Report #3 - Final: Restaurant Automation

SpeedByte

(https://github.com/Cpawel/SpeedByte)

Group # 15:
Jimmy Jorge
Tim Gilligan
Jose Figueroa
Paweł Derkacz
Husen Valikrimwala
Ramaseshan Parthasarathy
Vamshikrishnan Balakrishnan

Individual Contributions Breakdown:

Person	Contribution
Jimmy Jorge	16.5%
Tim Gilligan	16.5%
Paweł Derkacz	16.5%
Husen Valikrimwala	16.5%
Ramaseshan Parthasarathy	16.5%
Vamshikrishnan Balakrishnan	16.5%
Jose Figueroa	1%

Table of Contents

Summary of Changes:

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

1. Customer Statement of Requirements (CSR):

Actor - Host/Hostess:

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

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

Actor - Manager:

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

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

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

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

Actor - Busboy:

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

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

Actor - Customer:

I'm really picky about my food - I prefer knowing that the food I'm about to get is near perfection, at least to most people. Some kind of rating system might help me with decisions regarding just that. In consideration of the foregoing, I also want to know what exactly is in the food, in the case of allergies and for calorie counting purposes. All this talk about food has made me hungry, so I'll probably call somewhere for food; actually, that gives me another idea! If there was a digital menu, I wouldn't have to go through the hassle of having to speak with someone and waste time trying to understand them as they try to understand me - my phone isn't especially good, so this is more of a problem than you might think. And perhaps if I could order ahead of time, that would cut down my waiting to get the food - fantastic! Speaking of waiting, if

the menu also displayed how long an item would take to make, then I would know when to arrive at the restaurant to pick up the food when it's nice and hot. Although, there are also occasions when I like to just eat at the restaurant, but the line goes through the door sometimes! If only there was a way everyone already knew where to sit and could just get to it. What if there was a way to combine some of those ideas! Hey, let's say I order off of this menu, then maybe I can also reserve a seat for myself, and pay the bill while I'm eating. That way, I already have a seat and food ready to go - presto, time saved. Man, I'm a genius, eh?

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

Actor - Chef:

Day in and day out, I work in the kitchen - being a chef isn't terrible, but boy is it stressful. Having to make sure orders are filled out in a timely fashion and keeping track of dishes that have a few modifications requested to them by the customer really does a number on me sometimes. Not to mention, sometimes what the waiter writes down looks like downright chicken scratch, it all just makes the job harder than it has to be. So, what I would like to get from you guys is something that will help me read incoming orders clearly, with any possible

modifications done to them in a different color. That way, I don't have to rely on the waiter's chicken scratch, and I can clearly see what is different about this meal from the standard. Next on my list would be something to help tell the waiter that the food is ready to pick up - that's another one of my pet peeves; I have a fresh meal ready for a customer and it just sits around for 5 minutes, taking up space on the counter for more dishes and getting cold all the while.

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

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

Actor - Waiter/Waitress:

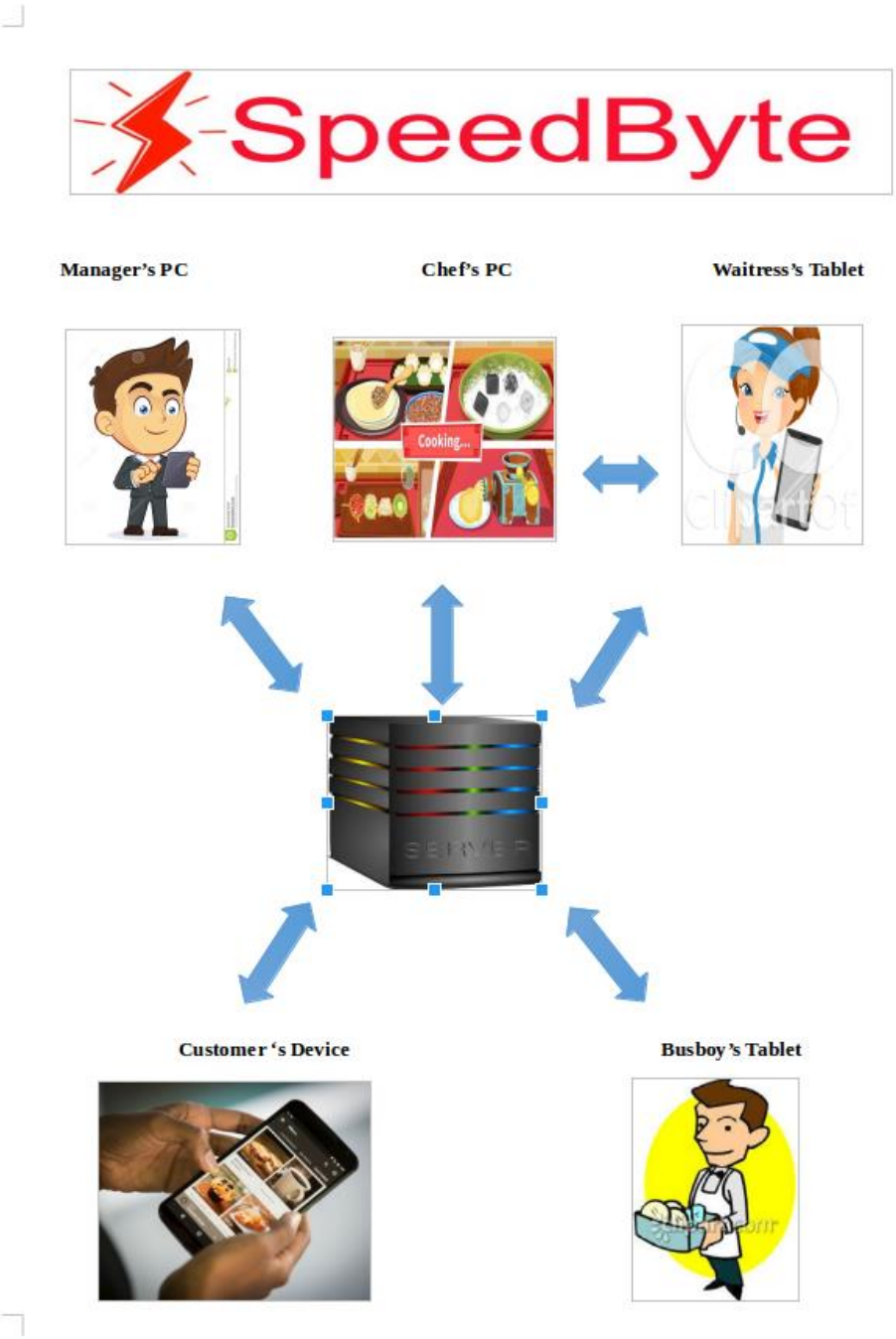
Man do I hate when the chef yells at me - he gets so agitated when I don't pick up stuff from him in the time he wants me to. It's not like I diddle-dally my way around the restaurant. I have customers to attend to as well, taking their orders and, well, waiting on them. I hear you guys are already implementing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

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

2. Glossary of Terms:

- **Customer:** Orders food and services from the restaurant.
- **Manager:** Manages inventory, payroll, employee list and charts, customer's bill in order to provide discount due to inconvenience, log-in interface to prevent unauthorized access of restaurant management and statistics for the restaurant.
- **Waiter:** Delivers the order.
- **Chef:** Reads the order placed from a terminal in the kitchen and cooks food accordingly.
- **Busboy:** Keeps track of the dirty tables and updates once the cleaning is done.
- **Host:** Assigns seats to people who come to the restaurant. Changes the status of table to "Occupied" to "Vacant". Keeps track of clock in and clock out function for payroll purpose.
- **Add/Edit Employee:** Button only on the Management page to add or edit the information of an employee at the restaurant.
- **Manage Inventory:** Button only on the Management page of all the items required for food preparation in the restaurant.
- **Manage Payroll:** Button only on the Management page to manage the payrolls of the various employees in the restaurant.
- **Reports Screen:** Statistical and graphical data analysis of the traffic flows in the restaurant.
- **Menu:** List of dishes served in the restaurant, which will be displayed on an android application.
- **Floor Layout:** Shows all tables in the restaurant along with their respective status.
- **Order Status:** Shows whether the order of a particular table is ready to be "served" or "cooking".
- **User interface:** The visual on the computer and tablet that allows user interaction with the system.
- **Wireless Access Point:** Accesses points support Wi-Fi wireless communication standard.
- **Database:** Where all the data such as menu items, inventory, scheduling, payment, payroll, employee information, customer information and orders are stored. (http://i.stack.imgur.com/iTINi.png)
- **Order Queue:** a list of orders that are placed in (FIFO) first in, first out order. These orders are sent to the chef's PC, where the chef can prepare them.
- **Walk-in Queue:** a list of table reservations that are placed in (FIFO) first in, first out order. These reservations are made on the Customer PC when the customer walks into the restaurant.
- **Reservation System:** Customers can reserve a table using an android application.

3. System Requirements:



3.1) Enumerated Functional Requirements

IDENTIFIER	PRIORITY	REQUIREMENT
REQ-01	5	Allow the customers to choose whether they want to dine in, take-out, or get food delivered.
REQ-02	5	Provide the customer with a click able menu.
REQ-03	3	Customer should be able to choose seating if dining.
REQ-04	5	The Chef PC should receive the order.
REQ-05	2	View estimated time of arrival for food and/or service.
REQ-06	4	Customer should be notified of order.
REQ-07	5	Customer should not be able to login as employee.
REQ-08	5	Customer should be able to choose the option to pay. (Cash or Online)
REQ-09	4	Busboy's tablet should be able to update the floor layout by changing table's status.
REQ-10	1	Manager's PC shall track restaurant trends.
REQ-11	1	Manager's PC should be able to track inventory of each dish sold over a period of time.
REQ-12	4	Chef's PC alerts waiter/Waitress when food is ready to be served/delivered.
REQ-13	2	Busboy's tablet shall notify the busboy what tables to clean.
REQ-14	4	Host shall Prioritize orders in a queue.
REQ-15	4	Chef's PC shall notify the system to update inventory when a meal is ready to be delivered.

REQ-16	3	Manager's PC should be able to access data.
REQ-17	2	Manager's PC should be able to analyze and predict supply usage.
REQ-18	4	Manager's PC will have the ability to edit the menu.
REQ-19	2	Manager's PC shall alert the manager when the inventory has a low stock of a particular item.
REQ-20	3	Host will let Chef's PC and Waiter's PC know which customers came first.
REQ-21	1	Manager's PC will have statistics on food popularity.
REQ-22	3	Host Should allow customer to rate each dishes
REQ-23	5	All data should be stored in the database.

3.2) Enumerated Nonfunctional Requirements

Identifier	Priority	Requirement
REQ-24	1	The program should be user-friendly and meet the standards of the restaurant.
REQ-25	4	User Manual should be made to provide assistance.
REQ-26	5	The system should be backed up to avoid loss of data.
REQ-27	2	Safety protocols should be placed to protect the data from unauthorized users.
REQ-28	2	Operating time between screens should be minimized.
REQ-29	5	The system should have a low mean time to failure (MTTF) and high reliability components.
REQ-30	1	The system should work on any device.
REQ-31	3	The system should be relatively easy to debug.
REQ-32	2	Employee devices should be convenient and easy to operate.

Usability:

The system will be user friendly and easy to operate. The users will download an app and create a login. Manager and Employees will have their own digital device to interact with and assist the customers. The customers will have options to dine-in, take-out, or have their food delivered. The restaurant's entire menu will then be available for the customer to choose from. The order will be transferred to the manager's and chef's devices with all the information needed.

Reliability:

The system will consistently excel in performance. The initial options such as dine-in, take-out, and delivery, will be the first available options. Then, the menu option will be available for the customer to choose from. There is no chance of getting "invalid input" as the system will have all the specific options on the screen. Each user will have a unique username and password. Hence, there will be fewer risks of unauthorized and unknown users to hack into the system. All the options and security access will be available to the administration only.

