Report #3

QuickBytes

http://35.188.32.57

Patrick Carra
Kegan Ronholt
Mark Norfolk
Clint Ryan
Mila Hose

November 16, 2020

Table of Contents

Individual Contributions Breakdown

Topics	Patrick Carra	Kegan Ronholt	Mark Norfolk	Clint Ryan	Mila Hose
Individual Contributions Breakdown	20%	20%	20%	20%	20%
Summary of Changes	20%	20%	20%	20%	20%
Customer Statement of Responsibilities	20%	20%	20%	20%	20%
Glossary of Terms	20%	20%	20%	20%	20%
System Requirements	20%	20%	20%	20%	20%
Functional Requirements Specification	20%	20%	20%	20%	20%
Effort Estimation using Use Case Points	20%	20%	20%	20%	20%
Domain Analysis	20%	20%	20%	20%	20%
Interaction Diagrams	20%	20%	20%	20%	20%
System Architecture and System Design	20%	20%	20%	20%	20%
Algorithms and Data Structures	20%	20%	20%	20%	20%
History of Work, Current Status, Future Work	20%	20%	20%	20%	20%
References	20%	20%	20%	20%	20%
Editing	20%	20%	20%	20%	20%
Organization	20%	20%	20%	20%	20%

Summary of Changes

- Use Case Diagram Revised
- Edited Fully Dressed Use Case Descriptions
- Updated Interface Requirements
- Added Sequence Diagrams
- 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 requirements and specifications for the following classes:
    - Manager
    - Host(ess)
    - Chef
    - Server
    - Busser
    - Delivery Liaison
    - Customer
- Updated Class Diagram
- Updated Interface Designs and Descriptions
- Included OCL
- Updated Description of Diagrams
- Added History of Work, Current Status, and Future Work section

1. Customer Statement of Requirements

QuickBytes is a fully automated web service that utilizes technology to create simpler and more efficient restaurant transactions for both employees and customers. In order to accomplish a solution that would be most effective for all parties involved, we have designated a few key roles and actors, which are described in more detail in the following section.

1.2 Customer Statement of Requirements

Manager

Problem: As a manager, I must be on top of my game. Taking care of an entire restaurant and keeping up with employees is tough stuff. With the way my business is set up, I must use paper for everything. If a customer has a grievance, then they leave a note. My workers use paper slips to clock in and their schedule is written down on a calendar every month. I also write down how much inventory we have and physically check when it's time to reorder. Lastly, I budget my expenses via paper as well! If I could find some way to consolidate all these tasks into something more manageable, I would have more time to promote my business!

Treatment: I would love an application that would allow me to track complaints and maybe even offer compensation for my guests. I wouldn't be too upset if this program also allowed me to schedule employees for work, track their salaries, and clock them in and out. If the program would calculate how many ingredients we use on average and tell us the deadline time to restock, that would be great, but it would be even better if it could calculate how much money we would have left after purchasing those items out of our monthly budget.

Plan of Action: The QuickBytes restaurant management program allows authorized personnel to manage, develop, and track their businesses. With this application, managers can schedule employees for work, manage break time, set employee performance statuses, track employee salaries, restock ingredients, calculate monthly or yearly income and expenses, resolve customer complaints, and more. It will track peak hours to show the busiest hours of our restaurant so that the manager can know when more employees will need to be staffed. This position will also handle worker grievances so that employees can send in grievances with customers or other employees with the choice of being anonymous.

Host(ess)

Problem: First impressions are everything in the restaurant world! Guests form an opinion about our business within moments of walking through the doors. Our customers require a warm greeting as well as quick service upon entering the establishment. With Covid-19, we need to adapt the way we think about a dine-in service but still provide a great experience for the guests. To achieve a professional warm atmosphere in a busy, fast-paced restaurant, I need to have instant access to information about available seating, estimated wait times, and guest reservations. Time is everything in this business and every second is precious.

Treatment: I need an application that keeps track of the tables in the restaurant. If I had a walk-in customer, I could ask their group number, locate a table in seconds, seat them, and mark on the application that the table is occupied. A current concern with seating is maintaining adequate social distancing, the app should find a way to ensure that guests are seated at least 6 feet apart. Ideally, I would want the app to also suggest possible potential table transfigurations that both maintain social distancing, but seat a larger walk-in group or family that comes in. With seat reservations, I need to find their information quickly, take them to their table, and register their arrival. In response to Covid-19 we are offering the option for guests to use digital-only menus, as well as the amount of contact with staff that they would prefer, I will need to ask their preference and direct them to the QR code that displays the online menu.

After seating walk-ins or reservation guests, I need to take their initial drink orders and should be ready to order appetizers in case they ask. I would also want the guests to be able to decide on the app how much interaction they would prefer with the staff- Perhaps a “Low Contact” experience vs a “Standard Contact” experience The app should enable guests to select a table and initial drink choices online before coming. This selection would allow me to simply guide them to their seats upon their arrival and bring the drinks. I also need the website to include a place where guests can check for information before calling. If the guests need to call in for information, I should have quick access to anything they may want to know. The system also needs to allow guests or myself to place reservations for future dates. If all of our tables are full, the app should suggest an estimated wait time, and manage the people on a waiting list.

Plan of Action: The QuickBytes solution to this problem is elegant and efficient. Our web application will store all restaurant information and have it formatted for quick access. The FAQ on the site will store information that will reduce the number of customers calling with questions. The online ordering and reservations system will also decrease the amount of people calling in to make reservations. Through the app, Hosts and Hostesses will be able to make and find reservations, find available tables, seat guests, place initial orders for tables, and manage the guest waiting list.

Chef

Problem: My kitchen is often a high stress, high speed environment that requires accurate orders, efficient processes, and timely delivery for my crew to be successful. Accurate orders ensure that customer's needs and wants are met including any special accommodations for any food allergies. Any returned food or mistakes could result in a repeat of preparing that dish resulting in lost time and increased frustration for my staff. Accurate orders also help to ensure kitchen staff can efficiently complete the order without needing additional input from the servers.

Treatment: What we need is a way to receive accurate orders from the servers/customers, an efficient way to organize and prioritize orders, and a way to notify the servers that orders have been completed. We also need a way to receive more input from the customer or server regarding an order if any confusion arises. It would also be nice to have a timer to keep track of how long a customer has been waiting.

Plan of Action: The QuickBytes restaurant management application will provide a way for kitchen staff to prioritize orders and keep track of how long a customer has been waiting. It will also allow kitchen staff to request additional information from the server/customer if any confusion may arise. There will be an additional queue available for any returned orders so that these can be prioritized above others to ensure customer satisfaction. Finally, orders that have been completed will be placed back into the servers' queue for pickup.

Server

Problem: By the time a customer is seated in my area, they are often somewhat disgruntled before my first interaction with them. They have experienced long and inaccurate wait times, they are hungry, and they often feel that other customers have erroneously been seated before them, because they see customer groups of variant sizes being seated first, due to the availability of tables. This makes it hard to interact with the customers in a positive manner, because they are already upset by the time they are seated in my area. This has a negative impact on my tips and sometimes results in requests to speak with a manager at the end of their stay, although I have done my very best to provide them with a positive experience.

Treatment: I need an application that will allow customers to have an accurate idea of wait-times before they are seated in my area, so they are not overly frustrated by their wait by the time they have been seated. It would also be very helpful to have a technology solution that allows customers in my area to request that a manager come over and speak with them so that I do not have to take time away from other customers in my area to hunt down a manager to go speak with a specific table. It would also be great if the application could provide an estimated wait time for the manager to arrive at the customer's table, so that the customer does not become even more upset.

Plan of Action: QuickBytes provides an elegant solution to this problem by allowing the customer to get an idea of their expected wait ahead of time, or to schedule a specific time to arrive and be seated at the restaurant (make a reservation). With the ability to know their wait ahead of time or plan their arrival, the customer will waste less time waiting around at the restaurant and can plan accordingly so they are not overly hungry by the time they arrive. This will allow more pleasant interactions between the server and the table of customers they are serving. QuickBytes will also allow customers to submit a request to speak to a manager, which will go to a queue with the table number. The next available manager will go and speak with the table, and then remove the request from the queue once they have helped provide a satisfactory resolution to the customer.

Busser

Problem: As a busser, I am under a lot of pressure to clear tables as quickly as possible and notify the host/hostess that they are available so that more customers can be seated. This is especially crucial during rush hour, when we often receive a lot of complaints regarding wait time. We do our best to accommodate customers and serve people as quickly as possible, but circumstances are often out of our control, such as the time it takes to prepare a party's meal or how long a table lingers to chat after their meal has been finished and their bill has been paid. I need an efficient way to notify my seating co-workers that a table has been cleared and is ready.

Treatment: Many restaurants already have POS systems on each table, which allow customers to do things like play games while they wait for their food to be served or pay their bill at the end of their meal without having to wait on the server. I need functionality that will further expand on this technology by allowing bussers to signal that a table is clean and ready for new customers through the POS technology system on the table. It would also be great if the technology could help keep a running tab on the tables that need to be cleared, in order of their priority, so that I can efficiently clear each table in order.

Plan of Action: QuickBytes will expand on the POS systems available at each table in a restaurant by providing a list of tables that need to be cleared. The tables will go into the queue based on both the order in which they are cleared, as well as the need for specific table size, based on the parties that are waiting to be seated. This way, the app ensures that tables are being cleared as soon as possible and clears tables first, which will provide an opening for a new group of customers to be seated. Once the busser has finished clearing a table, they simply need to signal that the table is clear using QuickBytes, which will remove the table from the list of tables waiting to be cleared, and send it back into the list of available tables for the host/hostess to use in seating more customers.

Delivery Liaison

Problem: Take-out orders require careful management in order to ensure customer satisfaction. As the delivery/pickup liaison accurate order entry is vital to ensure that I can deliver orders without confusion or frustration. Inaccurate or incomplete information will require me to contact the customer for more information. Prioritization and requested pickup/delivery times are a must since customers' needs may differ. Getting orders out to delivery drivers efficiently is difficult due to availability, order completion, and drive times.

Treatment: I need a system that will allow accurate order entry. I also require the ability to manage when orders are sent to the kitchen so that food does not get cold waiting for customer pickup or delivery. It would also be helpful to be able to notify delivery drivers that there are orders ready to be delivered.

Plant of Action: MMPK restaurant management application will provide a way for the delivery liaison to accurately and quickly enter orders. The customer may also enter their own orders for delivery. The liaison will have the ability to reprioritize and organize orders based on delivery and pickup times. Orders will be sent to the kitchen staff at the appropriate times by the liaison for fulfillment. The liaison will be assisted with coordinating delivery by mechanisms that allow them to view when an order should be completed and notifying drivers that an order is ready for delivery.

Customer

Problem: When I visit a restaurant, I expect convenience, cleanliness, quality of service, and quality of food. Whether I am a walk-in customer or have a reservation, I enjoy being greeted at the door and given clear instructions. I appreciate quick table selection and seating. If a table is not immediately available, I expect an efficient and well managed wait list. I want the option to make reservations and place orders online, but also desire the flexibility of choosing my meal at the establishment. I often enjoy trying different dishes but detest the idea of ordering something that I don't appreciate or isn't prepared well. It seems like a waste of time and an unnecessary hassle to flag down a waiter/waitress when I want to add something simple to my order or request refills for drinks.

Treatment: The application needs to have the option to reserve my table and/or order food online. I need digital menus as an option if I don't want to touch a physical menu. I appreciate suggested drink pairings with my meal choice. The app needs a convenient way to process payments and tips. I wonder if there is a way to pay for a dine-in meal before-hand and add the tip after service is completed. The payment option should include a way to split the bill. Meals should have visible ratings. A good feature might suggest dishes based on foods that I like. I need to have the option to add items to my order from the app. I would like a "call" button that alerts the waiter that I need service without having to raise my hand or flag them down.

Plan of Action: The QuickBytes application ensures convenient and efficient service for your customers. Our program will allow customers to place orders, reserve tables, and pay with no hassle. We will create and implement a rating system, allowing guests to choose highly rated meals. A "Smart Choice" feature will be built. "Smart Choice" elicits and stores information regarding a guest's favorite dishes or foods and offers meal options based on their tastes. Drink suggestions for each meal appear when a plate is chosen. While dining, guests can simply press a "call service" button on their app to alert the servers that they need attention. The application allows access to the meal ticket and the ability to add items. At the end of the meal, payments are processed swiftly and seamless, allowing for bill splits and tips.

2. Glossary of Terms

Application - A software program designed to perform specific functions and activities for a set of users.

Busser - Keeps track of the dirty tables, clears them and updates the table status when s/he is done cleaning it.

Chef - Accepts incoming orders from servers and prepares meals specified by each order.

Cross Platform - An application that can be used on a variety of device types.

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 QuickByte's network where restaurant information is accessible to certain user accounts.

Dine-in - To order and consume food at a restaurant location.

Host(ess) - Welcomes incoming customers and assigns servers and tables to customers.

Menu - A comprehensive list of meals and food items available for ordering.

Manager - Supervises restaurant staff, ensures customer satisfaction and confirms building security at closing.

Object-Orientation - A paradigm based on the concept of "objects", which can contain data and code.

Operating System - A system software that manages computer hardware, software resources, and provides common services for computer programs.

POS - Point of Sale System

Python - An interpreted, high-level and general-purpose programming language.

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

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

Restaurant Automation - The process of automating restaurant software to increase the efficiency of restaurant staff and improve customer satisfaction.

Screen - The window in which the user interface is displayed.

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

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

Table Chart - Displays the floor plan of a restaurant, including table location and table identifiers.

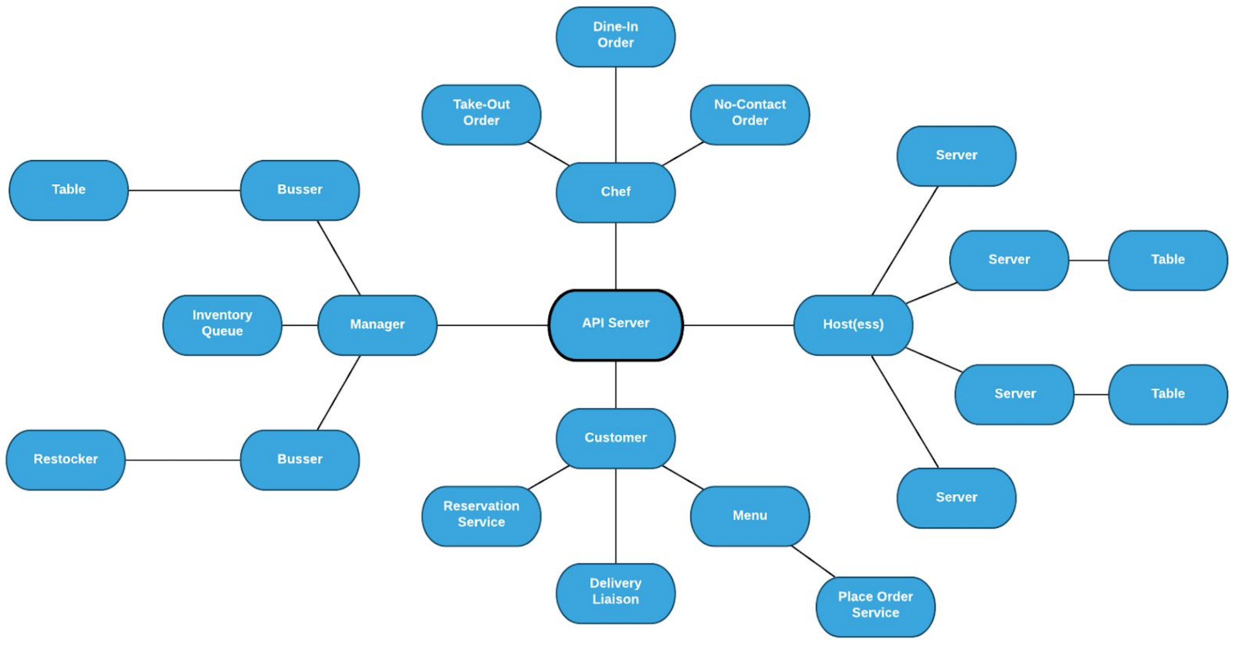
Table Status - A textual status that indicates whether a table is occupied, reserved, dirty or available to be assigned to guests.

Take-out - Orders that are prepared for customers that picking up their order instead of eating at the restaurant.

User Interface - The visual aspect of software that enables users to interact with visual stimuli.

3. System Requirements

3.1 Business Goals



The above diagram illustrates the business goals, key actors and their associated relationships.

3.2 Enumerated Functional Requirements

Priority 5 has greater precedence than Priority 1

Identifier	Priority	Requirement
REQ-1	4	The application will allow the Chef to notify the server that the order is ready and place the order back in their queue.
REQ-2	2	The application will allow the Chef to request more information about an order.
REQ-3	3	The application will allow orders to be organized and reprioritized by the delivery liaison according to delivery/pickup times requested.
*REQ-4	4	The application will allow the delivery liaison to notify the delivery driver/customer that an order is ready to be picked up or delivered.
REQ-5	4	The application shall track the status of tables (unavailable/being cleaned/available) and allow for assigning of tables.
REQ-6	1	The application should ensure that tables are filled in a way that enforces Covid-19 restrictions.
REQ-7	3	The application will display seating information for the host(ess).
REQ-8	5	The application will allow the host(ess) and/or guests to select “low-contact” mode.
REQ-9	5	The application shall allow guests to view a digital menu and place orders online.
REQ-10	2	The application website should have an FAQ page.
REQ-11	4	The application shall allow guests and/or host(ess) to reserve tables.
REQ-12	2	The application should have suggested drink pairings with a selected dish.
REQ-13	5	The application shall track orders and process payments digitally.
REQ-14	2	The application should have a review feature.

REQ-15	4	The application should have the option to request service.
**REQ-16	3	The application will allow a busser to make a request for additional cleaning supplies.
**REQ-17	2	The application will automatically translate text based on the user's location.
**REQ-18	1	The application will provide an option for the customer to receive a copy of their receipt in different formats (i.e. paper, e-mail, none).
**REQ-19	3	The application will allow the user to subscribe to updates in order to receive information about new menu items, specials, etc.
**REQ-20	4	The application should provide the user with real-time updates regarding their order wait time and expected order completion time.
**REQ-21	5	Employees should be able to use the QuickBytes application to clock in and out of work, as well as to track their weekly hours worked.
**REQ-22	5	A QuickBytes user should be able to register for an account based on their job duties or administrative level (i.e., busser, customer, chef).

Modified Functional Requirements

Identifier	Priority	Comments
*REQ-4	4	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-16	3	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-17	2	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-18	1	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-19	3	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-20	4	This feature is available for future work. Additional progress on the application may include this requirement.

*REQ-21	5	This feature is available for future work. Additional progress on the application may include this requirement.
*REQ-22	5	This feature is available for future work. Additional progress on the application may include this requirement.

(*) 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 Enumerated Non-functional Requirements

Identifier	Priority	Requirement
REQ-23	5	The application should be easy to use for the average person.
REQ-24	4	The application should be visually appealing to the user.
REQ-25	5	The application should support a fully functioning restaurant.
REQ-26	5	The application should be accessible from any platform.
**REQ-27	5	The application should be able to handle heavy user traffic for consistent throughput and availability.
**REQ-28	4	The application should prompt the user to perform updates in order to pull in the latest features and bug fixes.

Modified Non-functional Requirements

Identifier	Priority	Comment
*REQ-25	5	Testing under heavy traffic conditions is not possible at this time, and is available for future work. Additional progress on the application may include this requirement.
*REQ-26	4	Automatic updates are still in progress and are available for future work. Additional progress on the application may include this requirement.

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

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

3.4 User Interface Requirements

Manager Interface Requirements

Identifier	Priority	Requirement
MREQ-1	5	Login system for Manager/Owner
MREQ-2	4	Landing page contains a quick view of the budget, revenue sources, and daily tasks.
MREQ-3	3	Navbar to the left holds links to important business functions.
MREQ-4	2	The profile tab shows the Manager's information.
MREQ-5	3	The employees tab allows employees to be added.
MREQ-6	5	Income shows monthly expenses and income generated as well as a ledger functionality.
MREQ-7	4	The restock tab allows the manager to order more food items.
MREQ-8	2	The complaints tab displays both customer and worker complaints.
MREQ-9	5	The business tab allows for the scheduling of employees and their break times and also displays local events that may bring in more customers. Also shows peak business hours.
*MREQ-10	2	Pressing logout will end the user's session.

Modified Manager Interface Requirements

Identifier	Priority	Comment
*MREQ-10	2	Logout functionality still in progress and is available for future work. Additional progress on the application may include this requirement.

(*) This requirement will not be implemented by demo 2, and is available for future development.
(**) This requirement has been modified since the previous report.

Host(ess) Interface Requirements

Identifier	Priority	Requirement
HREQ-1	5	An interface for the Host(ess) to view and assign tables.
HREQ-2	3	An interface to create table reservations for guests.
HREQ-3	4	An interface to manage a waiting list.
HREQ-4	2	An interface to allow a Host(ess) to store information.
**HREQ-5	4	An interface for Host(ess) to place an order for pick-up on the customer's behalf.

Modified Host(ess) Interface Requirements

Identifier	Priority	Comment
**HREQ-5	4	This feature has been added and modified since the previous report.

(*) This requirement will not be implemented by demo 2, and is available for future development.
(**) This requirement has been modified since the previous report.

Chef Interface Requirements

Identifier	Priority	Requirement
CREQ-1	5	An interface for Chef to manage kitchen orders.
CREQ-2	5	A button to request additional information about an order from the server who took the order.
CREQ-3	4	A button to send the server a notification that the order has been completed.
CREQ-4	4	An interface for the Chef to modify food inventory.
**CREQ-5	4	An interface that allows the Chef to view orders that have been returned to the kitchen for modification and their precedence levels.

