



Report Three

Project Website:

<https://sites.google.com/scarletmail.rutgers.edu/restaurantautomation2018/home?authuser=1>

Project Blog:

<https://littlebitsse2018.wixsite.com/mysite>

GROUP #4

Team Name: Little Bits

Team Members:

- Esraa Abdelmotteleb
- Manvi Agarwal
- Johnathan Alcantara
- Hagar Elshentenawy
- Lakshmi Sai Nithya Garudadri
- Kevin Honeker
- Zachary Joseph
- Fareen Pourmoussavian
- Dafna Shochat

Contribution Report

Section	Sub-Section	Pts	Esraa Abdelmotteleb	Manvi Agarwal	John Alcantara	Hagar Elshentenawy	Lakshmi Sai Nitya Garudadi	Kevin Honeker	Zachary Joseph	Fareen Pourmousavian	Dafna Shochat	Teams
Project Management [13]	Report Coherence	5	0%	0%	0%	0%	0%	0%	100%	0%	0%	Customer
	Team Coordination	8	11%	11%	11%	11%	11%	11%	11%	11%	11%	Employee
	Summary of Change [5]	5	11%	33%	0%	11%	11%	0%	0%	33%	0%	Manager
Consumer Statement of Requirements [8]		6	11%	11%	11%	11%	11%	11%	11%	11%	11%	
Glossary of Terms [4]		4	11%	11%	11%	11%	11%	11%	11%	11%	11%	
System Requirements [8]	Enumerated Functional Req.	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Enum. Nonfunctional Req.	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	On-Screen Appearance Req.	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Stockholder/Actor Goals	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	UC: Casual Dscription	7	11%	11%	11%	11%	11%	11%	11%	11%	11%	
Functional Requirement Specifications [30]	UC: Case Diagram	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	UC: Traceability Matrix	1	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	UC: Full Description	10	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	System Sequence Diagram	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Effort Estimation [4]	4	0%	0%	0%	0%	0%	100%	0%	0%	0%	
Domain Analysis [25]	Domain Model	15	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	System Operational Contract	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Mathematical Model	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
Interaction Diagrams [40]	UML Diagrams	10	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Description of Diagrams	10	11%	33%	19%	11%	11%	3%	0%	11%	0%	
	Alternate Solution Description	10	11%	11%	17%	11%	11%	17%	11%	0	11%	
	Use Diagram Patterns	10	11%	10%	11%	11%	11%	11%	0%	11%	23%	
	Class Diagram and Interface Specifications [20]	5	11%	33%	11%	11%	11%	11%	0%	11%	0%	
	Signature Structures	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	OCL Diagrams	10	11%	0%	0%	11%	11%	0%	33%	33%	0%	
	Architectual Styles	5	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Identifying Subsystems	3	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Mapping Subsystems to Hardware	3	11%	11%	11%	11%	11%	11%	11%	11%	11%	
System Architecture and System Design [15]	Persistent Data Storage	1	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Network Protocol	1	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Global Control Flow	1	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Hardware Requirements	1	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Algorithm and Data Structures [4]	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
User Interface Deisgn and Implementation [11]	Data Structures	2	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	What Has Changed	4	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Ease to Use	7	11%	11%	11%	11%	11%	11%	11%	11%	11%	
Deisgn of Tests [12]	Coverage Tests	6	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Integration Testing	6	11%	11%	11%	11%	11%	11%	11%	11%	11%	
	Key Accomplishments	2	11%	0%	0%	11%	11%	0%	33%	33%	0%	
History of Work [5]	Future Works	2	11%	0%	0%	11%	11%	0%	33%	33%	0%	
	Responsibility Matrix	1	11%	0%	11%	11%	11%	11%	33%	11%	0%	
	TOTAL	200	21.49	24.69	21.24	21.49	21.49	23.64	25.39	23.69	19.39	

Table of Contents

Customer Statement of Requirements.....	1
Glossary of Terms.....	5
User Stories	8
System Requirements	11
Functional Requirements Specification	18
Effort Estimation	33
USING CLICKS.....	33
USING CASE POINTS.....	37
Domain Analysis.....	41
Domain Model.....	41
Concept definitions:	41
Association Definitions:.....	42
Attribute Definitions.....	44
Traceability Matrix	46
Domain Model Diagram.....	48
Systems Operation Contracts.....	49
Mathematical Model	51
Interaction Diagram	52
Class Diagram and Interface Specifications	82
Class Diagram	82
Data Types and Operations Signatures	85
Traceability Matrix.....	97
Design Patterns.....	100
Object Constraint Language.....	101
System Architecture and System Design	106
Architectural Styles.....	106
Identifying Subsystems.....	109
Mapping Subsystems to Hardware	112
Persistent Data Storage	113
Network Protocol	114
Global Control Flow	115
Hardware Requirements.....	116
Algorithms and Data Structures	117
Algorithms	117
Data Structures.....	124
User Interface Design and Implementation.....	125
Design/GUI	125
Ease of Use	138

LITTLE BITS

Design of Tests	140
Test Coverage	153
Integration Testing Strategy	153
History of Work	154
References.....	157

Summary of Changes

- New algorithms were added; Viewing Shifts Algorithm, Add Shifts Algorithm, Delete Shifts Algorithm, and Chat System
 - New Interaction Diagrams were made to reflect design patterns we learned in class and using outside recourses
 - Customers can make takeout orders. When a customer is placing an order, they can choose the takeout option. They will be then given an order number and a time estimation for when their order will be ready.
 - We now allow payment through gift cards and cryptocurrency.
 - Customers can choose to play Uno or pain tiles using the customer tab.
 - Updated the glossary of terms adding the following terms: Takeout, Cryptocurrency and Gift card.
 - Updated problem statements 1-5 to reflect the added features described under “changes to the system” above.
 - Added problem statement 6: Entertainment for kids
 - Added ST-C-7, ST-C-8 and ST-C-9 to the enumerated functional requirements
 - Added a new use case (UC-7)
 - Updated the traceability matrix to reflect the added use case and the change in the enumerated functional requirements
 - Updated the fully dressed descriptions
 - Updated the descriptions for interaction diagrams to reflect the added design patterns
 - Worked on website security. Changed how Employees sign in and input schedules.
 - Some functions’ return data types have changed. For some functions such as `getInventory()` and `getMenu()` we had said that the method would return an array or a 2D array but we are returning the table from the database, which is more like a structure.
-

- Some parameters of functions have changed such as updateMenu now takes in parameters such as ItemName, Quantity, UnitOfMeasure, ExpirationDate, CostPerUnit because we realized some functions require more information that might need to be used in the method. Another example of this is addEmployee which takes in a lot more parameters such as Name, Position, DateofBirth, PhoneNum, Username, Password, RepeatPassword.
 - After implementing certain methods we realized we changed some of the names such as deleteMenuItem to delMenu for simplicity purposes. Some methods such as Employee Management changed to Employee Portal() which takes you to a page where you have options such as adding and deleting employees
 - Removed time estimation
-

Customer Statement of Requirements

In the modern world, more and more businesses are automating their practices as it has proven to increase efficiency therefore increasing profit. Restaurant industry hasn't caught up yet with many restaurants still operating by taking orders manually. We propose a project to automate many aspects of restaurants using a system interface. This interface will allow easy access to customers, managers, and employees, and will change the way restaurants work. Below are some problems that restaurant owners found that could lower efficiency and lower customer satisfaction.

Customers:

Problem 1: Difficult to find available tables and to efficiently manage seating

Restaurants have a limited number of tables. When there's high traffic on the restaurant, sometimes customers might end up waiting a long time to be seated or come to find out that no tables will be available for the rest of the day. A solution to this problem is a customer interface within an overall restaurant system interface. Using a mobile device or a restaurant tablet, customers should be able to acquire information about availability and expected waiting time before they can be seated. The customer can then input information about the timing and the number of people in the party. A layout of the restaurant with the tables should be available to the customer based on the information that they provided so they can pick their desired table and time. This allows the customers to make an informed decision about whether to drive to a restaurant. An estimation will be made depending on the number of customers already in the restaurant and the number of reservations already made. The app can then use party size and desired timing to prioritize customers for each table and give an approximate waiting time.

Problem 2: Placing orders and getting a waiter's attention is a long process which can be optimized

After customers are seated, a waiter has to take their order which poses two problems. The waiter isn't always ready to take the order as soon as customers are ready to place it. Therefore, customers have to wait until the waiter is nearby so they can grab the waiter's attention and place their order. When the traffic is particularly high in the restaurant, customers sometimes have to wait for a significantly long time before they can just place their orders. The other issue is when customers take varying amounts of time to place an order and the waiter has to keep coming back to check whether they're ready. This can cause some confusion between the two parties as well as can lead to a customer being annoyed and having a poor experience. In addition, the waiter ends up wasting their time by having to move between tables asking every table whether they're ready to place their order. Another issue is trying to get assistance from a waiter. Customers have to flag their waiter down and sometimes aren't seen. Using a tablet placed on each table, customers should be able to view the entire menu and place their orders as soon as they're ready. The tablet should contain menu items that are organized into different categories. The customer can then navigate through the different categories and pick their desired menu items and add them to their cart. Items can also be edited or deleted from this interface. After

they are satisfied with their items, the order can then be placed and sent directly to the chef eliminating the need for a waiter to place the order. The tablet should also allow customers to request their waiter and the waiters should be able to see the notification on their tablet. Using this system, waiters can focus more on different tasks and customer satisfaction.

Problem 3: Long and tedious payment process which can be optimized

At restaurants some people take longer to eat than others. This makes it difficult for waiters to figure out when the customers are done eating and are ready to pay their bill. This means that waiters have to keep an eye on whether the customers are still eating and have to choose the right time to go ask the customers if they're done. In addition, a lot of times customers have to split the bill evenly or in proportions creating some hassle for waiters. This is a tedious task and takes a lot of time and patience and can waste time for waiters to do other things. Using the tablet placed on the customers table, they can select an option on the tablet that will show them their bill. The customers then can have the option to split the bill how they want and pay. An option for tipping will also be available to the customer, which waiters can view as well. The customers can pay using various payment methods such as credit cards, gift cards, PayPal, Cryptocurrency, or cash. When a customer decides to pay in cash, this will send an alert to the waiter to assist the customer. This system is different from the previous ones in that it also gives the customer full control of how they want to pay for their bill. It also gives them the option to go the traditional route by giving them the option to directly request the waiter for assistance.

Problem 4: No proper way for customers to leave feedback and review the restaurant and their service.

For a restaurant to be successful, it's important to get the customers feedback. Customer feedback is important to improve ambience and customer satisfaction. Customers should have the ability to express their thoughts, whether it's a positive or a negative review. Managers should also be able to respond to the customer's feedback and justify the reasons that led to the customers' substandard experience. The manager can use this feedback to improve and can also use the statistics to help manage different parts of the restaurant. We need a rating system that gives the customer an option of rating and adding comments about what they liked and what they disliked. This rating system should also have more advanced features that can show the number of times people have ordered that specific dish and reviews for that dish. Customers should also be able to rate and add reviews for certain employees who served them. This system is different from previous ones in that allows the customers to review their experience at the restaurant.

Problem 5: No entertainment for children

Before and after ordering, children tend to get bored while waiting for their food. To entertain them, we would like an option for children to be occupied while waiting. An option would be a gaming platform where there are a variety of games children can choose from and play while waiting.

Employees:

Problem 1: Difficult to make sure orders are made in the correct order they are placed

Manually taking the order and notifying the chef of them takes time. Chefs often go wrong in prioritizing different orders which leads to a delay in preparing a customer's order. In many cases, there are many special requests that the waiter forgets to inform the chef about, which can lead to low customer satisfaction. These problems can be resolved by a restaurant interface. In this interface, we can take order prioritization out of the chef's hands and should be able to automatically prioritize the orders on a first-come-first-serve basis. There should be different portals within the employee's subsection where the chef will be able to view the orders in the order they need to be made in. The chef will also be able to review the customer's preferences and requests. With this, the chance of making mistakes will significantly decrease since orders will go directly from customers to the chef without any intermediary person.

Problem 2: Difficulty with communication between waiters and the bussers

In today's restaurant world, it pays to be efficient. When customers have to wait for a table to be cleaned before being able to be seated it can lead to low ratings and can deter them from coming back. Generally, waiters don't have a proper system in place to communicate with the bussers. Due to this problem, restaurants waste time and possibly lose out on customers. When a customer has to wait, it can deter them from coming back and can lead to poor reviews and a poor overall experience. The waiters are usually in charge of communicating to other employees when a table has been vacated, when a table is being used, and when a table needs to be cleaned. To make this easier they need a system to properly communicate with the bussers. A solution to this issue is that within an interface, in an employee's portal, the waiters should be able to mark tables that need to be cleaned. Then bussers should be able to see this notification and then can also respond and mark when tables are cleaned and ready to go. Waiters should also have the ability to mark when a table is occupied, so there is less confusion between employees. This will save time and is an easy way for waiters and bussers to communicate. We would also want a chat system in place to help communicate amongst employees.

Managers:

Problem 1: Organizing inventory and managing how much to purchase to minimize waste

In restaurants managing inventory is usually a difficult task. This is a long and tedious task which can be optimized for efficiency and speed. Supplies can also have expiration dates that can lead to heavy consequences when neglected. Restaurants also have problems where they have too little of an item, and have to refuse a customer an order or have to change the recipe. This creates an issue for the chef, as well as for the manager, and can lower customer satisfaction. By having statistics on the popularity of orders and by having the inventory being updated after each order, our managers can have an estimate on what supplies they need more or less of. This system will minimize waste and can lower the risk of having an unsatisfied customer. This system will also be able to track if inventory items are expired need to be replaced or refilled. Our system will have the ability to automatically suggests orders to be made which managers can then decide whether to make those orders or not. This saves the manager's time and makes inventory management much more efficient and accurate.

Problem 2: Scheduling employees can be difficult with multiple preferences and availabilities

One of the most common issues that restaurants face is scheduling. Making employee schedules can be hard and often time are not as efficient. Managing employee availability can be very tedious and can often lead to conflict, last minute changes to work schedule, and understaffed shifts. By tracking the restaurant's activity, our managers can determine the proper number of employees needed for certain times. With this system, employees clock in using a web clocking system. Our managers can also confirm, deny, or give availability. This system will also allow our managers to monitor any employee's availability to work so the manager can build the work schedule accordingly. This will reduce the possibility of having understaffed shifts, which results in a better overall customer experience. Our employees can also see a detailed page on their schedule for the future and past, hours logged, current pay check, and how much they will receive on a certain day.

Problem 3: Managing finances can be optimized and made easier for managers

Our managers are typically given the responsibility of manually handling the finances of their respective restaurants and must make important financial decisions based on the financial health of their restaurant. This process can be cumbersome and overwhelming for some managers as the task of managing finances for any organization can become a huge problem without the help of software resources. By creating a financial management system where all income and costs incurred by the restaurant are automatically tracked, the manager's job of optimizing the use of restaurant funds will become significantly easier. The system will allow our manager to view profit projections, projected payroll costs, and keep track of restaurant transactions, such as the purchase of inventory items.

Glossary of Terms

Technical Terms:

Alert - notify or warn waiter of customer's request of assistance or attention

Algorithm - a process or set of rules to be followed in calculations or problem solving operation by a computer

Automate - technique of making a process, system, or apparatus operate without the use of human labor

Database - a structured set of data held in a computer, especially one that is accessible in various ways

Mobile Application - type of application software designed to run on a mobile device, such as a smartphone or tablet computer

NFC - (near-field communication) set of communication protocols that enable two electronic devices, one of which is usually a portable device such as a smartphone, to establish communication by bringing them within 4 cm of each other

System - a coordinated body of methods or a scheme or plan of procedure.

Tablet - wireless, portable personal computer with a touchscreen interface, smaller than a notebook computer but larger than a smartphone

Non Technical Terms:

Availability - table's status of being ready to seat new customers

Bill - recorded account of amount of money owed for food ordered

Chef - person who is responsible for preparing food (or overseeing individuals who are preparing food)

Chef portal - A site that can be accessed by chef

Cryptocurrency - A digital currency and a form of payment

Costs - the price paid to acquire, produce, accomplish, or maintain anything

Customer - any person entering the establishment with intentions of ordering food or drinks

Employee Portal - Schedule accessible by employees that lists their shifts and any open shifts they may cover.

Finances - the monetary resources of the restaurant.

Funds - money immediately available.

Gift Card – A prepaid money card for the restaurant

Inventory - a complete listing of stock on hand, such as raw materials and finished goods.

Manager - a person who has control or direction of the restaurant

Notes - comments places by the customer pertaining to the specific preparation or ingredients of a dish

Order - the food the customer requests.

Organization - the administrative personnel or apparatus of a business

PayPal - electronic commerce company that facilitates payments between parties through online funds transfers

Payroll - a list of employees to be paid, with the amount due to each

Prioritize - designate or treat (something) as more important than other things based on relation to time or urgency of a situation

Process - a systematic series of actions directed to some end

Profit - the difference between the amount earned and the amount spent in buying, operating, and running a business resulting in a financial gain

Reservation - placing a request to hold a table for a specific group or persons by requesting a head of time

Restaurant - an establishment where meals are served to customers

Scheduling - the process of assigning shifts to employees

Split the Bill - the act of breaking up the check into separate checks depending on each customer's individual order with the intention for each customer to pay separately

Software - the programs used to direct the operation of a computer, as well as documentation giving instructions on how to use them.

Statistics - the numerical facts or data themselves.

Table Availability Schedule - A database that shows the availability of the tables in the restaurant.

Takeout – Food that is ordered and consumed outside of the establishment

Tip - leaving gratuity of 15% - 20% of the check as a reflection of adequate service provided by the waiter

Traditional Restaurant - refers to how restaurants used to be run before technology took over

Traffic - refers to high volume of people or orders arriving in a short time span

Transactions - the exchange of money between two consenting persons; i.e. customer and employee.

Waiter - a man/woman whose job is to serve customers at their tables in a restaurant

User Stories

Customer:

Identifier	User story	Size
ST-C-1	I can order dishes at my pace without having to wait for a waiter to take my order each time	2
ST-C-2	Based on the size of my party I can reserve table(s) ahead of time and check their availability	4
ST-C-3	I can add/drop ingredients/allergens by my choice while ordering	4
ST-C-4	I can rate the ambience and service according to my satisfaction	2
ST-C-5	I can split the bill among our group and can tip if the service was worth it.	3

General Employee:

IDENTIFIER	USER STORY	SIZE
ST-E-1	I should be able to see when my shifts are for the month	5
ST-E-2	I should be able to give preferences for shifts	4
ST-E-3	I should be able to clock in and out for my shifts	4
ST-E-4	I can create reservations for customers	2

Waiter/Waitress:

IDENTIFIER	USER STORY	SIZE
ST-W-1	I can see which tables are mine	4
ST-W-2	I can see when a customer needs assistance	5
ST-W-3	I can mark tables are being used	5
ST-W-4	I can mark when customers leave the tables and are dirty	4
ST-W-5	I can see when a customer leaves a tip	2

Chef

IDENTIFIER	USER STORY	SIZE
ST-CH-1	As a chef, I have a chef portal that I can log into.	2
ST-CH-2	As a chef, I can see (in the chef portal) the orders customers placed in the order I need to make them.	4
ST-CH-3	As a chef, I can see the special requests customers make about each order so I can modify the order to meet the customer's request.	2

Busser

IDENTIFIER	USER STORY	SIZE
ST-B-1	I should be able to mark when tables are clean	4
ST-B-2	I can see when a table is dirty and when it is vacated	3

Manager

IDENTIFIER	USER STORY	SIZE
RT-M-1	I want to see my inventory to see if i am running low on items and see what I need to order	5
RT-M-2	I want to build a schedule to see when my employees will be working	4
RT-M-3	I want to see if items are expired to prevent the kitchen from using those items and use fresher ingredients	3
RT-M-4	I want to be able to change the rate of pay for any individual employee	3
RT-M-5	I want to be able to change the prices of the items on the menu	3
RT-M-6	I want to be able to view the popularity of each item on the restaurant's menu	4
RT-M-7	I want to be able to see my funds to determine my financial situation	5

System Requirements

Enumerated Functional Requirements

Identifier	User story	Size
ST-C-1	I can order dishes at my pace without having to wait for a waiter to take my order each time	2
ST-C-2	Based on the size of my party I can reserve table(s) ahead of time and check their availability	4
ST-C-3	I can add/drop ingredients/allergens by my choice while ordering	4
ST-C-4	I can rate the ambience and service according to my satisfaction	2
ST-C-5	I can split the bill among our group and can tip if the service was worth it.	3
ST-C-6	I can order for a takeout option	2
ST-C-7	I can play games on the customer tablet	1
ST-C-8	I can pay using credit cards, Pay Pal, Cryptocurrency, Gift Cards, or cash	2
ST-E-1	I can log and log out of my portal	2
ST-E-2	I can view tables that need to be cleaned and mark when they are cleaned	4
ST-E-3	I can see when tables need assistance from a waiter	5
ST-E-4	I can see when an order comes and and in what order people placed their orders.	6
ST-E-5	I can input my preferred schedule and can see when I am scheduled	5
ST-M-1	As a manager, I want to see my inventory to see if i am running low on items and see what I need to order	2
ST-M-2	As a manager, I want to build a schedule to see when my employees will be working	3
ST-M-3	As a manager, I want to see if items are expired to prevent the kitchen from using those items and use fresher ingredients	7
ST-M-4	As a manager, I want to be able to change the rate of pay for any individual employee	5
ST-M-5	As a manager, I want to be able to change the prices of the items on the menu	8

ST-M-6	As a manager, I want to be able to view the popularity of each item on the restaurant's menu	6
ST-M-7	As a manager, I want to be able to see my funds to determine my financial situation	1
RT-M-8	As a manager, I want to be able to monitor floor traffic at my restaurant	4

Enumerated Non Functional Requirements

Functionality: This software can be increased in size and used across many different size platform restaurants. This system allows for customers to be able to make reservations and seating's according to their preference. It also allows employees to make reservations for customers if they so choose. We also have created the ability for employees to adjust scheduling based on their availability. We also have allowed for a integrated user experience across the customer, employee, and manager segments. All are intertwined and rely on each other. The employees can see when an order went in and the order in which it was placed. It also maximizes efficiency by allowing bussers to mark when tables are clean and waiters to mark when tables need to be cleaned.

Usability: We have designed this system for 10 tables, five Waiters, two Chefs, two Bussers, one Manager. This will be able to be used for chain restaurants such as TGI Fridays, On The Border, Applebee's, etc. We will also be providing the users access to a training software, which if they need assistance on how to use our system, they can use the training system.

Reliability: We created this system so that restaurants can be more efficient and focus on the customer satisfaction. If a system is down, then it can lower customer satisfaction and cause harm to the restaurant. In order to mediate this potential issue, we will be providing 24-hour customer support, dedicated to helping the restaurants with any potential problems. We also provide technicians that can come on site to help manage the software and help manage updates.

Performance: This system will run in real time with a possible three seconds delay. The response time for our system will be nearly close to immediate, with the possibility of slight delay.

Supportability: This system will be monitored using GitHub. Any necessary changes can be done using GitHub. We can manage updates, problems, and can see who made what changes. Our system will be compatible with any tablets with access to the internet. This application will be used on primarily tablets.

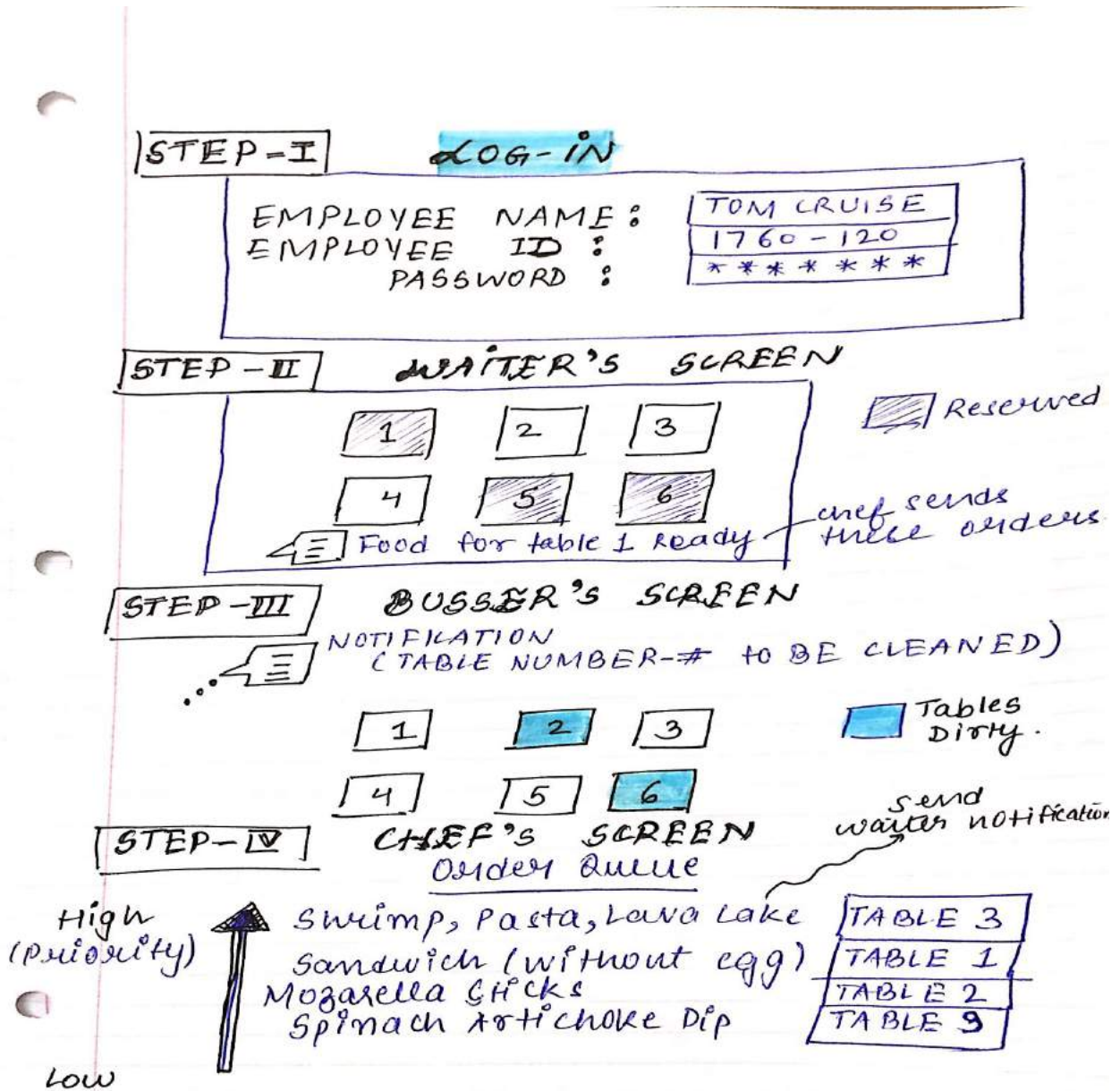
On-Screen Appearance

Customer: The customers will have the ability to see open tables, and reserve tables for their preferred time. This interface also provides the customer the ability to call a waiter for assistance. The customers can also look at the menu and make their selections and pay at the end of their meal. Finally, the customers can pay their bill and leave reviews to help improve the restaurant.

The wireframe illustrates a 10-screen user interface for a restaurant system:

- (1) Welcome:** Fields for "Name:" and "No. OF People:".
- (2) Seating Area:** A grid of 6 numbered boxes (1-6). Below the grid are checkboxes for "Reserve", "Reserved", and "Open".
- (3) Thank you:** Displays "Zp: 63749".
- (4) Please choose From:** A menu grid with categories: "Appetizers", "Salad & Soup", "Pasta Entrees", "Kids Food", "Drinks", and "Desserts".
- (5) Pasta Entrees:** Lists "Lasagna" for "\$8.00".
- (6) Lasagna:** Includes a "Description:" field, an "Enter Custom Comments" field, and an "Order" button.
- (6) Total:** Shows "Total: \$8.00" with buttons for "Place order" and "Add another item".
- (7) Billing Amount:** A detailed bill showing "Lasagna \$18.00", "tax \$0.00", "Enjoyed your service", and a total of "\$18.4" after adding a 15% tip.
- (8) Payment Method:** Radio button options for "Credit", "Pay Pal", and "Cash".
- (9) Would you like to rate:** Two sections for rating "Food?" and "Service?", each with a row of five stars.
- (10) Thank you For Visiting!**

Employees: This page shows how waiters will see tables and be able to mark when a table is dirty. It also shows which tables are reserved. It will also show a user's log in information. Finally, our system incorporates a chef's portal, showing the order in which customers orders come in.



STEP-V SCHEDULE

* can only view -

Time Spots	Mon	Tue	Wed	Thur	Fri	Sat	Sun
8:00 - 10:00 am	TOM	C	B	A	Z	Y	-
10:00 - 12:00 pm	T	-	W	-	-	-	-
12:00 - 2:00 pm	-	R	-	P	-	-	S
:	R	-	U	V	-	-	-
8:00 - 10:00 pm	-	-	-	-	J	M	N

STEP-VI CLOCK-IN / CLOCK OUT

PUNCH TYPE :-

clock in	v
clock out	
meal	
clock in	

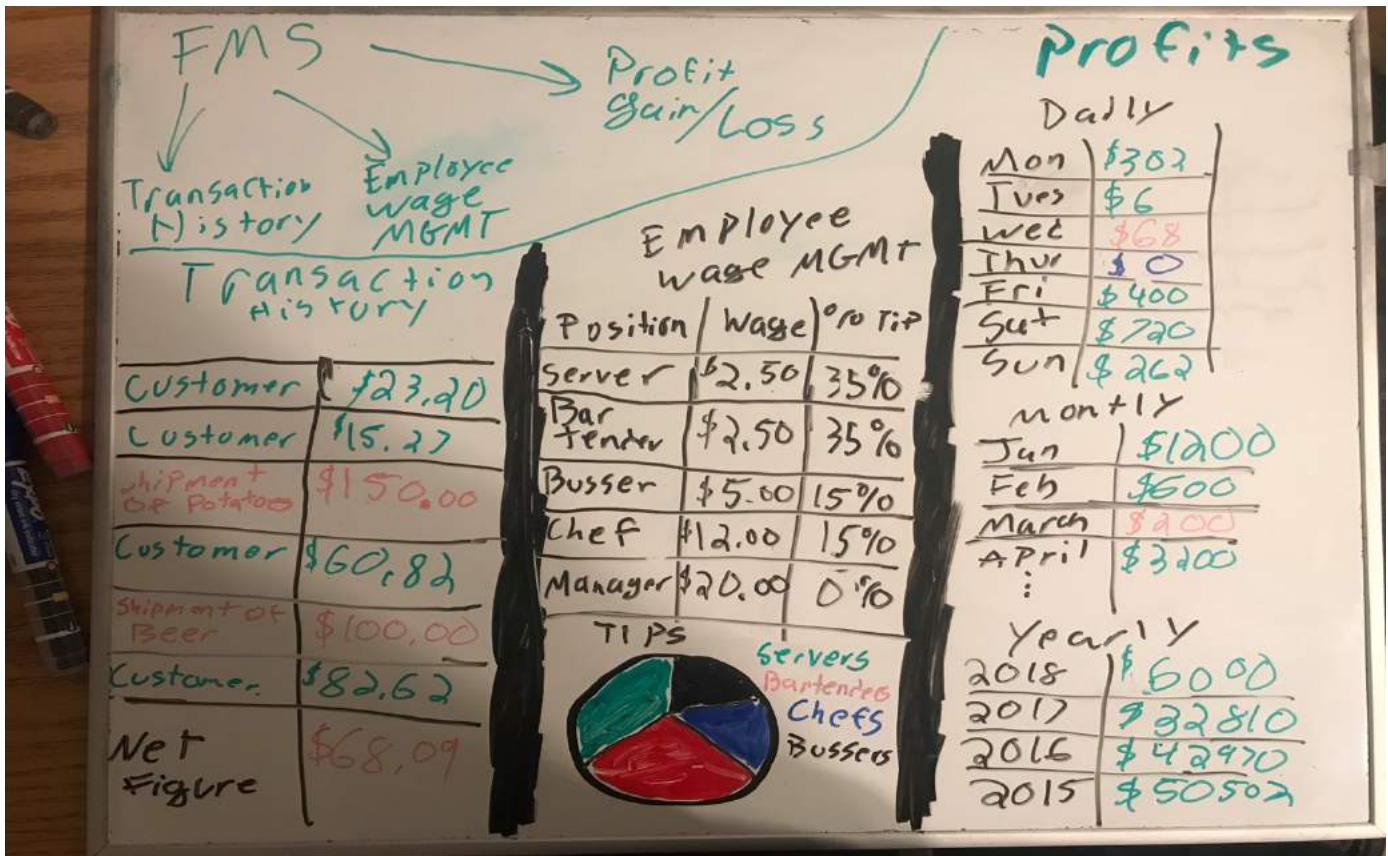
enter punch

LITTLE BITS

Manager: Here is how the scheduling would look once preferences are inputted. You also have an inventory showing what items are there, when they expire, and how much you have. Finally, there is a Financial description, giving various details on different financial statements.

Staff	Mon 18	Tue 19	Wed 20	Thu 21	Fri 22	Sat 23	Sun 24
Employee 1	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 2							
Position							
Employee 3		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Supervisor		Supervisor			
Employee 4	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 5	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	
Employee 6		3:00 AM - 7:00 AM					
Position		Trainee					
Employee 7	3:00 PM - 7:00 PM	7:00 AM - 3:00 PM		3:00 PM - 7:00 PM		7:00 AM - 3:00 PM	
Position	Manager	Manager		Manager		Manager	
Employee 8	3:00 AM - 7:00 AM				7:00 AM - 3:00 PM		
Position	Sales Associate				Sales Associate		
Employee 9		3:00 AM - 7:00 AM	3:00 PM - 7:00 PM	3:00 PM - 7:00 PM		11:00 AM - 3:00 PM	
Position		Cashier	Cashier	Cashier		Cashier	
Employee 10							
Position							
Employee 11	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 12							
Position							
Employee 13		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Sales Associate		Sales Associate			
Employee 14	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 15	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	

Item	Quantity	Expiration date	Date Added
△ Tomatoes	10 lbs	2/10	1/31
Salt	8 lbs	N/A	1/30
△ Cheese	1 lb	2/20	2/5
△△△ Onions	0 lbs	N/A	N/A
△△△ Potatoes	3 lbs	2/7	1/27
△△ Eggs	<1 Dozen	2/10	2/1



Functional Requirements Specification

Stakeholders:

There are a multitude of stakeholders that are a major part of our team. The first set of stakeholders that are directly involved are the users. The users include; customers, waiters, bussers, chefs, and managers. Customers do not have access to the portal that employees have, rather they will have access to an interface which allows them to place orders, notify waiters, and see different information regarding their order. They are not direct users, however the customer experience and satisfaction is the highest priority of our team. Employees will have a separate portal that contains their job specific functions. Another set of stakeholders will be the technicians. Technicians interests involve the maintenance of the system as well as updates with the interface. Another group of stakeholders will be the administration, focused on the promotion of our interface. Finally, we have a customer service team. This teams interest lies in helping the restaurants using our system with basic technical problems that can be fixed over the phone.

Actors and Goals:

Initiating Actors

Bussers:

Role - The employee who is in charge of the cleanliness of the restaurant.

Goal - Manage the status of tables, and clean tables that are marked dirty on the app.

Chef:

Role - The employee who is in charge of the kitchen and preparing meals.

Goal - Manage the ingredient inventory, manage the menu, and prepare meals on the order queue.

Customer:

Role - The restaurant visitor who orders food and drink from the restaurant for dining-in or taking-out.

Goal - Place, pay for, and receive orders quickly, and choose whether to eat-in or take-out.

Manager:

Role - The employee who manages the entire restaurant.

Goal - Manage employees and scheduling, keep track of inventory, and keep track of profits/losses.

Participating actors

Waiter:

Role – The restaurant employee that handles customer interactions such as adjusting bill, getting table set, and bringing food

Goal – To bring out food in a timely manner once it is done, to make sure tables are empty and ready for use, and to handle any customer needs

MySQL Database:

Role - The object that stores information needed for the system.

Goal - Store and modify information pertaining to inventory, employees, profits, losses etc..