Performance:

The system will be used by many customers and employees of the restaurant simultaneously, and will be able to handle it without any failures. To run the system efficiently and quickly, even with all of the possible traffic, the server will have to be efficient. To analyze the restaurant's performance for example: sales and inventory purchase, manager will have the option to compile the data all together and check the weekly/monthly performance. The restaurant will have a high-speed wireless connection for the system to perform its best.

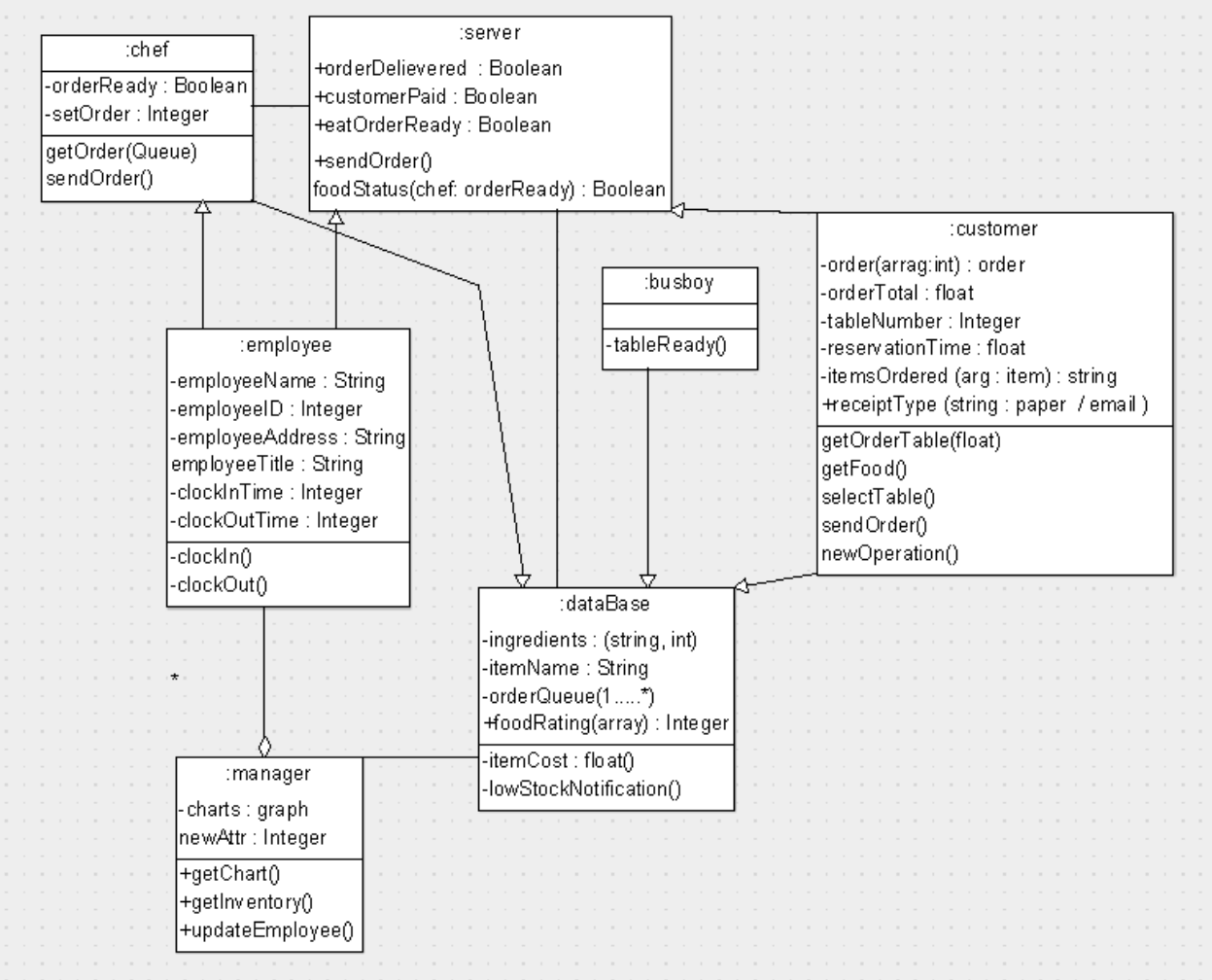
Supportability:

The system will support the modification. All the changes and replacement that an application might need in future will be allowed. For instance: change in menu, replacing the logo. Manager will have the ability to modify the menu. The restaurant manager will be the lone administrator of the system.

Implementation Requirements:

Android studio will be used to create our app. Node.JS will be used to create a server. Firebase will be used to implement a database to hold user/pass information.

Class Diagram:



3.3) On-Screen Appearance Requirements

Identifier	Priority	Requirement
REQ-33	5	SpeedByte will display a current seating chart of available tables to the customer.
REQ-34	3	SpeedByte color schemes will be visually appealing and appetizing.
REQ-35	4	Menu will display ingredients, wait time, and rating
REQ-36	5	SpeedByte will display interactive charts to manager tracking inventory, sales, etc.
REQ-37	3	Will display current order's recipe to chef
REQ-38	5	Will display what tables are "Available, Occupied, or Need to be Cleaned" to employees.
REQ-39	1	Will include pictures of each menu item
REQ-40	3	Scroll down menus for all options to eliminate unnecessary clicks

User Interface:

Use Case Scenario

- Customer opens app and has three options (Dine-in, Takeout, and Delivery).
- Customer chooses Dine-in option.
- Customer is asked to choose a reservation time.
- Customer selects "Eat Now."
- Customer is presented with a seating chart displaying available tables.
- Customer chooses table 4.
- Hostess checks the name and seats the customer.
- Customer is brought to the menu screen.
- Customer chooses "Pasta Faggioli" with a rating of 4.8/5 stars and wait time of 10 minutes.
- Customer is brought food after a ten minute wait.
- Customer eats food and pays bill.
- Customer is presented with a request to rate the dish.
- Customer rates dish 5 stars.
- Customer leaves the restaurant satisfied.

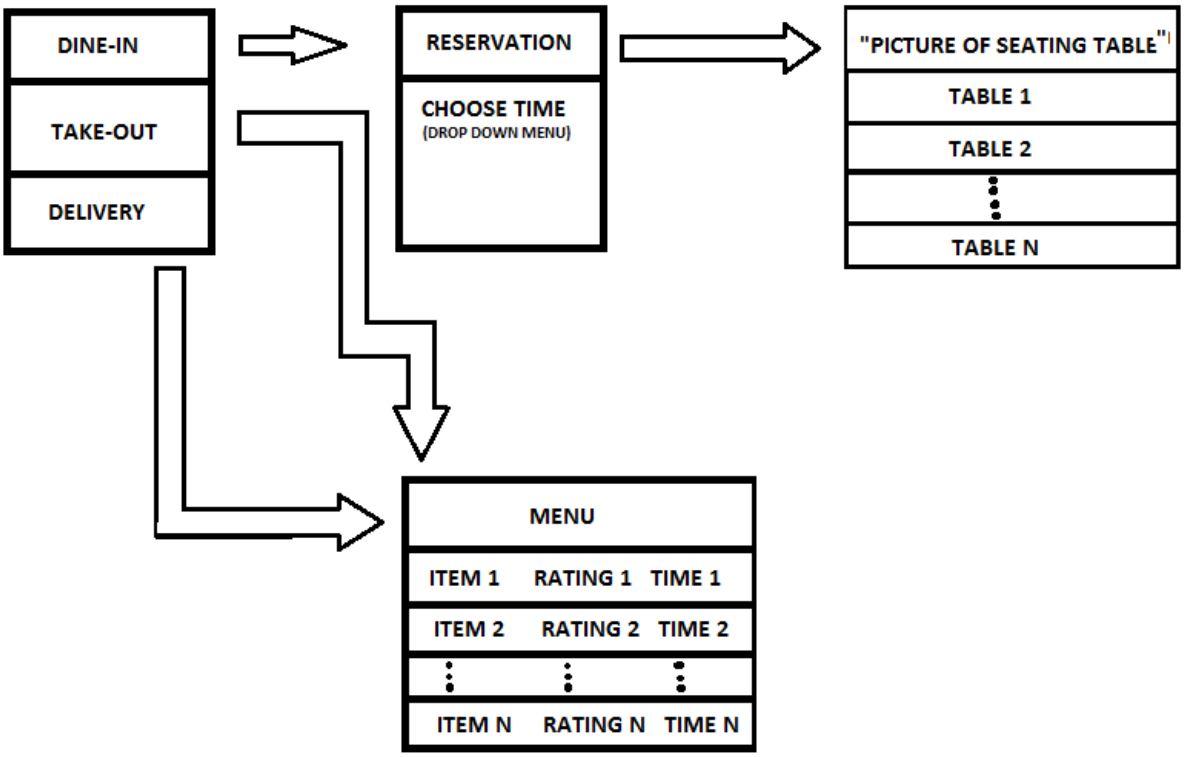


Figure: A sample on-screen interface for SpeedByte

(Updated interfaces in User Interface section..)

4. Functional Requirements Specification:

4.1) Stakeholders:

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

Restaurant Owner/Manager:

- Message Center will provide the manager easy communication among employees
- Statistics page will provide accurate details that will help improve overall profits
- Rating feature will show most popular dishes
- Inventory Tracker will notify the Manager when to order more food.
- On App ordering will eliminate the need for multiple waiters which will eliminate high cost.

Employee:

- Easy Interface will notify employees when to do their required task which will keep them on task.
- Message center will allow for easy communication between employees.

Customer:

- Rating System will show what dishes are recommended by previous customers.
- Approximated wait time will let customer know how long he will be waiting for dish.
- Removes the high cost for tipping waitress

4.2) Actors and Goals:

The following are the actors who will have to perform their duty as assigned (in chart below)

4.3) Use Cases:

