and out is carried through to the database for storage by the System.



For UC-3: View Menu, The Customer prompts the System to display the menu, the system requests the database to fetch items which match any filter, and the items are displayed to the user by the system.

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 interacting through the defined API (application programming interface)	1
Average	The actor is a person interacting through a protocol or through a text based user interface	2
Complex	The actor is a person interacting with the user interface	3

Actor Name	Description of Relevant Characteristics	Complexity	Weight
Customer	Customer is interacting with the system/user interface	Complex	3
Guest	Same as Customer	Complex	3
Employees	Same as Customer	Complex	3
Managers	Same as Customer	Complex	3
Chef	Same as Customer	Complex	3
Waiter/Waitress	Same as Customer	Complex	3
Busboy	Same as Customer	Complex	3
Server	Same as Customer	Complex	3
Database	Database is another system interacting through a protocol	Average	2

UAW(TurboYums) = 1* Average + 9 * Complex = 1*2 + 9*3 = 29

5.2 Unadjusted Use Case Weight

Actor Type	Description of how to recognize the actor type	Weight
Simple	- Simple user interface - Up to one participating actor (plus initiating actor) - Number of steps for the success scenario : <=3	5
Average	- Moderate interface design - Two or more participating actors - Number of steps for the success scenario : 4 to 7	10
Complex	- Complex user interface or processing - Three or more participating actors - Number of steps for the success scenario : >=7	15

Use Case	Description	Category	Weight
*UC-1: Reservation	Simple user interface. 6 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
UC-2: View Menu	Simple user interface. 5 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
**UC-3: Payment	Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
UC-4: Meal Prep	Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
*UC-5: Rate Food	Simple user interface. 1 step for the main success scenario. Two participating actors (Customer, Database)	Simple	5
UC-6: Food Filters	Simple user interface. 8 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
UC-7: Clocking In/Out	Simple user interface. 1 step for the main success scenario. Two participating actors (Employee, Database)	Simple	5
UC-8: Serving	Simple user interface. 2 steps for the main success scenario. Two participating actors (Server, Database)	Simple	5
UC-9: Placing an Order	Simple user interface. 2 steps for the main success scenario. Two participating actors (Customer, Database)	Simple	5
UC-10: Table Marking	Simple user interface. 2 steps for the main success scenario. Two participating actors (Server, Database)	Simple	5

UC-11: Earning Rewards	Simple user interface. 2 steps for the main success scenario. Two participating actors (Customer, Database)	Simple	5
UC-12: Redeeming Rewards	Simple user interface. 2 steps for the main success scenario. Two participating actors (Customer, Database)	Simple	5
UC-13: Take-Out	Simple user interface. 6 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
UC-14: Table Selection	Simple user interface. 10 steps for the main success scenario. Two participating actors (Customer, Database)	Average	10
UC-15: Floor Plan Adjustment	Complex user interface. Two participating actors (Server, Manager)	Complex	15
UC-16: Login	Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer/Employee, Database)	Simple	5
UC-17: Create Account	Simple user interface. 4 steps for the main success scenario. Two participating actors (Customer, Database)	Simple	5
*UC-18: Translation	Simple user interface. 2 steps for the main success scenario. Two participating actors (Customer, Database)	Simple	5
***UC-19: Menu Changes	Simple user interface. 4 steps for the main success scenario. Two participating actors (Manager, Database)	Average	10

UUCW(TurboYums) = 10* Simple + 8 * Average + 1*Complex = 10 * 5 + 8 * 10 + 1 * 15 = 145

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	Ease of use is very important to every user	0.5	5	0.5*5=2.5
T4	Easy to make changes	1	3	1*3=3

	required (different restaurants)			
T5	No unique training needs	1	0	1*0=0
T6	No direct access for third parties	1	0	1*0=0
T7	No requirement for reusability	1	1	1*1=1
T8	Concurrent use is required	1	4	1*4=4
Technical Factor Total:				16.5

TCF = 0.6 * 0.01* Technical Factor Total = 0.6+0.01 * 16.5 = 0.765

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=10
E6	Programming language or average difficulty will be used	-1	3	-1*3=-3
Environmental Factor Total:				12

ECF= 1.4-0.03*12=1.04

5.5 Use Case Points

UCP= UUCP* TCF* ECF

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

UUCP= UAW + UUCW = 29 + 145 = 174

TCF=0.765

ECF=1.04
For the case study, the final UCP is the following:
UCP= 174*0.765*1.04 = 138.43 or 138 use case points

5.6 Duration

Duration = UCP * PF = 138 * 28 = 3,864 hours

6 Domain Analysis

6.1 Domain Model

6.1.1 Concept Definitions

Responsibility	Type	Concept
R1: Coordinate activity between the customer, chef, server, busboy, etc.	D	Communicator
R2: Prompt the customer to select a table	D	Table Status
R3: Display the options for the customer, waiter, chef and manager respectively	D	Interface
R4: Queue incoming orders for the chefs to prepare	D	Order Queue
R5: Store employee login information, along with hours worked	K	Employee Profile
R6: Prevent invalid table selections	D	Table Status
R7: Handle payment processing	D	Payment System
R8: Store customer Reward points based upon previous orders	K	Customer Profile
R9: Display change of table status when a customer selects a table, leaves, and when a busboy cleans a table	D	Table Status
R10: Displays current orders to serve	K	Serving
R11: Store customer login information	K	Customer Profile
***R12:** Protect reserved tables from being selected before reservation time	D	Table Status
***R13:** Display favorites, top rated and most ordered under customer/user profile information	D	Customer Profile
R14: Store the customer order in the database	K	Customer Profile
R15: Manage interactions with the database	K	DB Connection

****R16:** Display filtered menu using user input from the Allergies/Dietary Restriction	D	Controller
R17: Allow the customer to conjoin tables of the restaurant to accommodate needs	D	Floorplan

Identifier	Requirement	Comments
***R12**	Protect reserved tables from being selected before reservation time	We are no longer providing the option to make a reservation, so we will not need to protect these tables.
***R13**	Display favorites, top rated and most ordered under customer/user profile information	We are no longer displaying individual item ratings so a top rated, favorites and most ordered display has been omitted.
****R16**	Display filtered menu using user input from the Allergies/Dietary Restriction	Food filtering has been implemented in a separate branch and presented in our second demo, however, there were merge conflicts that interfered with implementing it into the master.

(*) This requirement will not be implemented by demo 2, and is available for future development.

(**) This requirement has been modified since the previous report.

6.1.2 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 app.	User Action
Communicator ⇔ DB Connection	Modify or insert data into the database.	UpdateDB
Communicator ⇔ Order Queue	Send order and queue to the database	QueryDB
Controller ⇔ Food Status	Allow the user (staff) to update or view the food order status.	Update Food Status View Food Status
Controller ⇔ Table Status	Allow user to view table status	View Table Status
Controller ⇔ Payment System	Allow user to complete the payment	Make Payment

Payment System ⇔ DB Connection	Store payment record in the database	Record Payment
Interface ⇔ DB Connection	Get the data from the database for the user	QueryDB
Customer Profile ⇔ DB Connection	Stores earned rewards/points in the database	Reward System
Table Status ⇔ Interface	Displays the current table layout with the status of the tables	Display
Employee Profile ⇔ Interface	Displays the employee option	Display
Payment System ⇔ Customer Profile	Make updates to point balance for the user.	Update Rewards
Floorplan ⇔ Controller	Allow the user (manager) to adjust a table in the layout	Floorplan Change

6.1.3 Attribute Definitions

Concept	Attribute	Description
Customer Profile	accountUsername	Associated username of the customer (email). Guest account is assigned if no account.
	accountPassword	Password of user account
Interface	confirmOrder	Allows the user to confirm order after choosing food items
	receipt	Allows the user to choose which way they would like to receive the receipt (email, text, paper)
	rateMeal	Allows the user to rate a meal, and then have that rating stored and displayed
	tableStatus	Provides the user with the up-to-date status of each table in the restaurant. Tables can exist in 3 states: available, occupied and dirty
Payment System	paymentMade	Updates system depending on whether the payment has been made.
Food Status	orderStatus	Allows the chef to update the status of the current order being cooked
	orderReady	Allows chef to signal to waiter/waitress and customers that the order is finished
Order Queue	chefQueue	Keeps track of submitted food orders in the order that they were submitted for the chef to follow
Controller	tableList	Shows the customer the current tables available
	tableConfirm	Table is greyed out once it is picked by the customer and stays grey until cleaned
	paymentMade	Once the customer pays, the table is then confirmed and the order is initiated in the kitchen
Communicator	customerMeal	Each meal that the customer orders is stored in the database

	customerMealOrder	Meals in the queue are ordered and sent to a specific chef, to balance the chef work load
Floorplan	viewPlan	Displays the current layout of the restaurant
	tableAdjust	Allows the manager/server to adjust a table in the restaurant layout
Employee Profile	hoursWorked	Displays hours a particular employee has worked in a particular payment cycle
	tablesAssigned	Shows what tables the employee has been assigned recently (threshold to be determined)
	role	Role of the employee: food server, chef/cook, caterer etc.
Table Status	tableNumber	Displays the table number
	seatCount	Max number of people that can be seated at a particular table
	currTableStatus	Is table empty(white), occupied(grey) or uncleaned(red)?
Reward System	currUserPoints	Displays present user reward points balance
	rewardsRedeemable	Allows user to redeem available rewards
DB Connection	dbAuthenticationCreds	Stores the DB login for authorized personnel

6.1.4 Traceability Matrix

Use Case	PW	Domain Concepts									
		Customer Profile	Interface	Table Status	Payment System	Food Status	Order Queue	Controller	Communicator	Employee Profile	Floor Plan
*UC-1	3	X	X	X					X		X
**UC-2	3	X	X		X				X		
UC-3	5		X						X		
UC-4	3					X	X				
**UC-5	4		X								
UC-6	5		X					X			
UC-7	5									X	
UC-8	3					X	X				
UC-9	3	X									
UC-10	4			X							X
UC-11	4	X			X						
UC-12	4	X			X						
UC-13	3	X			X						
UC-14	4	X		X							X
UC-15	3	X									X
UC-16	5	X	X							X	
UC-17	4	X	X							X	
*UC-18	3		X								
***UC-19	3		X					X			

UC-2 - Payment: Customer uses payment so Customer Profile is involved. Payment information is collected and displayed with interface. Once payment is complete the table status can be updated. Communicator communicates information between different parts of payment system. Once payment happens the floor plan can be updated.

UC-3 - View Menu: Menu is displayed to users through the Interface. The Communicator communicates information between the interface and database.

UC-4 - Meal Prep: When food is being ordered and prepared its status is updated. Further when food is ordered it is added to the Order Queue.

UC-6 - Food filter: Selections on filtering are made on the interface and changes made are done by the Controller.

UC-7 - Clocking In/Clock out: Information stored by Employees clocking in/out is contained in the Employee's Profile

UC-8 - Serving: Once food is served is status is updated and thus its Food Status is changed.

UC-9 - Placing an Order: When someone places an order it is added to the Order Queue and its Food Status is changed.

UC-10 - Table Marking: When marking a table, its Table Status and Floor Plan is updated.

UC-11 - Earning Rewards: A customer may earn rewards and it is done while the customer is paying for their meal.

UC-12 - Redeeming Rewards: A customer may redeem rewards if they have enough points and it is done while the customer is paying for their meal.

UC-13 - Take-out: A customer may get take out and does not need to select a table, but does need to make a payment.

UC-14 - Table Selection: Utilizes the floor plan and interface to display it.