Modified Chef Interface Requirements

Identifier	Priority	Comment
**CREQ-5	4	This feature was not included in the previous report. It has been added to the requirements and implemented in completion since the first demo.

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

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

Server Interface Requirements

Identifier	Priority	Requirement
SREQ-1	5	The interface should have a way for a server to send an order from a table to the chef.
*SREQ-2	2	The interface should send a warning to the server when a customer at one of their tables calls for a manager.
SREQ-3	4	The interface should show all the tables in the server's section and display whether they are open.

*SREQ-4	3	The interface should show the approximate wait time for a server's section.
*SREQ-5	3	The interface should show how long a table has been occupied.
*SREQ-6	5	The interface should have a button for the servers to turn off the call service button on the customer's side when they have gone to a table that called them.
SREQ-7	5	The interface should have a way to send tables that need to be cleaned into the queue for the Busser.
SREQ-8	5	The server should receive a message when a chef has a question about an order.
SREQ-9	5	The server should receive an alert when an order is ready for a table.
SREQ-10	4	The interface should have a way for the server to view the orders for each of their tables and print them.

Modified Server Interface Requirements

Identifier	**Priority**	**Comment**
**SREQ-2	2	This feature is available for future work. Additional progress on the application may include this requirement.
*SREQ-4	3	This feature is available for future work. Additional progress on the application may include this requirement.
*SREQ-5	3	This feature is available for future work. Additional progress on the application may include this requirement.
*SREQ-6	5	This feature is available for future work. Additional progress on the application may include this requirement.

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

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

Busser Interface Requirements

Identifier	Priority	Requirement
BREQ-1	5	An interface to signal that a table has been cleared and is ready for service.
BREQ-2	5	An interface to display a queue of tables that need to be cleaned and cleared.
*BREQ-3	3	A button to flag a table as having an issue that will result in the table being cleared and ready for guests in the time expected.
BREQ-4	3	An interface that allows bussers to signal that they are running low on additional cleaning products (i.e., towels, sanitizing spray, etc.) so that they can continue to work without waiting on necessary supplies.

Modified Busser Interface Requirements

Identifier	Priority	Comment
**BREQ-3	3	This feature is available for future work. Additional progress on the application may include this requirement.

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

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

Delivery Liaison Interface Requirements

Identifier	Priority	Requirement
LREQ-1	5	An interface for the Delivery Liaison to enter in customer order.
LREQ-2	4	An interface to reorganize orders to prioritize for preparation by kitchen staff.
LREQ-3	4	Button to allow the Delivery Liaison to send an order to the kitchen for preparation.
**LREQ-4	2	Button to allow the Delivery Liaison to send a notification to the Delivery Driver.

Modified Delivery Liaison Interface Requirements

Identifier	Priority	Comment
**LREQ-3	2	This feature is available for future work. Additional progress on the application may include this requirement.

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

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

Customer Interface Requirements

Identifier	Priority	Requirement
CUREQ-1	3	A button to allow customers to select contact level.
CUREQ-2	5	An interface that allows a customer to place an order online.
CUREQ-3	3	A button that allows a customer to call their server over to the table.
CUREQ-4	5	An interface that allows a customer to pay for their meal.

Modified Customer Interface Requirements

Identifier	Priority	Comment
**CUREQ-4	5	This feature is partially implemented, but not fully integrated with third-party payment service providers at this time. This feature is available for future development.

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

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

4. Functional Requirements Specification

4.1 Business Goals

There are many stakeholders who have an interest in the QuickBytes system and its successful implementation. These stakeholders are described in more detail below:

1. Restaurant Owners - Have an intense interest in using this system, as it will improve restaurant efficiency, support owners by providing analysis and reporting, and improve the overall restaurant experience.
2. Employees - Have an interest in this system, as it will support their daily functions, expedite many of their more tedious tasks and improve the customer overall experience.
3. Restaurant Guests - Guests will benefit from this system through the customer experience which includes improved service, enhanced features for interaction, and expedite paying the bill.
4. Software Developers - Have an interest in the design and implementation of software systems that aid in automating restaurant processes.

4.2 Actors and Goals

Initiating Actors

Actor	**Role**	**Goal**
Customer	The customer is a patron of the restaurant who may choose to order dine-in or carryout. They will view the menu, order food, eat, and pay for their meal.	The goal of the customer is to receive their meal and pay quickly and conveniently.

Participating Actors

Actor	Role
Busser	The busier is an employee who is responsible for clearing dishes and utensils from tables and sanitizing the tables once they have been cleared. The busier monitors a queue of tables and begins cleaning the tables in a FIFO order once they are added to the queue.
Chef	The chef is responsible for preparing the orders in the kitchen. The chef will receive orders from the server and liaison and notify them when their orders have been completed or request additional information.
Host(ess)	The host/hostess is responsible for initially greeting the customers and either assigning seats or fulfilling reservations. The host can see when busboys have indicated a table as available.
Server	The server is the employee who is responsible for taking orders from customers and sending them to the Chef for preparation. The server is also in charge of delivering the food once it is ready. The server receives notifications that a meal is ready, so that they can pick it up and serve it to the customer.
Delivery Liaison	The liaison is responsible for taking customer orders over the phone and prioritizing the order in which they are sent to the Chef according to delivery and pickup time.
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.

4.3 Use Cases

4.3.1 Casual Description

UC-1: Order Tracking - Allows customers and servers to place orders.

Derivations: REQ-1, REQ-4, CREQ-1, CREQ-3, LREQ-2, LREQ-4, LREQ-5, SREQ-9, SREQ-8

UC-2: Order Status - Allows servers and chef to change/view order status.

Derivations: SREQ-1, REQ-9, CUREQ-2

UC-3: Table Management - Allows the host(ess), servers, and customers to view and assign tables. Allows the busser to see which tables need to be cleaned, and mark them available when ready.

Derivations: REQ-5, BREQ-1, BREQ-2, BREQ-3, HREQ-5, SREQ-3, SREQ-5, SREQ-7

**UC-4: Payment - Allows customers to pay for dine-in or carryout meals. Includes meal-split option.

Derivations: REQ-13, CUREQ-5

UC-5: Login - Allows employees and customers to log in and view relevant information.

Derivations: MREQ-1, MREQ-2, HREQ-4

UC-6: "Low-Contact" Mode - Allows customers, servers and/or host(ess) to place a table on low-contact mode.

Derivations: REQ-8, CUREQ-1

UC-7: Request Service - Allows customers to request their server to come by their table.

Derivations: REQ-15, CUREQ-3

UC-8: Waiting List Management - Allows host(ess) to manage the waiting list for customers. Allows customers to view their position on the waiting list and the estimated time until their meal is ready.

Derivations: HREQ-3, SREQ-4

UC-9: Employee Scheduling - Allows a manager to track and set their employees' schedule.

Derivations: MREQ-8

UC-10: Ingredient Management - Allows managers and chefs to track supplies and place orders when necessary.

Derivations: MREQ-7

UC-11: Table Reservation - Allows customers to make future reservations online.

Derivations: REQ-11, HREQ-2

***UC-12: Employee Timecard** - Allows employees to clock in/out of work and tracks weekly hours.

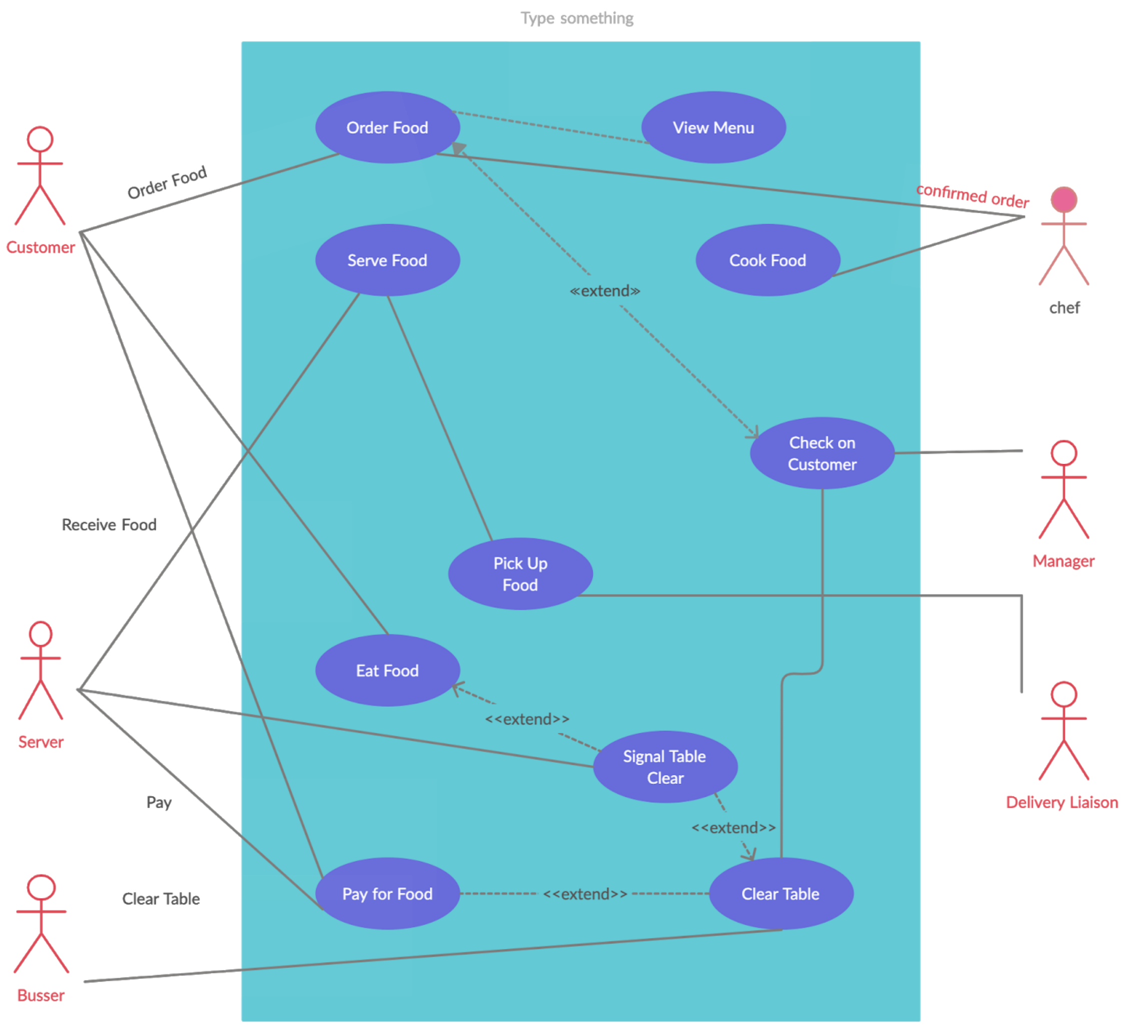
Derivations: MREQ-1, REQ-21

***UC-13: Create Account** - Allows various actors to create appropriate accounts based on their administrative level and responsibilities.

Derivations: REQ-22

Identifier	Requirement	Comments
****UC-4**	**Payment** - Allows customers to pay for dine-in or carryout meals. Includes meal-split option.	This feature is partially implemented, but not fully integrated with third-party payment service providers at this time. This feature is available for future development.
***UC-12**	**Employee Timecard** - Allows employees to clock in/out of work and tracks weekly hours.	This feature is available for future work. Additional progress on the application may include this requirement.
***UC-13**	**Create Account** - Allows various actors to create appropriate accounts based on their administrative level and responsibilities.	This feature is available for future work. Additional progress on the application may include this requirement.

4.3.2 Use Case Diagram



4.3.3 Traceability Matrix

Req't	PW	1	2	3	4	5	6	7	8	9	10	11	12	13
REQ-1	4	x	x											
REQ-2	2	x	x											
REQ-3	3	x	x											
REQ-4	4	x												
REQ-5	4	x	x	x					x					
REQ-6	1						x							
REQ-7	3								x			x		
REQ-8	5								x			x		
REQ-9	5		x											
REQ-10	2					x								
REQ-11	4								x					
REQ-12	2	x	x											
REQ-13	5				x									
REQ-14	2													x
REQ-15	4							x						
REQ-16	3			x										
REQ-17	2													x
REQ-18	1	x	x											x
REQ-19	3													x
REQ-20	4	x	x											x
REQ-21	5													x
REQ-22	5													x
REQ-23	5					x	x							

REQ-24	4													
REQ-25	5									x	x			
REQ-26	5													
REQ-27	5				x									
REQ-28	4	x						x				x		
MREQ-1	5					x								
MREQ-2	4				x									
MREQ-3	3													
MREQ-4	2			x										
MREQ-5	3						x							
MREQ-6	5		x											
MREQ-7	4							x						
MREQ-8	2												x	
MREQ-9	5									x				
MREQ-10	2													x
HREQ-1	5	x	x											
HREQ-2	3			x										
HREQ-3	4					x								
HREQ-4	2				x									
HREQ-5	4						x							
CREQ-1	5											x		
CREQ-2	5	x	x											
CREQ-3	4	x			x									
CREQ-4	4			x										
CREQ-5	4		x											

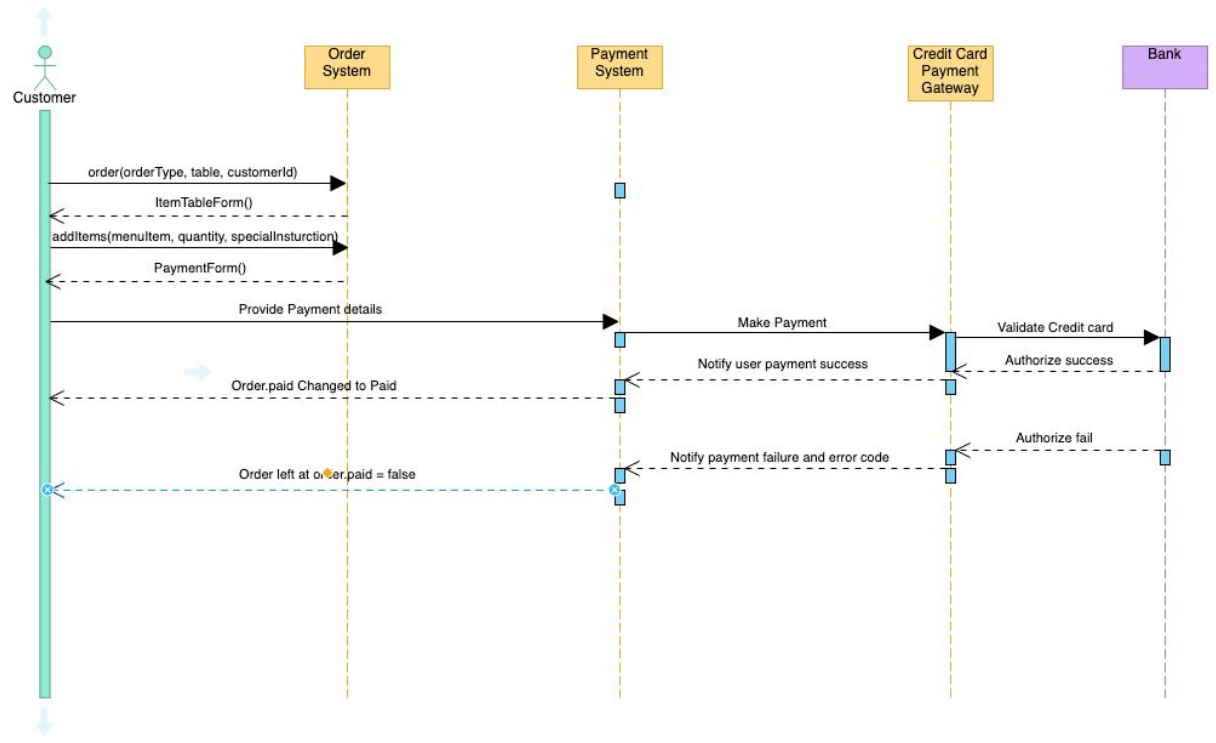
SREQ-1	5	x	x											
SREQ-2	2							x						
SREQ-3	4			x										
SREQ-4	3								x					
SREQ-5	3							x						
SREQ-6	5							x						
SREQ-7	5			x										
SREQ-8	5	x	x											
SREQ-9	5		x											
SREQ-10	4	x			x									
BREQ-1	5			x					x					
BREQ-2	5			x					x					
BREQ-3	3							x						
BREQ-4	3			x										
LREQ-1	5							x						
LREQ-2	4						x							
LREQ-3	4	x	x											
CUREQ-1	3	x	x											
CUREQ-2	5					x								
CUREQ-3	3						x							
CUREQ-4	5							x						

4.3.4 Fully Dressed Description

UC-1: Order Tracking
Related Requirements: REQ-1, REQ-4, CREQ-1, CREQ-3, LREQ-2, LREQ-4, LREQ-5, SREQ-9, SREQ-8
Initiating Actor(s): Server, Liaison
Actor's Goal: To keep track of a customer's order.
Participating Actors: Database
Preconditions: The user has placed their order through the server or through the online system.
Postconditions: The Server/Liaison are notified when a customer's order is ready to be picked up or served.
Flow of Events for Main Success Scenario: 1. → The employee signs on to the QuickBytes application using their credentials. 2. ← The system will display views and data appropriate to the user's permission levels (i.e., server or liaison). 3. → The employee pulls up information related to the order they want to track. 4. → The employee marks the order as delivered. 5. ← The user logs out of the application and their session ends.
Flow of Events for Alternate Success Scenario: 1. → The employee signs on to the QuickBytes application using their credentials. 2. ← The system will display views and data appropriate to the user's permission levels (i.e., server or liaison) 3. → The employee is unable to find the order information they are seeking and waits for more updated order information. 4. → The employee signs out of the application 5. ← The user's session ends.

UC-3: Table Management
Related Requirements: REQ-5, BREQ-1, BREQ-2, BREQ-3, HREQ-5, SREQ-3, SREQ-5, SREQ-7
Initiating Actor(s): Server, Busser
Actor's Goal: Mark tables as available, reserved, dirty, etc.
Participating Actors: Database
Preconditions: The restaurant has tables that can be used for seating customers.
Postconditions: The actor(s) are able to update the status of the table.
Flow of Events for Main Success Scenario: 1. → The server signs on to the QuickBytes application using their credentials. 2. ← The system will display the tables that are currently occupied and in the server's section. 3. → The server releases a table for cleaning. 4. ← The table is sent to the busser's queue for cleaning.
Flow of Events for Main Success Scenario: 1. → The busser signs on to the QuickBytes application using their credentials. 2. ← The system will display the tables that are currently occupied and in the busser's section. 3. → The busser marks tables as "clean" one-at-a-time, once they finish clearing them. 4. ← The table is sent to the host's lists of available tables for seating the next party of tables.