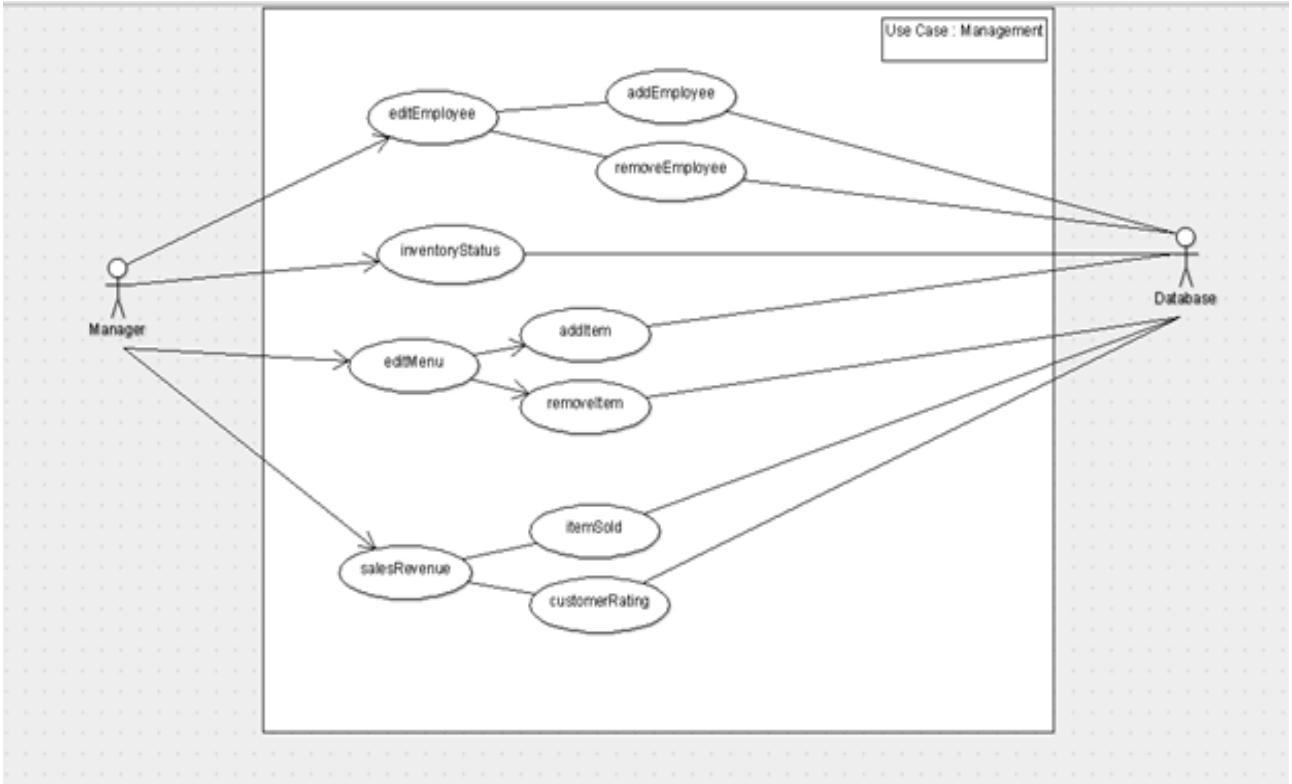
i.) Casual Description

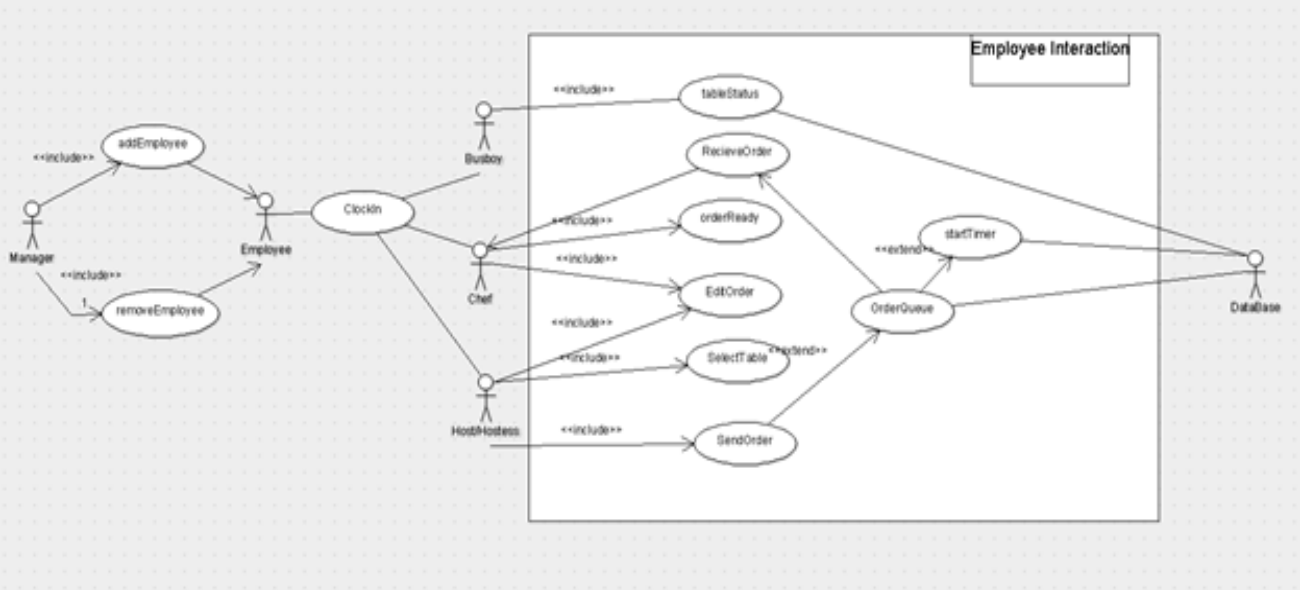
Actor	Goal	Use case	Casual Description	Type
Customer	Select Table	UC-1	Customer gets an option to choose a table.	Initiating
Customer	Place an order	UC-2	To choose dishes to place an order.	Initiating
Manager	Create Employee Account	UC-3	Create login for newly hired employee.	Initiating
Manager	Edit Employee Account	UC-4	Edit information of current employees. For ex: Adding/changing First name, Last name, Address, Contact detail, SSN, etc.	Initiating
Manager	Add Account	UC-5	Add more employee accounts.	Initiating
Manager	Delete Account	UC-6	Remove existing accounts of the employees who no longer work in the restaurant.	Initiating
Manager	Edit Menu	UC-7	Add/Delete Menu.	Initiating
Manager	Track Traffic	UC-8	Keep track of flowing traffic (customer).	Initiating
Manager	Manage Inventory	UC-9	Keep Track of Inventory.	Initiating
Manager	Manage Payroll	UC-10	Keep track of clock in/clock out of all the employees and do the payroll weekly accordingly.	Initiating
Manager	Manage Payment	UC-11	To verify if the payment is received online or not. To collect cash payment if a customer pays cash.	Initiating
All Employee	Login	UC-12	All employees will have to login to work.	Participating

Employee (Waitress)	Delivering order	UC-13	Employee will deliver the order. (Be it dine-in, take out or delivery)	Initiating
Busboy	Cleaning	UC-14	To clean and prepare a table right after a customer leaves.	Initiating
Chef	Prepare the order and Signal for a pick up	UC-15	To cook the food and signal the employee once the order is prepared and ready to be delivered.	Initiating
System	Recognizing an input or waiting number	UC-16	To recognize use's input and assign a vacant table to a customer. If the table is not available, it assigns a waiting number.	Initiating
System	Update Floor layout	UC-17	To change the status of all tables to Available or Occupied (Avail. => Occupied) (Occupied => Avail.)	Initiating
Database	Storing the data	UC-18	To update inventory, store employee information, and manage any other data.	Participating
System	Help	UC-19	The system should allow a customer to notify a waiter for assistance Through the android tablet instantly.	Initiating
System	Notify Employees	UC-20	To notify all the employees for a specific change in system. For instance: Notify manager regarding low inventory.	Initiating
Employees	Send Messages	UC-21	Employees should be able to send preset messages to each other.	Initiating

Customer	Create Account	UC-22	Customer should be able to create account to login	Initiating
Customer	Rating	UC-23	Customer should be able to rate individual dishes.	Initiating
Database	Successful Login	UC-24	To authenticate login credentials in order to login successfully	Initiating
System	System Confirmation	UC-25	System should be to display the confirmation message to the sender and recipients	Initiating

ii.) Use Case Diagram





iii.) Traceability Matrix

Green = highest priority. Yellow = second highest priority. Red = least priority.

REQ	PW	UC 01	UC 02	UC 03	UC 04	UC 05	UC 06	UC 07	UC 08	UC 09	UC 10	UC 11	UC 12	UC 13	UC 14	UC 15	UC 16	UC 17	UC 18	UC 19	UC 20	UC 21	UC 22	UC 23	UC 24	UC 25
01	5	X	X											X									X		X	
02	5	X	X					X				X						X							X	
03	3	X													X	X		X							X	
04	5		X						X	X							X	X			X	X				X
05	2		X						X		X				X	X	X	X	X				X		X	X
06	4		X											X				X					X		X	
07	5		X																				X		X	
08	5		X				X		X	X	X	X							X				X	X	X	X
09	4						X																			
10	1			X	X	X	X	X	X	X	X	X														
11	1		X	X			X	X		X	X		X						X		X				X	
12	4		X						X	X	X	X							X						X	
13	2						X			X	X	X	X				X	X	X						X	
14	4						X		X	X	X	X													X	
15	4		X											X	X	X	X	X	X						X	
16	3			X	X	X	X	X	X	X	X														X	
17	2			X						X									X						X	
18	4			X			X												X						X	
19	2			X												X									X	
20	3																X	X				X			X	
21	1			X					X																X	
22	3																							X	X	
23	5		X	X	X	X	X	X	X	X	X	X					X	X								
24	1		X	X									X			X	X	X	X	X						
25	4		X	X															X	X						
26	5			X																						
27	2		X	X															X						X	
28	2		X	X															X							
29	5	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X								
30	1	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X							
31	3	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X							
32	2			X	X	X	X	X	X	X	X	X	X			X	X	X					X			
33	1	X																X							X	
34	3	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X								
35	2	X	X																							
36	1		X	X					X	X	X														X	
37	3															X									X	
38	1														X		X	X							X	
39	5	X	X					X																		
40	3	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X								

iv.) Fully-Dressed Description

The following are the few important use cases along with their goals, Main Success Scenario and Alternate Success Scenario.

Customer/Employee/Host/Busboy/Database

Use Case: UC-1, UC-2, UC-17, UC-14	Select table (Dine in), Place an order, and Update floor layout.
Initiating Actors	Customer, Host, Busboy
Participating Actors	Employees, Chef, Database
Goal	The app will allow placing an order and choosing a table to dine in the restaurant.
Preconditions	The user already has an account created and is willing to dine-in the restaurant.
Post conditions	The Host assigns a table or a waiting number.

Main Scenario:

- Customer opts to dine-in and then chooses a table.
- System updates the floor layout once the table is chosen (changes the status of table from “available” to “occupied.”)
- The Host brings customer to the chosen table.
- Customer places order.
- Chef receives order and begins preparing order.
- Chef finishes preparing order, and signals the employees the food is ready.
- Employee receives signal and brings prepared order to the customer’s table (employee confirms order delivery).
- Customer finishes eating and pays the bill.
- System notifies the busboy to clean the table.
- Busboy cleans and prepares the table for future customers.
- Busboy notifies the system by changing table’s status to “available”.
- System updates the floor layout with the new available table(s).

Alternate Scenario:

- Customer selects unavailable table.
- Host recognizes user’s input and notifies customer with a message “Table is occupied” and asks user to wait or to choose another table.
- If customer opts to wait, Host updates the waiting queue in order; else employee updates the floor layout once the table is chosen.

Manager/Employee/Database

Use Case: UC-4	Edit Employee Account
Initiating Actor	Manager
Participating Actor	Employee and Database
Goal	Edit existing employee account in the database.
Preconditions	An employee entry exists in the database whose information needs to be updated.
Post conditions	Employee information is updated.

Main Success Scenario:

- Manager selects the interface to edit the existing employee account in the database. For Instance: To change First name, Last name, DOB, Address, password, SSN, etc.
- Manager and system must have SSN.
- Validate the fields.
- Employee information gets updated.

Alternate Scenario:

- Manager selects the interface to edit the information for an existing employee account in the database and a same name account with all the same information for example: First name, Last name, DOB and SSN already exists.
- System notifies an error message “Information cannot be updated and stored as same account already exists more than once”.
- Manager verifies and deletes one same account from the database.
- Manager accesses the sole account to update and save the information again.
- System allows and information gets updated and stored.

Chef/Employee/Database

Use Case UC-15	Preparing the Order
Initiating Actor	Chef
Participating Actor	Employee and Database
Goal	To cook the order and notify the employee when it is ready.
Preconditions	Customer places an order and chef receives the order.
Post conditions	The employee is notified when the food is ready to be delivered to the table. Plus, each time an order is prepared, the corresponding inventory items are decreased by the pre-specified quantity.

Main Success Scenario:

- Chef receives an order
- Once the order is prepared, Chef clicks on the button “ready”.
- System changes the status of the order to “ready” and notifies the employee.
- Employee picks up the order and delivers to the customer.
- System updates the inventory in the database.

Alternate Scenario:

- Chef receives an order and mistakenly changes the status of an order to “ready” with in few seconds.
- Chef must contact employee immediately regarding the mistake.
- Chef must fix the mistake on his tablet.
- System updates employee regarding the change in status of the order.
- System updates the inventory in the database.