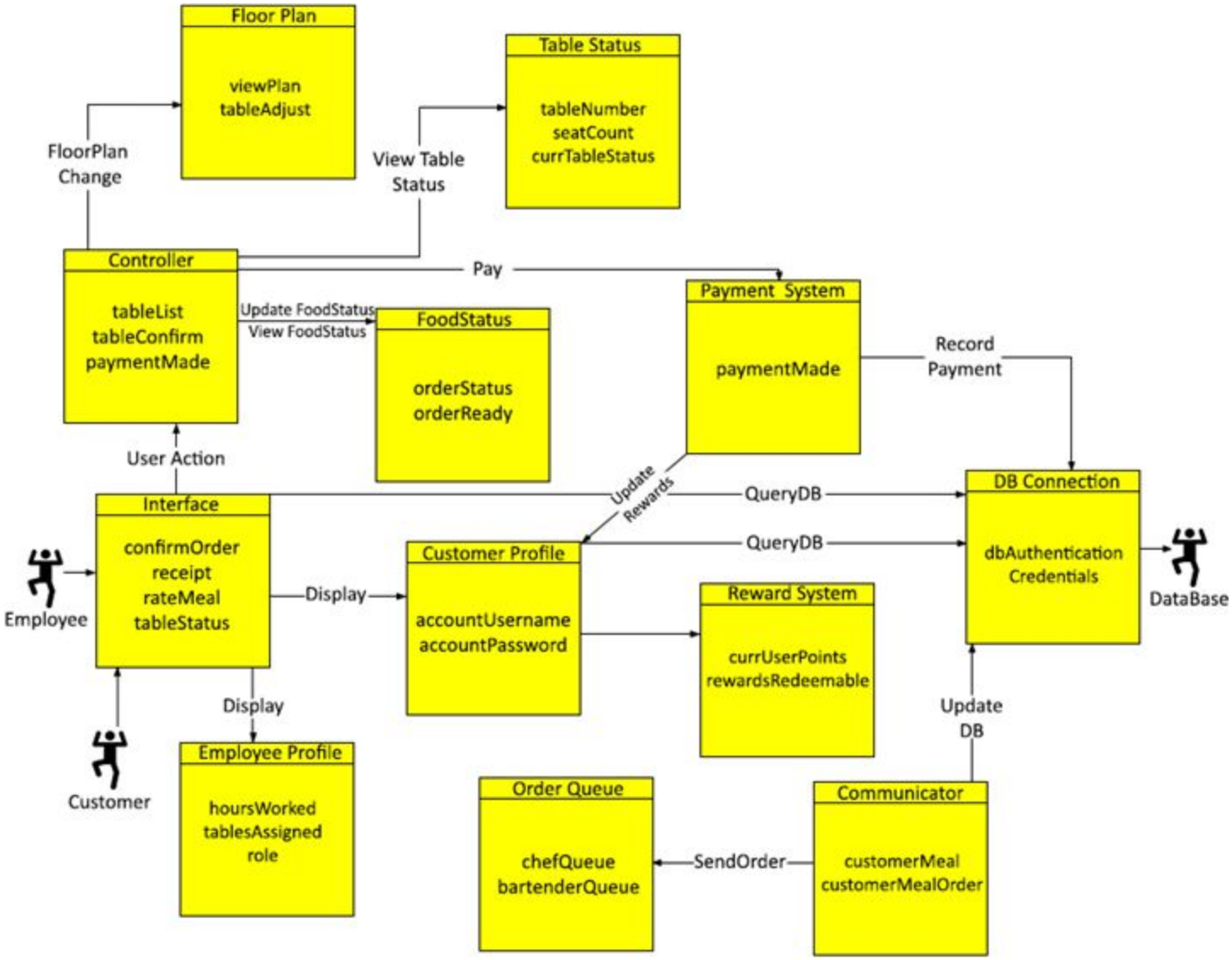
UC-15 - Floor Plan Status: Utilizes the floor plan and interface to display it.

UC-16 - Login: Any use must log in order to access their accounts, so customer and employee profiles are involved, in addition to this user login by typing on the screen and reading the information displayed by the interface.

UC-17 - Create Account: Customers and employees have the ability to create an account and therefore customer and employee profiles are involved. The user must interact with the application and the information that is displayed through the interface.

UC-19 - Menu Changes: In order to change the menu, a manager must interact with the application and the display using the interface and will then update the information in the database via the controller.

6.1.5 Domain Model Diagram



This diagram is a visual representation of the domain concepts in our project and how they interact with one another. Each individual domain concept was previously explained in the above sections.

6.2 System Operation Contracts

Operation	Payment
Use Case:	UC-2
Preconditions:	● The user has loaded the system ● The user has logged in as a Customer and has selected items from the menu.
Postconditions	● The user is prompted on payment method. ● The user is prompted on receipt option.

Operation:	View Menu
Use Case:	UC-3
Preconditions:	● The user has logged into their account/continued as guest ● a. Manager and waiter/waitress select the app options button in the top left, and then select the restaurant menu option b. Customer logs in with credentials or as guest and immediately sees the menu.
Postconditions	The viewer is presented with the menu interface.

Operation:	Clocking In/Clocking Out
Use Case:	UC-7
Preconditions:	● The user has loaded the system/logged in as employee
Postconditions	● The user has successfully clocked in/out ● The database saves the information for future use

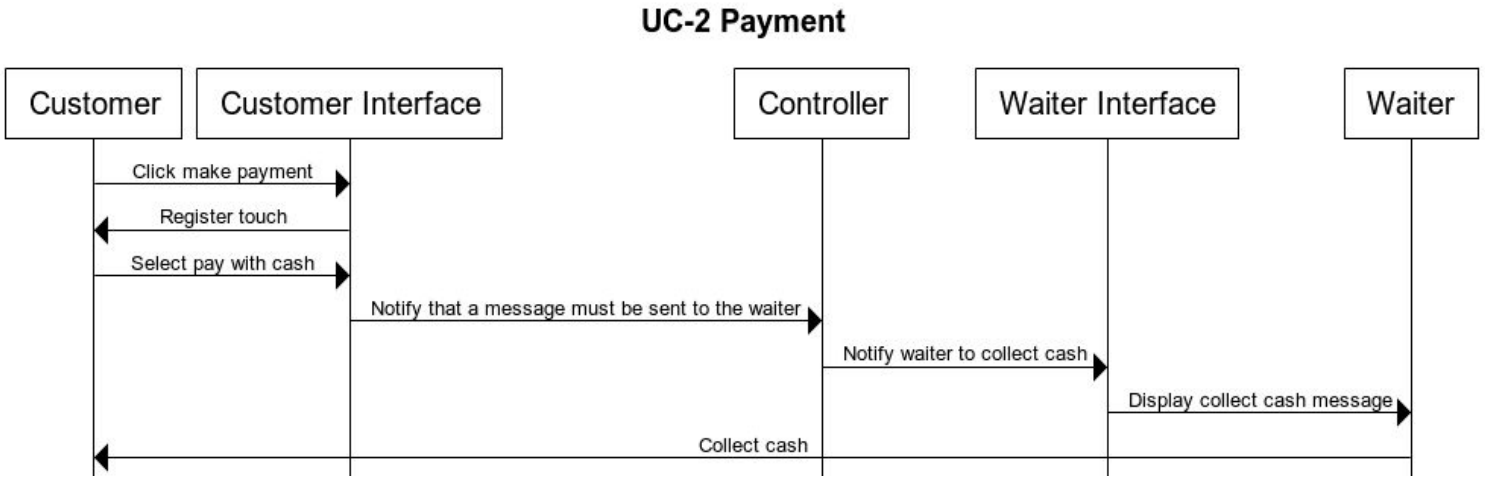
Operation:	Earning/Redeeming Rewards
Use Case:	UC-11
Preconditions:	● The user has loaded the system ● The user intends to make a purchase

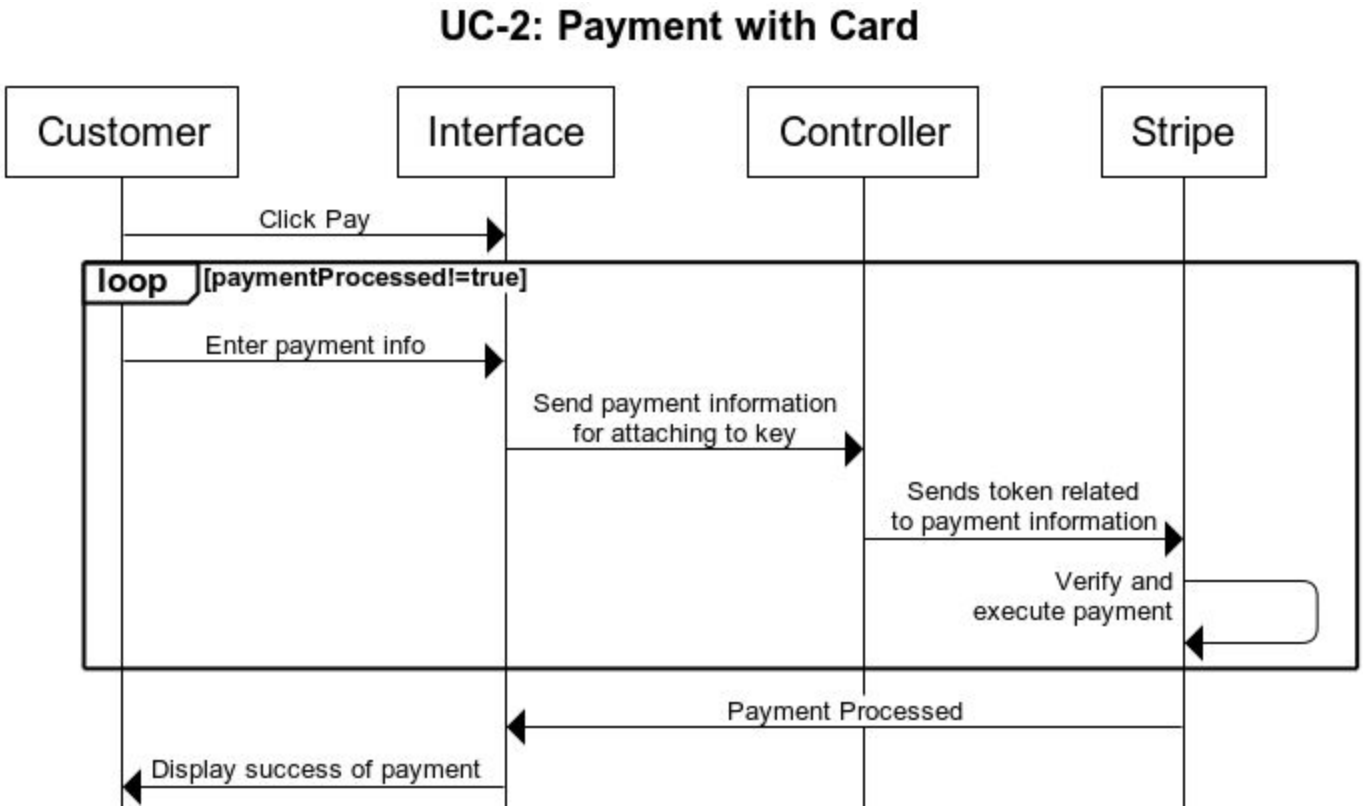
Postconditions:	• The user has earned an appropriate amount of points for how much was spent

7 Interaction Diagrams

UC-2: Payment

Payment with Cash





The above sequence diagrams display some of the potential paths that could be taken when attempting to pay utilizing the TurboYums payment processing. The second diagram shows a best case scenario while attempting to pay for a meal with a credit or debit card, after the customer interacts with the interface and enters their payment information, the information will be sent to the controller, which will then send the data to the specific database for the card (for eg: VISA, American Express, MasterCard etc.) in order to verify that the information sent is in fact a valid. The second diagram shows this process running smoothly with valid card information being entered on the first try. The third diagram shows a similar process but with invalid card information being entered, which results in a loop until the user enters valid information that may be used for payment.

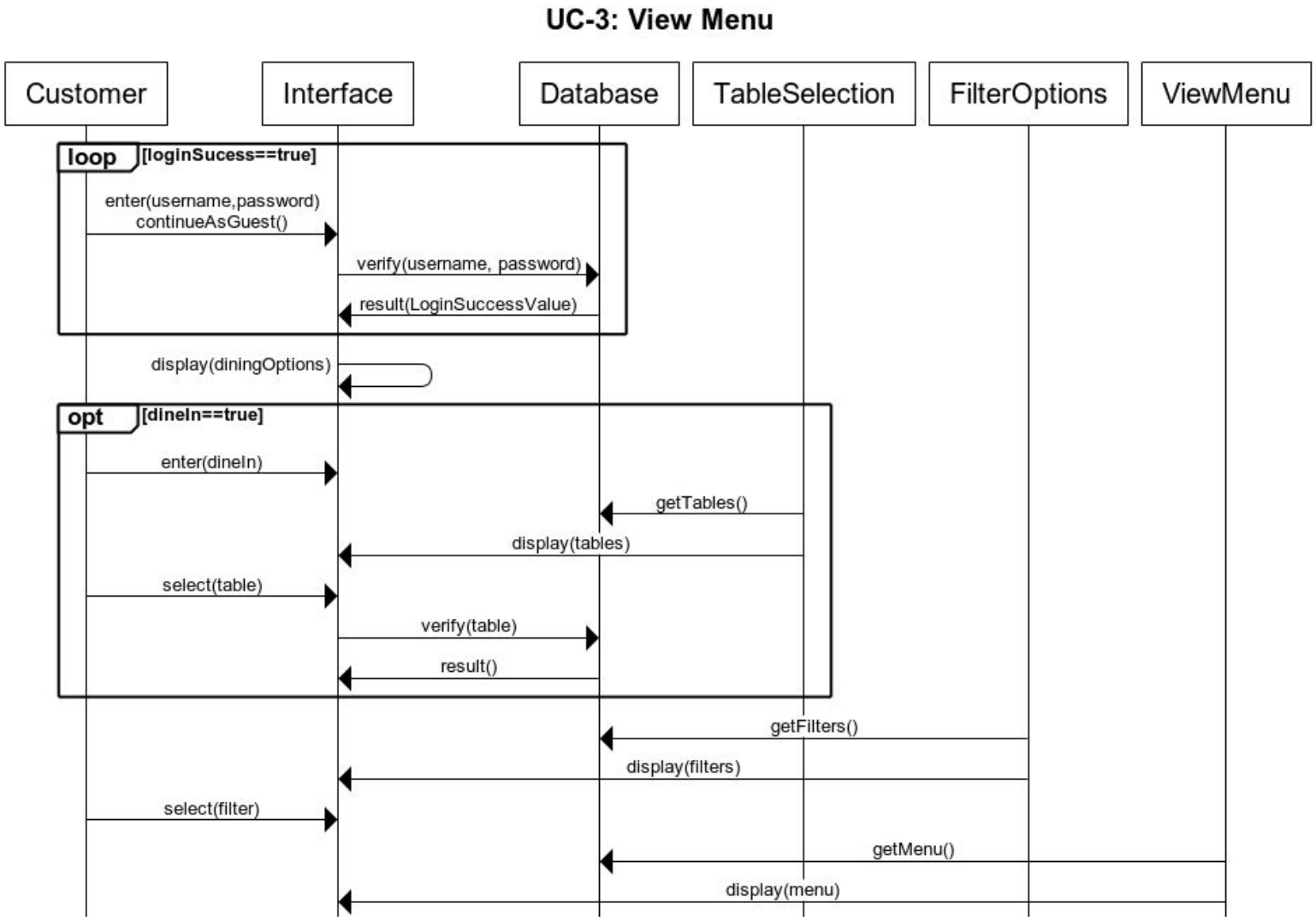
The first diagram shows what would happen if a customer attempts to pay with cash, which results in additional involvement of a waiter since a waiter must be notified of the payment in order to pick the cash up off of the table.