4.4 System Sequence Diagrams


Customer
Order System
Payment System
Credit Card Payment Gateway
Bank
order(orderType, table, customerId)
ItemTableForm()
addItems(menuItem, quantity, specialInsturction)
PaymentForm()
Provide Payment details
Make Payment
Validate Credit card
Authorize success
Notify user payment success
Order.paid Changed to Paid
Authorize fail
Notify payment failure and error code
Order left at order.paid = false

5. Effort Estimation Using Case Points

5.1 Unadjusted Actor Weight

Actor Type	**Description of how to recognize the actor type**	**Weight**
Simple	Actor interacts with system through API	1
Average	Actor interacts with system via text-based input	2
Complex	Actor interacts with system through GUI	3

Actor Name	**Description of Relevant Characteristics**	**Complexity**	**Weight**
Delivery Liaison	The liaison is responsible for taking customer orders over the phone and prioritizing the order in which they are sent to the Chef according to delivery and pickup time.	Complex	3
Chef	The chef is responsible for preparing the orders in the kitchen. The chef will receive orders from the server and liaison and notify them when their orders have been completed or request additional information.	Complex	3
Host(ess)	The host(ess) is responsible for initially greeting the customers and either assigning seats or fulfilling reservations. The host can see when busboys have indicated a table as available.	Complex	3

Customer	Customers use the service of the Restaurant. They may order dine-in, or carryout. They will view the menu, order food, eat, and pay for their meal.	Complex	3
Server	Servers must be a central part of the system. They may create, view, edit, delete, and return orders. They will send and receive alerts from the host, chef, busser, and customer.	Complex	3
Busser	Bussers clear tables in the order in which they become vacant and mark the tables as clean in a queue so the host can seat new guests.	Complex	3

UAW(QuickBytes) = 6 * Complex = 6 * 3 = 18

5.2 Unadjusted 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	Order placement	Complex	15
UC-2	Status updates	Simple	5
UC-3	Table management	Average	10
UC-4	Payment	Complex	15
UC-5	Login	Simple	5
UC-6	Low contact mode	simple	5
UC-6	Low contact mode	simple	5
UC-7	Request Service	simple	5
UC-8	Waiting List	simple	5
UC-9	Employee Schedule	simple	5
UC-10	Inventory Management	complex	15
UC-11	Table Reservation	Average	10
UC-12	Timecard	Simple	5
UC-13	Account creation	Complex	15
Total			115

UUCP(QuickBytes) = 12 + 115 = 127

5.3 Technical Complexity Factors

Technical Factor	Description	Weight
T1	Distributed System	2
T2	Response and Throughput Critical	1
T3	End user efficiency	1
T4	Complex internal processing	1
T5	Reusable design or code	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to Change	1
T10	Concurrent use	1
T11	Special Security Features	1
T12	Third parties	1
T13	Special training required	1

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor (Weight * Perceived Complexity)
T1	Distributed, Web based	2	3	2x3=6
T2	User expects good performance	1	3	1x3=3
T3	User expects efficiency	1	3	1x3=3
T4	Internal processing is simple	1	1	1x1=1
T5	Reusability simplifies implementation and maintenance	1	2	1x2=2
T6	Installation required by Engineer	0.5	3	0.5x3=1.5
T7	Ease of use extremely important	0.5	5	0.5x5=2.5
T8	No portability concerns	2	0	2x0=0
T9	Easy to change(Admin interface)	1	1	1x1=1
T10	Concurrent use is required	1	4	1x4=4
T11	Security is a concern	1	5	1x5=5
T12	No third party access	1	0	1x0=0
T13	No unique training	1	0	1x0=0
Total				29

TCF = Constant-1 + Constant-2 x Technical Factor Total = 0.6+(0.01x29)=0.89

5.4 Environmental Complexity Factors

Environmental Factor	Description	Weight
E1	Familiar with development process	1.5
E2	Application problem experience	0.5
E3	Paradigm experience	1
E4	Lead Analyst Capability	0.5
E5	Motivation	1
E6	Stable Requirements	2
E7	Part-time Staff	-1
E8	Difficult Programming Language	-1

Environmental Factor	Description	Weight	Perceived Complexity	Calculated Factor (Weight * Perceived Complexity)
E1	Familiar with development process	1.5	1	1x1.5=1.5
E2	Application problem experience	0.5	2	0.5x2=1
E3	Paradigm experience	1	3	1x3=3
E4	Lead Analyst Capability	0.5	3	0.5x3=1.5
E5	Motivation	1	5	1x5=5

E6	Stable Requirements	2	5	2x5=10
E7	Part-time Staff	-1	5	-1x5=-5
E8	Difficult Programming Language	-1	1	-1x1=-1
Total				16

ECF = Constant-1 Constant-2 Environmental Factor Total

ECF = 1.4+(-0.03x16) =.92

5.5 Use Case Points

UCP = UUCP x TCF x ECF

UCP = 127x0.89x.92= ~104 x 28PF = 2,912

6. Domain Analysis

6.1 Domain Model

6.1.1 Concept Definitions

Responsibility	Type	Concept
R1: Coordinate activity between the customer, chef, server, busser, etc.	D	Controller
R2: Display a list of tables and their status, such as occupied, dirty, etc.	D	Interface
R3: Prevent invalid table selections.	D	Table Status
R3: Store the customer order in the database	K	Profile

R4: Store the customer's login details in the database.	K	Profile
R5: Queue incoming orders for the chefs to prepare	D	Order Status
R6: Display a list of current orders that need to be served, in the order they were ready.	D	Order Status
R7: Handle payment processing.	D	Payment System
R8: Display a queue for bussers to see tables that need to be cleaned.	D	Table Status
R9: Enable host(ess) to assign available tables to servers and seat customers at those tables once they become ready.	D	Floor Plan
R10: Enable manager to send supply refills to active bussers.	K	Floor Plan

6.1.2 Association Definitions

Concept Pair	Association Description	Association Name
Customer Profile ⇔ DB Connection	Retrieve the customer's information from the database.	QueryDB
Interface ⇔ Controller	Enable the user to interact with the application based on their administrative position levels.	User Action
Controller ⇔ DB Connection	Modify or insert data into the database.	UpdateDB
Controller ⇔ Food Status	Add an order to the database and update the order queue.	Update Food Status View Food Status

Controller ⇔ Payment	Allow a user to complete a payment transaction.	Make Payment
Table Status ⇔ Interface	Provide a user interface for the active tables and their current status.	Display
Controller ⇔ Order Status	Send orders to the chef for preparation.	Update Order Status
Interface ⇔ Controller	Enable users call their server when they need something.	User Action

6.1.3 Attribute Definitions

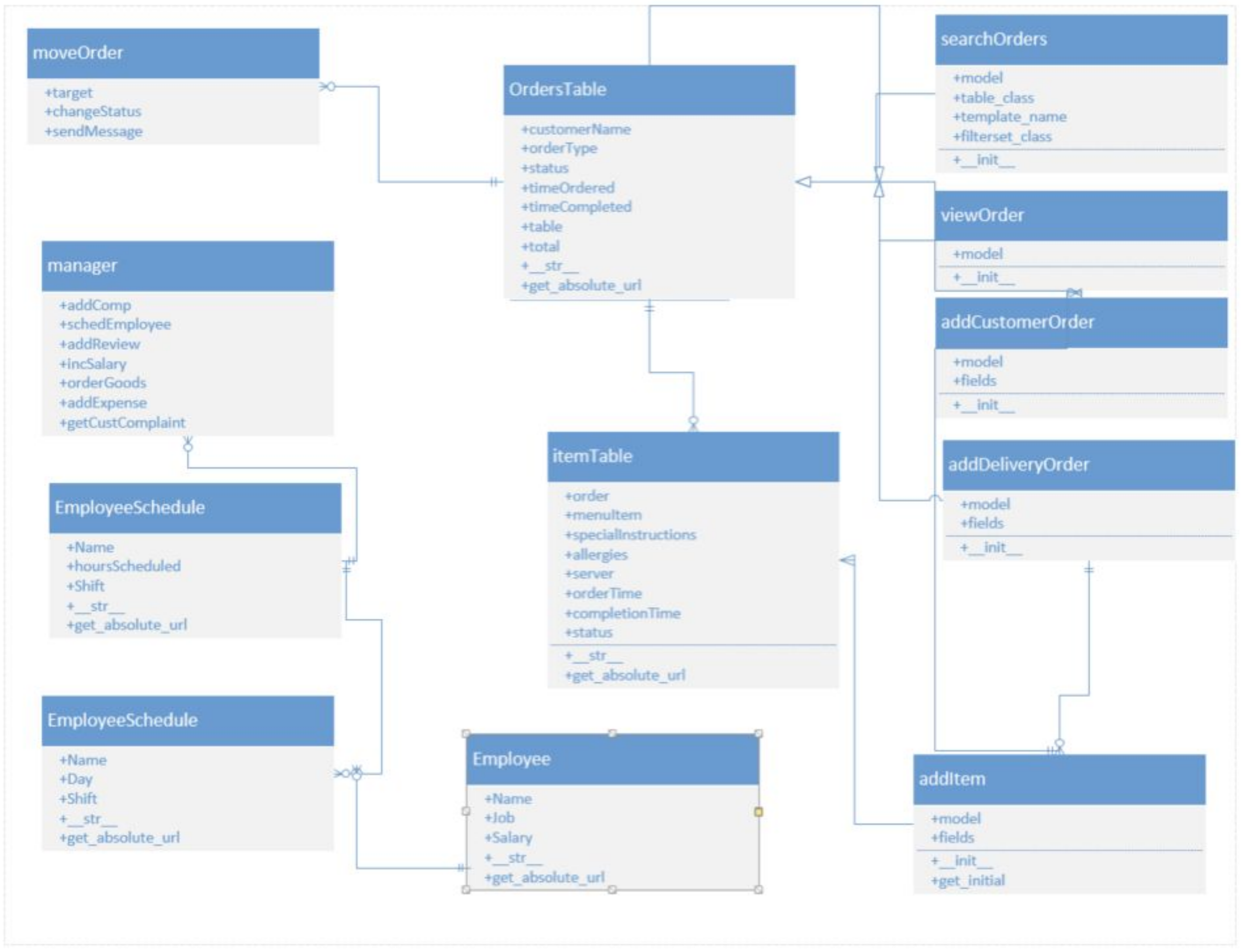
Concept	**Attribute**	**Description**
Controller	Order	All orders that are made on behalf of the customer and stored in the database.
Interface	viewData	Allows users to view and interact with their recent order history.
Table Status	viewTables updateTables	View the current tables and their dining room setup. Update a table’s status.
Profile	userName userPassword	The username associated with an account. The password associated with an account.
Order Status	viewOrder updateOrder	View the current orders in-queue. Update an order’s status.
Payment	paymentMade	Updates the system based on the payment status.

System		

6.1.4 Traceability Matrix

Use Case	PW	Controller	Interface	Table Status	Profile	Order Status	Payment System
UC-1	4	x			x		
UC-2	2	x		x			x
UC-3	3	x					
UC-4	4		x			x	
UC-5	4			x	x		
UC-6	1		x				
UC-7	3	x		x		x	
UC-8	5		x		x		
UC-9	5	x		x		x	
UC-10	2		x	x			
UC-11	4			x		x	
UC-12	2		x				x
UC-13	5	x					x

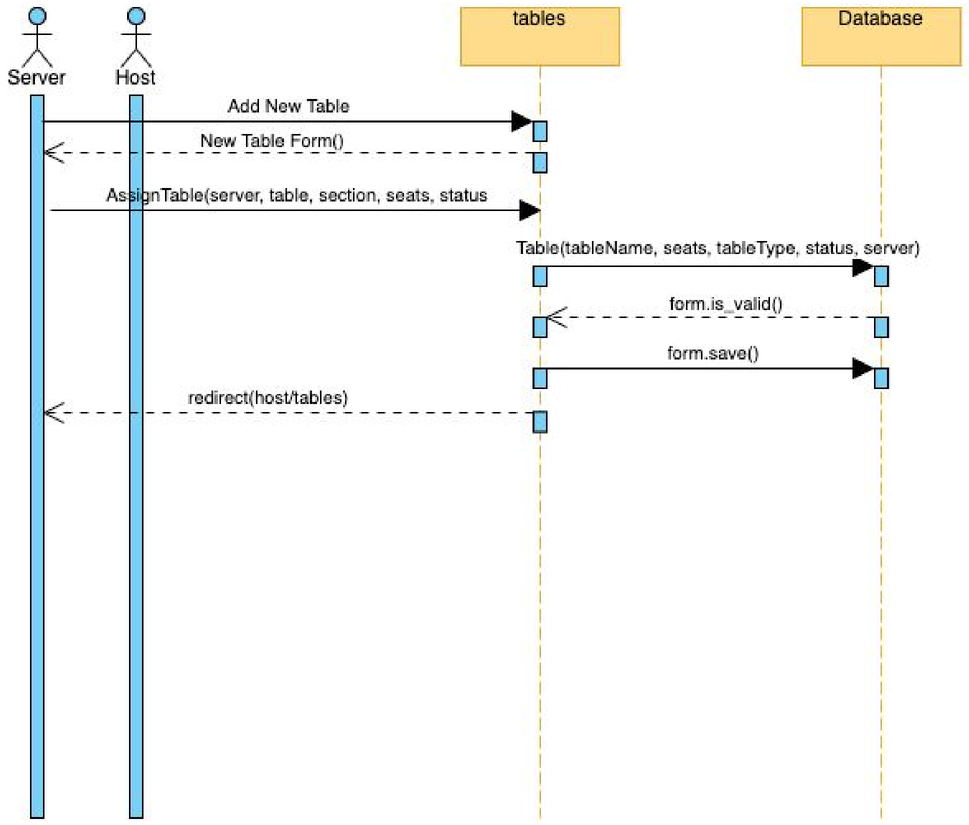
6.1.5 Domain Model Diagram



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

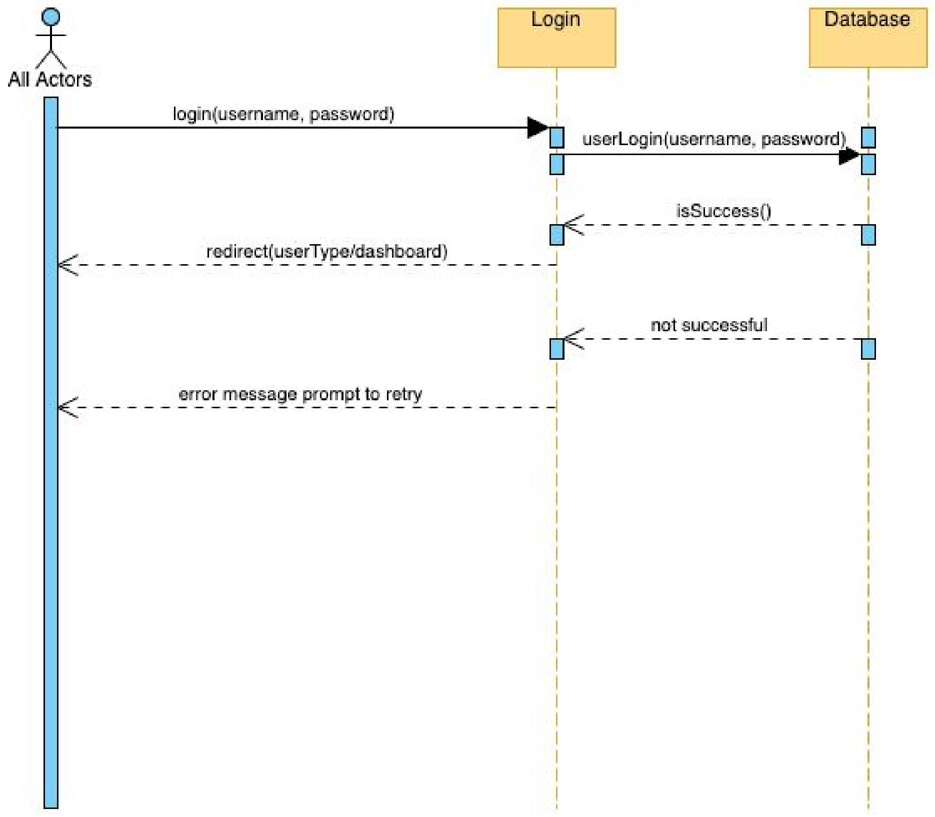
7. Interaction Diagrams

UC-3: Table Management



The above sequence diagrams display the various flows that exist for table management. Tables are primarily managed by servers and hosts and can be assigned, marked as clean or occupied and this information is persisted via the database.

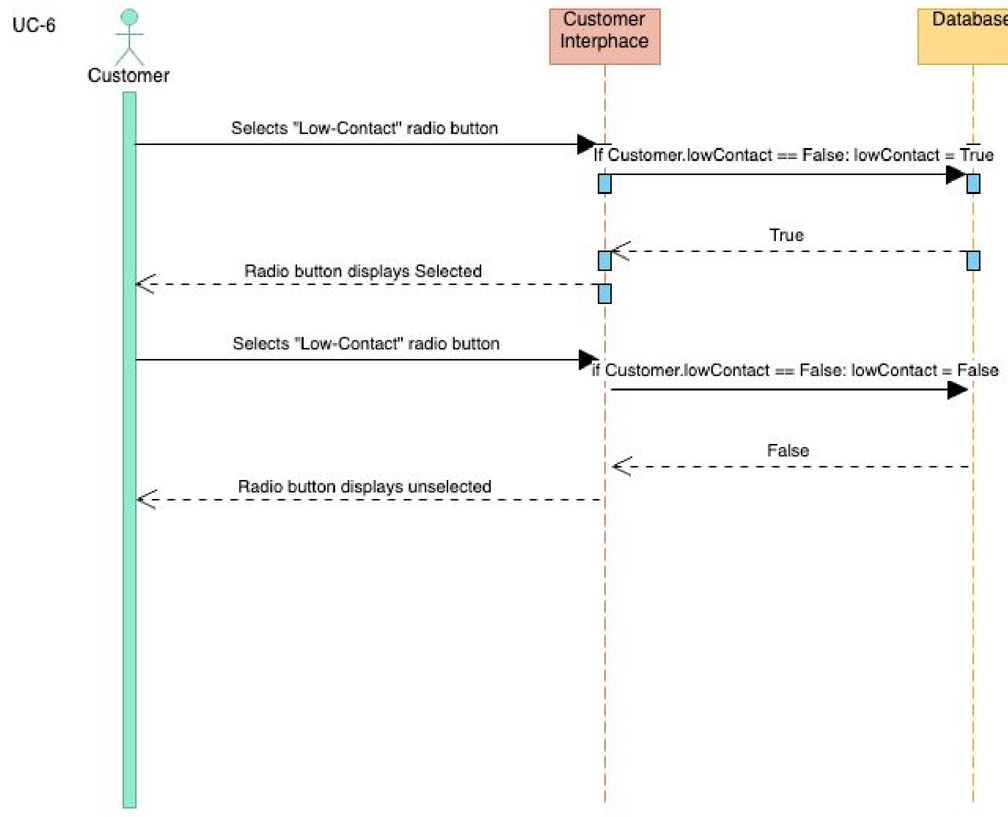
UC-5: Login



All Actors
Login
Database
login(username, password)
userLogin(username, password)
isSuccess()
redirect(userType/dashboard)
not successful
error message prompt to retry

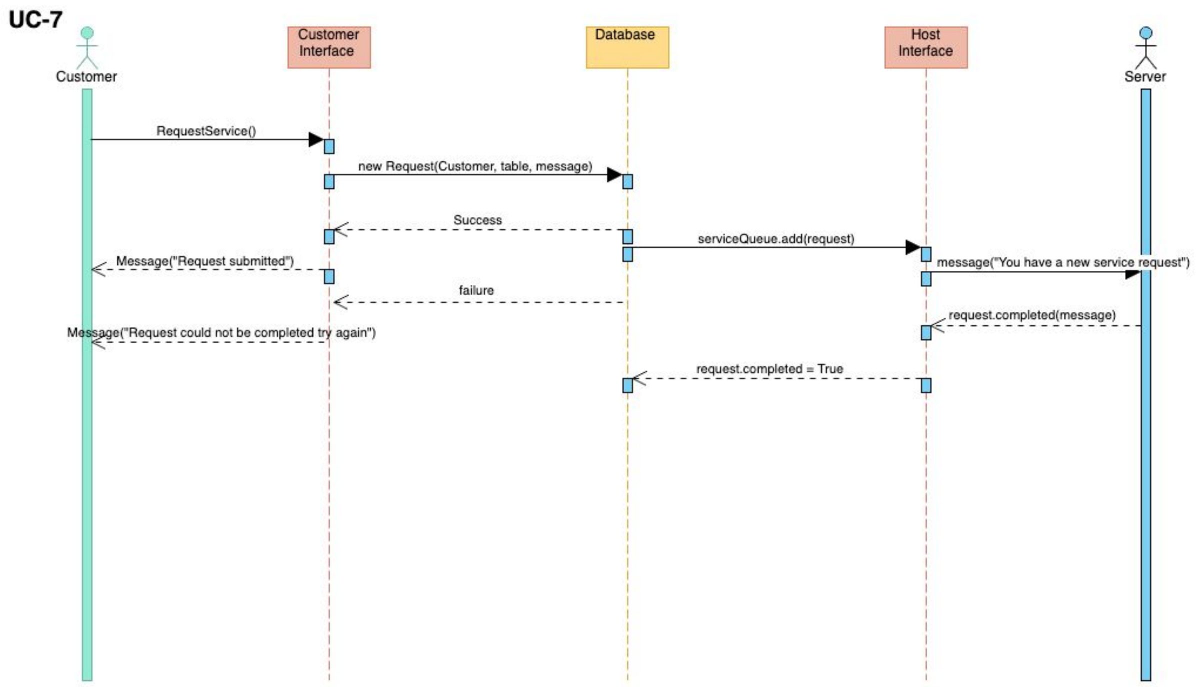
The above sequence diagrams display the various flows that exist for a user to log in to the QuickBytes application. Based on the user's administrative role, such as employee, customer, etc., they will be redirected to the appropriate views once they have been authenticated.

UC-6: “Low-Contact” Mode


UC-6
Customer
Customer Interphace
Database
Selects "Low-Contact" radio button
If Customer.lowContact == False: lowContact = True
True
Radio button displays Selected
Selects "Low-Contact" radio button
if Customer.lowContact == False: lowContact = False
False
Radio button displays unselected

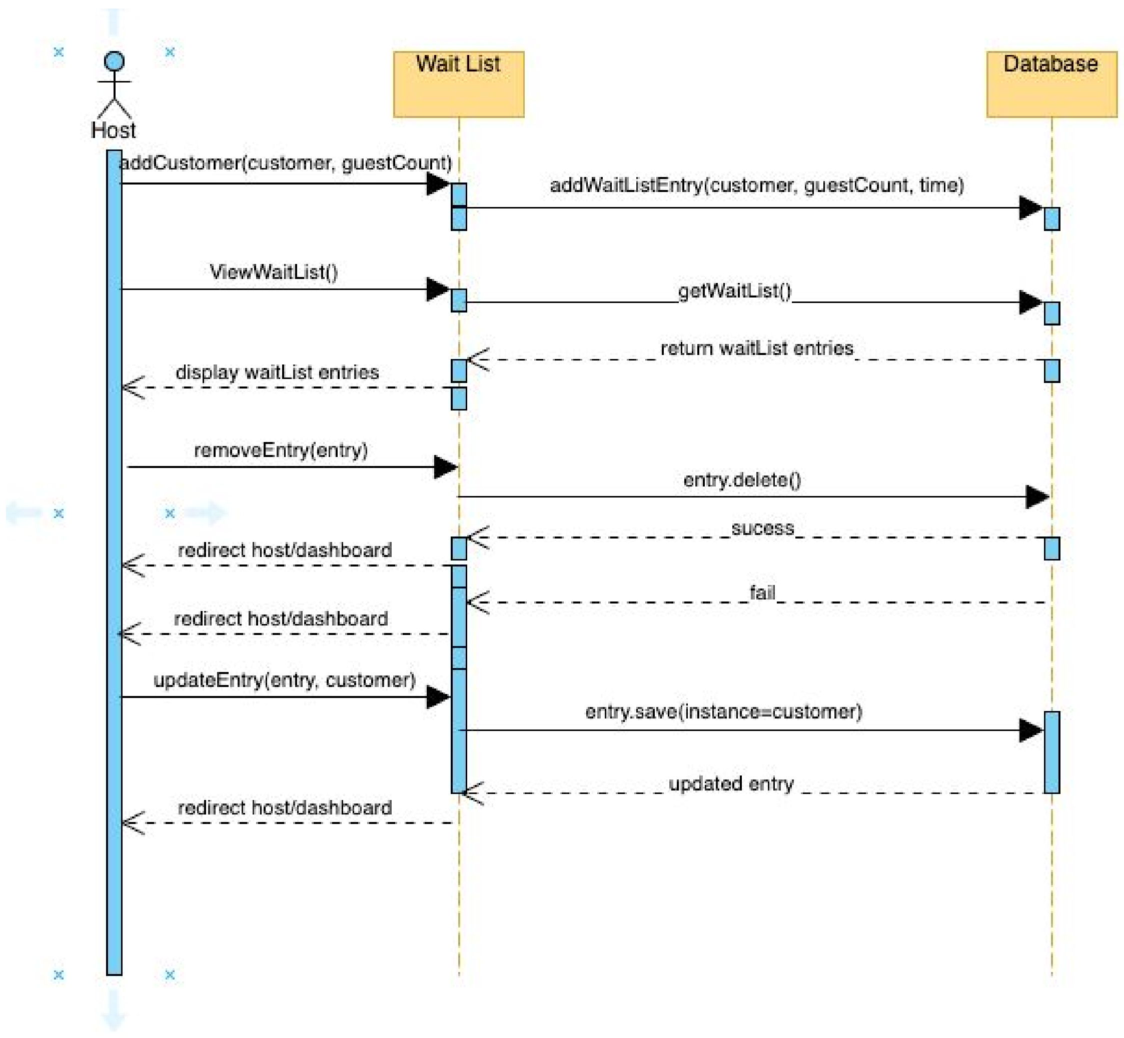
The above sequence diagrams display the various flows that exist for a customer to request “low-contact” mode when placing an order. There is a user interface that exists, with a radio button that the user can select to request “low-contact” mode for their order and restaurant experience.