Message Center - Customer/Employee/System/Database

Use Case: UC-21	Send messages
Initiating Actor	Employees
Participating Actor	Database
Goal	To send preset messages.
Preconditions	Message must exist in the database.
Post conditions	Message sent/Received.

Main Success Scenario:

- ➢ Manager chooses a preset message from the database.
- ➢ Manager sends a message to an employee.
- ➢ System notifies manager “Message Sent”
- ➢ System notifies employee “Message Received”

Encrypted Login - Employee/System/Database

Use Case: UC-12, UC-25	Login and Successful Login
Initiating Actor	Customer and Employees
Participating Actor	Database
Goal	To Login Successfully.
Preconditions	Customer and Employee must have existing.
Post conditions	Login successfully.

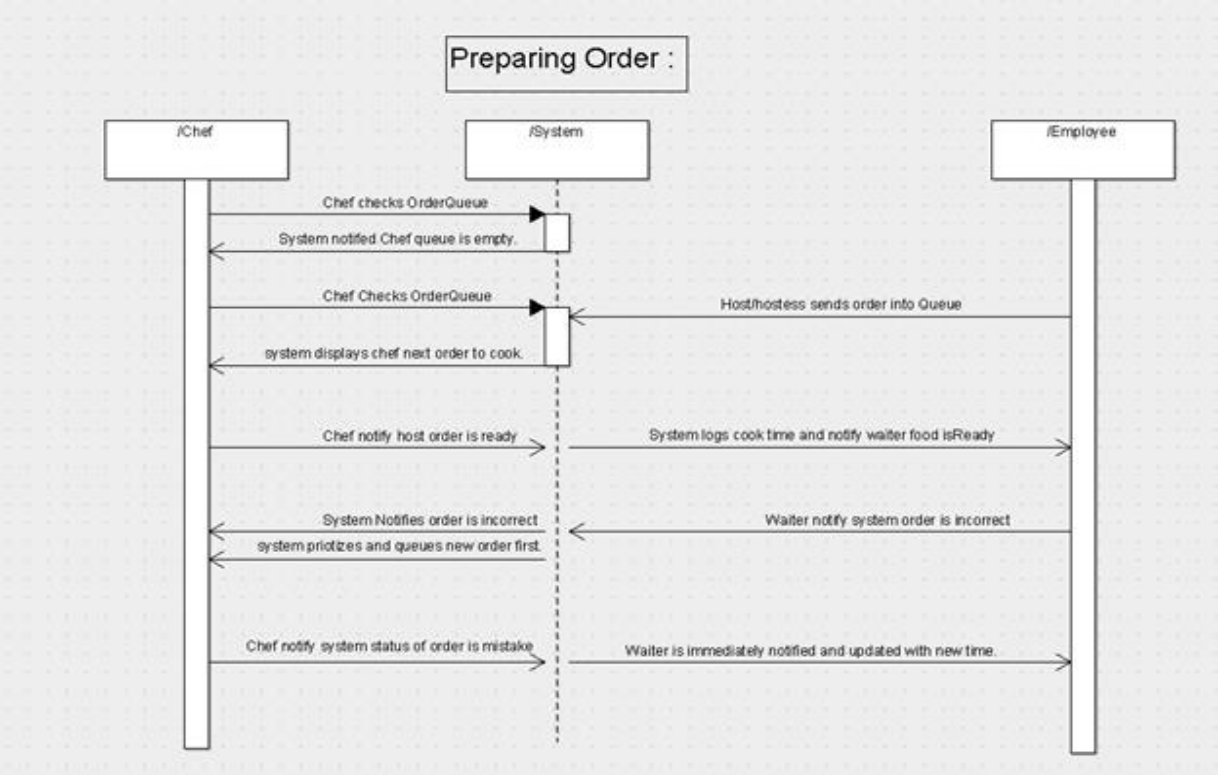
Main Success Scenario:

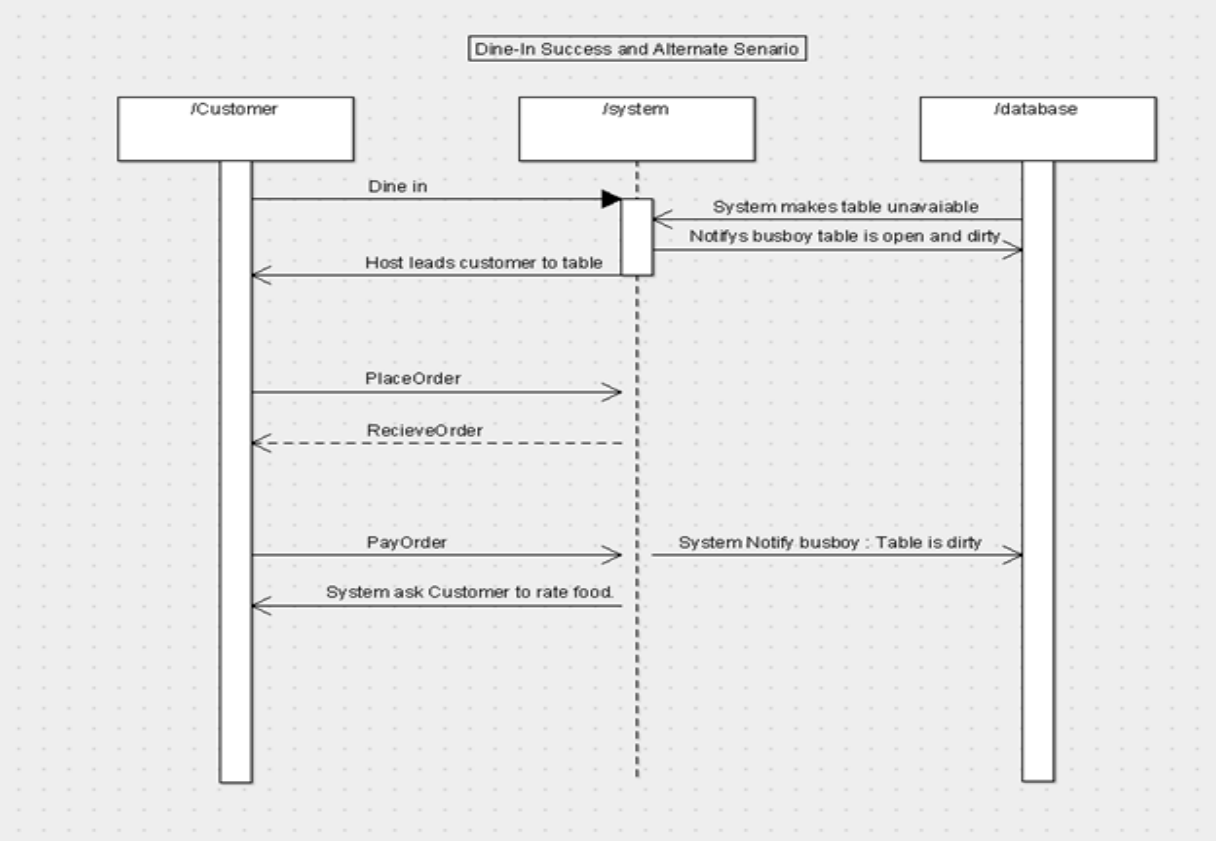
- ➢ Customer/Employee enters the login credentials (User ID and Password)
- ➢ System goes to the database and authenticates if the account exist or not.
- ➢ Validate the account.
- ➢ Login successfully.
- ➢ System takes Customer/Employee to the work center screen.

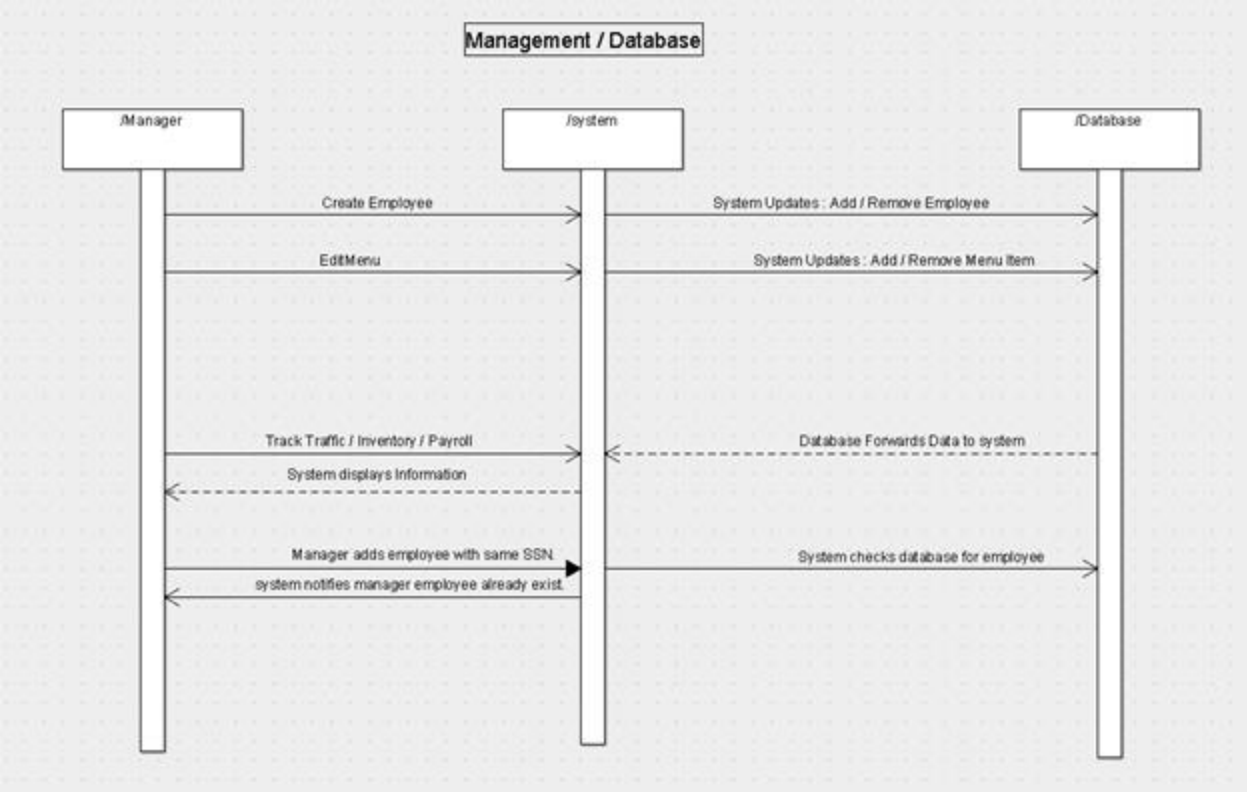
Alternate Scenario:

- ➢ Customer/Employee enters invalid login credentials (User ID and Password)
- ➢ System goes to the database and authenticates if the account exist or not.
- ➢ System notifies “ Invalid Login”
- ➢ Returns to the Login screen.

4.4) System Sequence Diagrams:






Management / Database
/Manager
/system
/Database
Create Employee
System Updates : Add / Remove Employee
EditMenu
System Updates : Add / Remove Menu item
Track Traffic / Inventory / Payroll
Database Forwards Data to system
System displays Information
Manager adds employee with same SSN
System checks database for employee
system notifies manager employee already exist

5. Effort Estimation:

Customer Interface:

The effort estimation varies greatly depending on the number of meals that the user orders as well as the dining option that is chosen.

Best Case Scenario:

- ➢ Customer selects Delivery. (1 click)
- ➢ Customer selects one dish. (1 click)
- ➢ Customer clicks checkout. (1 click)
- ➢ Customer pays cash.
- ➢ Customer selects not to rate the meal and instead closes the app.