The high cohesion and low coupling principles were heavily utilized in various different locations while creating the diagrams. The employment of a controller utilizes the high cohesion principle because it allows many of the other objects, suchs as the interfaces and database to perform their intended tasks (interacting with the user and checking the card information) without having to send messages to other objects involved in the diagram about the status of their actions. Instead of the interface sending the card information to the database, or the customer interface sending information to the waiter interface that cash must be received, the interfaces send this information to the controller so that the controller may then perform the appropriate tasks as needed. The low coupling principle was largely utilized in the cash payment process in order to keep things quick and clean. Information is telephoned down the line from one object to another, through small connections and links, especially between the controller, waiter interface and waiter, this end of the diagram is mostly singly linked objects.

The expert doer was most used for the interfaces and the databases because these objects are the only ones that can successfully perform the tasks that they are undergoing. The interface must serve as a bridge between the user and the software, and the card database is the only thing that can be used to verify the card information.

The diagrams have been made so that they resemble as close as possible to a pub-sub model hence focusing on a better design model than what was decided on before.

UC-3: View Menu



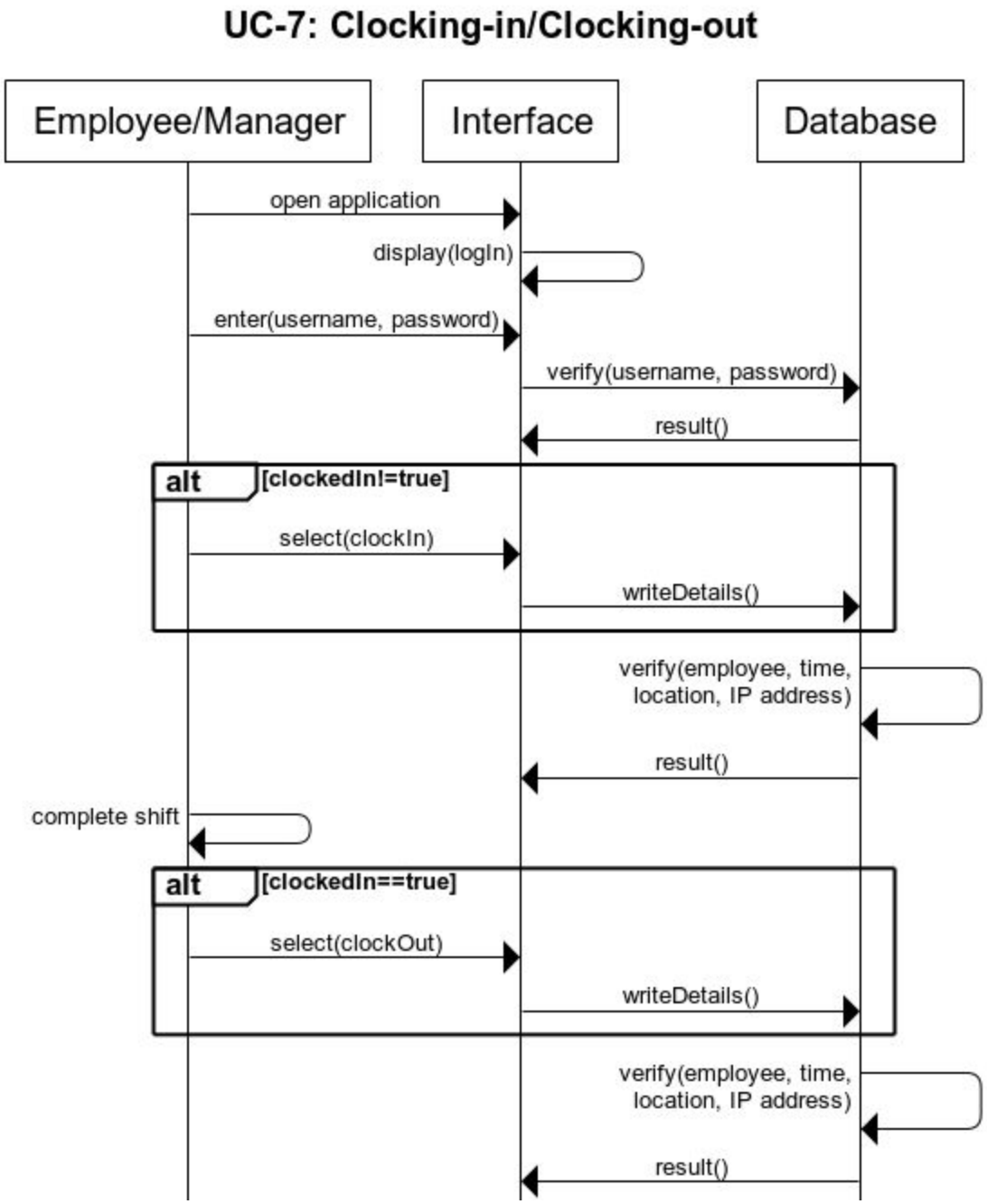
The diagram demonstrates the interaction between UC-3: View Menu. Once the user opens the app and logs in, the system verifies the login with the database. Once verified the user is brought to the next screen which is the dining option of either dine-in or take-out. After, the choice of dining is selected the user is brought to the next screen which is seating options. The seating option allows the user to see the full floor (table) layout and select the table of their choice by simply selecting the table number and automatically graying it out when confirmed. Then, users are moved along to the filter screen where they can indicate any dietary restrictions and after are brought to the menu screen with those filters taken into consideration when displaying the menu. The user is able to select the food items (on click) of their desired choice by adding it to their cart and finish off by confirming the order.

One design principle employed in this sequence is the High Cohesion Principle. The High Cohesion Principle says that an object should not take on too many computational responsibilities. This principle is utilized in that each class only takes on responsibilities that have to do with its specific functionalities and does not take on more than it can handle at a time.

Another design principle used was the Expert Doer Principle, which says that each class is an expert in a specific function. This is shown in the diagram because, as an example, the only function for TableSelection is simply to

display the table layout and have the guests select a table. After this function is over, responsibility is passed on to FilterOptions, where the sequence continues on.

UC-7: Clocking-in/Clocking-out

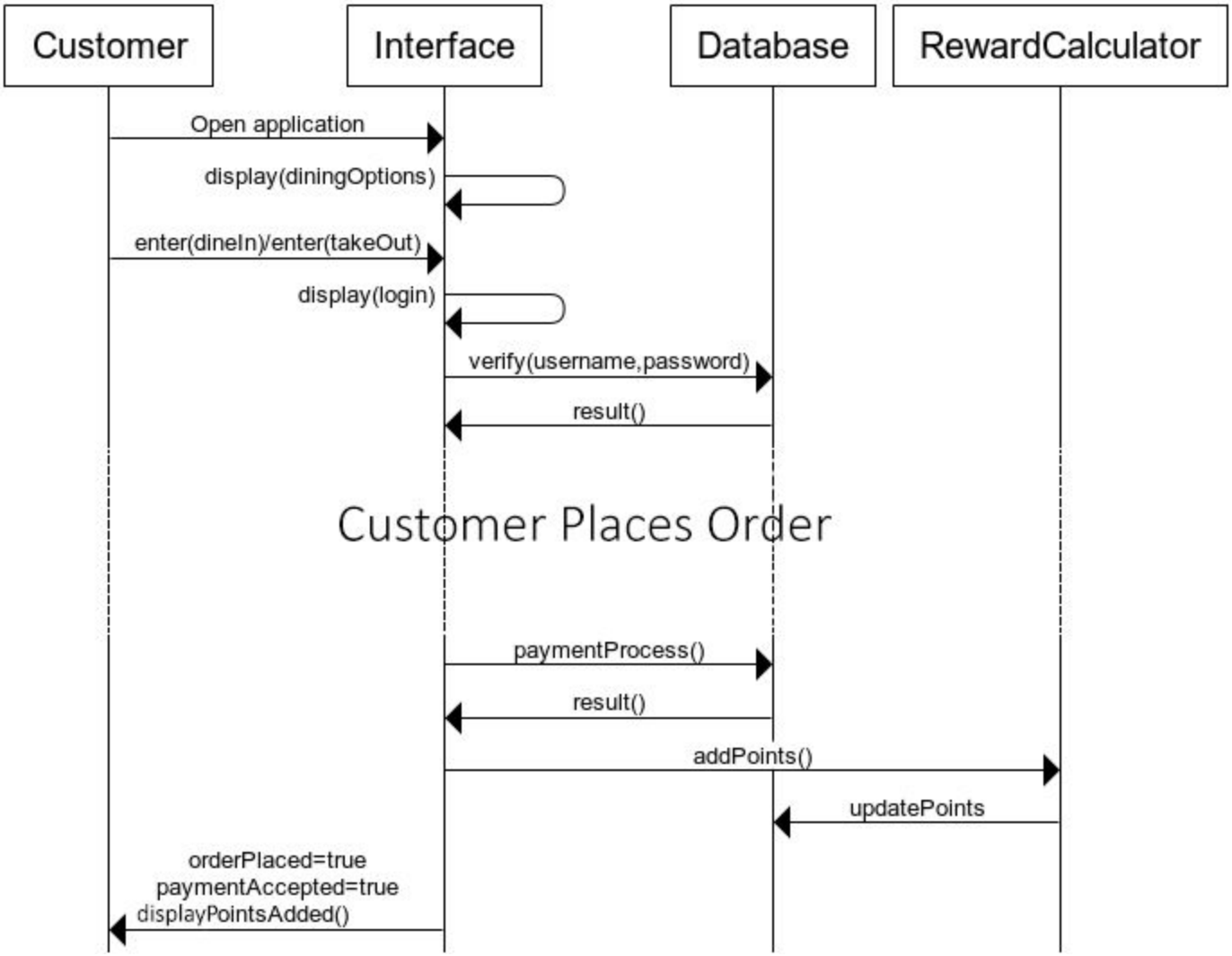


The diagram above demonstrates the interaction between objects and/or actors in UC-7: Clocking In/Out. The user must interact with the interface in order to open the application and log in. Once the user enters their login details, the interface will check them with the database and confirm their credentials. The user is then brought to an interface where they are presented with the option to clock in. Once they select the option to clock in, the system logs the user's name,

time of clock-in, location, and ip address. Later on in the day, the user is able to use the system again to clock out, where the system again logs the user's name, time, location, and IP address.

One design principle that is used is the High Cohesion Principle. All of the objects do not take on many computation responsibilities. Each object only focuses on the tasks they are specialized to do. The database has all the logged information required for objects to perform their task, while the objects simply have a calculation dedicated to their specified task (E.g. Clock in logs data at time of clock in, and vice versa for clock out).