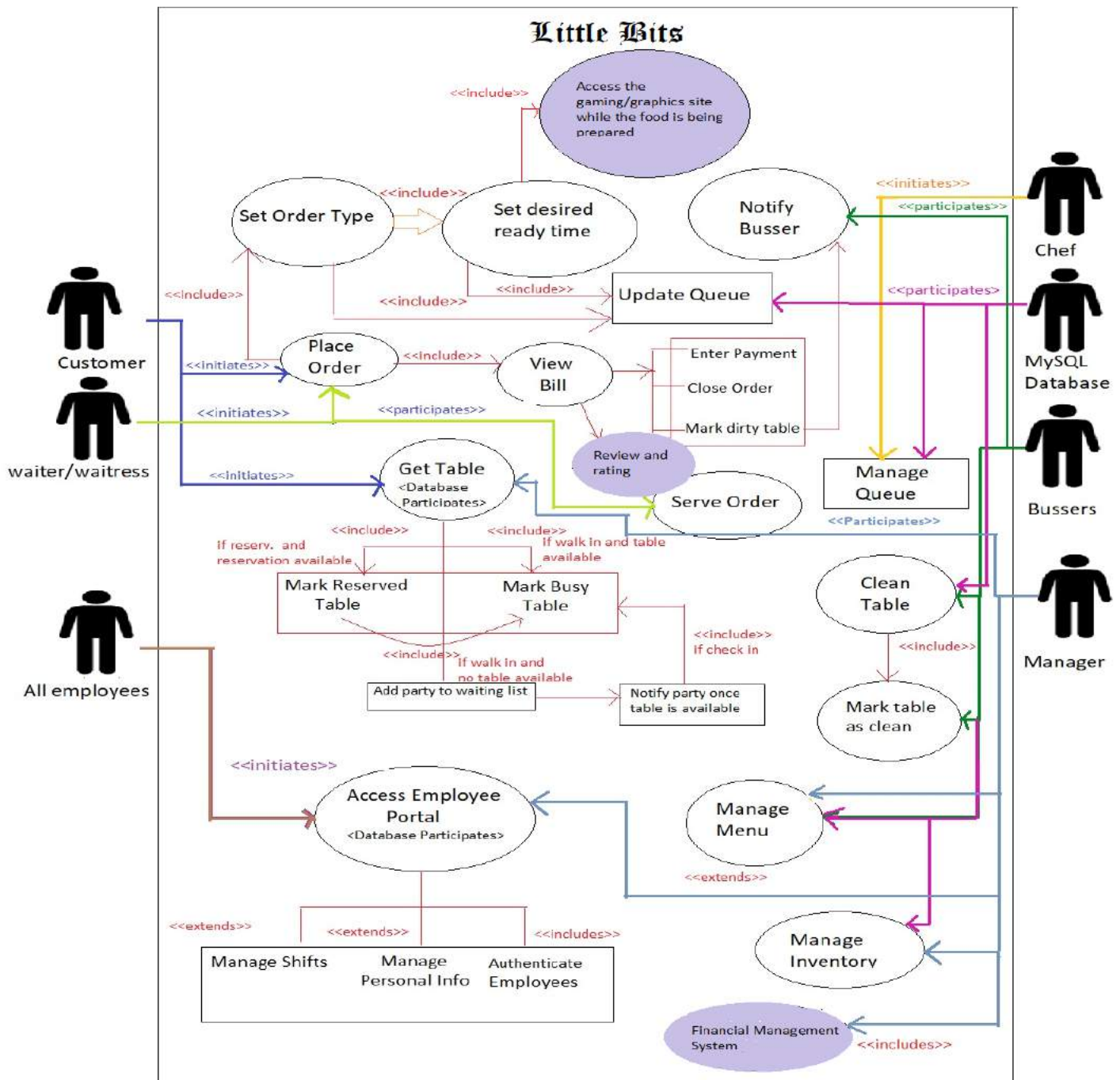
Use Cases

Casual Description

Our team's user stories will serve as the casual descriptions for the use cases.
Included is a weighting table to show level of effort and difficulty.

<u>User Case</u>	<u>Use Case Weight</u>
<u>UC-1</u>	<u>6</u>
<u>UC-2</u>	<u>8</u>
<u>UC-3</u>	<u>8</u>
<u>UC-4</u>	<u>6</u>
<u>UC-5</u>	<u>7</u>
<u>UC-6</u>	<u>8</u>
<u>UC-7</u>	<u>8</u>

Use Case Diagram



Traceability Diagram

User stories	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7
ST-C-1							
ST-C-2							
ST-C-3							
ST-C-4							
ST-C-5							
ST-C-6							
ST-C-7							
ST-C-8							
ST-E-1							
ST-E-2							
ST-E-3							
ST-E-4							
ST-W-1							
ST-W-2							
ST-W-3							

LITTLE BITS

ST-W-4							
ST-W-5							
ST-CH-1							
ST-CH-2							
ST-CH-3							
ST-B-1							
ST-B-2							
RT-M-1							
RT-M-2							
RT-M-3							
RT-M-4							
RT-M-5							
RT-M-6							
RT-M-7							
RT-M-8							

Fully-Dressed Description

Use Case UC-1: Managing Orders

Related Stories: ST-C-1, ST-C-6, ST-CH-2

Initiating Actor: MySQL Database

Actor's Goal: Orders placed and sent to the chef's tablet and the database are then queued and an algorithm then calculates estimated time for the order to be ready and displays it on to the customer's tablet.

Participating Actors: Customer, Chef

Preconditions:

- The restaurant menu is visually displayed on the customer's tablet
- Incoming orders will be displayed visually on the chef's tablet

Postconditions:

- The customer will be able to see when the order is completed

Flow of Events for Main Success Scenario:

→An order is placed by a customer and is sent to the database

←The database registers an order has been made and places the order in the order queue

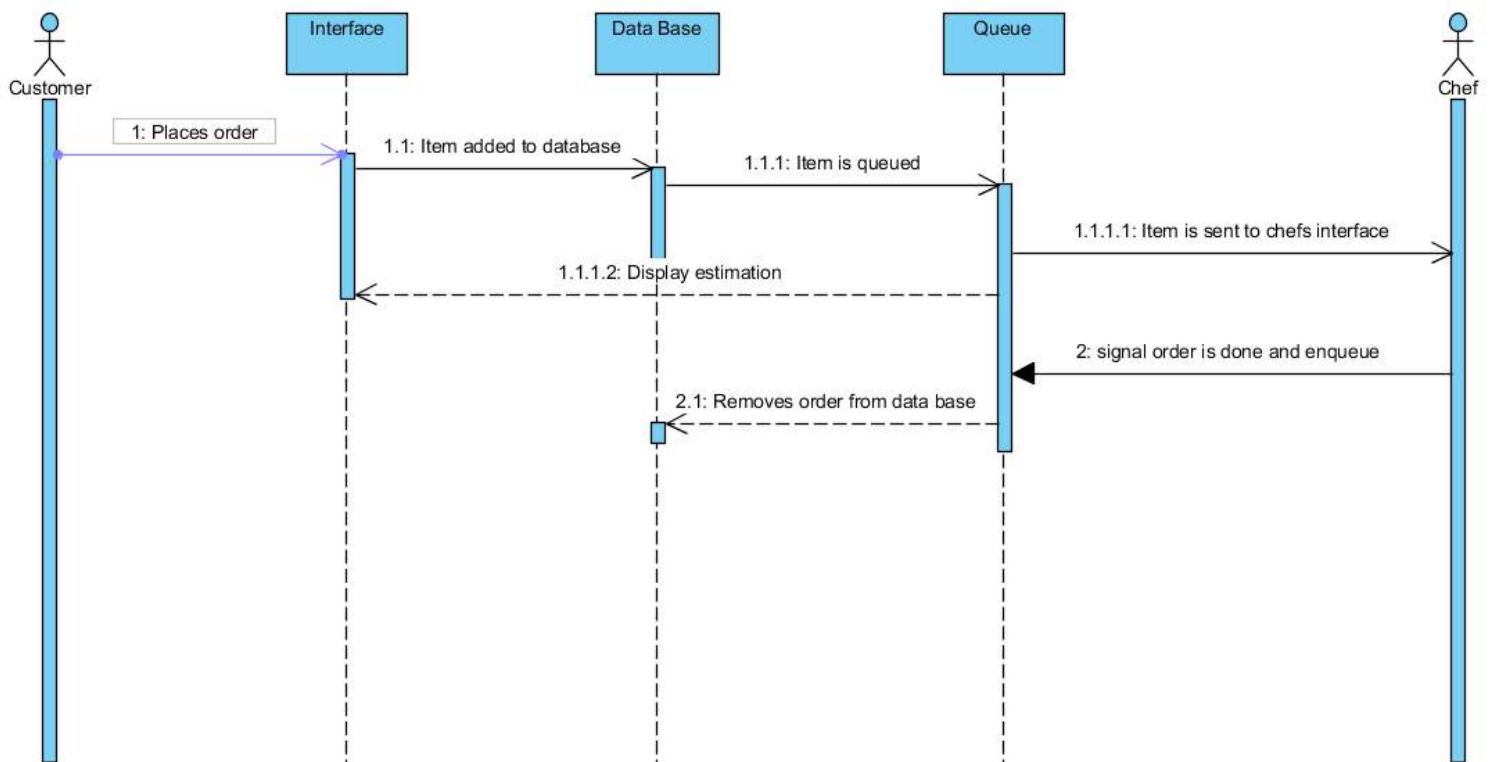
→The database displays the next order in the queue to the chef tablet for preparation

←An algorithm calculates an estimated time for preparation based on the orders queued and displays the calculated time to the customer's display

←Chef signals that the order is done and is removed from the queue

→Order gets removed from database

LITTLE BITS



Use Case UC-2: Traffic Monitoring

Related Stories: ST-C-2, ST-E-4, ST-W-1, ST-W-3, ST-W-4, ST-B-1

Initiating Actor: Customer

Actor's Goal: Have the ability to place reservations and place orders

Participating Actors: Waiter, Chef, Busser, Manager

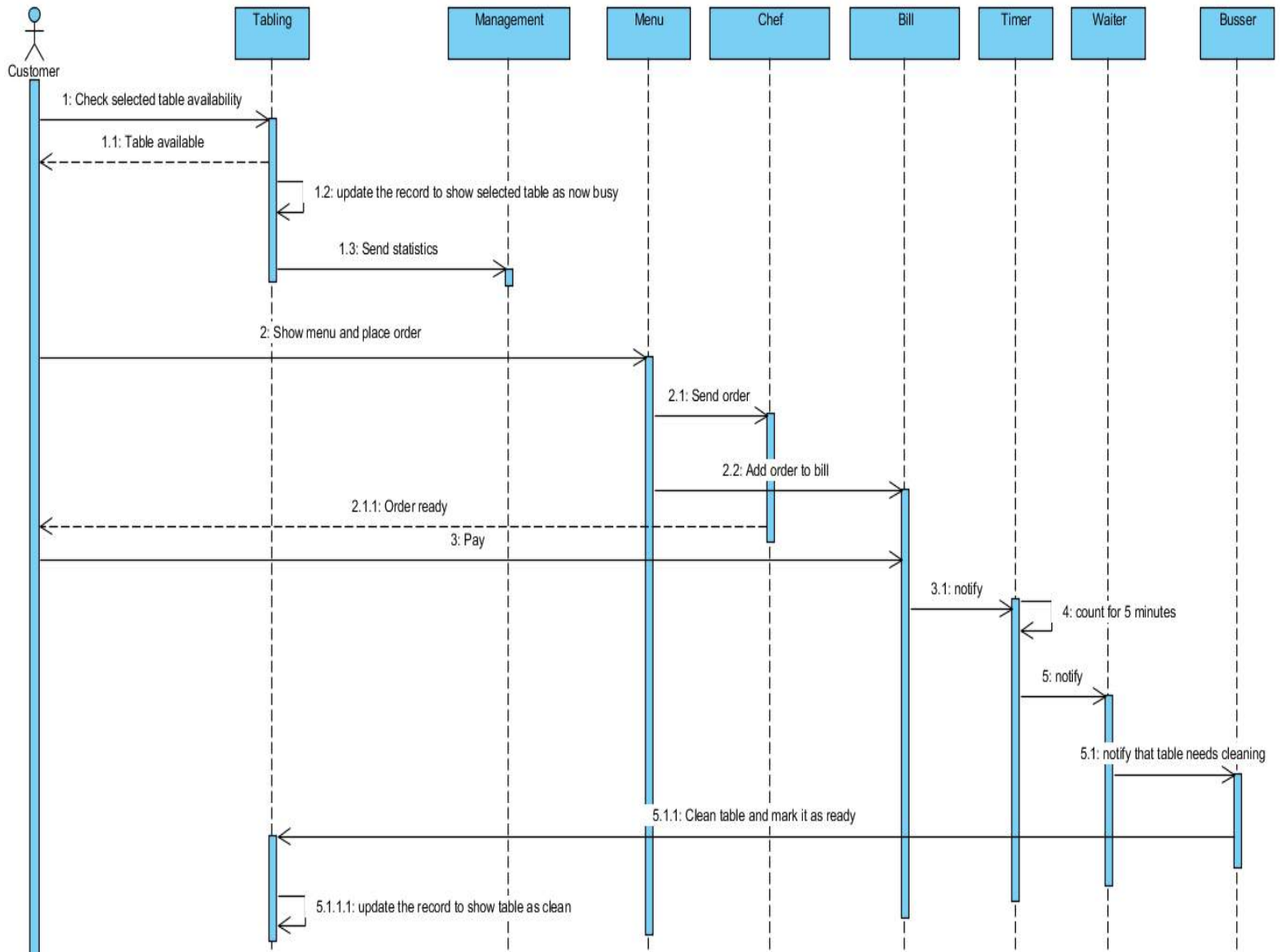
Preconditions: The user creates a reservation, come to the restaurant, and places an order

Postconditions: The waiter can see when a table is done and bussers can mark when tables are cleaned.

Flow of Events for Main Success Scenario:

- The customer goes online and selects a table and creates a reservation
- ←The system alerts the restaurant that this table needs to be reserved
- ←The system updates itself to show the table as now reserved.
- ←The system adjusts the algorithm that monitors traffic and gives statistics to managers.
- The customer shows up and sees menu and places order
- The chef clicks that the order is finished
- ←The system notifies the customer that the order is done, and the waiter is notified to bring the order to the table
- The customers finish the meal and pay their bill and then proceeds to leave the restaurant
- ←The timer counts for 5 minutes
- ←The timer notifies the waiter to check if the table is vacant
- The waiter marks that the table is vacated and dirty
- ←The busser is notified that a table is dirty and needs to be cleaned
- The busser cleans the table, and marks that the table is ready to be used
- ←The system is updated with an open table that can be viewed in the table availability

LITTLE BITS



Use Case UC-3: Customer Ordering

Related Stories: ST-C-1, ST-C-3, ST-C-5, ST-C-6, ST-CH-2

Initiating Actor: Customer

Actor's Goal: The ability to view the menu, ordered desired food and pay the bill

Participating Actors: Waiter, Chef and Manager

Preconditions: A tablet open on the customer interface

Postconditions:

- The customer can make the order
- The customer can pay the bill after consumption.

Flow of Events for Main Success Scenario:

→The customer shows up or navigates to the site and sees menu and places order

→ Customer picks the option to see the menu in the application.

← The system shows all the food and drink items in their respective categories, as well as the current order's bill on the right side.

→Customer chooses which food category they would like to add to the order.

←The system displays all the items associated with the picked category and an option to add to order.

→Customer selects the option to add the desired item to the order.

←The bill is updated and the customer has the option to place order or add new item

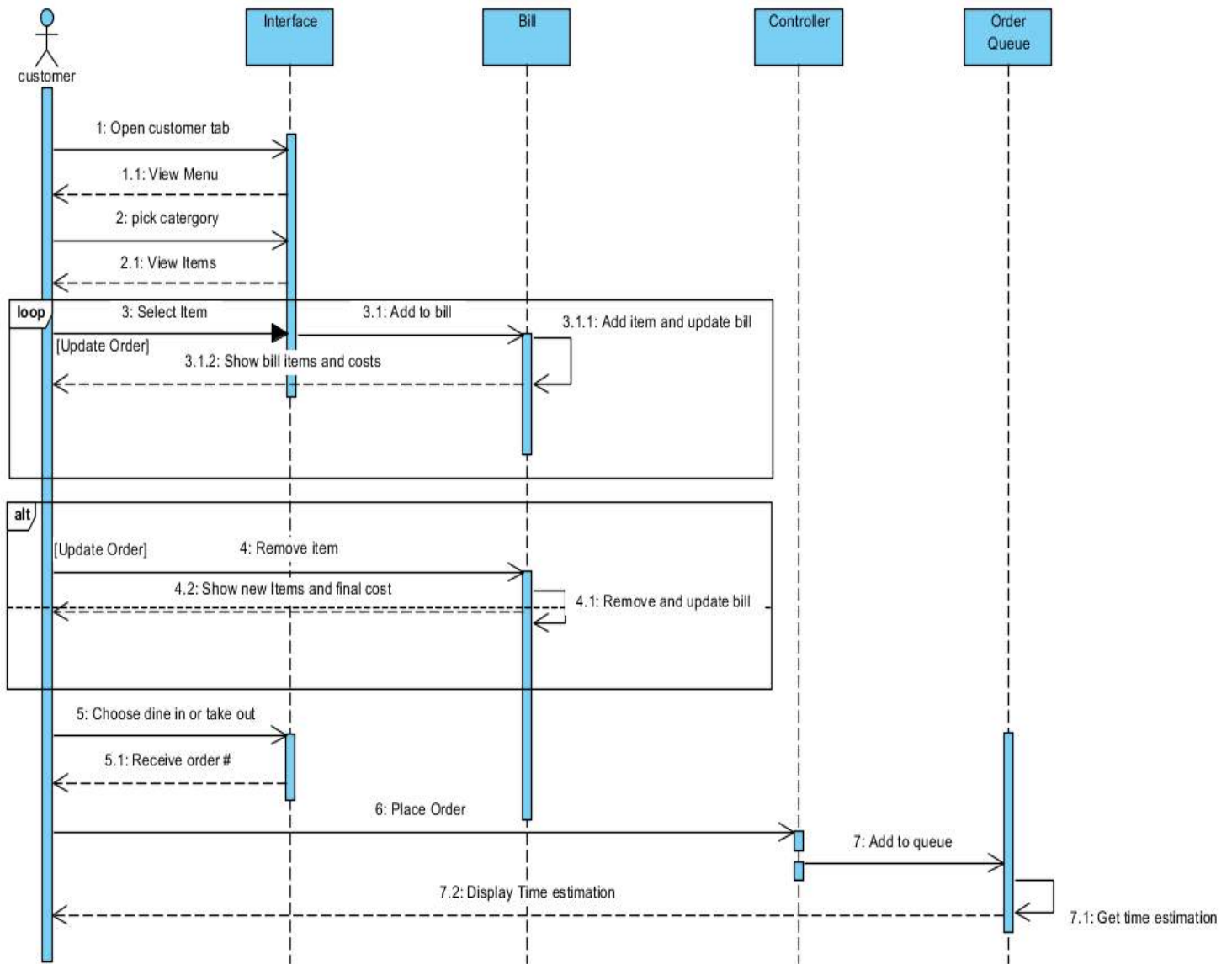
→Customer can choose take-out option

←If customer chooses takeout, the system shows order number

←The bill is updated and the order is sent to the chef's tablet in the order it was placed in the restaurant

←An estimation of the time to be ready is given to the customer

LITTLE BITS



Use Case UC-4: Payment Process

Related Stories: ST-C-5, ST-W-5

Initiating Actor: Customer

Actor's Goal: To pay their bill successfully

Participating Actors: Waiter and Customer

Preconditions: The Customer has clicked and opened the Pay Your Bill tab and has chosen one of the methods of payment.

Postconditions: The customer pays the bill with or without the Waiter's assistance depending on which option the customer chooses.

Flow of Events for Main Success Scenario:

→ The customer chooses the mode of payment of his bill (i.e Credit Card, PayPal, Gift Cards, Cryptocurrency, and Cash).

← The system opens up the tab according to the decision made by the customer.

→ The customer can either choose to split the bill or to pay all of it alone.

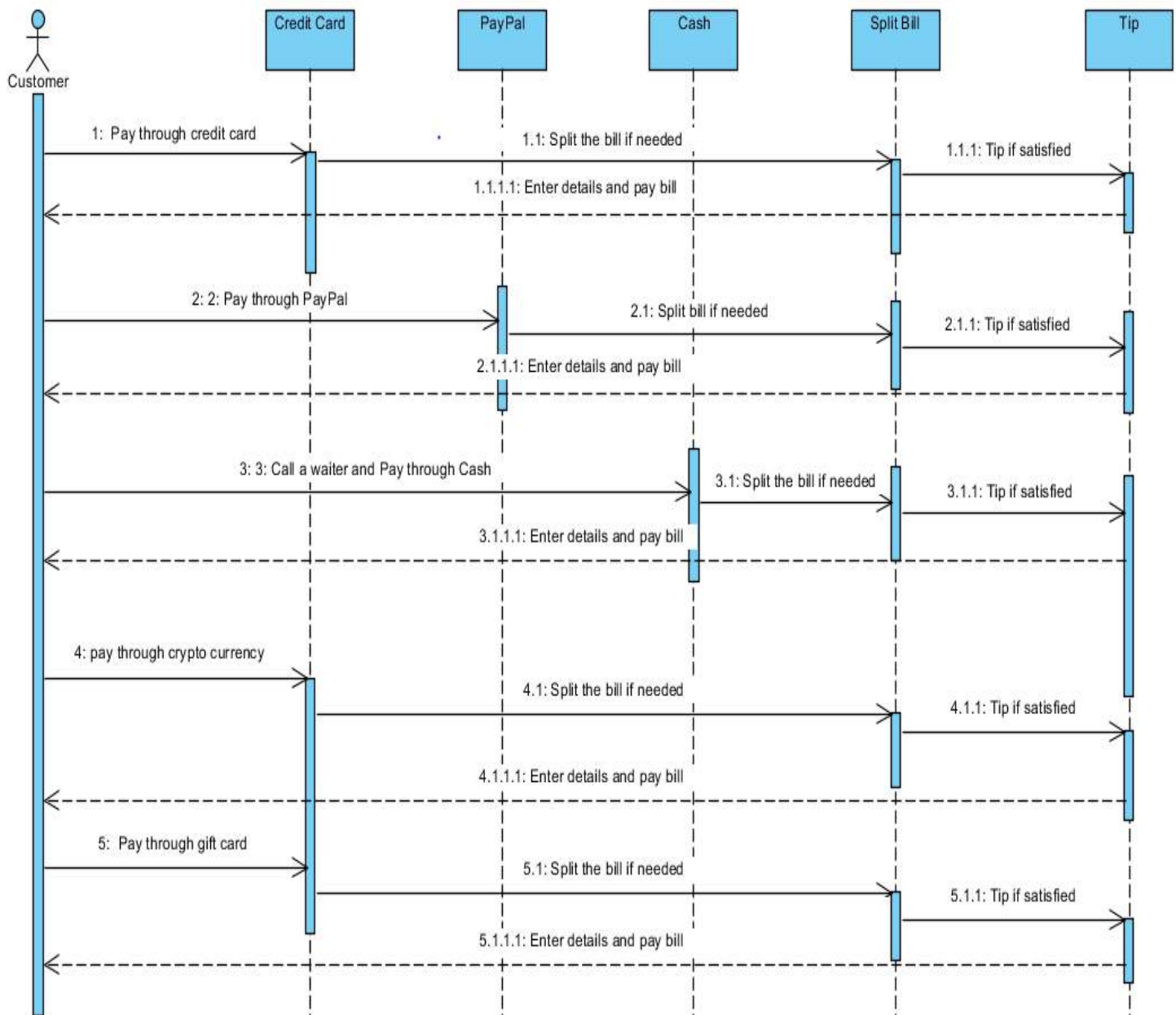
→ Based on the bill amount, the system notifies the waiter tipping amount to the customer.

→ The customer makes a decision whether to tip or not.

← If the customer chooses to tip, the billing amount plus the tipping amount gets credited to the restaurant. If not, the restaurant receives the bill.

← The system opens the review screen after notifying the customer that his payment was made.

LITTLE BITS



Use Case UC-5: Managing Employees

Related Stories: ST-E-1, ST-CH-1, ST-M-2, ST-M-4

Initiating Actor: General Employee

Actor's Goal: Have the ability to see their portal for their specific page which gives them their interface.

Participating Actors: General Employees

Preconditions: The user logs in which automatically clocks them in.

Postconditions:

- The user saves his preferences
- The user is automatically clocked out.

Flow of Events for Main Success Scenario:

→The employee logs in to the system

←The system automatically clocks in the employee and opens the portal depending on their position

→The employee then can see his or her schedule and can mark preferences for time availability

←The system takes into account all the time preferences and creates a list of times for the manager to see

→The manager then makes the schedule and posts it for employees to see

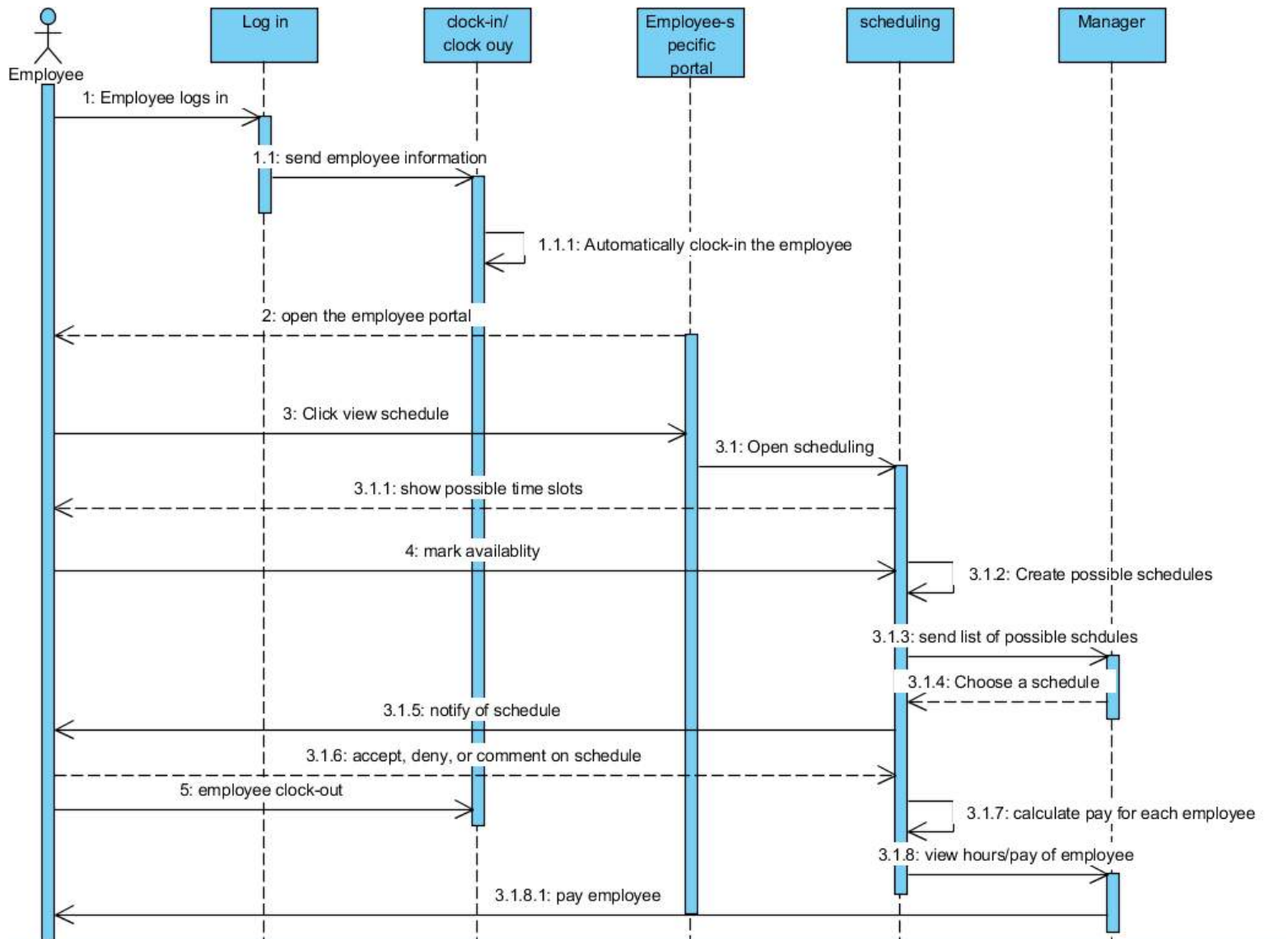
←Employees can then accept, deny, or comment if they can work or if the schedule is correct

→The employee clocks out

←The manager can see the hours and pay for the employees, and has the ability to adjust the pay

←The system calculates the pay stubs for employees to see based on logged hours

LITTLE BITS



Use Case UC-6: Managing Restaurant

Related Stories: ST-M-1, ST-M-3, ST-M-5, ST-M-6, ST-M-7, ST-M-8

Initiating Actor: Manager

Actor's Goal: To have control over the restaurant and its employees. Managers need to know the inventory and the financial situation of the restaurant to and adjust menu items and pricing as well as employee pay accordingly. Managers must also be aware of customer traffic.

Participating Actors: Database

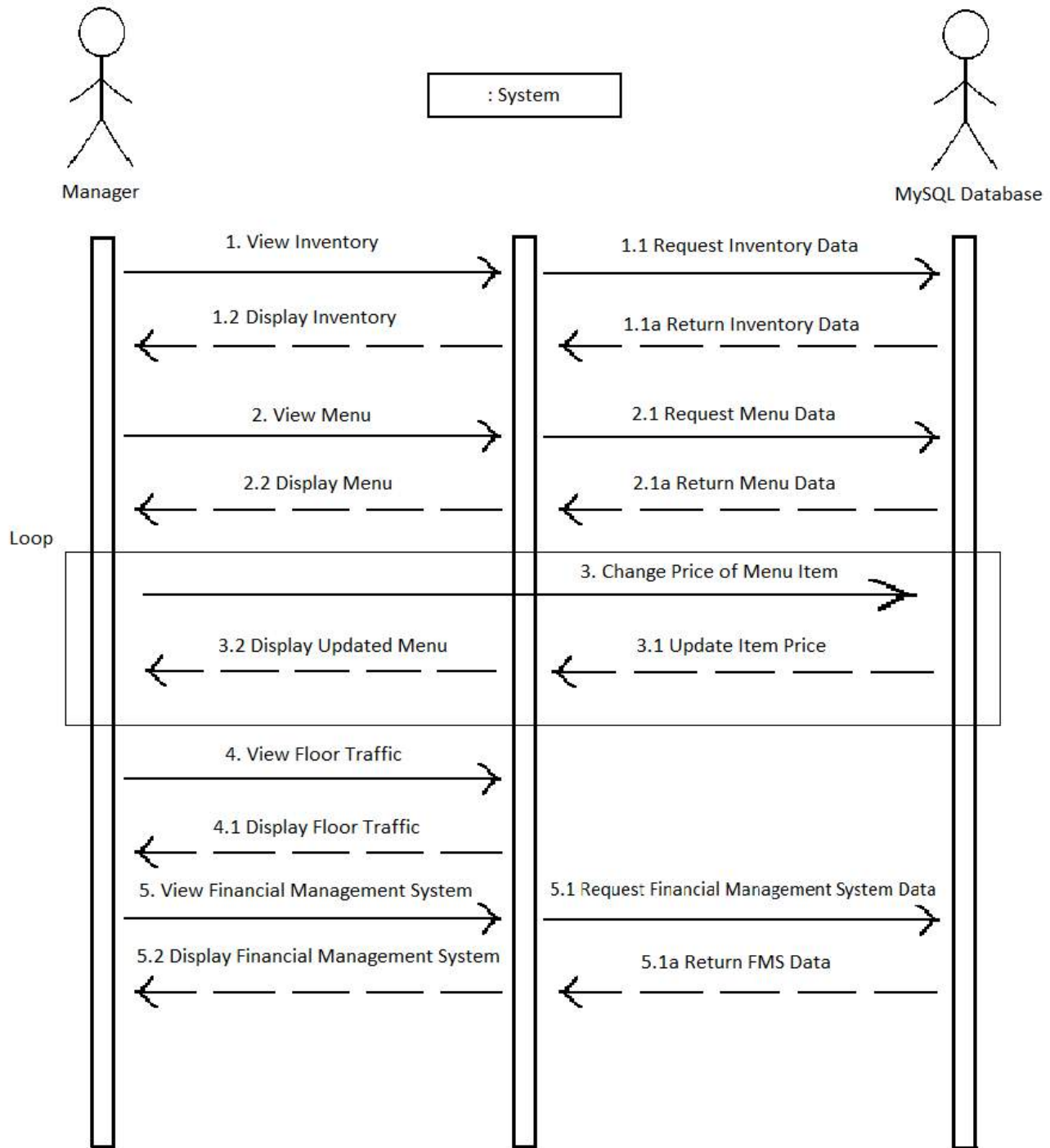
Preconditions: Manager opens portal and sees all the available options

Postconditions:

- The manager looks at the systems recommendations for inventory
- The manager selects their preferences and can make adjustments

Flow of Events for Main Success Scenario:

- 1) Manager views inventory database to see that the restaurant is running low on a certain ingredient
- ← 2) The database indicates that the ingredient is still fresh, but the manager determines that the supply will not last the night
- 3) Manager increases the price of items on the menu that utilize said ingredient
- 4) Manager views popularity of menu items on the database that use that ingredient as well as floor activity to determine how quickly the supply will deplete
- 5) Manager marks appropriate items as unavailable once the restaurant runs out of that ingredient
- 6) Manager views and edits financial information that pertains to the restaurant



Use Case UC-7: Child Entertainment

Related Stories: ST-C-9

Initiating Actor: Customer

Actor's Goal: To play Uno or Piano tiles

Participating Actors: Customer

Preconditions: none

Postconditions: The customer finishes playing the game

Flow of Events for Main Success Scenario:

→ The customer chooses the tab “games”

← The system opens up the tab

→ The customer clicks on the name of game they want to play

→ The game starts

Effort Estimation

USING CLICKS

UC-1

UC-1 effort estimation is covered by other use case effort estimations.

UC-2

Scenario 1: User reserves a table:

Navigation:

1. Customer clicks the “Reserve” button.
2. Customer enters the number of people in their party.
3. Customer selects the desired table.
4. Customer clicks “ok”

Scenario 2: Chef Prepares Order:

Navigation:

1. Select the option to go to the queue.
2. Chef selects an order from the queue, the items within the order (including attached notes or ingredients alterations) are displayed
3. Chef selects “in progress” once the order preparation is about to begin
4. Chef selects “complete” once the order is ready for pick-up

Scenario 3: User pays the bill:

Navigation:

1. Click the “Pay” button
2. Click “Split the bill” if user desires to
3. Enter the number of people among whom the bill will be split
4. Enter the payment amount or percentage for payer
5. Click “Pay”

Scenario 4: Busboy Cleans Table:

Navigation:

1. Select the option to go to the table layout.
2. Busboy selects a table that is “ready for cleaning” and click on it once to change its status to “cleaning in progress”
3. Busboy clicks on it once more to change its status to “ready for seating” upon completion of cleaning

UC-3

Scenario 1: Ordering food

Navigation:

1. Customer selects desired food category
2. Customer selects desired food item
3. Customer then writes needed comments and places the order
4. Customer then chooses their desired payment process
5. Customer enters payment details or calls waiter for cash payments

UC-4

Scenario 1: Payment of bill through Credit Card

Navigation:

1. Customer selects the payment option as 'Credit Card'
2. Customer has a choice to 'split the bill' if needed.
3. Customer can either choose to 'tip the waiter' or not.

Scenario 2: Payment of bill through PayPal

Navigation:

1. Customer selects the payment option as 'PayPal'
2. Customer has a choice to 'split the bill' if needed.
3. Customer can either choose to 'tip the waiter' or not.

Scenario 3: Payment of bill through Cash

Navigation:

1. Customer chooses to pay the bill by 'Cash'.
2. The waiter arrives to collect the bill.
3. Customer has a choice to 'split the bill' if needed.
4. Customer can either choose to 'tip the waiter' or not.

Scenario 4: Payment of bill through Gift Cards

Navigation:

1. Customer chooses to pay the bill by 'Gift Card'.
2. Customer has a choice to 'split the bill' if needed.
3. Customer can either choose to 'tip the waiter' or not.