Total clicks = 3

Worst Case Scenario:

- ➢ Customer selects Dine In option. (1 click)
- ➢ Customer selects a time from the scroll down menu. (1 click)
- ➢ Customer selects a table. (1 click)
- ➢ Customer selects N menu items (N clicks)
- ➢ Customer selects check out. (1 click)
- ➢ Customer rates each item. (N clicks)
- ➢ Customer closes app.

Total clicks = 4 + 2*N

Employee Interface:

The employee interface depends on different variables, like customer input.

Chef:

- ➢ Chef selects a table from the list (1 click)
- ➢ Chef reads menu items that are in queue to be prepared and clicks “Start dish.” (1 click)
- ➢ Chef completes dish and chooses “Dish complete.” (1 click)

Best and Worst Case Scenario:
Total clicks = 3

Server:

- Server sees that all dishes for table are prepared and clicks “brought to table.” (1 click)
- Server collects money from customer and selects “Meal Paid.” (1 click)

Worst and Best Case Scenario:
Total clicks = 2

Management Interface:

The management interface receives data from all of the other interfaces and logs them appropriately. The manager will be able to keep track of statistics such as inventory, income, and amount of customers, within a certain time period. These can be displayed as graphs for ease of use.

Manager selects Statistics:

- Manager opens app and is presented with clickable options for each statistic.
- Manager chooses an item to observe. (1 click)
- Manager is brought to a graph of that items progress hour by hour for the current day.
- Manager has the option to change time by selecting a preset time being (hourly, daily, weekly, monthly, and yearly). (1 click)
- Manager closes the app.

Best Case Scenario:
Total clicks = 1

Worst Case Scenario (per item):
Total clicks = 2

Manager sends Message:

- Manager selects preset message from database (1 click)
- Manager selects recipient from drop down list (1 click)
- Manager clicks “send” (1 click)

Total clicks = 3

Manager Successful Login:

- Manager types in username (1 input)
- Manager types in Password (1 input)
- Manager clicks login (1 click)

Total effort = 1 click, 2 inputs

Project Duration:

Use Case	Weight
1) Main Customer Scenario	**8**
2) Main Manager Scenario	**6**
3) Main Chef Scenario	**4**
4) Main Server Scenario	**4**
5) Main Busboy Scenario	**3**
6) Main Host Scenario	**2**

UUCP = 50

TCF	TCF Weight	TCF perceived Complexity	Complexity Factor
1	**2.0**	**5**	**2.0*5 = 10**
2	**2.0**	**5**	**2.0*5 = 10**
3	**1.5**	**4**	**1.5*4 = 6.0**
4	**1.5**	**3**	**1.5*3 = 4.5**
5	**1.0**	**4**	**1.0*4 = 4.0**
6	**1.0**	**3**	**1.0*3 = 3.0**

TCF Calculations:

TCF = C1 + C2 * Technical Factor Total = C1 + C2 * $\sum_{i=1}^{6}(W(i) * F(i))$

TCF = 0.6 + (0.01*37.5) = 0.975

TCF = 0.975

ECF	Weight	Perceived Impact	Calculated Factor
1	**0.5**	**1**	**0.5**
2	**1.0**	**3**	**3.0**
3	**0.5**	**2**	**1.0**
4	**1.0**	**4**	**4.0**
5	**1.5**	**3**	**4.5**
6	**1.0**	**1**	**1.0**

ECF Calculations:

ECF = C1 + C2 * Environmental Factor Total = C1+C2 * $\sum_{i=1}^{6}(W(i) * F(i))$

= 1.4 + (-0.03*14) = 0.98

ECF = 0.98

UCP Calculations:

UCP = UUCP * TCF * ECF

= 50 * 0.975 * 0.98 = 47.775

UCP = 47.775

Duration Calculations:

Duration = UCP * PF

= 47.775 * 28 = 1337.7

Duration = 1337.7 hours

6. Domain Analysis

6.1) Domain Model

i) Concept Definitions

Responsibility Description	Type	Concept Name
Populating reservation list with customers that have already ordered and selected a seat	K	Host/Hostess Menu
Check if seat selected by customer is available, waiting for new input if seat chosen is unavailable.	D	Customer Menu/Server
Recording of all information that is being sent, regarding food ordered, and time and day of order.	D	Server
Displaying in a coherent way all data that was collected, in the form of an infographic.	K	Server
Keep a backlog of authorized users and their passwords for different restaurants (for employee log-ins).	K	Server
Handle attempts to log in as an employee, and which menu to display for that employee.	D	Server
Acquire and display current items that have been ordered (in order for the chef to have a list of meals to make).	K	Chef Menu
Update seats in order to specify whether it is available, occupied, or in need of cleaning.	D	Server
Inform busboy when a table updates from occupied, so that it may be cleaned off.	K	Busboy Menu
Prompt users for a rating of the meal they enjoyed, and record the result.	D	Customer Menu
Display current stock to authorized user, and allow the user to modify the data.	D	Manager Menu
Display current employees to authorized user, and allow the user to make modifications (possibly to only users below authorized users standing - to prevent users from updating their own wages and such).	D	Manager Menu

Remind waiter of any tables that still have open orders, so that customers can get their orders in.	K	Waiter Menu
For deliveries and take-outs, have an active timer ticking down the ETA.	K	Customer Menu
For the chef, when he finishes making the meal, have him update the meal status so that the waiter is called to pick up the food.	D	Chef Menu and Waiter Menu
Have system allow for splitting the bill, and calculate the necessary splits.	D	Waiter Menu and Customer Menu
Modify food items on the menu from authorized user	K	Manager Menu

ii) Association Definitions

Concept Pair	Association Description	Association Name
Manager Menu ⇔ Server	An authorized user can prompt the server, from the Manager Menu, to deliver statistics on customer flux. From the Manager Menu, the user can also update information server-side, regarding other employees, food items, updates to stock, etc.	Send statistics
Host/Hostess Menu ⇔ Customer Menu	The customer menu sends updates to the Host's/Hostess' Menu regarding a reservation, if the customer has chosen to dine-in	Show reservation
Chef Menu ⇔ Waiter Menu	Chef sends information to waiter regarding the status of food, so that the waiter can pick it up in a timely fashion. The waiter delivers any orders from customers that don't use the app.	Sends status of food
Chef Menu ⇔ Customer Menu	When the customer orders through the app, the Chef's menu gets updated with their order. The Chef's menu updates the customer's menu to display how the food is coming along.	Update Food Status
Server ⇔ Customer Menu	Server keeps track of what the customer ordered, at what time, in what quantity, in order to have local data be as up to date as possible.	Receive Food Information
Waiter Menu ⇔ Busboy Menu ⇔	When the waiter has received payment, the table the payment was received from updates to the "dirty"	Review Table Status

Host/Hostess /Customer Menu	status, thus indicating to the busboy that a table needs cleaning. When the table is cleaned off, the busboy menu sends an update to the Host/Hostess and Customer menus indicating the table is available.	
Host/Hostess Menu ⇔ Customer Menu	When the host/hostess takes care of seating non-app users, those seats become updated to unavailable on the Host/Hostess Menu along with the Customer Menu.	Update Reservation

iii) Attribute definitions

Concept	**Attribute**	**Description**
1. Host Menu	listReservations	List of customer's reservation
2. Customer Menu	tableStatus	Checks to see if table is clean and available.
3. Server	listItem orderTime	Lists how many items ordered on the menu Provides Time and day of ordered food.
4. Server	graph	Provides and builds graphs for management.
5. Server	checkUser checkPassword	Checks if user is authorized to access data Checks if user input correct password associated with account.
6. Server	userLogin showMenu	User Logins into account Provides appropriate menu for employee
7. Chef Menu	getOrder	List food ordered from orderQueue
8. Server	tableStatus	Provides and updates status of table.
9. Busboy Menu	getTableStatus	Displays table status if empty and if it is clean/dirty
10. Customer Menu	rateItem	Allows user to rate meal
11. Manager Menu	getInventory	List current stock

	orderInventory	Add items to inventory
12. Manager Menu	displayEmployee editEmployee	Displays employee data to user Modify employee wage/address/name
13. Waiter Menu	openOrder	Checks if table has an open order.
14. Customer Menu	orderTime	Provides estimated time of food delivery for deliveries and take-out
15. Chef Menu	orderReady	Updates status of order
16. Waiter Menu	orderStatus	Checks if item ordered by customer is ready.
17. Waiter Menu / Customer Menu	splitBill	Allows user to split bill in multiple parts.
18. Manager Menu	editMenu	Modify food menu from authorized user.

iv) Traceability Matrix

	C-01	C-02	C-03	C-04	C-05	C-06	C-07	C-08	C-09	C-10	C-11	C-12	C-13	C-14	C-15	C-16	C-17	C-18
UC-01																		
UC-02	X	X																
UC-03												X						
UC-04					X							X						
UC-05					X							X						
UC-06					X							X						
UC-07																		X
UC-08			X								X							
UC-09			X		X													
UC-10					X							X						
UC-11					X							X						
UC-12						X												
UC-13													X					
UC-14								X	X									
UC-15							X								X			
UC-16	X							X					X					
UC-17								X					X					
UC-18																		
UC-19		X																
UC-20		X										X						
UC-21																		
UC-22					X	X												
UC-23										X								
UC-24					X	X												
UC-25																	X	

6.2) System Operation Contracts

Name:	Select Table
Responsibilities:	To select and update tables to either sit customers or to be cleaned
Use case:	UC-1
Exceptions:	None
Preconditions:	Table must be empty, either before cleaning or after cleaning
Post conditions:	Table is ready to be cleaned or table can now sit customers

Name:	Place an Order

Responsibilities:	To place an order and notify the chef.
Use case:	UC-2
Exceptions:	None
Preconditions:	Order is placed by all the users
Post conditions:	Order is placed in queue and chef is notified

Name:	Create Employee Account
Responsibilities:	To create a new account for a newly hired employee
Use case:	UC-3
Exceptions:	Only administration is allowed to create an employee account
Preconditions:	Administrator is allowed to create an account.
Post conditions:	Employee account is created.

Name:	Edit Menu
Responsibilities:	To modify the menu
Use case:	UC-7
Exceptions:	Only administration is allowed to edit menu
Preconditions:	Menu is available to be updated.
Post conditions:	Menu is updated.

Name:	Manage Inventory
Responsibilities:	Update the ingredients based on users input
Use case:	UC-9

Exceptions:	Item that does not exist in the database cannot be updated
Preconditions:	Inventory database has each item either 0 or n amounts
Post conditions:	Inventory either increments or decrements the quantity of item(s). Inventory will notify if there is an item out of stock.

Name:	Manage Payment
Responsibilities:	To prepare the bill
Use case:	UC-11
Exceptions:	Only administration is allowed to manage payment
Preconditions:	Table ordered has been completed and all the items are delivered.
Post conditions:	Payment is received.

Name:	Login
Responsibilities:	To allow users to input their ID and password
Use case:	UC-12
Exceptions:	Unauthorized with invalid inputs users cannot login in the system.
Preconditions:	Login information is available.
Post conditions:	User is either login or rejected based on the user valid input.

Name:	Delivering Order
Responsibilities:	Item status is changed by chef and waiter is notified that order is ready to be delivered
Use case:	UC-13

Exceptions:	None
Preconditions:	Order is waiting to be prepared by chef.
Post conditions:	Chef notifies the waitress that the order is ready to be delivered. Inventory is updated.

Name:	Cleaning
Responsibilities:	Table status has been changed to clean.
Use case:	UC-14
Exceptions:	Only the busboy is allowed to change the status of the table to clean (available) and dirty (unavailable).
Preconditions:	Table is dirty (unavailable).
Post conditions:	Table status is changed to clean (available).

