CSCI 441-A

Team B

Report #3: Restaurant Automation

Dec 4, 2023

https://github.com/ivanvelocastaneda/CSCI441_A-Team-B-Project.git

Bjarni Jonsson

Cheikh Abdoulaye Faye

Sokhna Khady Mbacke

Ivan Velo Castaneda

Table of Contents

Individual Contributions Breakdown

Topics	Bjarni	Cheikh	Ivan	Sokhna
Individual Contributions Breakdown	25%	25%	25%	25%
Customer Problem Statement	25%	25%	25%	25%
Goals, Requirements, and Analysis	25%	25%	25%	25%
Use Cases	25%	25%	25%	25%
User Interface Specification	25%	25%	25%	25%
System Architecture	25%	25%	25%	25%
Project size estimation based on use use case points	25%	25%	25%	25%
Analysis and Data Modeling	25%	25%	25%	25%
Interaction Diagrams	25%	25%	25%	25%
Class Diagram and Interface Specification	25%	25%	25%	25%
Algorithms and Data Structures	25%	25%	25%	25%

User Interface Design and Implementation	**25%**	**25%**	**25%**	**25%**
Design of Tests	**25%**	**25%**	**25%**	**25%**
Project Management and Plan of Work	**25%**	**25%**	**25%**	**25%**
References	**25%**	**25%**	**25%**	**25%**

Summary of Changes

- Updated problem statements for manager, host and customers.
- Updated Goals, Requirements, and Analysis
- Updated glossary
- Updated User Interface Requirements
- Updated User interface Design and Implementation
- Updated Analysis and Domain Modelling
- Updated Interaction Diagrams
- Updated Class Diagram and Interface Specification
- Updated Breakdown of Responsibilities
- Added to Plan of Work
- Updated Database Diagram
- Updated Class Diagram
- Updated Design of Tests
- Added Design Patterns and Object Constraint Language
- History of Work, Current Status, and Future Work

1. Customer Problem Statement(updated)

a. Problem Statement

Manager:

Running a restaurant and overseeing employees is a challenging responsibility. My daily tasks can potentially be overwhelming at times. Some tasks include, keep track of which employees are on duty, figure out the payroll for every pay period, ensure employees punctuality and their compensation. Any means to reduce the effort required for these tasks would be greatly appreciated. Additionally, I should have the flexibility to update the menu with ease to ensure that we are matching the times we are actually serving and make modifications if needed in case we run out of the certain item. I should also have the flexibility to add and remove employees from the system to ensure we do not encounter an instance where we do not have the necessary employees to properly run a shift. Farm to table serves as an employee portal, enabling employees to conveniently clock in and out while automatically calculating their compensation and hours worked. This feature will significantly make my life easier as I will no longer need to manually log employee hours or perform pay calculations. The site will handle these tasks smoothly. Furthermore, Farm to Table provides a swift interface for menu editing and staff adjustments, enhancing overall efficiency.

Waiter/Waitress:

Working as a waiter/waitress can be an incredibly demanding job, requiring us to be constantly on the move within the restaurant. We have to take customers orders, keep track of table orders, deliver orders quickly to ensure food stays hot and manage the process of closing out checks when customers are ready to pay their bill. With multiple tables in the restaurant, it can be a difficult process to keep track of what tables ordered what items and to determine what tables need to be cleaned and prepared for the next customer. Unfortunately, sometimes we have to send dishes back if it is not up to the customers standards or find something wrong with it. A site like Farm to Table would be immensely beneficial in allowing us to take customer’s orders, keep track of them in the site, and send them directly to the kitchen. We would also love to receive notifications whenever our food is finished getting prepared so the food does not stay on the window for too long. Lastly, we would like to close down tabs and input customer’s payment (whether they paid cash or card) in the site to ensure we are making the correct amount of tips at the end of the day.

Host/Hostess/Busboy/Busgirl:

When I am welcoming guests into the restaurant, I would like to ensure their experience goes as smoothly as possible. As groups of multiple sizes arrive, they often have specific requests and preferences regarding their seating arrangements based on the number of people in their and where they would like to seat within the restaurant. Sometimes, there might be a five-person party seeking a booth or multiple tables put together and it is our responsibility to ensure there is an available spot and they do not wait too long to get seated. However, it is occasionally uncertain how long it would be until tables become free to accommodate a party's seating preferences.

Finding an appropriate table for our guests can pose a challenge since we do not know the current status of tables. We have to physically go and check if tables are clean or not. It would be highly beneficial to have a means of easily tracking which tables are occupied, or unoccupied. This information would enable us to provide guests with an estimation of when a suitable seating arrangement can be made.

A site like Farm to Table can help us identify which tables are occupied, or unoccupied.

Customer(s):

I love going out to eat, but it can be frustrating at times. It is not the staff's fault whatsoever. Sometimes they are incredibly busy and the server can take a long time to ask for drinks, then go make them, and lastly take our order.Some sort of device on the table with a menu available to my party would be great in placing orders immediately without having to wait for my server to get to me if they are busy. It would also be a great way to have an interactive menu that could offer more information on each item. It would speed up the process of time spent waiting.

Cook/Chef:

Working in a kitchen requires a lot of patience and it can be a demanding job at times. We have to make sure the food is up to standard while simultaneously making it in a reasonable amount of time. We also have to make sure the order is accurate because it can be quite horrible if an order is made incorrectly. A person could potentially get hurt if their order contains an item they are allergic to as a result of miscommunication between us and the waiting staff. It is up to us and the waiting staff to ensure we have a happy customer because that means they would be more likely to come back in the future. It can be annoying at times trying to track a waiter/waitress down to let them know their food is ready to be delivered to their table. The food could get cold and take up space on the window, potentially risking a broken plate or the quality of the meal.

It would be of great help if we could have something to display all incoming orders. We would also love to have a way to let servers know that their food is finished without having to yell out their names to reduce the time the food sits on the window.

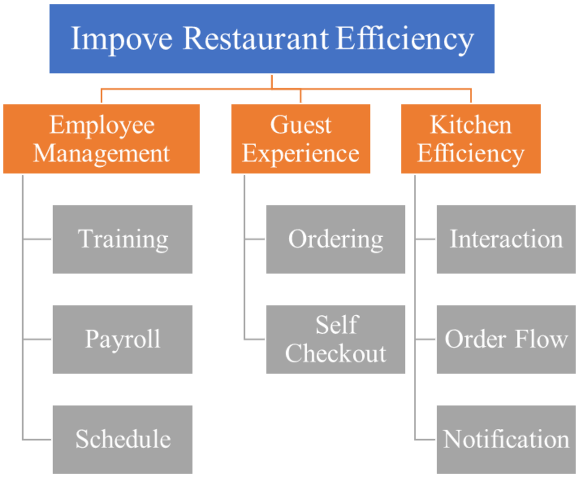
Farm to Table will generate a queue of incoming orders with a timestamp so that we are aware what food orders need to be made first so that we can stay on track. It would eliminate the necessity of handwritten orders thus eliminating any confusion in the server's poor handwriting since we would digitally have the details and requests of the server's tables. The site will allow us to send servers a ping to let them know they can come and pick up their meals so that they may be taken to their tables.

b. Decomposition into Sub-problems
Already described within each problem statement.

c. Glossary of Terms
Cook/Chef: A person who prepares and cooks food, typically as a profession or as a skilled practitioner
Customer: An individual or group of individuals who visit an eating establishment to purchase and consume food and beverages
Employee Portal: It serves as a centralized hub where employees can access a variety of resources, tools, and information related to their employment and workplace
Floor Plan: Refers to the physical arrangement and design of tables, seating areas, and other elements within the dining area of a restaurant
Host/Hostess: An employee responsible for managing the front-of-house operations and ensuring a smooth and welcoming experience for diners
Manager: An individual who holds a position of authority and responsibility within an organization or business
Menu: A written or printed list of food and beverage items that a restaurant offers to its customers.
Queue: Typically refers to a line or waiting area where orders wait their turn to be cooked at a restaurant
Reservation - An arrangement made in advance to secure a table.
Restaurant Automation: Refers to the use of technology and automated systems to streamline and improve various aspects of restaurant operations
Screen: Can refer to various digital displays or monitors used within the establishment for different purposes, often leveraging technology to enhance the dining experience
Tip: Refers to an additional sum of money that customers voluntarily leave for the staff as a token of appreciation for the service provided
Interfaces - The visual aspect of the software that allows user interaction.
Waiter/Waitress: Often referred to as a server, is an individual employed in the hospitality industry, typically at restaurants, cafes, or other dining establishments
Window: A platform used to place plates in

2. Goals, Requirements, and Analysis(updated)

a. Business Goals:



b. Enumerated Functional Requirements:

Identifier	Priority	Requirement
***REQ -1	5	The system will allow customers and employees to select items from the menu and put in an order through an interactive screen.
REQ-2	5	The system provides the host with an interactive screen that displays the current table layout in the restaurant.
REQ-3	4	The system will notify the chef with new orders and place them in the kitchen queue.
REQ-4	4	The system should automate and calculate employee hours and compensation.
***REQ -5	5	The system should allow managers to easily update menu items.
REQ-6	4	Employees should have the ability to clock in and out through an employee portal.
REQ-7	3	The system should offer waitstaff a notification system for when dishes are ready.
*REQ-8	4	The system should digitally process checks and payments

		to assist waitstaff in tip calculation.
REQ-9	3	The system should provide real-time wait time estimations for guests.
**REQ-10	5	The system should allow employees to login and logout.
REQ-11	2	The system should allow managers to add/remove employees from the system.
***REQ-12	3	The system should allow customers and employees to view an order's status.

Modified Functional Requirements:

Identifier	Priority	Requirement	Comments
***REQ-1	5	The system will allow employees to select items from the menu and put them in an order through an interactive screen located in the server/manager interface .	Instead of allowing customers to place orders, employees will be the only individual to be able to do such things.
***REQ-5	5	The system should allow managers to easily update menu items.	The system will no longer notify the staff about unavailable dishes.
*REQ-8	4	The system should digitally process checks and payments to assist waitstaff in tip calculation.	This part will not be implemented by Demo #2.
**REQ-10	5	The system should allow customers and employees to login and logout.	This part will not be implemented.
***RE	3	The system should allow employees to view an order's	The system will only allow employees to view an order's

Q-12		status.	status. Customers will not have the choice to do so.

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

c. Enumerated Nonfunctional Requirements:

Identifier	Priority	Requirement
***NFREQ-1	5	The system provides customers with an interactive screen that displays current table layout in the restaurant.
**NFREQ-2	5	The system must be secure, protecting all data especially payment and personal information in compliance with regulations.
**NFREQ-3	4	The system should integrate seamlessly with existing systems or software used in the restaurant.
NFREQ-4	4	The system should be scalable to accommodate the restaurant's growth or changes.
**NFREQ-5	3	The system should offer training modules or guides to assist employees in understanding the functionalities.
**NFREQ-6	3	The system should have a feedback mechanism for users to report issues or provide suggestions for improvement.
**NFREQ-7	4	The system must have a high availability, ensuring it remains operational during peak restaurant hours.
**NFREQ-8	3	The system should offer multi-language support for diverse customer bases.
***NFREQ-9	3	The system should allow customers to add extras or remove items from order.
NFREQ-10	3	The system should allow everybody to view item ingredients.

*NFREQ-11	3	The system should allow everybody to create a reservation.
**NFREQ-12	2	The system should allow customers to create an rewards account for points
*NFREQ-13	3	Allow customers to order take out

Modified Nonfunctional Requirements:

Identifier	Priority	Requirement	Comments
***NFREQ-1	5	The system provides employees with an interactive screen that displays all the tables in the restaurant.	Customers will no longer have the choice to see the current layout of the restaurant, only customers will be able to see the layout.
*NFREQ-2	5	The system must be secure, protecting all data especially payment and personal information in compliance with regulations.	This will not be implemented by Demo #2.
**NFREQ-3	4	The system should integrate seamlessly with existing systems or software used in the restaurant.	We will not implement this.
**NFREQ-5	3	The system should offer training modules or guides to assist employees in understanding the functionalities.	We will not implement this.
**NFREQ-6	3	The system should have a feedback mechanism for users to report issues or provide suggestions for improvement.	We will not implement this.

**NFREQ-8	3	The system should offer multi-language support for diverse customer bases.	We will not implement this.
*NFREQ-9	3	The system should allow customers to add extras or remove items from order.	This will not be implemented by Demo #2.
*** NFREQ-11	3	The system should allow everybody to create a reservation.	This will not be implemented by Demo #2.
**NFREQ-12	2	The system should allow customers to create an rewards account for points	We will not implement this.
*NFREQ-13	3	Allow customers to order take out	This will not be implemented by Demo #2.

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

d. User Interface Requirements

Identifier	Priority	Requirement
***USREQ-1	5	Interactive Menu Display: The interface should provide customers with a visual and interactive menu. This should include clear images of dishes, concise descriptions, and the price. The interface should be intuitive for customers to place an order directly from this menu.
***USRE	4	Table Layout Visualization: Hosts and waitstaff should

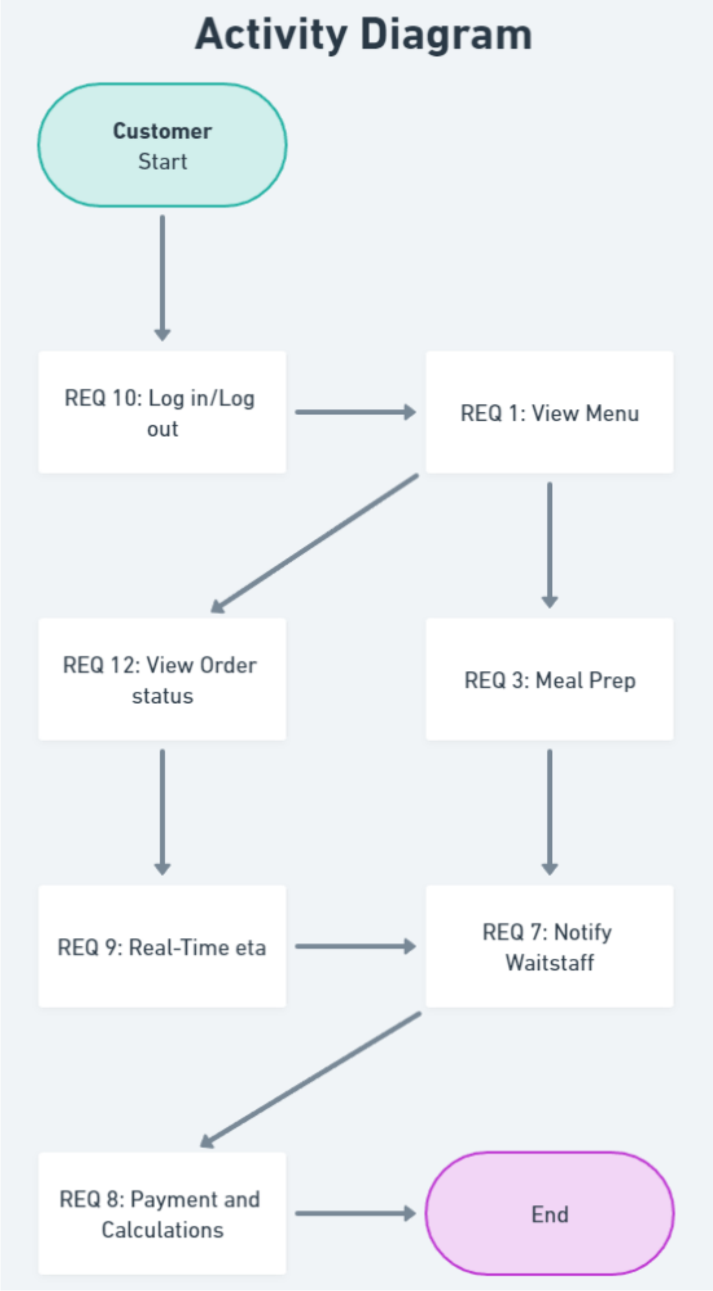
Q-2		have a graphical interface displaying the current status of all tables in the restaurant (occupied, free, needs cleaning). It should be possible to update the table status in real-time.
***USREQ-3	3	Interactive Order Display: The interface should provide customers with a visual and order status screen. This should keep customers more informed of their order status.
*USREQ-4	3	Customer Account Page: The interface provides customers with a web page where they can log into from home to place orders for take-out or make table reservations.

Modified User Interface Requirements

Identifier	Priority	Requirement	Comments
*** USREQ-1	5	Interactive Menu Display: The interface should provide customers with a visual and interactive menu. This should include clear images of dishes, concise descriptions, and the price.	Customers will not have the option to place an order from the menu.
*** USREQ-3	3	Interactive Order Display: The interface should provide employees with a visual and order status screen. This should keep customers more informed of their order status.	Customers will not have the option to see their order status. Only employees will know such information.
*USREQ-4	3	Customer Account Page: The interface provides customers with a web page where they can log into from home to place orders for take-out or make table reservations.	We will not implement this by Demo #2.

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

e. Activity Diagram



3. Use Cases

a. Stakeholders

i. Restaurant Owners - They have an interest in using the system to help optimize efficiency in the restaurant and provide quality customer service for customers.

ii. Employees and Managers - The system will streamline their workload of their process and make their job much easier.

iii. Customers - The system will enrich their dining experience since they will have a chance to interact with it.

iv. Developers - They have an interest in improving and implementing a system that would create solutions to small restaurants' problems without automation face on a daily basis.

b. Actors and Goals

i. Initiating Actors

Actor	Role	Goal
Manager	The manager is the employee in charge of managing the wait staff and the additional needs of the restaurant.	The goal of the manager is to manage employees and their schedules, keep track of inventory, monitor revenues and losses and also ensure restaurant customers have an enriching dining experience.
Employee	The employee is any type of wait staff at the restaurant, except for the manager.	The goal of the employee is to provide customers with an excellent dining experience.
Customer	The customer is a restaurant visitor who chooses to either dine in, order take out, views the menu, orders a meal, and pays for service	The goal of the customer is to have an enriching dining experience with minimal wait time and great service.

ii. Participating Actors

Chef	The chef is responsible for preparing and cooking food that is ordered by customers. They receive a queue of orders and prepare them as they come in. They then send a ping to the servers to let them know their food is ready.

Host/Hostess	The host/hostess is in charge greeting customers and seating them. They can see table status (occupied, unoccupied, dirty) through the database. After seating the customers, they then mark the table as occupied.
Waiter/Waitress	The waiter/waitress is responsible for taking orders from customers, sending them to the kitchen, and serving the food when it is ready. They receive notifications from the kitchen when their meal is ready so they can go and serve it.
Database	The database is a system that records a customer's order, table layout of the restaurant, menu options, etc. It acts as the storage of all information for our site to function properly.

c. Use Cases

i. Casual Description

UC-1: Clocking in/Clocking out-Allow employees to clock in when they first come in to work or after they take a break and clock out after finishing their shift or before taking a break.

Derivations: REQ-6

****UC-2: Log in/Log out**-Allow employees and customers to log in/log out into the system which will determine what interface they will have access to.

Derivations: REQ-10

UC-3: View Menu-Allows employees and customers to view the items on the menu and item ingredients.

Derivations: USREQ-1 / NFREQ-10

*****UC-4: Place Order**-Allows employees and customers to place orders and add/remove items from meals.

Derivations: USREQ-1 / REQ-3 / NFREQ-13 / REQ-1

UC-5: View/Update Table Status-Allows to view the table status of all tables whether occupied, unoccupied, dirty.

Derivations: USREQ-2

***UC-6: Make/View all reservations**-Allow only employees to view reservations. Also allow customers to make or cancel reservations.