UC-5

Scenario 1: Submitting employee availability

Navigation:

1. Employee clicks “Login”
2. Employee enters username and password
3. Employee clicks “Login”
4. Employee clicks “View Schedule”
5. Employee clicks on the time slots they are available
6. Employee clicks “Submit Availability”

UC-6

Scenario 1: Viewing Inventory

Navigation:

1. Manager selects “Login”
2. Manager enters username and password
3. Manager selects “Login”
4. Manager selects “Inventory”
5. Manager selects “View Items”

Scenario 1.2: Add Item to Inventory

Navigation:

1. Manager selects “Login”
2. Manager enters username and password
3. Manager selects “Login”
4. Manager selects “Inventory”
5. Manager selects “Add Items”
6. Manager inputs data corresponding to that item and hits Add

Scenario 1.3: Delete item from Inventory

Navigation:

1. Manager selects “Login”
2. Manager enters username and password
3. Manager selects “Login”
4. Manager selects “Inventory”
5. Manager selects “Remove Items”
6. Manager selects item that is going to be removed

Scenario 2: View Profits

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Finances”
5. Manager selects “Profits”

Scenario 3: View Transaction History

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Finances”
5. Manager selects “Transaction History”

Scenario 4: View Employee Wages

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Finances”
5. Manager selects “Employee Wages”

Scenario 5: View Employee Schedule

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Scheduling”

Scenario 6: View Restaurant Traffic

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Restaurant Traffic”

Scenario 7: View Menu and Ratings

Navigation:

1. Manager clicks “Login”
2. Manager enters username and password
3. Manager clicks “Login”
4. Manager selects “Menu and Ratings”

USING CASE POINTS

Unadjusted Actor weight (UAW)

Actor Name	Description	Complexity	Weight
Busser	Interacts with system via a graphical user interface	Complex	3
Chef	Same as busser	Complex	3
Customer	Same as busser	Complex	3
Manager	Same as busser	Complex	3
Waiter	Same as busser	Complex	3
MySQL Database	System interacting through a protocol	Average	2

UAW: 17

Unadjusted Use Case Weights (UUCW)

Use Case	Description	Category	Weight
UC-1 Managing Orders	Basic processing. 2 participating actors(Customer, Chef) 6 steps for main success scenario	Average	10
UC-2 Traffic monitoring	Complex interface. 4 participating actors (Waiter, Chef, Manager, Busser) 14 steps for main success scenario	Complex	15
UC-3 Customer Ordering	Complex interface 3 participating actors (Waiter, Chef, Manager) 9 steps for main success scenario	Complex	15
UC-4 Payment Processing	Moderately Complex Interface/Processing 2 participating actors (Waiter, Customer) 7 steps for main success scenario	Average	10
UC-5 Managing Employees	Complex Interface 1 Participating Actor(General Employees) 9 Steps for main success scenario	Average	10
UC-6 Managing Restaurant	Complex interface with very complex processing (For finances) 1 Participating Actor (MySQL Database) 6 steps for main success scenario	Complex	15

UUCW: 75

Technical Complexity Factor (TCFs)

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Distributed systems connected to a single database	2	4	8
T2	Performance needs to be good	1	3	3
T3	End user efficiency must be good	1	3	3
T4	Internal processing is critical	1	5	5
T5	No need for reusability	1	0	0
T6	Installation will need to be done by a technician	0.5	1	0.5
T7	Ease of use is very important	0.5	4	2
T8	No portability concerns	2	0	0
T9	Minimal requirements for change	1	1	1
T10	Congruent use is needed	1	4	4
T11	Login feature for basic security	1	1	1
T12	Payment process and games utilize third party access	1	2	2
T13	Product comes with an interactive tutorial system	1	2	2

Total: 32

$$\text{TCF: } 0.6 + 0.01 * 32 = 0.92$$

Environmental Complexity Factor (ECF)

Environmental Factor	Description	Weight	Perceived Impact	Calculated Factor
E1	Strong Familiarity with UML Process	1.5	4	6
E2	Average experience with application problems	0.5	3	1.5
E3	Some Experience with object-oriented approach	1	2	2
E4	Each subgroup had an adequate lead analyst	1	3	3
E5	Highly Motivated	1	4	4
E6	Stable requirements critical	2	5	10
E7	No part time staff	-1	0	0
E8	Programming languages are complex	-1	4	-4

Total: 22.5

ECF: $1.4 - 0.03 * 22.5 = 0.725$ **Use Case Points:** $(75+17) * 0.92 * 0.725 = 61.364$ **Project Duration:**

Productivity factor: 28 hours

Duration = $28 * 61.364 = 1718.192$

Domain Analysis

Domain Model

Concept definitions:

Responsibility Description	Type	Concept
R-01: Customer accesses the menu on his tab to place order.	D	OrderItem
R-02: System sends customer's order to the restaurant.	K	Order
R-03: System knows and displays status of the order for customer viewing	K	OrderStatus
R-04: Customer accesses the bill payment options on his tab.	D	Check
R-05: System keeps track of entire table's bill and tip calculations.	K	Check
R-06: In case payment using cash is selected, the waiter returns to system to retrieve check	D	Check
R-07: System receives payment and notifies busboy to clean table.	D	CleanTable
R-08: Chef receives all the order in the order queue	D	Chef
R-09: Chef accesses the queue and modifies it according to the order status.	D	OrderQueue
R-10: Chef displays the order status for every customer	K	OrderDisplay
R-11: Chef notifies the waiter when an order is ready	K	Waiter
R-12: Order queue is updated after each order is prepared and ready to be placed on the customer's table	D	OrderStatus
R-13: Notify busser to clean the table after a customer has left/has paid.	K	CleanTable,Busser
R-14: Waiter handles the table reservation system and updates it	D	TableRecord
R-15: Coordinate actions of concepts associated with this use case and delegate the work to other concepts	D	Controller
R-16: Render the retrieved records into an HTML document for sending to the actor's Web browser for display	D	PageMaker

R-17: HTML document that shows the actor the current context, what actions can be done, and outcomes of the previous actions	K	InterfacePage
R-18: Change employee information including position, status, wage, contact information, etc.	D	InfoChanger
R-19: Notify Manager about different inventory alerts	D	Notifier
R-20: Prepare a database query that best matches the actor's search criteria and retrieve financial information, employee records, and inventory information from the database	D	Database Connection
R-21: List of Menu item ratings for further investigation and comment descriptions	K	Investigation Request
R-22: Display floor traffic of the restaurant	K	TrafficDisplay
R-23: Prediction of display of profit projections	D	Predictor

Association Definitions:

Concept Pair	Association Description	Association Name
Customer ↔ Order	Customer orders items and places order.	Conveys Requests/ Requests Save
Customer ↔ Rate	Customer rates food items and customer experience	Provides data
Customer ↔ Waiter	Customer requests waiter's assistance	Conveys Request
Check ↔ Customer	Customers receives data about their bill and gives payment options	Provides data
Chef ↔ OrderQueue	Chef receives the order items from the order queue	Receives Data
Chef ↔ OrderDisplay	Customer can see the order status on his tablet displayed by the customer	Provides data
Waiter ↔ OrderStatus	Waiter periodically requests updates of the ordered items so that when the order is ready, they can deliver it the right table	Requests Updates

LITTLE BITS

Waiter ↔ TableRecord	Waiter marks the table as occupied when the customers seat themselves.	Provides Data
Chef ↔ OrderStatus	Chef passes updates to order status. (Notifies if the order is ready or not)	Provides Data
Busser ↔ CleanTable	Busser cleans the table once the payment has been processed.	Receives Data
Busser ↔ TableRecord	Busser updates the status of tables online once they have been cleaned up.	Provide Data
Controller ↔ Page Maker	Controller passes requests to PageMaker and receives back pages prepared for displaying	Convey Requests
Page Maker ↔ Database Connection	Database Connection passes data retrieved data to PageMaker to render them for display	Provides Data
Investigation Request ↔ Database Connection	Database Connection receives the menu items and corresponding ratings and sends it to Investigation Request to display more details about each menu items	Show Data
Database Connection ↔ Notifier	Database Connection passes the inventory information to Notifier which displays any inventory-related notifications	Notify
InfoChanger ↔ Database Connection	Database Connection passes retrieved data to the InfoChanger to change employee information	Change Info
Controller ↔ Database Connection	Controller passes search requests to Database Connection	Conveys Requests
Page Maker ↔ Interface Page	Page Maker prepares the Interface Page	Prepares
Database Connection ↔ Predictor	Database Connection receives financial information and sends it to Predictor to estimate future profits	Estimates

Attribute Definitions

Concept	Attributes	Attribute Description
OrderItem	Name	Name of menu item
	Cost	Cost of menu item
	Category	Category of menu items
Order	Customer info	Information about the customer: name or ID
	Order number	The ID number created for each customer order
	Table number	Number associated with the customers table
	Check	The most recent customers bill
Check	ID	Customers Associated ID
	Items	Food items ordered
	Cost for each items	Cost for each item ordered
	Total Cost	The total cost for all items ordered
	Tip	Customers preferred tip amount payment
Entertainment	Game	Games available for customers enjoyment
Manager	Name	Name of Manager
	Manager ID	Manager has a unique identification credential associated with them for login
	Privileges	Manager has specific privileges including menu modifications, hire/terminate employees, track profit/loss, etc.t0
Waiter	Name	Name of the waiter
	Position	Position held by the employee at the restaurant
	Employee ID	Each employee has a unique identification credential associated with them for login
	Contact Information	The employee's address, email address, phone number, etc.

	Table Association	The tables to which the waiter is serving
OrderQueue	Orders	Orders are placed on a queue within the system, which can be accessed by the chef
TableStatus	Clean and Empty	Customers can be seated in it
	In Use	There are customers seated in the table
	Dirty	There are no customers seated in the table and it needs to be cleaned
Notifier	Inventory quantity	Signals to user(manager) that items in the inventory are running low
Investigation Request	Menu	List of menu items, ratings, and comments for investigation
InfoChanger	Employee change	Change employee attributes and values.
PageMaker	Records	Information retrieved from the database to be rendered
	Document	HTML document created to display retrieved information to the user
InterfacePage	Document	HTML document displaying to the actor the current context, what actions can be done, and outcomes of the previous actions
TrafficDisplay	Traffic	Information relative to the floor activity in the restaurant
Predictor	Customer Traffic	Information pertaining to the amount of customers coming to the restaurant
	Finances	Information relative to the profits and losses as well as payroll costs

Traceability Matrix

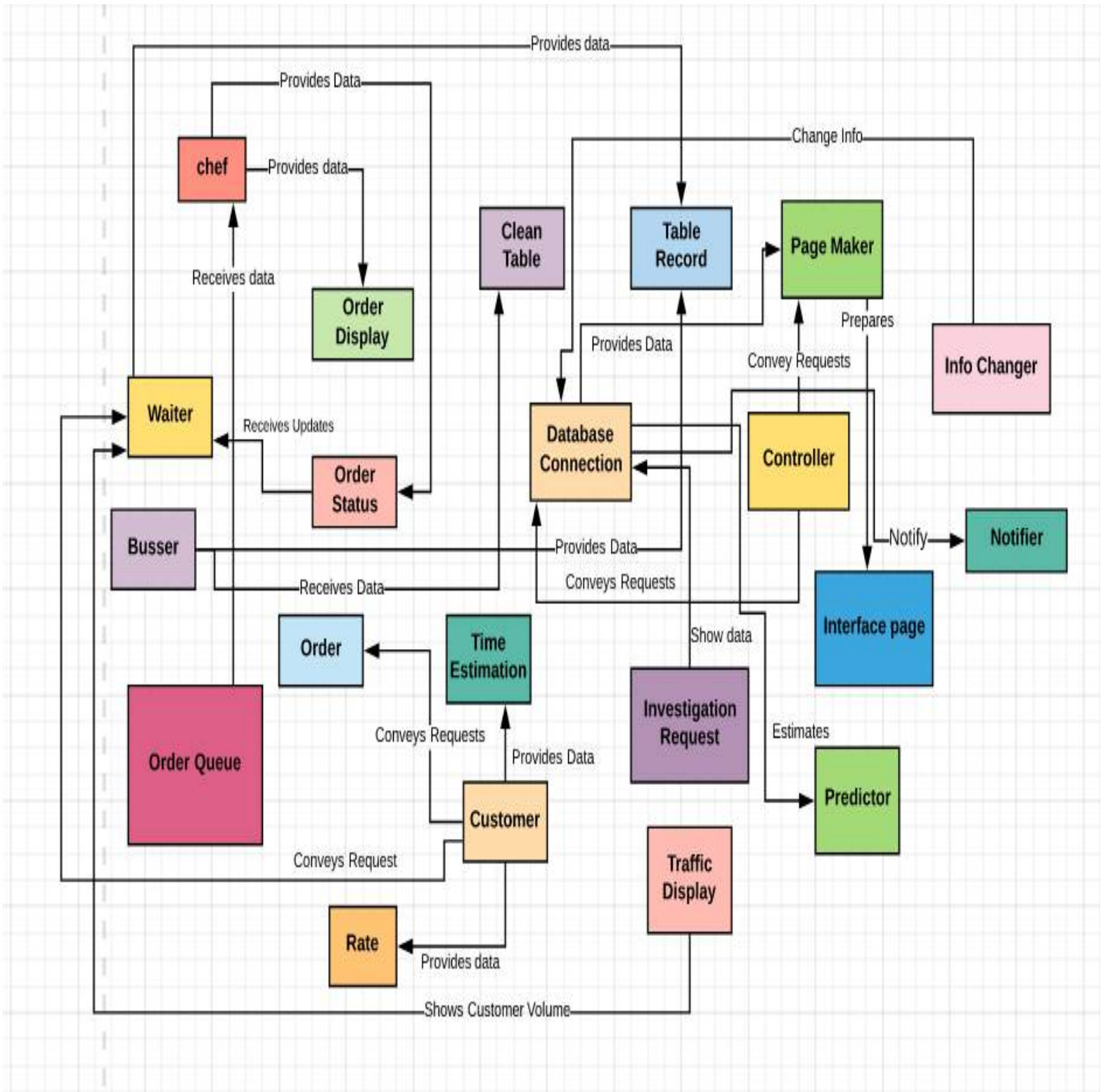
Use case	Order	OrderItem	Check	Entertainment
UC-1	X	X		
UC-2				
UC-3	X	X		
UC-4		X		
UC-5		X	X	
UC-6	X	X		
UC-7				X

Use case	OrderQue	Order Display	Order Status	CleanTable	TableRecord	Order	Check
UC-1	X	X				X	X
UC-2				X	X		
UC-3		X				X	X
UC-4	X					X	X
UC-5	X						
UC-6	X	X	X	X	X		

LITTLE BITS

Use Case	Controller	Page Maker	Interface Page	Info Change	Notifier	Database Connection	Investigation Request	Traffic Display	Predictor
UC-1	X	X	X		X	X	X		
UC-2	X	X	X			X	X	X	
UC3	X	X	X			X	X		
UC-4	X	X	X		X	X	X		
UC5	X	X	X	X		X	X		
UC6	X	X	X			X	X	X	X

Domain Model Diagram



Systems Operation Contracts

Name:	Order Management
Responsibilities:	Allow the customer to place an order and to view when said order will be prepared
Use Cases:	UC-1
Exceptions:	None
Preconditions:	Menu is displayed on the customer's tablet
Postconditions:	Order queue is displayed on the chef's tablet; order for designated table is displayed on the waiter's tablet

Name:	Traffic Monitoring
Responsibilities:	Place reservation, Place Orders, Mark Tables Empty, Mark Tables Cleaned
Use Case:	UC-2
Exception:	None
Preconditions:	The user creates a reservation, come to the restaurant, and places an order
Post-conditions:	The waiter can see when a table is done and bussers can mark when tables are cleaned

Name:	Customer Ordering
Responsibilities:	View the menu, ordered desired food and pay the bill
Use Cases:	UC-3
Exceptions:	None
Preconditions:	A tablet open on the customer interferences
Postconditions:	The customer is able to order and get assisted with food orders.

Name:	Payment Process
Responsibilities:	Pay the bill successfully
Use Cases:	UC-4
Exceptions:	None
Preconditions:	The Customer has clicked and opened the Pay Your Bill tab and has chosen one of the methods of payment.
Postconditions:	The customer pays the bill with or without the Waiter's assistance depending on which option the customer chooses.

Name:	Managing Employees
Responsibilities:	See Schedule, Log In/Out, Adjust Schedule Preferences, Auto Clock In/Out
Use Case:	UC-5
Exception:	None
Preconditions:	The user logs in which automatically clocks them in
Post-conditions:	Preferences are saved and user logs out

Name:	Restaurant Management
Responsibilities:	To allow the manager of the restaurant to control employee scheduling and pay rates, view inventory status, manage finances, and monitor floor traffic in the restaurant
Use Cases:	UC-6
Exceptions:	None
Preconditions:	Manager login page is displayed on the manager's tablet
Postconditions:	Manager changes to previously stored database information is saved; profit projection is calculated and displayed as desired by the manager

Mathematical Model

Polynomial Regression:

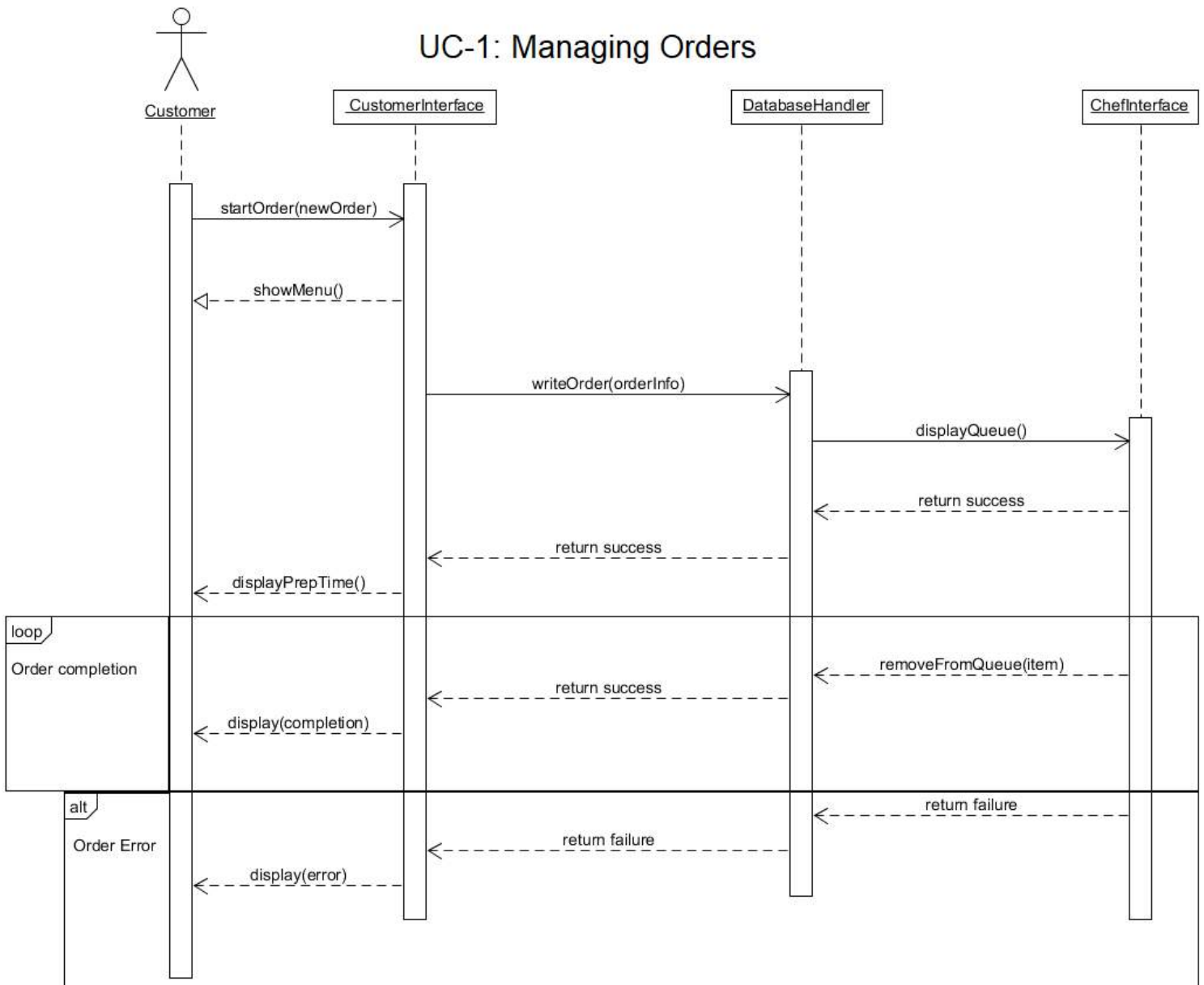
For the financial management system, a mathematical model will be used in making predictions on the level of the profit the restaurant will experience given the customer traffic in a given amount of time and the popularity of the restaurant's menu items. With two variables being quantity of items sold and total income collected, we will create an estimation of total profits collected over a time period specified by the user, the manager. Polynomial regression is a technique which takes the form:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \dots + \beta_h X^h$$

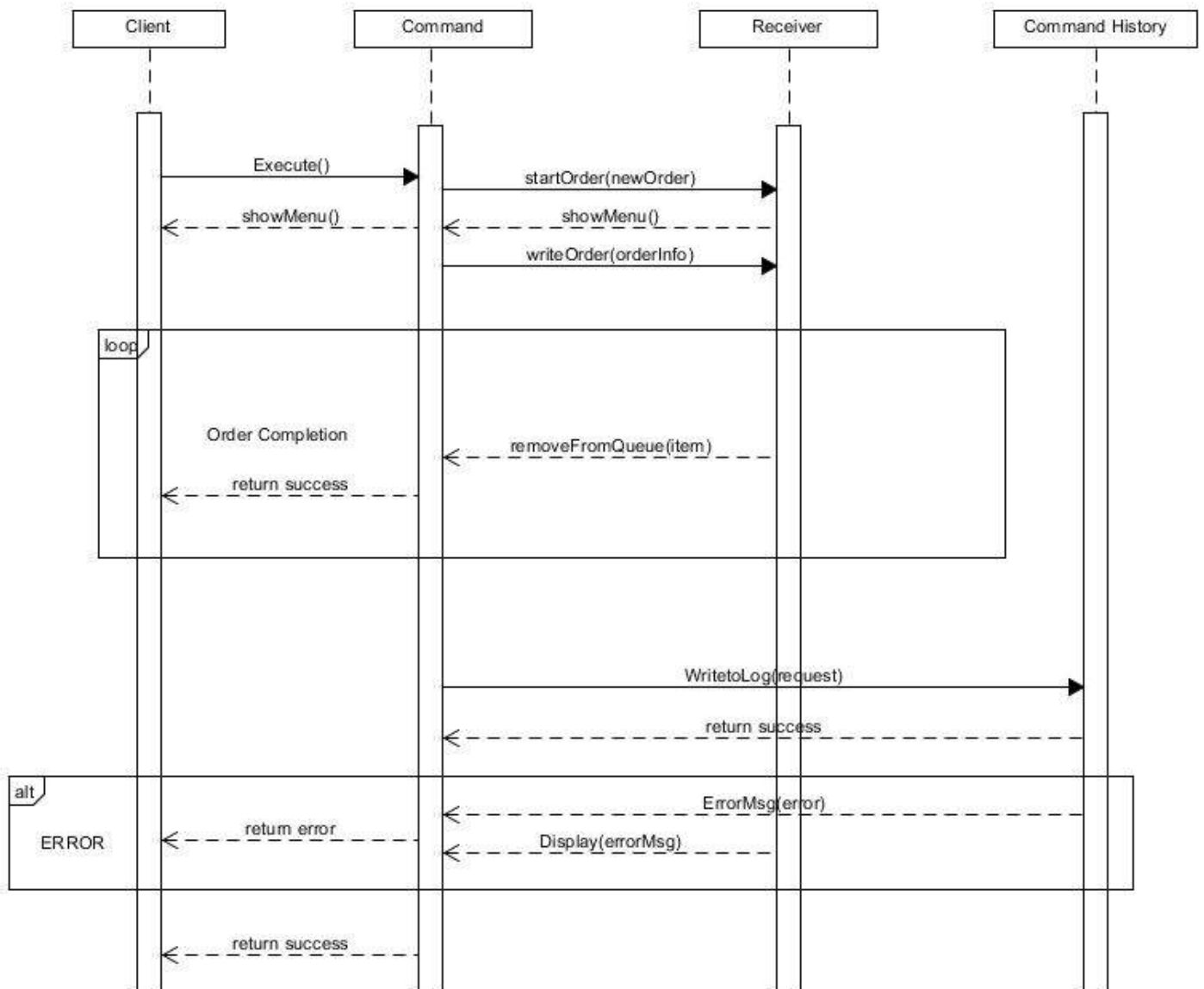
Where X represents the total quantity of items sold, Y represents the total income collected by said menu items, ϵ represents the error and h represents the degree of the polynomial. The manager will be able to choose the degree of the polynomial to use in making a projection on the profits of the restaurant. Using established data points taken over a period of time, two months for example, the system will begin to make calculations on profit projections for months and/or years in advance, given input for the desired time period from the manager.

Interaction Diagram

UC-1: Managing Orders



UC-1: Managing Orders

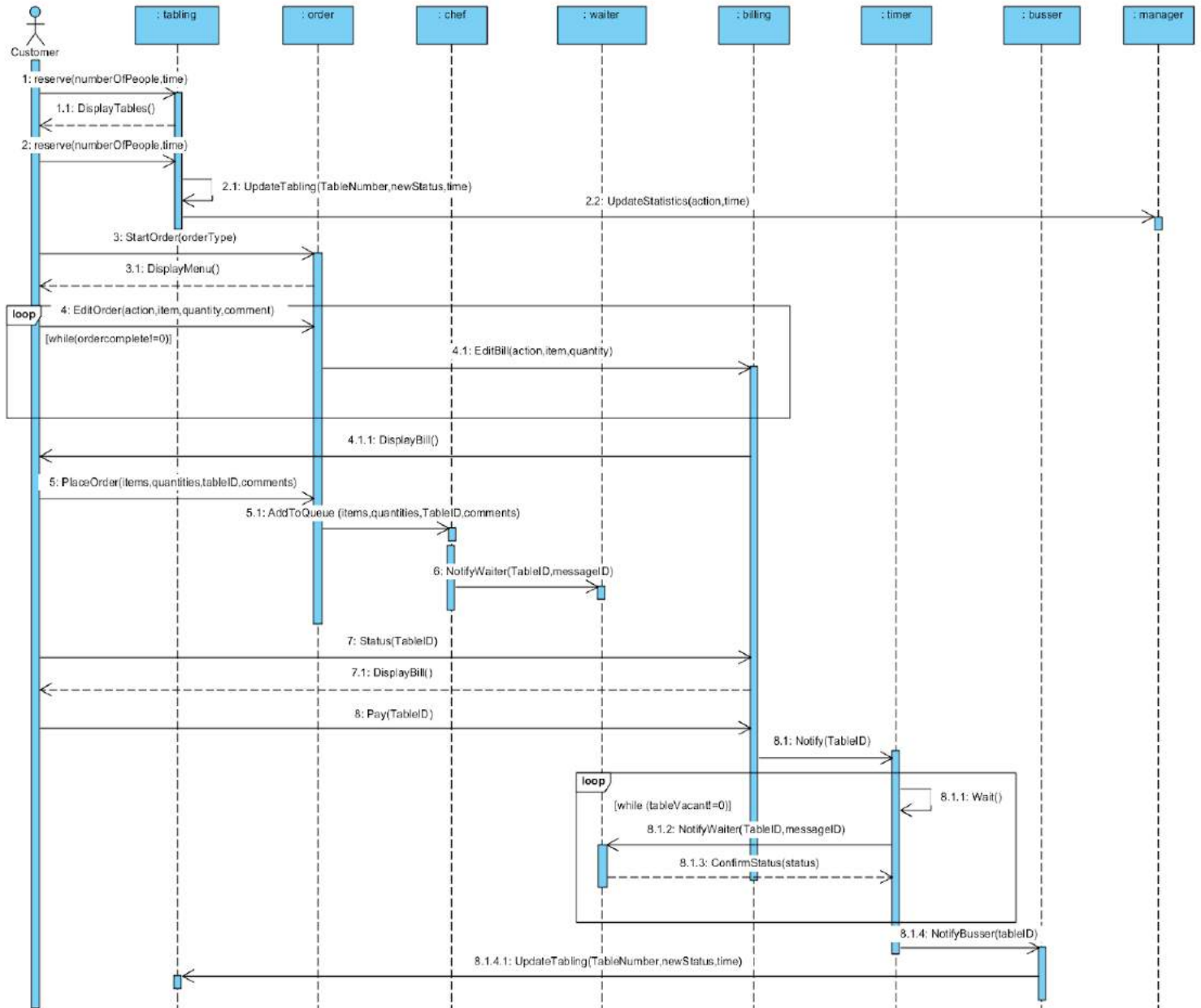


Explanation:

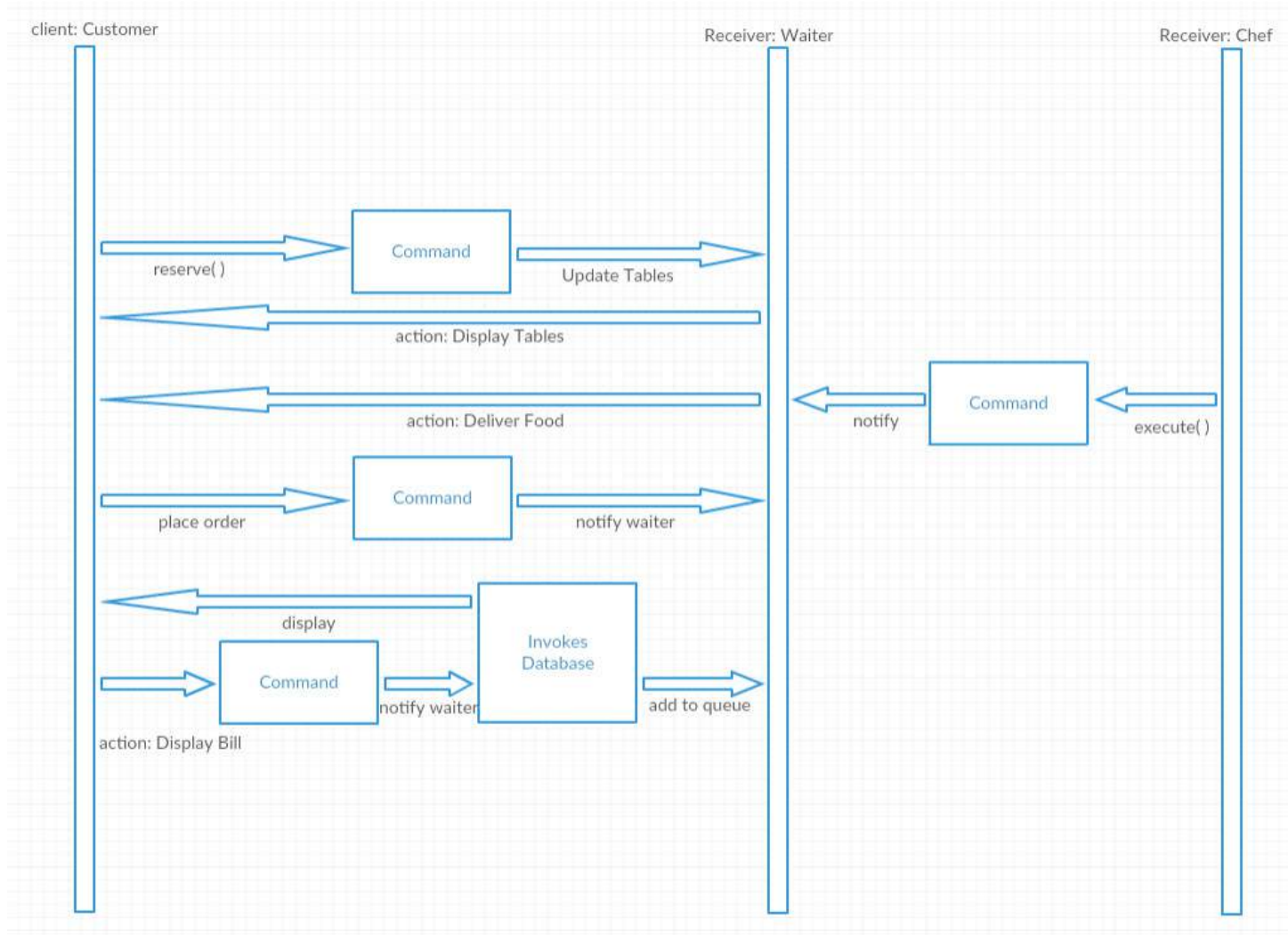
The customer places an order from the menu interface. Once the customer places the order, the order info gets sent to the chef's queue which is present on the chef's interface at all times. After the chef receives the order and completes the order, it is clicked as finished and the waiter and customer will be notified that the order is done and will be arriving.

In the customer interface, using the Command design pattern helps in using a command which would facilitate running the entire program and facilitate the execution of certain actions as they are requested. In our new design, the application we coded will allow the customer to call different customer functions and they will be executed one-by-one by the application. The use of the Command design pattern specifically improved the design because this design was helpful in helping to make sure the program runs nicely. As soon as the client opens the application, the driving file will begin to execute commands as the client requests them to be done. The customer functions will be activated by a button press made by the clients(customer(s)) on the customer interface in the menu.

UC-2: Traffic Monitoring



LITTLE BITS



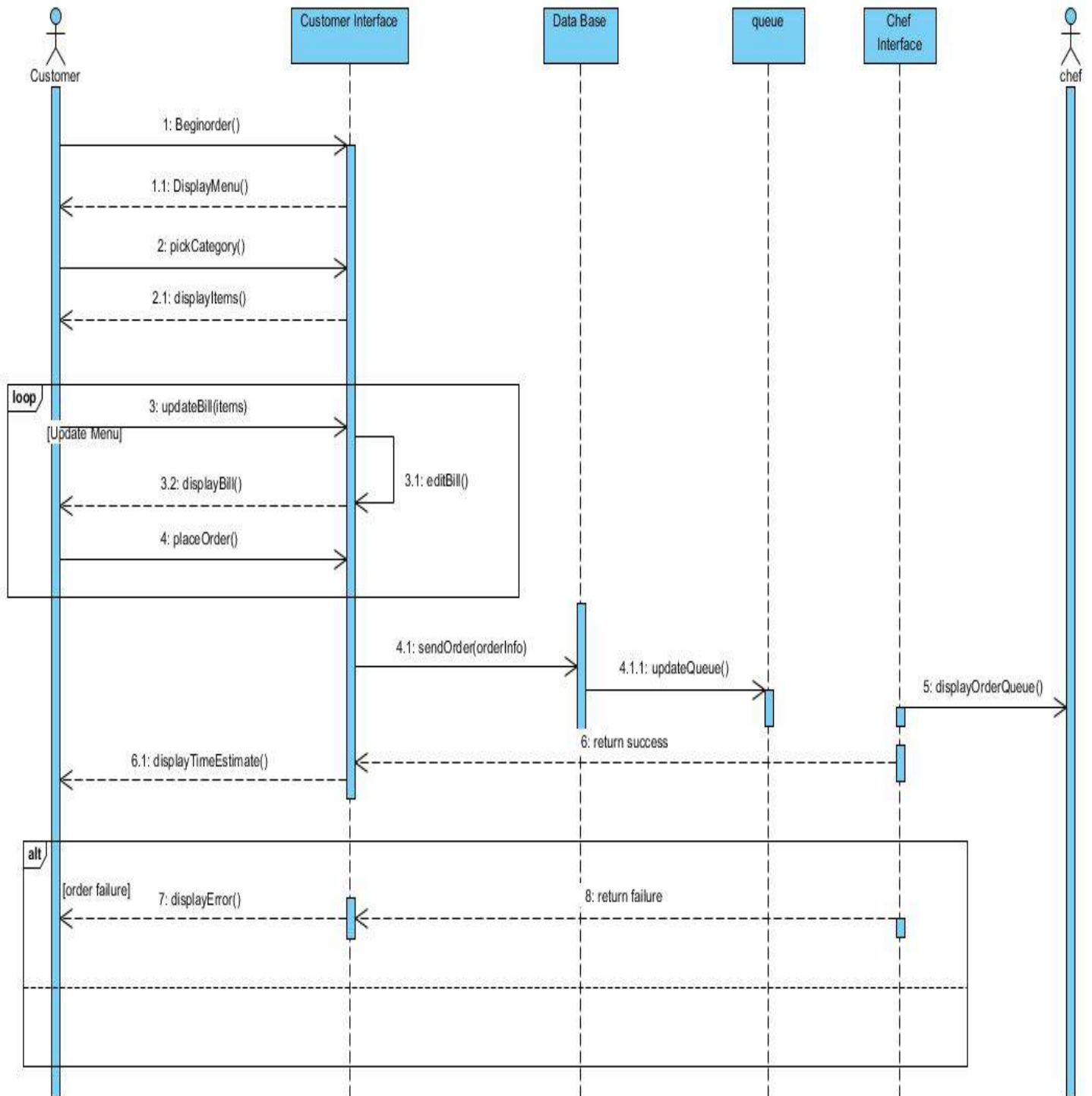
Explanation:

The customer will go online to make a reservation. They click reserve to reserve a table. A map of the tables in the restaurant will show available tables marked as green. The customer will then choose the desired table and reserves it. The system will then update its records of which tables are available. Once the customer shows up, they can then be seated and can take a look at the menu. They can add or remove items from their order and they can attach notes to their order. Every time the order is edited, the bill for that order is also updated. When the customer is ready, they can place the order. The order is then added to the order queue and the chef is then notified about the order. When customers are done, they can pay their bill. Once the customers pay their bill and leave, there will be a timer which after a five-minute waiting period, the waiter can confirm if the customers left. If confirmed, then the table will be marked empty and dirty. If not confirmed, the timer will go for another five minutes until it is confirmed. This will notify the busser that a table needs to be cleaned. Once the table is cleaned, the busser will mark that it is ready for use, which will update the table map. Then customers ready to place reservations or employees looking for available tables can see open tables.

Though it may not be evident at first the Command Pattern Interaction Diagram provides a uniform method signature to the Server or Receiver. This allows clients to decide with to execute different commands. In this specific user case the command pattern to useful to bundle the receivers and centralize the client, being the customer. The client is able to execute jobs such as reserving a table, placing an order, and displaying the bill. Then each interaction is sent to the respective receiver.