UC-11: Earning Rewards



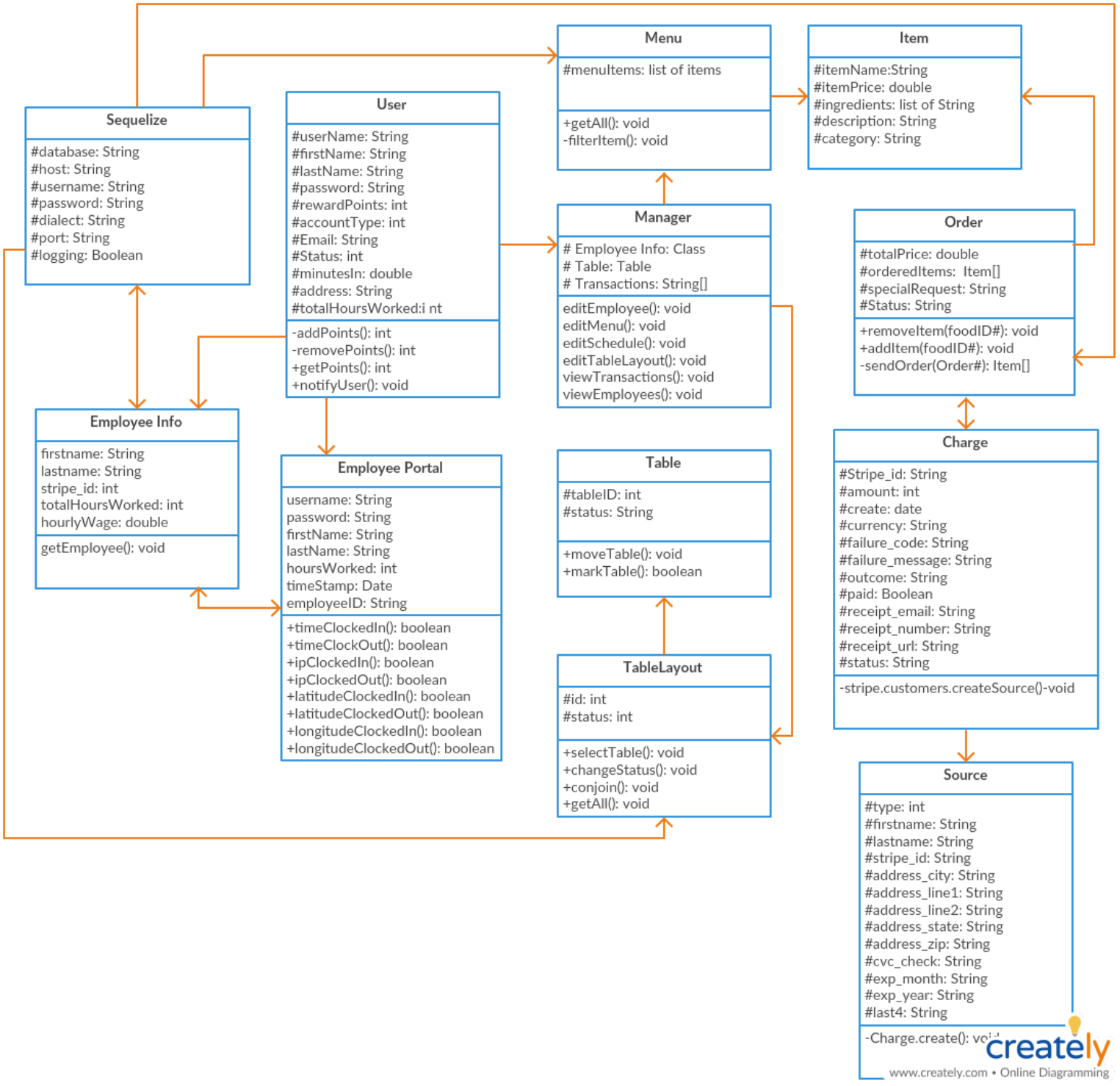
The diagram above demonstrates the interactions involved in UC-11: Earning Rewards. Initially the Customer will launch the application and select their dining option. The interface will prompt the user for their Once the user enters their login details, the interface will check them with the database and confirm their credentials. Upon sign in, the interface will display a menu and allow them to place their order (This process was left out of the diagram, as this process is lengthy, and doesn't differ in this use case). Once the customer is done with the ordering process, they will be directed to payment (see UC-2). Upon completing the payment, the interface will carry details of the transaction to the reward calculator. The reward calculator will calculate an amount of points to give the user proportional to their total (as determined by the restaurant owner). The reward calculator will then update the database with the customer's new balance, then the interface will display their new balance to the user along with their receipt confirming the transaction was completed successfully.

One design principle that is used is the High Cohesion Principle. All of the objects do not have many computation responsibilities. The interface simply displays information to the user and prompts them for input. The database will

store and return related information for each task. Also, the reward calculator will only be responsible for calculating rewards.

8 Class Diagrams and Interface Specification

8.1 Class Diagrams



This is a representation of all of the objects used in the software and which ones interact with each other.

8.2 Data Types and Operation Signatures

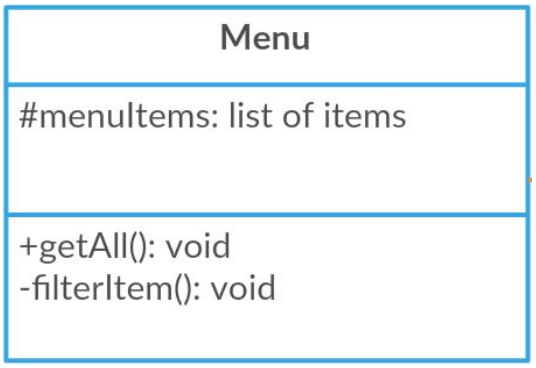
Class: Menu

Attributes:

- menuItems: list of Item - The food items available to the customer.

Methods:

- getAll() - The database retrieves all of the menu items.
- filterItem() - The database filters out menu items, according to preference.



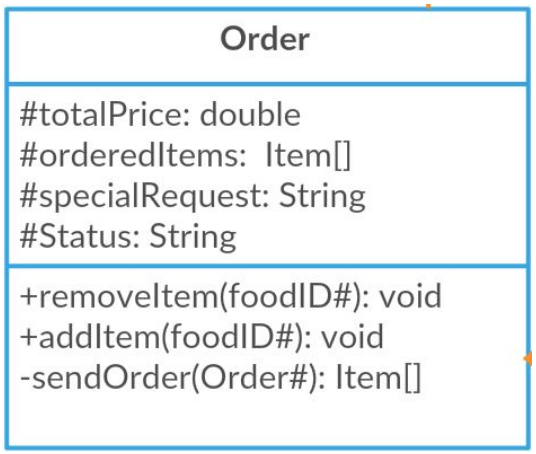
Class: Order

Attributes:

- totalPrice:double - The total price of all items in the order.
- orderedItems: Item[] - The list of food items in the cart picked by the customer.
- specialRequest: String - A place for customers to express any request not available in the UI.
- Status: String - Maintain the status of the order

Methods:

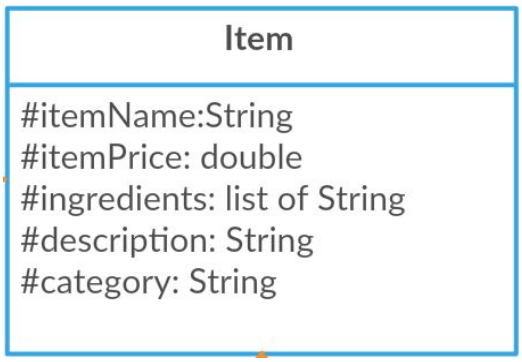
- removeItem(foodID#)- removes food from existing order- will take a food ID and quantity.
- addItem(foodID#)-adds food/beverage indicated by the customer to the current order.
- sendOrder(Order)- sends the order to the kitchen to be made.



Class: Item

Attributes:

- itemName: String - The name of an item.
- itemPrice: double - The price of each item.
- ingredients: list of String - a list of the different ingredients needed to make the item.
- description: String - each item is described with a few short sentences.
- category: String - a string to keep track of what category an item may be included in



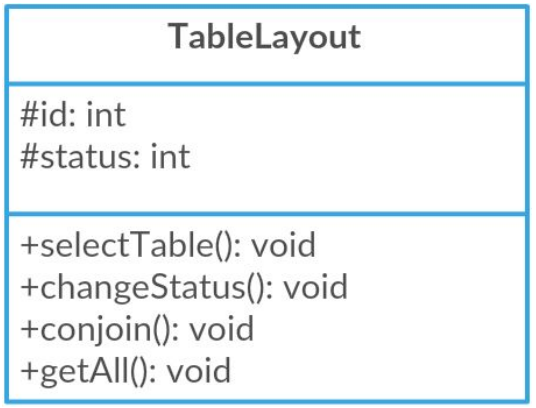
Class: TableLayout

Attributes:

- id: int - The ID of a table.
- status: int - The current state/status of a table - ‘green’ is available, ‘red’ is occupied, and ‘coral’ is dirty.

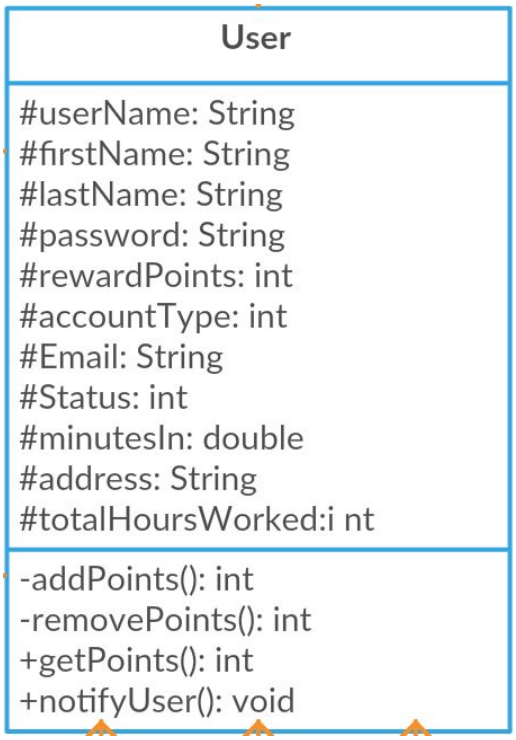
Methods:

- selectTable() - The customer selects a table.
- changeStatus() - The status of a table will change status upon selection.



- conjoin() - The customer is able to select two tables to conjoin them.
- getAll() - The database retrieves all of the tables in the restaurant.

Class: User



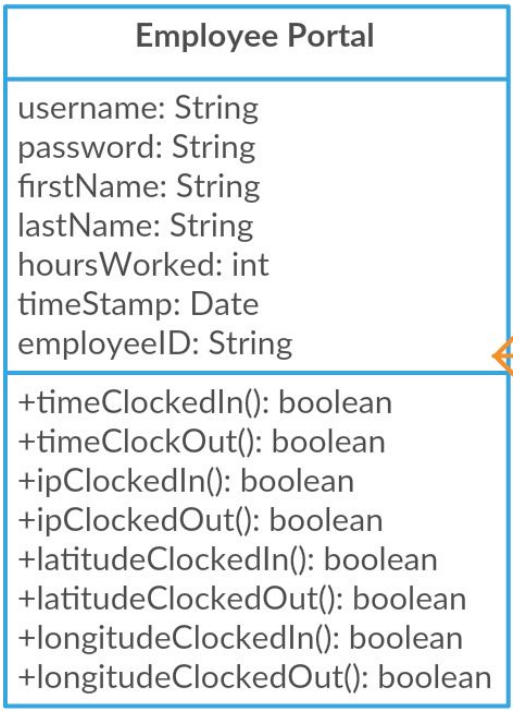
Attributes:

- userName: String - The username of the user.
- firstName: String - The first name of the user.
- lastName: String- The last name of the user.
- password : String - the password of the user.
- rewardPoints: int - How many rewards points the user has.
- rewardBalance: double - The amount of rewards the user has (1 point is not 1 reward)
- accountType: int - The type of user account.
- Email: String - Email address of the user.
- Status: Int - an integer value to keep track of whether or not an employee is clocked in.
- minutesIn: Double - record the amount of minutes that a user has spent clocked in
- address: String - The user's address.
- totalHoursWorked: int - total hours worked by user if the user is of the employee account type.

Methods:

- addPoints() - Add an integer of reward points to the user's balance.
- removePoints() - Remove an integer of reward points to the user's balance.
- getPoints() - Get the user's point balance.
- notifyUser() - If the user has an active session, send them a message.

Class: Employee Portal - This class is made to allow employees to clock-in and clock-out of their shifts (including breaks).



Attributes:

- username: String - This is the individual employee's self-made username to be able to have an account.
- password: String - This is the individual employee's self-made password to keep their profile protected.
- firstName: String - This is the individual employee's first name.
- lastName: String - This is the individual employee's last name.
- hoursWorked: int - This will keep track of the amount of time that the individual employee has worked during the current shift.
- timeStamp: Date - The time and date of the clock-in/clock-out.
- employeeID: String - This is an employee specific ID, so that the system and manager can identify the employee's role and have the ability to edit and track.

- timeClockedIn: String - Employee has the ability to clock in when beginning their shift, time is recorded.
- timeClockedOut: String - Employee has the ability to clock out when ending their shift, time is recorded.
- ipClockedIn: String - Employee clocks in, IP is recorded so employee cannot use a non company owned computer or log from home.
- ipClockedOut:String -Employee clocks out, IP is recorded so employee cannot use a non company owned computer or log from home.
- latitudeClockedIn: String - Employee clocks in, latitude is recorded so employee cannot use a non company owned computer or log from home.
- latitudeClockedOut: String - Employee clocks out, latitude is recorded so employee cannot use a non company owned computer or log from home.
- longitudeClockedIn: String - Employee clocks in, longitude is recorded so employee cannot log from home.
- longitudeClockedOut: String - Employee clocks out, longitude is recorded so employee cannot use a non company owned computer or log from home.

Methods:

- clockedIn(): boolean - Employee has the ability to clock in when beginning their shift.
- clockedOut():boolean - Employee has the ability to clock out when ending their shift.