UC-7: Request Service


UC-7
Customer
Customer Interface
Database
Host Interface
Server
RequestService()
new Request(Customer, table, message)
Success
serviceQueue.add(request)
Message("Request submitted")
message("You have a new service request")
failure
request.completed(message)
Message("Request could not be completed try again")
request.completed = True

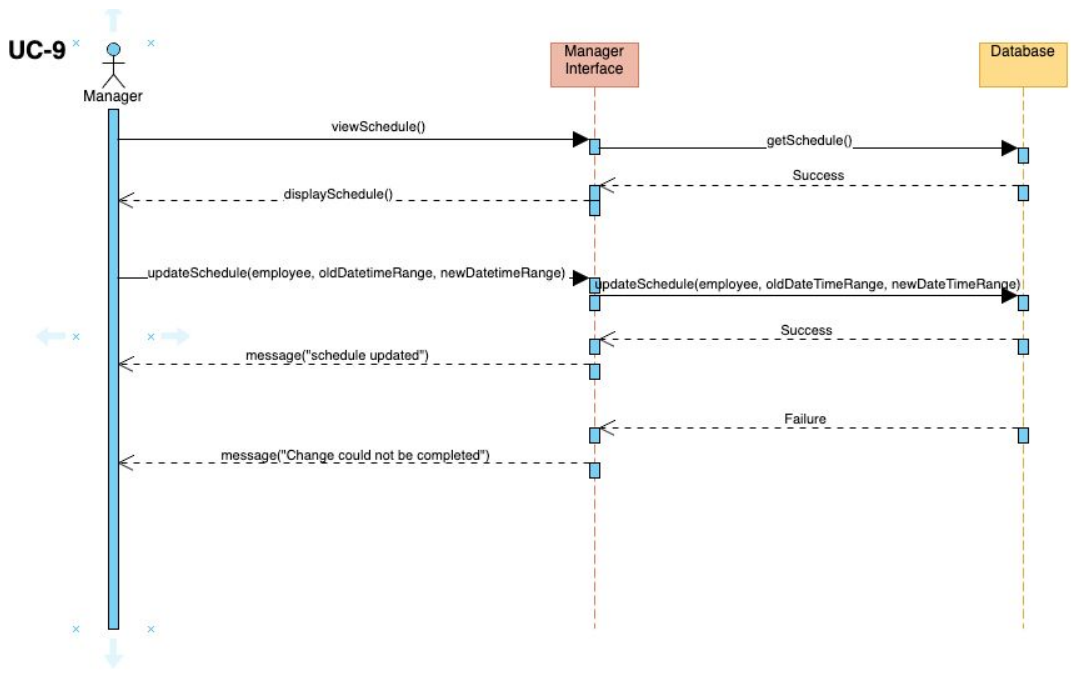
The above sequence diagrams display the various flows that exist for a customer to be able to request service from their waiter or waitress while dining in at the restaurant. When a customer submits a request for service, it is added to the server's queue to check on customers who have requested the service.

UC-8: Waiting List Management



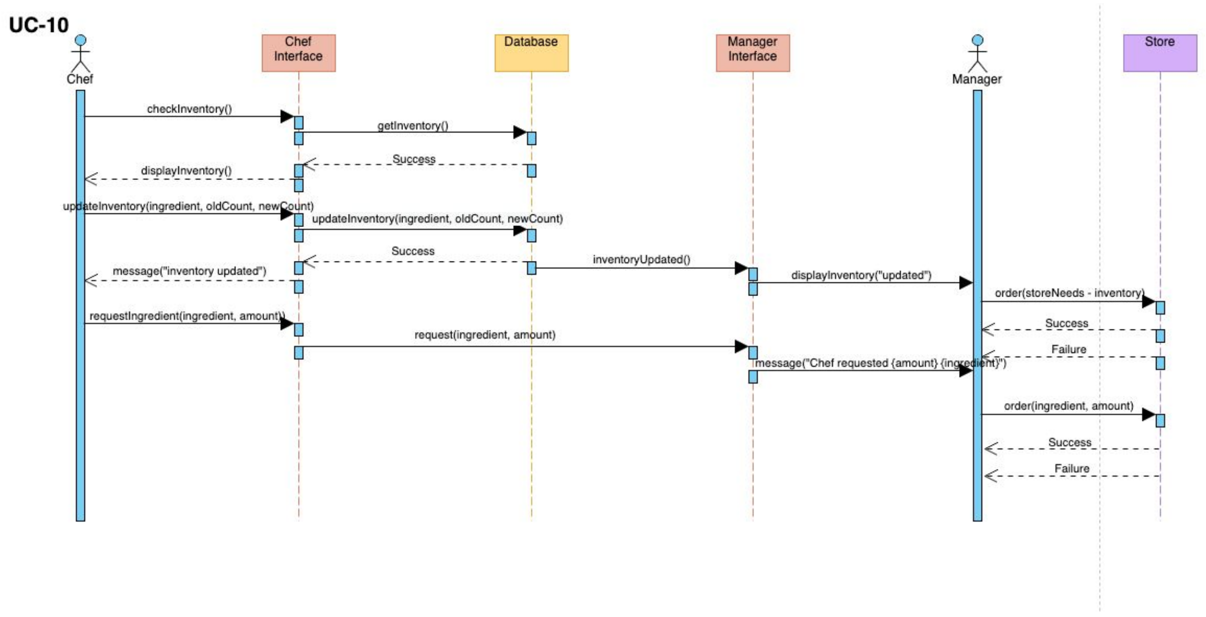
The above sequence diagrams display the various flows that exist for managing customer waitlists. The waitlist information is displayed to the host(ess), who uses the data to seat customers and provide wait times accurately.

UC-9: Employee Scheduling


UC-9
Manager
Manager Interface
Database
viewSchedule()
getSchedule()
Success
displaySchedule()
updateSchedule(employee, oldDatetimeRange, newDatetimeRange)
updateSchedule(employee, oldDateTimeRange, newDateTimeRange)
Success
message("schedule updated")
Failure
message("Change could not be completed")

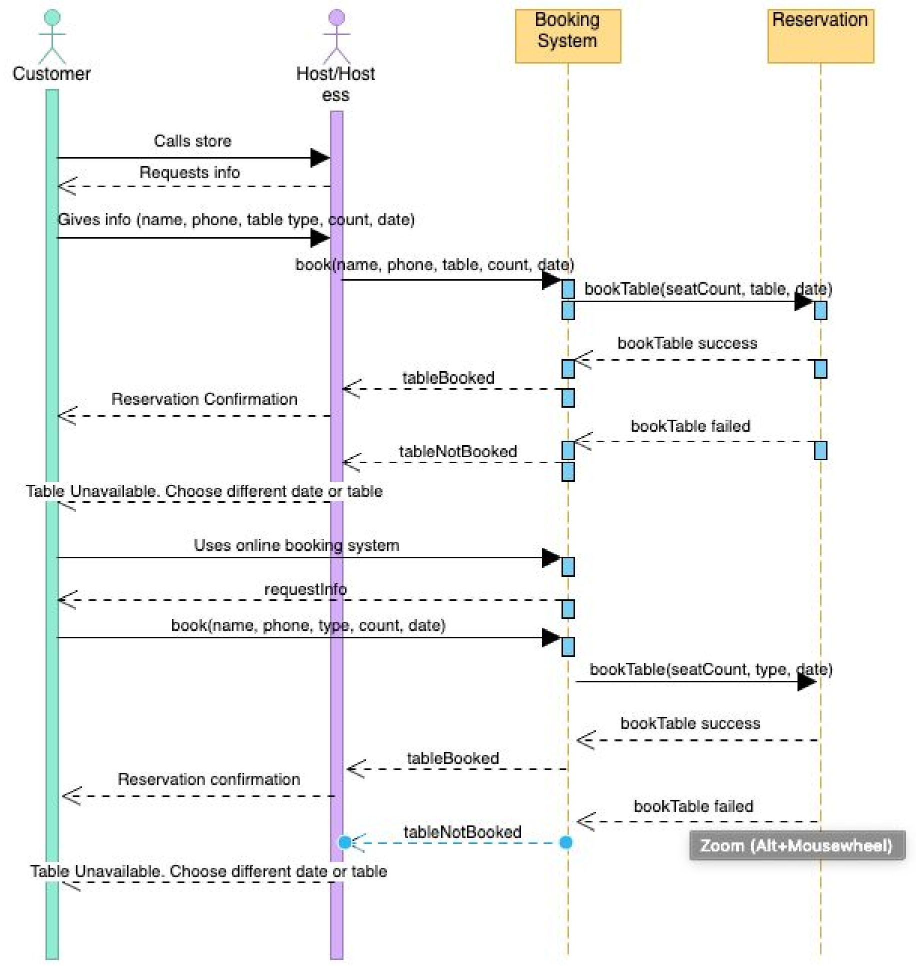
The above sequence diagrams display the various flows that exist for a manager to view their employees' schedules and create new schedules based on the restaurant needs.

UC-10: Ingredient Management


UC-10
Chef
Chef Interface
Database
Manager Interface
Manager
Store
checkInventory()
getInventory()
Success
displayInventory()
updateInventory(ingredient, oldCount, newCount)
updateInventory(ingredient, oldCount, newCount)
Success
inventoryUpdated()
message("inventory updated")
displayInventory("updated")
order(storeNeeds - inventory)
requestIngredient(ingredient, amount)
Success
request(ingredient, amount)
Failure
message("Chef requested {amount} {ingredient}")
order(ingredient, amount)
Success
Failure

The above sequence diagrams display the various flows that exist for a chef to manage the ingredients and inventory they need in order to accurately and efficiently prepare customer orders. The Chef is presented with a user interface and a form to submit ingredient inventory updates and place orders for additional supplies.

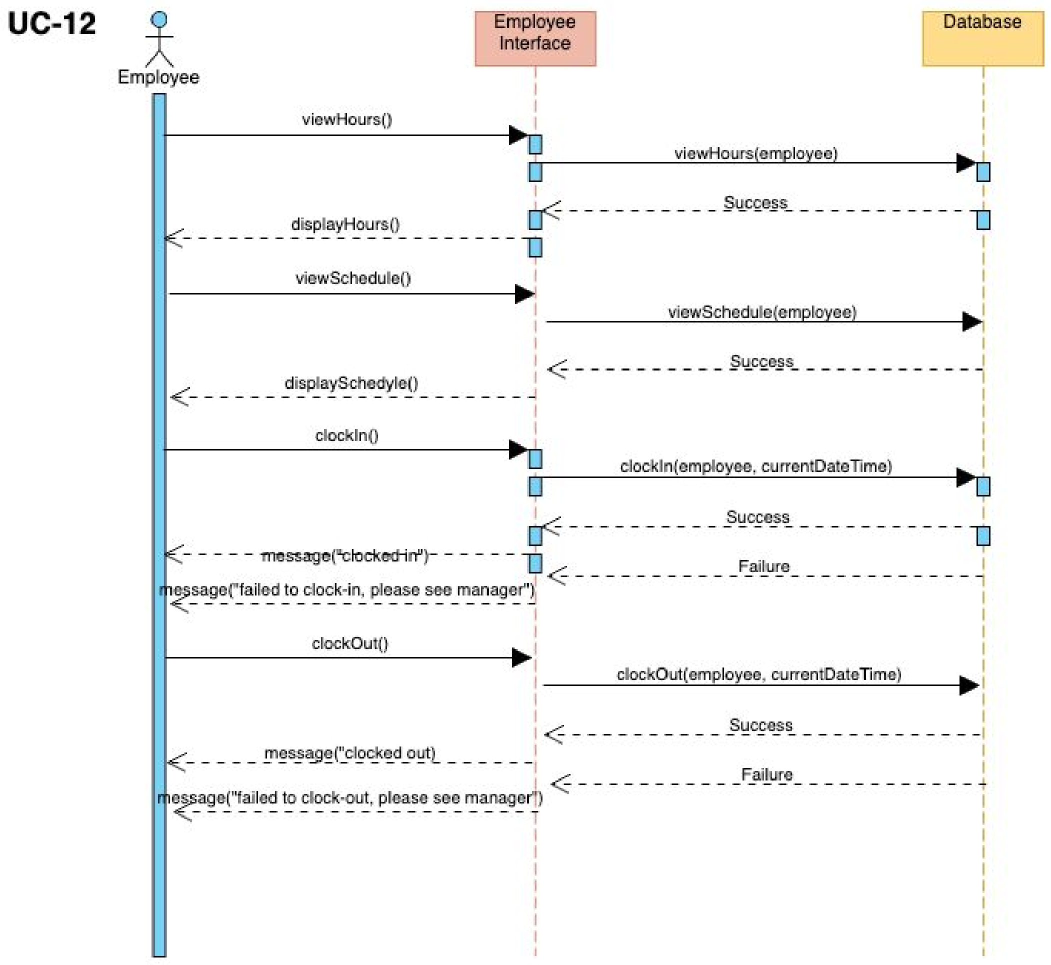
UC-11: Table Reservation



Customer
Host/Hostess
Booking System
Reservation
Calls store
Requests info
Gives info (name, phone, table type, count, date)
book(name, phone, table, count, date)
bookTable(seatCount, table, date)
bookTable success
tableBooked
Reservation Confirmation
bookTable failed
tableNotBooked
Table Unavailable. Choose different date or table
Uses online booking system
requestInfo
book(name, phone, type, count, date)
bookTable(seatCount, type, date)
bookTable success
tableBooked
Reservation confirmation
bookTable failed
tableNotBooked
Table Unavailable. Choose different date or table
Zoom (Alt+Mousewheel)

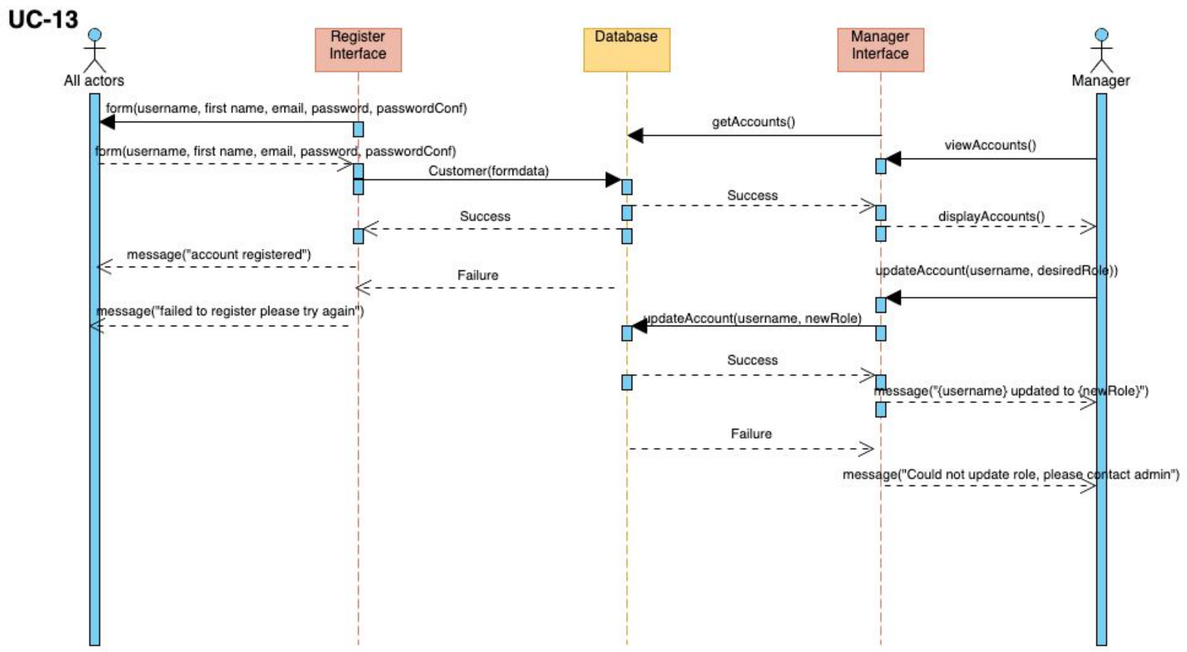
The above sequence diagrams display the various flows that exist for table reservations. Tables are reserved by customers in advance, based on the date and time that a customer for which a customer would like to reserve the table. If the table is not occupied or has not already been reserved, the customer is able to reserve it.

UC-12: Employee Timecard



The above sequence diagrams display the various flows that exist for an employee to be able to clock in and out of the time tracking system. A user can also track the hours they have worked in a given time period.

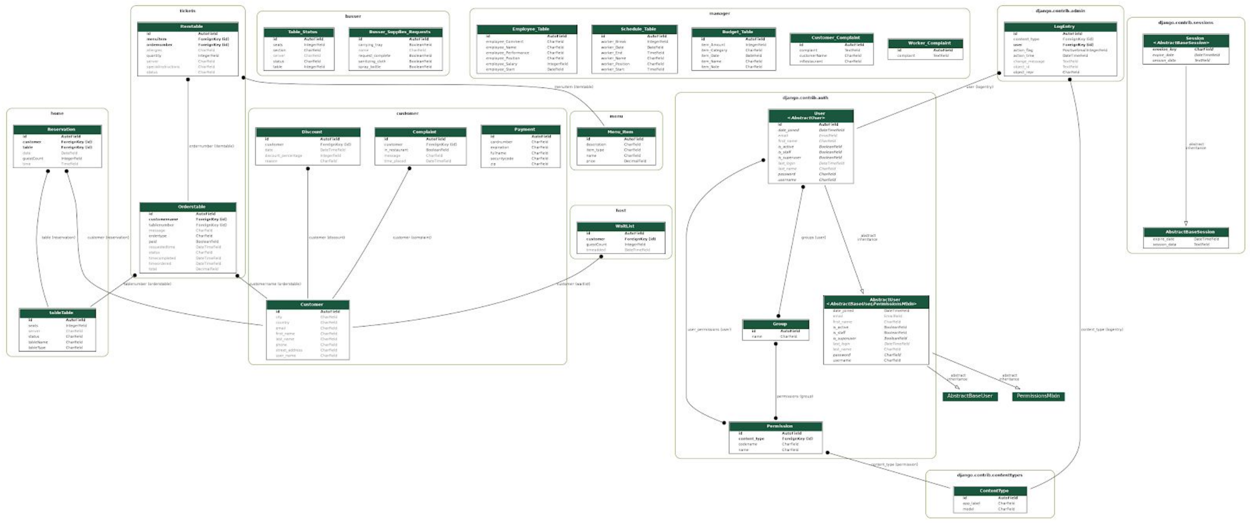
UC-13: Create Account


UC-13
All actors
Register Interface
Database
Manager Interface
Manager
form(username, first name, email, password, passwordConf)
form(username, first name, email, password, passwordConf)
Customer(formdata)
Success
message("account registered")
Failure
message("failed to register please try again")
getAccounts()
viewAccounts()
Success
displayAccounts()
updateAccount(username, desiredRole))
updateAccount(username, newRole)
Success
message("{username} updated to {newRole}")
Failure
message("Could not update role, please contact admin")

The above sequence diagrams display the various flows that exist for creating a new account using the QuickBytes software. When a user registers for an account, they must create an account with a unique username. If none exists, they are able to create an account appropriate to their role, otherwise they must continue until they have selected a unique username based on their account type.