Derivations: NFREQ-11

***UC-7: Print Out Reports**-Allow managers to print out employee's reports at the end of their shift.

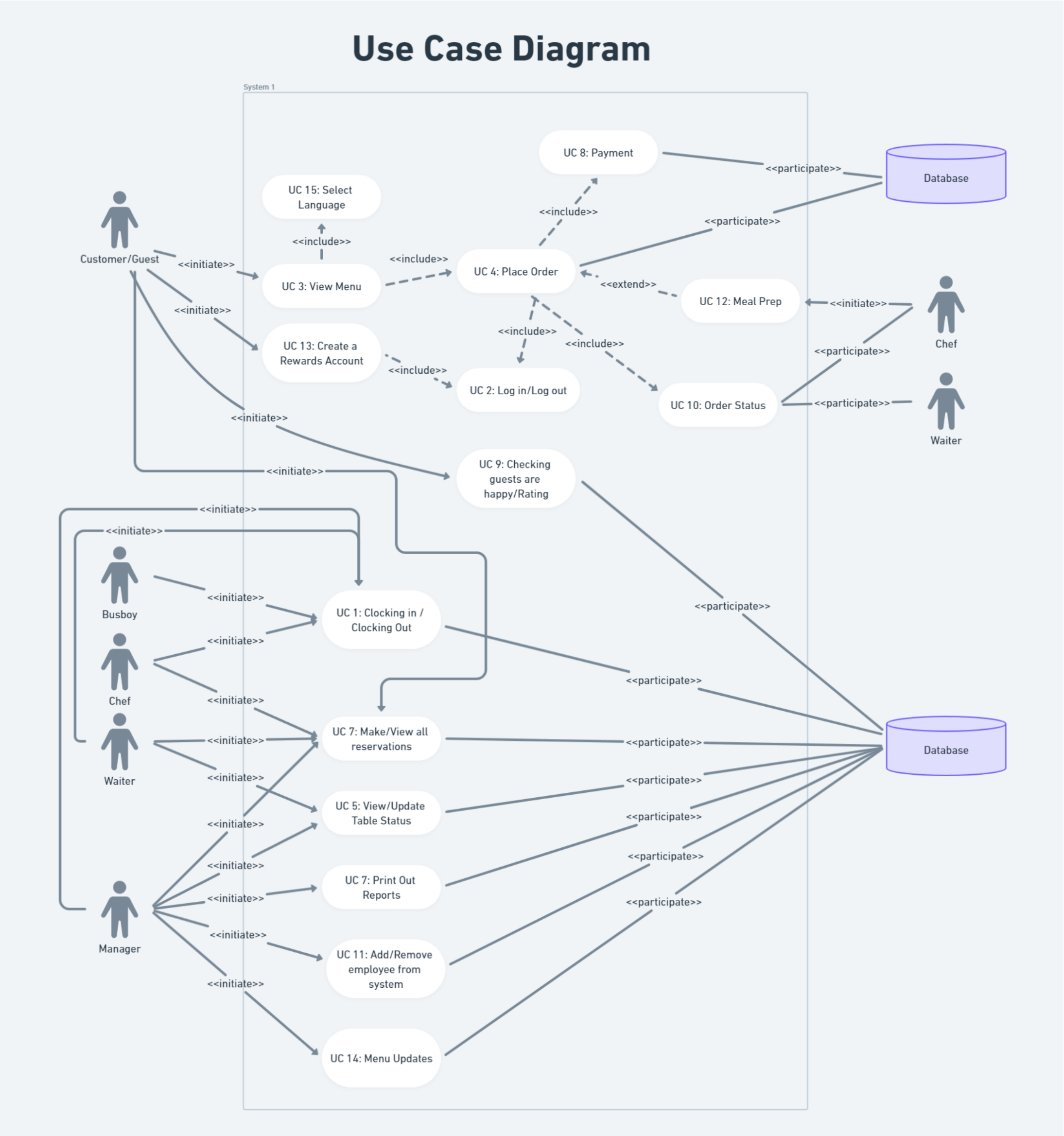
Derivations: REQ-4
***UC-8: Payment**-Allows employees and customers to split the bill and allow customers to pay on the spot (credit-card reader) if they desire it.
Derivations: REQ-8
****UC-9: Checking guests are happy/Rating**-Allows customers to give feedback for the service provided throughout the visit.
Derivations: NFREQ-6
UC-10: Order status-Allows employees to update and view order status, and customers to view order status.
Derivations: USREQ-3 / REQ-12 / REQ-2
UC-11: Add/Remove employees from system-Allow manager to add employees to system when hiring and remove employees from system when firing.
Derivations: REQ-11
UC-12: Meal Prep-Allows chefs to see incoming orders with a timestamp and notify employees when their order is ready.
Derivations: REQ-3 / REQ-7
****UC-13: Create a Rewards Account**-Allows customers to create a rewards account.
Derivations: NFREQ-12 / USREQ-4
UC-14: Menu Updates-Allow manager to add or remove items from menu.
Derivations: REQ-5
****UC-15: Select Language-**Allow customers to select a different language.
Derivations: NFREQ-8

Identifier	Requirement	Comments
*UC-2	Allow employees and customers to log in/log out into the system which will determine what interface they will have access to	We will not implement this.
*** UC-4	Allows employees to place orders and add/remove items from meals.	Customers will not be able to place orders.

*UC-6	Allow only employees to view reservations. Also allow customers to make or cancel reservations.	We will not implement this by demo #2.
*UC-7	Allow managers to print out employee's reports at the end of their shift.	We will not implemented this by Demo #2.
*UC-8	Allows employees and customers to split the bill and allow customers to pay on the spot (credit-card reader) if they desire it.	We will not implemented this Demo #2.
**UC-9	Allows customers to give feedback for the service provided throughout the visit.	We will not implement this.
**UC-13	Allows customers to create a rewards account.	We will not implement this
**UC-15	Allow customers to select a different language.	We will not implement this

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

ii. Use Case Diagram



iii. Traceability Matrix

REQ’t	PW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
REQ-1	5				X											
REQ-2	5										X					
REQ-3	4				X								X			
REQ-4	4							X								

REQ-5	5														X	
REQ-6	4	X														
REQ-7	3												X			
REQ-8	4								X							
REQ-9	3															
REQ-10	5		X													
REQ-11	2											X				
REQ-12	3										X					
NFREQ-1	5															
NFREQ-2	5															
NFREQ-3	4															
NFREQ-4	4															
NFREQ-5	3															
NFREQ-6	3									X						
NFREQ-7	4															
NFREQ-8	3															X
NFREQ-9	3									X						
NFREQ-10	3			X												
NFREQ-11	3						X									
NFREQ-12	2													X		
NFREQ-13	3				X											
USREQ-1	5			X	X											
USREQ-2	4					X										

USREQ-3	3										X					
USREQ-4	3													X		

iv. Fully-Dressed Description

UC-3: View Menu

Related Requirements:
USREQ-1, NFREQ-10

Initiating Actor:
Employee (for assisting customers or placing orders), Customer (for placing orders or inquiries)

Actor's Goal:
To view detailed information about menu items, including images, descriptions, prices, and ingredients

Participating Actors:
Database, Interactive Menu Display

Preconditions:
The restaurant's menu is updated and available in the system.
The system is operational and accessible by both employees and customers.

Postconditions:
The employee or customer has successfully viewed the desired menu items and their details.

Flow of Main Success Scenario:

1. ← The employee or customer accesses the system to view the menu.
2. ← The database retrieves the menu items, including images, descriptions, prices, and ingredients.
3. ← The Interactive Menu Display presents the menu items in an organized and visually appealing manner with suggestions.
4. ← The employee or customer can select individual items to view more detailed information, including ingredients.
5. ← The employee or customer can navigate through different sections of the menu with ease.

Flow of Events for Alternate Success Scenario:

1. ← The employee or customer accesses the system to view the menu.
2. ← The database fails to retrieve the menu items due to a system error.
3. ← The employee or customer is informed of the system error and is asked to try again later.
4. ← The menu remains inaccessible until the error is resolved.

UC-4: Place Order

Related Requirements:
USREQ-1, REQ-3, NFREQ-13, REQ-1

Initiating Actor:
Employee (for assisting customers or placing special orders), Customer (for placing orders).

Actor's Goal:
To efficiently place an order, customize meals by adding or removing items, and ensure the kitchen is notified of the new order.

Participating Actors:
Database, Interactive Menu Display, Kitchen Queue System.

Preconditions:
The restaurant's menu is updated and available in the system.
The system is operational and accessible by both employees and customers.
The kitchen is operational and ready to receive new orders.

Postconditions:
The order has been successfully placed and added to the kitchen queue.
The customer receives a confirmation of their order.

Flow of Main Success Scenario:

1. ← The employee or customer accesses the Interactive Menu Display to view the menu.
2. ← They select desired items, customizing them by adding or removing ingredients as needed.

3. ← Once the order is finalized, they confirm the order.
4. ← The system saves the order in the database and notifies the kitchen via the Kitchen Queue System.
5. ← The chef receives the new order and begins preparation.
6. ← The customer receives a confirmation, including order details and an estimated wait time.

Flow of Events for Alternate Success Scenario:

1. ← The employee or customer accesses the Interactive Menu Display to view the menu.
2. ← They select desired items but encounter a system error when trying to customize or confirm the order.
3. ← The employee or customer is informed of the system error and is asked to try again.
4. ← If the error persists, they may need to place the order manually or seek assistance.

UC-10: Order Status

Related Requirements:
USREQ-3, REQ-12, REQ-2

Initiating Actor:
Employee (for updating and viewing order status)

Actor's Goal:
To keep the order process transparent and informed for both employees and customers.

Participating Actors:
Database, Interactive Display Screen.

Preconditions:
An order has been placed by the customer.
The system is operational and accessible by both employees and customers.

Postconditions:
The order status is updated and visible to relevant parties.

Flow of Main Success Scenario:

1. ← The employee accesses the system to view or update the order status.
2. ← The database retrieves the current status of the order.
3. ← If accessed by an employee, they have the option to update the order status (e.g., "preparing", "ready for pickup", "served").
4. ← The updated status is saved in the database.
5. ← The Interactive Display Screen shows the updated status to the employee

Flow of Events for Alternate Success Scenario:

1. ← The employee accesses the system to view the order status.
2. ← The database fails to retrieve the current status due to a system error.
3. ← The employee is informed of the system error and is asked to try again later.
4. ← The order status remains unchanged until the error is resolved.

UC-12: Meal Prep

Related Requirements:
REQ-3 / REQ-7

Initiating Actor:
Chef

Actor's Goal:
Efficiently prepare meals in accordance with customer order, and notify the waitstaff when dishes are ready for serving.

Participating Actors:
Kitchen Queue System, Notification System, Waitstaff, .

Preconditions:
The restaurant must be open and actively serving customers.
Necessary ingredients and equipment should be available in the kitchen.
The Kitchen Queue System is functional and displays incoming orders.
The Notification System is set up to alert waitstaff.

Postconditions:
Meals are prepared and ready to be served.

Order status is updated to “Prepared” in the order management system and the waitstaff has been notified that the dish is ready for serving.

Flow of Main Success Scenario:

1. ← The chef accesses the Kitchen Queue System to view incoming orders.
2. ← Orders are displayed with a timestamp indicating when they were placed.
3. ← The chef begins preparing the dishes based on the order of arrival and priority.
4. ← Once a dish is ready, the chef uses the Notification System to alert the relevant waitstaff.
5. ← The waitstaff receives the notification and proceeds to serve the dish to the customer.

Flow of Events for Alternate Success Scenario:

1. ← The chef accesses the Kitchen Queue System to view incoming orders.
2. ← The system fails to display new orders due to a technical glitch.
3. ← The chef informs the management about the system error.
4. ← Until the system is restored, manual communication may be required between the waitstaff and the chef to manage orders.

*UC-13: Create a Rewards Account

Related Requirements:
NFREQ-12, USREQ-4

Initiating Actor:
Customer

Actor’s Goal:
To avail discounts based off their spending at the restaurant

Participating Actors:
Database

Preconditions:
The user gets into and loads the system
The user plans to complete a purchase

Postconditions:
The user has earned an appropriate amount of points based on how much they spent

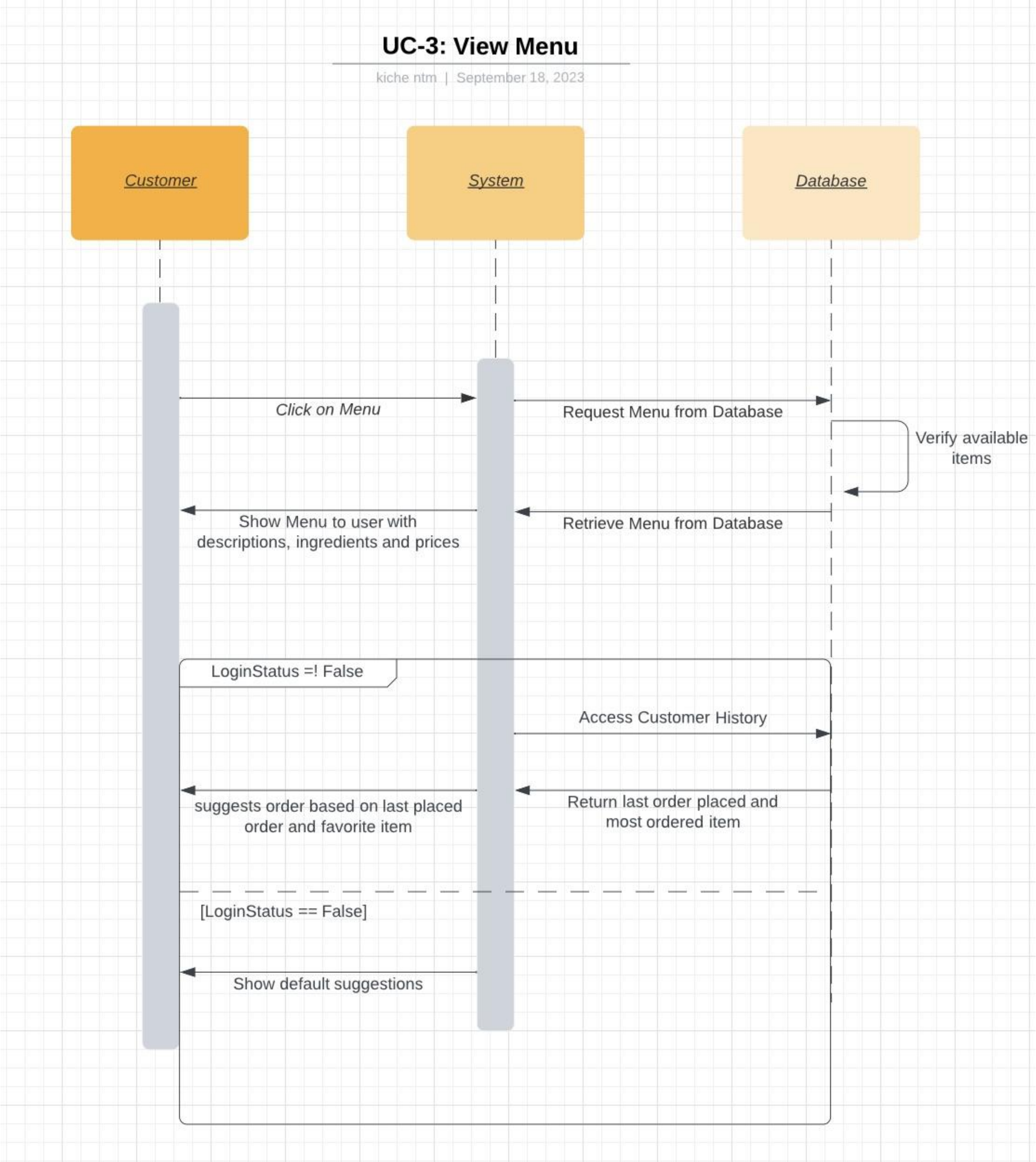
Flow of Main Success Scenario:

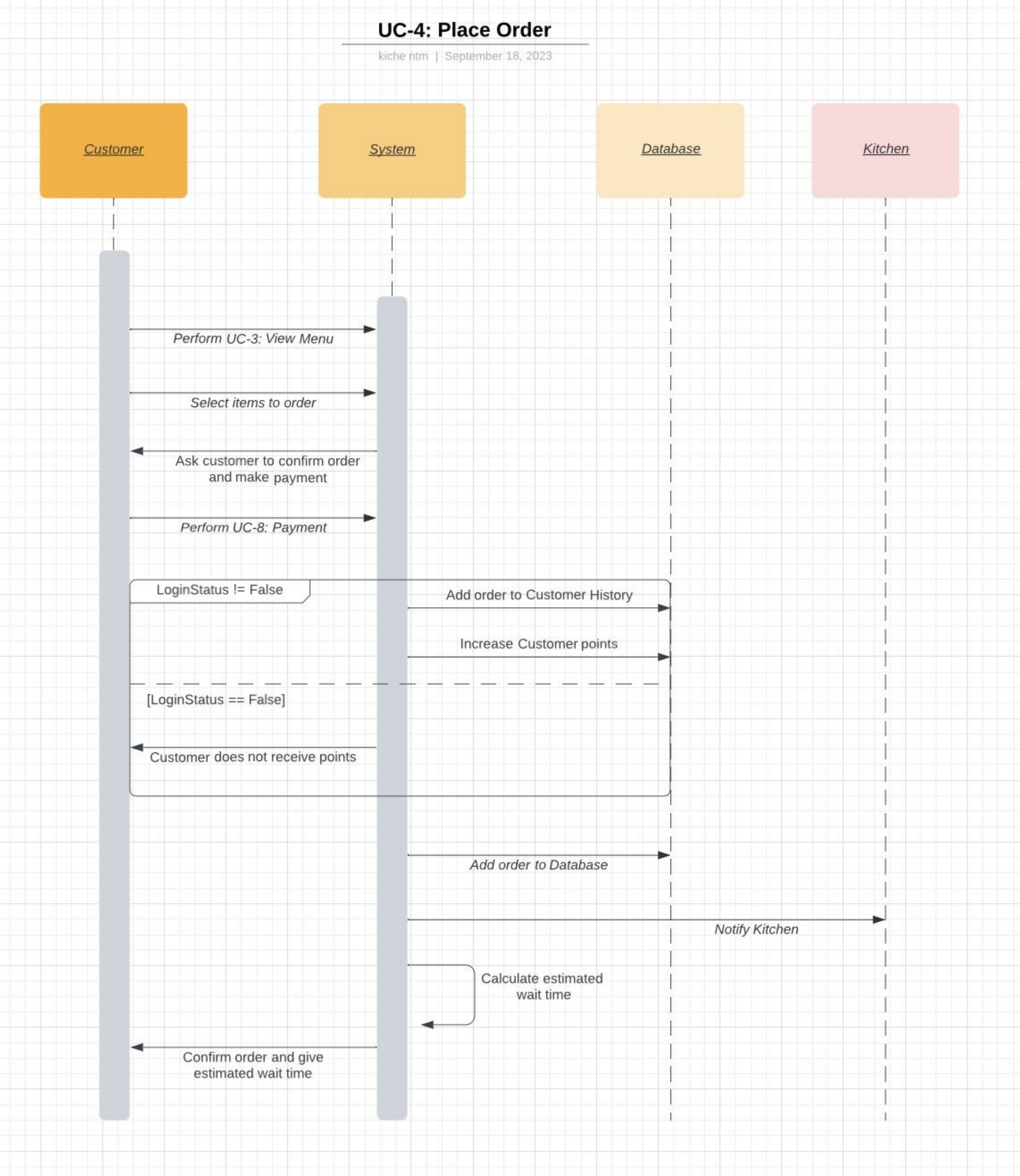
1. ← At the beginning, the system prompts the customer to either create an account or log in.
2. ← The database will retrieve the customer's information with their current point balance.
3. ← The customer selects their desired food items and proceeds to confirm the order.
4. ← The customer enters payment.
5. ← The database is updated with the customer's points based on their transaction.

Flow of Events for Alternate Success Scenario:

1. ← At the beginning, the system prompts the customer to either create an account or log in.
2. ← The database will retrieve the customer's information with their current point balance.
3. ← The customer selects their desired food items and proceeds to confirm the order.
4. ← The customer does nor completes or enters payment.
5. ← The database indicates that the transaction is incomplete and points have not been added.

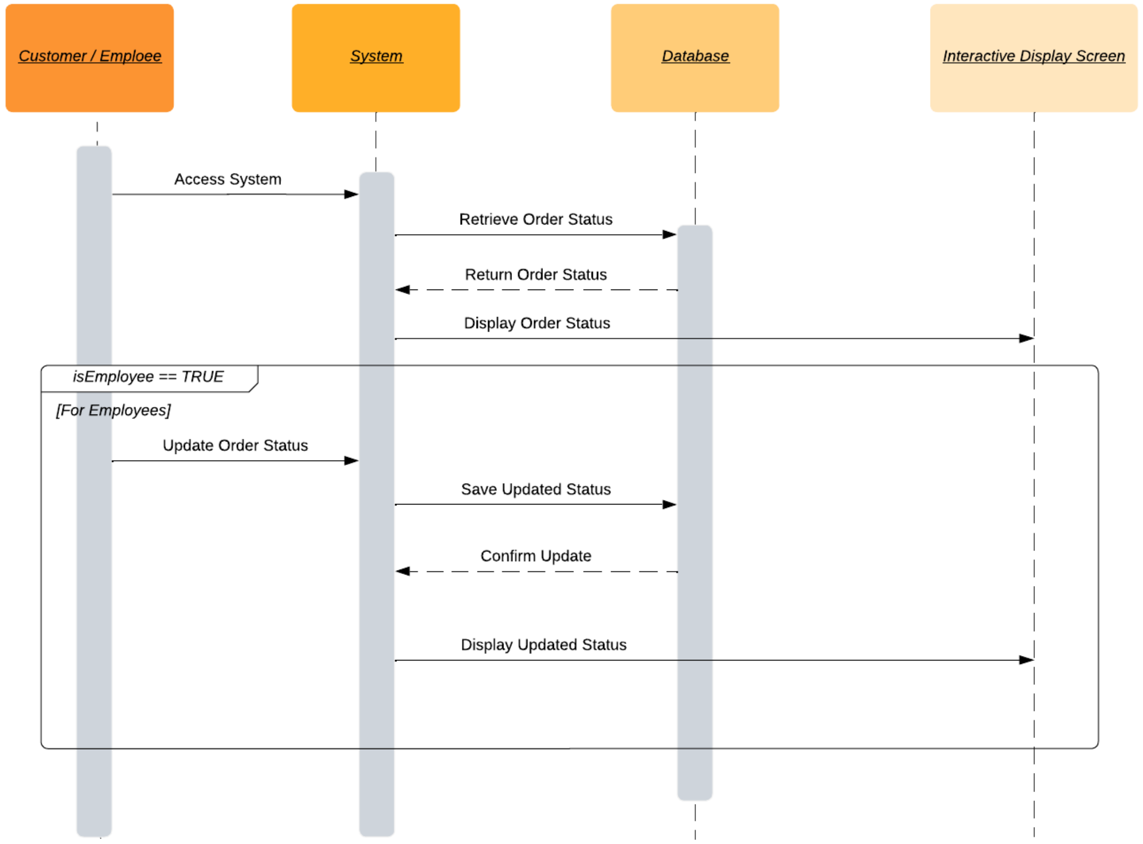
v. System Sequence Diagrams
*UC-13: Create a Rewards Account sequence diagram was not completed for Demo #2.