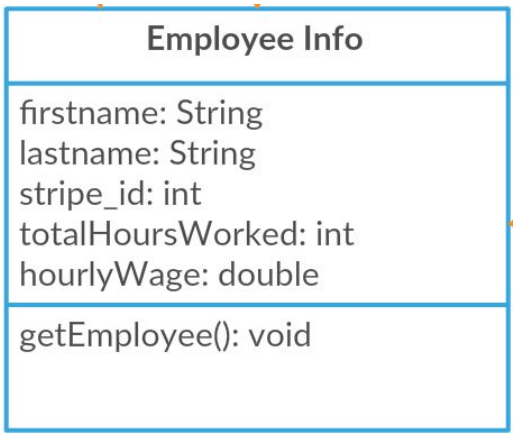
Class: Employee Info - This class is made to hold employee information, if the employee wishes to access it.

Attributes:

- firstname: String - The first name of the employee.
- last name: String - The last name of the employee.
- stripe_id: int - The unique ID of the employee.
- email: String - The email of the employee.
- totalHoursWorked: int - This will keep track of the total amount of time that the individual employee has worked during the payment cycle.
- hourlyWage: double - This will keep track of the individual employee's working hourly wage.

Methods:

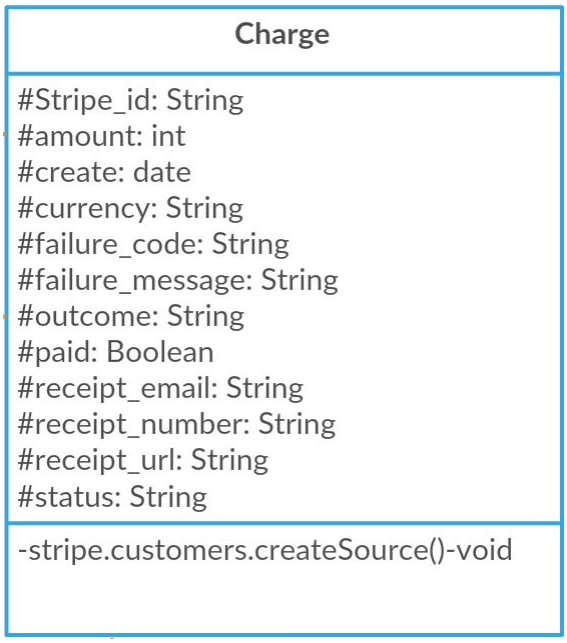
- getEmployee() - This will display the employee's information.



Class: Charge - This class is created to help maintain the information that is associated with a transaction occurring.

Attributes:

- Stripe_id: String - The unique ID of the charge
- amount: int- The amount being charged
- created: date- The time and date that the transaction was enacted.
- currency: String, - The type of currency being used
- failure_code: String - The code associated with the reason for failure
- failure_message: String,- The message displayed when there is a failure



- outcome: String, - The outcome of the charge
- paid: Boolean - used to tell if the order has been paid for or not
- receipt_email: String, - The email the receipt will be sent to
- receipt_number: String, -The unique receipt number
- receipt_url: String, - url associated with the receipt
- status: String, - associated with the status of a charge

Methods:

- stripe.customers.createSource() - Creates a new Source and defines all associated variables

Class: Source - This class will keep track of all of the details about the method of payment, if it was done by cash or card, and if the transaction is completed by card then the card information will be stored.

Attributes:

- firstname: String, - The first name of the person being charged
- lastname: String, - The last name of the person being charged
- stripe_id: { type: Sequelize.STRING, primaryKey: true }, - The charge's unique stripe ID
- address_city: String, - The city of the billing address
- address_country: String, -The country of the billing address
- address_line1: String, - The address of the billing address
- address_line2: String, - The alternate address of the billing address
- address_state: String, - The state of the billing address
- address_zip: String, - The zip code of the billing address
- cvc_check: String, - The check to make sure the security code of the payment card is correct
- exp_month: String, - The expiration month of the payment card
- exp_year: String, - The expiration month of the payment card
- last4: String, - The last 4 digits of the payment card

Method:

- Charge.create() - Creates a new charge for the table and defines all associated variables

8.3 Traceability Matrix

	Software Classes										
Domain Concepts	Menu	Order	Item	TableLayout	User	Employee Portal	Employee Info	Transaction	PaymentMethod	Charge	Source
Customer Profile		X			X			X	X	X	X
Interface	X	X	X	X	X	X			X		
Payment System					X			X	X	X	X
Food Status		X									
Order Queue		X									
Controller	X			X				X	X		
Communicator		X			X						
Table Layout				X							
Employee Profile					X	X	X				
Table Status				X							
Reward System					X			X			
DB Connection		X	X		X		X	X	X		

- Customer Profile:
  - User: Allows customer to view and edit account specific data.
  - PaymentMethod: Allows customer to select preferred payment method.

    - Source: Allows customer to have a saved payment method.
    - Charge: Allows customer to pay for his order.
- Interface:
    - Menu: The restaurant's menu is accessible via the interface.
    - Order: Orders can be placed through the interface.
    - Item: All available items are presented on the interface.
    - TableLayout: Table selection is done via the interface.
    - User: User data is presented via the app interface.
    - Employee Portal: The portal is accessible via the app interface.
    - PaymentMethod: Chosen through the interface.
- Payment System:
    - User: The purchase is logged to the user's account after the payment is made.
    - Transaction: The system attempts to validate and confirm payment has been made.
    - Payment Method: The system uses the selected payment method in order to complete payment.
    - Source: The system allows the customer to have a saved payment method.
    - Charge: The system allows the customer to pay for his order.
- Food Status:
    - Order: Food Status initiates once an order is placed.
- Order Queue:
    - Order: Ordering a food item adds it to the queue.
- Controller:
    - Menu: Controller requires that items be chosen from the menu.
    - TableLayout: Controller requires that a valid table is selected.
    - PaymentMethod: Controller requires a valid Payment Method from customer.
- Communicator:
    - Order: Communicator allows the customer's order to be communicated to the chef via queue.
    - User: Allows accounts to validate information from server, such as login.
- Floorplan:
    - TableLayout: Allows the table to be conjoined to be selected.
    - User: Users will be able to select a table and change the status.
- Employee Profile:
    - User: Contains employee personal information.
    - Employee Portal: Allows employees to clock in/out and view work status.
    - Employee Info: Allows employees to view their current earnings and status.
- Table Status:
    - TableLayout: Allows users to select a table and view its status.
    - User: Can be viewed and edited by a user.
- Reward System:
    - User: All rewards points are saved to a user's profile.
- DB Connection:
    - Order: Orders will be sent to the database once sent by the customer.
    - TableLayout: Table information is stored on the database.
    - User: User account information will be stored on the database.
    - Employee Info: Employee information will be stored on the database.
    - Payment Method: Saved and preferred payment methods will be stored on the database.

8.4 Design Patterns

We used the Publisher - Subscriber design pattern in TurboYums in order to make classes interact in our use cases. This design pattern makes an event detector object tell multiple, decoupled event doer objects when events are available. The Publisher - Subscriber pattern is extremely useful to our team because we can easily make multiple use cases using the same basic format of four actors. These actors are user, interface, controller, and database. There is a user (such as a customer, employee, etc.) who interacts with a user interface. This user interface interacts with a controller, and this controller interacts with a database of users' information. The subscriber (or doer) in a use case is just the employee who wants to clock in/out, customer who wants to order, manager who wants to edit employee information, and others. The publisher in each case is the database that holds information about users, and the controller that tells the relevant interface what to do. Publisher - subscriber is used when the publisher has unrelated responsibilities. An example of such responsibilities is again in UC-2, the user one (customer) has to interact with the publisher (controller), the user two (waiter) has to interact with the publisher, but the two users do not actually interact with each other.

Other use cases we made have more than just the four basic actors described above. UC-3 has additional actors such as the filterOptions and viewMenu classes, and when applicable, tableSelection. Publisher - subscriber is again useful because there is loose coupling between tableSelection and filterOptions/viewMenu, a user can view the menu if that user is taking food out and not selecting a table.

8.5 Object Constraint Language (OCL)

MainScreen:
Context MainScreen::clickLogin
Invariant: privateKey, username
Pre-conditional: findViewById(R.id.username)
Pre-conditional: Intent intent = new Intent(MainScreen.this,tableSelection.class)
Post-conditional: MainScreen.user.startActivity()

Context MainScreen::clickSignUp
Invariant: privateKey, firstName, LastName, accountType, username, password, email
Pre-conditional: findViewById(R.id.signup)
Pre-conditional: Intent intent = new Intent(MainScreen.this, signUp.class)
Post-conditional: MainScreen.signUp.startActivity()

Context MainScreen::clickContinueAsGuest
Invariant: privateKey, menu
Pre-conditional: findViewById(R.id.menu)
Pre-conditional: Intent intent = new Intent(MainScreen.this, menu.class)
Post-conditional: MainScreen.menu.startActivity()

LoginActivity:
Context: LoginActivity::passwordChecked:boolean
Invariant: auth, loginButton, signupLink
Pre-conditional: username = usernameText.getText().toString()
Pre-conditional: password = passwordText.getText().toString()
Post-conditional: return isValid

SignupActivity:
Context: SignupActivity::signup
Invariant: progressDialog
Pre-conditional: firstname = firstnameText.getText().toString()
Pre-conditional: lastname = lastnameText.getText().toString()
Pre-conditional: email = emailText.getText().toString()
Pre-conditional: username = usernameText.getText().toString()
Pre-conditional: password = passwordText.getText().toString()
Pre-conditional: accountType = accountTypeText.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: Alert.alert(‘Sign up failed.’)
Context: SignupActivity::checked:boolean
Invariant: firstname, lastname, email, username, password, accountType
Pre-conditional: firstname.isEmpty()
Pre-conditional: lastname.isEmpty()
Pre-conditional: email.isEmpty() || !Patterns.EMAIL_ADDRESS.matcher(email).matches()
Pre-conditional: username.isEmpty() || username == existingUsername
Pre-conditional: password.isEmpty()
Pre-conditional: accountType.isEmpty() || accountType != 0 || accountType != 1 || accountType != 2
Post-conditional: return isValid

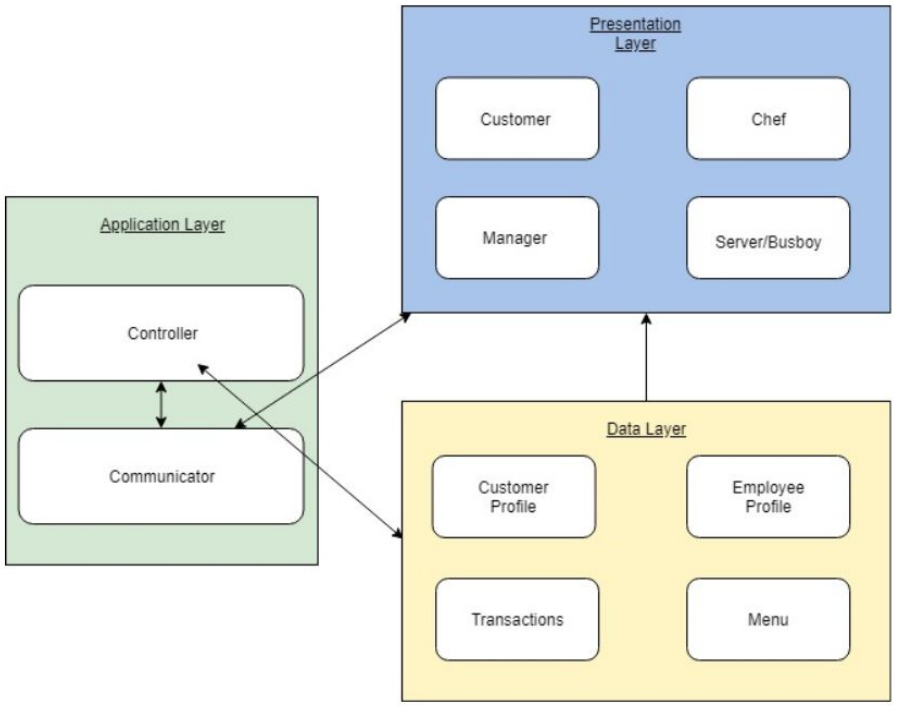
9 System Architecture and System Design

9.1 Architectural Styles

For our application, we are using the Layered Pattern mode, which is commonly used for general desktop applications. The three layers are presentation, application, and data layers. The presentation layer is seen by the users on the front-end, both customers and restaurant staff. Application and data layers are on the back-end and keep track of user-entered information, as well as respond to commands from the presentation layer. This model is appropriate for our application as the layers are abstracted from each other to therefore run parallel with the code.
Our application also uses the Client-Server architecture model which is a one to many relationship. The server provides/stores data. In the restaurant, the servers are the many different interfaces for customers and for the restaurant staff. Users are the clients that have to enter information into three interfaces for; ordering food from the menu, making reservations, logging in to the rewards system, and making payments. The employees are clients who use the clocking in/out server to log their shifts so they will be paid. Managers are clients who use their options server to edit employee information and scheduling, edit the menu, and edit table information.
Another architectural style utilized is the Peer-to-Peer model. This model is a distributed application architecture that partitions tasks between peers. This will be used for the payment of the order by a customer to the restaurant and the model is decentralized so every payment gets processed separately. In our application this model is used when if the customer pays by card then the peer to peer model becomes a payment service to transfer funds from customers bank account to the restaurant account.