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 Orderstable:

1. Attributes:
    - customerName - Name of customer
    - orderType - Type may be delivery, dine-in, or carry-out
    - status - The order status waiting, preparing, etc
    - itemsOrdered - A list of items ordered
    - timeOrdered - The time at which the order was started
    - timeCompleted - Completion time of order
    - table - The table number if the customer is dining in
    - total - The total cost of an order
2. Methods:
    - __str__ - string representation of the object
    - Get_absolute_url - provides a redirection upon successful completion of an edit function

Class itemTable:

1. Attributes:
    - Order - the associated order number for this item
    - menuItem - the associated menuItem for this item
    - specialInstructions - user supplied instructions for prepping this item
    - Allergies - user supplied allergies if they exist
    - Server - associated server if one exists
    - orderTime - time the item was added to an order
    - completionTime - time the item was completed in the kitchen
    - Status - the status of an item waiting, preparing, completed, delivered
2. Methods:
    - __str__ - the string representation of the object

Class addCustomerOrder:

1. Attributes:
    - Model - the class item from which this class inherits its attributes
    - Fields - the pertinent fields to include for this order
2. Methods:
    - __init__ - constructor for the class

Class addDeliveryOrder:

3. Attributes:
    - Model - the class item from which this class inherits its attributes
    - Fields - the pertinent fields to include for this order
4. Methods:
    - __init__ - constructor for the class

Class viewOrder:

5. Attributes:
    - Model - the class item from which this class inherits its attributes
6. Methods:
    - __init__ - constructor for the class

Class searchOrders:

7. Attributes:
    - Model - the class item from which this class inherits its attributes
    - Table_class - the associated table for this set of orders
    - Template_name - the html template to include with this view
    - Filterset_class - the filters to include with the searchOrder class

8. Methods:
    - __init__ - constructor for the class

Class addItem:

9. Attributes:
    - Model - the class item from which this class inherits its attributes
    - Fields - the pertinent fields to include for this order
10. Methods:
    - __init__ - constructor for the class
    - Get_initial - provides initial values for order to provide association

Class moveOrder:

11. Attributes:
    - Target - what status this order will have
12. Methods:
    - changeStatus - changes status of order
    - sendMessage - sends a message to the appropriate person in the restaurant if necessary

Class moveOrder:

13. Methods:
    - addComp - will discount some or all of an order
    - schedEmployee - will schedule an employee for a shift
    - addReview - provide the employee with the periodic review
    - incSalary - will allow a manager to give an employee a raise
    - orderGoods - will allow the manager to order more food or beverage items
    - addExpense - will allow the manager to add expenses for tracking
    - getCustomerComplaint - allows a manager to receive customer complaints

Class employeeSchedule:

14. Attributes:
    - Name - employee name
    - hoursScheduled - total number of hours scheduled for the week
    - Shift - shifts that an employee is scheduled for
15. Methods:
    - __str__ - string representation of employee schedule
    - Get_absolute_url - allows for redirection upon successful completion of editing a schedule

Class Employee:

16. Attributes:
    - Name - employee name
    - Job - the associated job title
    - Salary - the salary of the employee
17. Methods:
    - __str__ - string representation of employee schedule
    - Get_absolute_url - allows for redirection upon successful completion of editing a schedule

Class BusserSuppliesRequests:

3. Attributes:
    - name - Name of busser
    - sprayBottle - A boolean value to be specified if needed
    - sanitizingCloth - A boolean value to be specified if needed
    - carryingTray - A boolean value to be specified if needed
    - requestComplete - A boolean value to be specified if needed
4. Methods:
    - __str__ - string representation of the object
    - Get_absolute_url - provides a redirection upon successful completion of an edit function

Class tableStatus:

5. Attributes:
    - server - Name of server
    - section - Table section identifier
    - table - Table identifier
    - seats - The number of seats a table has
    - status - The table's status
6. Methods:
    - __str__ - string representation of the object
    - Get_absolute_url - provides a redirection upon successful completion of an edit function

8.3 Traceability Matrix

	Software Classes						
Domain Concepts	**Order**	**Customer**	**Delivery**	**Schedule**	**Tables**	**Status**	**Item**
Controller		x			x		
Interface	x	x		x			x
Table Status		x					
Profile	x		x			x	
Order Status				x	x		
Payment System			x				
Table Layout		x		x		x	
Interface	x		x		x		
Table Layout		x		x		x	
Supplies			x	x			
Login			x				x
Register		x					x

- Controller
  - Menu: Controller forces items to be chosen from the built-in menu.
  - TableLayout: Controller insists that a valid table is selected.
  - PaymentMethod: Controller requires a valid Payment Method from the customer.
- Interface
  - Menu: The restaurant's menu is viewable through the user interface.
  - Order: Orders can be viewed and edited through the user interface.
  - Table: Tables can be viewed through the user interface.

- Table Status
    - Busser: A busser can update a table’s status.
    - Host(ess): A host(ess) can update the table’s status and assign it to servers.
    - Server: A server can release a table to a Busser for cleaning.
- Profile
    - User: Allows customers to place an order for pick-up.
- Order Status
    - Chef: A chef can update an order’s status.
    - Server: A server can update an order’s status.
    - Manager: A server can update an order’s status.
- Payment System
    - User: A user can submit payment for an order.
- Table Layout
    - Floor Plan: A floor plan with the table layout is displayed via the user interface.
- Supplies
    - Busser: A busser can request supplies from the manager via a queue.
    - Manager: A manager can send a busser additional supplies while on the floor.
- Login
    - Customer: A customer can log in to their account to view recent or past orders.
    - Server: A server can log in to their account to see their tables.
    - Busser: A busser can log in to their account to see their queue.
    - Chef: A chef can log in to their account to see their orders.
    - Manager: A manager can log in to their account to see a list of employees and statistics about their restaurant.
- Register
    - User: A user can register for an account based on their administrative permission levels.

8.4 Design Patterns

Because we were using the Django framework to build out the requirements for QuickBytes, one of the primary design patterns that we used was the MVC pattern, or Model-View-Controller pattern. Each Django template is stored as a view, and each class is represented by a model. The controllers are also class-based and handle business logic, as well as HTTP requests to render views dynamically using the data retrieved from the models.

One of the primary advantages of the MVC pattern is that it allows for a separation of concerns and keeps the different parts of an application separated in a clean and cohesive manner. Some would also argue that the way in which the views and models interact is almost like a publish-subscribe pattern, in which the view is the subscriber, which subscribes to updates from the model. When the model publishes (updates), it sends those publications to the view layer of a Django application.

8.5 Object Constraint Language (OCL)

MainScreen:
Context MainScreen::clickLogin
Invariant: primaryKey, id
Pre-conditional: findViewById(R.id)
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: primaryKey, 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()
Post-conditional: return isValid

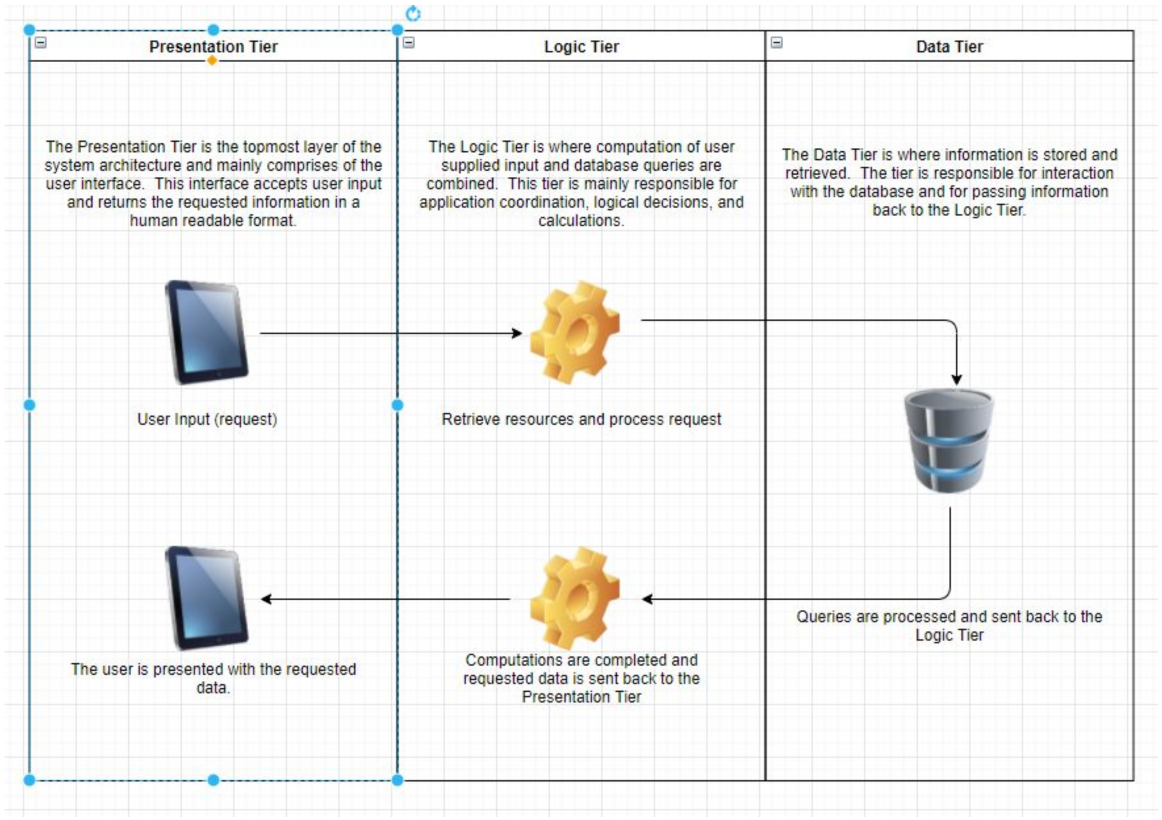
9. System Architecture and System Design

9.1 Architectural Styles

We are using the Layered Pattern in our application architecture, which is often used for web applications. The three layers consist of a presentation tier, a logic tier, and a data tier. The presentation tier is observed by the users on the front-end, which are both customers and restaurant staff. Logic and data tiers are on the back-end and keep track of user-entered information, as well as respond to commands from the presentation tier.

Our application also uses Client-Server architecture, which is a one-to-many relationship. The application receives data from the client and stores this data in a database. The QuickBytes software is used by a variety of different clients, such as web browsers, POS systems and graphical seating charts.

9.2 Identifying Subsystems


Presentation Tier
Logic Tier
Data Tier
The Presentation Tier is the topmost layer of the system architecture and mainly comprises of the user interface. This interface accepts user input and returns the requested information in a human readable format.
The Logic Tier is where computation of user supplied input and database queries are combined. This tier is mainly responsible for application coordination, logical decisions, and calculations.
The Data Tier is where information is stored and retrieved. The tier is responsible for interaction with the database and for passing information back to the Logic Tier.
User Input (request)
Retrieve resources and process request
Queries are processed and sent back to the Logic Tier
The user is presented with the requested data.
Computations are completed and requested data is sent back to the Presentation Tier

The subsystem above displays our three-tier architecture - which consists of a Presentation Tier, a Logic Tier, and a Data Tier. Each tier has a specific responsibility to the entire system. The presentation tier handles the user interface level. This layer holds the information dealing with the Chef, Manager, Busboy/server, and Customer Interfaces.

The logic tier is the middle tier of this architecture, which handles communication between the top and bottom tiers, 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. In this tier, the Controller is utilized to facilitate tasks between the tiers.

The lowest tier is the data tier, 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 tier only needs to worry about its own operations, and has no knowledge of the other existing tiers. Due to tier of isolation, changes made in one tier will not impact the other tiers. As such, the architecture makes testing and diagnosing issues more manageable.

9.3 Mapping Subsystems to Hardware

The QuickBytes application will be accessed via clients running on tablets and POS systems in the restaurant. There will also be an API server and database. The application has a RESTful API; POST and GET requests are initiated from the client, and will send 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.

The application is written in Python, and also utilizes the Django framework for quick templating and object-orientation. The persistence layer is implemented using SQLite. To store data permanently, the application needs an external database that is hosted on a different server to send the information to the application in real time. The application is using HTML, CSS and Bootstrap on the frontend.

9.4 Persistent Data Storage

Data that needs to outlive a single execution of the system includes the following:

- Usernames and Passwords
- Orders and Order History
- Comments and Complaints
- Business Income and Expenditure

- Customer Payment information (For easier payment access)
- Discounts and rewards
- Notifications
- Meal Ratings
- Table Reservations
- Customer Order time

9.5 Global Control Flow

Execution Orderness: The application is event driven. The user creates an action for the system, and the system handles it from there. The initial action is the same for all users, which is logging. If the user is a customer, they will have different options rather than if they were a manager. Customers can view order history, place orders from the menu, view their favorite meals, file complaints, and see discounts they have earned. The manager experience differs. Managers can restock ingredients, view income and expenses, schedule employees for shifts and breaks, and process customer grievances.

9.6 Hardware Requirements

Our application is designed to work with all modern technology. The preferred experience is a desktop or laptop computer. Supported OS are Mac, Windows, and Linux. Any web enabled device can access our system via web browser. Phones and tablets are supported as well.

10. Algorithms and Data Structures

10.1 Algorithms

There are many important algorithms that can help implement many of the use cases defined in this report. So far, the majority of our algorithms are not very complex in nature. Most algorithms involve simple iterative constructs and consist of updating the persistence layer for the application.

Searching and sorting algorithms, which are more complex, but can also help code become more efficient as the code base grows, may also be helpful in future development. It has only been several months, but so far we see the user base growing and expect to need to implement more sophisticated algorithms in the near future.

10.2 Data Structures

The primary data structure we are using is the Python dictionary (array). This is because it has a simple O(n) look up time, which does not add much delay to our performance. Python dictionaries are very flexible. It is very easy to add onto a dictionary, and it only takes O(1) time.

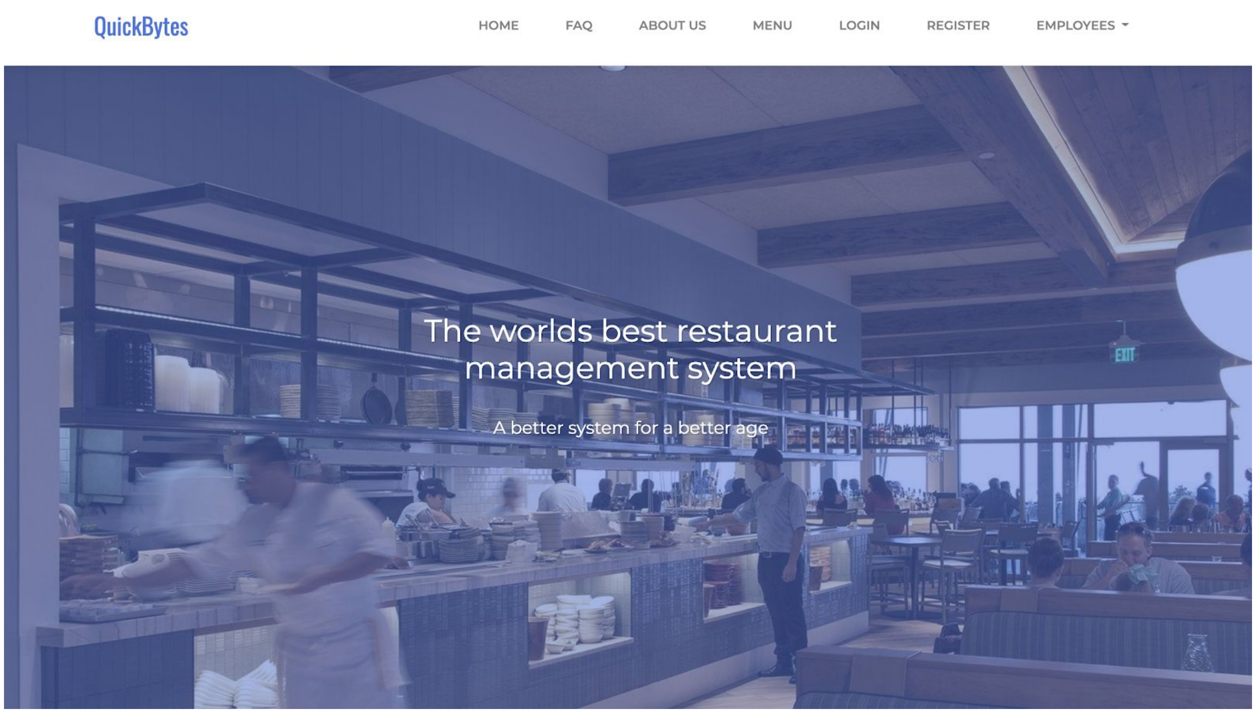
Dictionaries work well with Django, which is not surprising, as Django is a Python framework. However, the array (dictionary) data structure made development much more simple and created a path for rapid prototyping in Python/Django.

The QuickBytes application also makes use of the Queue data structure in the sense that many of the application logic depends on a queue. For example, both the order system and the table cleaning system depend on a FIFO ordering of items in order to serve customers the most efficiently.

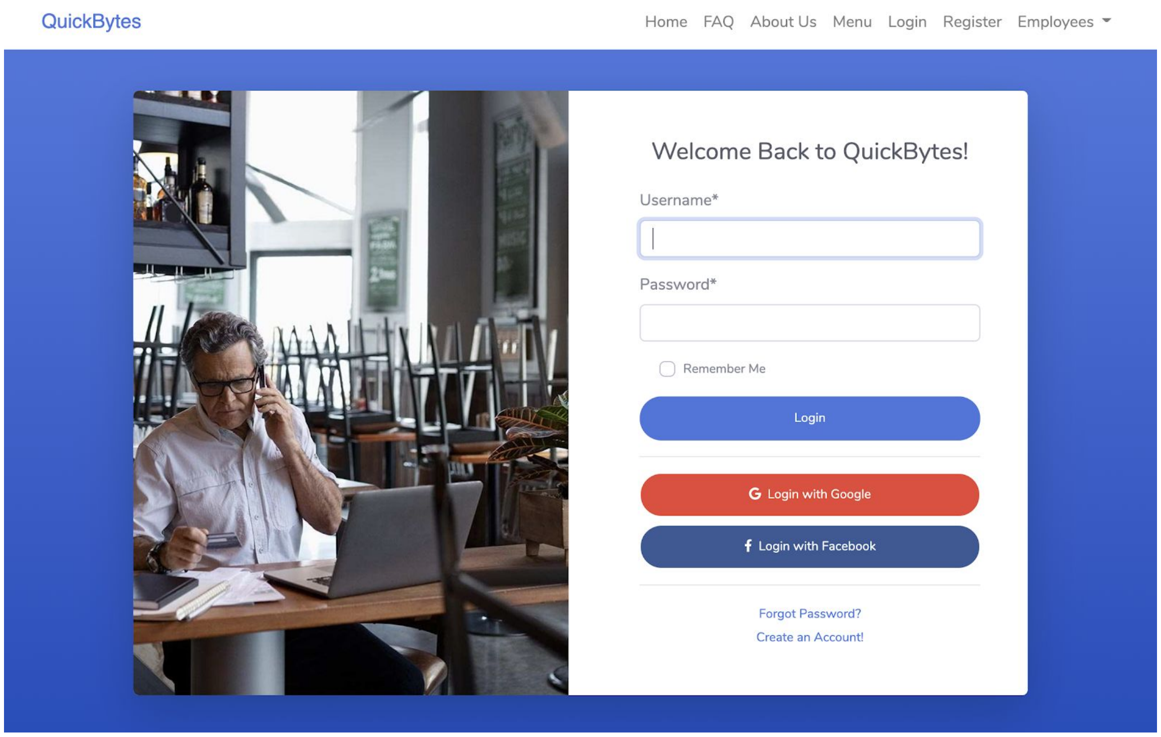
11. User Interface Requirements

Shared Interfaces

Home - Initial view the user sees on requesting the QuickBytes address.


QuickBytes
HOME
FAQ
ABOUT US
MENU
LOGIN
REGISTER
EMPLOYEES
The worlds best restaurant management system
A better system for a better age

Register view - Form for new users to register for the QuickBytes application.


QuickBytes

Home FAQ About Us Menu Login Register Employees

Welcome Back to QuickBytes!

Username*

Password*

Remember Me

Login

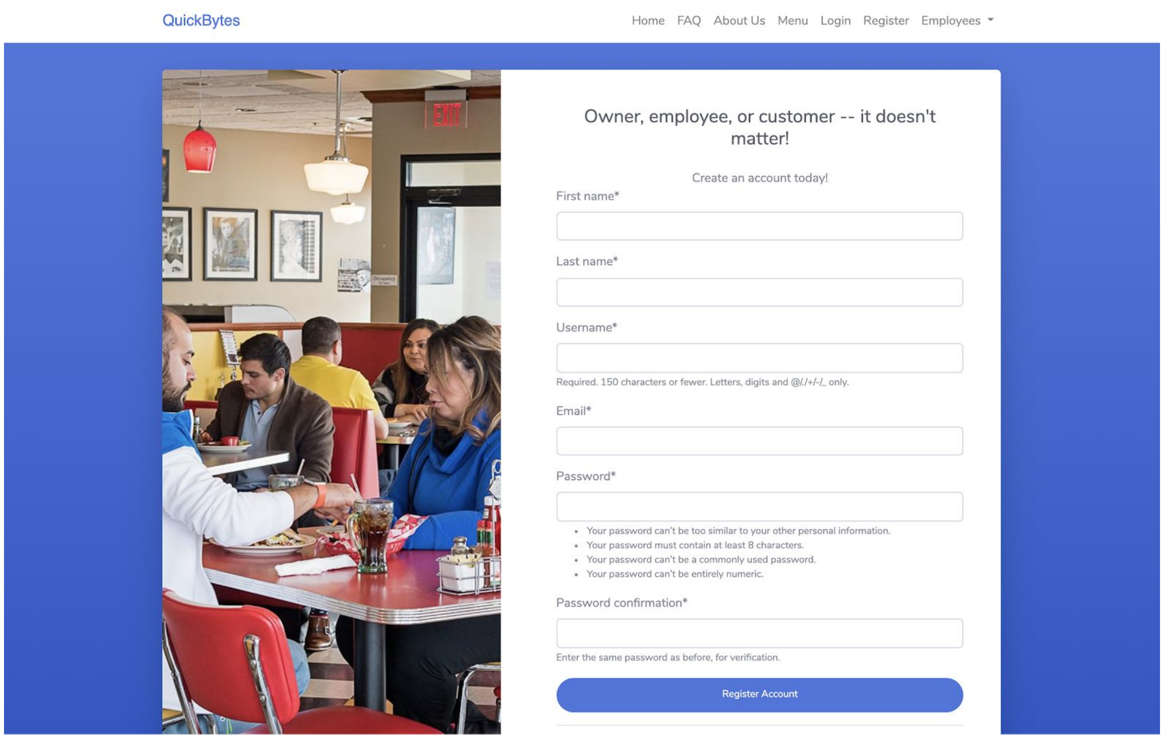
Login with Google

Login with Facebook

Forgot Password?