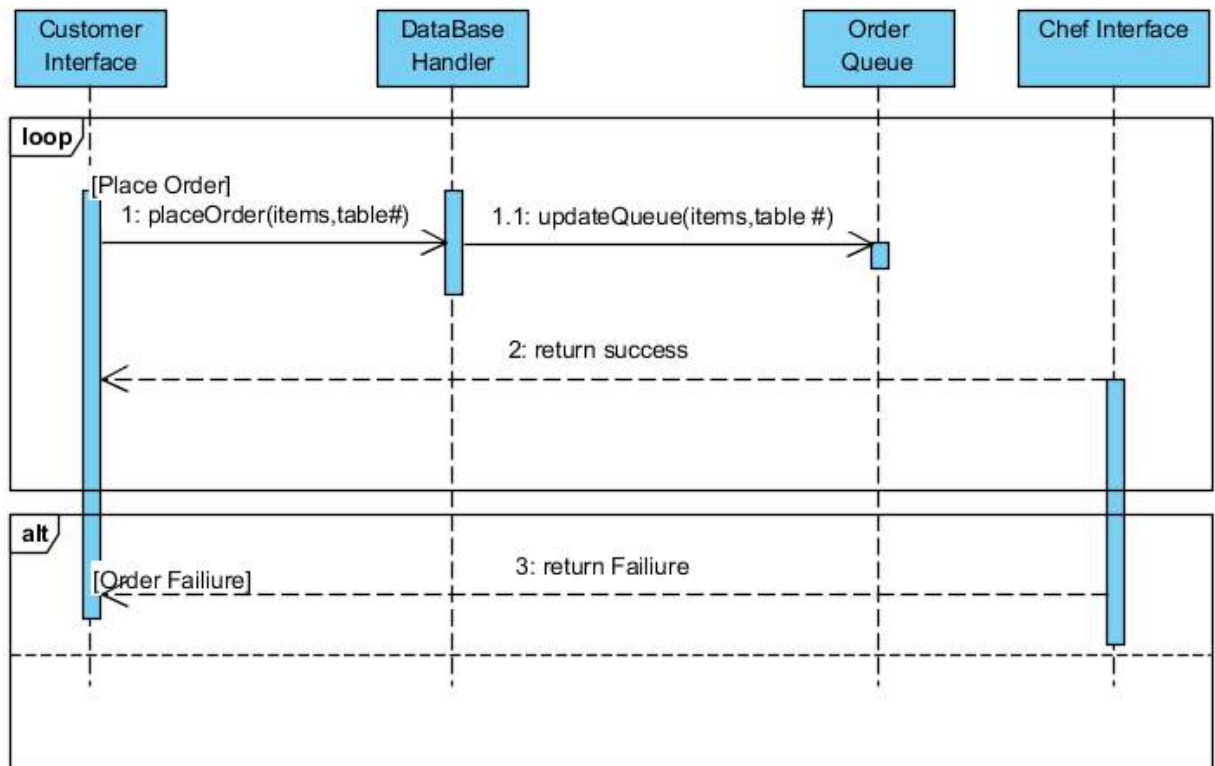
UC-3: Customer Ordering

sd ID-3



LITTLE BITS

sd ID3DP



Punlisher-Subscriber
format

Explanation:

The main modules are menu display, creating an order, and playing games.

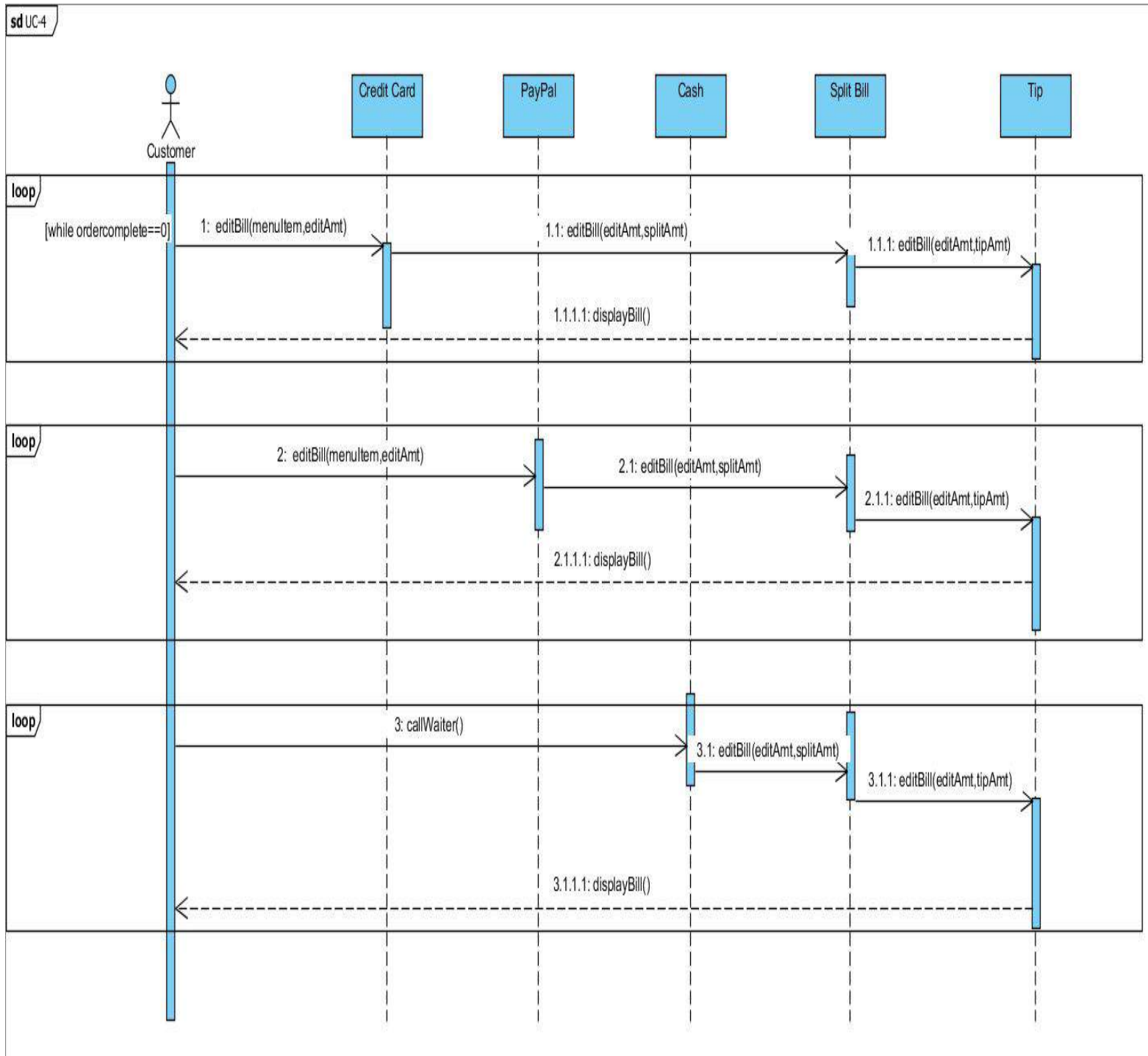
This case begins when the customer or user sits down at the table and initiates interaction with the tablet. Upon beginning an order the customer is shown several options including: Order Food, Play Games, and SOS. Each button has a very important function in the dining experience.

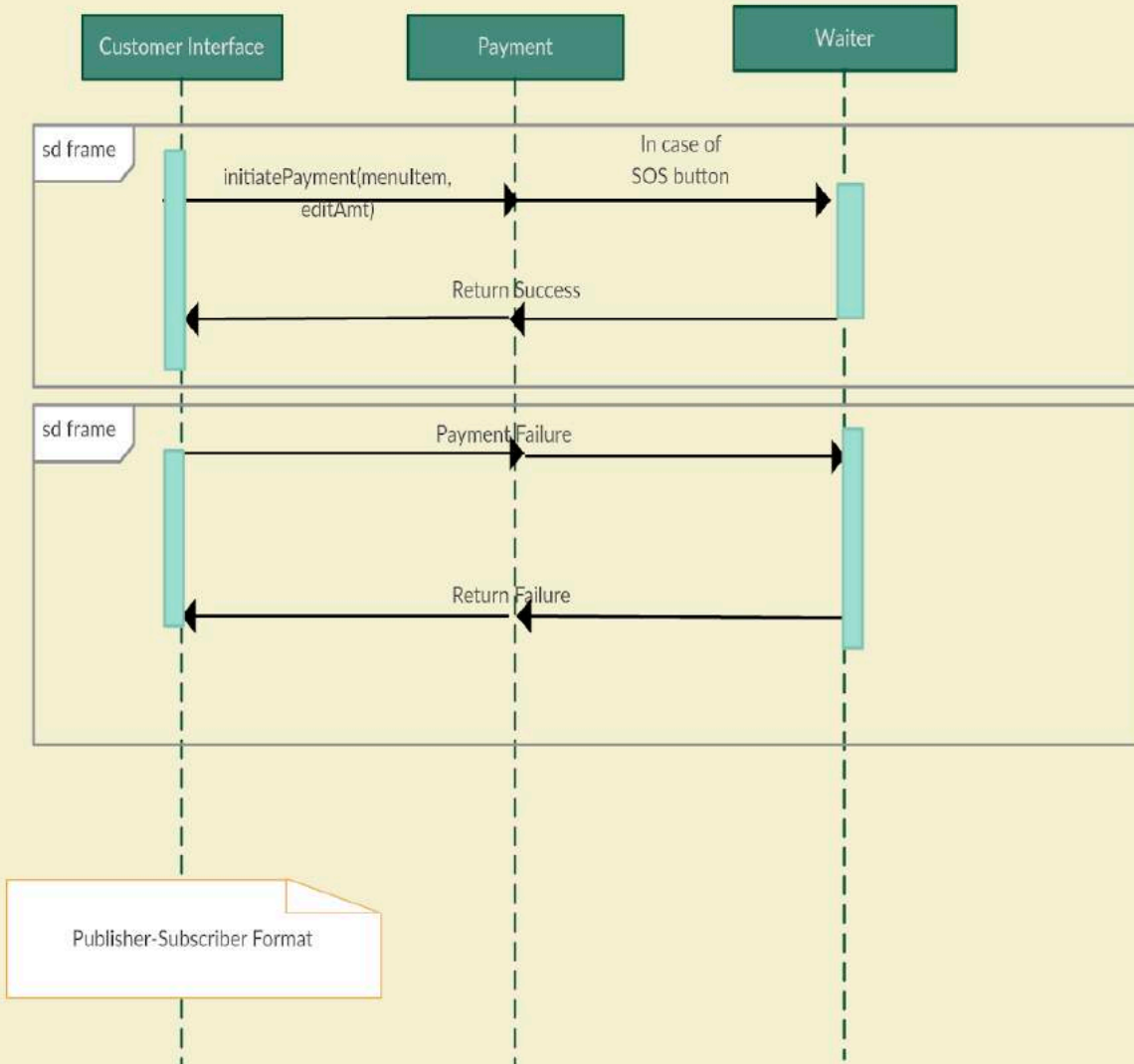
Ordering food is the main priority of this user case, this covers everything from choosing an entree to receiving the food, and everything in between. Upon selection of the “Order Food” option the system displays the menu categories such as Pasta Entrees, Drinks, Dessert, and Seafood. The customer can see the menu categories and browse them. Once the customer selects a category then their specific item list is displayed by the system for the customer to view. The customer can then select what items they want from the menu and write in any specific requests associated with the item such as food allergies or meat tenderness. Once the customer is ready to place the order for that item they select the “Order” button on the display. This will send the item and its specifications to the chef, the customer will then receive a notification stating how much time they should expect to wait till their food is ready.

Additional to the ordering the customer also has an “SOS” option where the customer can send a signal to their waiter indicating that they need help or have an issue that needs to be addressed. The waiter then receives the signal and can choose to respond to it from their personal tablet. Another option available to the customer is the “Play Games” option where customers can choose to play a free game while they wait for their food. Upon selecting a game on the display, the customer can begin to play and can choose to stop the game an any point.

We used the publisher-subscriber pattern because it allows us to notify doers/subscribers when the information/event of their interest becomes available, without knowing what for or how this information will be used. It simplifies the relationships between the the users and the systems and allows updating it much easier.

UC-4: Payment Process





Explanation:

This is a continuation of the dining experience when the user is ready to pay their bill and leave the establishment.

The customer can choose to pay their bill at any point after placing their food order. Most dining individuals do this once they have finished their food, this is convenient because here the customer can select if they need any takeout boxes if they have any food leftover that I want to take home. Once the customer is ready to pay they can select the option to pay their bill. At this point the display will show three methods to pay the bill: credit card, gift card, cryptocurrency, PayPal, or with cash. If the customer selects credit card the display will indicate that the customer is to swipe their card through a port on the tablet and indicate whether they are using Credit or Debit. If the customer selects to pay via PayPal then the following screen will ask the customer to enter their PayPal user ID and password.

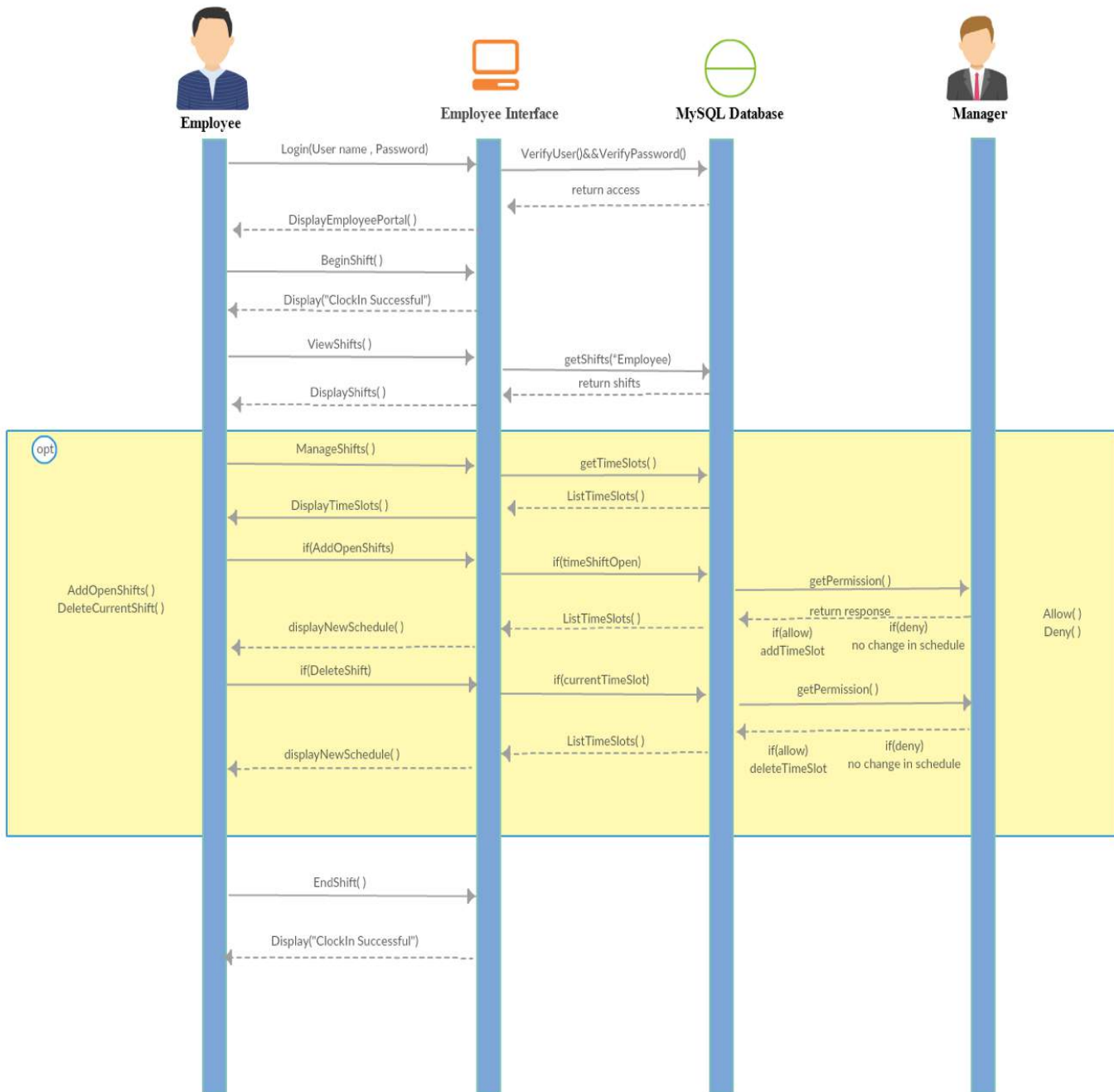
If the customer selects to pay in cash, then the customer will be faced with a message stating that their waiter is on their way. This will also send a message to the waiter's personal tablet saying that the table is ready to pay in cash. Once the waiter arrives at the table the customers can pay the waiter directly and receive change if needed.

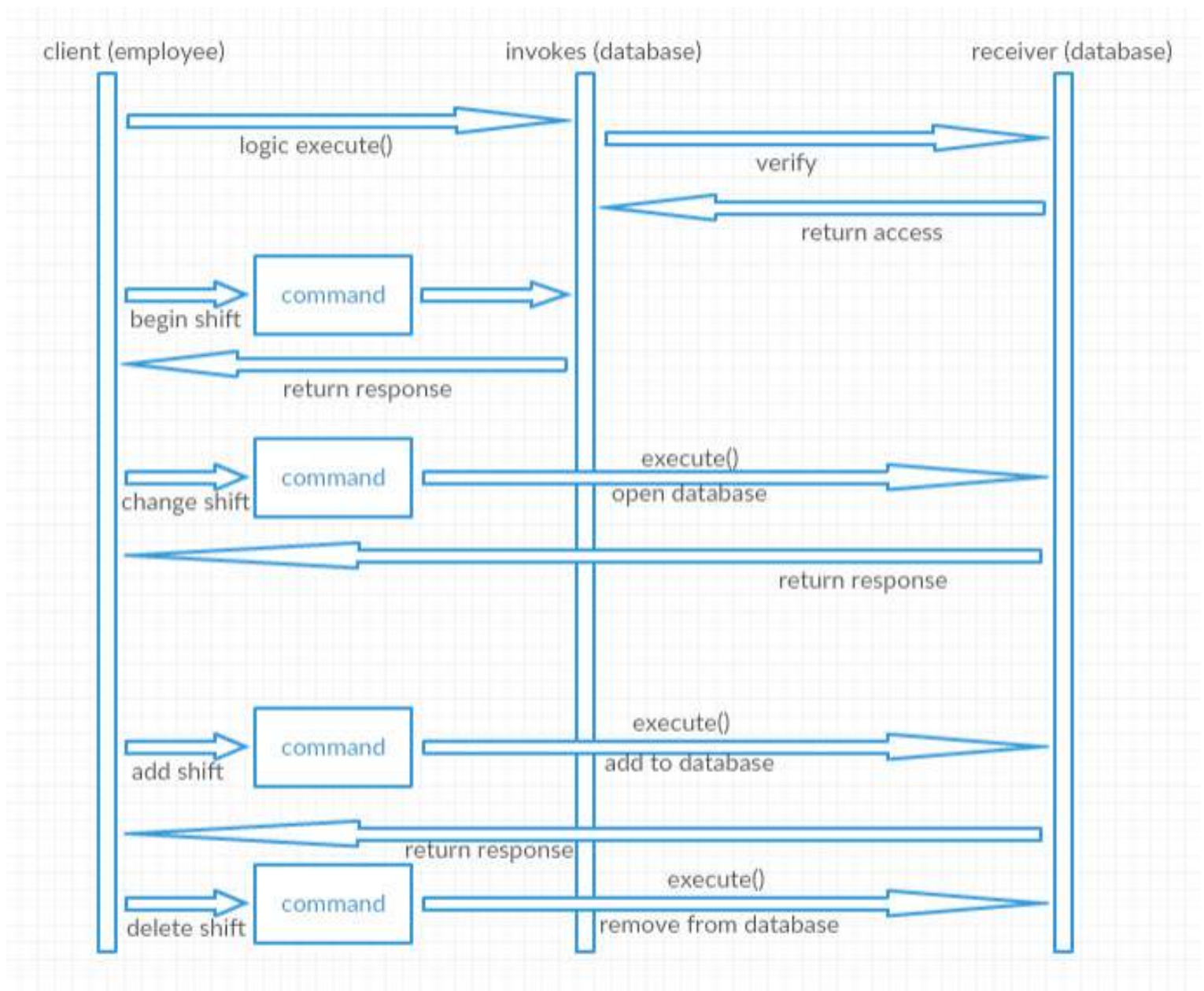
Upon selecting a form of payment the customer is also able to select if they require any takeout boxes for any leftover food. They can select how many boxes they need depending on how many food orders were placed. The waiter is notified by the customer's selection and will bring the boxes over when collecting cash payments or after Credit Card or PayPal payments have been processed.

If the customer wished to split the bill, then they may do so by selecting that option. The customer can then select which items they wish to move to the second bill. When paying for each bill a different payment method can be used on each bill. If the customer wished to split the bill more, then they can choose to call the waiter over and he/she can split the bill further.

We used the publisher-subscriber pattern because it allows us to notify doers/subscribers when the information/event of their interest becomes available, without knowing what for or how this information will be used. It simplifies the relationships between the users and the systems and allows updating it much easier.

UC-5: Managing Employees





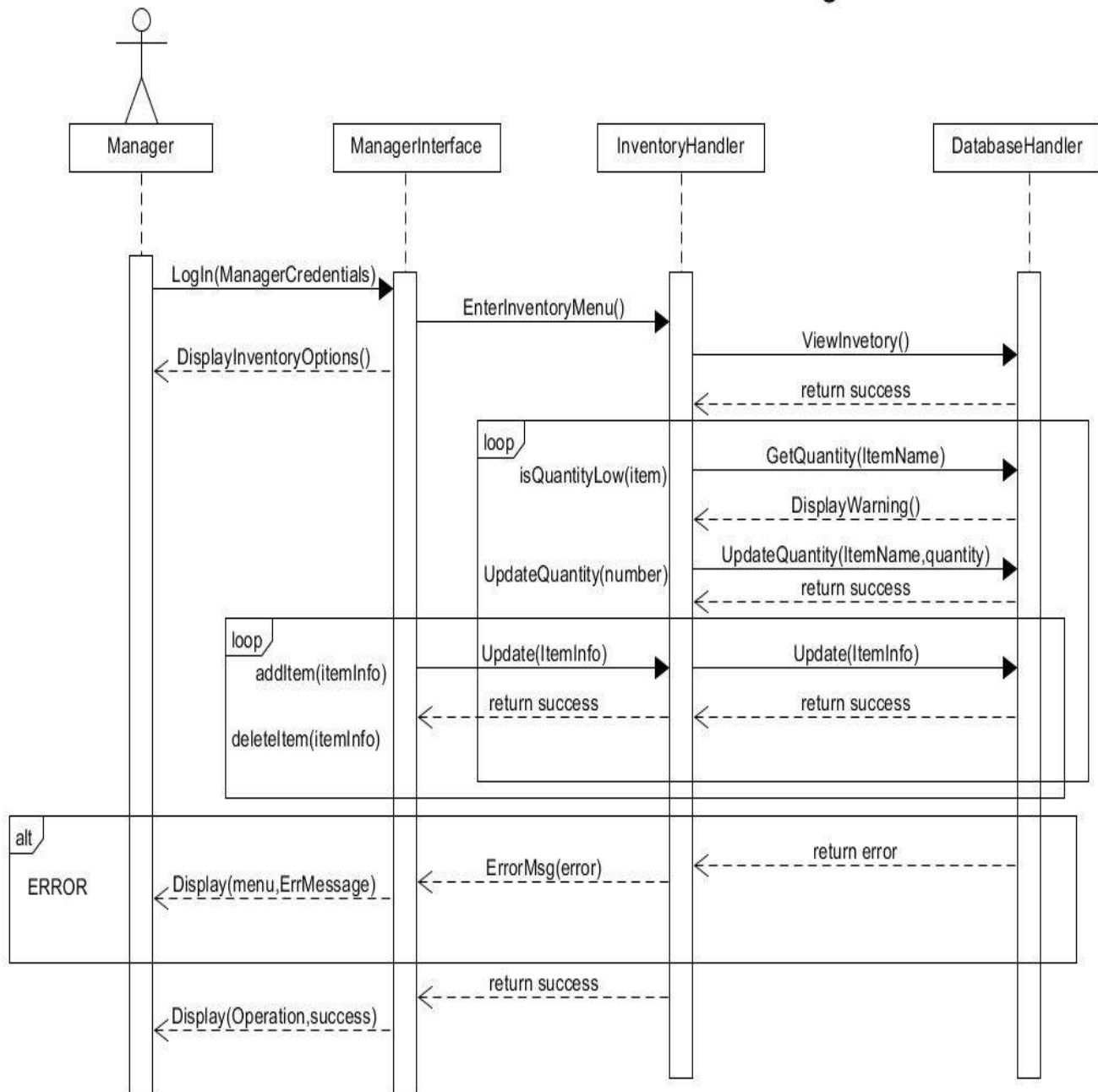
Explanation:

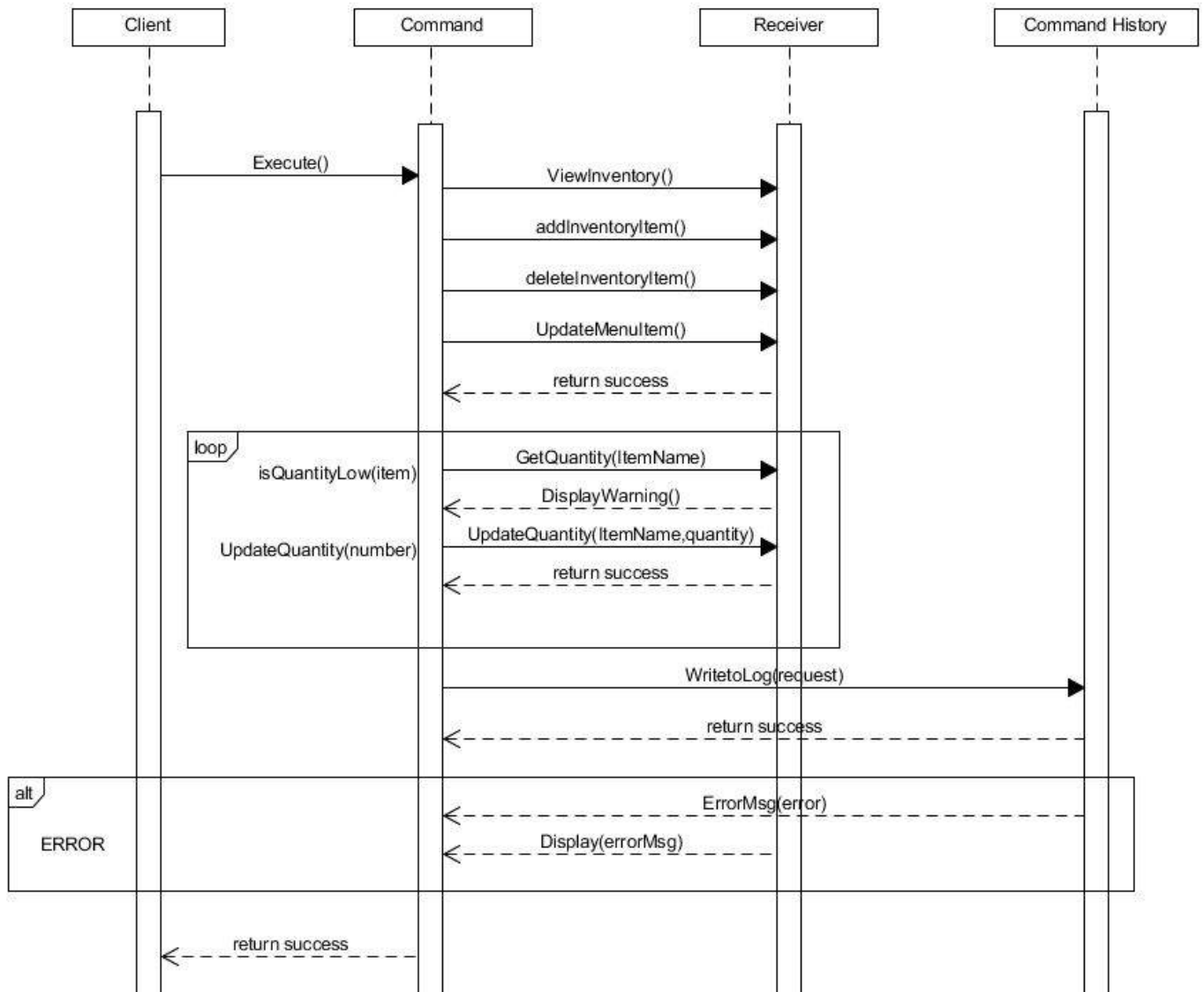
The employee (waiter, busser or chef) will first log into the employee portal. The employee can login by inputting their user identification on the log in page. Once they put in their identification code, they will be automatically clocked in. Once in the portal the employee can view their portal and all the available options for their position. There will also be a tab in the portal for schedule management. In this tab, employees can view shifts and also put preferences for future shifts. In the tab the employee will see open shifts that need to be assigned. Once the employee inputs their preference, the manager will be able to see all of the options and can create the schedule based on the employee's availability. Once the schedule is created, it will be posted so employees can see the new schedule and can confirm or deny.

The advantages of using the Command Pattern Interaction Diagram include the ability to decouple the business logic from the parameter preparation. Decoupling allows clients to evolve independently by different developers. The command method `execute()` does not expect a return but a response may be sent separately. In this user case the commands change shift, add shift, and delete shift are sent from the client to the receiver via the `execute()` function. The database acts as both the receiver and the invoker, while the employee acts as the client.

UC-6: Managing Restaurant

UC-6a: Restaurant Management



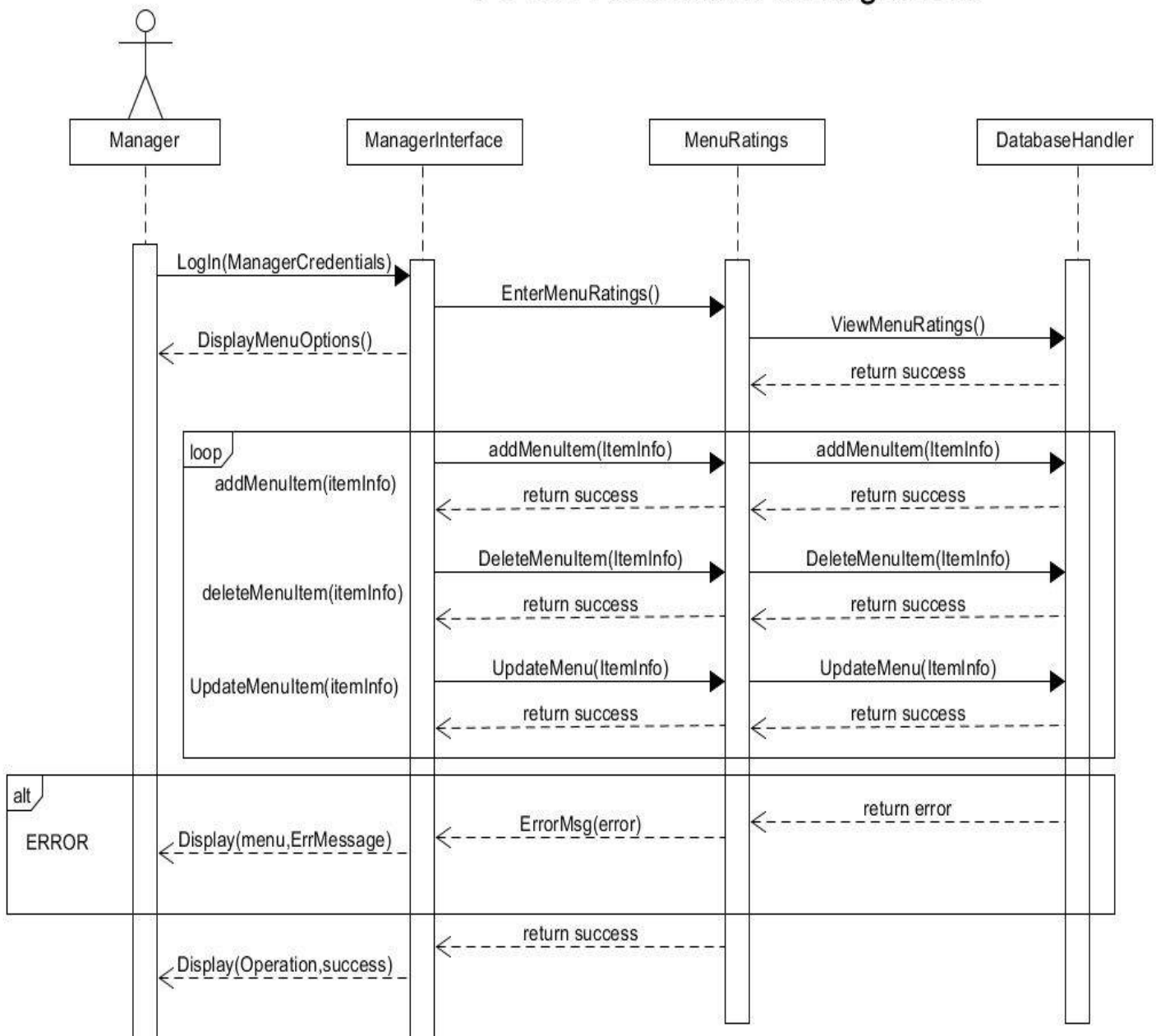


Explanation:

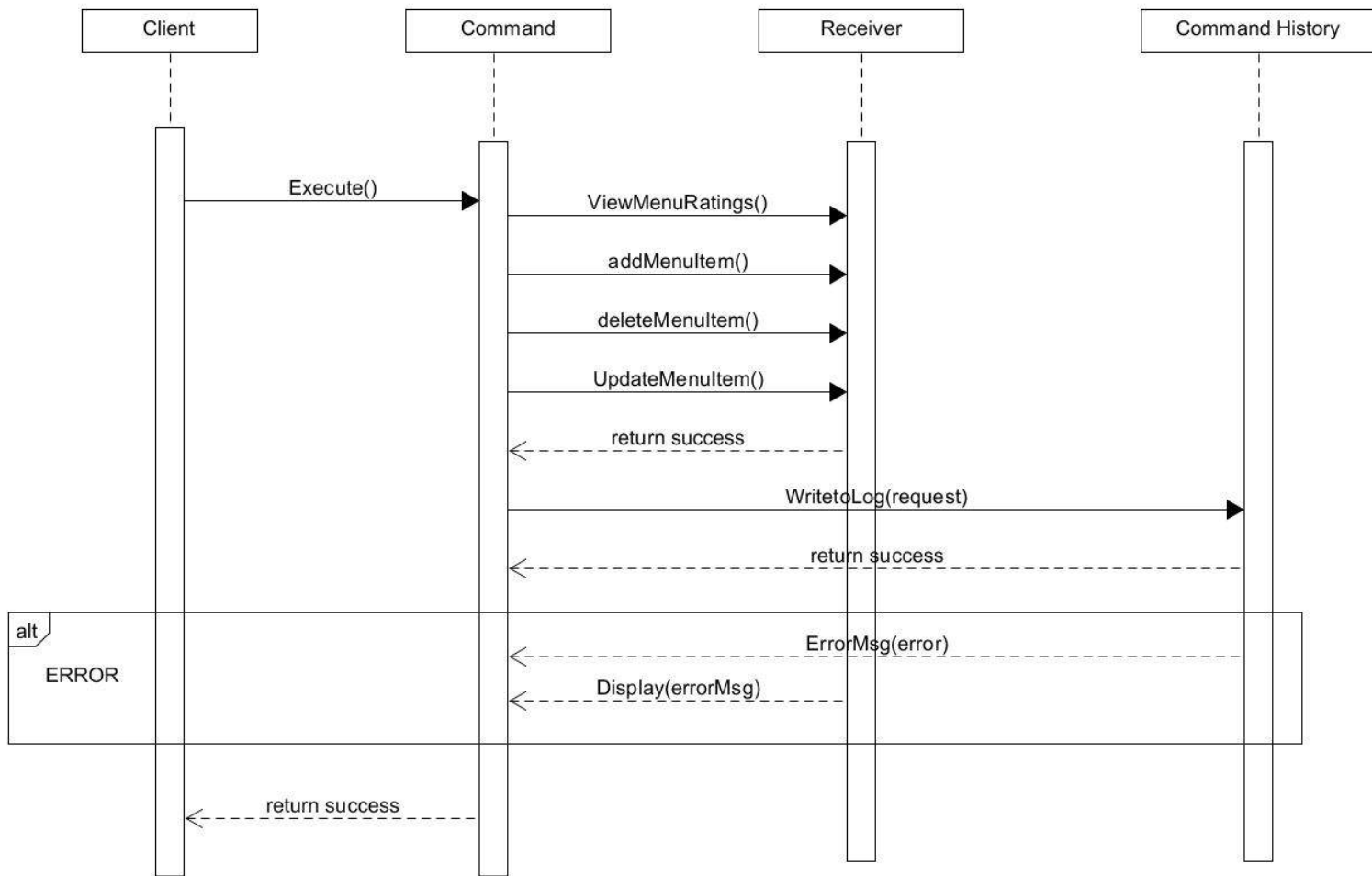
This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the inventory menu. After the manager prompts the system to display the inventory menu, they will be allowed to add or delete items from the inventory as is needed. While viewing the inventory, the system will display to the manager a warning if any of the items in the inventory is below a given threshold in quantity. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.

In the manager interface, we utilized the Command design pattern which would enable us to use a central command file which would drive the entire program and facilitate the execution of certain actions as they are requested. In our new design, the flask application we coded will allow the manager to call certain inventory manipulation functions and they will be executed one-by-one by the application. The use of the Command design pattern specifically improved our design because this design was the most conducive to the flow of our program. As soon as the client opens our application, the driving file will begin to execute commands as the client requests them to be done. Each inventory function will be activated by a button press made by the client on the manager interface, in the inventory menu.