9.2 Identifying Subsystems

UML Package Diagram



The subsystem above displays our three layer architecture - which consists of a Presentation Layer, an Application Layer, and a Data Layer. Each layer has a specific responsibility to the entire system.

The presentation layer handles the user interface level. This layer holds the information dealing with the Chef, Manager, Busboy/Waiter/Waitress, and Customer Interfaces. The application layer is the middle tier of this architecture, which handles communication between the top and bottom layers, essentially managing the overall operation. It will run the business logic, which is the set of rules required for running the

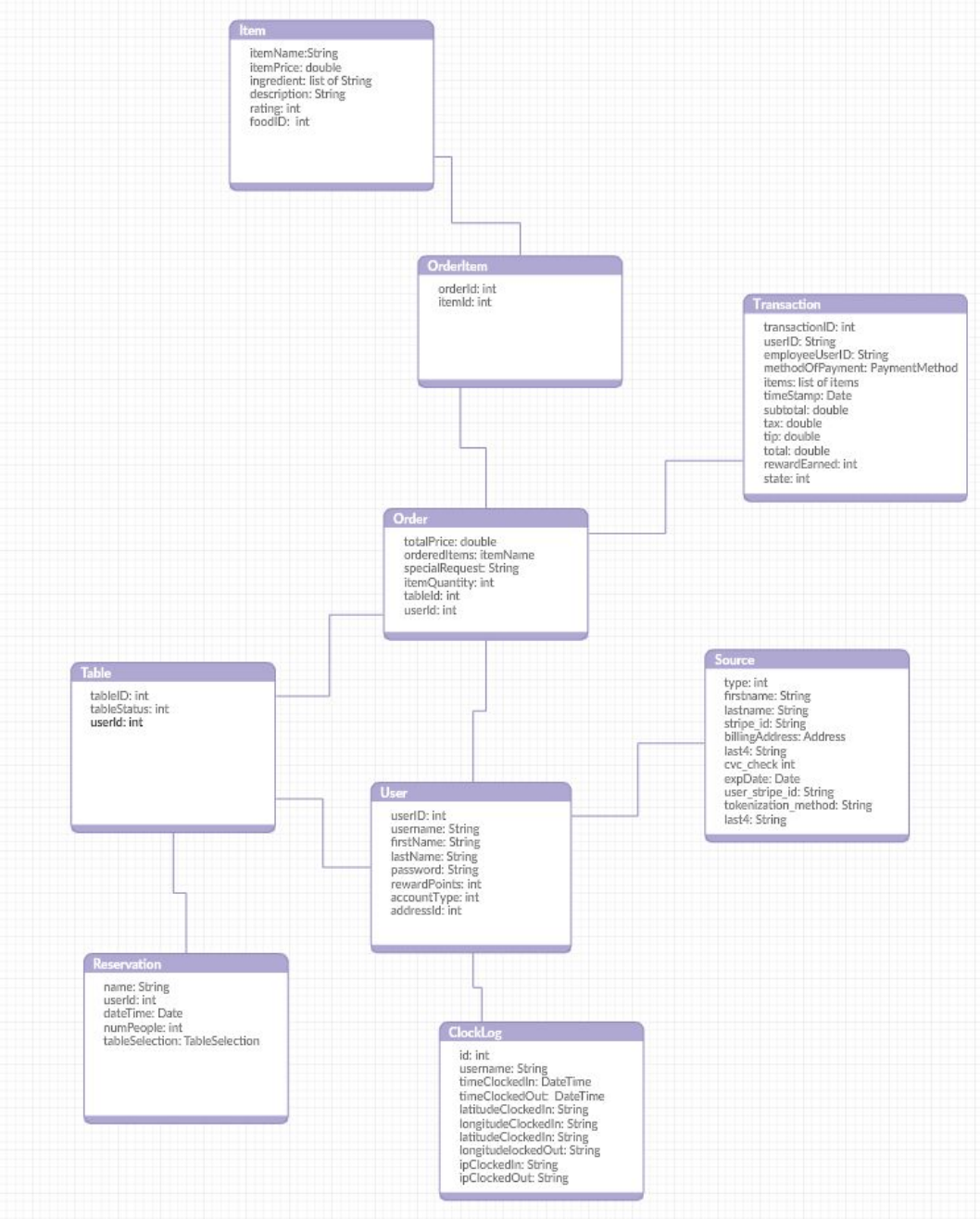
application for the guidelines given by the organization. Contained within this layer is the Communicator and Controller. The Controller is utilized to facilitate tasks between the layers and the Communicator is used to pass information between the layers. The lowest layer is the data layer, which deals with the storage and retrieval of data. This tier will hold information pertaining to the Employee and Customer Profiles, as well as the available menu and restaurant transactions.

Each layer doesn't need to worry about the other layers, due to separation of concerns. Also, due to layers of isolation, changes made in one layer don't generally affect the other layers. Because of this, the architecture makes testing and diagnosing issues more manageable.

9.3 Mapping Subsystems to Hardware

For our software there will be a client running on tablets and cellphones in the restaurant. There will also be a backend server, which will contain the REST API and database. The application can be run on many different devices as long as it meets the requirements listed in Section 3.7 Hardware Requirements. Our system will have a RESTful API used to communicate with our servers. POST and GET requests are initiated from the client, and will yield a response from the server POST requests supply additional data from the client to the server. GET requests will retrieve useful information for the client. There also is a physical server that we will control using Node.js. For the server Node.js contains packages Express and Sequelize which helps the communication between the server and the database. Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. Express is used to create handle routing and process requests from the client. Sequelize is a promise-based ORM for Node.js, which will enable the system to operate on objects which map to the relational database. This supports MySQL and features solid transaction support, relations, read replication and more. For our database we will be using MySQL. MySQL is a Relational Database Management System that uses Structured Query Language (SQL). SQL helps with adding, accessing and managing content in a database.

9.4 Persistent Data Storage



Our software does need to store persistent data that needs to outlive a single execution of the system. For one, the user's information needs to be saved in a user table. Our software needs to remember the customer's first name and last name, username and password, as well as their Reward Points earned and payment information. For an employee thee software also needs to remember the employee's first and last name, username and password, salary, time worked, work schedule, etc. Our software will also need to hold onto the restaurant's transactions, so that the manager will be able to access it whenever they please and have the ability to track earnings (total revenue and profit). Since the manager should also be able to adjust the table layout, our software needs to have the ability to remember the restaurant's current table layout. Other important information that needs to be saved is the available menu, along with

the totaled ratings calculated for each item, item prices, item names, the recipes, and item number. This information will all be stored in a MySQL database running on our web server.

9.5 Network Protocol

In order to complete the restaurant automation application, our team is implementing a REST API. A REST API will perform actions on the server based on HTTP POST and GET requests. HTTP REST lends itself well to a client server architecture. The front-end of the application will be built with React Native, which will handle performing the requests. The user interacts with the application and when they perform a task, a request will be sent to a server via HTTP where the information will be processed. For example, if the user orders food after clicking confirm, the order will be sent via HTTP POST request to the server which will respond back with information including estimated wait time, which is necessary for the client to display to the user. Another example is when the user seeks information about how many reward points the user earned, a GET request from the client to the server will be issued requesting the user's reward points balance, the server will process this request and in its response body will include the balance.

9.6 Global Control Flow

Execution Orderness: This is procedure driven in a "linear" fashion. The order of the application differs depending on the different users. The initial action is the same for all users, which is logging in or continuing as a guest. If the user is a customer, then the order of the application are as follows: user will be able to choose dining option, select a table, filter out allergies/dietary restrictions, select items from menu into cart, then proceed to cart, with an option to view the order progress, and finally to the payment screen. The user has very little control of the execution orderness. The owner/manager experience is less linear than the customer. They have the option to view the table layout and change it whenever needed. The manager can view/manage employees/employee information through an employee management screen. The employees also have more flexibility in their application order, they can few the table selections and move the table around, clock in and clock out, and view orders.

Time Dependency: While there are no timers in the application, some things do rely on the physical time such as keeping track of orders. The customer/user will be able to view the order progress. This will depend on when the chef finishes preparing the order and notifying when done. Also, the employees will be having a clocking in/out view where the timer will keep track of the time from when the employee signs in to the time the user signs out this will have a time stamp to each employeeID.

Concurrency: We utilize different threads in our server because you can do multithreading as every order will be taken care separately. Hence, there are different threads asynchronously handling different operations on our server.

9.7 Hardware Requirements

Our software application will be able to be run on mobile devices such as smartphones and tablets utilizing network support (WiFi or Mobile data). The layout for both types of devices (smartphones and tablets) will be identical, as they are both using the same version of the application. The user application is not intended to be used on a computer, so there is no developed layout for desktop.

This software will be compatible with iOS and Android operating systems. For iOS, we will be supporting iOS 12 and later, which covers over 80% of the iPhones in circulation. For the Android operating system, we are aiming to support API 23-Marshmallow, by supporting this operating system we will be supporting 71% of current Android phones. This software will be interacted with via a touchscreen interface. Our software will have a color display, with a minimum resolution of 1330 x 700 pixels upto 2960 x 1440 pixels. A minimum storage capacity of 50 Megabytes must be available for the user to download and install the application. The application will communicate over the restaurant Wi-Fi and will push and pull requests to communicate with a server, as well as require GPS access.

10 Algorithms and Data Structures

10.1 Algorithms

For employee working information and clocking in/out, the algorithms will be calculations for how much an employee works. The system takes note of the current time when an employee clocks in and out. In our code, there is a method that will make the time worked calculations more simple. The method will subtract timeIn from timeOut, and return the result of hours worked during the employee's shift. If the manager wants to figure out payroll, the Salary() method will calculate the annual salary of the given worker given hourly wage (wage), and number of hours worked (hoursWorked). This method gives a running total of a worker's salary for the year. The worker's salary for the day is calculated by hoursWorked * wage, this salary is added to expectedPay, then this process is repeated for every working period throughout the year.

When a customer is making a transaction, the order subtotal is calculated by adding the price of all items in the order, then a tax rate is applied to calculate a total. Once this information is presented to the user, and they enter their tip, that is added to calculate the total amount due for the transaction. The stripe API will be used to implement payment, and so all the information attached to this order will be used to call the stripe API. Once the transaction goes through, the rewards system will increment a counter of how many visits the visiting customer has. Once they reach a set amount of visits (decided by the restaurant), they will earn a reward, which will be a set amount off of their next purchase (as decided by the restaurant as well).

For the menu side of our application, customers will have the option to view a menu. Once customers view the menu, they will have the option to browse through everything in the menu or they can filter things out. For example, if a customer has a food allergy they can choose to filter out all items that have that ingredient in the meal. This is very helpful for customers with food allergies and other dietary restrictions. Once an ingredient is selected on the filter, our system will check the database for all the items including that ingredient. Then our database will send back the list of foods without the filtered ingredient and only display food items without the filtered ingredient. Once customers know what they want, they will be able to add and remove food items from their order as they please.

10.2 Data Structures

We can utilize a queue data structure for the Employee Interface. A queue is an ordered collection of items where the addition of new items happen at one end and the removal of existing items occur at the other end. This structure of a queue using the ordering principle of first in first out also known as "first come first served". In the Employee Interface, the queue is used when orders come in from the system the chef is notified first of that one and works on that meal and then serves it out which removes it from the queue. This performance of accessing is O(n) and to remove is O(1) this is a very efficient data structures to use for this scenario.