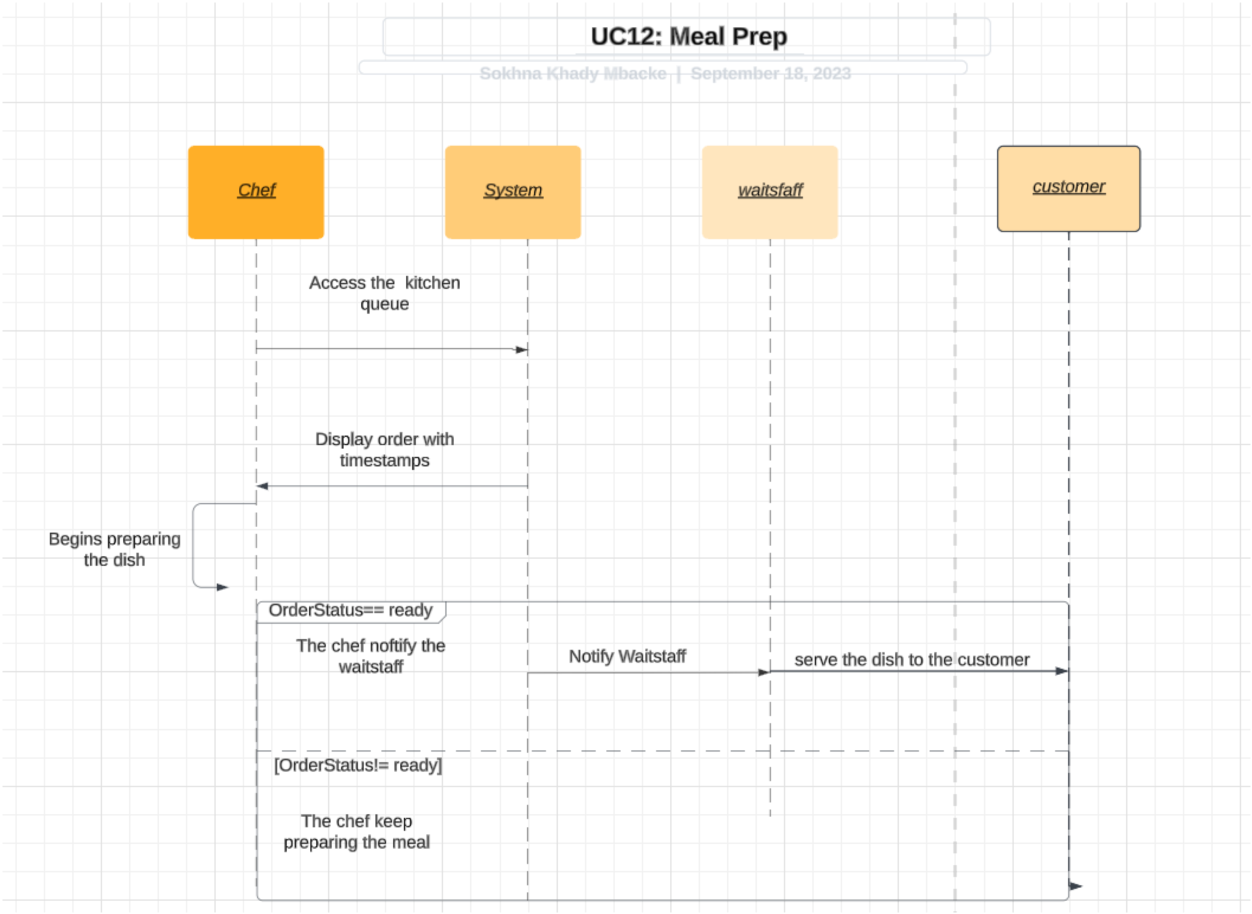

UC-3: View Menu
kiche ntm | September 18, 2023
Customer
System
Database
Click on Menu
Request Menu from Database
Verify available items
Show Menu to user with descriptions, ingredients and prices
Retrieve Menu from Database
LoginStatus =! False
Access Customer History
suggests order based on last placed order and favorite item
Return last order placed and most ordered item
[LoginStatus == False]
Show default suggestions

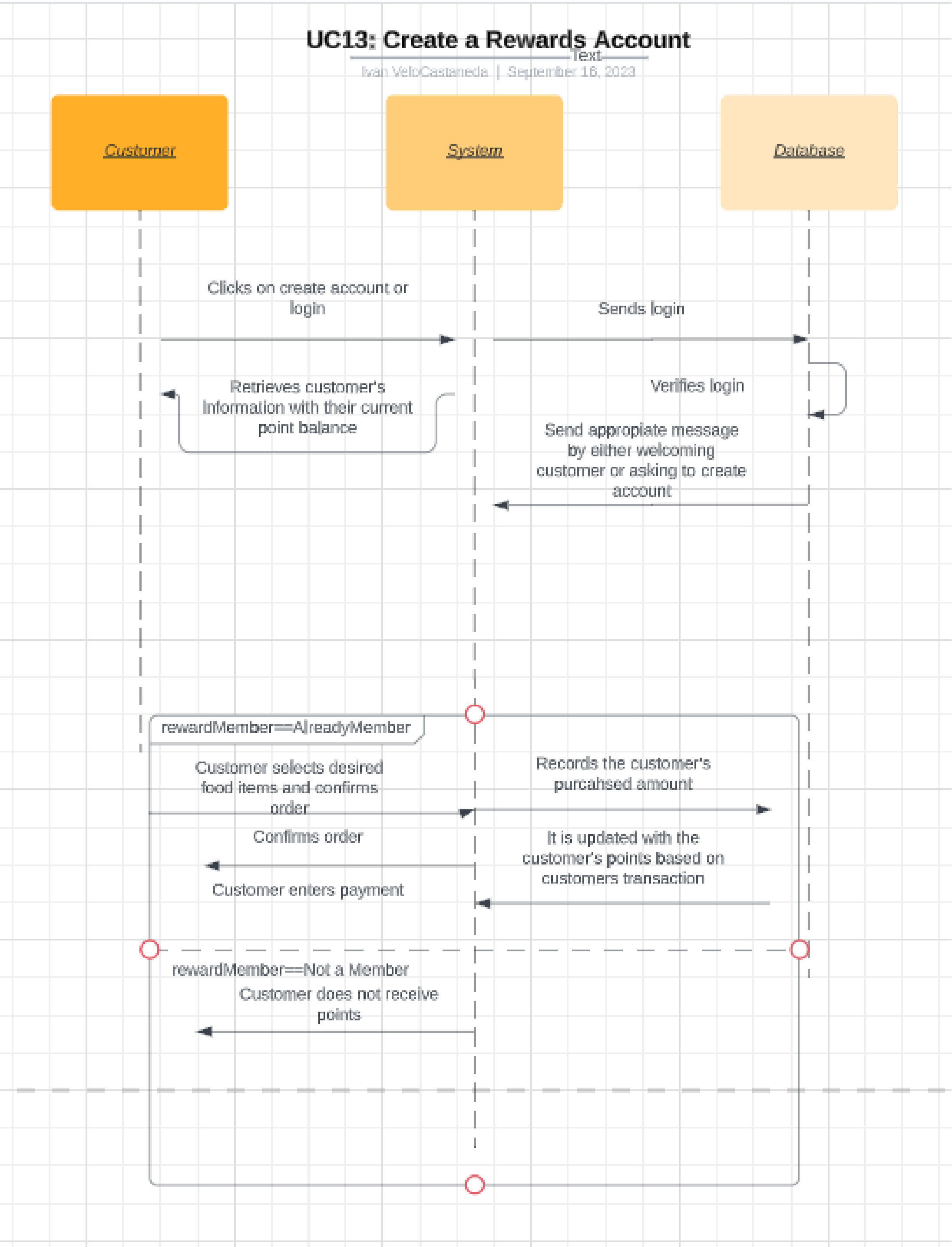

UC-4: Place Order
kiche ntm | September 18, 2023
Customer
System
Database
Kitchen
Perform UC-3: View Menu
Select items to order
Ask customer to confirm order and make payment
Perform UC-8: Payment
LoginStatus != False
Add order to Customer History
Increase Customer points
[LoginStatus == False]
Customer does not receive points
Add order to Database
Notify Kitchen
Calculate estimated wait time
Confirm order and give estimated wait time

UC-10: Order Status

Bjarni Jonsson | September 18, 2023




UC12: Meal Prep
Sokhna Khady Mbacke | September 18, 2023
Chef
System
waitsfaff
customer
Access the kitchen queue
Display order with timestamps
Begins preparing the dish
OrderStatus== ready
The chef noftify the waitstaff
Notify Waitstaff
serve the dish to the customer
[OrderStatus!= ready]
The chef keep preparing the meal


UC13: Create a Rewards Account
Text
Ivan VeloCastaneda | September 16, 2023
Custumer
System
Database
Clicks on create account or login
Sends login
Retrieves customer's information with their current point balance
Verifies login
Send appropiate message by either welcoming customer or asking to create account
rewardMember==AlreadyMember
Customer selects desired food items and confirms order
Records the customer's purcahsed amount
Confirms order
It is updated with the customer's points based on customers transaction
Customer enters payment
rewardMember==Not a Member
Customer does not receive points

4. User Interface Specification(updated)

1. Preliminary Design

   1. **View Menu User Interface Specification**

      a. After successfully signing in to their role-specific interface, managers and servers will be able to navigate through the restaurant's menu using intuitive categories which will be available to them by clicking on the "View Menu" button. Each menu item will be presented with essential details such as name, description, price, and a detailed list of ingredients.

   2. **Place Order User Interface Specification**

      a. Within their role specific interface, managers and servers will have the capability to manage customers orders and navigate the restaurant's table layout. When the user clicks on the "Place Order" directs users to the table layout view, which presents an overview of all the tables within the restaurant. Users can interact with this layout by selecting tables to take customer orders.

      b. Furthermore, users can efficiently manage their orders by clicking on the "View Tables" button, which displays a list of current tables and their associated details.

   3. **Order Status User Interface Specification**

      a. Within their role specific interface, managers and servers will have the capability to see the status of their existing orders in the restaurant. When the user clicks on the "Order Status" button, they will be able to update or check the status of orders in real-time. This will allow wait staff and customers to stay informed about order progress and effectively manage customer expectations throughout their dining experience. It is important to note that customers will not have the capability to update orders. They will only be able to see their order progress.

   4. **Meal Prep User Interface Specification**

      a. Within their role specific interface, chefs will be able to see incoming orders sent by servers. The Chef's specific interface will enable chefs to view detailed order information and effectively communicate with servers when an order is prepared and ready for delivery to the respective table. When they click on the "Orders" button, they will see detailed information about each order, including meal items, special requests, and table number.

b. Once a table's order is ready, chefs will have the ability to let a server know when their food is ready by clicking on the "Ready" button. This action notifies the respective server that the order is prepared and ready to be served to the designated table.

5. **Create a Rewards Account User Interface Specification**
    a. Creating a rewards account will allow customers to earn points through purchases which will earn them points for recent transactions. They will have access to see a detailed transaction history detailing earned and redeemed for transparency.
    b. After creating a rewards account and log in using their credentials, a dashboard displaying the user's current points balance and summary of their rewards program status. There will be a clear visualization of earned, redeemed, and expiring points.
    c. Customers will also be able to manipulate their user profile settings which can be used to update their contact information, change their password, etc. Such a program aims to empower customers and foster their customer engagement and loyalty within the restaurant's rewards program.
6. **Reports User Interface Specification**
    a. A sales report that provides a summary of daily, weekly, or yearly sales.
    b. An inventory report that tracks the levels of ingredients in the kitchen and supplies.
    c. A labor cost report that analyzes salaries and wages. It will also compare labor costs to sales to determine labor cost percentages. Lastly, it would aid to optimize staffing levels to maintain efficiency and control costs.
    d. A food cost report that calculates the cost of ingredients and compares food costs to revenue to calculate food cost percentage. It will help adjust menu pricing and control food expenses.
    e. A profit and loss report that provides an overview of revenues, costs, and expenses. It will help analyze net profit or loss over a specific period.

f. A customer feedback report that summarizes reviews, comments, and ratings. It will help identify areas of improvement based on customer feedback and enhance customer satisfaction and the refinement of services.

g. A payment and transactions report that will track payment methods (cash, credit card, rewards payments) used by customers. It will provide an insight into popular payment methods.

2. User Effort Estimation

Figure 1: Home Page of the System

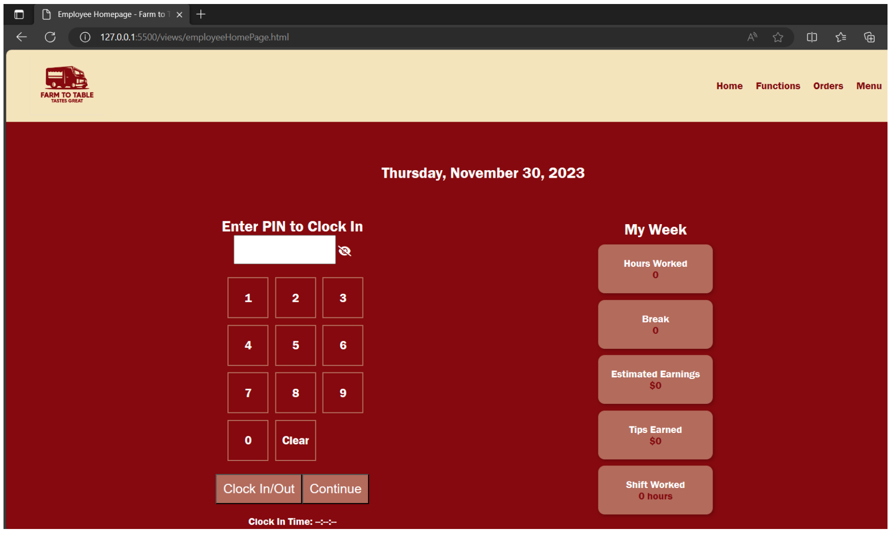


Figure 2: Servers/Manager Interface

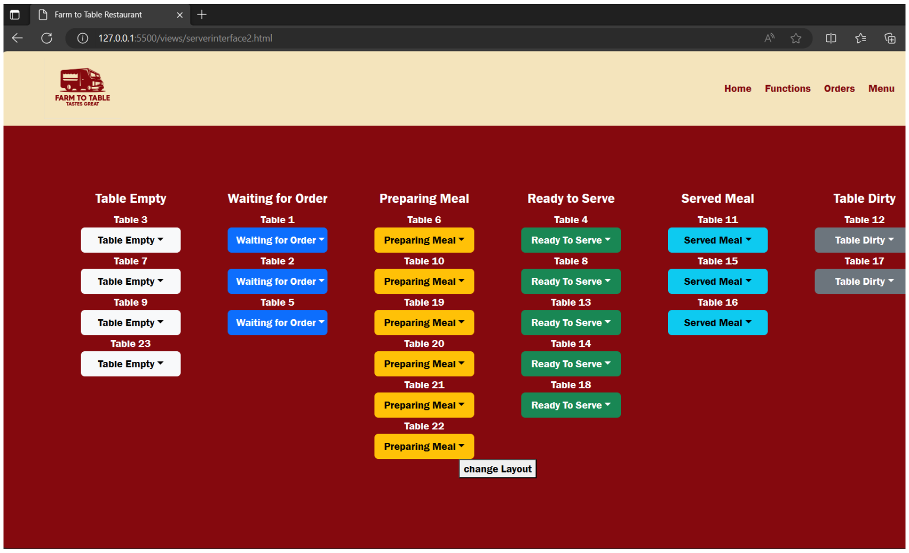
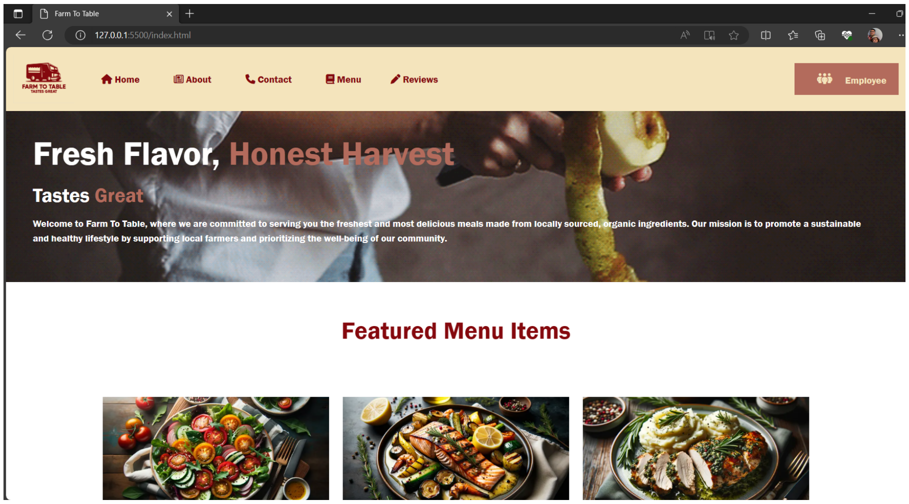


Table Empty	Waiting for Order	Preparing Meal	Ready to Serve	Served Meal	Table Dirty
Table 3 Table Empty ▾	Table 1 Waiting for Order ▾	Table 6 Preparing Meal ▾	Table 4 Ready To Serve ▾	Table 11 Served Meal ▾	Table 12 Table Dirty ▾
Table 7 Table Empty ▾	Table 2 Waiting for Order ▾	Table 10 Preparing Meal ▾	Table 8 Ready To Serve ▾	Table 15 Served Meal ▾	Table 17 Table Dirty ▾
Table 9 Table Empty ▾	Table 5 Waiting for Order ▾	Table 19 Preparing Meal ▾	Table 13 Ready To Serve ▾	Table 16 Served Meal ▾	
Table 23 Table Empty ▾		Table 20 Preparing Meal ▾	Table 14 Ready To Serve ▾		
		Table 21 Preparing Meal ▾	Table 18 Ready To Serve ▾		
		Table 22 Preparing Meal ▾			

Figure 3: Customer Interface



Fresh Flavor, Honest Harvest

Tastes Great

Welcome to Farm To Table, where we are committed to serving you the freshest and most delicious meals made from locally sourced, organic ingredients. Our mission is to promote a sustainable and healthy lifestyle by supporting local farmers and prioritizing the well-being of our community.

Featured Menu Items

1. **Clock in/Clock out:** The following sequence of actions would allow an employee with the code “6254” to either clock in or clock out with a total of 2 mouse clicks and 4 keystrokes.
   NAVIGATION: Total 2 mouse clicks, as follows
   a. Click on the “Clock in” or “Clock out” button.
   b. Click on the “Confirm” button.
   DATA ENTRY: Total of 4 keystrokes, as follows
   a. Press “6”
   b. Press “2”
   c. Press “5”
   d. Press “4”
2. **Log in:** The following sequence of actions would allow an employee with the code “6254” to log in into the system to place an order with a total of 1 mouse click and 4 keystrokes
   NAVIGATION: Total 1 mouse click, as follows
   a. Click on the “Continue” button.
   DATA ENTRY: Total of 4 keystrokes, as follows
   a. Press “6”
   b. Press “2”
   c. Press “5”
   d. Press “4”
3. **View Menu:** The following sequence of actions would allow an employee to view the menu and item ingredients with a total of mouse 1 click and 0 keystrokes.
   NAVIGATION: Total 1 mouse click, as follows
   a. Click on the “Menu” button.
   DATA ENTRY: None

4. **Place Order:** The following sequence of actions would allow an employee with the code “6254” to place an order for table 1 with a total of 8 mouse clicks and 4 keystrokes. The following order is as follows:

   2 coffees and 2 classic burgers with french fries.

   NAVIGATION: Total 8 mouse clicks, as follows

   a. Click on the “Continue” button.
   b. Click on the “Table 1” icon.
   c. Click on the “Dine in” button or “To go” button.
   d. Click on the “Coffee” button twice under the “Beverages” section.
   e. Click on the “Classic Burger” button twice under the “Burgers/Sandwiches” section.
   f. Click on the “French Fries” button twice under the “Sides” section.
   g. Click on the “Confirm” button.
   h. Click on the “Send Order” button.

   DATA ENTRY: Total of 4 keystrokes, as follows

   i. Press “6”
   j. Press “2”
   k. Press “5”
   l. Press “4”

   If the user is already signed in, then the data entry portion of this scenario would not be necessary.

5. **View and Update Table Status:** The following sequence of actions would allow an employee with the code “6254” to view and update the status of a table with a total of 5 mouse clicks and 4 keystrokes. In this case, we want to update the status of table 1 from “dirty” to “unoccupied”.