UC-6b: Restaurant Management



LITTLE BITS

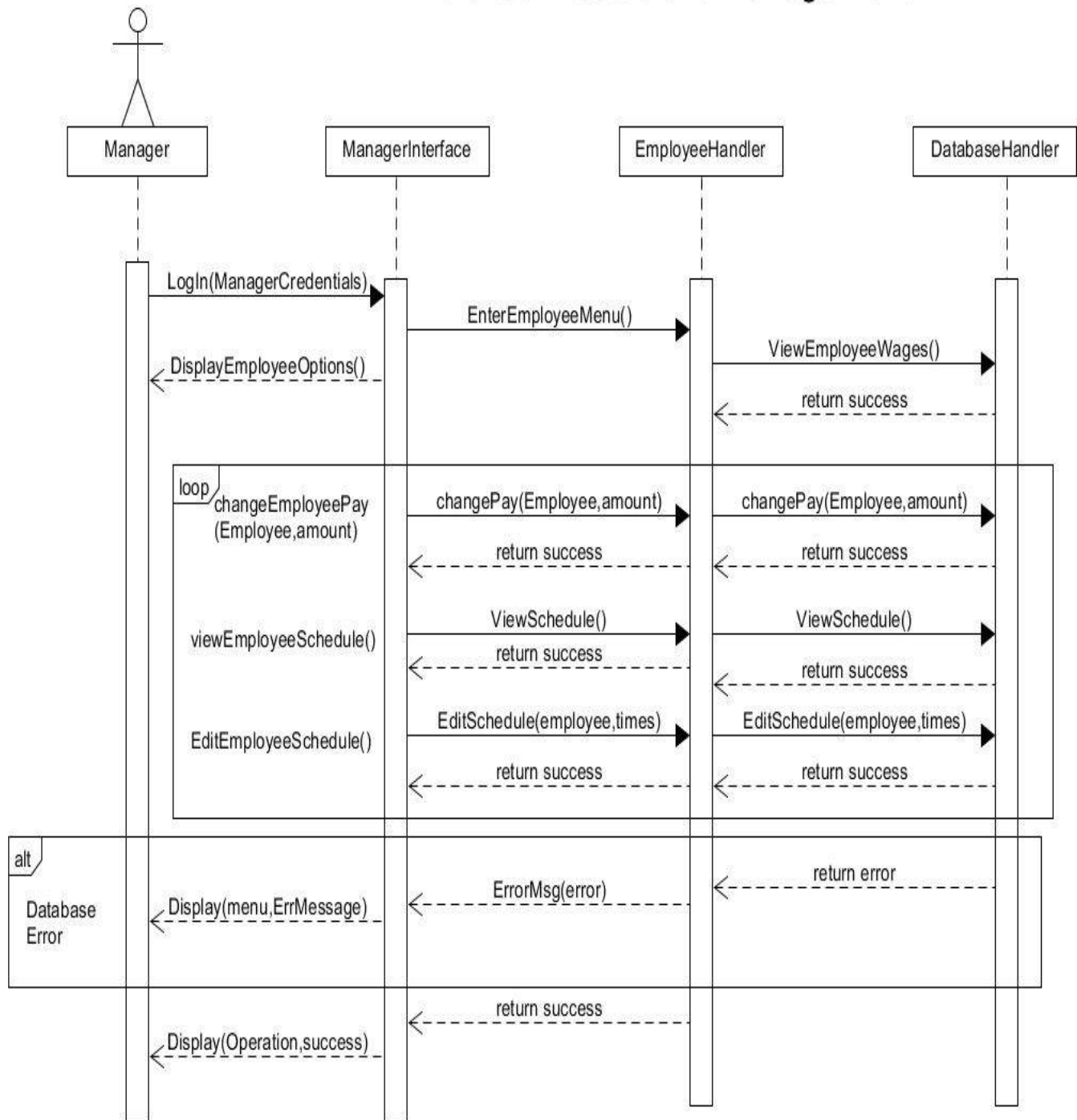


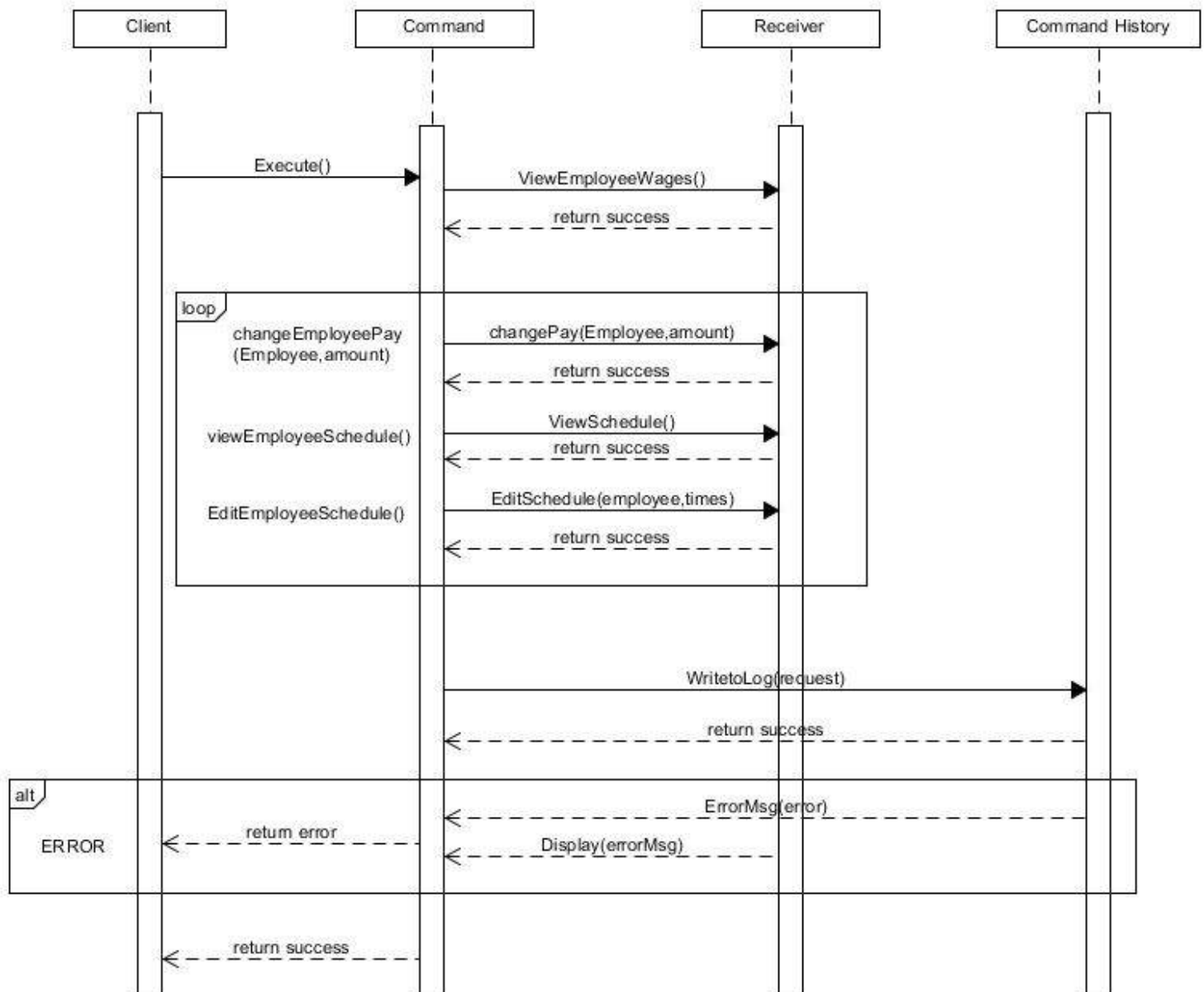
Explanation:

This diagram outlines the process the program goes through to display the restaurant's menu to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the restaurant's menu and editing options for the menu. From there, the manager may decide to view the ratings of each of the restaurant's menu items, or decide to edit the list of items on the menu itself. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.

In the manager interface, we utilized the Command design pattern which would enable us to use a central command file which would drive the entire program and facilitate the execution of certain actions as they are requested. In the new design the flask application we coded will allow the manager to update the menu via add, delete, and update functions. Additionally the manager will be able to view the ratings of menu items. Each of these functions will be executed individually by the application. The use of the Command design pattern allows the driving file to execute commands as soon as the client requests them. Each Menu function will be activated by a button press by the client on the manager interface in the menu and ratings menu.

UC-6c: Restaurant Management



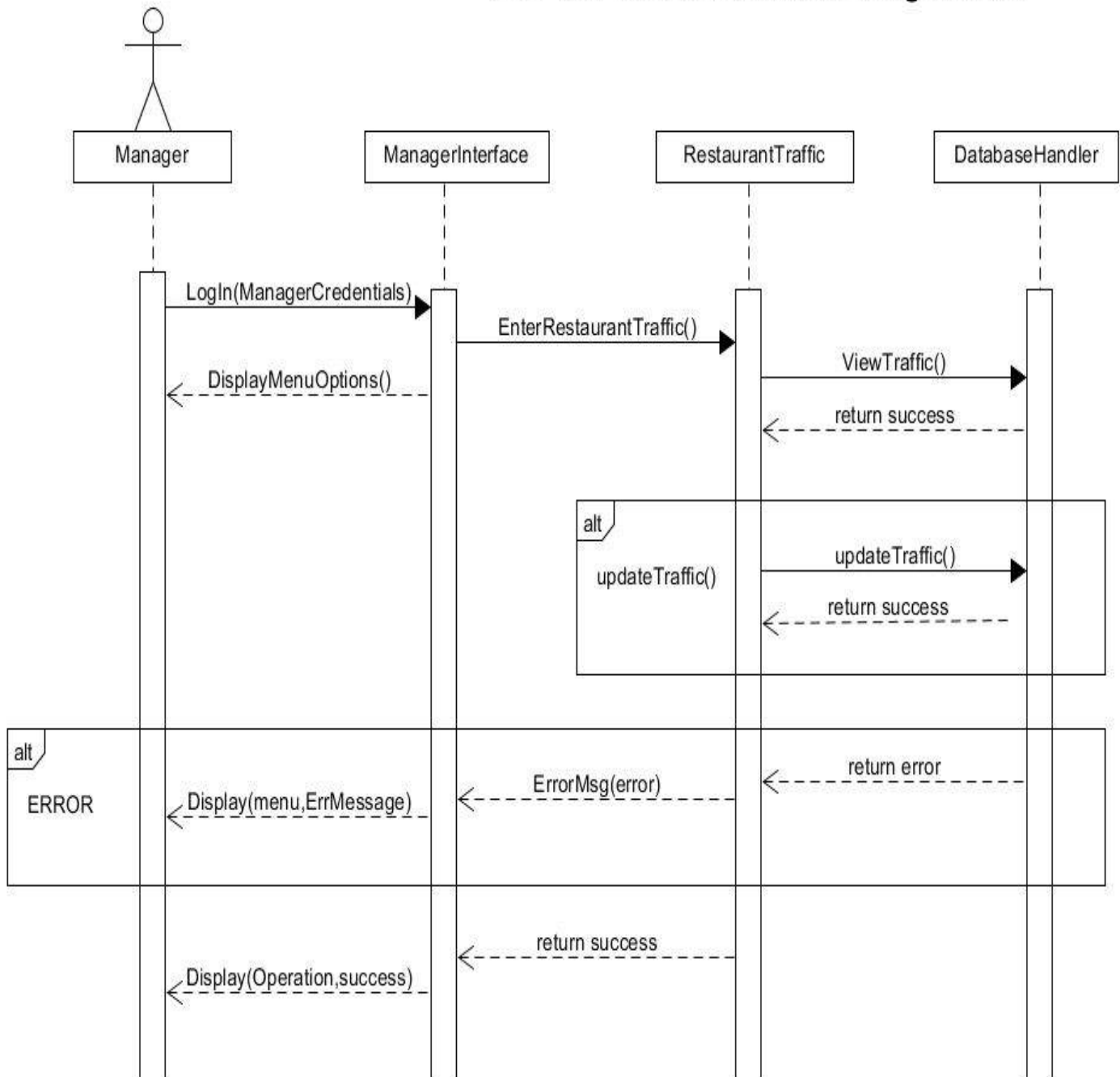


Explanation:

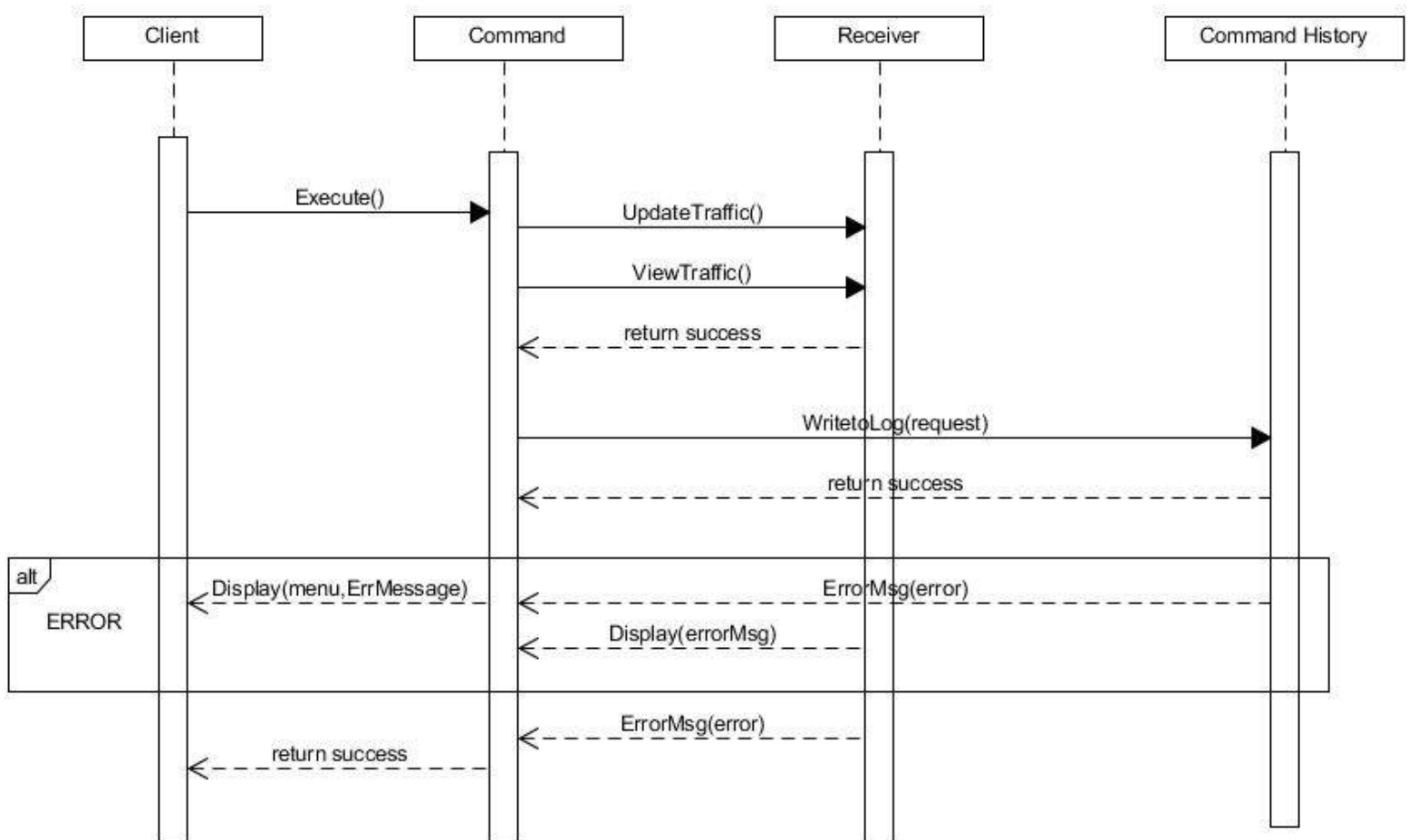
This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the employee management menu. From this point, the manager can decide to view the complete schedule of all employee shifts, to change the schedule of employee shifts, or to change a given employee's rate of pay. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.

In the manager interface, we utilized the Command design pattern which would enable us to use a central command file which would drive the entire program and facilitate the execution of certain actions as they are requested. In this new design, the manager is able to manage employee wages and scheduling by using view and edit functions. The application executes these functions individually. Every function will be activated when the client presses the corresponding button in the manager interface, in the employee scheduling menu and the financial management menu.

UC-6d: Restaurant Management



LITTLE BITS

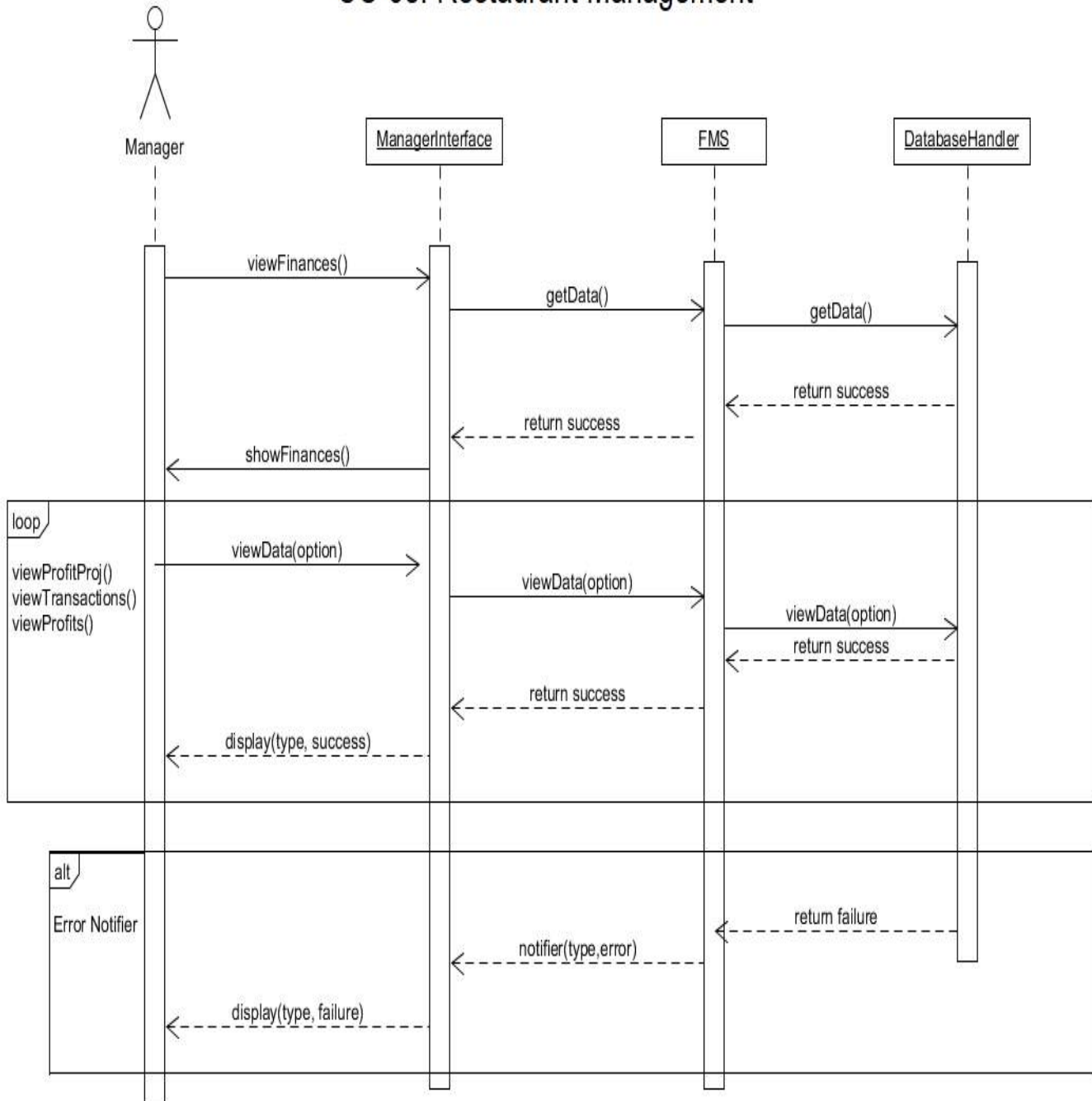


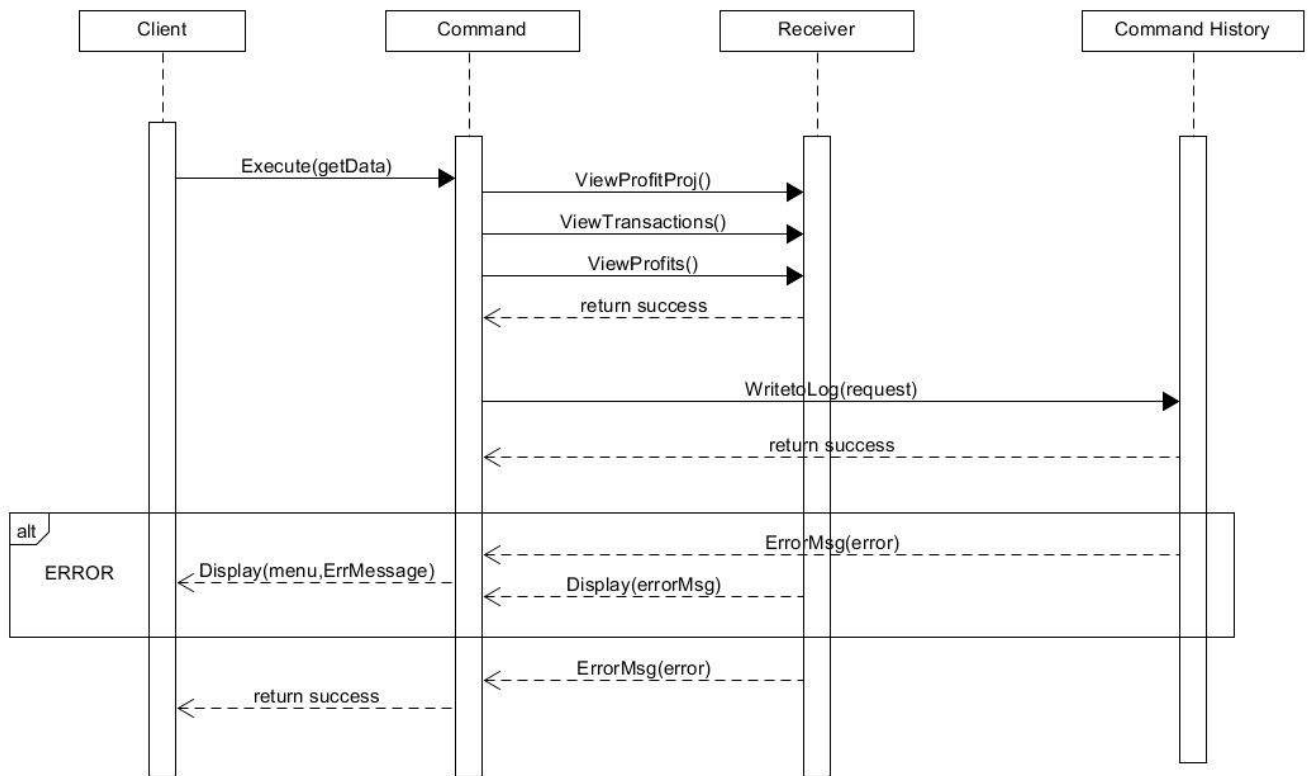
Explanation:

This diagram outlines the process the program goes through to display the inventory to the manager. Once the manager enters a valid set of credentials to login into the manager interface, they are allowed to view the floor traffic of the restaurant. The database will continually update the image of the floor traffic and continue to display said traffic to the requesting manager. If the database encounters any errors in execution of any of these manager-prompted functions, an error message will be returned to the manager detailing the error, such as attempting to remove an item that doesn't exist.

In the manager interface, we utilized the Command design pattern which would enable us to use a central command file which would drive the entire program and facilitate the execution of certain actions as they are requested. In the new design, only one function needed to be created to fulfill the functionality of the traffic interface, which is that the driving application will continuously update the traffic view of the restaurant every time a state change has been made to the state of the restaurant, whether that be a table's state changing from clean to dirty or changing from occupied to free, the application will take the request from the client and execute the function. The change to the restaurant state will then be saved in the command history log and normal operation of the application will resume.

UC-6e: Restaurant Management





Explanation:

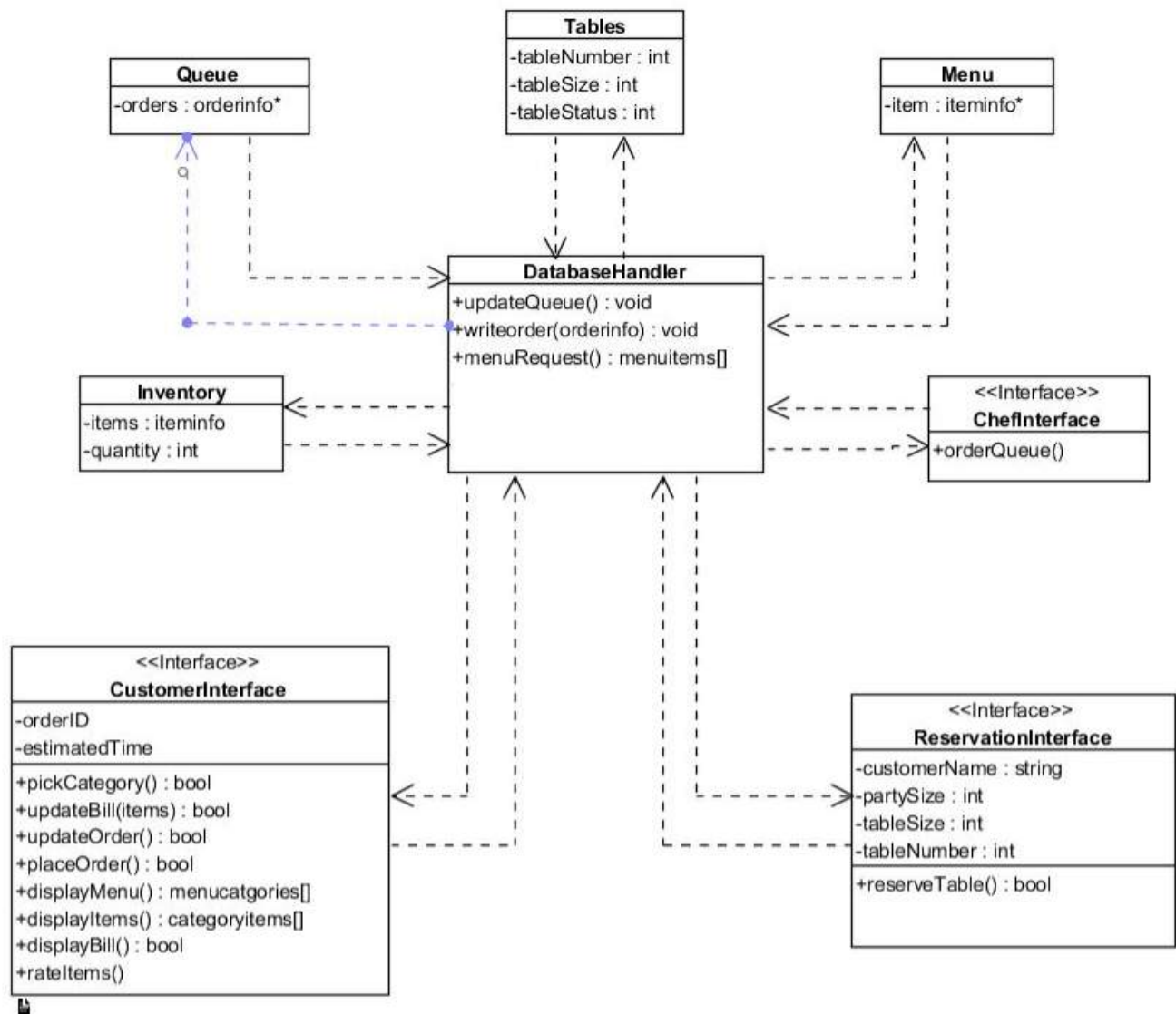
From the Manager interface, the manager can access the restaurant's financial records. The request is sent to the Financial Management System (FMS) where the manager can view profits, the transaction history, and a projection of future profits based on current data. The database is updated on a regular basis to ensure the manager receives up-to-date information.

In the manager interface, we utilized the Command design pattern which would enable us to use a central command file which would drive the entire program and facilitate the execution of certain actions as they are requested. In our new design, the flask application we coded will allow the manager to call different financial functions and they will be executed one-by-one by the application. The use of the Command design pattern specifically improved our design because this design was helpful and helped our program run smoothly. As soon as the client opens our application, the driving file will begin to execute commands as the client requests them to be done. Each financial function will be activated by a button press made by the client(manager) on the manager interface, in the finances menu.

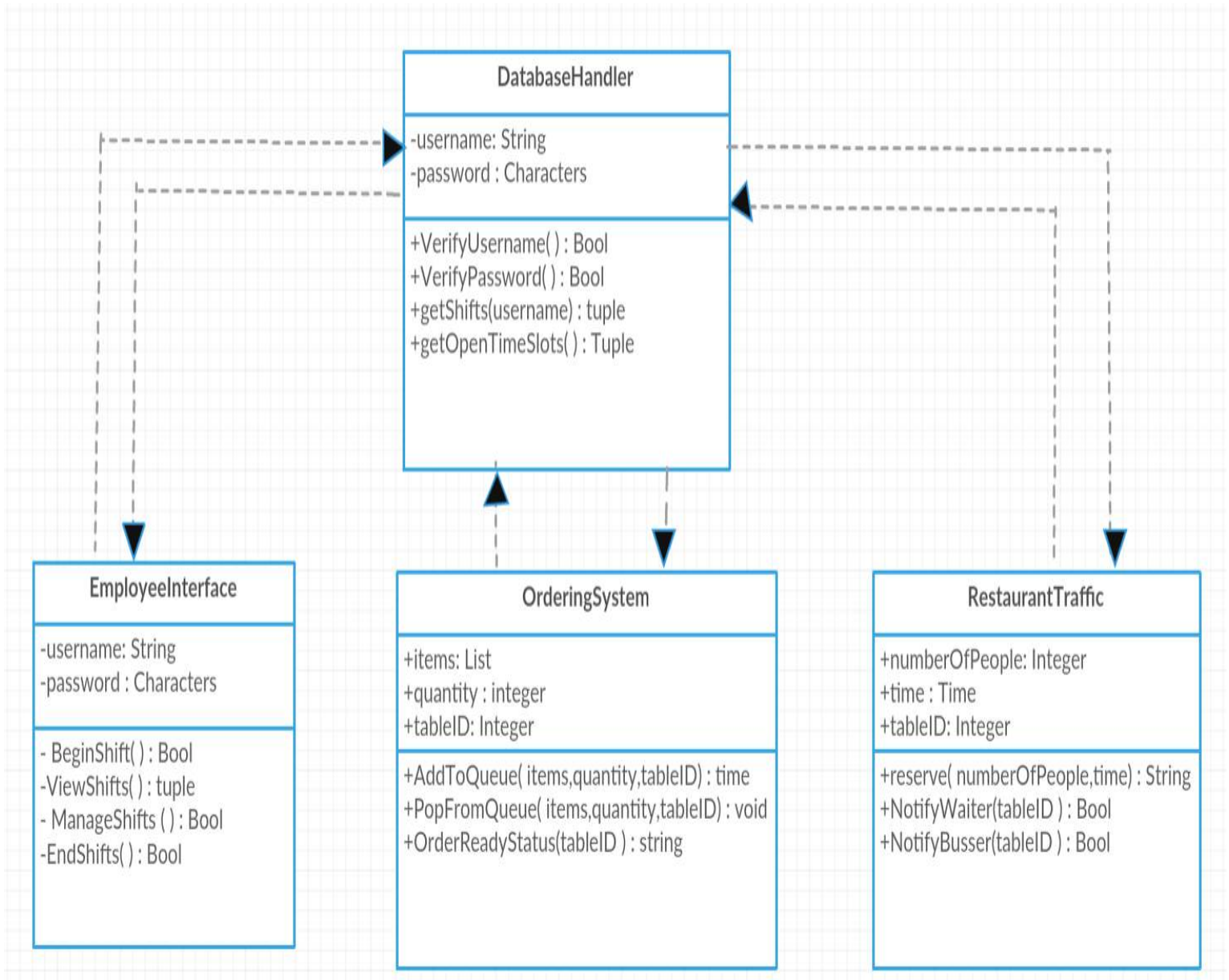
Class Diagram and Interface Specifications

Class Diagram

Customer:

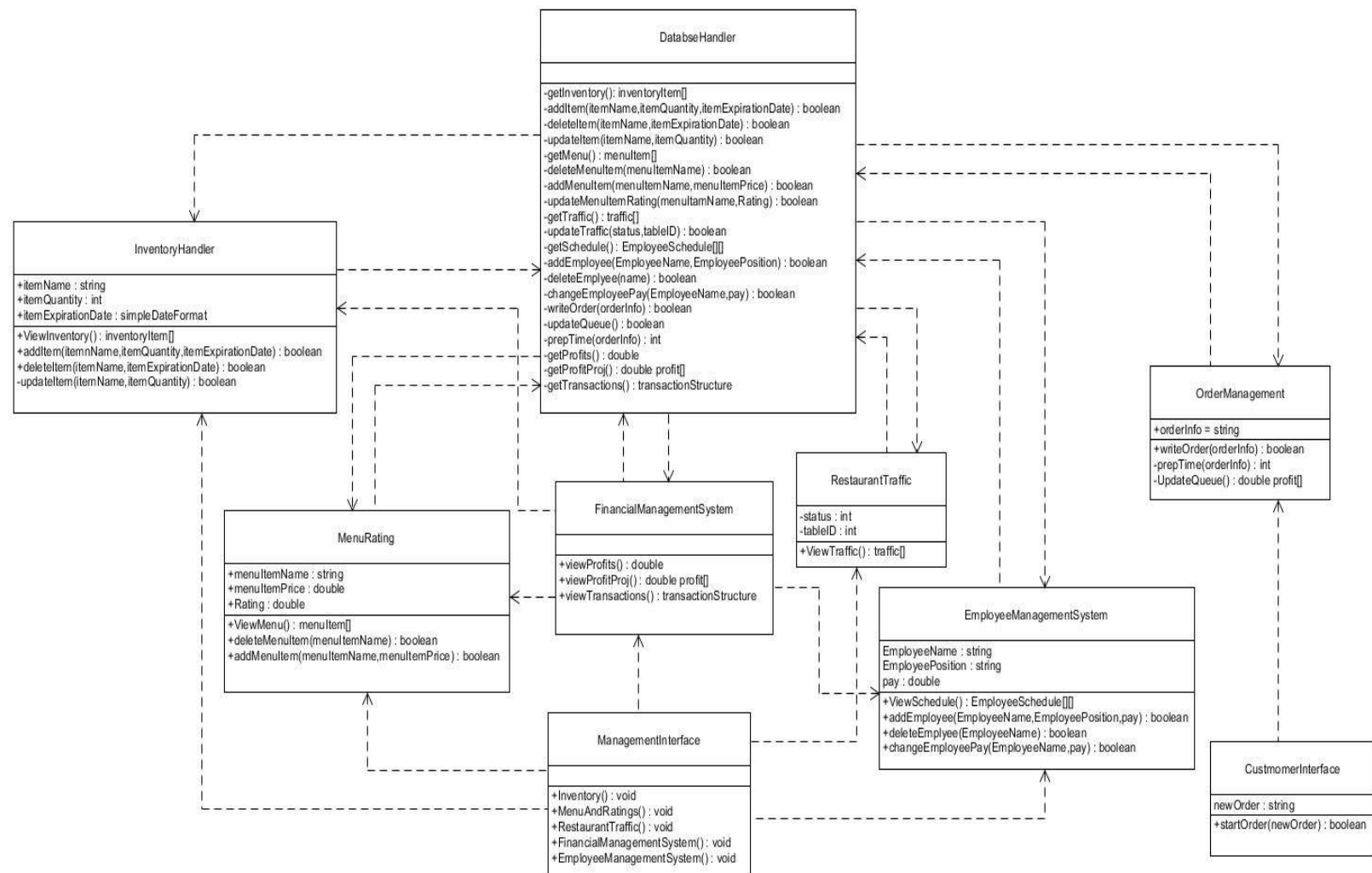


Employees:



LITTLE BITS

Manager:



Data Types and Operations Signatures

Customer:

1. Reservation Interface:

Attributes:

-customerName: string	A string value corresponding to the customer reserving the table.
-partySize: int	An integer value corresponding to the party size of the customer reserving.
-tableSize:int	An integer the shows the amount of people that can be seated on the table.
-tableNumber:int	An integer that represents the table reserved number.

Methods:

+reserveTable():bool	Method that allows the customer to reserve available tables according to the party size and availability.
----------------------	---

2. Customer Interface:

Attributes:

-orderId: int	An integer given to identify the customers placed order.
---------------	--

Methods:

+pickCategory(): bool	Method called to display the items of each category
+updateBill(*items): bool	Method that updates the customer's bill when customers add or remove items.
+updateOrder(): bool	Method that allows the customer to add or remove items from their bill.
+placeOrder(): bool	Method that allows the customer to place their order when satisfied.
+displayBill(): bill info*	Method that gives information about the items ordered, their individual price and the total bill.
+displayMenu(): bool	Method that displays the menus categories and options.
+displayItems(): bool	Method that displays the items of each category and their price.
+rateItems()	Method that allows the customer to rate the items that they ordered.

Employee:

1. **Employee Interface:** The employees' login, clock in, view their shifts, request for changes, clock out and logout using the employee interface

Attributes:

-username:string	An employee's username that's used to gain access to the employees portal.
- password:characters	A password that should match with the one that the employee created initially to log in.

Operations:

-BeginShift() : void	This method lets the employee clock in letting the manager know when they began their shifts so that the manager can keep track of their working hours and pay them accordingly
-ViewShifts() : tuple	The employee can view their shifts and keep track of their schedules. This method returns a tuple that consists of the date, day and time.
-ManageShifts() : Bool	The employee can add or delete shifts after getting the manager's approval..
-EndShift() : Bool	This method lets the employee clock out stating that his shift has ended.

2. **Ordering System:** This class places the customer's order in the order queue, pops the order from the queue after the chef has prepared it and notifies the waiter.