6.3) Mathematical Model

Algorithm for Customer's order:

If the customer wants to dine in, he/she will be able to select that option. Then the customer will be given a list of tables to choose from. After that, a menu will be displayed which the customer will be able to select items to be added to the order list. After the customer finalizes their order, they will be provided with an approximate wait time. If the customer wants to either take out or have to food delivered, the items from the menu will need to be selected to add to the order list. If the delivery option is chosen, the customer also need to enter in the address of delivery. In both of these options, the customer will be given a total wait time.

```
while(phone number is not valid){
   Customer enters his/her phone number
}

if(customer is dining in){
   while(tables are not full)                //customer selects table
                  Customer can select the table and make changes if wanted
         while(items are being selected)      //customer selects menu item
      Chef sees order and order is added to customer's orderList
   if(food is done cooking)                  //food notification
                  Notify customer and remove item from orderList
}
```

```
if(customer is ordering takeout){
   while(items are being selected)
      The selected items will be added to the orderList and sent to the chef
         For the list of items,
      Provides estimated time of preparation and start countdown timer
}
if(customer is ordering delivery){
   while(items are being selected)
      The selected items will be added to the orderList and sent to the chef
         Provide a food preparation + delivery time to the customer
}
```

Algorithm for Employee/Busboy/Chef:

The waitress/busboy will be able to see which tables need cleaning. The cook will receive an updated list of the customers' order lists. And when the food is prepared, the cook will signal the waiter to serve the food to the customer.

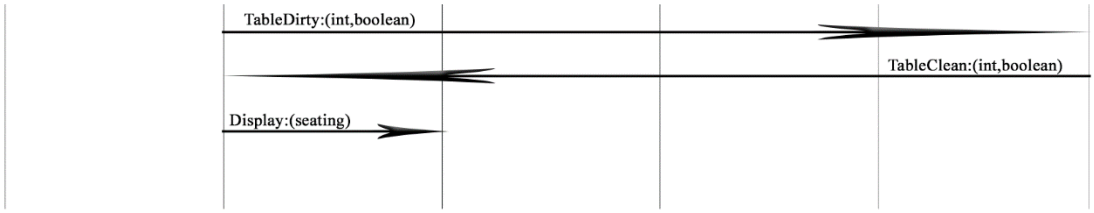
```
while(customers are ordering food){
   if(food is not prepared){
      The cook prepares the food and signals the waiter through the app that the food is done
   }


   if(customer finishes eating and pays for food){
      Waiter cleans the table that the customer was at
   }
}
```

More detailed algorithms in Design of Tests section.

7. Interaction Diagrams:

Sequence Diagrams:



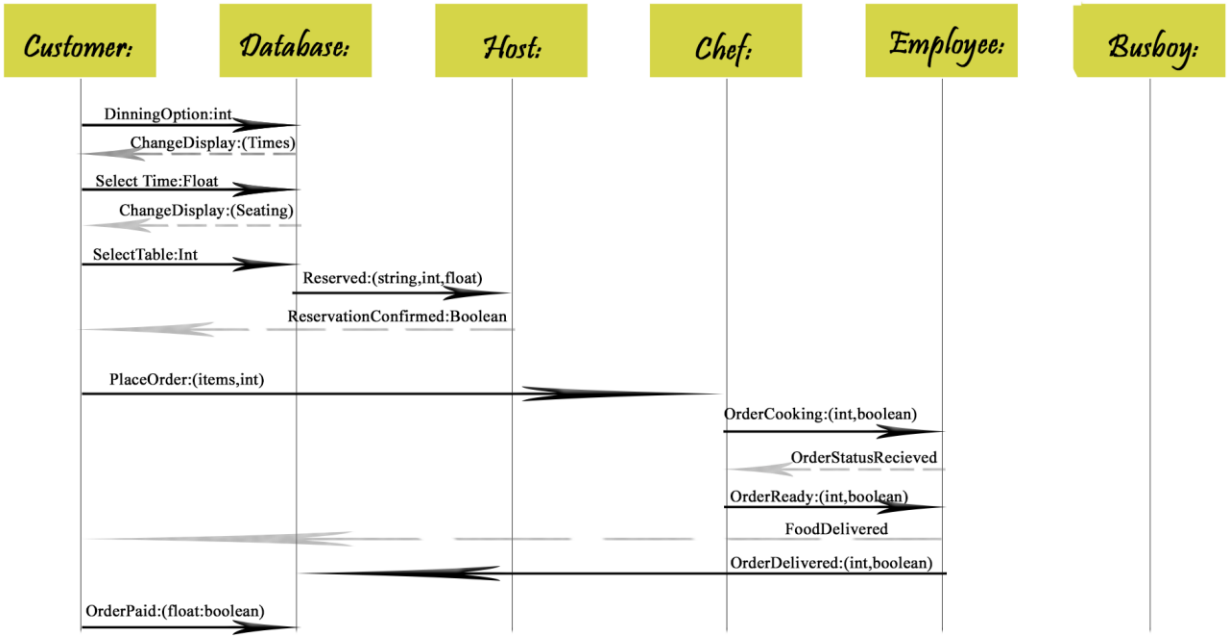


Diagram 1: The above diagram represents the communication throughout all the different users for a typical customer transaction. The customer is the one who sets the stage for the rest of the actors which makes it a perfect example to show the interactions amongst all of the actors.

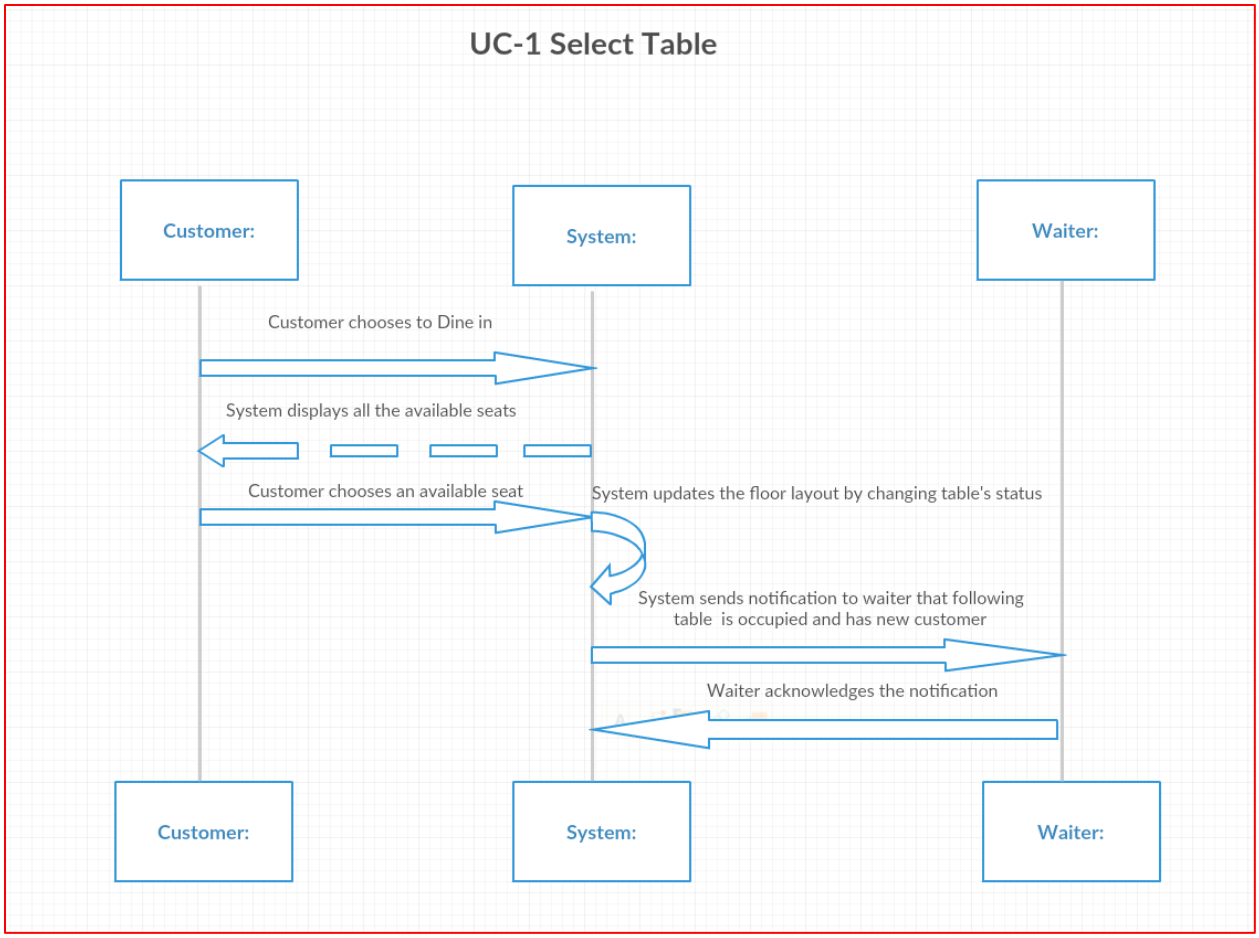


Diagram 2: The above diagram represents the communication through customers and the system when selecting the table. The customer choosing to dine in and then selecting a table leads to the rest of the interactions. The system updates the floor layout once the table is chosen (changes the status of table from “available” to “occupied”). The Host then brings customer to the chosen table and the customer places their order.

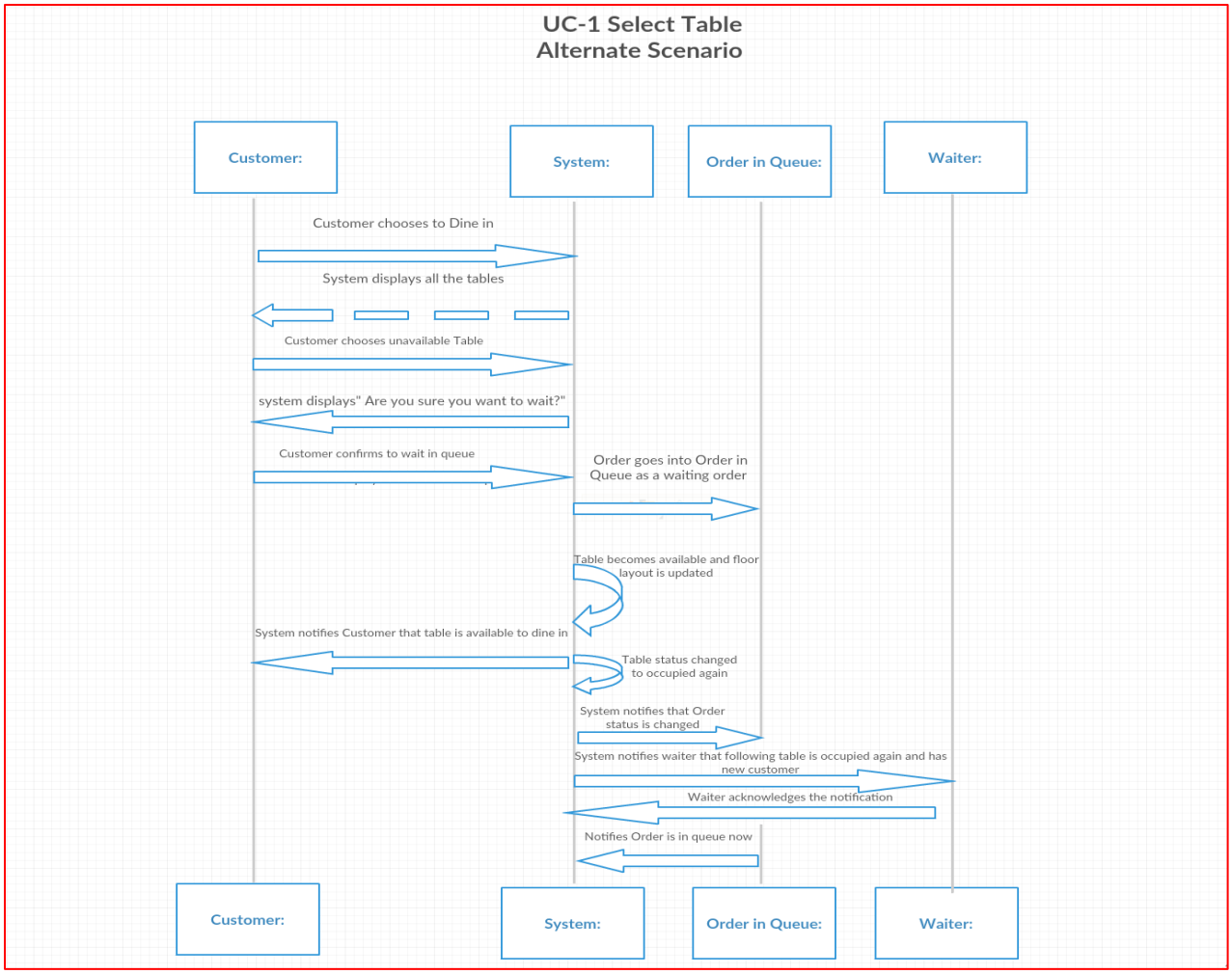


Diagram 3: The above diagram represents an alternate scenario of the user selecting an unoccupied table. There are no tables available for the user to select. So the system gives the user two options: 1) wait for an unoccupied table 2) cancel his/her reservation. If the customer manages to get a table, he/she will able to place an order which will then be placed in a queue (waiting list of orders). At the end of a sequence, the order will go to the chef.