There will be an items table within the SQL database. A database is good for querying and selecting items based on attributes. This tables primary key will be foodId and the other columns in the table will be food name, food price, food

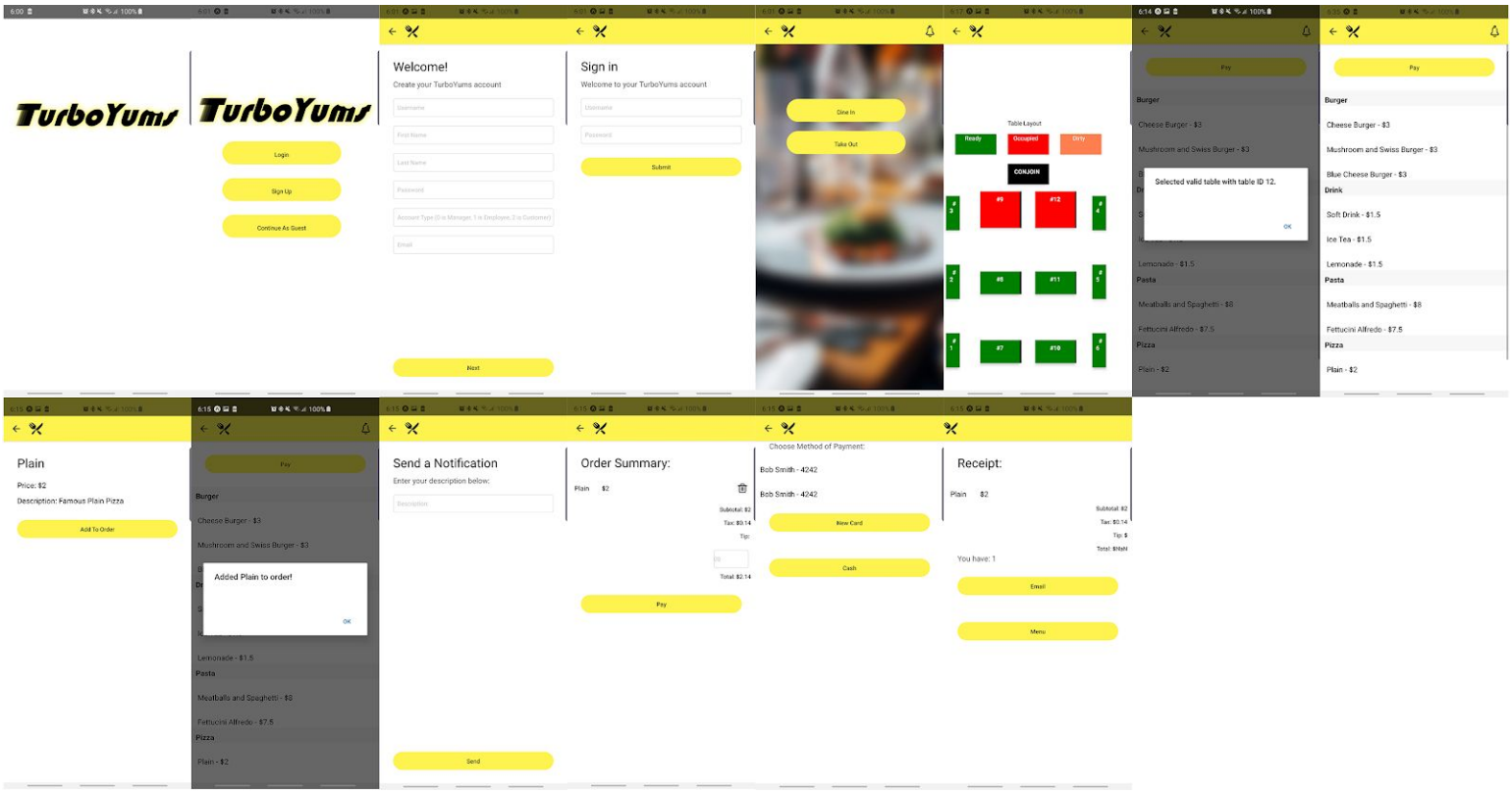
ingredients and to improve efficiency it may be helpful to add common allergies and restrictions as columns for example, contains nuts in a meal.

In order to keep track of employee working information, such as time worked and current status, we will use a SQL database. The table employee in the database will have primary key, employee's username, so that we can easily find an entry pertaining to that employee. The columns keep track of the employee's time clocked in, time clocked out, whether or not the clocking was done on site, their current clock status, the total number of hours worked, each employee's individual wage, and the estimated pay that is to be received on the next pay cycle.

When a customer is preparing an order, the system will create an order object, which will include a collection of items included in the order, as well as an associated table, customer, and server. The method of payment will be represented by an object, there will be a field dictating what method of payment this is (cash, card, etc.), a field associating it with a user, and also fields detailing the information regarding the method of payment (card number, expiration date, billing address…). When a payment goes through, This will also update the User object's rewards information. All the objects used are stored tables in our relational SQL database.

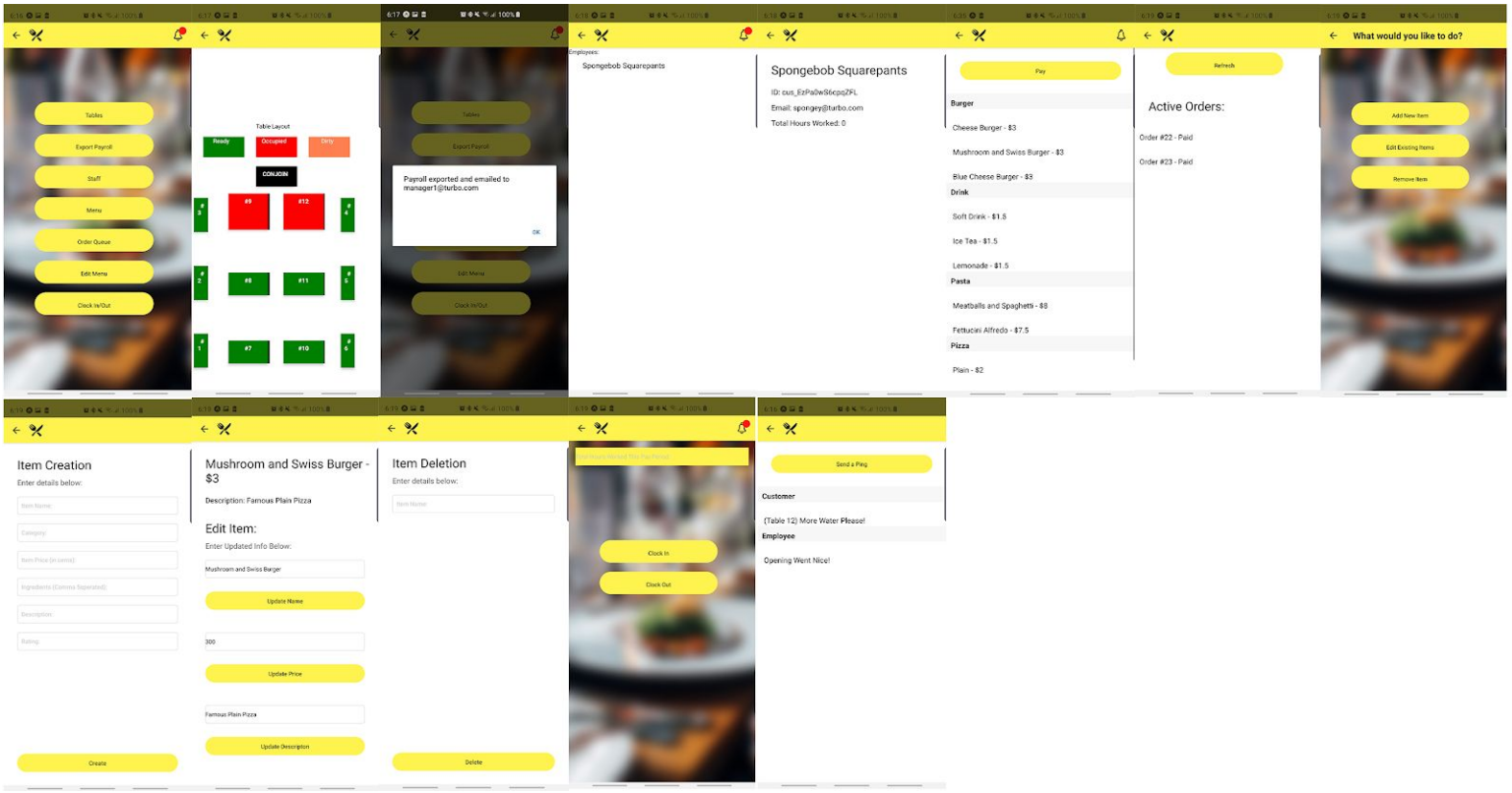
11 User Interface Design and Implementation

Customer Interface



Once customers launch the app, they are greeted by our splash screen followed by a home page. From there, they are free to create a new account, login to their existing account, or continue as a guest. Once logged in, they have the option of choosing “Dine In” or “Take Out”. Take Out takes them immediately to the menu, while Dine In allows the customer to select an available table of their choosing. The customer is then able to browse the menu and add whatever they like to the order. While logged in, they can tap the bell icon on the top right to send notifications to their server for anything they might want to notify them about. Once satisfied with their current selection, they are able to view their order and pay for it with their payment method of choice. They are finally represented with a receipt transaction and the ability to have the app email them the receipt.

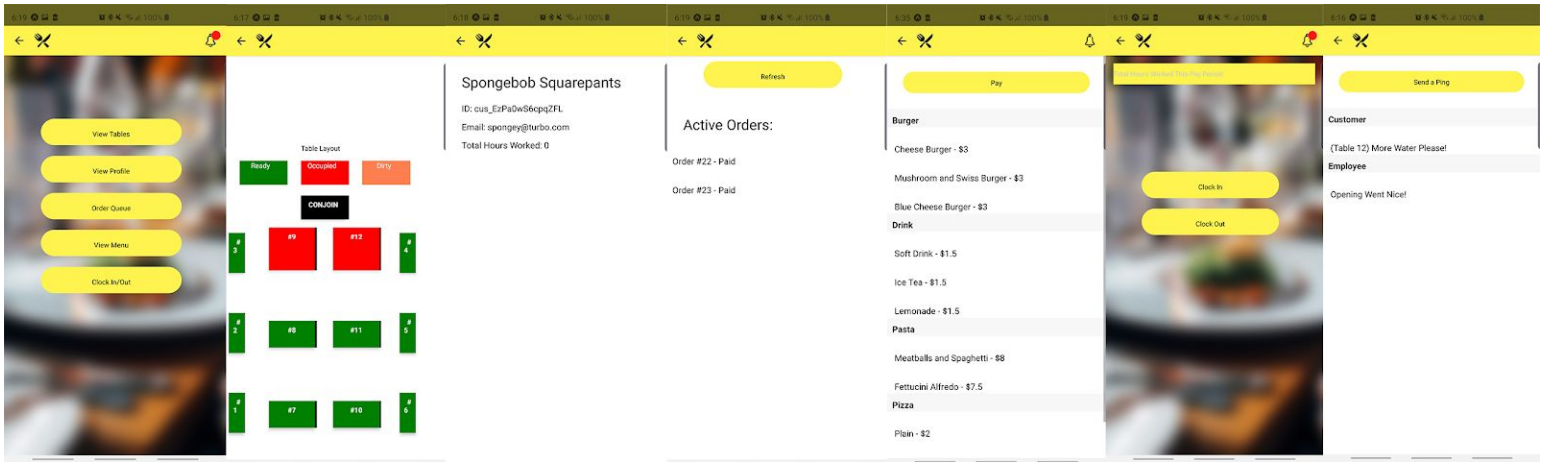
Manager Interface



Once a manager logs in via the login screen mentioned before, they are presented with a portal with access to several features. The first option shows them the current status of all tables in the restaurant. The second button causes the app to send the manager an email containing Payroll information. The third button takes the manager to an interface where they can view all current employees, where each can be individually selected to view employee information. The next button allows the manager to view the menu as normal. Next, Order Queue allows the manager to see the status of all current orders. The Edit Menu features allows the manager to add items to the menu as well as edit and remove existing ones. Adding an item simply requests that they fill in the required information. Editing an item allows the manager to select an item from the menu and edit all of the properties manually. Item Deletion requests the item name and deletes the item from the menu. The final button brings the manager to an interface that allows them to clock in or out of their

shift. Along with this, clicking on the top right bell icon allows managers to see all current pings from customers and coworkers alike, as well as giving them the option to send a ping themselves.

Employee Interface



The employee interface is similar to the manager interface, however there are less features due to them not having managerial access. The View Tables button allows them to view all available tables in the restaurant. The View Profile option allows them to view their own employee information. Order Queue presents the employee with a queue showing the status of all existing orders in the restaurant. Employees are able to access the restaurants menu via this interface in the event they want to order food for themselves. The final button allows them to clock in/out of their shift. Clicking on the bell icon on the top right of the interface allows the employee to view all existing pings/notifications from customers and coworkers alike, as well as being able to send pings themselves.

12 Design of Tests

12.1 Test Cases

Test Case Identifier: TC-1 **Use Case Tested:** UC-16 **Pass/Fail Criteria:** The test will pass if the user is able to successfully log in to the system. The test will fail if the user inputs an invalid username or password. **Input Data:** username, password	
Test Procedure:	**Expected Result:**
Step 1: User types in an invalid username and/or password. Step 2: User types in a valid username and password.	Server denied the log-in attempt. Message pops up, instructing the user to try again using a valid log-in. Server allows the log-in attempt. Depending on the user type, the system allows the user to access the specified interface.

Test Case Identifier: TC-2 **Use Case Tested:** UC-7 **Pass/Fail Criteria:** The test will pass if the user is able to successfully clock in and out for their shift and the server is able to verify their location. The test will fail if the user is unable to clock in or out, or the server is unable to verify the location. **Input Data:** Selecting either the clock-in or clock-out button	
Test Procedure:	**Expected Result:**
Step 1: User selects to clock-in or out outside of the restaurant. Step 2: User selects to clock-in or out within the restaurant.	Server allows the clock-in/clock-out attempt. The employee's current status updates to keep track of whether they are currently clocked in/out. Database records that a clock-in/out attempt has been made in a foreign location, along with the current time, user, location, and updated clocking status. System allows the user to access the specified interface. Server allows the clock-in/out attempt. The employee's current status updates to keep track of whether they are currently clocked in/out. Database records the clock-in/clock-out attempt, along with the current time, user, location, and updated clocking status. Depending on the employee type, the system allows the user to access the specified interface.