Attributes:

+items : list	A list of items that the customer orders from the menu
+quantity : integer	Total number of items that the customer ordered
+tableID : integer	The table ID of the customer that placed the order.

Operations:

+AddToQueue(items, quantity, tableID) : time	A method that adds the customer's order to the order queue and the returns the amount of time it's going to take to prepare.
+OrderReadyStatus(tableID) : string	This method returns a string that is sent over to the waiter's interface notifying him that the order is ready.
+PopFromQueue() : void	This method removes that order that has been prepared from the order queue.

3. **Restaurant Traffic:** This class manages the restaurant traffic and makes reservations.

Attributes:

+numberOfPeople : integer	The number of people that are present so that the reservations can be made accordingly
+time : time	Time when the customer places a reservation

Operations:

+reserve(numberOfPeople , time) : String	This method places a reservation for the customer and returns the time when the customer first reserved the table.
+NotifyWaiter(tableID) : Bool	This method notifies the waiter for any calls from the customer.
+NotifyBusser(tableID) : Bool	This method notifies the busser as soon as the customer processes the payment for him to clean the table.

4. **Database Handler:** This class is the main class that has dependency with the action of other classes. It verifies the login credentials, and gets shifts and open time slots.

Attributes:

-username : string	Employee's username to login into the system
-password : characters	Employee's password to login into the system

Operations:

+VerifyUser() : Bool	This method checks if the username is correct
+VerifyPassword() : Bool	This methods checks if the password is right
+getTimeShifts(username) : tuple	This method returns a tuple whose elements are the employee's date, day and time of when he's going to work.
+getTimeSlots() : tuple	This method returns a tuple with all open shifts containing the date, day and time.

Manager:**1. Database Handler:**

Methods	Description
<ul style="list-style-type: none">• <code>getInventory(): inventoryItem[]</code>	The Database Handler will retrieve all inventory data stored in the main restaurant database for use by the Inventory Handler
<ul style="list-style-type: none">• <code>addItem(itemName, itemQuantity, itemExpirationDate): boolean</code>	This function will allow for the manager to add an item to the restaurant inventory given a name, quantity of the item and the item's expiration date
<ul style="list-style-type: none">• <code>deleteItem(itemName, itemExpirationDate): boolean</code>	This function will allow for the manager to delete a given item from the restaurant inventory given a name and the item's expiration date
<ul style="list-style-type: none">• <code>updateItem(itemName, itemQuantity, itemExpirationDate): boolean</code>	This function will allow for the manager to update an item's quantity given the item's name and expiration date
<ul style="list-style-type: none">• <code>getMenu(): menuItem[]</code>	The Database Handler will retrieve all menu data stored in the main restaurant database for use by the Menu Management System
<ul style="list-style-type: none">• <code>deleteMenuItem(menuItemName): boolean</code>	This function will allow for the manager to delete a given menu item from the restaurant menu given a name
<ul style="list-style-type: none">• <code>addMenuItem(menuItemName, price): boolean</code>	This function will allow for the manager to add a given menu item to the restaurant menu given a name and price
<ul style="list-style-type: none">• <code>getTraffic(): traffic[]</code>	The Database Handler will retrieve all floor traffic data stored in the main restaurant database for use by the Traffic Monitoring System
<ul style="list-style-type: none">• <code>getSchedule(): employeeSchedule[][]</code>	The Database Handler will retrieve all employee scheduling data stored in the

	main restaurant database for use by the Employee Management System
<ul style="list-style-type: none"> • addEmployee(employeeName, employeePosition, pay): boolean 	This function will allow for the manager to add an employee to the restaurant employee database given a name, position and pay rate
<ul style="list-style-type: none"> • deleteEmployee(employeeName): boolean 	This function will allow for the manager to delete an employee from the restaurant employee database given a name
<ul style="list-style-type: none"> • changeEmployeePay(employeeName, pay): boolean 	This function will allow for the manager to change the pay rate of an employee given the employee's name
<ul style="list-style-type: none"> • writeOrder(orderInfo): boolean 	This function will send a customer's finished order to the order queue, where it will be interpreted by a chef to cook
<ul style="list-style-type: none"> • updateQueue(signal): boolean 	This function will receive a signal from the chef interface to update the queue based on the completion of the chef's assigned order
<ul style="list-style-type: none"> • prepTime(orderInfo): int 	Given order information, this function will return an estimated preparation time for the order
<ul style="list-style-type: none"> • getProfits(): double 	The Database Handler will retrieve all profit data stored in the main restaurant database for use by the Financial Management System
<ul style="list-style-type: none"> • getProfitProj(timePeriod): double profit[] 	The Database Handler will take all of the relevant financial information and input the information into the profit projection formula, which will in turn, return the set of profit projections for a given set of times
<ul style="list-style-type: none"> • getTransactions(): transactionStructure 	The Database Handler will retrieve the restaurant's entire transaction history stored in the main restaurant database for use by the Financial Management System

2. Inventory Management System:

Attributes	Description
<ul style="list-style-type: none">• itemName: string	The name of the inventory item will be stored as a string
<ul style="list-style-type: none">• itemQuantity: double	The quantity of a given item will be stored as a double
<ul style="list-style-type: none">• itemExpirationDate: simpleDateFormat	The expiration date will be stored as a date/time data type, which is available in Python libraries
Methods	Description
<ul style="list-style-type: none">• viewInventory(): inventoryItem[]	This function will take the data given by the Database Handler and send the array of inventory items to the Manager Interface to be displayed to the Manager
<ul style="list-style-type: none">• addItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will request the Database Handler to add an item to the restaurant inventory database given an item name, quantity, and expiration date
<ul style="list-style-type: none">• deleteItem(itemName, itemExpirationDate): boolean	This function will request the Database Handler to delete an item from the restaurant inventory database given an item name and expiration date
<ul style="list-style-type: none">• updateItem(itemName, itemQuantity, itemExpirationDate): boolean	This function will request the Database Handler to update an item's quantity in the restaurant inventory database given the item's name, quantity, and expiration date

3. Traffic Monitoring System:

Attributes	Description
<ul style="list-style-type: none">status: int	This variable will hold an integer value serving as an indicator as to what state a table is in (dirty, clean, occupied, etc.)
<ul style="list-style-type: none">tableID: int	This variable will hold a given table's ID number
Methods	Description
<ul style="list-style-type: none">viewTraffic(): traffic[]	This function will request traffic information from the Database Handler to be used to display the tabling diagram to the requesting Manager

4. Menu Management System:

Attributes	Description
<ul style="list-style-type: none">menuItemName : string	Name of the item on the menu
<ul style="list-style-type: none">menuItemPrice : double	Price of the menu item
<ul style="list-style-type: none">Rating : double	Number of stars given to a dish
Methods	Description
<ul style="list-style-type: none">ViewMenu() : menuItem[]	Displays the whole menu which will include the menu item, price and rating associated with it
<ul style="list-style-type: none">deleteMenuItem(menuItemName, menuItemPrice) : boolean	Deletes the item on the menu that has the corresponding name and price and returns true if it worked and false otherwise
<ul style="list-style-type: none">addMenuItem(menuItemName, menuItemPrice) : boolean	Adds an item to the menu specifying the name and price associated with it. If it works then it returns true and if it does not work then it will return false

5. Financial Management System:

Attribute	Description
<ul style="list-style-type: none">timePeriod : simpleTimeFormat	Amount of time starting from projection starting date to ending date that will display days,weeks,months,years
Methods	Description
<ul style="list-style-type: none">viewProfits() : double	Displays the amount of money made
<ul style="list-style-type: none">viewProfitProj(timePeriod): double profit[]	Displays the how much money the owner will make from starting date to the ending date and returns an array that contains the amount of money the owner is projected to make
<ul style="list-style-type: none">viewTransactions() : transactionStructure	Displays transactions made such as paying employees, cost of food that is in the inventory now, amount of money gained/lost each day. This will return a structure of transactions made so the owner is aware where his money is going

6. Management Interface:

Methods	Description
<ul style="list-style-type: none">Inventory() : void	Displays the inventory options
<ul style="list-style-type: none">MenuAndRatings(): void	Display the options for menu and ratings
<ul style="list-style-type: none">RestaurantTraffic() : void	Display options for Restaurant traffic
<ul style="list-style-type: none">FinancialMangementSystem() : void	Display financial options
<ul style="list-style-type: none">EmployeeMangement() : void	Shows different options for employees

7. Employee Management System:

Attributes	Description
<ul style="list-style-type: none">• EmployeeName : string	Name of the employee
<ul style="list-style-type: none">• EmployeePosition : string	What the role of the employee is
<ul style="list-style-type: none">• Pay : double	Amount of money paid to employees
Methods	Description
<ul style="list-style-type: none">• ViewSchedule() : EmployeeSchedule[][]	Displays the schedule of employees who will be working on what days. It returns a 2D array with names and times of when people will be working
<ul style="list-style-type: none">• addEmployee(EmployeeName, EmployeePosition,pay) : boolean	Adds an employee to the system specifying their name, position they will be working and their rate of pay. This method will return true if it worked and false otherwise
<ul style="list-style-type: none">• deleteEmployee(EmployeeName, EmployeePosition,pay) : boolean	Adds an employee to the system specifying their name, position they will be working and their rate of pay. This method will return true if it worked and false otherwise
<ul style="list-style-type: none">• changeEmployeePay(EmployeeName, pay) : boolean	Change the rate of pay for the specified employee. If it was altered correctly it will return true and false otherwise

8. Order Mangement System:

Attributes	Description
<ul style="list-style-type: none">• orderInfo : string	Lists the items that were ordered
Methods	Description
<ul style="list-style-type: none">• writeOrder(orderInfo) : boolean	Writes down everything that was ordered by reading in the order information returning true if it happened correctly and false if any problems came up
<ul style="list-style-type: none">• prepTime(orderInfo) : int	Time it takes to prepare the food
<ul style="list-style-type: none">• UpdateQueue() : boolean	Updates the queue of orders that are placed

9. Chef Interface:

Attributes	Description
<ul style="list-style-type: none">• Signal : int	Signal that tells the system to update the queue
Methods	Description
<ul style="list-style-type: none">• updateQueue(orderInfo) : int	Updates the queue of orders. The chef will remove the order from the queue once it has been done. Sends the signal to update the queue.

10. Customer Interface:

Attributes	Description
<ul style="list-style-type: none">• newOrder : string	Describes what the new order is
Methods	Description
<ul style="list-style-type: none">• startOrder(newOrder) : boolean	Writes down everything that was ordered by the customer and sends it to the queue.

Traceability Matrix

Customer:

	Software Classes		
	Order	OrderItem	Check
Domain Concepts			
Queue	X		
Tables	X	X	X
Menu	X	X	X
Inventory		X	
ChefInterface			
ReservationInterface			
CustomerInterface	X		X
DatabaseHandler		X	X

The Concepts were derived very simply from the idea of a restaurant in sync with its customers. Everything is connected one way or another and it can be told by simply looking at the names of the classes. For example: The matrix shows that for ordering purpose we use queue, tables and the menu. We need a time interface for queue, all the user interfaces as well as the database handler. For ordering an item, we use the data such as tables, menu, inventory, and the database. And lastly, when a customer leaves, we check for the table number, the items he ordered, his user interface for the ratings and reviews and the database data.

Employees:

Classes → Domain concepts ↓	Chefs Interface	Customer Interface	Table Management	Order Management	Employee Task Management
Order Status	X	X		X	
Order Display		X		X	
Order Queue	X	X			
Clean Table			X		X
Table Record	X		X	X	X

The Domain Concepts came from simply just taking a look at the various different functions of our system. We saw what we needed to accomplish and created various tasks. Order Status is done to give customers a description of how long their food would take and in what stage it is in. Order Display displays the Order Status to the customer as well as the waiter. Order Queue is for the chef to see what order the orders are placed so he can make the food in the correct order it came in. This will affect Order Status which in turn will affect Order Display. Clean Table is there so we can properly manage tables. Managing tables entails cleaning, marking tables dirty, and also will be needed in customer reservations. Table Record is made for the waiters to mark a table empty and dirty.

LITTLE BITS

Manager:

Classes → Domain concepts ↓	Database Handler	Inventory Handler	Financial Management System	Menu Ratings	Manager Interface	Restaurant Traffic	Employee Management System	Order Management	Customer Interface
Controller	X	X	X	X	X	X	X	X	X
Page Maker			X	X	X	X	X	X	X
Interface Page					X				X
Info Changer	X	X	X	X			X		
Notifier	X	X			X	X			
Database Connection	X	X	X	X	X	X	X	X	X
Investigation Request	X		X	X			X	X	
Traffic Display						X			
Predictor	X		X						

The Controller is essential to all parts of the program, as it coordinates everything and allows the different classes to communicate with each other. The Page Maker displays the UI for all of the relevant classes, so it is connected to everything that has a UI. The Interface Page allows users to interact with the program, so it is a part of any interface. The Info Changer is related to any part that allows saved data to be updated, this includes employee records, inventory items, menu items, etc. The Database Connection is another essential part of the system, as all aspects classes need access to the database. Notifier shows messages to the user, so it is connected to interface changes and the database. The Investigation Request has been expanded to anything that may request information from the database. Traffic Display, only relates to the restaurant traffic monitor, but it is accessed by both employees and the manager. The Predictor makes financial predictions based off of information stored on the database, so it is connected to both the financial management system and the database itself.

Design Patterns

Customer:

We use a modified builder design pattern for our pages. While the original builder pattern separate the construction of a complex object from its representation, allowing the same construction process to create various representations. Our modified pattern separates the construction of one page from its representation, allowing the same template to be used to create various pages. We chose to use this pattern to unify the user interface and make it easier for the customer to understand.

We also use our own invented design pattern. We call it DB pattern. Variables that are shared between more than one page (or part of the system) are loaded into the database and retrieved from it. We chose database vs other forms of communication because it is cheaper and allows us to avoid using cookies. It is also less vulnerable to errors

Employees:

The design patterns that all class diagrams use are behavioral design patterns. These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects, that is, communication between the chef's terminal and the employee's terminal; communication between the busser's and waiter's terminal; communication between the chef and the customers' interfaces; communication between the waiter's and the customers' interfaces.

We will be using the command pattern in this case because when a customer places an order, the chef takes the order (command) and prepares the items. These patterns help identify the control flow of the algorithm and highlights the parts where validation is necessary

Manager:

We used the command design pattern for all our design patterns because they focus on the classes' objects communication. Command patterns focus on communication between objects, for example, the managers computer and the database. The command pattern will be especially useful when the manager sends, for example, a new inventory item to the database to be saved.

Object Constraint Language

Customer:

Reservation Interface

Invariants:

The information entered should be true.

Name:string==True

TableNumber:int==valid(i.e within 10 tables)

Precondition:

The user must know his name,party size and the table number he wants to book.

partySize():int

tableNumber():int

Postcondition:

Once the customer has reserved a table he can either wait for the table to be emptied if busy or just go up to the table reserved and occupy it.

endReservation(): bool

Customer Interface

Invariants:

Menu is placed in front of customer, ordered items have to be on menu

orderInfo:String == true

Precondition:

At least one item has to be in menu for item to be placed

+items : list

quantity : integer

pickCategory(): bool

placeOrder():bool

Postcondition:

Once the order is placed the customer can either update the items he ordered and pay the bill.

updateOrder():list

displayBill(): list

Employees:

Employee Interface

Invariants: Login information must be true

Username:string == true

Password:character == true

PreConditions: Once logged in the employees can view shift, add shifts, delete shifts with approval by manager

BeginShift() : void

ViewShifts() : tuple

-ManageShifts() : Bool

PostConditions: Once the employee is done with shift, they can clock out and will be logged out

EndShift() : Bool

Ordering System

Invariants: Menu is placed in front of customer, ordered items have to be on menu

orderInfo:String == true

PreConditions: At least one item has to be in menu for item to be placed

+items : list

quantity : integer

AddToQueue((items, tableID)

PostConditions: Once order is complete, remove from queue

PopFromQueue() : void

Restaurant Traffic

Invariants: Number of people at table has to be true and a table has to be empty for it to be reserved

numberOfPeople : integer < tableSeatingNumber

PreConditions: Customers place reservation and shows time when placed. Whatever time is first, will be seated

reserve(numberOfPeople , time) : String

PostConditions: Payment is processed, and table is marked dirty and notifies busser

NotifyBusser(tableID) : Bool

Database Handler

Invariants: Login information must be input

Username:string == true

Password:character

PreConditions:

VerifyUser() : Bool

VerifyPassword() : Bool

PostConditions: Shows open shifts after logged in

+getTimeSlots() : tuple

+getTimeShifts(username) : tuple

Manager:

Manager Interface:

Invariants: Manager Username and Password must be correct

PreCondition: When manager successfully logs in manager would be able to see different options for what he wants to do.

Inventory() : void

Menu&Ratings(): void

RestaurantTraffic() : void

Finances() : void

EmployeePortal() : void

PostCondition: manager successfully seen and/or edited the information successfully

Menu Interface

Invariants: Manager Username and Password must be correct

PreCondition: When manager logs in successfully and goes into menu options, the manager is able to view and edit information regarding the menu.

Edit menu items:

addMenu(Category, ItemName, ItemPrice, IngredientName, Quantity, UnitofMeasure) : Bool

delMenu(Category, itemName) : Bool

Updmenu(Category, itemName, itemPrice) : Bool

menuPage() : MenuTable

PostCondition: manager successfully seen and/or edited the information successfully

Inventory Interface

Invariants: Manager Username and Password must be correct

PreCondition: When manager logs in successfully and goes into inventory options, the manager is able to view and edit information regarding the inventory.

Edit Inventory:

addInv(ItemName, Quantity, UnitOfMeasure, ExpirationDate, CostPerUnit) : Bool

delInv(ItemName,ExpirationDate)

updInv(ItemName, Quantity, UnitOfMeasure, ExpirationDate, CostPerUnit) : Bool

inventoryPage() : inventoryItem[]

PostCondition: manager successfully seen and/or edited the information successfully

Finances Interface

Invariants: Manager Username and Password must be correct

PreCondition: When manager logs in successfully and goes into finances options, the manager is able to view information regarding the manager's financial situation.

Look at finances:

TransactionHistory() : TransactionStructure

profitProjection(TimePeriod) : projectionStruct

profitByWeek() : ProfitStruct

PostCondition: manager successfully seen and/or edited the information successfully

Employee Portal Interface

Invariants: Manager Username and Password must be correct

PreCondition: When manager logs in successfully and goes into employee options, the manager is able to view and edit information regarding his/her employees.

Look at Employee Portal:

addEmployee(Name, Position, DateofBirth, PhoneNum, Username, Password, RepeatPassword)
: Bool

deleteEmployee(FullName) : Bool

updateEmployee(FullName, Name, Position, DateofBirth, PhoneNum, Username, Password,
RepeatPassword) : Bool

ViewRoster() : RosterStruct

PostCondition: manager successfully seen and/or edited the information successfully

Database Handler

Invariants: Manager Username and Password must be inputted

PreCondition:

VerifyUser() : bool

VerifyPass() : bool

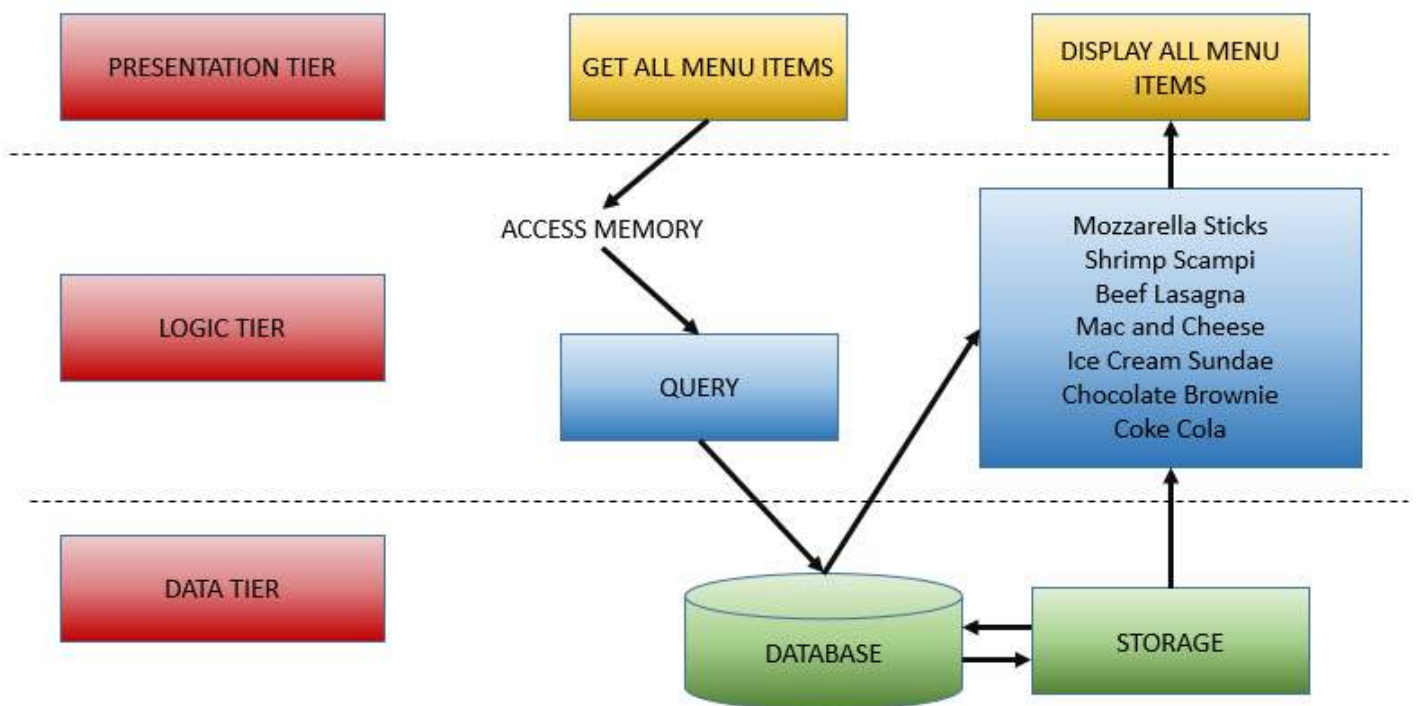
PostCondition: manager can view and edit information after information is verified

System Architecture and System Design

Architectural Styles

Customer:

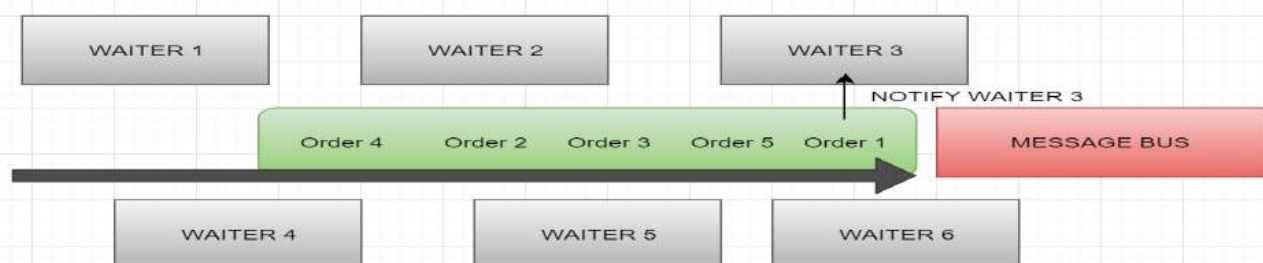
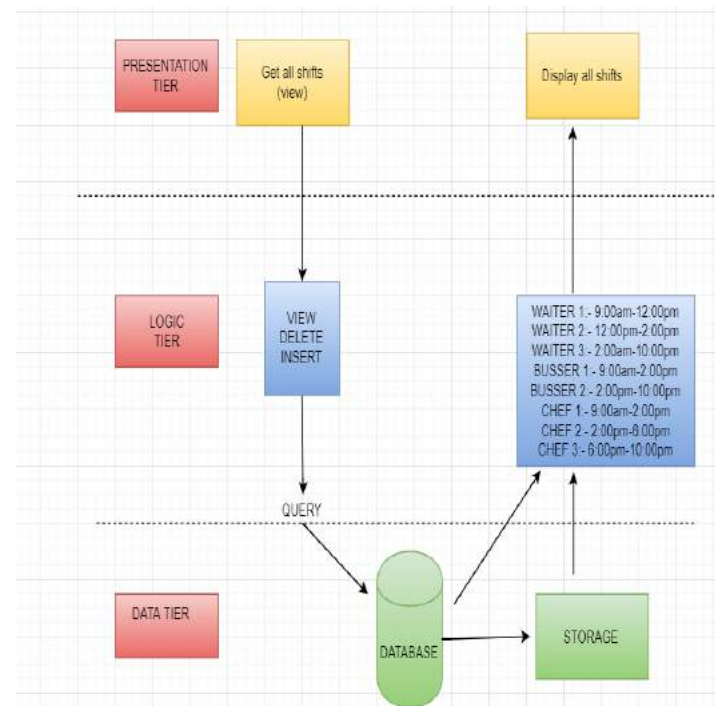
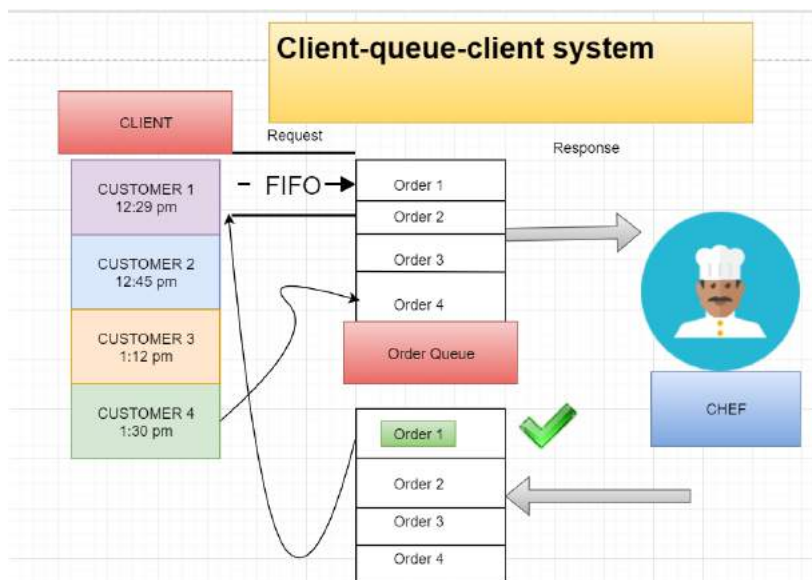
The customer's architecture styles are a combination of various forms of architectures. Starting from reservation, this is client server architecture when the client makes a choice what table they want based off the ones shown as available by the server. Following that customers would be fronted with the menu interface where the layered architecture comes into play. After finishing the meal, the customers deal with payment for the meal and rating of their experience. Paying for the meal is component based architecture because it is dependent of the customer's choice for what form of payment they wish to use. Rating the experience and food is client server and message bus architecture. When used in harmony, the customer is able to enjoy their experience.



LITTLE BITS

Employees:

The architectural style that we will be using is not limited to a single architectural style and is a combination of component based, service-oriented, multi-layered and client-server architecture. The ordering system uses the client-queue-client system architecture. Instead of communicating directly, the client and server exchange data with one another by storing it in a queue on the server. The waiters check the status of the order by using a message bus because a message bus specializes in transporting messages between applications (chef and the waiter). The login and logout as well as the tabling system uses a client server architecture with the clients being the waiters, customers, and bussers who update the current status of the table as independent clients and interact with a common server that records the statuses. The employees view their shifts and request modification using a multi-layered architecture/multi-tier with the presentation tier i.e. the employee interface requesting to view the shifts. The logic tier identifies employee's request to modify shifts and the data tier retrieves all the shifts.

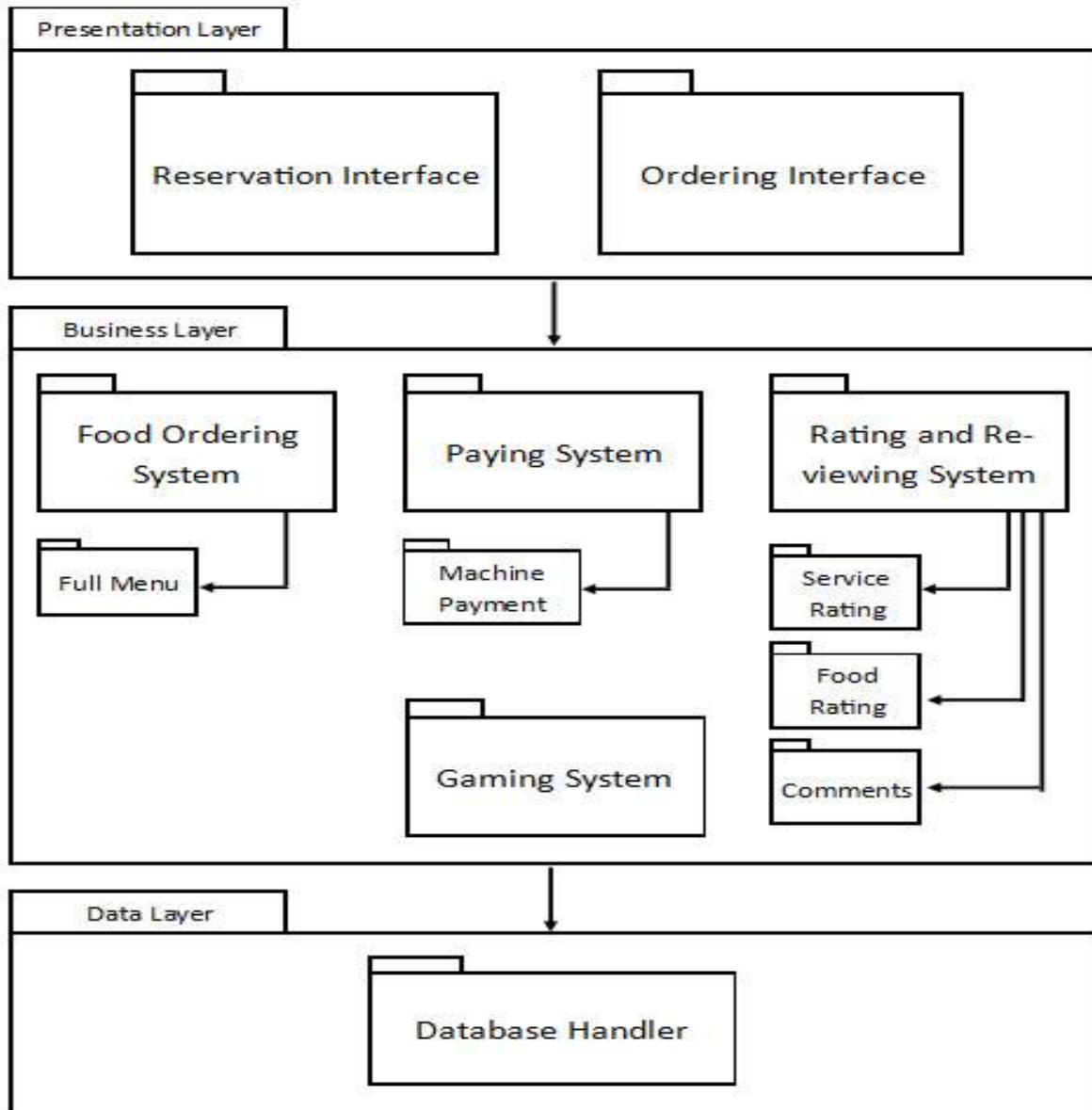


Managers:

The financial management system, the employee management system, and the inventory management system will need to use a multi-layered architecture for calculating and presenting the restaurant's financial information, employee scheduling and, the inventory system. The presentation tier will need to provide an easy to read interface for the managers. The logic tier will process requests from the manager, and later calculate and process the relevant information from the data tier. A client-server model will be necessary for storing the manager's login credentials as well as providing a view of the restaurant floor and every employee's status (i.e. viewing who is assigned to any given table). The menu rating and editing system will also run off of a multi-layered architecture. From the presentation tier, the manager sends a request to view ratings or edit the menu, the request is processed in the logic tier, which then retrieves the relevant information from the database. The information is processed, and sent back to the presentation tier for the manager.

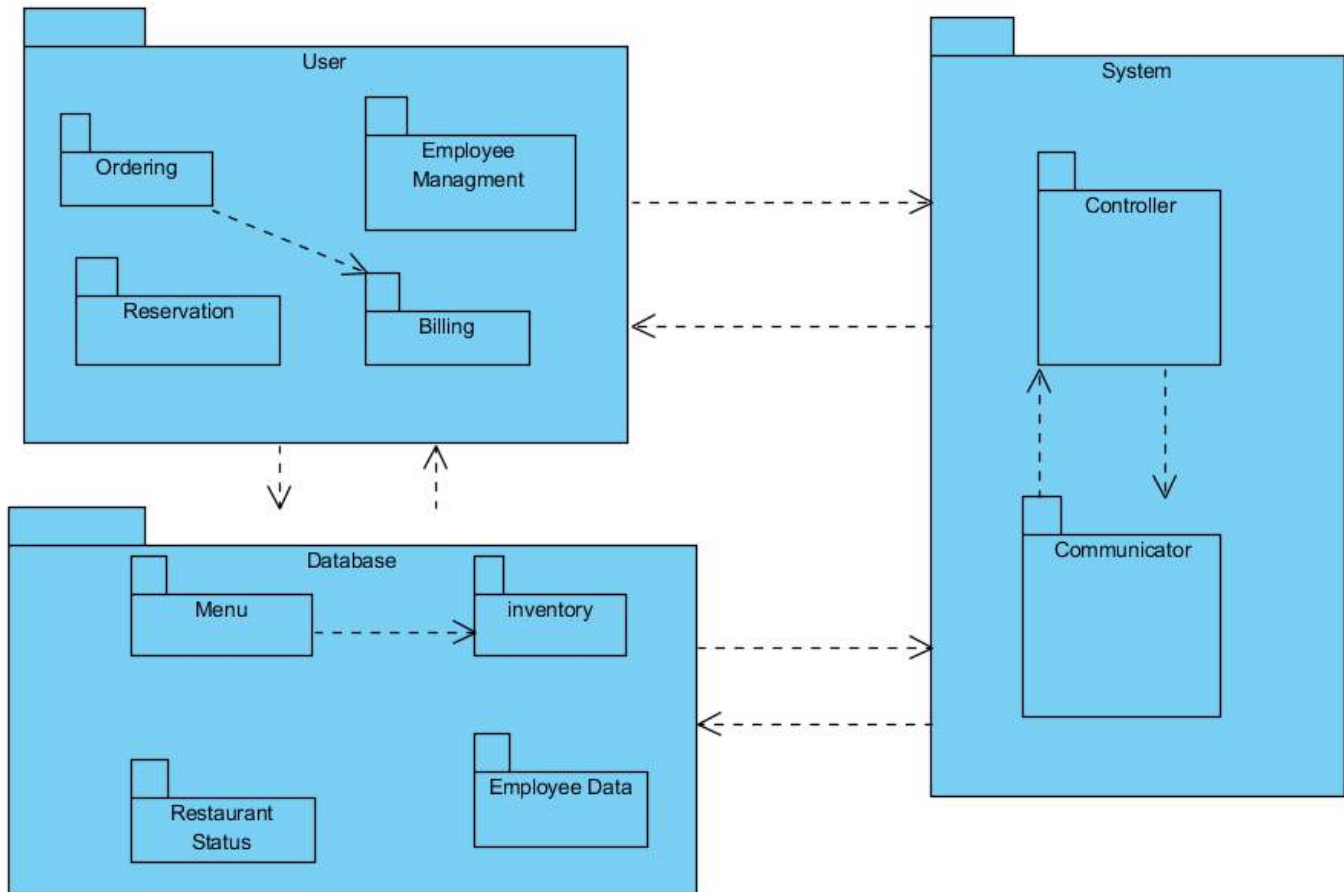
Identifying Subsystems

Customer:



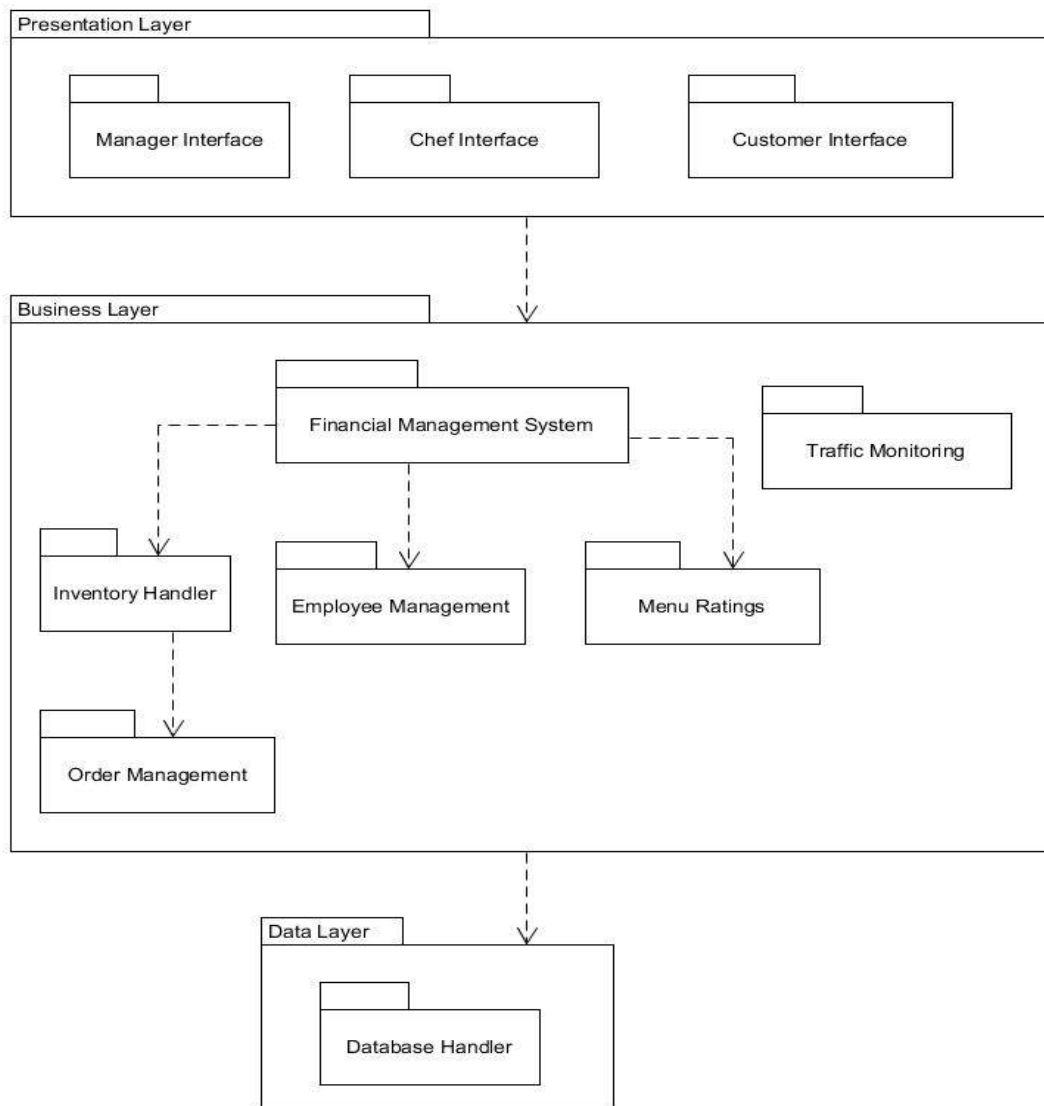
From arrival at the establishment to departure the customers is faced with two interfaces: Reservation Interface and Ordering Interface. In the Reservation Interface the customer is able to select where they wish to dine in the restaurant. The Ordering Interface brings in the various business layers. Food Ordering System allows the customer to select items off the menu that they wish to order. The Payment System deals with the electronic payment by the customer, this is unrelated to cash payments. The Rating and Reviewing System allows for the restaurant to be able to reanalyze their food and their employees based on their customers' experiences. All of the systems rely on the data found in the restaurant database, so they rely on the Database Handles to exchange data between various subsystems.