Create an Account!

Login view - Form for returning users to log in to the QuickBytes application.


QuickBytes

Home FAQ About Us Menu Login Register Employees

Owner, employee, or customer -- it doesn't matter!

Create an account today!

First name*

Last name*

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

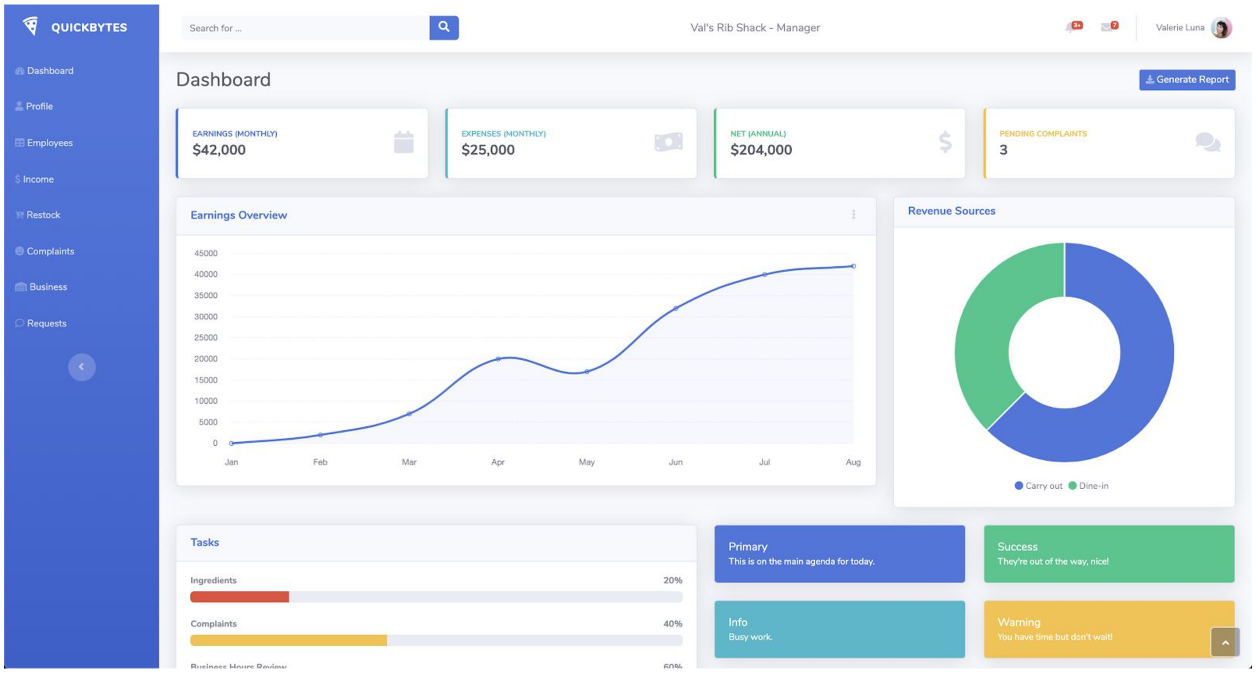
Password confirmation*

Enter the same password as before, for verification.

Register Account

Manager Interface

Dashboard - A manager can view and filter restaurant revenue and inventory.



Dashboard

EARNINGS (MONTHLY)	$42,000
EXPENSES (MONTHLY)	$25,000
NET (ANNUAL)	$204,000
PENDING COMPLAINTS	3

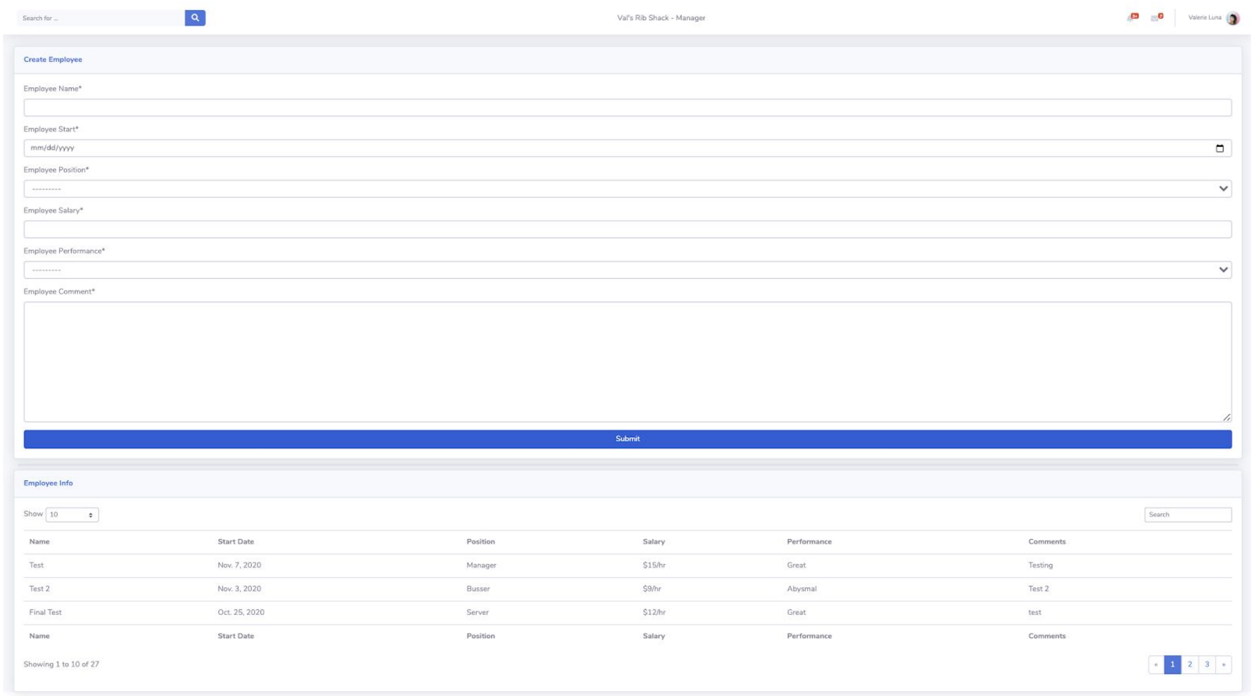
Primary
This is on the main agenda for today.

Success
They're out of the way, nice!

Info
Busy work.

Warning
You have time but don't wait!

Employee Creation - A manager can create and view existing employees for a restaurant.



Create Employee

Employee Name*

Employee Start*
mm/dd/yyyy

Employee Position*

Employee Salary*

Employee Performance*

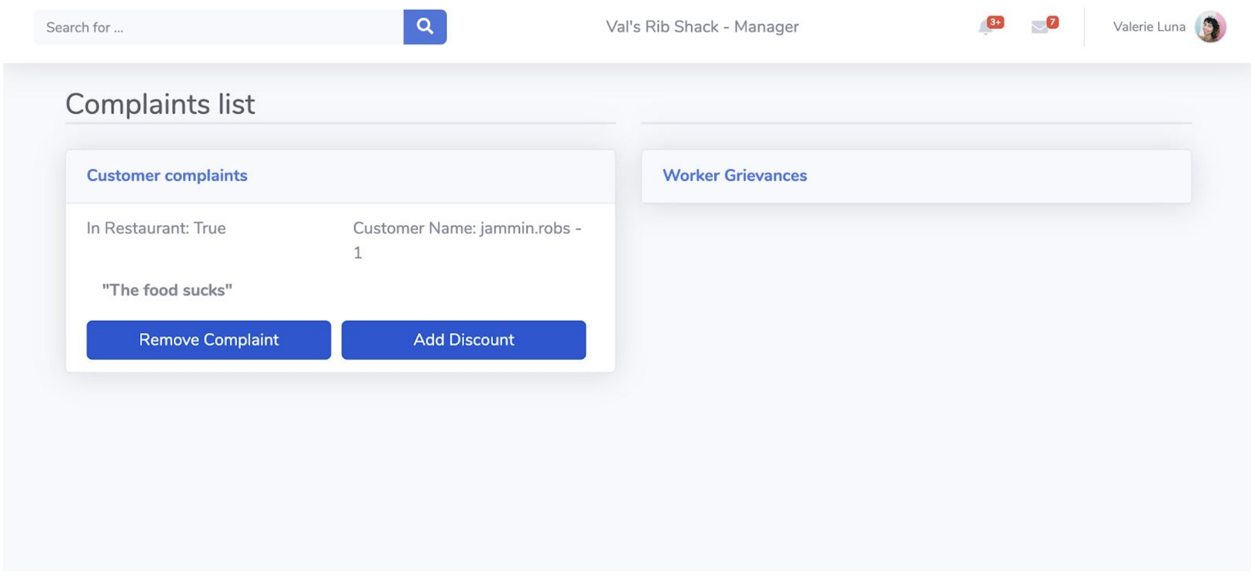
Employee Comment*

Employee Info

Name	Start Date	Position	Salary	Performance	Comments
Test	Nov. 7, 2020	Manager	$15/hr	Great	Testing
Test 2	Nov. 3, 2020	Busser	$9/hr	Abysmal	Test 2
Final Test	Oct. 25, 2020	Server	$12/hr	Great	test
Name	Start Date	Position	Salary	Performance	Comments

Showing 1 to 10 of 27

Complaints - A manager can view and address complaints from both employees and customers.



Complaints list

Customer complaints

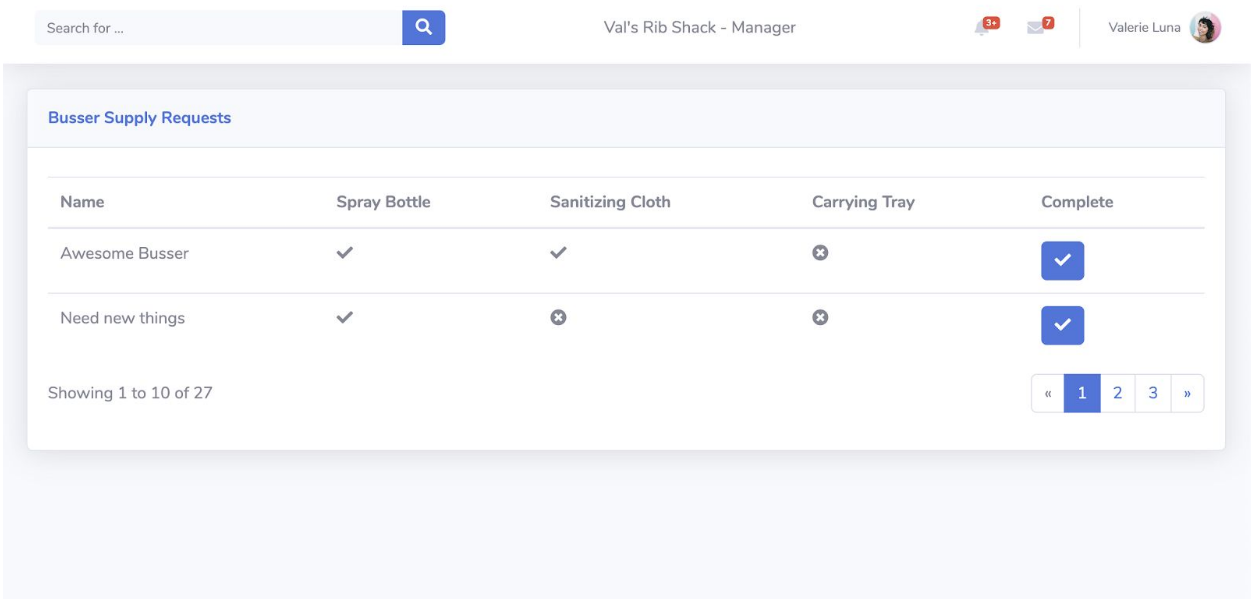
In Restaurant: True

Customer Name: jammin.robs - 1

"The food sucks"

Worker Grievances

Supply Requests - A manager can view and fulfill supplies requests from bussers on the floor.



Busser Supply Requests

Name	Spray Bottle	Sanitizing Cloth	Carrying Tray	Complete
Awesome Busser	✓	✓	⊗	✓
Need new things	✓	⊗	⊗	✓

Showing 1 to 10 of 27

Host(ess) Interface

Dashboard - Allows a host(ess) to view and assign tables, as well as create reservations.

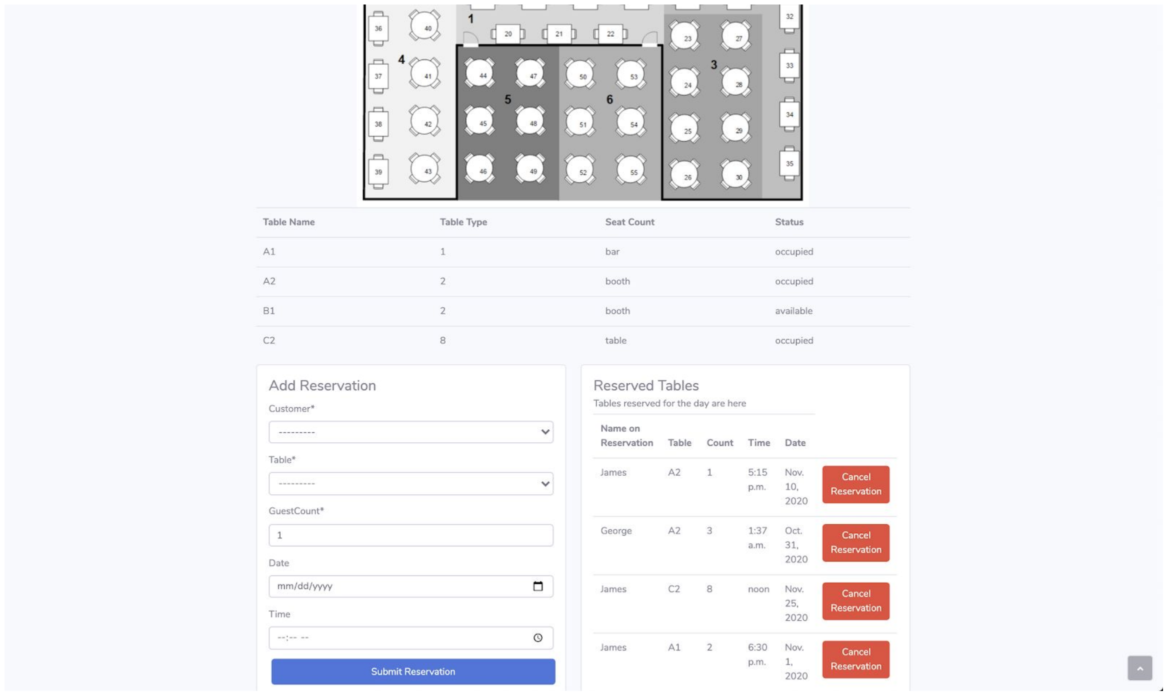


Table Name	Table Type	Seat Count	Status
A1	1	bar	occupied
A2	2	booth	occupied
B1	2	booth	available
C2	8	table	occupied

Add Reservation

Customer*: ---------

Table*: ---------

GuestCount*: 1

Date: mm/dd/yyyy

Time: --:-- --

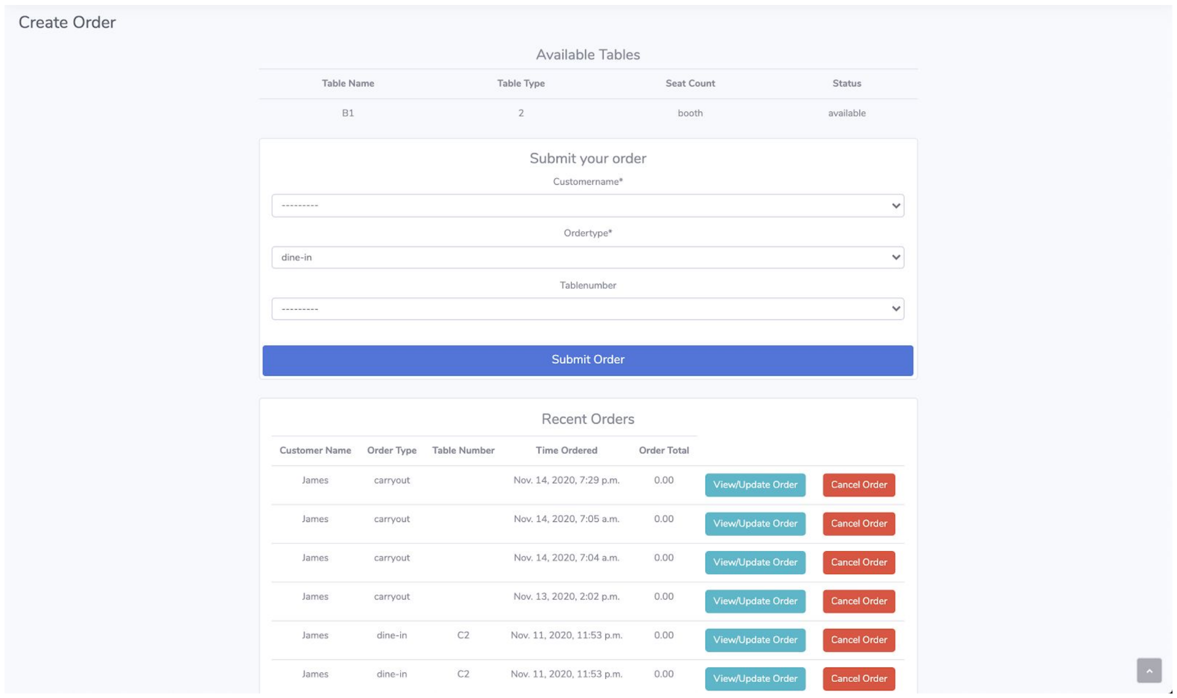
Submit Reservation

Reserved Tables

Tables reserved for the day are here

Name on Reservation	Table	Count	Time	Date	
James	A2	1	5:15 p.m.	Nov. 10, 2020	Cancel Reservation
George	A2	3	1:37 a.m.	Oct. 31, 2020	Cancel Reservation
James	C2	8	noon	Nov. 25, 2020	Cancel Reservation
James	A1	2	6:30 p.m.	Nov. 1, 2020	Cancel Reservation

Orders - Allow a host to place orders on customers' behalf when they are dining take-out.



Create Order

Available Tables

Table Name	Table Type	Seat Count	Status
B1	2	booth	available

Submit your order

Customername*: ---------

Ordertype*: dine-in

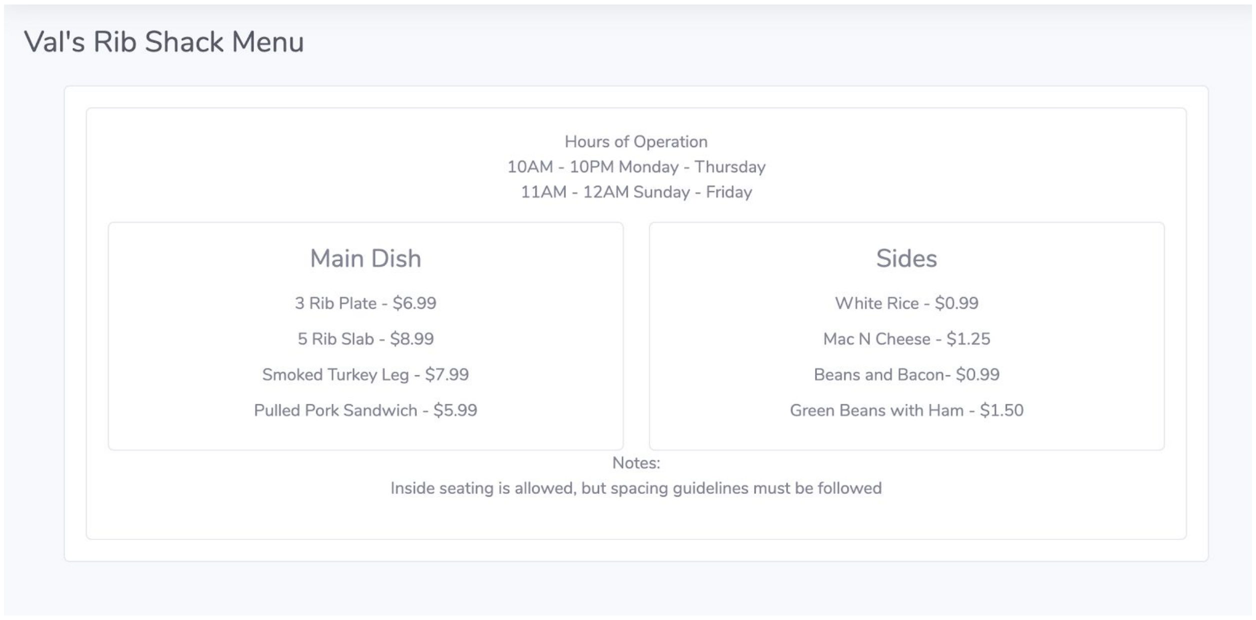
Tablenumber: ---------

Submit Order

Recent Orders

Customer Name	Order Type	Table Number	Time Ordered	Order Total		
James	carryout		Nov. 14, 2020, 7:29 p.m.	0.00	View/Update Order	Cancel Order
James	carryout		Nov. 14, 2020, 7:05 a.m.	0.00	View/Update Order	Cancel Order
James	carryout		Nov. 14, 2020, 7:04 a.m.	0.00	View/Update Order	Cancel Order
James	carryout		Nov. 13, 2020, 2:02 p.m.	0.00	View/Update Order	Cancel Order
James	dine-in	C2	Nov. 11, 2020, 11:53 p.m.	0.00	View/Update Order	Cancel Order
James	dine-in	C2	Nov. 11, 2020, 11:53 p.m.	0.00	View/Update Order	Cancel Order

Restaurant Data - A host(ess) can view information about the hours and menu for the restaurant.