   NAVIGATION: Total 6 mouse clicks, as follows

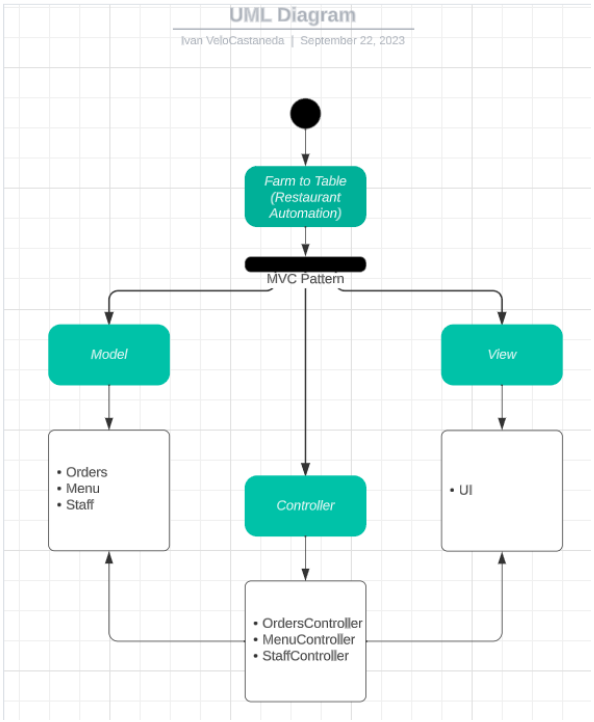
   a. Click on the “Continue” button.
   b. Click on the “Table 1” icon.
   c. Click on the “Update Status” button.
   d. Select “Unoccupied” from the drop down menu.
   e. Click “Confirm”.

   DATA ENTRY: Total of 4 keystrokes, as follows

   f. Press “6”
   g. Press “2”
   h. Press “5”
   i. Press “4”

5. System Architecture

a. Identifying Subsystems



The subsystem above displays our three layer architecture which consists of Model, View, and Controller. Each layer has a specific responsibility to the entire system. The Model consists of the data business logic of the system, including orders, menus, and staff. The View contains the user interface responsible for displaying the menu, taking orders, and managing employees. It is also responsible for user interactions. Lastly, the Controller acts as an intermediary between the model and view, handling user input, updating the model, and ensuring the view reflects the changes accordingly. There are specific controllers for different entities like orders, menus, and staff, managing the flow of data and actions.

b. Architecture Styles

For our website, we will use the Model-view-controller pattern, MVC pattern, a widely adopted architectural approach for web development. Since some team members are already familiar with this pattern, it is a well grounded choice. Given that a restaurant system involves a diverse functionalities such as order processing, menu management, and staff operations, the MVC pattern will provide an organized framework to handle these tasks efficiently. It will allow for flexibility and scalability, enabling different members to work on separate components simultaneously and accommodate changes in the restaurant's operations. Restaurants often have different user interfaces for customers, staff, and management. It will also allow the development of multiple views tailored to each user group, providing a more personalized and intuitive user experience. Lastly, it will allow management to modify or update specific components without impacting the entire system as they may need to make changes to menu offerings, or operational processes, leading to cost-effective maintenance and upgrades.

In addition to MVC, our website will adopt the Client-Server model, an architecture where the server centrally manages and provides data to various client interfaces. The system will include different interfaces for customers, servers, chefs, and managers. In our case, these interfaces will cater to customers, servers, chefs, and managers, each offering distinct functionalities such as ordering, making reservations, and menu management. The one-to-many relationship in the Client-Server model will ensure efficient data handling and interaction.

Furthermore, our website will integrate the Peer-to-Peer model to streamline customer payments. This model will enable separate processing of payments, facilitating secure fund transfers from the customer’s bank account to the restaurant’s account. This approach will optimize the payment process, enhancing the overall transaction experience.

c. Mapping Subsystems to Hardware

Yes, our system needs to run on multiple computers. Here's the breakdown:

1. **Server Subsystem (Node.js with Express and MySQL)**: This will be hosted on a dedicated machine. This subsystem is responsible for handling all the business logic, database operations, and serving API endpoints. Given the need for fast response times and the potential for high concurrent requests, especially during peak restaurant hours, it's crucial to have this on a robust server machine.
2. **Database Subsystem (MySQL)**: The database will be hosted on a separate dedicated machine. This separation ensures that the database operations do not bottleneck the server's performance, and it also provides an added layer of security. By isolating the database, we can implement stricter access controls and monitoring.
3. **Client Subsystem (Web Browser)**: This will run on various devices, including desktops, laptops, tablets, and smartphones. The client subsystem is responsible for rendering the user interface and interacting with the server subsystem via API calls. It will cater to different user interfaces for customers, staff, and management.

d. Connectors and Network Protocols

Given that our system runs on multiple machines, we will primarily use the following communication protocols:

1. **HTTP/HTTPS**: This will be the primary protocol for communication between the client (web browser) and the server (Node.js with Express). We chose HTTP/HTTPS because it's a standard protocol for web applications, ensuring compatibility across various devices and browsers. HTTPS will be used to encrypt the data during transmission, providing an added layer of security, especially for sensitive operations like payments and user authentication.
2. **MySQL Protocol (via JDBC in Node.js)**: For communication between the server subsystem and the database subsystem, we will use the MySQL protocol through JDBC connectors in Node.js. This choice is due to our use of MySQL as the database system. The JDBC connector provides a standardized way to connect Node.js applications to MySQL databases, ensuring efficient and secure data operations.
3. **Peer-to-Peer Protocols**: For the Peer-to-Peer model used in customer payments, specific protocols will be adopted depending on the payment gateway or service we integrate. This ensures secure and direct fund transfers between customer accounts and the restaurant's account.

e. Global Control Flow

- **Execution orderliness:** The system follows an event-driven fashion. It does not follow a strict linear procedure where every user has to go through the same steps. Instead, it waits for events triggered by user interactions. Users can generate actions in different orders based on their needs and preferences.
- **Time dependency:** There are no specific timers in the system, and it is not a real-time system with strict time constraints. It responds to events as they occur, without periodic time requirements.

f. Hardware Requirements

The following hardware requirements are essential to ensure the reliable and efficient operation of our system.

- **Screen display**: The client subsystem (web browsers) requires screens with varying resolutions, as it runs on devices such as desktops, laptops, tablets, and smartphones. Specific display requirements may vary based on device types and screen sizes.
- **Disk storage:** Both the server subsystem and the database subsystem will require adequate disk storage.The requirements will depend on the volume of data and system usage.

- **Communication network:** the system relies on a stable communication network with a minimum network bandwidth of 56 Kbps for smooth data exchange between clients and the server, especially during peak restaurant hours.
- **Database server:** The dedicated machine hosting the MyQSL database subsystem should have sufficient computing resources to handle database operations efficiently.
- **Security Measures:** Adequate security measures, such as firewalls, intrusion detection systems, and encryption protocols should be in place to safeguard data integrity and protect against threats.

6. Effort Estimation Based on use Case Points(Updated)

a, Unadjusted Actor Weight

Actor Type	Description of how to recognize the actor type	Weight
Simple	The actor engages via the specified API, which stands for the application programming interface.	1
Average	The actor is an individual engaging via a protocol or using a text-based user interface.	2
Complex	The actor engages with the user interface.	3

Actor Name	Description of Relevant Characteristics	Complexity	Weight
Customer	Customer interacts with the system	Complex	3
Employees(Chef , Servers, Managers,	Same as Customer	Complex	3

Hosts)			
Database	Database is another system interacting through protocol	Average	2

UAW(Farm to table) = 1* Average + 2 * Complex = 1 * 2 + 2 * 3 = 8

b. Unadjusted Use Case Weight

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

	of steps for the success scenario : >=7	

Identifier	Number of actors	Number of requirements	Complexity	Weight
UC-1: Clocking in /Clocking out	2: Employees and database	1: REQ-6	Simple	5
*UC-2: Log in/Log out	3: Employees, customers and database	1: REQ-10	Average	10
UC-3: View Menu	3: Employees, customers and database	2: NFREQ-10 and USREQ-1	Complex	15
UC-4: Place Order	3: Employees, customers and database	4: REQ-1, REQ-3, NFREQ-13, USREQ-1	Complex	15
UC-5: View/Update Table Status	3: Employees and customers	1: USREQ-2	Average	10
*UC-6: Make/View all reservations	3: Employees, customers and database	1: NFREQ-11	Average	10
*UC-7: Print Out Reports	2: Manager and database	1: REQ-4	Simple	5
*UC-8: Payment	3: Employees, customers and database	1: REQ-8	Average	10
*UC-9: Checking guests are happy/Rating	2: Customers and database	2: NFREQ-6 and NFREQ-9	Average	10

UC-10: Order status	2: Customers and employees	3: REQ-2, REQ12, and USREQ-3	Complex	15
UC-11: Add/Remove employees from the system	2: Manager and database	1: REQ-12	Simple	5
UC-12: Meal Prep	2: Chefs and employees	2: REQ-3 and REQ-7	Average	10
**UC-13: Create a Rewards Account	2: Customers and database	2: NFREQ-12 and USREQ-4	Average	10
UC-14: Menu Updates	2: Manager and database	1: REQ-5	Simple	5
**UC-15: Select Language	2: Customers and database	1:NFREQ-9	Simple	5

UCW(Farm to Table) = 5*Simple + 7*Average + 3*Complex = 5 * 5 + 7 * 10 + 3 * 15 = 25 + 70 + 45 = 140

c. Technical Complexity Factors

Technical factor	**Description**	**Weight**	**Perceived Impact**	**Calculated Factor(Weight * Perceived Impact)**
T1	User expects good performance but nothing out of the ordinary	1	3	3
T2	End Users expect efficiency but no special	1	3	3

	demands			
T3	Ease of use is quite crucial to every user	0.5	5	2.5
T4	Easy to make changes	1	3	3
T5	No need for unique training	1	0	0
T6	Restrictive access for third parties	1	0	0
T7	There is no requirement for reusability	1	1	1
T8	Coexisting use is required	1	4	4
Environmental Factor Total				16.5

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

d. Environmental Complexity Factors

Environmental factor	**Description**	**Weight**	**Perceived Impact**	**Calculated Factor(Weight * Perceived Impact)**
E1	Beginner familiarity with the UML-based development	1.5	1	1.5
E2	Some familiarity	0.5	2	1

	with application problem			
E3	Some knowledge of object-oriented approach	1	2	12
E4	Beginner lead analyst	0.5	1	0.5
E5	Stable requirements expected	2	5	10
E6	Programming language or average difficulty will be used	-1	3	-3
Environmental Factor Total				12

ECF= 1.4-0.03*12=1.04

e. Use Case Points

UCP= UUCP* TCF* ECF From the above calculations, the UCP variables have the following values: UUCP= UAW + UUCW = 8 + 140 = 148 TCF=0.765 ECF=1.04 For the case study, the final UCP is the following: UCP= 148*0.765*1.04 = 117.75 or 118 use case points

f. Duration

Duration = UCP * PF = 118 * 20 = 2,360 hours

7. Analysis and Domain Modeling(updated)

a. Conceptual Model

i. Concept definitions

Responsibility	**Type**	**Concept**
R1:	Facilitate communication among customer, chef, server, and busboy	Communicator
R2:	Show choices for customer, waiter, chef, and manager	Interface
***R3:**	Guide the customer in choosing a table	Table Status
***R4:**	Block selection of unavailable tables	Table Status
***R5:**	Ensure reserved tables are not chosen before the reserved time	Table Status
R6:	Update table status upon customer actions (selection, departure, cleaning by busboy)	Table Status
R7:	Design and layout of the restaurant's seating and table arrangement	Floorplan
***R8:**	Update floorplan based on changes in table arrangements or renovations	Floorplan
R9:	Display a visual representation of the restaurant's layout to staff and customers	Floorplan

R10:	Ensure efficient flow and movement within the restaurant based on the floorplan	Floorplan
R11:	Line up orders for chef's preparation	Order Queue
R12:	Show orders that are ready to be served	Serving
R13:	Indicate when an order is received, in preparation, ready, or delivered	Order Status
***R14:**	Notify the customer of any changes in their order status	Order Status
R15:	Allow staff to update the status of an order	Order Status
R16:	Provide a history of all order statuses for a particular order	Order Status
***R17:**	Oversee transaction processing	Payment System
R18:	Manage the flow of data between the user interface and the system	Controller
R19:	Ensure proper routing of user requests to appropriate system components	Controller
R20:	Validate user inputs before processing	Controller
R21:	Handle errors and provide appropriate feedback to users	Controller

R22:	Oversee database interactions	DB Connection
R23:	Maintain employee login details and work hours	Employee Account
***R24:**	Save customer login details	Customer Profile
***R25:**	Keep track of customer's reward points based on past orders	Customer Profile
***R26:**	Showcase favorites, top picks, and frequently ordered items in customer/user profile	Customer Profile
R27:	Record the customer's order in the database	Customer Profile
***R28:**	Manage and allocate reward points to customers based on their purchases	Reward System
***R29:**	Offer special promotions or discounts to customers through the reward system	Reward System
***R30:**	Allow customers to redeem their reward points for meals or discounts	Reward System
***R31:**	Provide a dashboard for customers to view and manage their reward points	Reward System

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

ii. Association definitions

Concept Pair	**Association Description**	**Association Name**
Customer Account ⇔ DB	Retrieve customer’s data	DBQuery

Connection	from the database	
Customer Account ⇔ Interface	Display customer’s options	Display
Interface ⇔ Controller	Allows users to interact with the system	User Action
Communicator ⇔DB Connection	Modify or insert data into the database	DBUpdate
Communicator ⇔ DB Queue	Allows user to send orders and queue to the database	DBQuery
Controller ⇔ Order Status	Allow the staff to update or view the status of their food orders	Update Order Status View Order Status
Controller ⇔ Table Status	Allows users to view the status of a table	View Table Status
Controller ⇔ Payment System	Allows users to complete payments	Make Payment
Payment System ⇔ DB Connection	Stores payments records of past transactions in the database	Record Payment
Interface ⇔ DB Connection	Retrieve the data from the database for the users	DBQuery
Customer Profile ⇔ DB Connection	Store earned points in the database	Reward System
Table Status ⇔ Interface	Displays the table layout with the proper status of the tables	Display
Employee Account ⇔ Interface	Displays the employee’s actions	Display
Payment System ⇔ Customer Account	Update the points balance for the users	Update Rewards

Floorplan ⇔ Controller	Allow the manager to adjust the table layout of the restaurant	Floorplan Layout

iii. Attribute Definitions

Concept	**Attribute**	**Description**
Customer Account	userEmail userPassword	Associated email and password of the customer. Guest account is assigned if there is no account.
Interface	sendOrder Receipt rateFood tableStatus	Allow the user to send orders after choosing food items. Allow the user to choose which way they would like to receive their receipt(email, paper) Allow the user to rate their meal and then have that rating stored and displayed Provides the user with the up-to-date status of each table in the restaurant. 2 possible states, “occupied” and “empty”
Payment System	paymentMade	Updates the system on whether or not a payment has been made
Order Status	orderStatus orderReady createdAt updatedAt	Allows the chef to update the status of orders being cooked. Allows the chef to send a signal to the waiter/waitress and customers that the order is finished.

		Allows the chef to see what time the order was sent. Allows the chef to see what time the order was updated.
Order Queue	chefQueue	Keeps track of incoming food orders in the order that they were submitted for the chef to follow.
Controller	tableList tableConfirm	Shows the employees the current tables available. Table is marked in green once it is picked by an employee and it turns back to white when customers leave.
Communicator	customerOrder	Each customer order is stored in the database.
Floorplan	viewPlan tableAdjust	Displays the current layout of the restaurant. Allows the manager to adjust the layout by either adding tables or deleting tables.
Employee Account	hoursWorked timeClockedIn timeClockedOut role	Display the employee’s worked hours in a particular payment cycle. Display the employee’s id. Display the time at which the employee clocked in on a particular day. Display the time at which the employee clocked out on a particular day. Display the role of the employee.

Table Status	tableStatus	Display if the table is empty(white) or occupied (green).
Reward System	userBalance redeemPoints	Display the current reward points balance of the user. Allow the user to redeem points.
DB Connection	dbAuthenticationCreds userID	Store the DB login for authorized personnel. Display the employee’s id.

iv. Traceability matrix

Use Case	**PW**	**Domain Concepts**											
		Customer Account	Interface	Payment System	Order Status	Order Queue	Controller	Communicator	Floor plan	Employee Account	Table Status	Reward System	DB Connection
UC-1			X							X			X
UC-2		X	X							X			X
UC-3			X										X
UC-4		X	X		X	X		X				X	X
UC-5			X		X	X	X		X		X		
UC-6		X	X								X		
UC-7										X			X
UC-8		X		X								X	X
UC-9		X											X
UC-10					X	X							

Use Case	PW	Domain Concepts											
UC-11										X			X
UC-12			X		X	X							X
UC-13		X								X			X
UC-14			X							X			X
UC-15			X										X

b. System Operation Contracts

Operation	**View Menu**
Use Case:	UC-3
Preconditions:	● The user logs in to their account. ● Manager and server select the “View Menu” button on the bottom right
Postconditions:	The user is taken to the menu interface

Operation	**Place Order**
Use Case:	UC-4
Preconditions:	● The user logs into their account ● Manager and server select the “Dine in” button or “To Go” button ● Manager and server select the

	table number and enter the number of guest at the table
Postconditions:	The user is taken to the place order interface

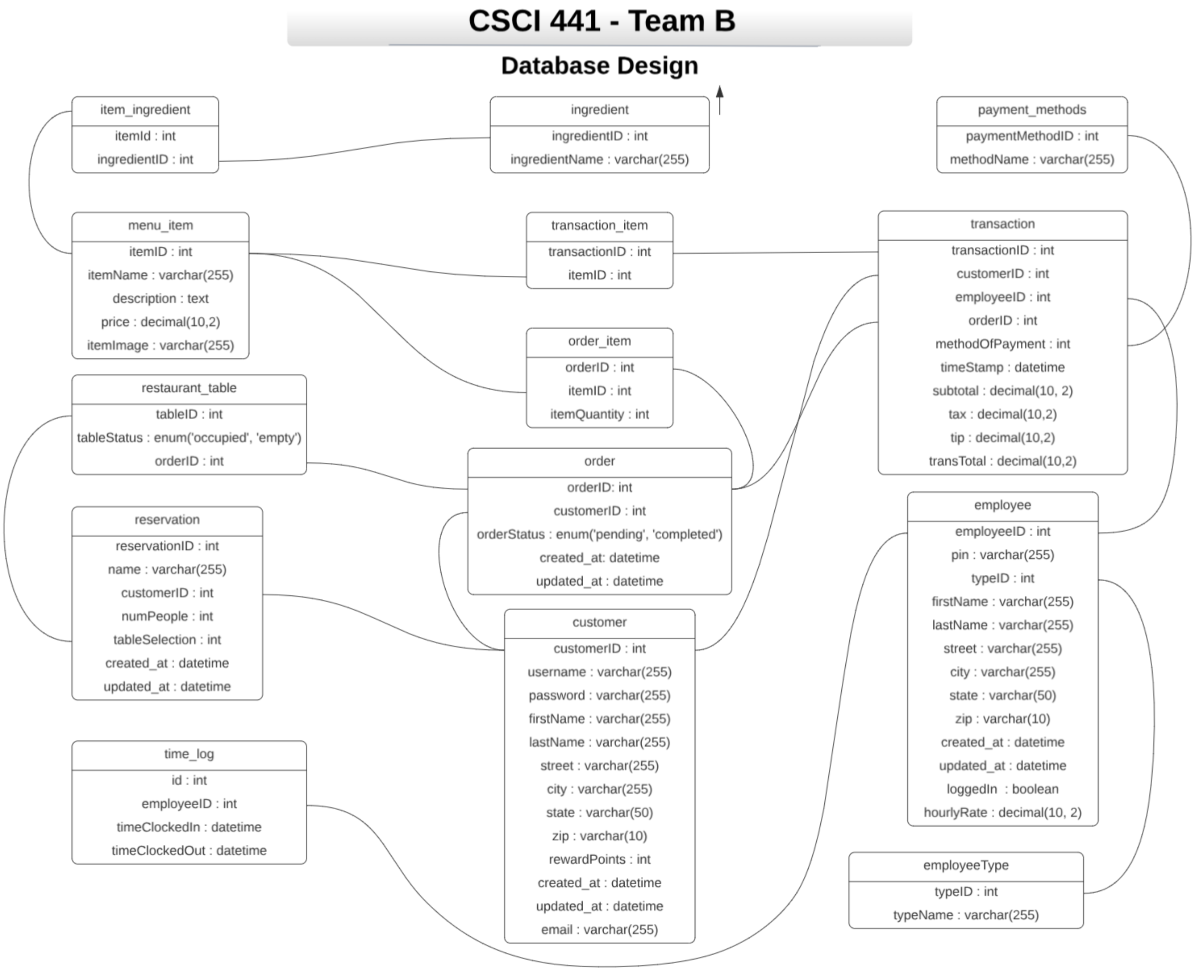
Operation	**Order Status**
Use Case:	UC-10
Preconditions:	● The user logs into their account ● Manager, customer and chef select the “Order Status” button on the bottom left
Postconditions:	The user is taken to the order status interface where their order is displayed

Operation	**Meal Prep**
Use Case:	UC-12
Preconditions:	● The user logs into the system ● The chef selects the “View Orders” button on the bottom left
Postconditions:	The user is taken to the chef interface where all the orders are displayed along with their status

Operation	**Create Rewards Account**
Use Case:	UC-13
Preconditions:	● The user logs into their rewards account

	● The user makes a purchase
Postconditions:	The user has earned rewards points based on how much they spent

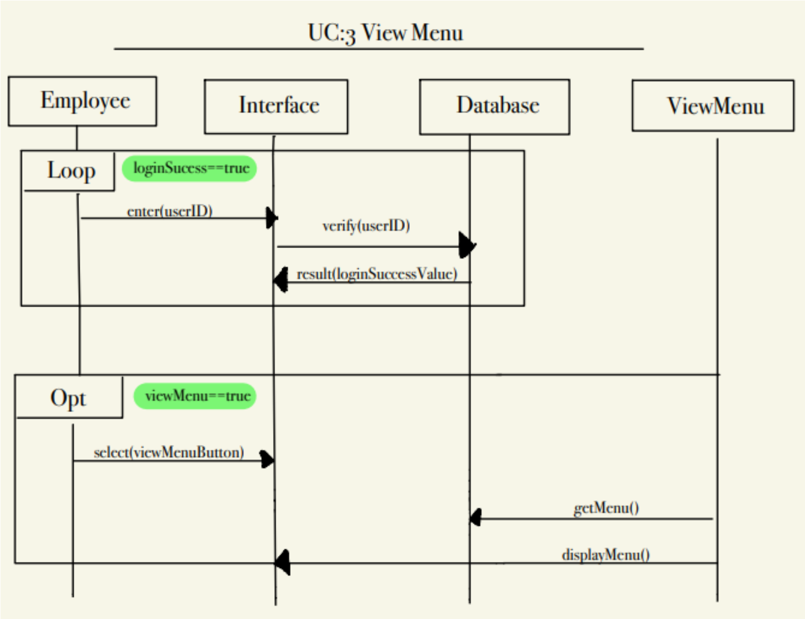
c. Data Model and Persistent Data Storage



Our system will not require persistent data storage beyond a single system execution. Initially, the user's data will be stored in designated tables based on their role. For employees, the system will retain essential information such as first and last names, a unique 4-digit identifier (last four digits of their social security number), salary, hours worked, work schedule, turnover time, sales, credit card tips, cash tips, and food percentages. Customers' data will consist of first and last names, email addresses, passwords, and earned reward points. In the case of managers, the system will store restaurant transactions for easy access, enabling them to track total revenue and profit. Additionally, managers will have the ability to personalize the menu by adding or removing items based on their sales performance. Crucial data to be stored includes aggregate ratings for each menu item, item prices, item names, recipes, and item IDs or numbers. All this data will reside in a MySQL database hosted on our web server.