Employees:



We have a user package that includes all the packages that users need direct access to like ordering, reserving, billing and Employee management. We then have database where all our data is stored. This includes the menu, employee data, our inventory and the restaurant status: that is, which tables are available, how many workers are currently in the restaurant and all other data pertaining to the current status of the restaurant. Finally, there's the system package. This includes the controller and the communicator. All of our main packages have mutual dependencies and some of our sub-packages have dependencies as well.

Manager:



This UML package diagram describes the entire manager application within the larger restaurant automation software. The manager application is responsible for displaying relevant information to the manager, customer, and chef interfaces. The application utilizes six main subsystems to accomplish its functionality, which are the financial management system, the inventory management system, the employee management system, the menu management system, the order management system, and the traffic monitoring system. Each of these systems rely on data from the main restaurant database, and thus they rely on the database handler to exchange data between the subsystems and the database.

Mapping Subsystems to Hardware

Customer:

Due to the nature of client-server architecture, the hardware hierarchy of the system is straightforward. Understanding that the system is running on the client-server model, the server runs on the master computer which has the database and all applicable data. Customers have tablets through which they access the restaurant automation application which all contain the communicator, interface processing, and controller. From a customer's standpoint, this structure allows for each user to control what he/she wants to order and keeps the traffic/transfer of data simple.

Employees:

Although our architecture style is complex as it combines multiple styles, our hardware mapping is quite simple. There is a master computer that stores our database and runs our main program. Every table has a "table tablet" that can only access the customer interface of our system and runs all the related process. All waiters have "waiter tablets" that can only access the waiter interface of our system and runs all the related process. Similarly, every chef has a tablet that can only access the chef interface of our system and runs all the related process.

Manager:

Aside from having a desktop that has full access to all subsystems, managers will also have a portable tablet that will have some of the subsystems on it. For example, the financial management system will not be accessible on the tablet for privacy and security reasons, but managers will be able to pull up the restaurant floor traffic view. On the tablet, managers will have access to the restaurant floor traffic view, inventory view, and employee scheduling view. Both the desktop and the tablet will have access the database, which is on a server. On the desktop, managers will have access to the financial management system, which includes profits, employee wages, and profit projections. The desktop will also have access to the inventory manager, which will allow the manager to view and update the inventory. Managers will have the ability to edit menu options and view the ratings of menu dishes. Managers have the ability to create and edit employee scheduling and information from the desktop as well. The restaurant floor view will be the same on the desktop as it is on the tablet. It is notable that everything that is available on the tablet will also be available on the desktop.

Persistent Data Storage

Customer:

The system stores information which needs to be longstanding. In particular, we document details which make management's work easier and also provide clean documentation for future reference. This includes reviews and comments given by the customers, changes made to restaurant proceedings (i.e. the menu), ratings provided by the customers. In SQL terms, we have tables and fields which are to hold this data permanently with backups planned. Backups ensure long term reliability and establish that maintenance of the data is of paramount importance which is the backbone of persistent data storage acting highly appropriate for our purposes.

Employees:

For the employees to be able to view their shifts at any time of the day, we need to save the data that outlives a single execution of the system. Python allows us to permanently store content. A file can contain structured data or unstructured. Since we will be recording the shifts systematically, we will be using a structured database. The database would include columns as the days of the week and the rows as the entry of each employee. The SQL Database will be required for adding, deleting and updating the records which requires manager's authentication and approval. The following diagram given an idea of how the shifts are visible on the employee's interface.

Staff	Mon 18	Tue 19	Wed 20	Thu 21	Fri 22	Sat 23	Sun 24
Employee 1	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 2							
Position							
Employee 3		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Supervisor		Supervisor			
Employee 4	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 5	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	
Employee 6		3:00 AM - 7:00 AM					
Position		Trainee					
Employee 7	3:00 PM - 7:00 PM	7:00 AM - 3:00 PM		3:00 PM - 7:00 PM		7:00 AM - 3:00 PM	
Position	Manager	Manager		Manager		Manager	
Employee 8	3:00 AM - 7:00 AM				7:00 AM - 3:00 PM		
Position	Sales Associate				Sales Associate		
Employee 9		3:00 AM - 7:00 AM	3:00 PM - 7:00 PM	3:00 PM - 7:00 PM		11:00 AM - 3:00 PM	
Position		Cashier	Cashier	Cashier		Cashier	
Employee 10							
Position							
Employee 11	3:00 AM - 7:00 AM		3:00 PM - 7:00 PM				
Position	Manager		Manager				
Employee 12							
Position							
Employee 13		11:00 AM - 3:00 PM		11:00 AM - 3:00 PM			
Position		Sales Associate		Sales Associate			
Employee 14	3:00 AM - 7:00 AM		7:00 AM - 3:00 PM				
Position	Trainee		Trainee				
Employee 15	3:00 AM - 7:00 AM					11:00 AM - 3:00 PM	
Position	Assistant Manager					Assistant Manager	

Managers:

Our system will need to outlive a single execution of the system given this software will be the core to the operation of a restaurant. The management side of the software will need to store login information for all employees and managers working at the restaurant, inventory information, menu information, and financial information for the restaurant, which will all be stored in separate sections of a large relational database. Login information will be stored in a private table, which will associate a given login username and password with a valid employee or manager object, which itself will contain all personal information pertaining to said employee or manager. Inventory information will be stored in a public table with each item being described by name, quantity, cost per unit, and low quantity threshold. Menu information will also be held in a public table described by name, price, and average customer rating. Financial information will be held in a private table containing the total transaction history of the restaurant, which is described by money lost/gained and date of transaction; daily earnings/losses are entered into this table as singular transactions rather than each customer order being entered as its own transaction.

Network Protocol

Our system will be running on multiple terminals/machines (for the chef, waiters, customers and bussers) and therefore we will require a communication protocol. Python provides two levels of access to network services, one of which is at a low level where you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. We will be using the HTTP protocol with port number 80 because we're using web pages to develop our software which is HTTP's main functionality. The python modules used are httplib, urllib, xmlrpclib. Writing our communication protocol in Python will allow for easy portability with the several other modules throughout the restaurant automation software that are also written in Python.

Global Control Flow

Execution Orderliness:

“Little Bits” is a procedure driven software. Practically everything executes in a linear fashion. In the restaurant business you have many different sub teams. For the sake of simplicity, we combined this into three teams. An employee team, a Customer team, and a Manager Team. The customer would enter the restaurant and push in the details (ex. Number of seats and time) allowing the program to seat him in a optimal position. Then the customer would make his order whether through his/her tablet or waiter’s tablet. The order will be received by the Chef and when it is ready will notify the waiter to take the order directly to the customer. When the customer finishes eating he will pay the bill whether through the tablet or waiter and thus marking the table dirty for the busboy to come clean it. This clearly shows that all actions are made in a linear fashion all depending on the previous action.

For the employee teams it is linear because all the various employees need to access all relevant systems with ease. All actions are based off of each other. When a customer places a reservation, the table management has to change and a waiter is notified a table they are assigned. Then once the order is placed it is sent to the order queue for the chef to make the orders. Then the order time is calculated and displayed. Finally, once a customer leaves, the table is marked dirty using a timer and a busser is notified. This is a simple explanation showing the linearity of the system.

For the Managers team, this system must be procedure-driven. It must execute in a linear fashion in order to maintain consistency. In a fast paced restaurant setting a manager will need to access everything with ease, so a system that changes with different events is not optimal.

Time Dependency:

This system is periodic as for everything done from being seated to paying the bill is done in a periodic manner. For the customer’s team, this is all time dependent because the time the customer makes the order places it in a queue for the chef to cook. Also calculation of wait time for the customer to receive the order is also on real time basis. For the employee’s section, there is a simple timer. The timer is started in real time when a customer pays their bill. Every five minutes a notification will be sent to the waiter, which the waiter confirms or ignores if the customer has left the table. Until the notification is accepted that the customer leaves, the timer will then repeat until it is completed. Finally, for the manager’s team, there are no timers in our system due to the fact our system is an event-response type. Their system does not depend on real time. The manager system does not need to be constantly updated with respect to time. All functions on the manager aspect of the applications depend on events initiated by relevant users.

Concurrency:

Little Bits will contain multiple threads, which will involve multiple systems running independently at once. Multiple customers will be placing orders at the same time which is why we need concurrency. The solution is to run multiple threads into the queue. Synchronization for the customer's team is not needed because each thread is independent of the other. For the employee's team, they will also contain multiple threads, which will involve multiple systems running independently at once. Multiple tables will be placing orders at one time and this will need to be sent to the Chefs tablet which is why we need concurrency. The solution is to run multiple threads into the OrderQueue. Synchronization for this team is needed because some threads such as the ones for the OrderStatus and OrderQueue will need to work together. For the manager's team it will use threads to manage login sessions between multiple managers that are accessing the system simultaneously. There will be synchronization so that one manager doesn't overwrite the other one. For example, if one manager is adding an item to the inventory database and another is removing or adding another item there will be locks imposed so that the data will not be overwritten.

Hardware Requirements

1. Ziosk Z-400 Tablet
 - a. Display: 1024x600 WSVGA LCD
 - b. Battery: 7.4 VDC Li-Ion polymer 8720 mAh custom rechargeable pack
 - c. Wireless: 802.11 a/b/g/n/ac plus Bluetooth 4.0 (LE)
 - d. Card Reader: 3DES DUKPT Encrypting Magnetic Stripe Reader
 - e. Description: The Ziosk Z-400 Tablet will need to have an eight inch diagonal, to hold a minimum of 1 GB of hard drive space, and to handle a network bandwidth of around 60 Kbps to handle the transfer order information to and from each tablet.
2. Dell Inspiron Desktop
 - a. Processor: 5th Generation Intel® Core™ i5-7700 processor (8MB Cache, up to 4.20 GHz)
 - b. Operating System: Windows 10 Pro 64-bit English
 - c. RAM: 8GB, 2400MHz, DDR4
 - d. Hard Drive: Dual Storage (3.5' 500GB HDD + 2.5' 128GB SSD)
 - e. Wireless: Dell Wireless 1707 Card 802.11bgn + Bluetooth 4.0
 - f. Video Card: NVIDIA® GeForce™ GTX 1050 with 2GB GDDR5 graphics memory

Algorithms and Data Structures

Algorithms

Table Reserving Algorithm:

As customers enter the restaurant they are asked to specify their party size among other things into a tablet. This algorithm uses the party size and compares it against the available tables to see which table should be designated to the customer. The algorithm utilizes a sorted list (sorted and prioritized by a first come first serve basis) to take input from and then compares the party sizes of all customers and find all the vacant and suitable tables. The customer gets to choose one among them according to its position in the restaurant (People preferring a view). If the party size is less than the table size and no other tables are available, then they are given the table. In case the party size is larger than the capacity of the vacant tables or if the tables which can accommodate the party size are busy, then the customers have to wait until they are emptied.

```
while (sorted list of customers does not equal 0)
{
    while (iterating through the list of available tables)
    {
        while (iterating through sorted list of customers)
        {
            if (size of table is equal to a party size of the customers)
            {
                //Highlight allowed tables the fit the party for the customer to
                reserve
                //Allow the customer to reserve and display reservation number
            }
            else if(size of table is greater than party size and no other party
            matches    table size)
            {
                //Highlight allowed tables the fit the party for the customer to
                reserve
                //Allow the customer to reserve and display reservation number
            }
        }
    }
}
```

Bill Payment Algorithm:

As customers place their order the bill amount gets added up according to the price and the quantity of the item ordered. When the customer wants to pay the bill, the total bill amount is displayed with three modes of payment. Based on the customer's preferred payment method, the algorithm matches the mode of payment selected by the customer with the payments options in the database and displays the corresponding payment page. On each page, the customer has the option to tip and rate the food and share his/her customer experience in place of the conventional method of payment of the bill.

```
while (bill of customer not equal to zero)
{
    if (payment method equals credit card)
    {
        // Display Tipping option according to the bill amount
        //calculate bill amount accordingly and display payment page on tablet
        // Acknowledge payment
        //Display rating option
    }
    if (payment method equals paypal)
    {
        // Display Tipping option according to the bill amount
        //calculate bill amount and display paypal page
        // Acknowledge payment
        //Display rating option
    }
    if (payment method equals cash)
    {
        //send notification to the table waiter to assist the customers
        //Display rating option
    }
}
```

Order Status Algorithm:

```
class OrderQueue:
    def init (self):
        self.orderItems = [ ]
    def isEmpty(self):
        return self.orderItems == [ ]
    def enqueue(self, orderItems)
        self.orderItems.insert(0,orderItems)
    def dequeue(self):
        return self.orderItems.pop()
    def size(self):
        return len(self.orderItems)

q = OrderQueue()
def orderStatus:
    while(q.size!=0)
        if (q.isEmpty == True):
            print ("No orders in progress")
        else:
            if (q.size ==1)
                print ("Cooking...")
                setTimer()
                //as soon as the timer goes off, print ("Order is ready")
                q.dequeue()
                //notify waiter
                print ("No more remaining orders")
            else:
                print ("order number 1 out of q.size() is cooking")
                print ("(q.size-1) orders are currently in progress")
                setTimer()
                //as soon as the timer goes off, print ("order is ready")
                //notify waiter
                q.dequeue()
                q.enqueue(order #x)
```


Viewing shifts Algorithm:

The employee logs in and his profile is accessed from the database.

The user interface has different tabs one of which include “view shifts” and when the employee clicks on it, his shifts are accessed.

```
global login_user
conn = sqlite3.connect('employee shiftsdb.db')
print("*****")
print(login_user)
v=(login_user,)
cursor = conn.execute("SELECT * from EMPLOYEESHIFTS WHERE
NAME == ? COLLATE NOCASE",v)
return render_template("empviewshift.html", rows = cursor.fetchall())
```

Add shifts Algorithm:

Once the employee logs in, he views his shifts and can send a request to the manager to add his shifts.

```
global login_user
if request.method=='POST':
    shiftstart = request.form['ssa']
    spa = request.form['spa']
    sea = request.form['sea']
    epa = request.form['epa']
    conn = sqlite3.connect('employeeeshiftsdb.db')
    cursor = conn.execute("SELECT * from EMPLOYEEESHIFTS")
    for row in cursor:
        conn.execute("INSERT INTO EMPLOYEEESHIFTS
(NAME,ROLE,DAY,SHIFTSTART,SHIFTEND,STARTPERIOD,ENDPERIOD) \
VALUES (?,?,?,?,?,?,?)",(fname, role,
day,shiftstart,spa,sea,epa));
    conn.commit()
    print "Records created successfully"
    conn.close()
```

Delete shifts algorithm:

Once the employee logs in, he can send requests to the manager to delete his shifts.

```
global login_user
if request.method=='POST':
    fname = login_user
    role = request.form['role']
    day = request.form['daya']
    shiftstart = request.form['ssa']
    spa = request.form['spa']
    sea = request.form['sea']
    epa = request.form['epa']
    conn = sqlite3.connect('employeeeshiftsdb.db')
    cursor = conn.execute("SELECT * from EMPLOYEEESHIFTS")
    for row in cursor:
        conn.execute("DELETE FROM EMPLOYEEESHIFTS
(NAME,ROLE,DAY,SHIFTSTART,SHIFTEND,STARTPERIOD,ENDPERIOD) \
VALUES (?,?,?,?,?,?,?)",(fname, role,
day,shiftstart,spa,sea,epa));
    conn.commit()
    print "Records deleted successfully"
    conn.close()
```

Chat System:

```
def messageRecived():
    print( 'message was received!!!' )

@socketio.on( 'my event' )
def handle_my_custom_event( json ):
    print( 'received my event: ' + str( json ) )
    socketio.emit( 'my response', json, callback=messageReceived )
```

JAVASCRIPT CODE:

```
var socket = io.connect( 'http://' + document.domain + ':' + location.port )
// broadcast a message
socket.on( 'connect', function() {
    socket.emit( 'my event', {
        data: 'User Connected'
    } )
    var form = $( 'form' ).on( 'submit', function( e ) {
        e.preventDefault()
        let user_name = $( 'input.username' ).val()
        let user_input = $( 'input.message' ).val()
        socket.emit( 'my event', {
            user_name : user_name,
            message : user_input
        } )
        // empty the input field
        $( 'input.message' ).val( '' ).focus()
    } )
} )
// capture message
socket.on( 'my response', function( msg ) {
    console.log( msg )
    if( typeof msg.user_name !== 'undefined' ) {
        $( 'h1' ).remove()
        $( 'div.message_holder' ).append( '<div class="msg_bbl"><b style="color:
#000">'+msg.user_name+'</b> '+msg.message+'</div>' )
    }
} )
```

Data Structures

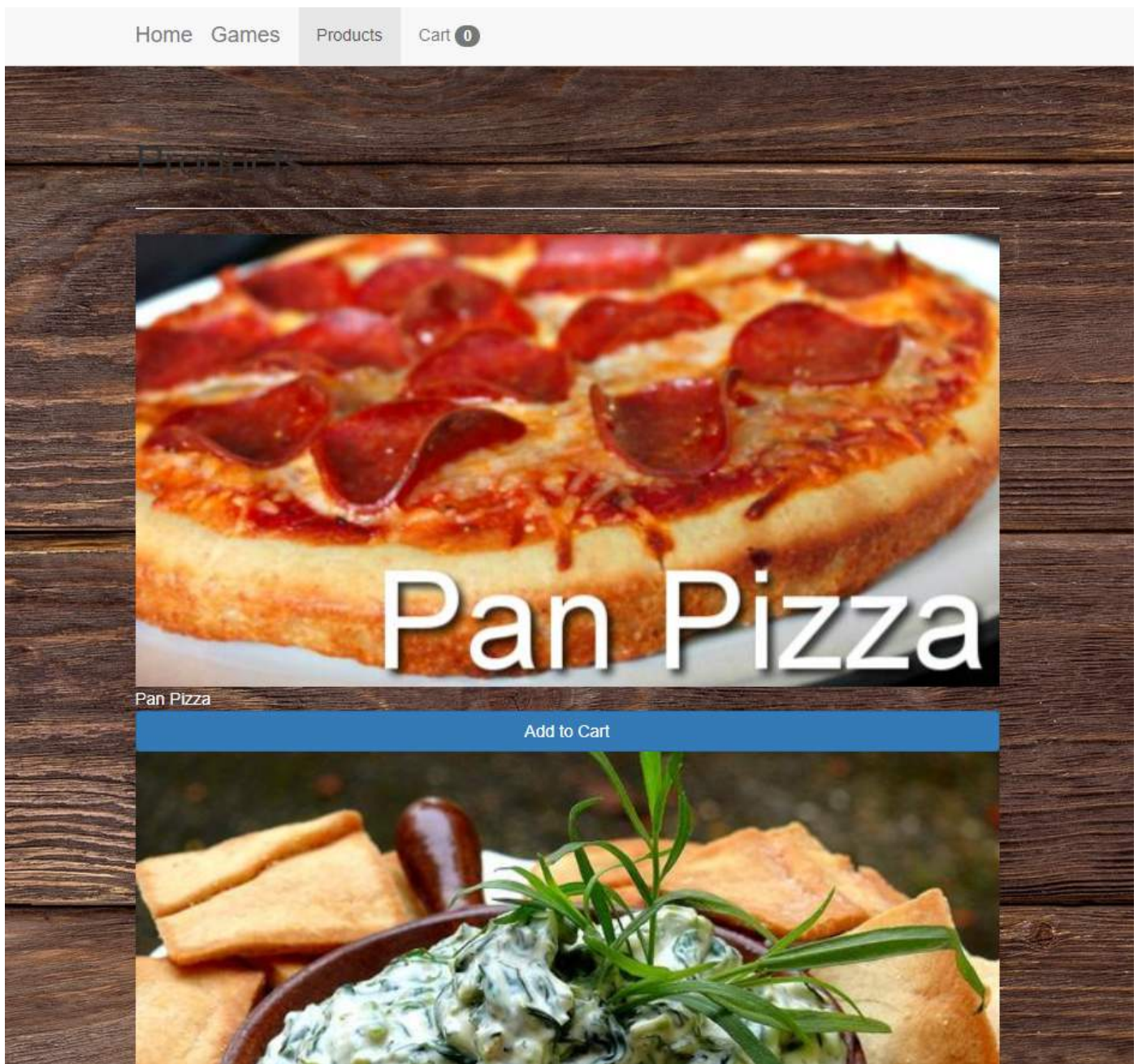
For the customer section, they will use a variety of data structures. For viewing and identifying the occupied tables for the reservation interface, a queue is storing information needed (such as availability) transfers this information to the customer reserving. Database is also used for storing a variety of information. The database stores table information (table size, number), it also stores the menu items and their quantities. For the employees' section, all orders placed by a customer are added to a queue which is accessible by the chef. The queue will be implemented using the data structure called priority queue where the chef can assign priorities manually and use the "order status algorithm" to notify the customers. Finally, for the manager segment, they use data structures in order to increase performance. They will be using arrays for now to make sure the system works for simplicity purposes. Then later on, once everything is working, they will possibly switch to linked lists which are similar to arrays but are more dynamic and help save more space and be more efficient, or switch to trees to increase the speed of our system. This will be up for discussion later after the system is fully functioning and operational for the design.

User Interface Design and Implementation

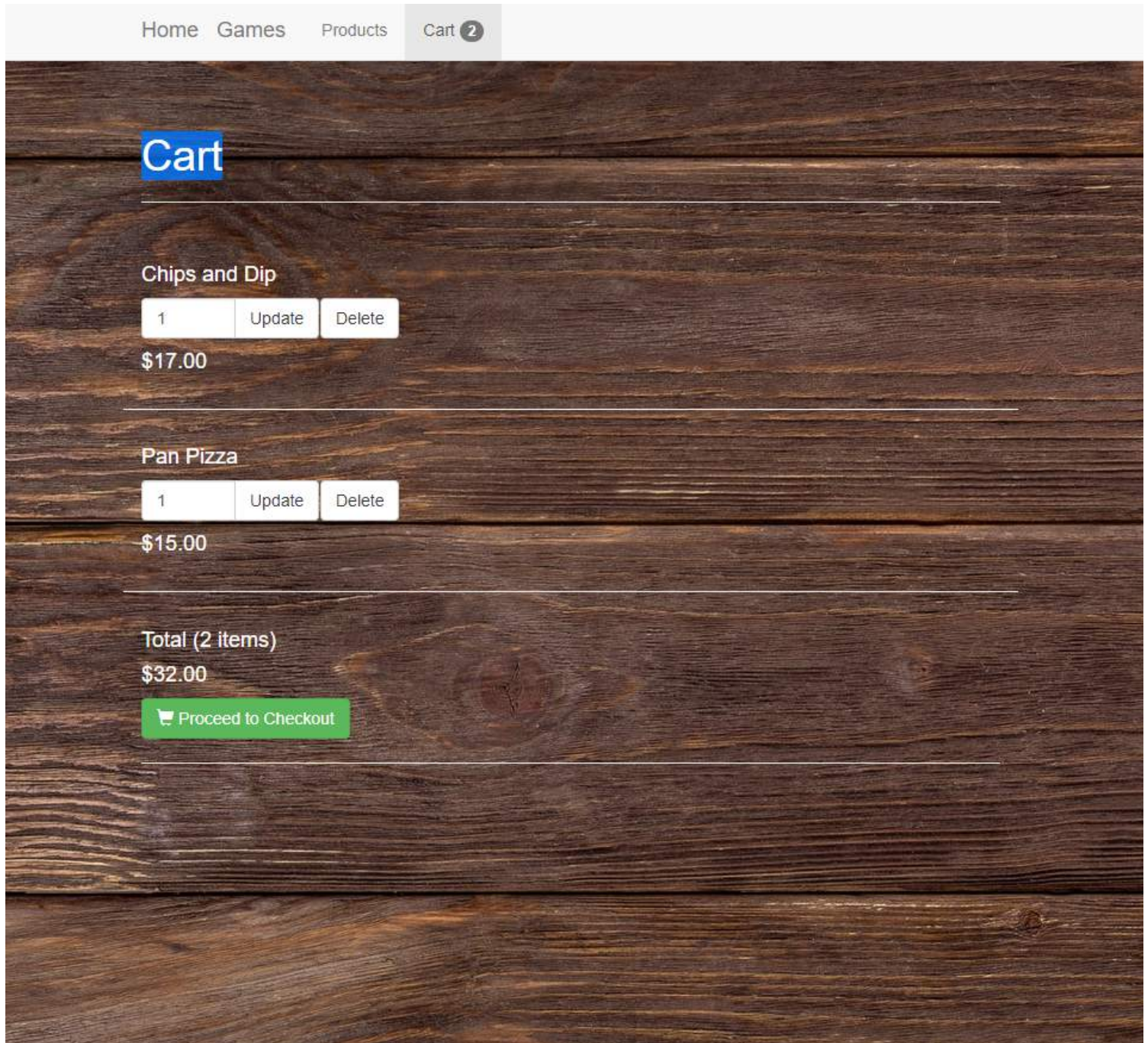
Design/GUI

Customer:

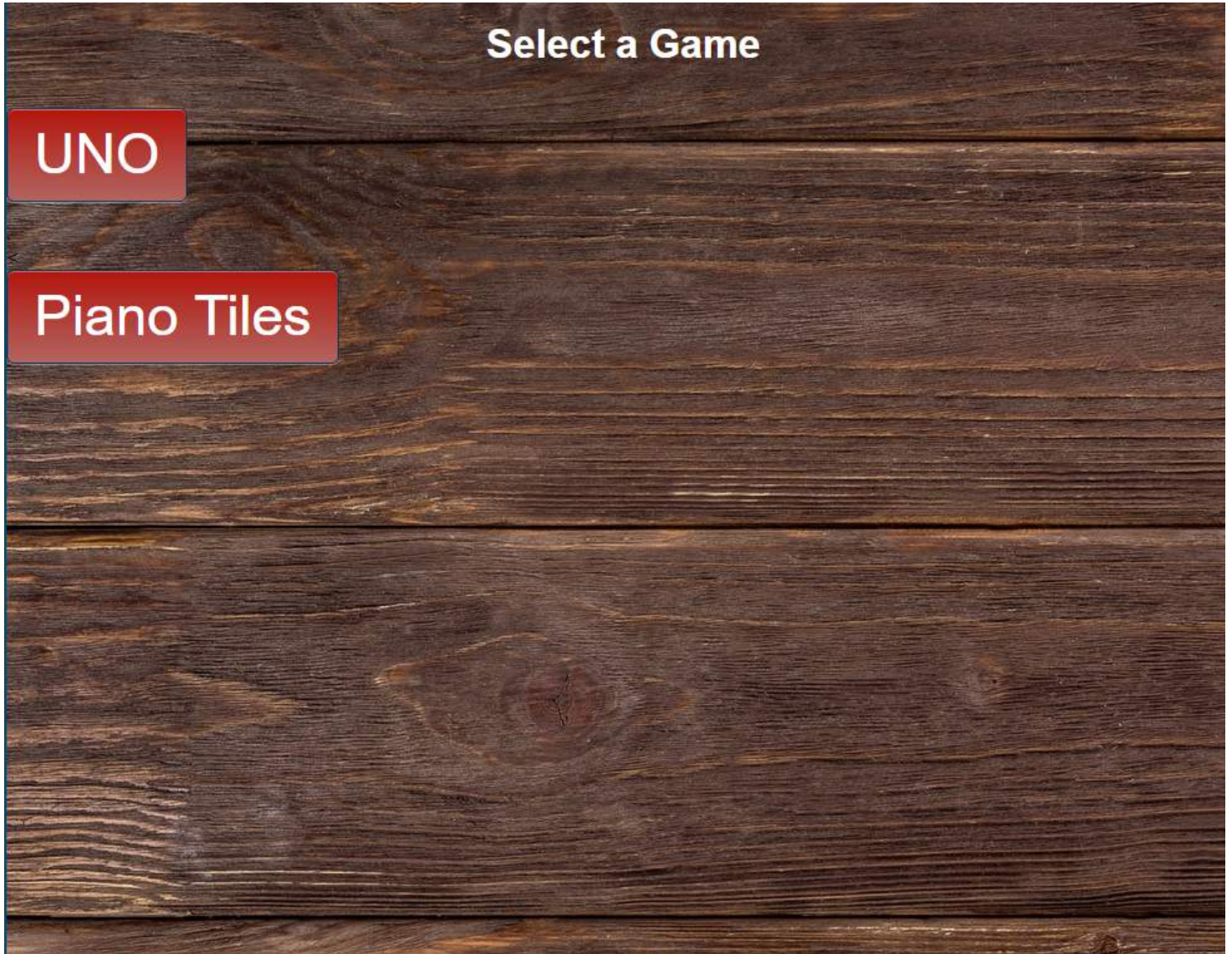
This is the ordering system



Customer can select desired items and put them in cart



After adding items to cart, customer can check items in cart and the total price



For entertainment while waiting for the order, customers can play games

OBJECTIVES

The game is played in three rounds. Try to get rid of all the cards in your hand before your opponents. When its your turn try to match the card on the Discard pile, either by number, colour or symbol.

PASS BUTTON

If you don't want to play the card picked up using the draw card button you can press the pass button to end your turn without playing any card.

DRAW CARD

If you don't have a matching card you must draw a card from the Draw pile. If the card picked up can be played, you are free to do this in the same round. Otherwise, play move to the next person in turn.

UNO™ BUTTON

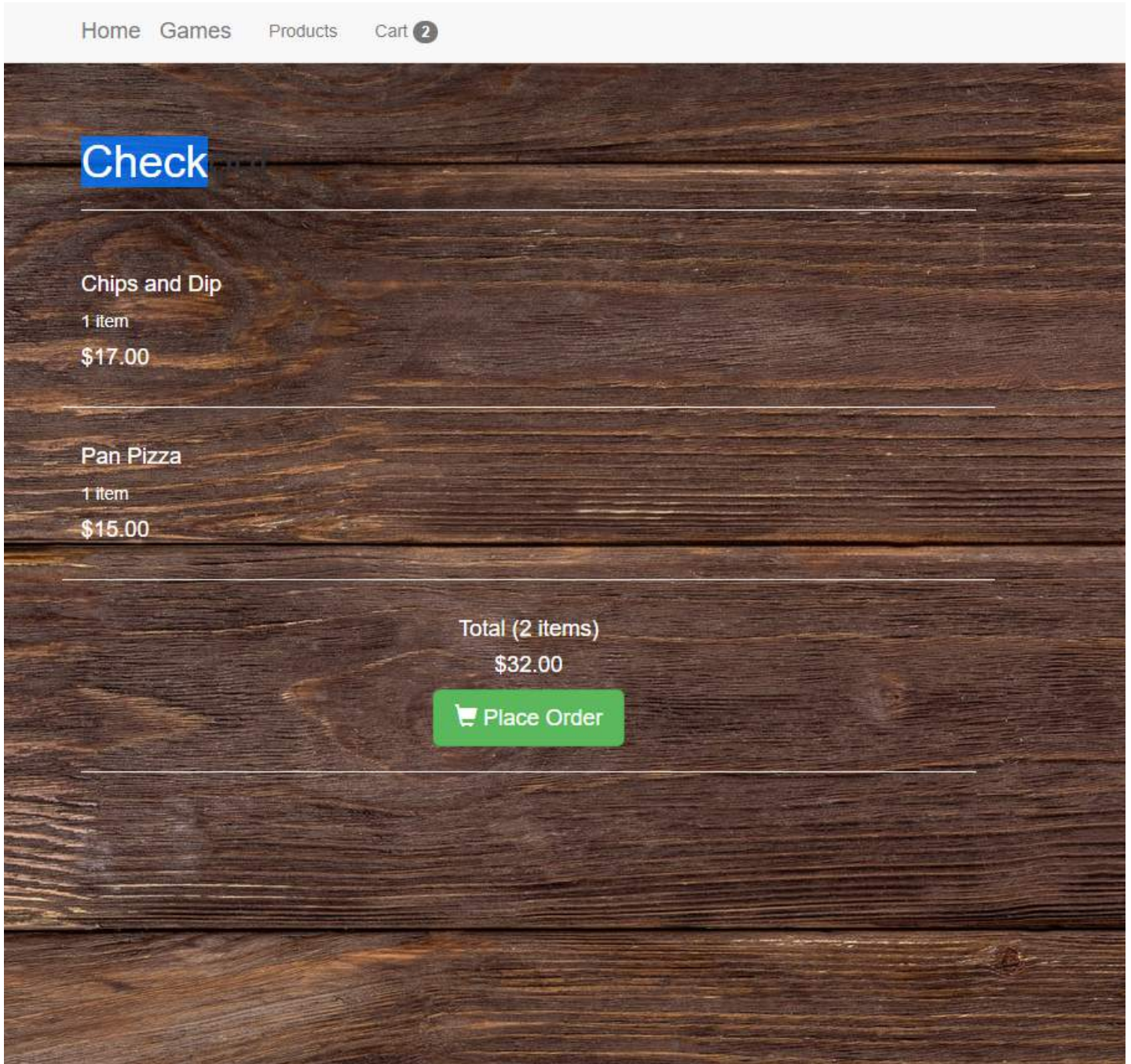
When you have only one card left you must press the UNO™ button. Failure to do this results in having to pick up 2 cards from Draw pile.

THE GAME STARTS IN 38 SECONDS

UNO and associated trademarks and trade dress are owned by, and used under license from, Mattel, Inc. © 2009 Mattel, Inc. All RI

More [Board Games Online](#) on Silvergames.com!