Val's Rib Shack Menu

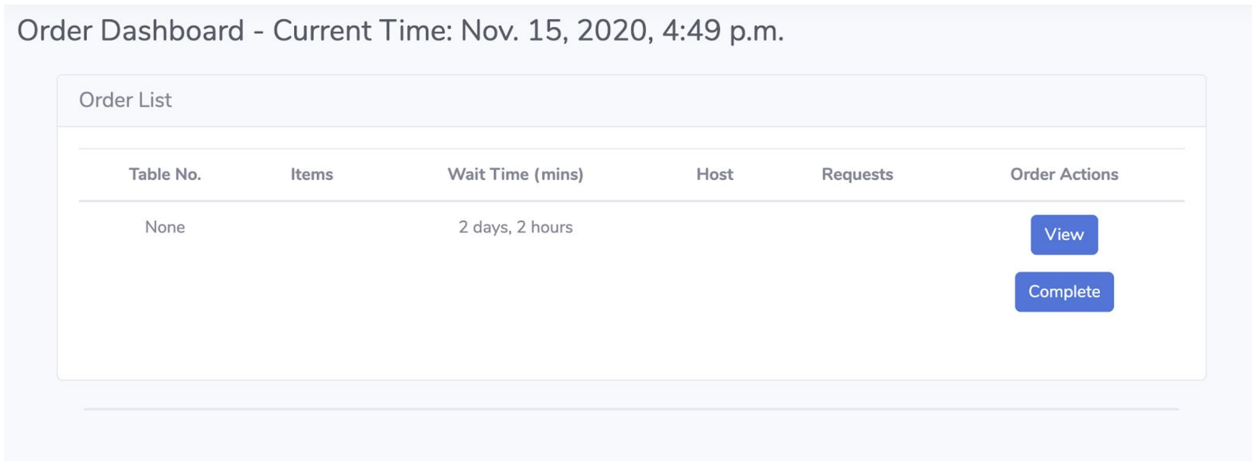
Hours of Operation
10AM - 10PM Monday - Thursday
11AM - 12AM Sunday - Friday

Main Dish	Sides
3 Rib Plate - $6.99	White Rice - $0.99
5 Rib Slab - $8.99	Mac N Cheese - $1.25
Smoked Turkey Leg - $7.99	Beans and Bacon- $0.99
Pulled Pork Sandwich - $5.99	Green Beans with Ham - $1.50

Notes:
Inside seating is allowed, but spacing guidelines must be followed

Chef Interface

Dashboard - Enables the kitchen staff to view incoming orders and mark them complete.

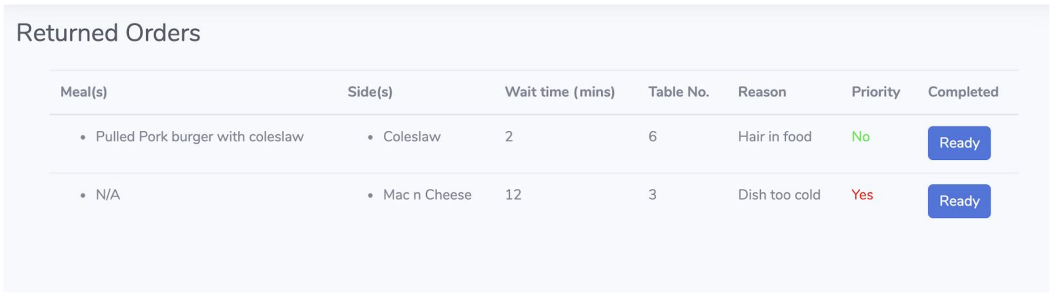


Order Dashboard - Current Time: Nov. 15, 2020, 4:49 p.m.

Order List

Table No.	Items	Wait Time (mins)	Host	Requests	Order Actions
None		2 days, 2 hours			View Complete

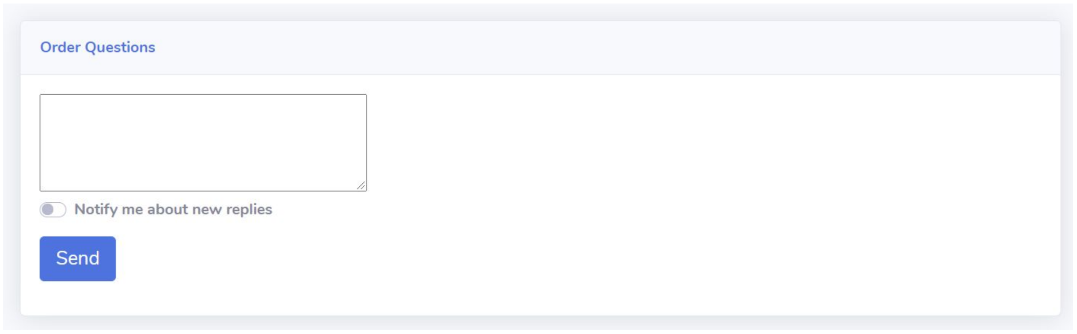
Returned orders - A list of orders that were returned by the server and that need to be remade for various reasons.



Returned Orders

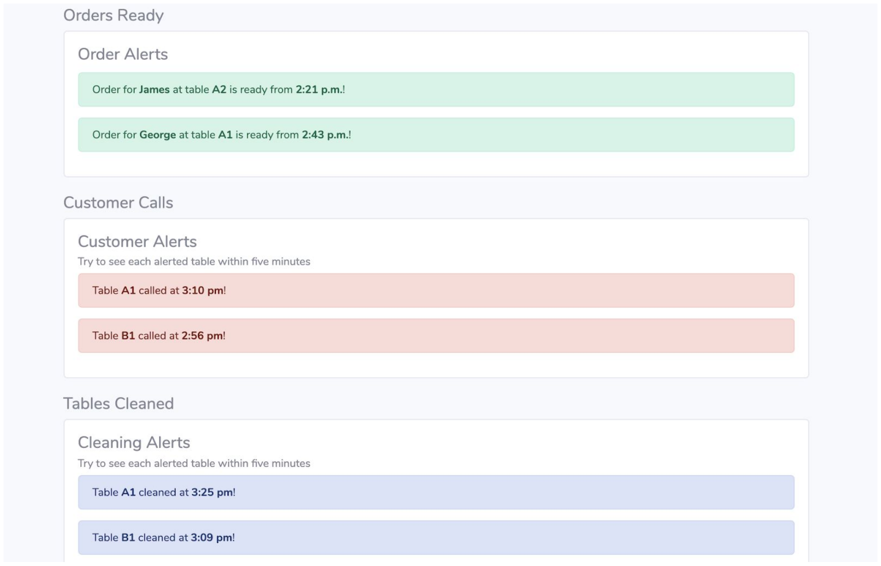
Meal(s)	Side(s)	Wait time (mins)	Table No.	Reason	Priority	Completed
• Pulled Pork burger with coleslaw	• Coleslaw	2	6	Hair in food	No	Ready
• N/A	• Mac n Cheese	12	3	Dish too cold	Yes	Ready

Message queue - Provides a way for kitchen staff to send messages and/or questions to the server regarding orders in their queue.



Server Interface

Alerts - Displays a list of important notifications for servers, such as orders that are ready, tables that have pinged their server and tables that are clean and ready.



Orders Ready

Order Alerts

Order for **James** at table **A2** is ready from **2:21 p.m.**!

Order for **George** at table **A1** is ready from **2:43 p.m.**!

Customer Calls

Customer Alerts

Try to see each alerted table within five minutes

Table **A1** called at **3:10 pm**!

Table **B1** called at **2:56 pm**!

Tables Cleaned

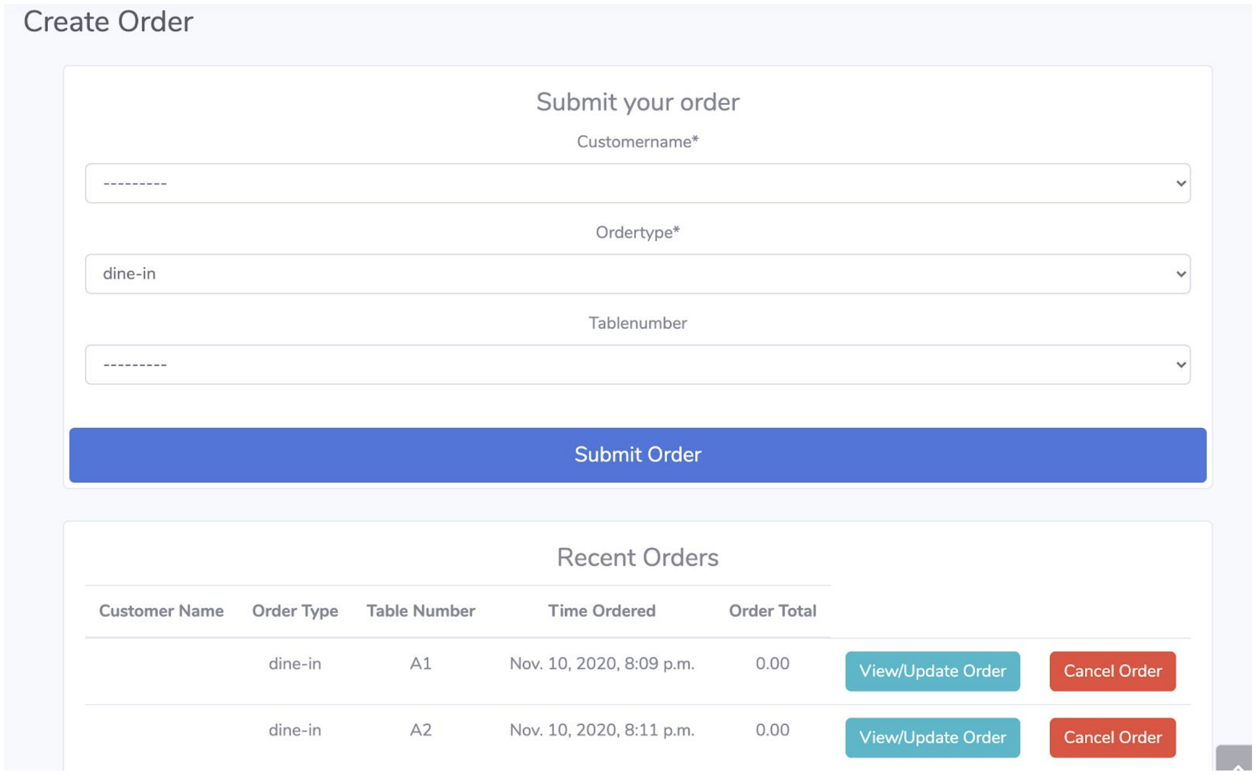
Cleaning Alerts

Try to see each alerted table within five minutes

Table **A1** cleaned at **3:25 pm**!

Table **B1** cleaned at **3:09 pm**!

Order - Enables servers to submit customer orders to the kitchen for preparation.



Create Order

Submit your order

Customername*

Ordertype*

dine-in

Tablenumber

Submit Order

Recent Orders

Customer Name	Order Type	Table Number	Time Ordered	Order Total		
	dine-in	A1	Nov. 10, 2020, 8:09 p.m.	0.00	View/Update Order	Cancel Order
	dine-in	A2	Nov. 10, 2020, 8:11 p.m.	0.00	View/Update Order	Cancel Order

Tables - Displays a list of the servers' active tables and allows servers to release tables once their parties have left.

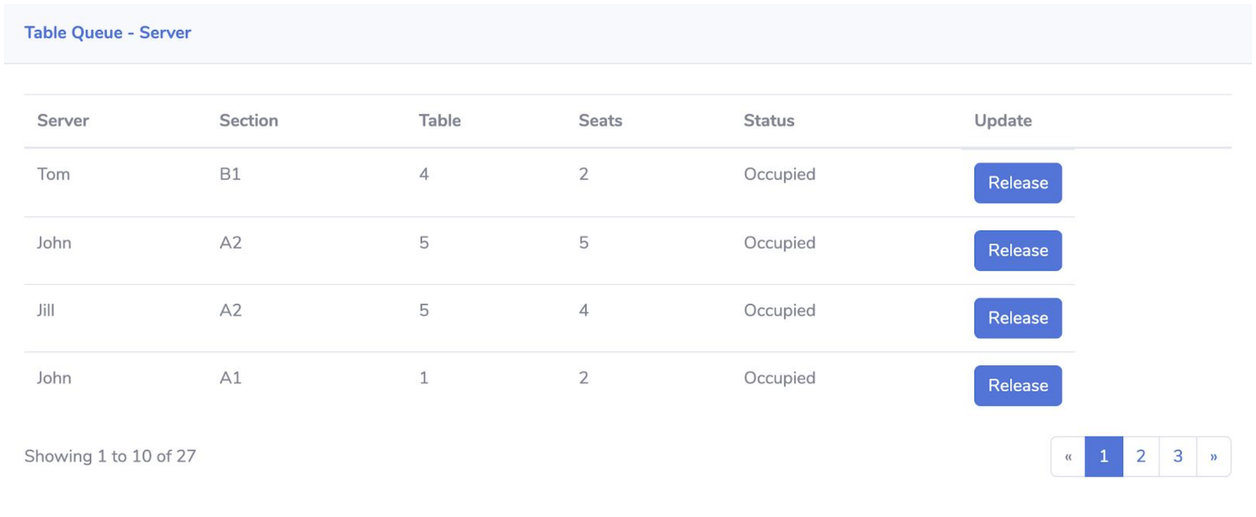


Table Queue - Server

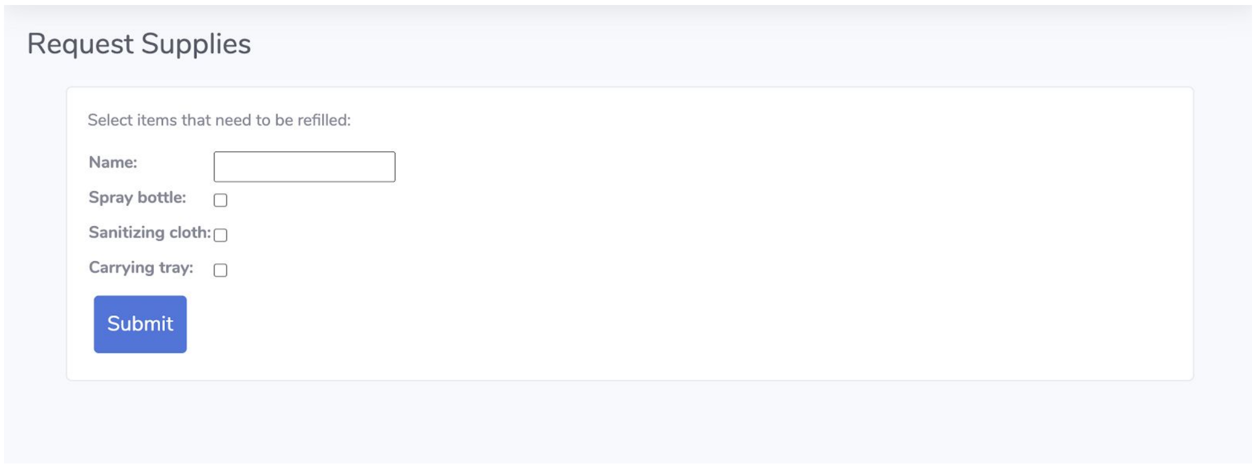
Server	Section	Table	Seats	Status	Update
Tom	B1	4	2	Occupied	Release
John	A2	5	5	Occupied	Release
Jill	A2	5	4	Occupied	Release
John	A1	1	2	Occupied	Release

Showing 1 to 10 of 27

« 1 2 3 »

Busser Interface

Supplies - Provides an interface for a busser to request additional cleaning supplies for their area.



Request Supplies

Select items that need to be refilled:

Name:	
Spray bottle:	☐
Sanitizing cloth:	☐
Carrying tray:	☐

Submit

Table Queue - A list of dirty tables, with functionality to mark them as “clean” once they have been cleared and sanitized.

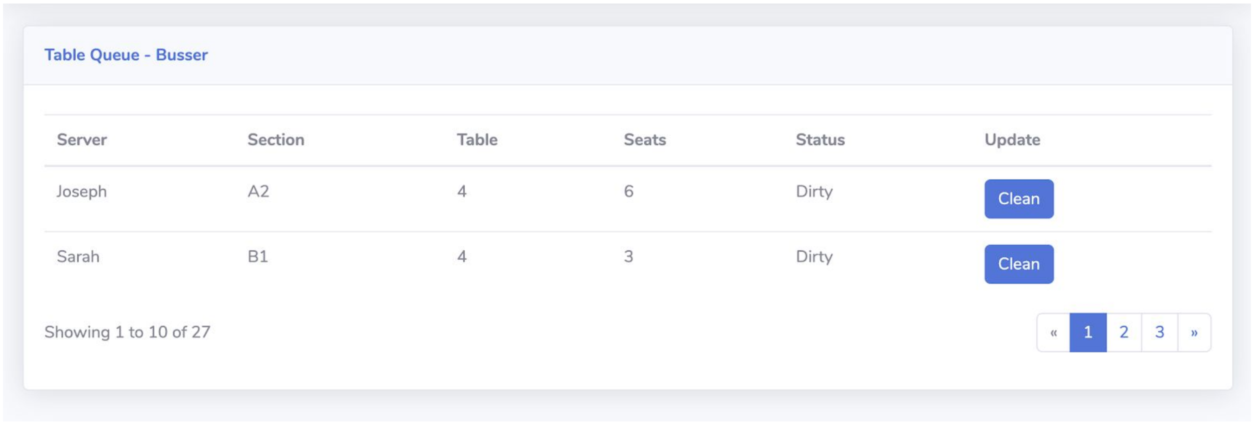


Table Queue - Busser

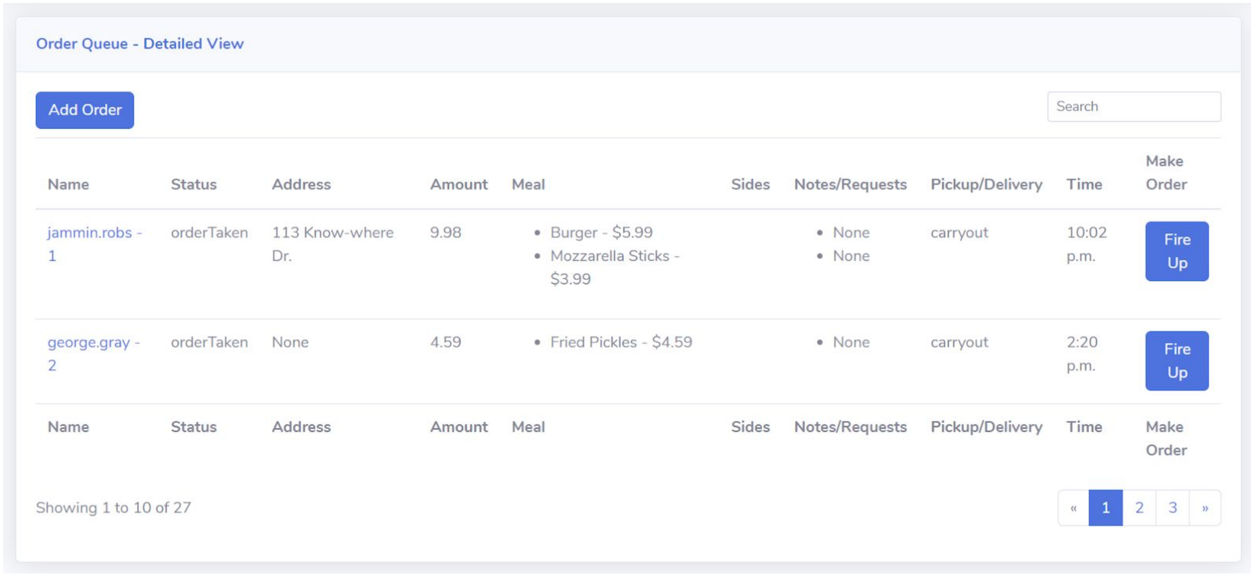
Server	Section	Table	Seats	Status	Update
Joseph	A2	4	6	Dirty	Clean
Sarah	B1	4	3	Dirty	Clean

Showing 1 to 10 of 27

« 1 2 3 »

Delivery Liaison Interface

Dashboard - Allows the delivery liaison to manage and add orders in their queue.