8. Interaction Diagrams(Updated)

UC:3 View Menu



The diagram demonstrates the interactions in UC-3: View Menu. Here's a breakdown of the described interactions:

- The user accesses the site and initiates the login process
- The system will check whether the user is in the database by verifying their login credentials (userID).
- If the provided credentials match those in the database, the user is verified.
- Based on the user's role(host, waiter, manager, chef, guest), the system will direct the user to the specific interface tailored to that role.
- Lastly, the system will display a "View Menu" option that the users can click to see the menu and the details of it.

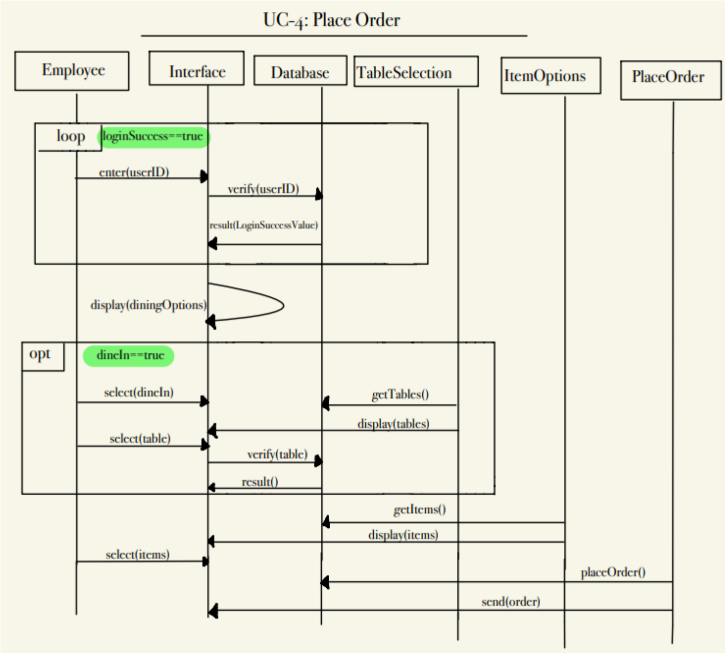
The design principles employed in this sequence are the High Cohesion Principle and the Expert Doer Principle.

- The High Cohesion Principle suggests keeping related functionality grouped together within a module, class, or component.
    - The principle is employed in the "User Authentication" component which consists of handling all tasks related to user verification and validation.
    - The principle is employed in the "Role-Based Interface Handling" function which is responsible for directing the user to the appropriate interface based on their role.
- The Expert Does Principle suggests responsibility is assigned to the class that has the necessary information to fulfill it effectively.

  - ○ The principle is displayed in the “User Authentication” component which will possess the necessary information and methods to authenticate the user effectively against the database.
  - ○ The principle is employed in the “Role-Based Interface Handling” function which will utilize its knowledge to handle this responsibility effectively.
  - ○ The principle is employed when the user clicks on the “View Menu” option which will determine the appropriate action and display the restaurant menu and details of each menu item accordingly.

In this diagram, the Publisher-Subscriber pattern is used to streamline interactions related to the "View Menu" scenario. The system component managing menu-related changes would serve as the Publisher, notifying Subscribers (such as different user interfaces for various roles) about any alterations. Each Subscriber would express interest in receiving menu updates. When menu changes occur, the Publisher alerts relevant Subscribers, ensuring specific interfaces display the updated menu details tailored to users' roles. For instance, after successful authentication and role identification, the system guides users to role-specific interfaces featuring the "View Menu" option. Upon selecting this option, the Publisher would notify respective Subscribers, prompting the display of menu details aligned with the users' roles. By adopting this pattern, the system maintains flexible interactions, allowing smooth updates to be generated across different interfaces without messing with the core menu logic within each interface.

UC-4:Place Order



The diagram demonstrates the interactions in UC-4: Place Order. Here’s a breakdown of the described interactions:

- The user accesses the site and initiates the login process

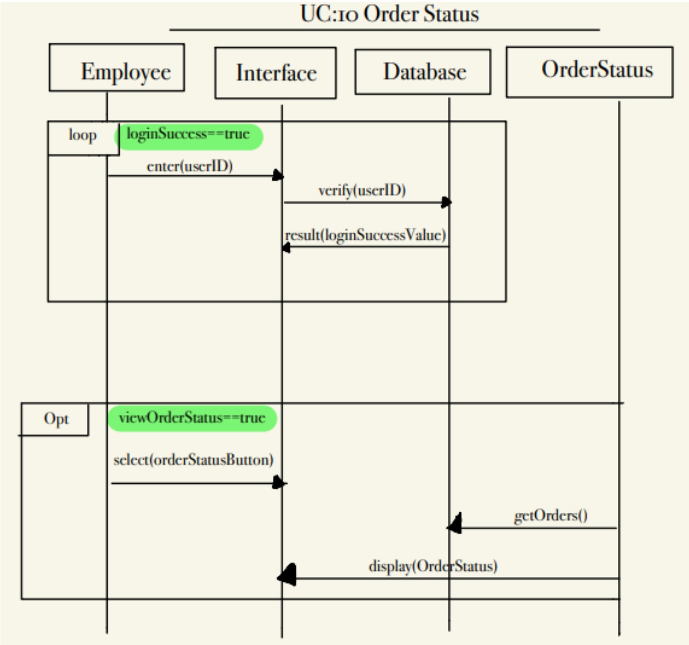
- The system will check whether the user is in the database by verifying their login credentials (userID).
- If the provided credentials match those in the database, the user is verified.
- Based on the user’s role(host, waiter, manager, chef, guest), the system will direct the user to the specific interface tailored to that role.
- The system will display “Dine in”, “To Go”, or “View Menu” options (options will vary depending on the role). After the dining option is selected, the user is brought to the next screen which is seating options. The seating option allows the user to see the full layout of the restaurant and select the table of their choice.
- After selecting the table number, the user will be moved along to the item options screen where the user selects the customer’s food items.
- After selecting all the items, the user will click the “Place Order” button and the order will be sent to the kitchen for preparation.

The design principles employed in this sequence are the High Cohesion Principle and the Expert Doer Principle.

- The High Cohesion Principle suggests keeping related functionality grouped together within a module, class, or component.
  - The principle is employed in the “User Authentication” component which consists of handling all tasks related to user verification and validation.
  - The principle is employed in the “Role-Based Interface Handling” function which is responsible for directing the user to the appropriate interface based on their role.
- The Expert Does Principle suggests responsibility is assigned to the class that has the necessary information to fulfill it effectively.
  - The principle is displayed in the “User Authentication” component which will possess the necessary information and methods to authenticate the user effectively against the database.
  - The principle is employed in the “Role-Based Interface Handling” function which will utilize its knowledge to handle this responsibility effectively.
  - The principle is employed when the user clicks on the “Dine In” or “To Go” options which will determine the appropriate actions and allow the user to place orders and send them to the kitchen for preparation.

In the "Place Order" diagram, the Publisher-Subscriber pattern could optimize interactions. The system component responsible for managing the order placement process, related to item selection, table allocation, and order transmission to the kitchen, might serve as the Publisher. It hangs onto the order status and notifies Subscribers about any modifications or updates. Various interfaces or components, such as those linked to specific user roles, could function as Subscribers, subscribing to receive real-time updates regarding order changes. When a user selects "Dine In" or "To Go" options and proceeds to place orders, the Publisher would notify Subscribers, triggering the appropriate actions and facilitating order transmission to the kitchen. Implementing this pattern ensures seamless communication and updates across various interfaces without embedding order-specific logic within each interface, thereby ensuring system flexibility and scalability.

UC-10: Order Status



The diagram demonstrates the interactions in UC-10: Order Status. Here’s a breakdown of the described interactions:

- The user accesses the site and initiates the login process
- The system will check whether the user is in the database by verifying their login credentials (userID).
- If the provided credentials match those in the database, the user is verified.
- Based on the user’s role(host, waiter, manager, chef, guest), the system will direct the user to the specific interface tailored to that role.
- Lastly, the system will display an “My Orders” option that the users can click to see their current orders along with their status.

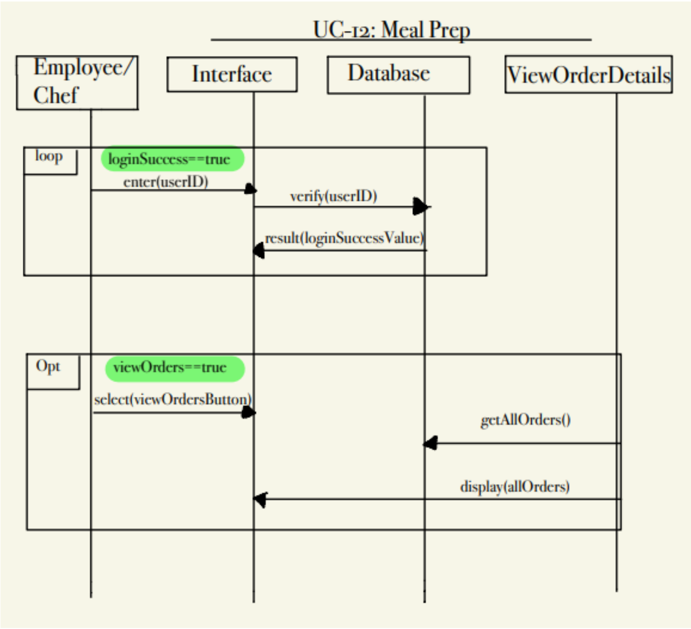
The design principles employed in this sequence are the High Cohesion Principle and the Expert Doer Principle.

- The High Cohesion Principle suggests keeping related functionality grouped together within a module, class, or component.
    - The principle is employed in the “User Authentication” component which consists of handling all tasks related to user verification and validation.
    - The principle is employed in the “Role-Based Interface Handling” function which is responsible for directing the user to the appropriate interface based on their role.

- The Expert Does Principle suggests responsibility is assigned to the class that has the necessary information to fulfill it effectively.
    - The principle is displayed in the “User Authentication” component which will possess the necessary information and methods to authenticate the user effectively against the database.
    - The principle is employed in the “Role-Based Interface Handling” function which will utilize its knowledge to handle this responsibility effectively.
    - The principle is employed when the user clicks on the “My Orders” option which will determine the appropriate action and display the orders of the server or manager along with their information and current status.

In this diagram, the Publisher-Subscriber pattern is used to streamline interactions related to the "Order Status" scenario. The system component managing order-related changes would serve as the Publisher, notifying Subscribers (such as different user interfaces for various roles) about any alterations. Each Subscriber would express interest in receiving order updates. When order changes occur, the Publisher alerts relevant Subscribers, ensuring specific interfaces display the order details tailored to users' roles. For instance, after successful authentication and role identification, the system guides users to role-specific interfaces featuring the "Order Status" option. Upon selecting this option, the Publisher would notify respective Subscribers, prompting the display of order details aligned with the users' roles. By adopting this pattern, the system maintains flexible interactions, allowing smooth updates to be generated across different interfaces without messing with the core order status logic within each interface.

UC-12: Meal Prep



The diagram demonstrates the interactions in UC-12: Meal Prep. Here's a breakdown of the described interactions:

- The user accesses the site and initiates the login process
- The system will check whether the user is in the database by verifying their login credentials (userID).
- If the provided credentials match those in the database, the user is verified.
- Based on the user's role(host, waiter, manager, chef, guest), the system will direct the user to the specific interface tailored to that role.
- Lastly, the system will display a "View Orders" option that the users can click to see all incoming orders along with a timestamp (the orders will be sorted based on the time they were sent to the kitchen, making it easier for chefs to see which orders need to be made first).

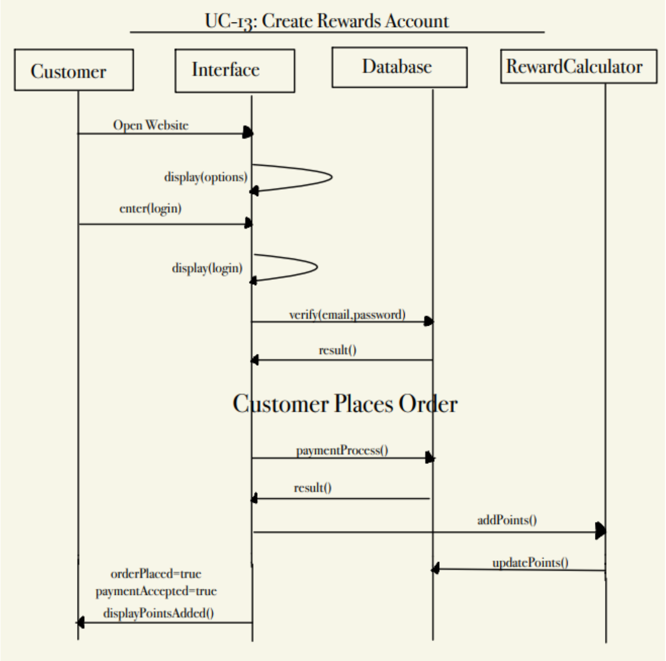
The design principles employed in this sequence are the High Cohesion Principle and the Expert Doer Principle.

- The High Cohesion Principle suggests keeping related functionality grouped together within a module, class, or component.
    - The principle is employed in the "User Authentication" component which consists of handling all tasks related to user verification and validation.
    - The principle is employed in the "Role-Based Interface Handling" function which is responsible for directing the user to the appropriate interface based on their role.
- The Expert Does Principle suggests responsibility is assigned to the class that has the necessary information to fulfill it effectively.
    - The principle is displayed in the "User Authentication" component which will possess the necessary information and methods to authenticate the user effectively against the database.
    - The principle is employed in the "Role-Based Interface Handling" function which will utilize its knowledge to handle this responsibility effectively.
    - The principle is employed when the user clicks on the "View Orders" option which will determine the appropriate action and display all incoming orders sent to the kitchen along with a timestamp.

In this diagram, the Publisher-Subscriber pattern could optimize interactions related to the "View Orders" scenario. The system's component managing incoming orders and their timestamps would function as the Publisher. It would be responsible for updating Subscribers, which could include various interfaces or components interested in real-time order updates. When new orders are placed or modified, the Publisher would notify the relevant Subscribers, prompting role-specific interfaces (such as the kitchen interface for chefs) to display the updated orders list, sorted by timestamps. After successful user authentication and role identification, users would be directed to interfaces pertinent to their roles, featuring the "View Orders" option. Selecting this option triggers the

Publisher to notify Subscribers, ensuring timely display of incoming orders with timestamps. Employing this pattern allows for seamless updates across different interfaces without intricately embedding order-handling logic within each interface, thereby promoting flexibility and scalability in system functionality.

UC-13 Create Rewards Account



The diagram above demonstrates the interactions in UC-11: Create Rewards Account. Here's a breakdown of the described interactions:

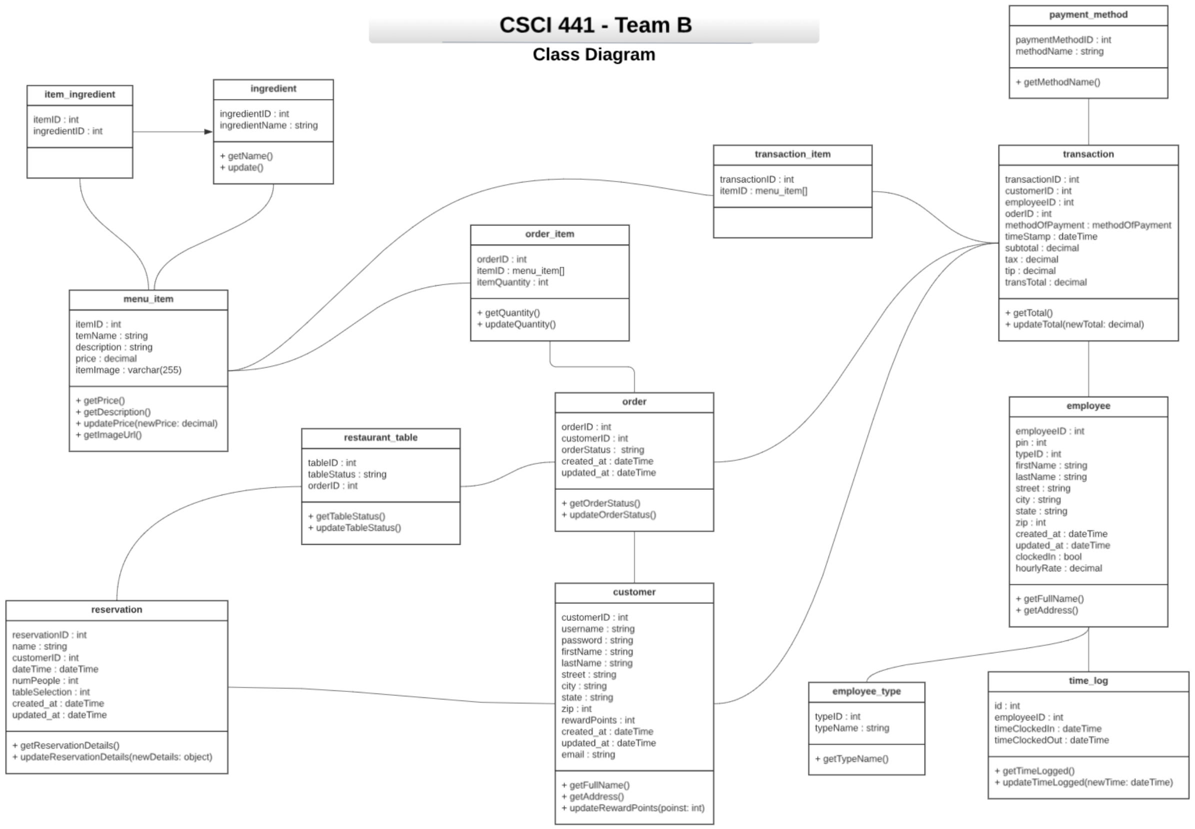
- Initially the customer will open the website and proceed to log in. The interface will verify their account credentials by checking with the database and confirm their account existence.
- After successfully login, the interface will present a menu and enable the customer to place an order. Once the order is placed, the customer will be guided to the payment process.
- Upon successful payment, the interface will transmit the transaction details to the reward calculator. The calculator will determine the points to be awarded based on the total order amount, a value defined by the restaurant's owner or manager.
- Lastly, the calculator will update the customer's balance in the database. The interface will then display the new balance to the customer, along with a receipt confirming the successful completion of the transaction.

The design principles present in this sequence is the High Cohesion Principle, Expert Doer, and Separation of Concerns.

- The High Cohesion Principle is employed in the interface, primarily dealing with user interaction, the calculator with points calculation, and the database with the management of accounts.
- The Expert Doer is present in the design with the reward calculator, calculating points based on the order total, handles this responsibility effectively, and possesses the necessary information to perform the task efficiently.
- The Separation of Concerns suggests the separation of functionalities into distinct components. The interface focuses on user interaction, the reward calculator on points calculation, and the database on account information.
- In the diagram of "Create Rewards Account," the Publisher-Subscriber pattern can optimize interactions between distinct system components. The interface handling order placement, payment processing, and displaying transaction details would serve as the Publisher. It manages state changes related to successful transactions and notifies Subscribers, like the reward calculator, about these updates. Upon successful payment, the Publisher notifies the reward calculator Subscriber, which computes points based on order totals defined by restaurant parameters. Subsequently, the calculator updates the customer's balance in the database. Once the balance is updated, the Publisher notifies Subscribers to display the new balance and transaction confirmation to the customer. This pattern ensures efficient communication between components without tightly coupling functionalities, aligning with design principles that advocate for modular and flexible systems.