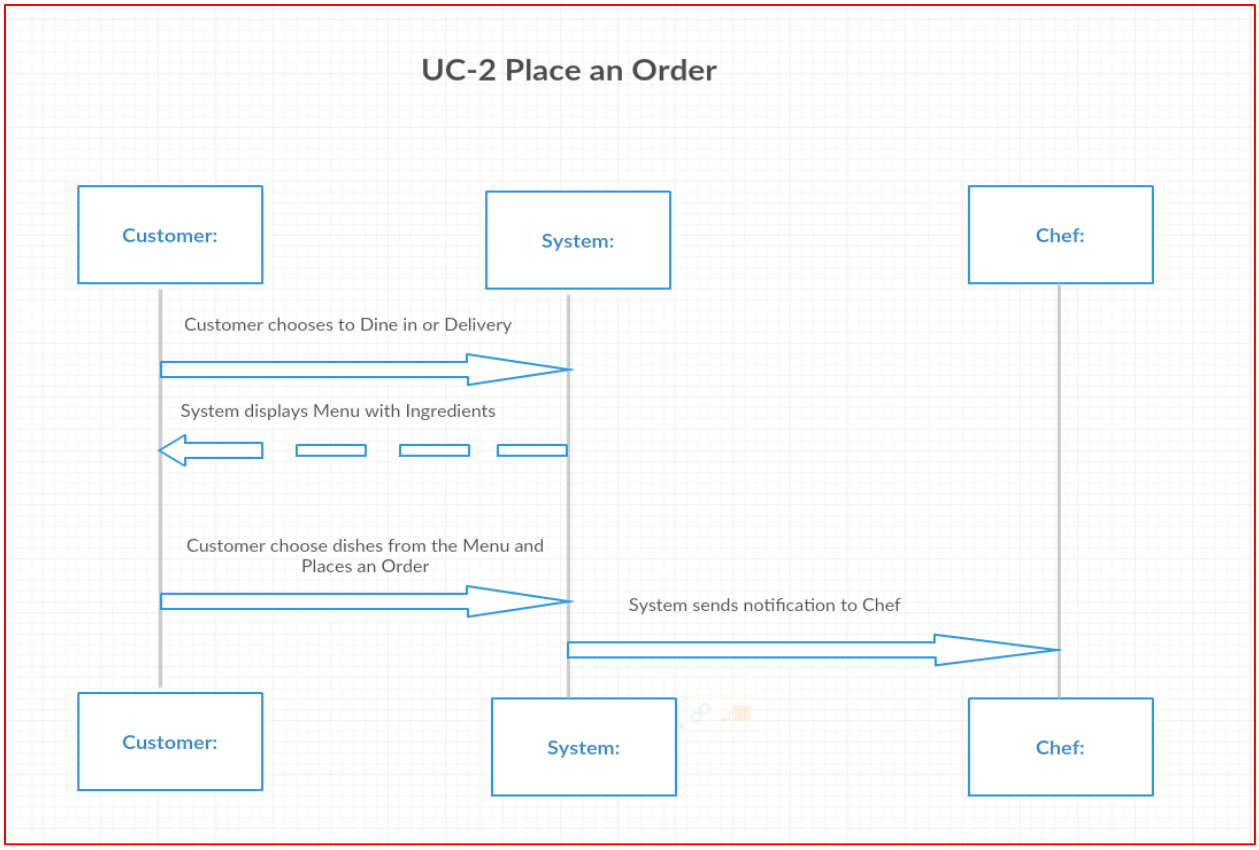


Diagram 4: The above diagram represents the communication through customers and the system when placing an order after being seated at a table. The chef receives the order and begins preparing the order. The chef then finishes preparing the order, and signals the employees that the food is ready. Employee then receives signal and brings prepared order to the customer's table (employee confirms order delivery).

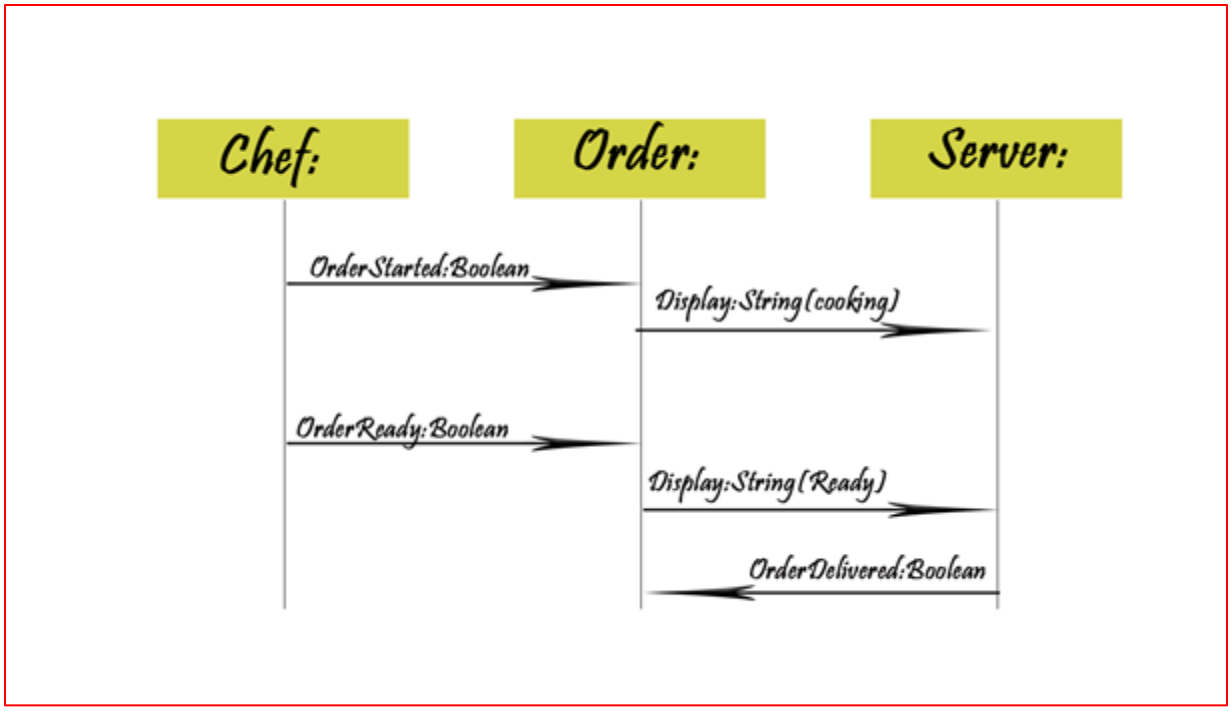


Diagram 5: The above interface shows the simplicity of the Chef's interface. The chef simply tells the system when an order is started which will notify the server accordingly. The chef then finishes preparing the order, and signals the employees that the food is ready for pickup.

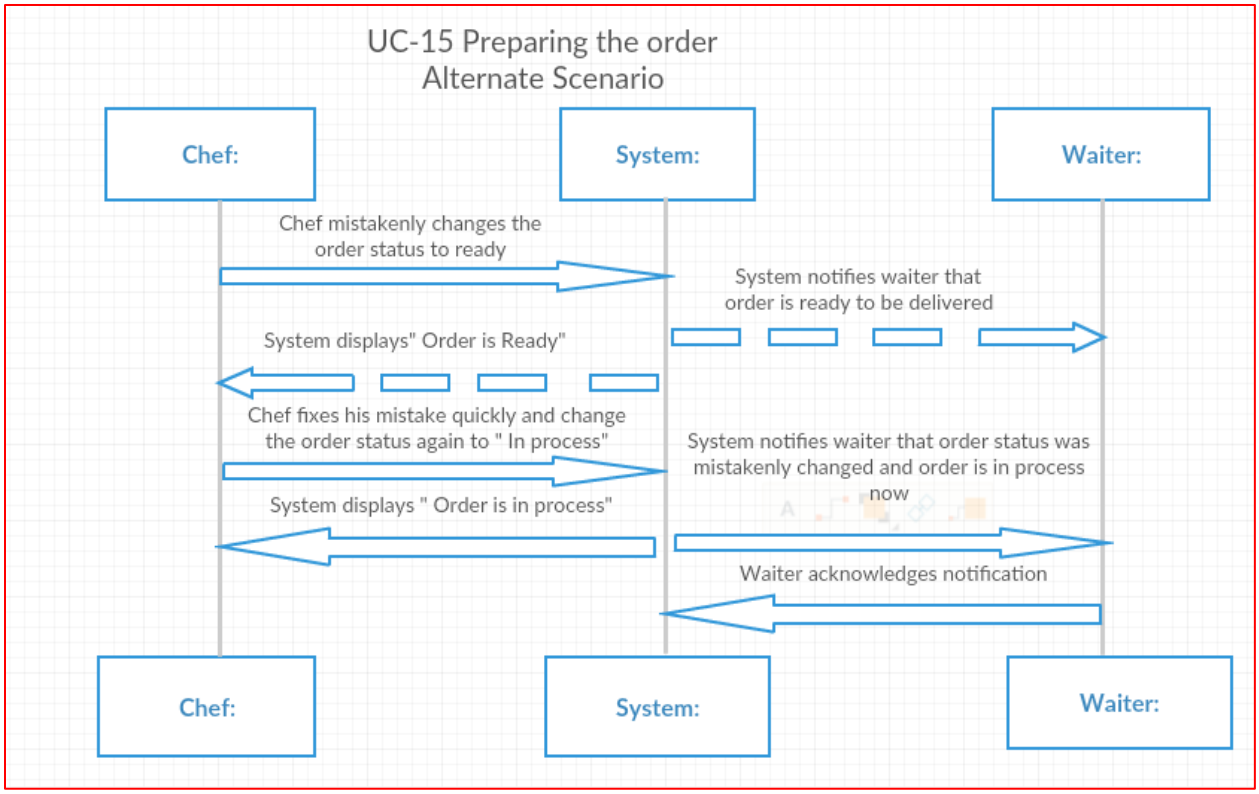


Diagram 6: The above diagram represents an alternate scenario of the chef mistakenly changing the order status from “In Process” to “Ready”. System displays that order is ready. Plus, system notifies Waiter that the order is ready to be delivered. Chef realizes that the order status was changed mistakenly and fixes it quickly on his tablet. System notifies waiter that the order status was changed mistakenly and order is in process again. Waiter acknowledges notification.