Uno game option



To pay for order, customers can place their order and pay through paypal or credit/debit

LITTLE BITS

[Home](#) [Menu](#) [Games](#) [Payment](#) [Cart](#) [SOS](#)

Name *

Phone

Email

Number of Attendees *

1

Comments / Additional Requests

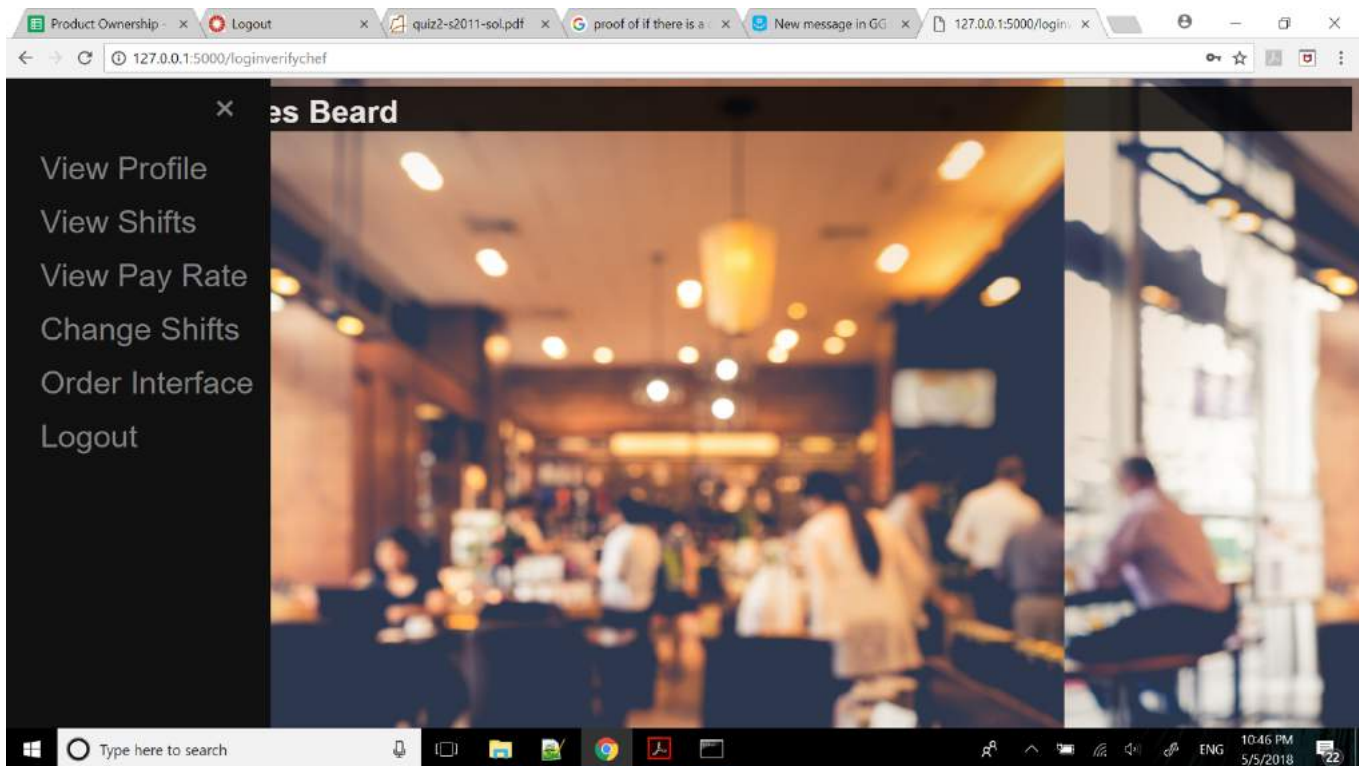
Submit

The floor plan diagram illustrates the restaurant's layout. It includes a legend with three categories: 'Available' (blue square), 'Unavailable' (grey square), and 'Booked' (red square). The layout shows a kitchen area on the right, a bar area at the bottom right, and a main dining area with several tables. Some tables are marked as 'Booked' (red), while others are 'Available' (blue). The background of the reservation form is a wooden table with a plate of food and a red and white checkered cloth.

For Reservation

LITTLE BITS

Employees:



Employee Interface

Table Number	Menu items	Quantity	Order Status	Order Status
3	Chicken Nuggets	2	In Progress	Ready
4	Mini Pizza	1	In Progress	Ready
4	Green Salad	1	In Progress	Ready
5	Tomato Soup	3	In Progress	Ready
5	Apple Pie	1	In Progress	Ready

Go back to Main Page

Table Orders

LITTLE BITS

Shifts

Day	Shift Start	Shift End
Monday	8:00 am	3:00 pm
Wednesday	8:00 am	3:00 pm
Friday	8:00 am	3:00 pm
Sunday	8:00 am	3:00 pm

[Go back to Main Page](#)



Employee Shifts

Holly Beske Hi

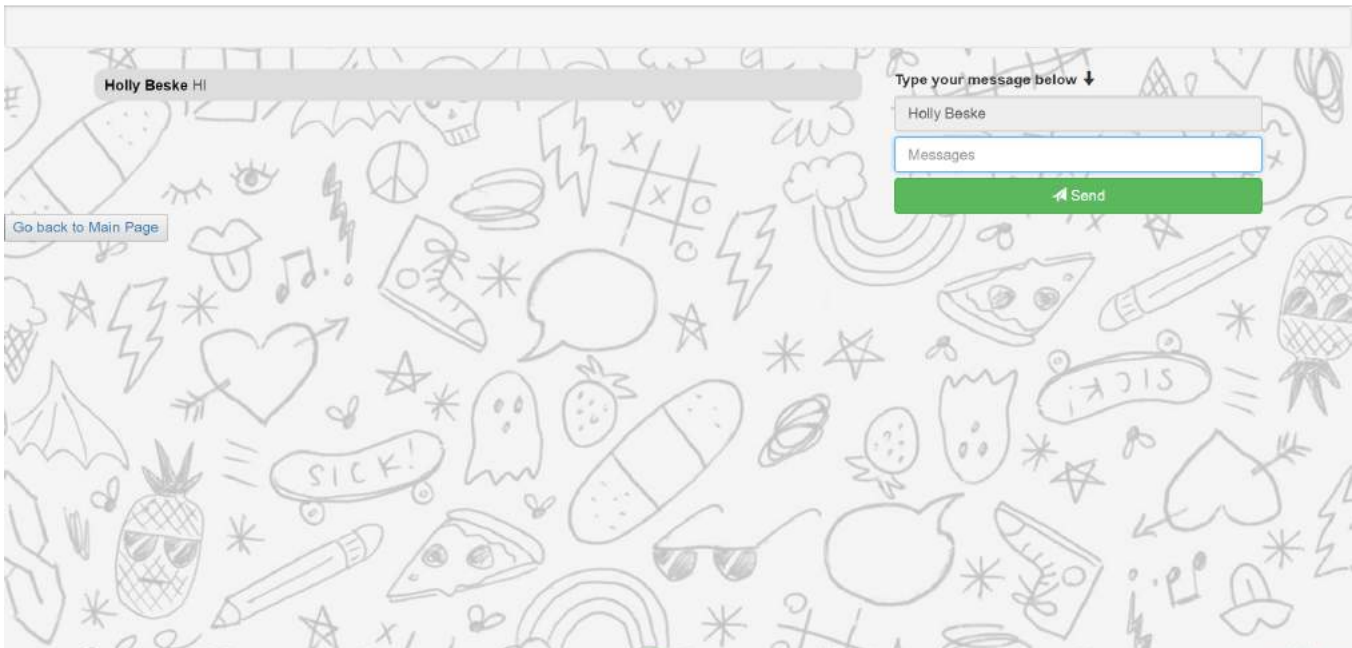
Type your message below ↓

Holly Beske

Messages

[Send](#)

[Go back to Main Page](#)



Employee Chat System

LITTLE BITS

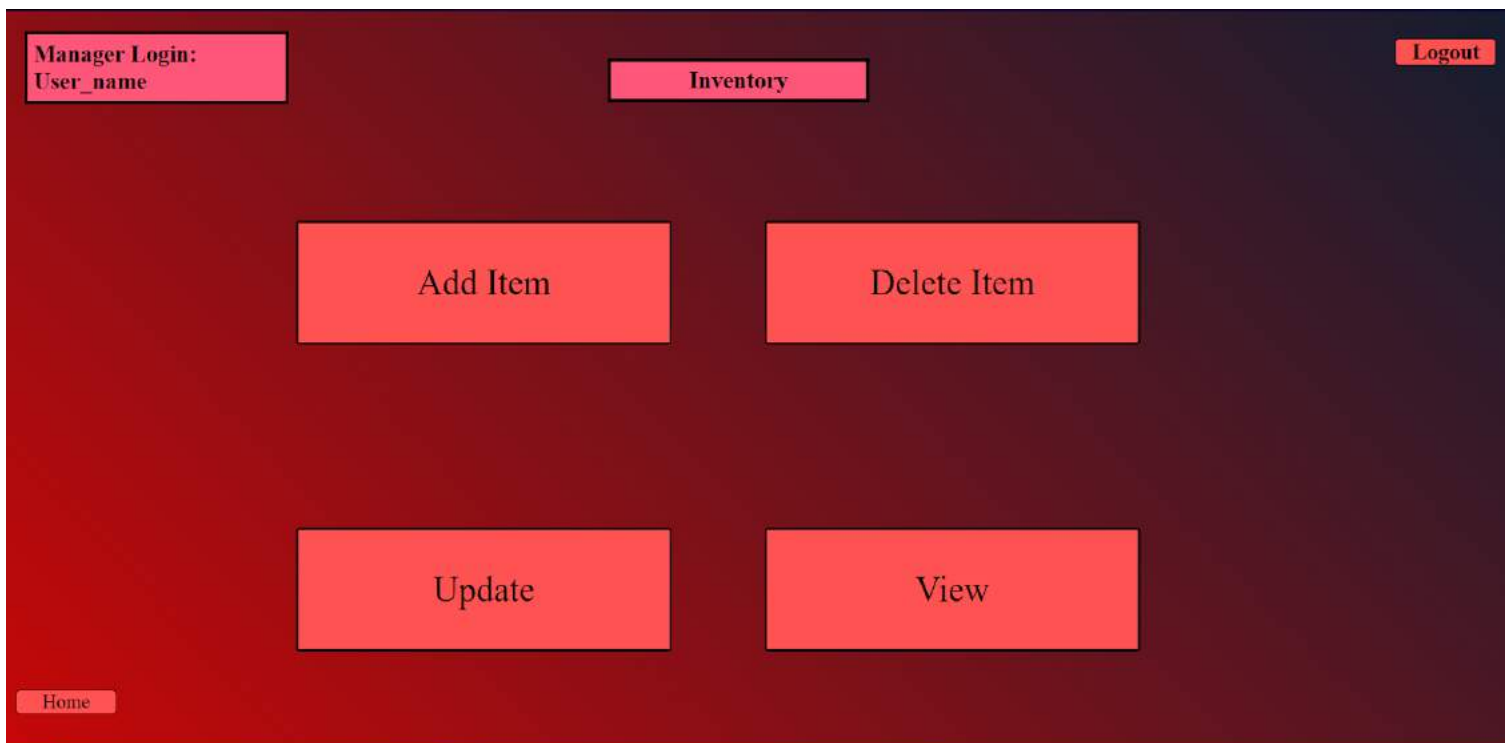
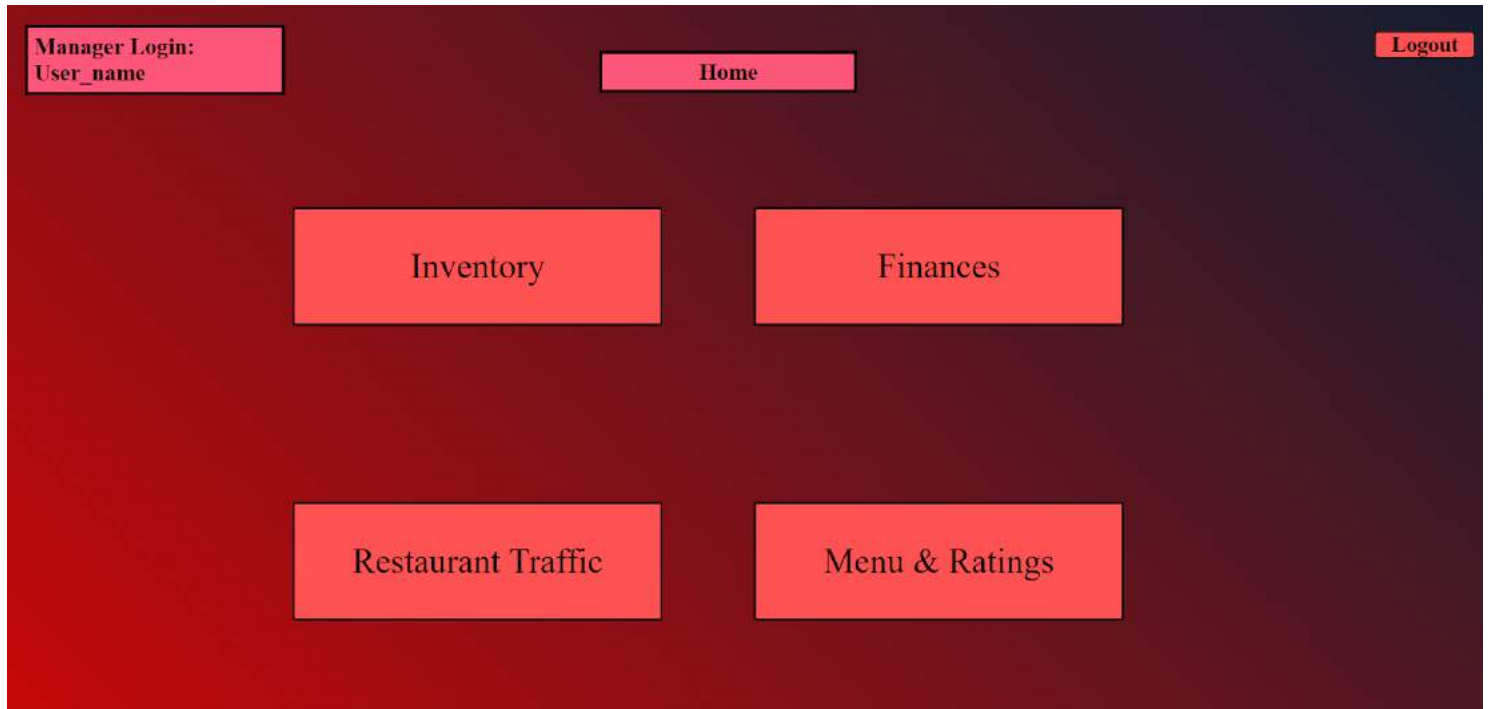
Table Number	Reserved	Vacant	Occupied	Dirty	Clean	Refresh
6	N	N	Y	Y	N	Refresh
7	N	Y	N	N	Y	Refresh
8	Y	Y	N	Y	N	Refresh
1	N	Y	N	Y	N	Refresh
2	N	Y	N	Y	N	Refresh
9	Y	Y	N	N	Y	Refresh
10	N	N	Y	Y	N	Refresh
3	N	N	Y	N	Y	Refresh
5	Y	Y	N	N	Y	Refresh
4	Y	Y	N	N	Y	Refresh

Go back to Main Page

Waiter – Busser Interface

LITTLE BITS

Managers:



Inventory

LITTLE BITS

Manager Login: User Name

Inventory Item Information

Logout

Add Item

Enter Item Information

Item Name

Quantity

Unit Of Measure ('lbs'(pounds) or 'gal'(gallons))

Expiration Date (Format: YYYY-MM-DD)

Cost Per Unit (Format: x.xx., eg. 1.23)

submit

Add to Inventory

Manager Login: User Name

Inventory Item Information

Logout

Item Name

Expiration Date (Format: YYYY-MM-DD)

submit

Cancel

Delete from Inventory

LITTLE BITS

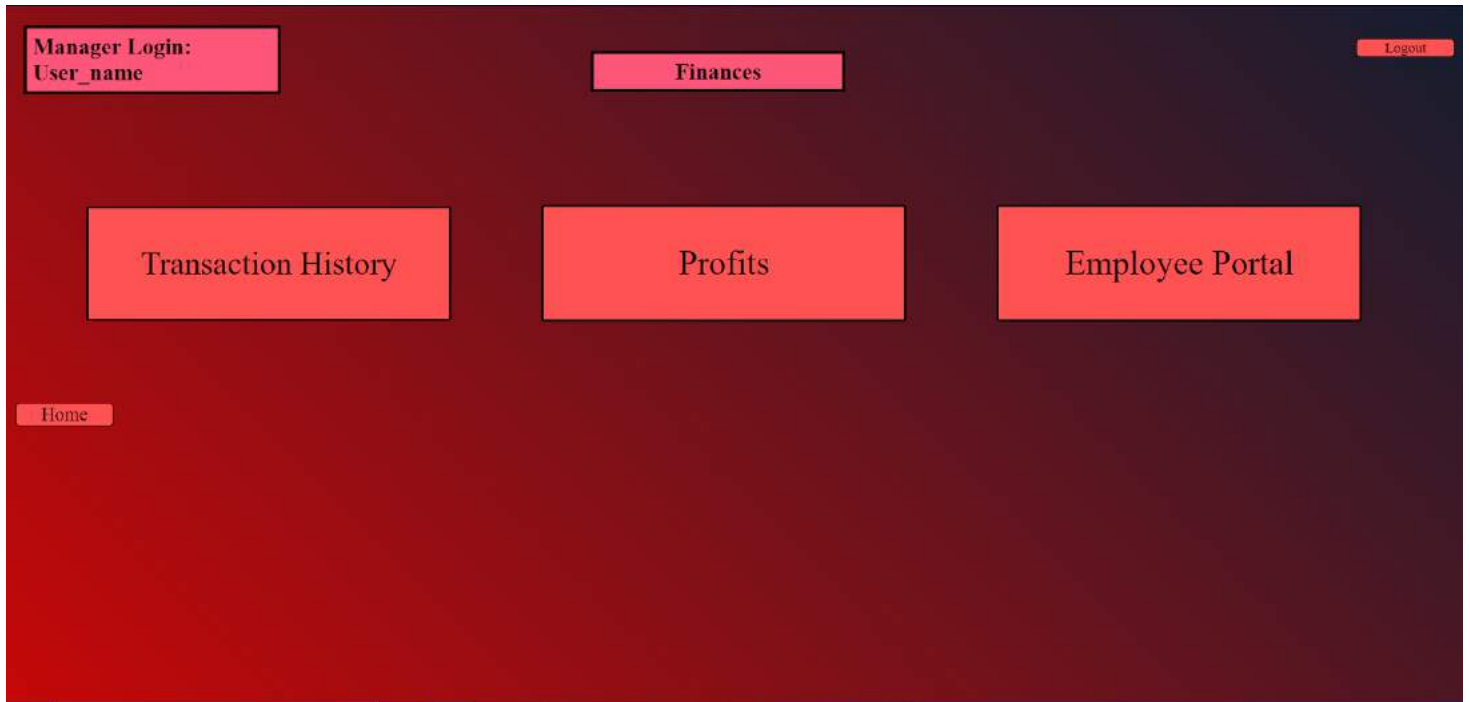
This screenshot shows a web application interface for managing a menu. The background is a dark blue gradient. At the top left, a box contains the text 'Manager Login: User_name'. At the top center, a box is labeled 'Menu & Ratings'. At the top right, a red 'Logout' button is visible. In the center, there are four large, light blue buttons arranged in a 2x2 grid: 'Add Menu Item', 'Delete Menu Item', 'Update', and 'View'. At the bottom left, there is a red 'Home' button.

Menu and Ratings

This screenshot shows a web application interface for adding a new menu item. The background is a dark blue gradient. At the top left, a box contains the text 'Manager Login: User Name'. At the top center, a box is labeled 'Menu Item Information'. At the top right, a red 'Logout' button is visible. In the center, there is a light blue form box containing five input fields with labels: 'Item Name', 'Item Price', 'Ingredient Name', 'Quantity of Ingredient', and 'Unit of Measure'. Below these fields is a red 'submit' button. At the bottom left, there is a red 'Cancel' button.

Add to Menu

LITTLE BITS



Finances

This screenshot shows the 'Transaction History' table. At the top left, there is a 'Manager Login: User_name' field. In the top right corner, there is a 'Logout' button. The table has three columns: 'ID', 'Name', and 'Cost'. The table contains 15 rows of data, each with 'row 0' in the 'ID' column, 'row 1' in the 'Name' column, and 'row 2' in the 'Cost' column. A 'Back' button is located in the bottom left corner.

ID	Name	Cost
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2
row 0	row 1	row 2

Transaction History

Ease of Use

Customers:

Reservation

- CUSTOMER ARRIVES
- Type in name and party size
- Select table location preference
- Given wait time

At the Table

- Either open menu or play games
- In menu, select category
- Then select item and add specifications if applicable
- When finished selecting items, press the ORDER button
- Screen displays wait time for all food in total
- In games, select game and stop whenever you want
- Also SOS button available

When Ready to Pay

- Selecting Pay option
- Select form of payment or splitting the check
- Pay or call waitress for cash
- Choose if they want to receive a receipt
- Customer leaves

Employees:

To create a system that is easy to use, we tried to eliminate having a multitude of functions. We created a simple interface that simply, when an employee logs in, they will be automatically directed to their positions portal. The user interface is very user friendly. Each tab takes allows the user to access one or more of our system's functions. Employees can easily enter their availability through clicking on a calendar to mark the days they are available and can then enter the specific times they are available (during which they desire to have their shifts) through choosing the start and end time from a drop down menu. We wanted to make as many things automated as possible, to help manage tasks so that the employee can focus on customer satisfaction.

Managers:

For the managers in particular when the user interface is implemented, management interface will be simple and user friendly, where it will only take a few clicks to see your restaurant menu, finances, inventory, employees and restaurant traffic. The managers will need to work with a system that is quick to learn and easy to memorize. A quick and simple interface will allow managers to spend less time staring at a screen and more time engaging with both the employees and the customers.

Design of Tests

Test-Case Identifier: TC-1

Function Tested: reservingTable(): void throws exception

Pass/Fail Criteria: The test passes if a customer is able to reserve a table successfully by entering all the required input details

Test Procedure	Expected Results
Call function (Pass)	Table will be reserved
Call function (Fail)	If the party size is more or less than the available table sizes or if the restaurant is full or if the customer enters wrong details

Test-Case Identifier: TC-2

Function Tested: billPayment(): void throws exception

Pass/Fail Criteria: The test passes if a customer is able to pay his/her bill successfully by choosing one of the payment options and entering the required details

Test Procedure	Expected Results
Call function (Pass)	Bill will be paid
Call function (Fail)	If the customer enters wrong details or is unable to choose one of the available payment options

LITTLE BITS

Test-Case Identifier: TC-3

Function Tested: Notify waiter (waiterID, messageID)

Pass/Fail Criteria: The chef can indicate when the order is finished and the waiter is notified

Test Procedure	Expected Results
Call function (Pass)	1- the waiter gets a notification to pick up the order and deliver it to the customer. 2- the order disappears from the order cue.
Call function (Fail)	1- The waiter doesn't get a notification to pick up the order and deliver it to the customer. 2- the order doesn't disappear from the order cue. 3- the wrong waiter gets notified.

Test-Case Identifier: TC-4

Function Tested: Notify waiter (waiterID, messageID)

Pass/Fail Criteria: A waiter is notified five minutes after a customer pays their bill

Test Procedure	Expected Results
Call function (Pass)	1- A popup message shows to the specific waiter who we want to be notified. The message says "is table X vacant?" where X is the table number of the customer. 2- If the waiter chooses "No," result 1 is repeated after 5 minutes till the waiter chooses yes. 3- if the waiter chooses "Yes," a popup message shows to the busser saying "Table X is vacant. Please clean it."
Call function (fail)	The waiter is not notified or the wrong waiter is notified.

LITTLE BITS

Test-Case Identifier: TC-5

Function Tested: notifyBusser (TableID)

Pass/Fail Criteria: The waiter can notify the busser to clean a table

Test Procedure	Expected Results
Call function (Pass)	a popup message shows to the busser saying “Table X is vacant. Please clean it.”
Call function (fail)	The pop up message does not show.

Test-Case Identifier: TC-6

Function Tested: UpdateTabling (TableID)

Pass/Fail Criteria: A busser can mark a table as clean

Test Procedure	Expected Results
Call function (Pass)	The restaurant layout shows the table as clean
Call function (fail)	The restaurant layout shows the table as dirty, inUse or not available.

Test-Case Identifier: TC-7

Function Tested: DisplayTimeSlots();

Pass/Fail Criteria: Employee can see the time slots they can sign up for.

Test Procedure	Expected Results
Call function (Pass)	Available time slots show
Call function (fail)	Available time slots don’t show or unavailable time slots show as available

LITTLE BITS

Test-Case Identifier: TC-8

Function tested Tested: GetPermissions()

Pass/Fail Criteria: Employee can ask for permission to add/delete a time slot to their schedules.

Test Procedure	Expected Results
Call function (Pass)	In the tab “schedules” in the manager portal, a message shows saying “X is asking to add /drop the the following time slot.” X is the employee.
Call function (fail)	The message doesn’t show up

Test-Case Identifier: TC-9

Function Tested: payEmployee()

Pass/Fail Criteria: Manager can pay employees and can adjust the pay

Test Procedure	Expected Results
Call function (Pass)	1- manager can see the number of hours every employee worked and the recommended pay 2- manager can adjust pay 3- a money transfer is sent to the employee with the amount the manager chose.

Test-Case Identifier: TC-10

Function Tested: ViewInventory(): inventoryItem[][]

Pass/Fail Criteria: The test passes if the manager is able to view the inventory successfully

Test Procedure	Expected Results
Call function (Pass)	Display Inventory
Call function (Fail)	Database failure

LITTLE BITS

Test-Case Identifier: TC-10a

Function Tested: addItem(itemName, itemQuantity, itemExpirationDate): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to add an item to the inventory successfully by entering all the required item details

Test Procedure	Expected Results
Call function (Pass)	Item added to inventory
Call function (Fail)	Item amount is less than or equal to 0, no name

Test-Case Identifier: TC-10b

Function Tested: deleteItem(itemName, itemExpirationDate): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to delete an item from the inventory successfully by telling the system which item he wants removed

Test Procedure	Expected Results
Call function (Pass)	Item deleted from inventory
Call function (Fail)	Item does not exist, item name blank

Test-Case Identifier: TC-10c

Function Tested: updateItem(itemName, itemQuantity, itemExpirationDate): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to update an item in the inventory successfully by entering in the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item in the inventory is updated
Call function (Fail)	Item does not exist, quantity is less than 0

LITTLE BITS

Test-Case Identifier: TC-11

Function Tested: ViewMenu(): menuItem[]

Pass/Fail Criteria: The test passes if the manager is able to view the menu

Test Procedure	Expected Results
Call function (Pass)	Display Menu
Call function (Fail)	Database Failure

Test-Case Identifier: TC-11a

Function Tested: addItem(menuItemName, menuItemPrice): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to add item to the menu, by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item added to menu
Call function (Fail)	itemPrice was \$0.00 or less or no price or no name

Test-Case Identifier: TC-11b

Function Tested: deleteMenuItem(menuItemName, menuItemPrice): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to delete item to the menu, by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item removed to menu
Call function (Fail)	Incorrect item name and price, price is less than 0, item does not exist

LITTLE BITS

Test-Case Identifier: TC-11c

Function Tested: UpdateMenuItem(menuItemName, menuItemPrice): boolean

Pass/Fail Criteria: The test passes if the manager is able to update an menu item by inputting the item's fields

Test Procedure	Expected Results
Call function (Pass)	Item is updated
Call function (Fail)	Incorrect item name and price, price is less than 0, item does not exist

Test-Case Identifier: TC-12

Function Tested: ViewSchedule(): EmployeeSchedule[][]

Pass/Fail Criteria: The test passes if the manager is able to see the employee schedule

Test Procedure	Expected Results
Call function (Pass)	Displays Employee Schedule
Call function (Fail)	Database failure

Test-Case Identifier: TC-12a

Function Tested: addEmployee(EmployeeName, EmployeePosition,pay): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to add an employee to the system

Test Procedure	Expected Results
Call function (Pass)	Employee added to system
Call function (Fail)	Incorrect position or pay<0 is inputted

LITTLE BITS

Test-Case Identifier: TC-12b

Function Tested: deleteEmployee(EmployeeName, EmployeePosition,pay): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to remove an employee from the system

Test Procedure	Expected Results
Call function (Pass)	Employee is removed from the system
Call function (Fail)	Incorrect name or position or pay<0 is inputted

Test-Case Identifier: TC-12c

Function Tested: ChangeEmployeePay(EmployeeName, EmployeePosition,pay): boolean throws exception

Pass/Fail Criteria: The test passes if the manager is able to change an employee's wages

Test Procedure	Expected Results
Call function (Pass)	Employee's pay is updated
Call function (Fail)	Incorrect name or position or pay<0 is inputted

Test-Case Identifier: TC-13

Function Tested: updateQueue(orderInfo): int throws exception

Pass/Fail Criteria: The test passes if the queue is updated correctly by the chef

Test Procedure	Expected Results
Call function (Pass)	Displays updated queue
Call function (Fail)	Queue is not updated, Queue is updated incorrectly

LITTLE BITS

Test-Case Identifier: TC-14

Function Tested: inventory (): void throws exception

Pass/Fail Criteria: The test passes if the system launches the inventory interface

Test Procedure	Expected Results
Call function (Pass)	Displays inventory menu
Call function (Fail)	System crashes

Test-Case Identifier: TC-14a

Function Tested: MenuAndRating(): void throws exception

Pass/Fail Criteria: The test passes if the system launches the Menu and Ratings interface

Test Procedure	Expected Results
Call function (Pass)	Displays Menu and Ratings options
Call function (Fail)	System crashes

Test-Case Identifier: TC-14b

Function Tested: RestaurantTraffic(): void throws exception

Pass/Fail Criteria: The test passes if the system launches the Restaurant Traffic interface

Test Procedure	Expected Results
Call function (Pass)	Displays restaurant traffic view
Call function (Fail)	System crashes

Test-Case Identifier: TC-14c

Function Tested: FinancialManagementSystem(): void throws exception

Pass/Fail Criteria: The test passes if the system displays the Financial Management System interface

Test Procedure	Expected Results
Call function (Pass)	Displays FMS menu
Call function (Fail)	System crashes

Test-Case Identifier: TC-14d

Function Tested: EmployeeManagementSystem(): void throws exception

Pass/Fail Criteria: The test passes if the system correctly displays the employee management system interface

Test Procedure	Expected Results
Call function (Pass)	Displays Employee management options
Call function (Fail)	System crashes

Test-Case Identifier: TC-15

Function Tested: startOrder(newOrder):boolean

Pass/Fail Criteria: The test passes if the customer is able to submit a new order

Test Procedure	Expected Results
Call function (Pass)	Order is sent to the queue
Call function (Fail)	Database failure, invalid order (i.e. order item is out of stock)

LITTLE BITS

Test-Case Identifier: TC-16

Function Tested: viewTraffic(): traffic[]

Pass/Fail Criteria: The test passes if the system displays correct information about the current restaurant traffic and table statuses

Test Procedure	Expected Results
Call function (Pass)	Displays restaurant information
Call function (Fail)	Database failure, displays incorrect or outdated floor view/information

Test-Case Identifier: TC-17

Function Tested: writeOrder(orderInfo): boolean throws exception

Pass/Fail Criteria: The test passes if there are no problems with the customer's order

Test Procedure	Expected Results
Call function (Pass)	Order is sent to the queue and returns a confirmation to the customer
Call function (Fail)	Order is not sent to the queue and an error message is returned to the customer

Test-Case Identifier: TC-17a

Function Tested: prepTime(orderInfo): int throws exception

Pass/Fail Criteria: The test passes if the system displays the correct preparation time for the order

Test Procedure	Expected Results
Call function (Pass)	Displays the correct approximate time for order completion
Call function (Fail)	Database failure, displays no or incorrect preparation time for order

LITTLE BITS

Test-Case Identifier: TC-17b

Function Tested: updateQueue(): boolean throws exception

Pass/Fail Criteria: The test passes if the queue is correctly updated with the customer's order

Test Procedure	Expected Results
Call function (Pass)	Displays correct order queue
Call function (Fail)	Database failure, queue is updated incorrectly or not updated at all,

Test-Case Identifier: TC-18

Function Tested: viewProfits(): double throws exception

Pass/Fail Criteria: The test passes if the manager is successfully able to obtain profits

Test Procedure	Expected Results
Call function (Pass)	Displays profits of type double
Call function (Fail)	Database failure, profits are not updated

Test-Case Identifier: TC-18a

Function Tested: viewProfitProj(timePeriod): double profit[]

Pass/Fail Criteria: The test passes if the manager is successfully able to see his profit projections

Test Procedure	Expected Results
Call function (Pass)	Displays profit projections
Call function (Fail)	Database failure, profit projections are not calculated, not enough

LITTLE BITS

Test-Case Identifier: TC-18b

Function Tested: viewTransactions(): transactionStructure

Pass/Fail Criteria: The test passes if the manager is successfully able to see his profit projections

Test Procedure	Expected Results
Call function (Pass)	Displays transactions made such as paying employees, cost of food that is in the inventory now, amount of money gained/lost each day
Call function (Fail)	Database failure, transactions not up to date

Test Coverage

The tests cover most essential classes such as Reservation, Ordering and Payment processes implemented by the customers. More tests will be designed when we see it needed in the future as we develop the customer interface in the Web application. Some of the tests cover the basic possibilities within UC2 and UC5 including sign. For the manager section the test covers the important classes which deal with inventory management, employee management, restaurant traffic, financial management systems. More tests may be required when the system is implemented and needs to be tested for durability.

Integration Testing Strategy

Big Bang integration testing is an integration testing strategy where all units are linked at once resulting in a complete system. We will not use this as our primary testing strategy but instead use big bang integration testing to test our fully developed software. This is useful to test if the back end database and the front end interaction works smoothly. Big bang testing also helps to look for loopholes in the software when all components are integrated together. For example: Combining all the applications from the customer side (placing orders, SOS call button, processing the payment and reserving the tables), the employee's side (marking the status of the table, order queue, and accessing the schedules) and the manager's side, all these functionalities must be embedded together and work in the right order.

We will be using mixed(sandwich) testing for testing our primary components. Sandwich testing combines both top down and bottom up approach. We break the application into three parts- customers, employees and manager, where using permutations each and every application pair is tested separately and the corresponding results are recorded. The pairs include the customer chef pair where tests include the ordering queue and the processing time, the customer busser pair where after the payment is processed, the busser is notified, the waiter busser pair where both the parties are responsible for updating the table status, the manager employee pair where the schedules/shifts are managed, the manager chef pair where they coordinate with the ingredients. This strategy is useful and works quite well because before we combine everything and move on to big bang, we need to make sure that two components interact with each other in an orderly manner.

A few functionalities like the financial management system are independent and their testing is done using the big bang testing strategy.

History of Work

Customer:

Completed Work:

- Designing the UI for ordering system, table reservation and payment processes.
- Creating a database for the ordering system.
- Included two games on the website

Current Status:

Our current plan of work is as follows:

- Making a few minor changes in the ordering system.
- Implementing the table reservation system.
- Implementing the payment process.
- Implementing the SOS button.

Employees:

- March 11 - We as an employees team drew the outlines for what we wanted our interface to look like. We also finalized the details we needed to complete for the final demo.
- March 15- We finalized the log in and log out page. This included creating a database for all the employee profiles.
- March 25- We created the database for the scheduling. We created the interface which allowed employees to input their scheduling preferences. It also shows when the employees were scheduled to work.
- March 27 - Created the interface which allowed waiters to mark tables dirty and for bussers to acknowledge they were dirty and can mark when they are cleaned.
- March 29 - Developed an outline of how to manage restaurant traffic. This includes waiters seeing when tables orders are ready and when chefs are completed with the order. It also shows waiters when there is a request for their service.
- March 30- HTML pages finished on front end to make UI look better and to run the page. Updated the UI to change colors for when a table was cleaned or dirty.

These deadlines were later than the deadlines originally planned in the first two reports which are partly due to changing frameworks from Django to Flask. We planned to have more time to complete the interface and requirements, however due to exams and scheduling we had trouble implementing everything for the first demo. We wanted to make adjustments by the 27th, however it ended up coming to the wire, and us working until the demo.

Future Work

For the second demo we will be working on implementing sockets with Flask. We also learned that we had some security issues with the scheduling. We realized that users could possibly input schedule times for other employees which could cause issues. We wanted to create an algorithm which would schedule based off user inputs. We also need to finish the traffic monitoring system. We also are working on creating a interface for soley the chef that no other employee aside from the managers can see. The Chefs and waiters will have a interface that they can both see certain details such as tables that needed to be cleaned.

Accomplishments

- Learned how to use Flask
- Learned how to use Python, SQL, and HTML in one page
- Creating a system for table management-when a table is empty and needed to be cleaned
- Learned to properly manage a restaurant, like understanding how to properly manage a schedule

Manager:

- March 24 - worked with other groupmates to draw an outline for what UI will look like and functionalities needed for the demo and drew the layout of pages when user logs in and what the user will see after clicking on one of the options on the front end.
- March 27 - Database created for inventory and menu items with sample inventory and sample menu. Started to create methods to add items to inventory, and menu.
- March 28 - started to work on methods to remove items from inventory and menu.
- March 29 - HTML pages created on front end with css to make UI look better. And for better navigation. Backend for adding and deleting inventory and menu items are done. Linked part of the backend to the front end
- March 30 - Linked inventory and menu database to front end so manager is able to see the items in his inventory and view the menu items. Manager should be able to add and remove items from the inventory and menu databases.
- March 31 - started working on a method to update menu prices which will update database of menu items.

These deadlines were later than the deadlines originally planned in the first two reports which are partly due to changing frameworks from Django to Flask. Flask is a lot more useful and simpler since it linked javascript, html, css, python, and sql aspects of the project together which makes our lives a lot easier. This way we were able to connect the python file and add sqlite aspects which is the database and connect the front end using html files with css added to it.

Accomplishments

- Learning new languages and learning how to implement them
- Combining multiple languages and connecting them all

Future Work

- Create the finance part where we can obtain the numbers which in order to do so we would need output from the customer/employee subsystems.
- Implement profit projections which would tell the manager how much money he would make in the future given current the manager's current financial information.
- Edit the code to make it a little more efficient and faster so the system can run smoothly.

References

Hardware Req. Reference: <https://fccid.io/XOX-Z400/User-Manual/Users-Manual-2576495>

<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf>

http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

https://www.tutorialspoint.com/software_testing_dictionary/big_bang_testing.htm

<https://neutrium.net/mathematics/least-squares-fitting-of-a-polynomial>

<https://onlinecourses.science.psu.edu/stat501/node/324>

<https://fccid.io/XOX-Z400/User-Manual/Users-Manual-2576495>

<http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2015-g3-report3.pdf>

<https://neutrium.net/mathematics/least-squares-fitting-of-a-polynomial/>

<https://onlinecourses.science.psu.edu/stat501/node/324>

<http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementingaQueueinPython.html>

https://sourcemaking.com/design_patterns

https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Architecture/Design_Patterns