9. Class Diagram and Interface Specification(Updated)

a. Class Diagrams



b. Data Types and Operation Signatures

Class: Item_Ingredients

a. Attributes:
   i. ingredientName: String - The name of the ingredient.
   ii. quantity: double - The quantity of the ingredient.
   iii. unit: String - The unit of measurement for the quantity.

b. Operations: None

c. Meaning: Represents an ingredient and its quantity used in a menu item.

Item_Ingredients
#ingredientName: String #quantity: Double #unit: String

Class: Ingredients

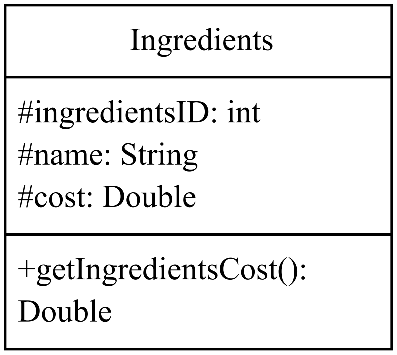
d. Attributes:

i. ingredientID: int - A unique identifier for the ingredient.
ii. name: String - The name of the ingredient.
iii. cost: double - The cost of the ingredient.

e. Operations:

i. getIngredientCost():Double - Calculates the cost of the ingredient.

f. Meaning: Represents individual ingredients used in menu items. The **getIngredientCost()** operation calculates the cost of the ingredient.



Ingredients

#ingredientsID: int
#name: String
#cost: Double

+getIngredientsCost(): Double

Class: Menu_Item

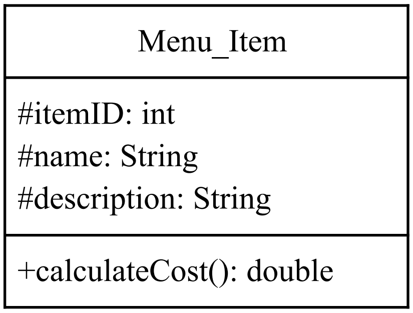
g. Attributes:

i. itemID: int - A unique identifier for the menu item.
ii. name: String - The name of the menu item.
iii. description: String - A description of the menu item.
iv. price: double - The price of the menu item.

h. Operations:

i. calculateCost(): double - Calculates the total cost of the menu item.

i. Meaning: Represents a menu item with a name, description, and price. The **calculateCost()** operation calculates the total cost of the menu item.



Menu_Item

#itemID: int
#name: String
#description: String

+calculateCost(): double

Class: Order_Item

j. Attributes:

i. orderID: int - A unique identifier for the order item.
ii. menuID: MenuItem - Reference to the menu item.
iii. ItemQuantity: int - The quantity of the menu item ordered.

k. Operations:

i. calculateTotalCost(): double - Calculates the total cost of the order item.

l. Meaning: Represents an item that is part of an order, with a reference to the menu item and the quantity ordered. The **calculateTotalCost()** operation calculates the total cost of the order item.

Order_Item
#orderID:int #menuItem: MenuItem #quantity: int
+calculateTotalCost(): double

Class: TransactionItem

m. Attributes:

i. transactionItemID: int - A unique identifier for the transaction item.

ii. orderItem: OrderItem - Reference to the ordered item.

n. Operations:

i. calculateItemCost(): double - Calculates the cost of the transaction item.

o. Meaning: Represents an item within a transaction, with a reference to the order item. The **calculateItemCost()** operation calculates the cost of the transaction item.

Transaction_Item
#transactionItemID: int #orderItem: OrderItem
+calculateItemCost(): double

Class: Payment_Method

p. Attributes:

i. methodID: int - A unique identifier for the payment method.

ii. name: String - The name of the payment method.

q. Operations: None

r. Meaning: Represents different methods of payment, such as cash, credit card, or mobile wallet.

Payment_Method
#methodID: int #name: String

Class: Transaction

s. Attributes:

i. transactionID: int - A unique identifier for the transaction.

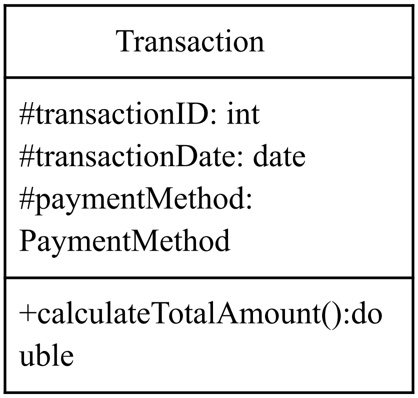
ii. transactionDate: Date - The date of the transaction.

iii. paymentMethod: PaymentMethod - Reference to the payment method used.

t. Operations:

i. calculateTotalAmount(): double - Calculates the total transaction amount.

u. Meaning: Represents a transaction made by a customer, including the date, payment method, and the total amount paid. The **calculateTotalAmount()** operation calculates the total transaction amount.



Transaction

#transactionID: int
#transactionDate: date
#paymentMethod: PaymentMethod

+calculateTotalAmount():double

Class: RestaurantTable

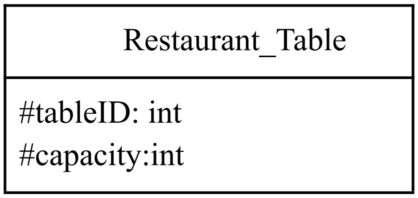
v. Attributes:

i. tableID: int - A unique identifier for the table.

ii. capacity: int - The seating capacity of the table.

w. Operations: None

x. Meaning: Represents a table in the restaurant, with a unique ID and a specified seating capacity.



Restaurant_Table

#tableID: int
#capacity:int

Class: Order

y. Attributes:

i. orderID: int - A unique identifier for the order.

ii. table: RestaurantTable - Reference to the table where the order is placed.

iii. orderItems: List<OrderItem> - List of ordered items.

z. Operations:

i. calculateTotalOrderCost(): double - Calculates the total cost of the order.

aa. Meaning: Represents an order placed by a customer at a specific restaurant table. It includes a reference to the table and a list of ordered items. The **calculateTotalOrderCost()** operation calculates the total cost of the order.

Order
#orderID:int #table: restaurant table #orderItems: List<orderItem>
+calculateTotalOrderCost(): double

Class: Customer

- Attributes:
  i. customerID: int - A unique identifier for the employee.
  i. UserName: int - the UserName of the customer.
  ii. firstName: string- first name of the customer
  iii. lastName: String - last name of the customer.
  iv. Street: string - the street address of the customer.
  v. City: string - city where the customer resides.
  vi. State: string - state where the customer resides
  vii. Zip: int - zip code where the customer resides.
  viii. rewardPoints: int -the customer points earned.
  ix. Created_at: dateTime - time when customer account was created.
  x. Update_at: dateTime - time when customer account was last updated.
  xi. Email: string - email used by customer to log in.

- Operations: None
- Meaning: Represents a customer who visits the restaurant, with a unique customer ID, name, and contact information. This class can be used to store and manage customer information for reservations and orders.

customer
#customerID: int #name: String #firstName: string #lastName: String.

#Street: string
#City: string
#State: string# Zip: int
#rewardPoints: int
#Created_at: dateTime
#Update_at: dateTime
#email : string

Class: Employee

bb. Attributes:

i. employeeID: int - A unique identifier for the employee.
ii. Pin: int - the code pin the employee has to enter.
iii. firstName: string- first name of the employee
iv. lastName: String - last name of the employee.
v. Street: string - the street address of the employee
vi. City: string - city where the employee resides.
vii. State: string - state where the employee resides
viii. Zip: int - zip code.
ix. clockedIn: bool - Boolean if employee is clocked in or not.
x. hourlyRate: decimal - employees' hourly rate.
xi. Created_at: dateTime - time when employee account was created.
xii. Update_at: dateTime - time when employee account was last updated

cc. Operations: None

2.

a. Meaning: Represents an employee working at the restaurant, with a unique ID, name, and an associated employee type.

Employee
#employeeID: int #pin: int #name: String #firstName: string #lastName: String .#Street: string #City: string #State: string #Zip: int #clockedIn : bool #hourlyRate: decimal

#Created_at: dateTime #Update_at: dateTime

Class: ReservationCustomer

b. Attributes:
   i. customerID: int - A unique identifier for the customer.
   ii. name: String - The name of the customer.
   iii. contactInfo: String - Contact information for the customer.

c. Operations: None

d. Meaning: Represents a customer who made a reservation, with a unique ID, name, and contact information.

Reservation
#customerID: int #name: String #contactInfo: String

Class: Employee_Type

e. Attributes:
   i. typeID: int - A unique identifier for the employee type.
   ii. typeName: String - The name of the employee type.

f. Operations: None

g. Meaning: Represents different types of employees in the restaurant, such as chef, waiter, or manager.

employee_type
#typeID: int - A unique identifier for the employee type. #typeName: String - The name of the employee type.

Class: Time_Log

h. Attributes:
   i. logID: int - A unique identifier for the time log entry.
   ii. employee: Employee - Reference to the employee.
   iii. timeClockedIn: DateTime - Timestamp for employee clock in
   iv. timeClockedOut: DateTime - Timestamp for employee clock out

i. Operations: None
j. Meaning: Represents a time log entry for an employee, recording their activities and the timestamp when the activity occurred.

Time_Log
#logID: int #employee: Employee #timeClockedIn: DateTime #timeClockedOut: DateTime

c. Traceability Matrix

Classes	**Domain Concepts**											
	Customer Account	Interface	Payment System	Order Status	Order Queue	Controller	Communicator	Floor plan	Employee Account	Table Status	Reward System	DB Connection
customer	X										X	X
Menu_item		X										X
ingredient		X										
item_ingredient		X										
order	X			X	X		X					X
order_item		X										
transaction	X		X						X			X
transaction_item		X										

Classes	**Domain Concepts**											
payment_method	X	X										
employee									X			X
employee_type		X										
time_log									X			X
restaurant_table		X				X		X		X		
reservation	X	X				X				X		X

Customer Account:

- Customer: Allow customers to create accounts and add personal info
- Order: Add order to customer account.
- Transaction: Allow customer to pay for order
- Payment method: Allow customer to choose payment method
- Reservation: Allow customer to make reservation

Payment System:

- Transaction: Allow payment during transaction

Order Status:

- Order: Allow customers and workers to view and update order status

Order Queue:

- Order: Allow chef to keep track of orders

Controller:

- Restaurant_table: Allow employees to view and change table status
- Reservation: Allow employees to make reservations for customers

Communicator:

- Order: store each order in database

Floorplan:

- Restaurant_table: Allow manager to view all tables in order to change display of tables

Employee Account:

- Employee: Allow employees to their personal info to the system
- Transaction: Keep track of which employee made which transaction
- Time_log: Allow workers to clock in and clock out

Table Status:

- Restaurant_table: Display table status
- Reservations: Allow interface to show which tables are available for reservations

Reward System:

- Customer: Add reward to customers account

DB Connection:

- Customer: Add or fetch customer info from database
- Menu_item: Retrieve or modify menu items in database
- Order: Store each order in the database to keep record
- Transaction: Store each payment made in the database for records
- Employee: Add or fetch employee info form database
- Time_log: Store the exact time each employee clocked in and clocked out
- Reservation: Store every present and future reservation made

d. Design Patterns

We used the broker pattern, this pattern is particularly well-suited for systems where components need to interact through remote method calls, making it ideal for web applications that involve client-server communication and database interactions.

The Broker pattern facilitates efficient communication between different components of the system, the front-end interface, the server-side logic, and the database. It essentially acts as a middleman, coordinating the interactions between these components. This is particularly useful in a web application like the one we are developing, where requests from the web interface need to be processed by the server before any interaction with the database can occur. By employing this pattern, the application will be able to handle client requests more efficiently, as the broker will manage these interactions, ensuring that requests are routed to the appropriate components and responses are returned swiftly to the user.

The Broker pattern supports a modular design, which aligns well with the collaborative nature of the project. Each team member can work on different modules (front-end, server-side, database) independently, and the Broker pattern ensures these modules interact seamlessly. This approach not only improves the development process but also simplifies maintenance and future updates to the website. Each module can be updated or replaced without affecting the others, as long as it adheres to the established communication protocols set by the Broker. This flexibility is key in web development, where technologies and user requirements are constantly evolving.

e. Object Constraint Language (OCL)

MainScreen:

Context: MainScreen::clickLogin
Invariant: privatePin, userName

Pre-conditional: findViewById(R.id.userName)
Pre-conditional: Intent intent = new Intent(MainScreen.this, restaurantTable.class)
Post-condition: MainScreen.user.startActivity()

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

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

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

LoginActivity:
Context: LoginActivity::pinChecked:boolean
Invariant: auth, loginButton, privateKey
Pre-conditional: pin = pinInt.getInt()
Post-conditional: return isValid

SignupActivity:
Context: SignupActivity::signup
Invariant: progressDialog
Pre-conditional: firstName = firstnameText.getText().toString()
Pre-conditional: lastname = lastnameText.getText().toString()
Pre-conditional: pin = pinText.getText().toString()
Pre-conditional: street = streetText.getText().toString()
Pre-conditional: city = cityText.getText().toString()
Pre-conditional: state = stateText.getText().toString()
Pre-conditional: zip = zipText.getInt()
Pre-conditional: accountType = accountTypeText.getText().toString()
Post-conditional: onSignUpSuccess()

Post-conditional: onSignUpFailed()

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

Context: SignupActivity::onSignUpFailed
Invariant: signupButton
Pre-conditional: If sign up failed
Post-conditional: Alert.alert('Sign up failed.')

Context: SignupActivity::checked:boolean
Invariant: firstname, lastname, pin, street, city, state, zip, accountType
Pre-conditional: firstname.isEmpty()
Pre-conditional: lastname.isEmpty()
Pre-conditional: pin.isEmpty() || !Patterns.PIN.matcher(pin).matches()
Pre-conditional: street.isEmpty()
Pre-conditional: city.isEmpty()
Pre-conditional: state.isEmpty()
Pre-conditional: zip.isEmpty()
Pre-conditional: accountType.isEmpty() || accountType != 1 || accountType != 2 || accountType !=3
Post-conditional: return isValid

10. Algorithms and Data Structures

a. Algorithms

Regarding the work data of employees, the system employs algorithms to calculate work hours. The system tracks the timestep when an employee starts and ends their shift. In the code, there is a method that simplifies the calculation of hours worked. This method calculates hours worked by deducting the clock-in time (timeClockedIn) from the clock-out time(timeOut) to determine the hours worked during an employee's shift.

For payroll computations, particularly if a manager needs to calculate the salaries of employees, we will implement a Salary() method. This method will calculate the annual salary of a given worker based on their hourly (wage) and the number of hours worked (hoursWorked). It will provide a running total of the worker's yearly salary. To get the daily salaries, the worker's salary is calculated by multiplying hoursWorked by the wage, and this daily salary is then added to the

expected pay(expectedPay). This calculation will be repeated for every working period throughout the year.
When a customer initiates a transaction, the system calculates the subtotal by adding the prices of all items in the order. Then, the tax rate is applied to calculate the grand total. After presenting this information to the user, the system gives the customer the option to enter a tip, which is then added to determine the final amount due for the transaction. For payment processing, we will utilize an API(to be determined). All order-related details will be employed in calling the API.
After a successful transaction, our rewards system will come up with a certain amount of points based on how much the customer spent on their order, the amount of points will be determined by the manager.
On the menu side of our website, customers have the flexibility to view the menu. Once customers have made their selections, they will be able to add or remove food items from their order according to their preferences or dietary restrictions.

b. Data Structures

For the employee interface, we will employ a queue data structure which is a structured collection of items where new items are added at one end and existing items are removed from the other end. This queue employs the principle of “first in first out” using the ordering system. Within the employee interface, the queue is utilized to manage orders received from the system. When an order comes in, the chef is promptly notified, prepares the meal, and serves it, consequently removing it from the queue. The performance of accessing an item is O(n) and removal is O(1), making it highly efficient for this specific scenario.

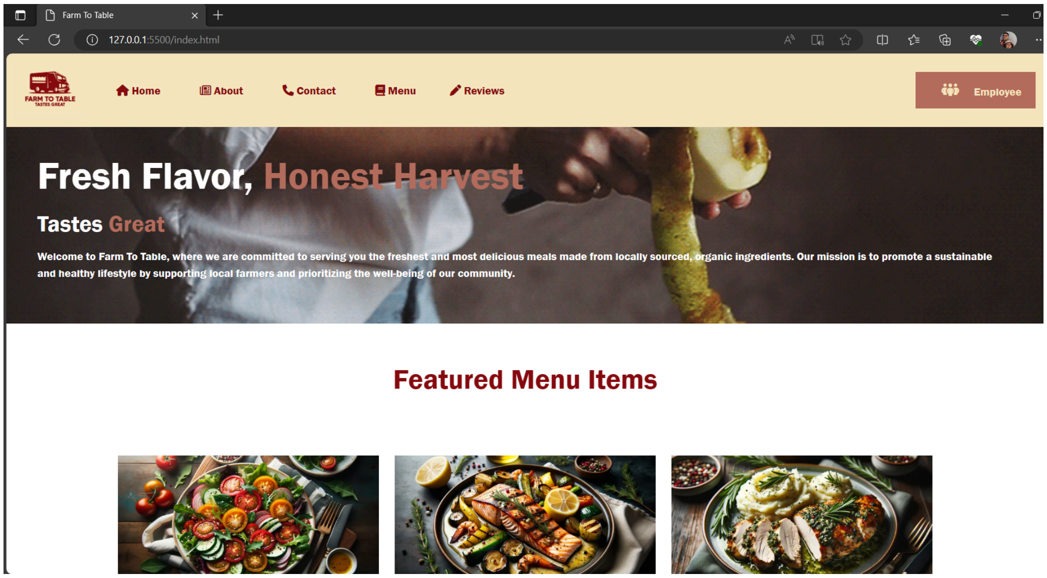
In our SQL database, we’ve established an “items” table. Databases are highly efficient for querying and selecting items based on attributes. The table’s primary key will be ‘itemID’, and an additional column will include name, price, and ingredients. To enhance efficiently, it would be beneficial to incorporate common allergies and restrictions columns to indicate if a meal contains nuts for example.

To monitor employee work data, such as hours worked and current status, we will also utilize a database. An “employee” table in the database will have a primary key, the employee’s userID, allowing easy access to an entry related to that employee. The column will track the employee’s clock-in time, clock-out time, clocking stats, total hours worked, wage, and an estimated pay for the upcoming pay cycle.

When a customer is putting an order together, the system will generate an order object that will include a collection of items within the order, and associated details like the table number, customer, and server involved. The payment method will also be represented by an object, which will contain fields specifying the payment type (card or cash), associating it with a user, and including information about the payment method (card number, expiration date, billing address, etc). After a successful payment, this action will update the user’s reward information (only if the user is a reward’s member). All of these objects will be stored as tables within our relational SQL database.

11. User Interface Design and Implementation(updated)

Figure 1-5: Customer Interface

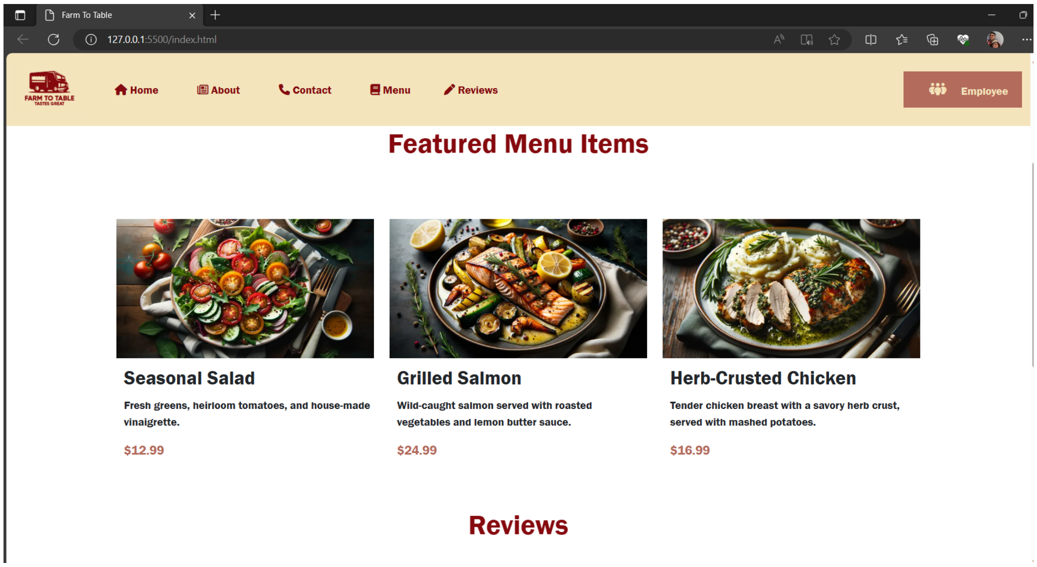


Fresh Flavor, Honest Harvest

Tastes Great

Welcome to Farm To Table, where we are committed to serving you the freshest and most delicious meals made from locally sourced, organic ingredients. Our mission is to promote a sustainable and healthy lifestyle by supporting local farmers and prioritizing the well-being of our community.