Order Queue - Detailed View

Add Order

Search

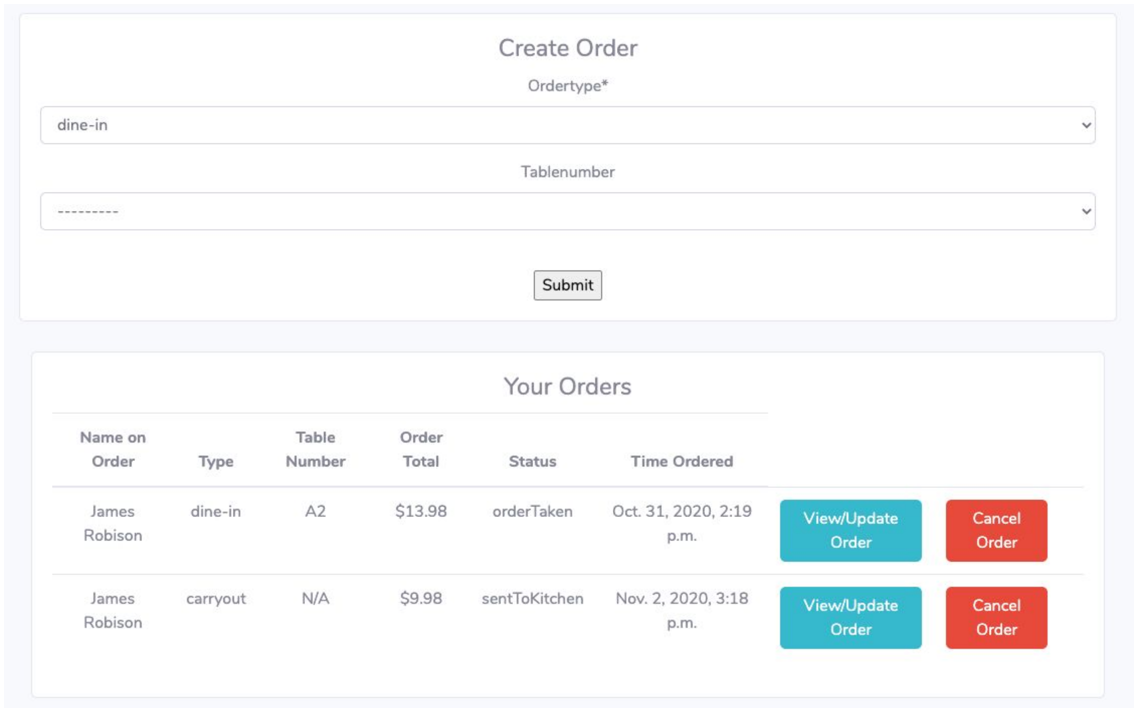
Name	Status	Address	Amount	Meal	Sides	Notes/Requests	Pickup/Delivery	Time	Make Order
jammin.robs - 1	orderTaken	113 Know-where Dr.	9.98	• Burger - $5.99 • Mozzarella Sticks - $3.99		• None • None	carryout	10:02 p.m.	Fire Up
george.gray - 2	orderTaken	None	4.59	• Fried Pickles - $4.59		• None	carryout	2:20 p.m.	Fire Up
Name	Status	Address	Amount	Meal	Sides	Notes/Requests	Pickup/Delivery	Time	Make Order

Showing 1 to 10 of 27

« 1 2 3 »

Customer Interface

Order - Provides an interface by which customers can place an order online.



Create Order

Ordertype*

dine-in

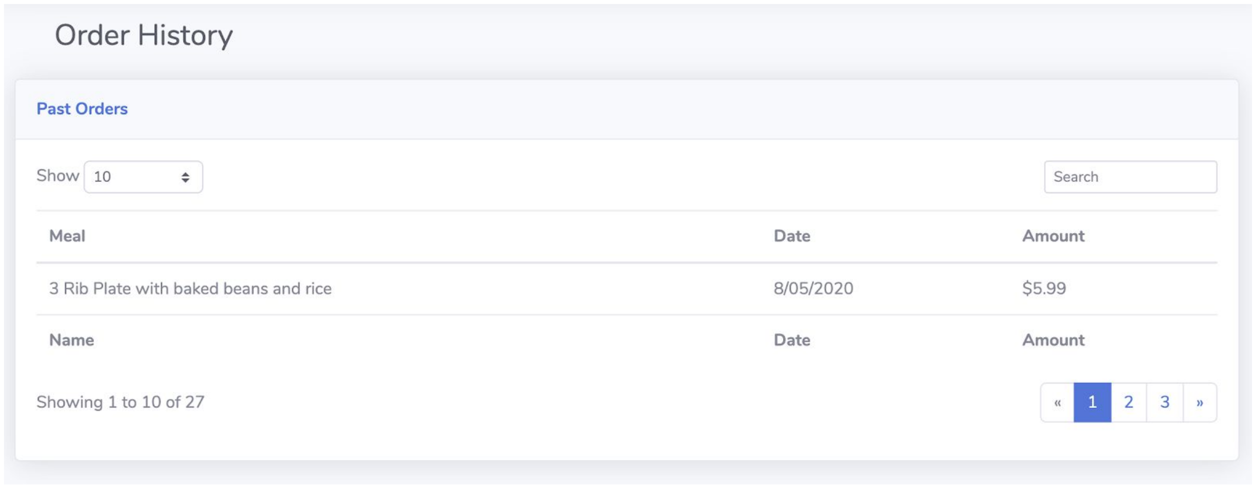
Tablenumber

Submit

Your Orders

Name on Order	Type	Table Number	Order Total	Status	Time Ordered		
James Robison	dine-in	A2	$13.98	orderTaken	Oct. 31, 2020, 2:19 p.m.	View/Update Order	Cancel Order
James Robison	carryout	N/A	$9.98	sentToKitchen	Nov. 2, 2020, 3:18 p.m.	View/Update Order	Cancel Order

Order History - Allows customers to view their recent orders.



Order History

Past Orders

Show 10

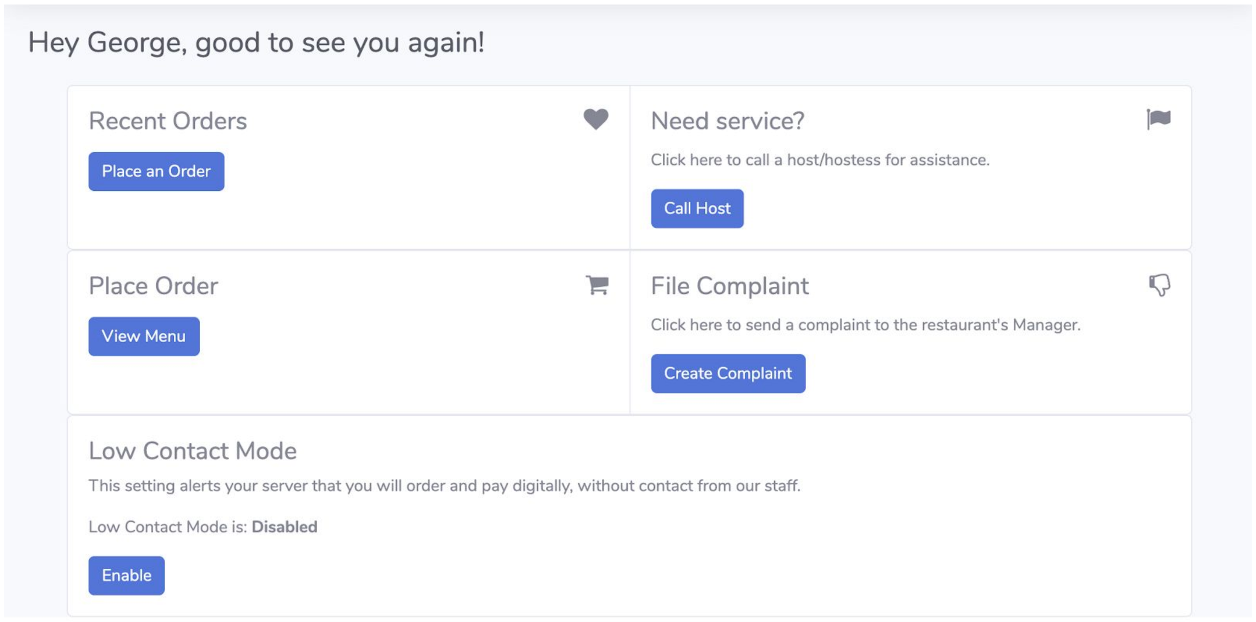
Search

Meal	Date	Amount
3 Rib Plate with baked beans and rice	8/05/2020	$5.99
Name	Date	Amount

Showing 1 to 10 of 27

« 1 2 3 »

Service Requests - An interface that allows customers to call for service when they need something, such as a drink refill or the check.


Hey George, good to see you again!

Recent Orders

Need service?

Click here to call a host/hostess for assistance.

Place Order

File Complaint

Click here to send a complaint to the restaurant's Manager.

Low Contact Mode

This setting alerts your server that you will order and pay digitally, without contact from our staff.

Low Contact Mode is: **Disabled**

12. Design of Tests

12.1 Unit Tests

Test Case Identifier: TC-1 **Use Case Tested:** UC-1 **Pass/Fail Criteria:** Test passes if user can initiate and complete an order **Input Data:** customerName, orderType, status, table, menuItem, specialInstructions, allergies, server	
Test Procedure	**Expected Result**
Step 1: User initiates a new order filling in customerName and orderType. **Step 2**: User selects addItem **Step 3**: User fills in menuItem, specialInstructions, allergies, server, and	Upon successful completion the user is redirected to their main order page. The user is then taken to the addItem Page. The user upon successful completion will

status	be redirected to the main order page.

Test Case Identifier: TC-2 **Use Case Tested:** UC-2 **Pass/Fail Criteria:** Test passes if the user can view active orders and chef can change order status. Test fails if query does not return table status or changes cannot be made to order status **Input Data:** customerName or table	
Test Procedure	**Expected Result**
Step 1: User enters the customerName or table. Chef enters change to order status **Step 2**: Order status is displayed	Upon successful query the customer is shown the status of their order. Upon success, the order status changed by chef is updated in the database.

Test Case Identifier: TC-3 **Use Case Tested:** UC-3 **Pass/Fail Criteria:** Test passes if customers can view tables, hosts/hostesses can view and assign tables, servers and bussers can change the status of a table. **Input Data:** table	
Test Procedure	**Expected Result**
Step 1: Host views available tables and assigns table to customer **Step 2**: Server marks guests have left and table needs to be cleaned **Step 3**: Busboy cleans and marks that table is	Upon success, available tables should be able to be queried by customers and employees. Upon success, Hosts/Hostesses and servers can assign tables to customers Upon success, servers and busboys can

cleaned	change the status of tables from dirty to clean or vice versa.

Test Case Identifier: TC-4 **Use Case Tested:** UC-4 **Pass/Fail Criteria:** Test passes if payment is processed by system for customer or server. Test fails if payment is not processed by the bank. **Input Data:** price, customerName, phone, email, cardNumber, expirationDate, securityCode, zip	
Test Procedure	**Expected Result**
Step 1: Bill amount is entered into the system. **Step 2**: Financial information of guests are entered or card is swiped. **Step 3**: Verification is returned from the bank that the transaction was successful.	Upon Success, the system will receive confirmation from bank that transaction was successfully processed.

Test Case Identifier: TC-5 **Use Case Tested:** UC-5 **Pass/Fail Criteria:** Test passes if employee or customer can successfully log in. Fails if employee or customer is unable to log in to the system. **Input Data:** email, username or password	
Test Procedure	**Expected Result**
Step 1: User enters username or password **Step 2**: User submits information	If successful, the user is directed to the appropriate landing page. If not successful, the user is asked to try again and provided the option to reset password.

Test Case Identifier: TC-6 **Use Case Tested:** UC-6 **Pass/Fail Criteria:** Low-contact mode is able to be engaged **Input Data:** Boolean value from user	
Test Procedure	**Expected Result**
Step 1: User selects low-contact mode **Step 2**: System alerts the restaurant, and guest orders and pays exclusively through the application.	If successfully, instructions for low contact mode will appear to customer, and they will be walked through the ordering and payment processes

Test Case Identifier: TC-7 **Use Case Tested:** UC-7 **Pass/Fail Criteria:** Passes if the user can put in request to the service queue. **Input Data:** table, needHelp(true)	
Test Procedure	**Expected Result**
Step 1: customer selects that they desire service, and enters table number **Step 2**: Their table is placed in service queue for server to see **Step 3**: After receiving assistance, the server takes the table out of the queue.	If successful the customer can select the request service button and their table will appear in the service queue. Server will be able to mark that table was served and they will be popped out of queue

Test Case Identifier: TC-8 Use Case Tested: UC-8 Pass/Fail Criteria: Passes if a user can see the waiting list, view the position of a customer in line and estimate the time until they are seated. Input Data: customer name, current time	
Test Procedure	**Expected Result**
Step 1: User enters the customer’s name and current time **Step 2**: User verifies the customers place in line and estimated wait time.	If successful, the user will see the customer placed on the waitlist. The user will be able to see the time the customer was placed on the waitlist and estimate the time until they are seated

Test Case Identifier: TC-9 Use Case Tested: UC-9 Pass/Fail Criteria: Manager may schedule an employee for a shift and view the schedule Input Data: Employee, schedule, date and time	
Test Procedure	**Expected Result**
Step 1: Manager will select an employee **Step 2**: Manager will set a time and date for their shift **Step 3:** Manager will confirm that the employe has been scheduled	The app will accept the user entered. The app will set the time and date for the scheduled shift and redirect to the schedule. The newly entered shift will be reflected on the schedule.

Test Case Identifier: TC-10 **Use Case Tested:** UC-10 **Pass/Fail Criteria:** Users will be able to track and order ingredients for the restaurant **Input Data:** Ingredient, quantity	
Test Procedure	**Expected Result**
Step 1: User will select the ingredient needed. **Step 2**: User will select the quantity needed. **Step 3**: User will verify the amount	The app will accept the ingredient selected. The app will accept the quantity needed. The user will be able to verify that the correct amount and type of ingredients have been ordered.

Test Case Identifier: TC-11 **Use Case Tested:** UC-11 **Pass/Fail Criteria:** Host(ess) or customer is successfully able to place table reservation **Input Data**: guestName, guestPhone, guestEmail, table, date, partyCount	
Test Procedure	**Expected Result**
Step 1: User supplies customer information **Step 2**: User enters table choice and date **Step 3**: User received confirmation or denial message	The app will accept the user information and add the table reservation to the DB.

Test Case Identifier: TC-12

Use Case Tested: UC-12

Pass/Fail Criteria: Successful if employee can clock in and out of system

Input Data: employee id, password, startTime, endTime

Test Procedure	**Expected Result**
Step 1: employee clocks in to work using id and password **Step 2**: employee clocks out of work using id and password	Employee clocks in using id and password, employee clocks out using id and password. System tracks how many hours employee worked

Test Case Identifier: TC-13

Use Case Tested: UC-13

Pass/Fail Criteria: Successful if new account is created successfully. Fail if account is unable to be created

Input Data: username, password, repeated password, email, repeated email

Test Procedure	**Expected Result**
Step 1: User enters chosen username, email, and password **Step 2**: User is requested to re-enter email and password **Step 3**: If the email and password match, the account is created and added to DB	The user is able to enter required information, and their account is added to the DB.

(*) Test cases are still being implemented, and we hope to have them completed by Demo 2.

12.2 Test Coverage

Our test cases cover all of the essential modules in our application. Each key actor that we have worked with over the course of the semester has its own suite of test cases, and together these work together to provide holistic coverage for the QuickBytes application. We also have test cases that go beyond the individual actors and modules to cover additional features, such as the login functionality. By testing each of our individual units of code, we have achieved optimal test coverage throughout the entire application.

12.3 Integration Testing Strategy

Integration is of critical importance for the QuickBytes application, as all of the individual software modules do need to be tested as a unified whole in order to validate the core functionality of the application. Our approach to integration testing consists of a bottom-up approach, which means first testing the individual units, such as the functional requirements of the Busser or the Host(ess), and then testing these individual pieces together, such as validating that when a Busser marks a table as "clean," that data is received by the Host(ess) and able to be processed by the Host(ess) accordingly. Through this approach, we know that if each individual unit has been tested successfully and the communication between units is also successful, the application is stable.

13. History of Work, Current Status, and Future Work

13.1 History of Work

August 17th - August 23rd
We formed a group and created a Discord server where we could chat about our ideas for the team project. We ultimately decided on a restaurant automation system and came up with a team name. We started discussing potential features and the actors for our project. We discussed potential technologies/languages that we wanted to use for the project, and ultimately landed on the Python/Django stack. We decided that our main actors would be the manager, chef, delivery liaison, host/hostess, customer, busser and server.

August 24th - September 2nd
We started brainstorming for our project proposal and divided up responsibilities among the group based on the actors that we had previously discussed. Each member of our group wrote up their own problem statement, treatment and plan of action for the roles that they were assigned. We continued to flush out our ideas for the project, and also decided on some Covid-19 specific features to make the application timely and relevant for the current worldwide situation.

September 3rd - September 13th
We set up a Google Cloud virtual machine and started making our initial changes on the server. There were a lot of learning curves at this stage in the process. Many of our group had not used Python/Django prior to this project and were also new to SSH-ing into the Google Cloud server. There were some initial growing pains as everyone set up SSH keys, configured their IDEs (primarily Visual Studio Code) to enable SSH-ing into the Google Cloud Server, and reviewed learning materials on the syntax of Python and the basic project structure of a Django application.

September 13th - September 23rd
We started working on Report #1 and began creating the assets necessary for the report, such as UML diagrams, specific requirements for each actor, use cases, key stakeholders, etc. Each member of our team wrote the sections for the actors for which they were assigned. We were short on time and did not realize the breadth and depth of Report #1, so we ended up having to submit it somewhat incomplete.

September 24th - October 13th
We continued to build out the specified features for the QUICKBYTES application. Many of the initial growing pains that we experienced when we began working on the app became alleviated at this point and we started making significant progress on the technical aspects and feature requirements that we had set out to accomplish. We added functionality to create customers, to create orders for those customers, and the features required to send those orders to a chef. We also implemented a menu and restaurant seating chart for the application. We also completed the majority of the manager functionality. We worked together to complete Report #2 and submitted this report for review.

October 14th - November 2nd
We had a lot of success in terms of progress on the application. We implemented functionality for the busser to be able to send a request to a manager for more cleaning supplies, while on the floor working to clear tables. We also implemented the ability for a server to submit orders and look at an overview of the tables in their section. We also implemented a login page for the application. We spent a lot of time in the application completing core requirements. The last few days before the Demo 1 due date, we all recorded our own feature recaps and then meshed them together to complete our presentation for Demo 1.

November 3rd - November 30th
At this point, we switched our focus from coding to writing out the final draft of Report #3. Our first two reports were short several sections, so we had a lot of work to do to ensure that our final report was complete and contained all of the required sections, data, diagrams and recaps.

In addition to much of our time spent on the report, we completed the functionality for the busser to receive a queue of dirty tables and then mark those tables as clean and available once they cleared them. This functionality also expanded the functionality of the server, who could now mark tables as ready for cleaning, and the host/hostess to create new tables and assign those tables to servers in various sections. We also implemented some payment features, but realized that we would not be able to submit a real payment using Stripe, so we settled for modifying the database to simulate the payment behavior. We also chose to let go of some features that we originally planned to implement, such as inventory updates, restocking features and adding sides to the menu.

13.2 Current Status

- A host(ess) can create a new table
- A host(ess) can assign a table to a specific server/section
- A host(ess) can make reservations for customers
- A host(ess) can place orders
- A server can release a table for cleaning to the busser for their section
- A server can create and place a new order
- A server can view and edit current orders
- A manager can schedule employees for work
- A manager can read customer complaints and worker grievances
- A manager can apply discounts for customers who complain
- A manager can create new entries in their budget book
- A manager can view requests from bussers for new cleaning supplies
- A manager can create new employees
- A manager can mark a request as “complete” once a busser has received new supplies
- A busser can view a queue of tables that need to be cleaned
- A busser can can mark a table as “Available” again, once the table has been cleared
- A busser can submit a form to request refills on cleaning supplies
- A delivery liaison can add a customer order
- A delivery liaison can update a requested time for delivery or pickup
- A delivery liaison can send an order to the kitchen for prep
- A delivery liaison can respond to a the chefs request for more information
- A chef can request more information from the server/delivery liaison
- A chef can view an order
- A chef can complete an order
- A customer can place an order
- A customer can submit a complaint
- A customer can view their discounts
- A customer can make reservations

13.3 Future Work

Enhanced Delivery Features: Some of the delivery features were deprioritized for our project, due to the time and constraints of third-party app integrations. To expand on this application, we may complete these third-party integrations so that orders can be organized and reprioritized by the delivery liaison according to delivery/pickup times requested. We would also integrate a feature that enables the delivery liaison to notify the delivery driver/customer that an order is ready to be picked up or delivered.

Payment Processing: Due to limitations of time and resources, we were not able to integrate with a third-party payment system, such as Stripe. In order for real customers to pay with real money, we would like to integrate with payment service providers, such as Stripe, PayPal, ApplePay, etc., in order to allow customers to pay for their orders in advance through our application. This would speed up the time staff needs to interact with customers once they come to pick up their food, and it would also allow third-party delivery drivers to come and pick up a customers meal and simply drop it off to them, because the order has already been paid for.

Review System: As QuickBytes becomes more and more popular, we will depend on the reviews of our key actors in order to continue to gain business with other professionals in the restaurant industry. Given the time, we would like to implement an internal review system, which will allow customers and restaurant personnel to review the QUICKBYTES software and give praise and/or make recommendations for how to iterate on the current features or new features that they would like to see in the application.

14. References

Bruegge, Bernd and DuAllen H.Dutoit. Object-Oriented Software Engineering: Using UML, Patterns and Java. Second Edition, Prentice Hall, Upper Saddle River, NJ, 2004.

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

Django Documentation. (2019, December 2). Retrieved November 5, 2020, from https://docs.djangoproject.com/en/3.1/releases/3.0/

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/

Hamilton, Kim, and Miles, Russ. Learning UML 2.0. O’Reilly Media, Inc. 2006.

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

Marsic, Ivan. Software Engineering. New Brunswick: Ivan Marsic, 2012 https://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

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