Test Case Identifier: **TC-3
Use Case Tested: UC-6
Pass/Fail Criteria: The test will pass if the user is able to successfully select different filters and the menu is able to adapt in real time and filter out the items that contain the selected ingredients. The test will fail if the menu does not properly remove the items containing the selected filters.
Input Data: selecting different filters checkboxes

Test Procedure:	**Expected Result:**
Step 1: User selects different ingredient in the menu options	Menu allows the filter options to be selected and the new get request to the server will not contain the items made with the filtered ingredients. The remaining items are re-organized and the user is able to order any of the remaining items.
Step 2: User does not select any filters	Server get request contains all items available on the menu and the menu displays them.

Test Case Identifier: TC-4
Use Case Tested: UC-9
Pass/Fail Criteria: The test will pass if the user is able to successfully able to add items to the checkout cart and place the order
Input Data: item array

Test Procedure:	**Expected Result:**
Step 1: User selects different items and presses confirm order button	Server verifies that all items added to the cart are valid and the system prints “order successful”. The order is sent to the chef’s profile to be added to the order queue.
Step 2: User does not select any items and presses confirm order button	System recognizes the order as invalid and and indicates that the user must add items to the cart before placing an order.

Test Case Identifier: TC-5
Use Case Tested: UC -2
Pass/Fail Criteria: This test will pass if the user puts a valid credit or debit card into the program and it is recognized and accepted, or if an invalid credit card is inputted and is rejected. This test will fail if a valid card number is put in and rejected, or if an invalid card is accepted.
Input Data: Credit or debit card information

Test Procedure:	**Expected Result:**

Step 1: Type in incorrect card information. Step 2: Type in correct card information.	System indicates an invalid card number to the user, and then prompts the user to try again. The system processes the card number, accepts the card number and then deducts the charge from the users account and add the balance to the restaurant account.

Test Case Identifier: TC-6
Use Case Tested: UC-2
Pass/Fail Criteria: The test passes if the program properly sums the total amount of items, and applies tax and tip appropriately. The test case fails if the program cannot properly sum the items, apply tax or tip.
Input Data: User selects which items of food that they wish to add to the cart and input the tip amount they would like to add

Test Procedure:	**Expected Result:**
Step 1: User finishes selecting the desired items.	System will keep track of the selected items and their prices.
Step 2: Automatically calculate appropriate total.	System will add the price of the items together, calculate and add tax, and add tip in order to formulate the total.
Step 3: Manually calculate the total and compare the results with the program.	Person must calculate the total in order to check whether or not the system is doing it correctly.

Test Case Identifier: TC-7
Use Case Tested: UC-11
Pass/Fail Criteria: The tests passes if the program successfully calculates the proper amount of reward points for the user and adds it to their user account
Input Data: Customer Attendance

Test Procedure:	**Expected Result:**
Step 1: User must have an account or create an account. Step 2: User must log into their account in order to have points awarded to them.	User has a fully functioning account. User successfully logs in.

Step 3: System calculate the appropriate amount of total points the user has. Step 4: Manually calculate the total amount of points that they should have and compare the results with the program.	The program is successfully able to add the proper amount of new reward points to the existing balance. Compare correct answer to the answer that the program calculated.

Test Case Identifier: TC-8
Use Case Tested: UC-12
Pass/Fail Criteria: The system passes if the user is able to successfully apply their reward to their order, ie:if the reward was a free beverage the user should not be charged for that beverage. The test fails if the award is not properly applied to the bill

Test Procedure:	**Expected Result:**
Step 1: The user must select the reward that they want to redeem and must meet the criteria for that reward. Step 2: The user goes to pay his bill. Step 3: The user pays his bill.	The user successfully selects that reward that they would like to get with their redeemed points. The system displays the corresponding screens and allows the user to input his payment information. The user is successfully able to pay the bill and did not have to pay for the item that was awarded.

Test Case Identifier: TC-9*
Use Case Tested: UC-5
Pass/Fail Criteria: The test will pass if the user is able to successfully rate the food by selecting how many stars it deserves. The test fails if this is not so.

Test Procedure:	**Expected Result:**
Step 1. User has placed an order and eaten his/her food. User then presses the payment option on the device. Step 2: User is on payment screen and user will be able to select how many stars each item deserved.	The system takes the user to the payment screen and is able to see the food items he/she had ordered. When the amount of stars are selected then the stars should turn a solid color to indicate that was a star selected.

Test Case Identifier: TC-10***
Use Case Tested: UC-15
Pass/Fail Criteria: The test will pass if the manager or waiters are able to change a table's status to either Ready, Occupied, or Dirty as desired. When a table is pressed the table's color/status should change immediately

Test Procedure:	**Expected Result:**
Step 1: The employee presses on the view tables button. Step 2: The employee taps on a table.	The system displays the current table layout of the restaurant. The table will visually indicate that it's status has changed from the previous state to the new state.

Test Case Identifier: TC-11***
Use Case Tested: UC-15
Pass/Fail Criteria: The test will pass if all conjoined tables allow orders placed from the connected tables to be treated as if they were all placed from one table.

Test Procedure:	**Expected Result:**
Step 1: The customer will tap the conjoin button. Step 2: The customer taps two different table to be conjoined.	The screen will indicate that two tables must be selected to be conjoined. The database will give the two different tables the same order id so that the kitchen interprets the incoming orders as one large table.

Test Case Identifier: TC-12***
Use Case Tested: UC-14
Pass/Fail Criteria: The test will pass if customer is navigated to menu after selecting a table that is Ready (green).

Test Procedure:	**Expected Result:**
Step 1: The customer will login with credentials. Step 2: The customer taps a table that is green.	The current table layout will be displayed showing all tables' statuses. The database will check to see that the table is ready and will navigate the user to the menu. The selected table will have it status changed from ready to occupied.

Modified Test Cases

Identifier	Test Case	Comments
**TC-3	Testing the menu filtering feature.	Food filtering has been implemented in a separate branch and presented in our second demo, however, there were merge conflicts that interfered with implementing it into the master.
*TC-9	Testing the rating feature.	This is something that will be part of future work. Additional work on this application may implement these features.
***TC-10	Testing that the tables are able to change status once tapped.	This is a new test case we have added, since we created new features.
***TC-11	Testing the conjoin tables feature.	This is a new test case we have added, since we created new features.
***TC-12	Testing that selecting on an available table will navigate the customer to the menu.	This is a new test case we have added, since we created new features.

(*) This test case will not be implemented by demo 2, and is available for future development.

(**) This test case has been modified since the previous report.
(***) This test case has been added.

12.2 Test Coverage

All of our test cases cover the essential classes that are necessary to the operation of TurboYums. As we develop more of our classes and methods, we will add and adjust test cases as needed. Testing will be done for as many possible cases that a class could go through. The test procedures for the classes will have the format of both a pass procedure and a fail procedure in response to a user input. If the test fails due to a faulty user input, the system will prompt the user to try again and correct the input. An example of a test is testing the login screen, where if the user inputs username and/or password incorrectly, the system will have the user enter their information again. This fail case will repeat until the user inputs their correct information, at which point the user can login and start their shift. There are test cases for inputs customers, employees, and managers. Some test cases for the customers could be when they are creating an order. There could be a test case that makes sure if a customer wants to remove an item from an empty order, it should display an error on the screen. Test cases for an employee trying to modify the table status could be that an error shows up if an employee is trying to clean a table that current has customers sitting at it. Our test cases will be specific so that it can cover all possible cases. We want our app to be efficient as possible with no errors which is why we will have test cases to show that TurboYums works properly and has been tested thoroughly.

12.3 Integration Testing

We decided on utilizing the strategy of bottom up testing. Bottom-up testing is an approach to integrated testing in that the lowest level components are tested first, and those components are used to help test the higher level components. This ensures that the building blocks of the code operate as needed before using them in other sections of the code. With this approach, we will be able to test what we are working on as we complete the different layers of the project. This method of testing is the most appropriate for our project. If we were to choose a different method of testing, it would be much more difficult to understand what the cause of the bug is. Whether the issue is coming from the integration of the different code components or if the problem is with how the classes are fundamentally designed and coded. By understanding the relationships between the objects in the system, the bottom up testing approach is more efficient and straightforward in that you can quickly narrow down where the problem lies prepping it for remedy. A concrete way of representing the components of our system and how they would relate in this context would be that we test each of the employees independent and personalized tasks in the application. For example, for the chef, we would test the implementation of the queuing system and make sure that it updates the server when the chef proclaims that a dish is cooked and ready to serve. After going through the individual functions, we test the features which call for interactions between more than one object or class.

13 History of Work, Current Status, and Future Work

January 28 - February 1:

We got together in class during this time to form a team to move forward with on a project. Choosing a project was no difficulty as a few of the team members had experience working at a restaurant/fast food chain. Therefore, we found it would be fitting if we chose the Restaurant Automation idea as this would solve a problem that a members felt especially connected to. Thus, we moved on to writing the proposal. The proposal was written as a collaboration effort over google drive where we split work to complete it.

February 2 - February 14:

We used the feedback from the proposal to plan our steps during this time. Some of the feedback was to come up with solutions to potential problems the TA listed out that we may not have thought of. Also, to look into more of the previous year reports on the functions they created. Everyone contributed their assigned parts throughout this period over the shared drive. The team got together to discuss the programs we plan on using.

February 15 - March 1:

During this time, we worked on completing the first report and continued to collaborate on the google drive. We took into consideration the feedback that was given and fixed those parts in the report. We continued to meet outside the classroom to solidify the software that would be used and made sure everyone understood the purpose of why we chose a certain language to program in. We spent a few days setting up our laptops to have all those necessary softwares downloaded such as docker, react native and MySQL. During this period, we also spent time discussing how we would split the work and who would be on which team. We had a payment team, menu team and clocking in/out team with each one having a few features to implement for the demo.

March 2 - March 15:

With the full report 2 due at the beginning of March, we worked on completing that first before starting to program. Everyone made an effort to complete a part of the report when he/she had a chance to. With a few weeks before spring break, we met up a few times in person to resolve any downloading issues people had. We then broke into sub groups to discuss among ourselves how we should divide the work and what features to tackle first. Then we started coding the necessary classes that were needed for our subgroups.

March 16 - March 26:

During spring break, everyone spent a lot of their time working on getting the frontend and the backend done to present at the demo. The features we focused on were restaurant menu, clock in/out for employees and payment of an order. For menu the user is to be able to view menu and add items to an order. For clock in/out the features created were user profile and an employee portal. Lastly, the payment team worked on being able to pay an order and earning rewards. Each subgroup had a branch on Github and on the last day of spring break, we merged all the branches and ran a final test on it to make sure the features were working smoothly. We then also spent that day to create the brochure and powerpoint for the demo.

March 27- April 10:

We spent the day everyone got back from spring break to practice for the demo. We rehearsed how we were going to explain the features we programmed and how we were to demonstrate it. After the demo presentation, we regrouped to discuss the feedback the TA's gave us at the demo and wrote it down to look into it at the next meeting. The next meeting, we first discussed the feedback of fixing up the menu by adding images and making the software easy for restaurant owners to use as they do not know any scripting language. We took these into consideration and again split up into new subgroups. The features that are going to be implemented are kitchen/management/host/server/customer interface, email receipts, table layout and adjustments, redeeming rewards, employee portal and filtering/remove for menu.

April 10- April 22:

We are going to be working on report 3 to finish it completely before the demo. Our main focus is going to be on implementing the features as mentioned previously early so we can work on putting finishing touches. We are pushing ourselves to showcase new features and will be spending this time making sure our software functions properly through detailed tests. The team will be meeting up with their sub groups various times and each subgroup will be again working in their branch on GitHub when committing. The few days before the demo is when we will be merging the branches and fix any bugs we may face.

April 22 - May 6:

We have finished our demo and did really well with our overall grade. With the final report deadline coming up, we are working to finish the report by revising the report and adding more detail where necessary. Also, we are updating any diagrams that are outdated. Our reflective essays are being written to reflect on our perspective on the difficulties faced and the benefits of working as a team. Overall, our demo and report has been very successful. We all collaborated/communicated really well with each other and split the amount of work evenly to be able to create and cover more features for this demo.

14 References

Richards, Mark. “Software Architecture Patterns.” O'Reilly | Safari, O'Reilly Media, Inc., https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

Marsic, Ivan. “Software Engineering.” *Professor Zoran Gajic - Home Page*, www.ece.rutgers.edu/~marsic/Teaching/SE/.

Cinergix Pyt. Ltd., “Diagramming & Collaboration” https://creately.com/app/

Hanov Solutions Inc., “WebSequenceDiagrams” https://www.websequencediagrams.com/

Dimitrovski, Stephan, et al. *Why W8*. 2018, *Why W8*. https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2018f-g4-report3.pdf

El Warraky, Omar, et al. “Food•E•Z.” *Google Sites*, sites.google.com/site/sefoodez/home. https://hub.packtpub.com/what-is-multi-layered-software-architecture/

“Distribution Dashboard | Android Developers.” Android Developers, developer.android.com/about/dashboards. https://developer.android.com/about/dashboards

“IOS.” Wikipedia, Wikimedia Foundation, 5 Mar. 2019, en.wikipedia.org/wiki/IOS. https://en.wikipedia.org/wiki/IOS