Featured Menu Items



Featured Menu Items

Seasonal Salad

Fresh greens, heirloom tomatoes, and house-made vinaigrette.

$12.99

Grilled Salmon

Wild-caught salmon served with roasted vegetables and lemon butter sauce.

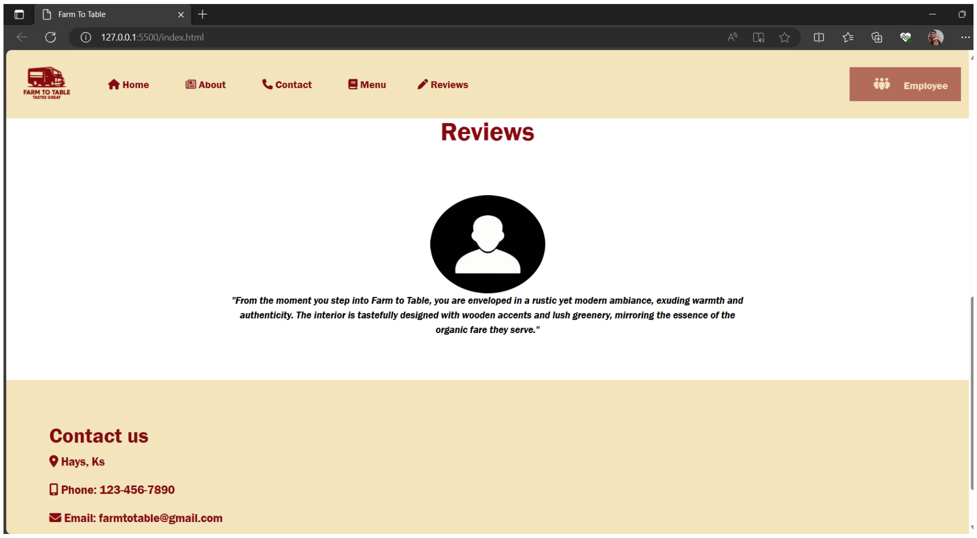
$24.99

Herb-Crusted Chicken

Tender chicken breast with a savory herb crust, served with mashed potatoes.

$16.99

Reviews



Reviews

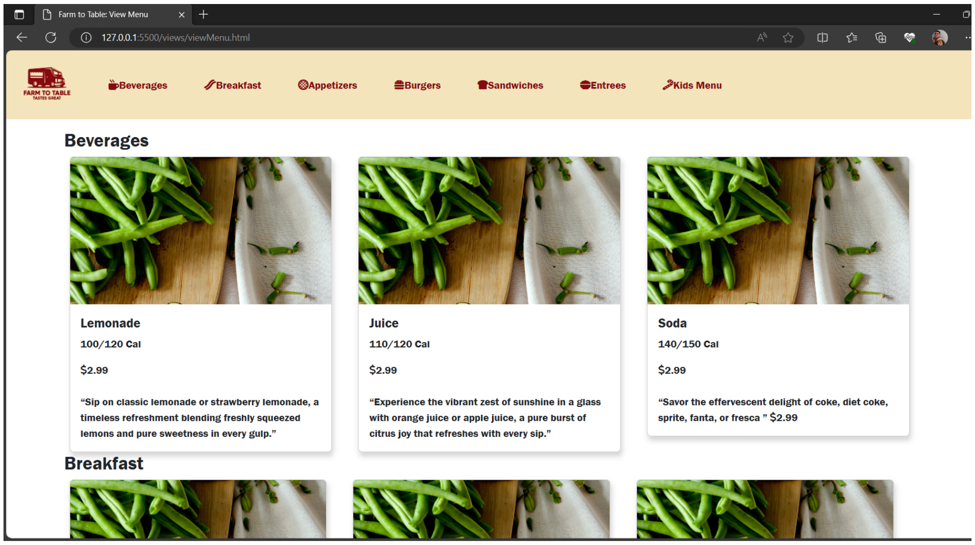
"From the moment you step into Farm to Table, you are enveloped in a rustic yet modern ambiance, exuding warmth and authenticity. The interior is tastefully designed with wooden accents and lush greenery, mirroring the essence of the organic fare they serve."

Contact us

Hays, Ks

Phone: 123-456-7890

Email: farmtotable@gmail.com



Beverages

Lemonade

100/120 Cal

$2.99

"Sip on classic lemonade or strawberry lemonade, a timeless refreshment blending freshly squeezed lemons and pure sweetness in every gulp."

Juice

110/120 Cal

$2.99

"Experience the vibrant zest of sunshine in a glass with orange juice or apple juice, a pure burst of citrus joy that refreshes with every sip."

Soda

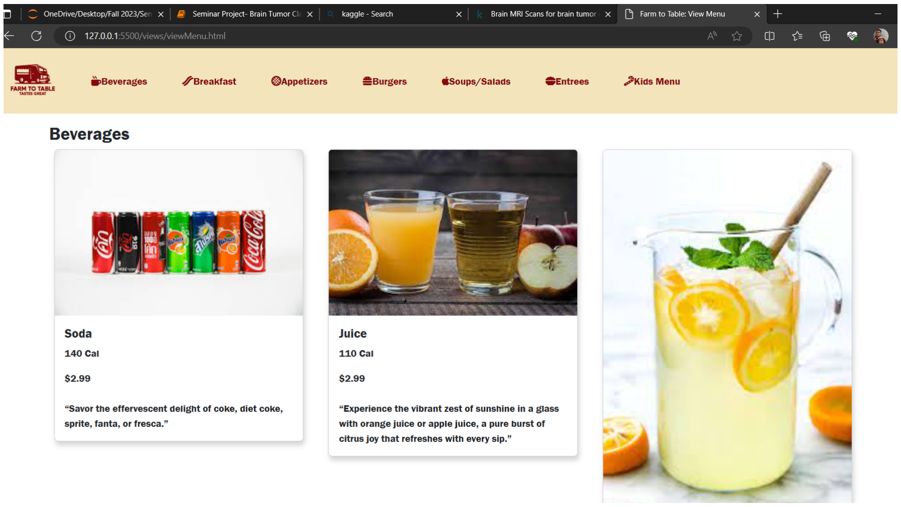
140/150 Cal

$2.99

"Savor the effervescent delight of coke, diet coke, sprite, fanta, or fresca " $2.99

Breakfast

Once customers launch the website, they are greeted with the home page. From here, customers can look over the menu items by clicking on the "Menu" option.


OneDrive/Desktop/Fall 2023/Sen
Seminar Project- Brain Tumor Cl
kaggle - Search
Brain MRI Scans for brain tumor
Farm to Table: View Menu
127.0.0.1:5500/views/viewMenu.html
FARM TO TABLE
Beverages
Breakfast
Appetizers
Burgers
Soups/Salads
Entrees
Kids Menu

Beverages

Soda

140 Cal

$2.99

"Savor the effervescent delight of coke, diet coke, sprite, fanta, or fresca."

Juice

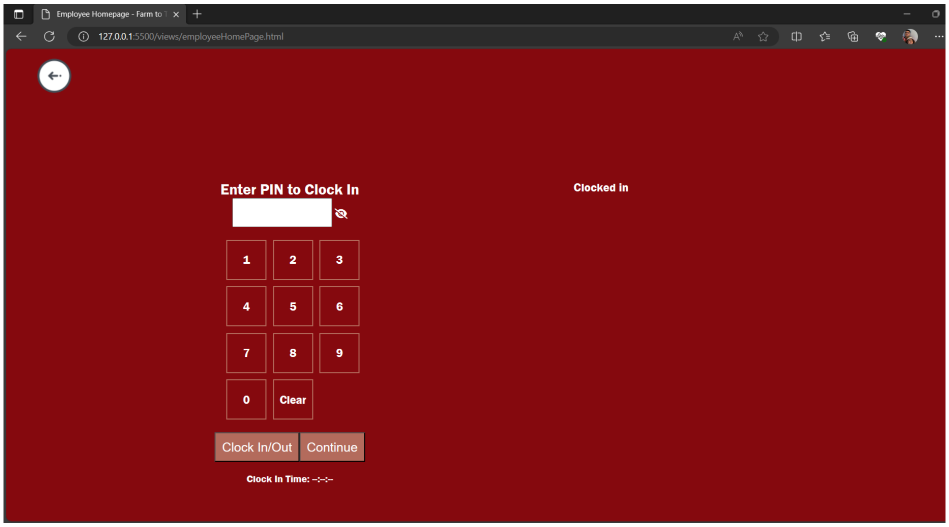
110 Cal

$2.99

"Experience the vibrant zest of sunshine in a glass with orange juice or apple juice, a pure burst of citrus joy that refreshes with every sip."

An employee on the other hand can also use the customer homepage to access the Employee interface that we already demonstrated in earlier examples by clicking on the "Employee" button.

Figures 6-14: Employee Interface



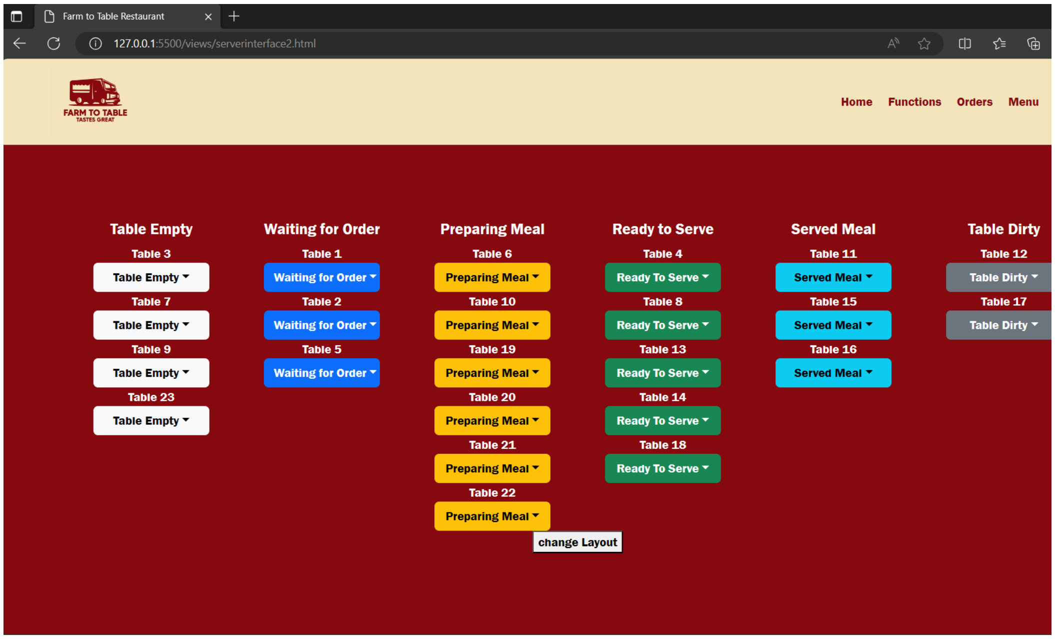
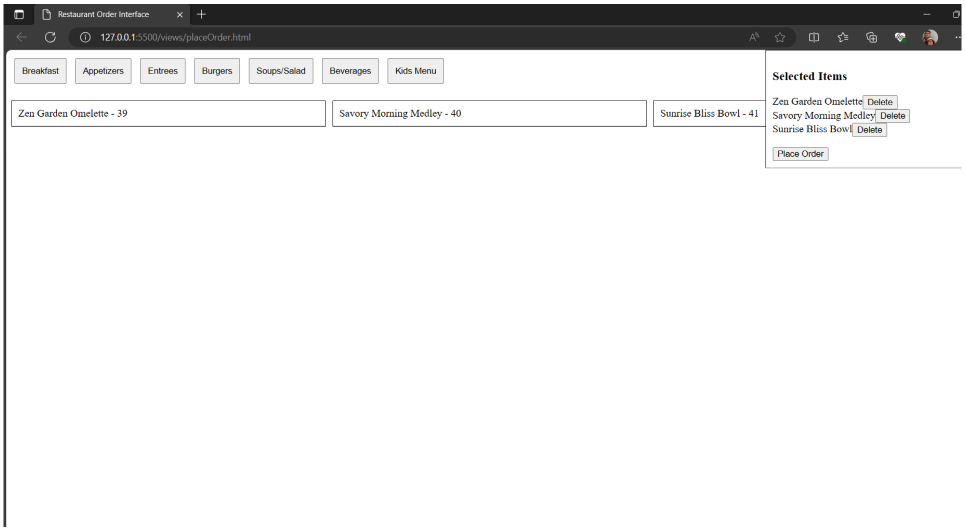
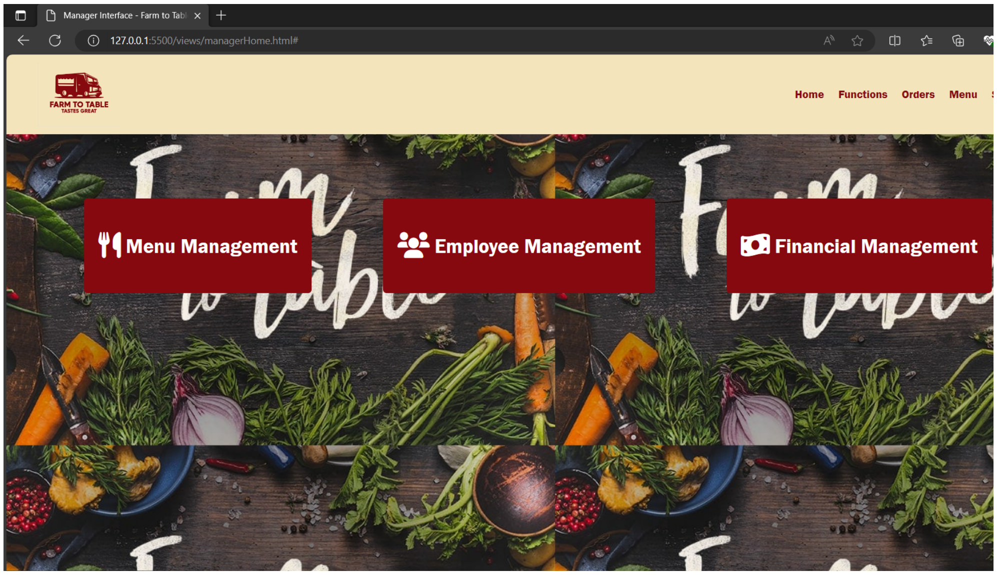


Table Empty	Waiting for Order	Preparing Meal	Ready to Serve	Served Meal	Table Dirty
Table 3 Table Empty	Table 1 Waiting for Order	Table 6 Preparing Meal	Table 4 Ready To Serve	Table 11 Served Meal	Table 12 Table Dirty
Table 7 Table Empty	Table 2 Waiting for Order	Table 10 Preparing Meal	Table 8 Ready To Serve	Table 15 Served Meal	Table 17 Table Dirty
Table 9 Table Empty	Table 5 Waiting for Order	Table 19 Preparing Meal	Table 13 Ready To Serve	Table 16 Served Meal	
Table 23 Table Empty		Table 20 Preparing Meal	Table 14 Ready To Serve		
		Table 21 Preparing Meal	Table 18 Ready To Serve		
		Table 22 Preparing Meal			

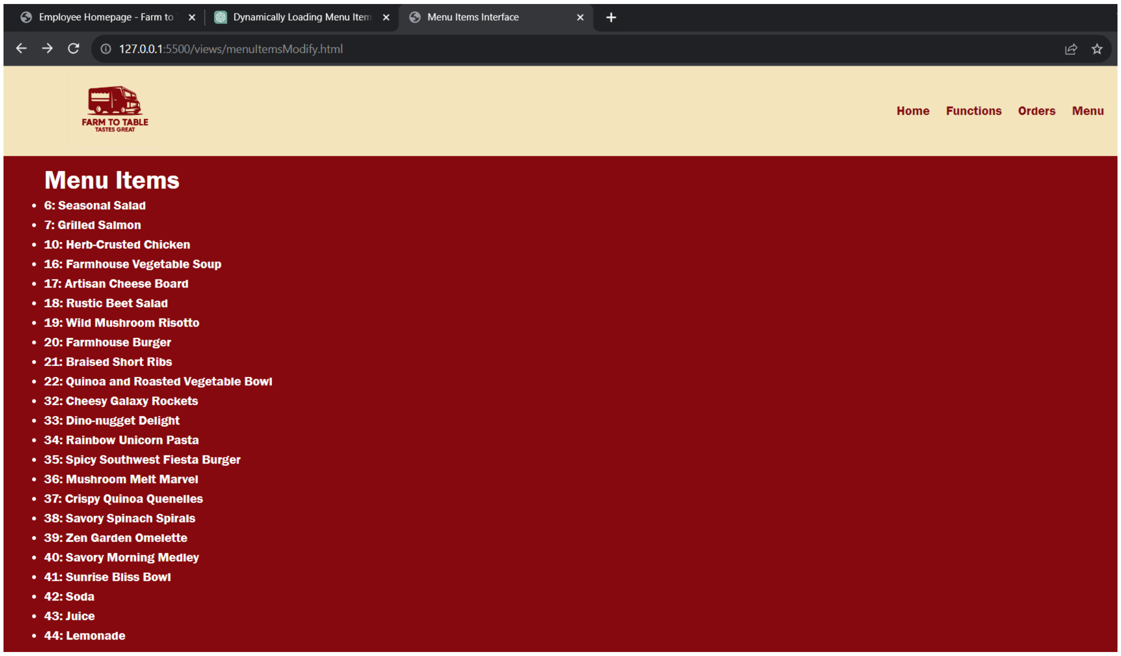
After signing in through the employee homepage, servers and managers can see all the available tables within the restaurant. The tables would be categorized based on their current status. Servers can place orders by clicking on the "Waiting for Order" tables. After clicking on a table, they would be prompted with 2 options. First option is "Dine In", that is for putting in orders for customers dining inside the restaurant. Second option is "To Go", which is for putting in orders for customers wanting food to go. The following page will display which will be used to place orders (It is still a work in progress and will be ready for Demo #2).


Restaurant Order Interface
127.0.0.1:5500/views/placeOrder.html
Breakfast
Appetizers
Entrees
Burgers
Soups/Salad
Beverages
Kids Menu
Zen Garden Omelette - 39
Savory Morning Medley - 40
Sunrise Bliss Bowl - 41
Selected Items
Zen Garden Omelette Delete
Savory Morning Medley Delete
Sunrise Bliss Bowl Delete
Place Order

Also, servers and managers would be able to access the menu by clicking on “Menu” located on the top of the page. If they click on the “functions” button that would display the following page:



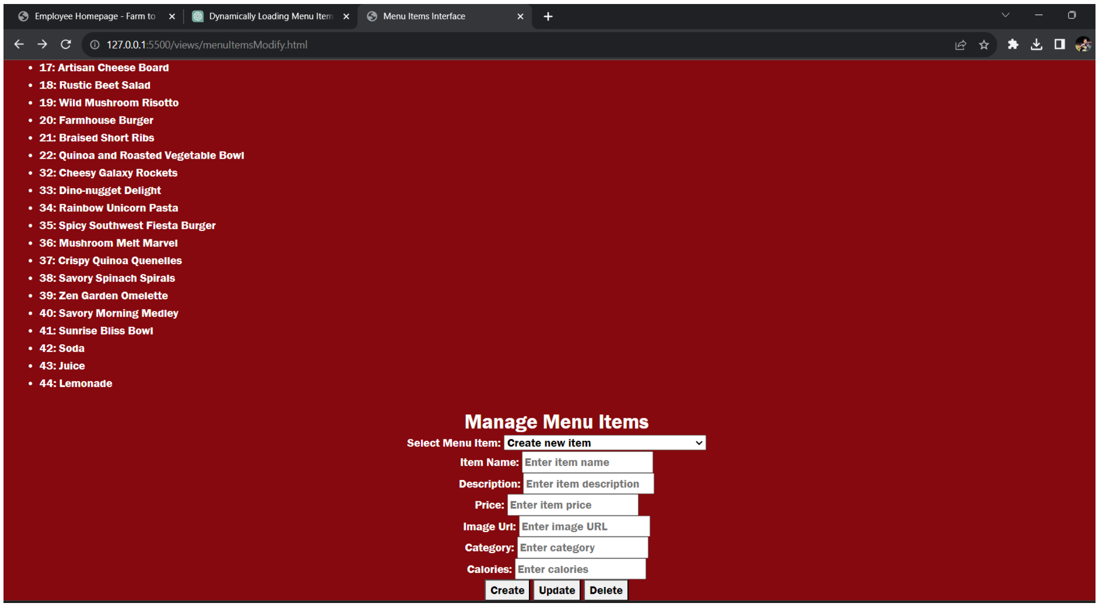
In this page, managers would be able make changes to the menu, manage employees, and print out financial reports. Clicking on the “Menu Management option displays the following page:



Employee Homepage - Farm to
Dynamically Loading Menu Item
Menu Items Interface
127.0.0.1:5500/views/menuItemsModify.html
FARM TO TABLE
TASTES GREAT
Home
Functions
Orders
Menu

Menu Items

- 6: Seasonal Salad
- 7: Grilled Salmon
- 10: Herb-Crusted Chicken
- 16: Farmhouse Vegetable Soup
- 17: Artisan Cheese Board
- 18: Rustic Beet Salad
- 19: Wild Mushroom Risotto
- 20: Farmhouse Burger
- 21: Braised Short Ribs
- 22: Quinoa and Roasted Vegetable Bowl
- 32: Cheesy Galaxy Rockets
- 33: Dino-nugget Delight
- 34: Rainbow Unicorn Pasta
- 35: Spicy Southwest Fiesta Burger
- 36: Mushroom Melt Marvel
- 37: Crispy Quinoa Quenelles
- 38: Savory Spinach Spirals
- 39: Zen Garden Omelette
- 40: Savory Morning Medley
- 41: Sunrise Bliss Bowl
- 42: Soda
- 43: Juice
- 44: Lemonade



Employee Homepage - Farm to
Dynamically Loading Menu Item
Menu Items Interface
127.0.0.1:5500/views/menuItemsModify.html

- 17: Artisan Cheese Board
- 18: Rustic Beet Salad
- 19: Wild Mushroom Risotto
- 20: Farmhouse Burger
- 21: Braised Short Ribs
- 22: Quinoa and Roasted Vegetable Bowl
- 32: Cheesy Galaxy Rockets
- 33: Dino-nugget Delight
- 34: Rainbow Unicorn Pasta
- 35: Spicy Southwest Fiesta Burger
- 36: Mushroom Melt Marvel
- 37: Crispy Quinoa Quenelles
- 38: Savory Spinach Spirals
- 39: Zen Garden Omelette
- 40: Savory Morning Medley
- 41: Sunrise Bliss Bowl
- 42: Soda
- 43: Juice
- 44: Lemonade

Manage Menu Items

Select Menu Item: Create new item

Item Name: Enter item name

Description: Enter item description

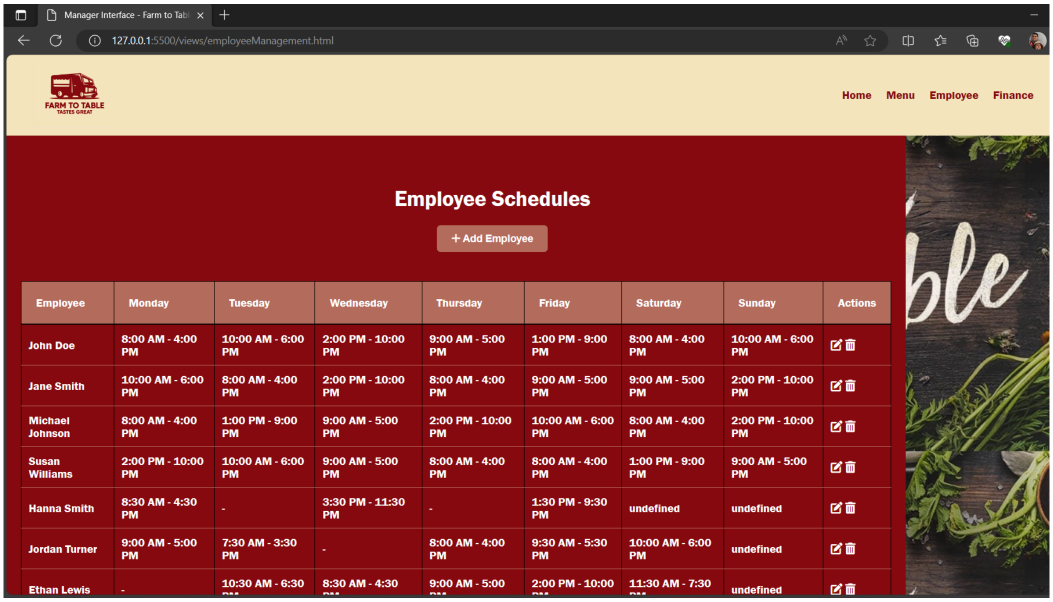
Price: Enter item price

Image Url: Enter image URL

Category: Enter category

Calories: Enter calories

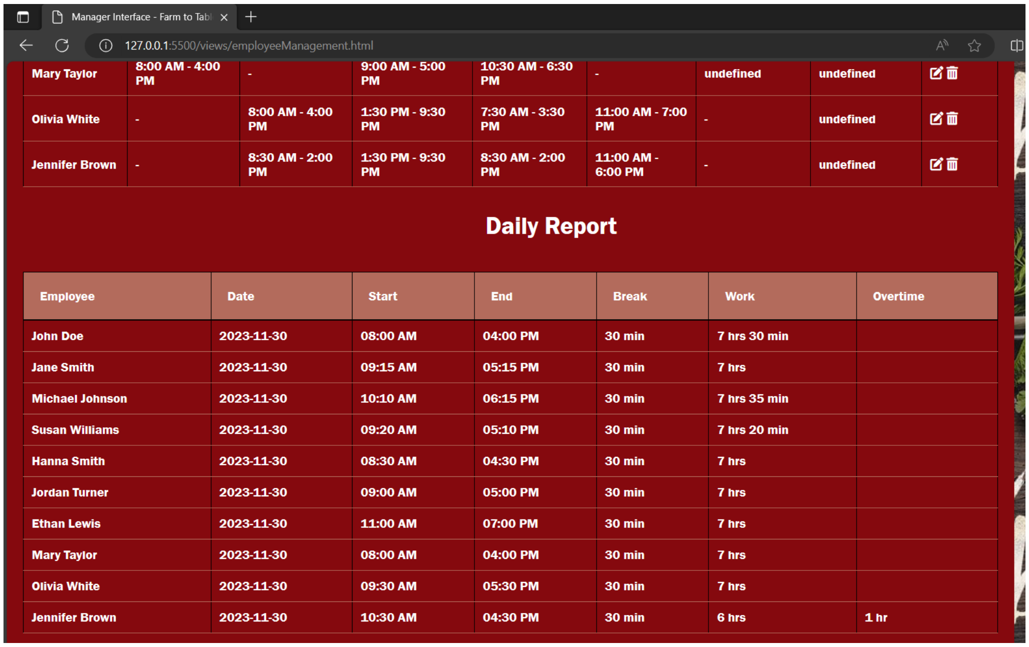
In this page, managers are able to add or remove menu items. Adding an item simply requests that they fill in the required information. Updating an item allows the manager to select an item from the menu and update all of the properties manually. Deleting an item requests the item name from the drop down menu and deletes the item from the menu. Clicking on the employee management displays the following pages:


Manager Interface - Farm to Tabl
127.0.0.1:5500/views/employeeManagement.html
FARM TO TABLE
TASTES GREAT
Home
Menu
Employee
Finance

Employee Schedules

+ Add Employee

Employee	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Actions
John Doe	8:00 AM - 4:00 PM	10:00 AM - 6:00 PM	2:00 PM - 10:00 PM	9:00 AM - 5:00 PM	1:00 PM - 9:00 PM	8:00 AM - 4:00 PM	10:00 AM - 6:00 PM	
Jane Smith	10:00 AM - 6:00 PM	8:00 AM - 4:00 PM	2:00 PM - 10:00 PM	8:00 AM - 4:00 PM	9:00 AM - 5:00 PM	9:00 AM - 5:00 PM	2:00 PM - 10:00 PM	
Michael Johnson	8:00 AM - 4:00 PM	1:00 PM - 9:00 PM	9:00 AM - 5:00 PM	2:00 PM - 10:00 PM	10:00 AM - 6:00 PM	8:00 AM - 4:00 PM	2:00 PM - 10:00 PM	
Susan Williams	2:00 PM - 10:00 PM	10:00 AM - 6:00 PM	9:00 AM - 5:00 PM	8:00 AM - 4:00 PM	8:00 AM - 4:00 PM	1:00 PM - 9:00 PM	9:00 AM - 5:00 PM	
Hanna Smith	8:30 AM - 4:30 PM	-	3:30 PM - 11:30 PM	-	1:30 PM - 9:30 PM	undefined	undefined	
Jordan Turner	9:00 AM - 5:00 PM	7:30 AM - 3:30 PM	-	8:00 AM - 4:00 PM	9:30 AM - 5:30 PM	10:00 AM - 6:00 PM	undefined	
Ethan Lewis	-	10:30 AM - 6:30	8:30 AM - 4:30	9:00 AM - 5:00	2:00 PM - 10:00	11:30 AM - 7:30	undefined	
Mary Taylor	8:00 AM - 4:00 PM	-	9:00 AM - 5:00 PM	10:30 AM - 6:30 PM	-	undefined	undefined	
Olivia White	-	8:00 AM - 4:00 PM	1:30 PM - 9:30 PM	7:30 AM - 3:30 PM	11:00 AM - 7:00 PM	-	undefined	
Jennifer Brown	-	8:30 AM - 2:00 PM	1:30 PM - 9:30 PM	8:30 AM - 2:00 PM	11:00 AM - 6:00 PM	-	undefined	

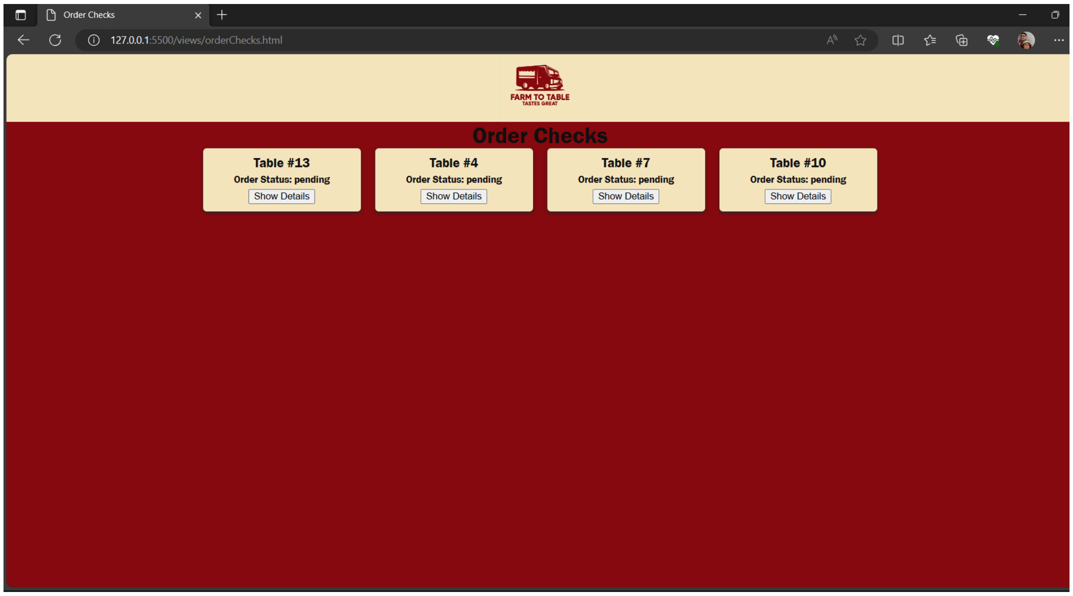

Manager Interface - Farm to Tabl
127.0.0.1:5500/views/employeeManagement.html

Daily Report

Employee	Date	Start	End	Break	Work	Overtime
John Doe	2023-11-30	08:00 AM	04:00 PM	30 min	7 hrs 30 min	
Jane Smith	2023-11-30	09:15 AM	05:15 PM	30 min	7 hrs	
Michael Johnson	2023-11-30	10:10 AM	06:15 PM	30 min	7 hrs 35 min	
Susan Williams	2023-11-30	09:20 AM	05:10 PM	30 min	7 hrs 20 min	
Hanna Smith	2023-11-30	08:30 AM	04:30 PM	30 min	7 hrs	
Jordan Turner	2023-11-30	09:00 AM	05:00 PM	30 min	7 hrs	
Ethan Lewis	2023-11-30	11:00 AM	07:00 PM	30 min	7 hrs	
Mary Taylor	2023-11-30	08:00 AM	04:00 PM	30 min	7 hrs	
Olivia White	2023-11-30	09:30 AM	05:30 PM	30 min	7 hrs	
Jennifer Brown	2023-11-30	10:30 AM	04:30 PM	30 min	6 hrs	1 hr

The pages display employee schedules and it keeps track of their shifts. In order to add an employee to the schedule, you simply click on the “Add Employee” button and fill in the required information.

Lastly, If they click on the “Orders” button that would display the following page:



Order Checks

Table #13
Order Status: pending

Table #4
Order Status: pending

Table #7
Order Status: pending

Table #10
Order Status: pending

This page serves as a hub for accessing all orders sent to the kitchen, presented in the chronological order they were dispatched. This facilitates cooks in determining the sequence of orders for preparation. Each order includes a timer, providing servers with insight into the time required for order preparation. Clicking on the “Show Details” button will also display order details which will allow servers to cash out their tables(this page is till a work in progress and will be ready by Demo #2).

12. Design of Tests(Updated)

a. Test Cases

Test Case Identifier: TC-1**
Use Case Tested: UC-2 Log in/ Log out
Pass/Fail Criteria: The test will pass if the user is able to successfully log in to the system. The test will fail if the user inputs an invalid username or password.
Input Data: username, password

Test Procedure:	**Expected Result:**
Step 1: User types in an invalid username and/or password.	Server denied the log-in attempt. Message pops up, instructing the user to try again using a valid log-in.
Step 2: User types in a valid username and password.	Server allows the log-in attempt. Depending on the user type, the system allows the user to access the specified interface.

Test Case Identifier: TC-2
Use Case Tested: UC-1: Clocking in/Clocking out
Pass/Fail Criteria: The test will pass if the user is able to successfully clock in and out for their shift and the server is able to verify their location. The test will fail if the user is unable to clock in or out, or the server is unable to verify the location.
Input Data: Selecting either the clock-in or clock-out button

Test Procedure:	**Expected Result:**
Step 1: User selects to clock-in or out outside of the restaurant.	Server allows the clock-in/clock-out attempt. The employee's current status updates to keep track of whether they are currently clocked in/out. Database records that a clock-in/out attempt has been made in a foreign location, along with the current time, user, location, and updated clocking status. System allows the user to access the specified interface.
Step 2: User selects to clock-in or out within the restaurant.	Server allows the clock-in/out attempt. The employee's current status updates to keep

	track of whether they are currently clocked in/out. Database records the clock-in/clock-out attempt, along with the current time, user, location, and updated clocking status. Depending on the employee type, the system allows the user to access the specified interface.

Test Case Identifier: TC-3* **Use Case Tested:** UC-13: Create a Rewards Account **Pass/Fail Criteria:** The test will pass if a user creates a new account. The test will fail if the username or email already exists. **Input Data:** username/email	
Test Procedure:	**Expected Result:**
Step 1: User enters username or email that already exists	Server denied the user from creating an account. Pops up message to enter different username or email
Step 2: User enters username or email that does not already exist	Server allows the user to create an account.

Test Case Identifier: TC-4* **Use Case Tested:** UC-8: Payment **Pass/Fail Criteria:** The test will pass if the user enters a valid payment method and the payment method has enough funds. The test will fail if payment method is not valid or funds are insufficient **Input Data:** payment method	
Test Procedure:	**Expected Result:**
Step 1: User enters invalid payment method or valid payment method with insufficient funds	System rejects the payment method and prompts the user to use a different payment method.
Step 2: User enters valid payment method	System accept payment

with enough funds	

Test Case Identifier: TC-5* **Use Case Tested:** UC-6: Make/View reservations **Pass/Fail Criteria:** The test will pass if the user makes a reservation when there is at least 1 free table. The test will fail if there are no free tables. **Input Data:** reservation	
Test Procedure:	**Expected Result:**
Step 1: User makes a reservation when there are no free tables. Step 2: User makes reservation when there is a free table	Server denies reservation and pops message to announce user that all tables are occupied Server accepts reservation and changes the status of the table to reserved

Test Case Identifier: TC-6 **Use Case Tested:** UC-5: View/Update Table Status **Pass/Fail Criteria:** The test will pass if the user changes table status while no one is sitting on the table. The test will fail if table is not empty **Input Data:** table status	
Test Procedure:	**Expected Result:**
Step 1: User changes table status while customers are sitting on the table Step 2: User changes table status while no one is sitting on the table	System will reject and pop up message that user can not change the table status at the moment System will accept and change the table status

Modified User Interface Requirements

Identifier	Test Case	Comments
**TC-1	Testing the log in/log out feature	Instead of login in/login out, the staff members will have to enter its 4 digit identifier and click continue to access the server interface. If the individual is not in the system,

		it will not allow it to access the server interface. The employee must be in the database.
*TC-3	Testing the rewards feature.	This is something that will be part of future work. Additional work on this application may implement these features.
*TC-4	Testing the payment feature.	This is something that will be part of future work. Additional work on this application may implement these features.
*TC-5	Testing the reservations feature.	This is something that will be part of future work. Additional work on this application may implement these features.

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

b. Test Coverage
All our essential system classes are encompassed by our test cases, crucial for the system's functionality. As we continue developing additional classes and methods, we'll adapt and supplement our test cases accordingly. Our testing approach involves comprehensive coverage of potential scenarios each class might encounter. Different parts of the project will have distinct test cases, incorporating diverse testing methodologies throughout our implementation process. Employing a bottom-up testing strategy, we'll break down the problem into smaller components, progressively building towards the main objective. During unit testing for use cases, our primary focus will be to ensure adherence to the project's requirements. If any discrepancies arise, adjustments to the use cases will be made to align with the project's specifications.

c. Integration Testing
We opted for employing the bottom-up testing strategy, which involves testing the lowest level components first and then leveraging these components to test higher-level components. This approach ensures that the foundational elements of the code function correctly before their integration into other sections. By adopting this method, we can test each layer of the project as we complete it. This testing method aligns well with our project's needs. An alternative testing method would pose greater difficulty in pinpointing the source of bugs, whether stemming from code integration or fundamental design and

coding issues within classes. The bottom-up approach offers efficiency and clarity by elucidating relationships between system objects, facilitating quicker identification of issues for resolution. Illustratively, we independently test and personalize tasks for each employee in the application. For instance, testing the chef's queuing system implementation ensures proper updates to the server when the chef signals a dish is ready. Subsequently, we assess features involving interactions between multiple objects or classes after testing individual functions.

13. History of Work, Current Status, and Future Work(Updated)

a. History of Work

August 21-September 4

After getting together after class and forming the team, we created a discord to brainstorm ideas for the project and establish a line of communication. After brainstorming for a day or 2, we all came up to the conclusion of doing a restaurant automation project since one of the members already had experience working at a restaurant and the rest of the members felt connected to such an idea. We got together on the 31st of August to write up the proposal and discuss our strengths and weaknesses to decide what we are all capable of accomplishing.

September 5-September 10

We used the feedback from the proposal to plan the next few steps of the project. Some of the feedback helped come up with possible solutions the professor brought to our attention. We also took into consideration previous projects and reports and the functions they created. We could not get together during this time but everyone contributed to their assigned parts of the report. Bjarni started working on the database design using a UML diagram. He also created a SQL query for the database.

September 11-September 17

We discussed Bjarni's database design during this time and the SQL query database he created. Also, we brainstormed user interface ideas and the appearance of the site. We came up with a model that best suited what we want our site to look like and accomplish. Lastly, we started working on part 2 of Report 1 and divided up the work.

September 18-October 1

We took this time to continue discussing models and we agreed on the MVC model. Shortly started implementing the user interface and database.

October 2-November 8

We took this time to create and implement the interfaces of the system. Each team member was assigned a specific interface. We did biweekly meetings during this time to focus on the implementation of interfaces. Ivan worked on the customer interface. Sokhna worked on the employee homepage and the manager interface. Cheikh worked on the server and chef interfaces. Bjarni continued working on the database of the project. We got together the week of Demo #1 to get ready for the presentation and discuss our designs for each of the interfaces.

November 9-November 19

We took time to integrate the front-end and back-end portions of the project. Unfortunately, we did not have a fully implemented project. We had to get rid of a few functional and nonfunctional requirements and we focused on the most important ones to meet the deadline of the project and at least have a much simpler but still efficient project.

November 27-December 8

We took this time to get started with the testing part of the project. We also used this time to fix bugs and finalize report 3. We updated any outdated diagrams and sections we lost points in prior reports. We also wrote our reflective essays and got together one last time to prepare and present our final model to the class during for Demo #2.

b. Current Work

We successfully developed fully operational user interfaces catering to both staff members and customers. Throughout this project, we achieved a significant portion of the critical functional requirements we identified as crucial. These achievements notably encompassed functionalities such as presenting menus to both customers and staff members, facilitating table visibility within the restaurant premises, enabling order placement and modification, and dynamically updating table statuses based on their order progress (whether empty or ready to be served). Our system also facilitated managerial control over the waitstaff and menu modifications, empowering employees to place orders, and granting chefs visibility into incoming orders. These accomplishments reflect a substantial portion of our initial goals, laying a strong foundation for the project's functionality.

c. Future Work

There remains a substantial amount of unfinished tasks within this project. Specifically, key components such as integrating the payment system for customers—enabling transactions via both cash and credit cards—remain unimplemented despite the presence of the foundational back-end code. Additionally, a crucial aspect of the project that was left unaddressed is the functionality enabling customers to create accounts and earn points based on their online purchases, with the necessary back-end infrastructure already in place. The majority of pending tasks revolve around the customer-facing features, presenting ample opportunities for improvement in the coming weeks or months. Lastly, the project lacks the implementation of managerial tools allowing for the tracking of net sales, beverage percentages, and similar managerial metrics. These outstanding components represent substantial areas for future development and enhancement within the project.

14. References

Links to previous restaurant automation reports
https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/
Software Engineering Project Report
https://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html
Restaurant Automation Project Description
https://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/RestaurantAutomation.pdf
10 Common Software Architectural Patterns in a nutshell
https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013