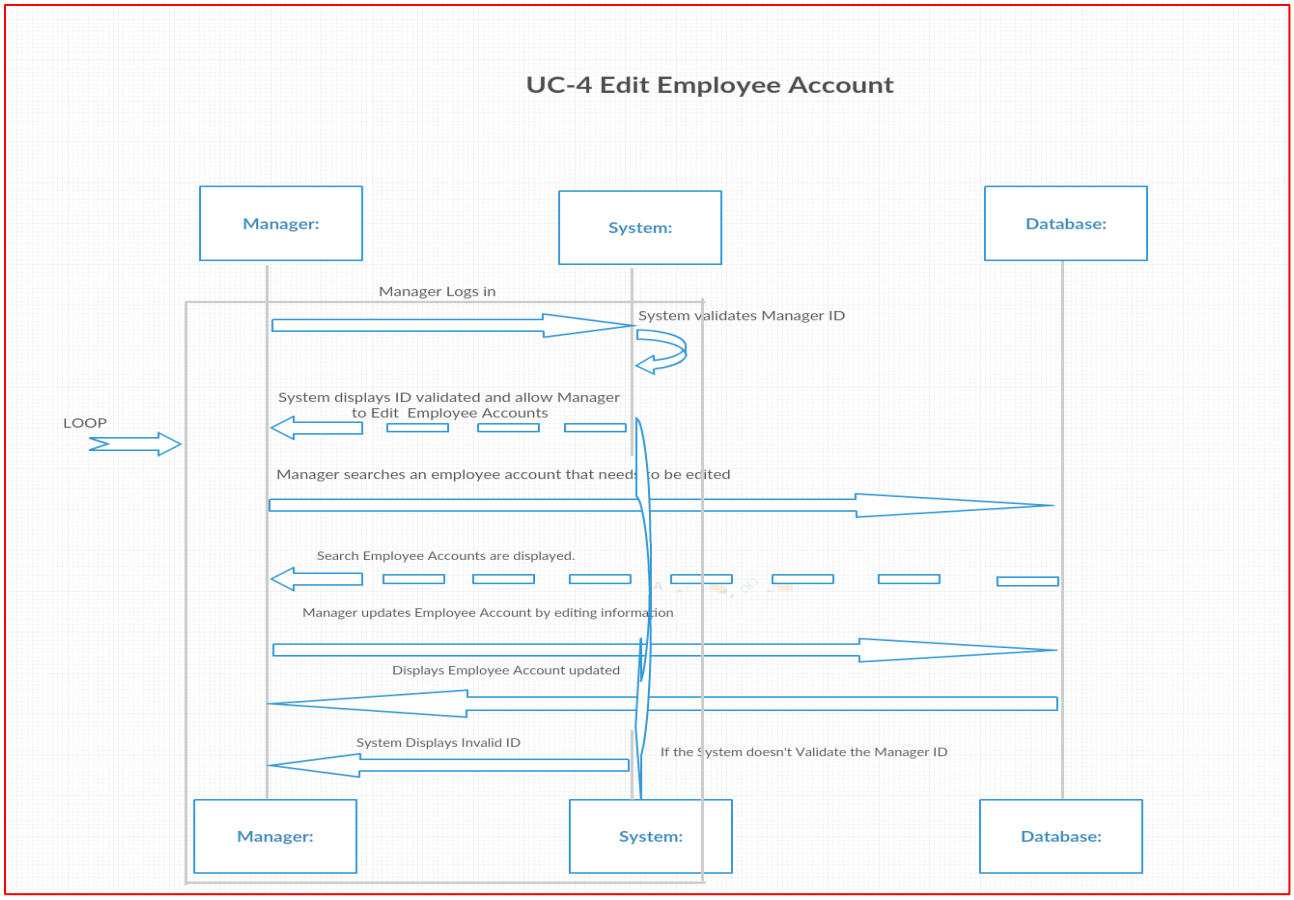


Diagram 7: The above diagram illustrates the manager's ability to edit the information of current employees. The manager logs in and is presented with a list of currently active employees. The manager will also be able to update any logistical data about his/her employees. He will also be able to add or remove employees from the list.

UC 21 - Message Center

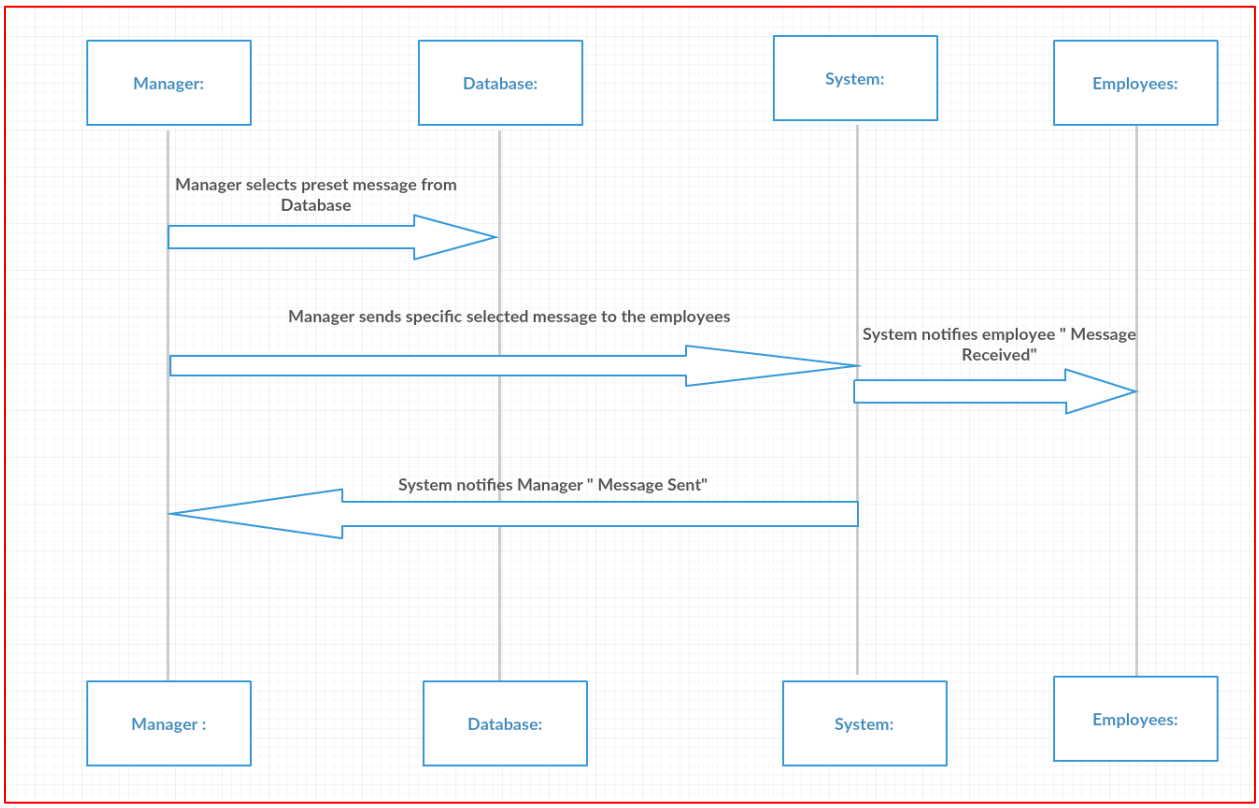


Diagram 8: The manager selects a preset message that is hardcoded to the database. The manager will select which employee to send the message to. Once the message is sent, the manager will be notified that the message was sent. The recipient of the message will be notified they have a message.

UC 12, 24, 25 – User Login

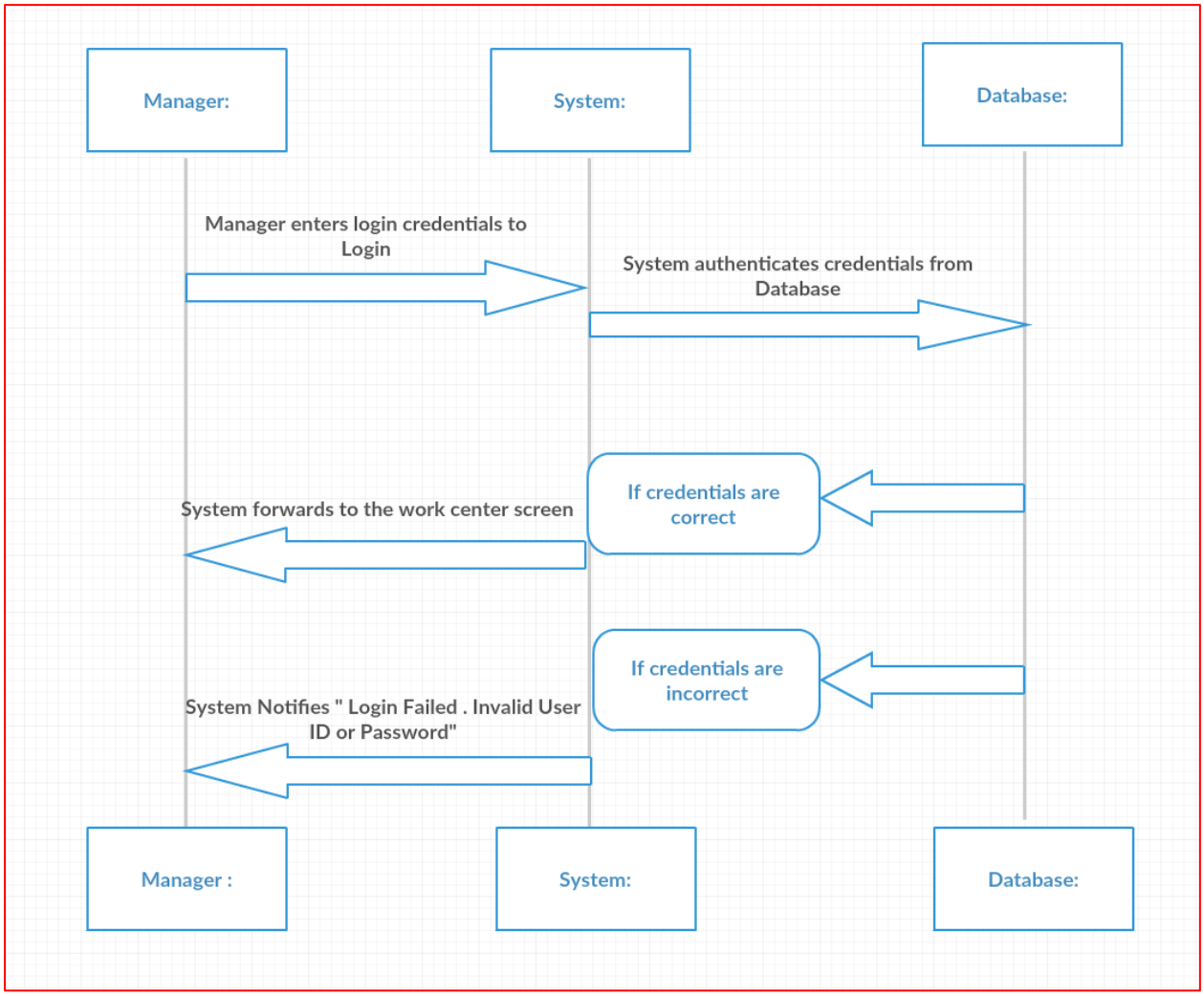


Diagram 9: The manager opens the app and is brought to the login screen. The manager types in their username and password and then clicks login. The system checks the inputted information in the database. If the entered information is not found in the database “Invalid Login” will be displayed to the user and they will be returned to the login page. If the information is correct then the Manager will be brought to the main Manager screen.

8. Class Diagrams and Interface Specification:

8.1) Class Diagram

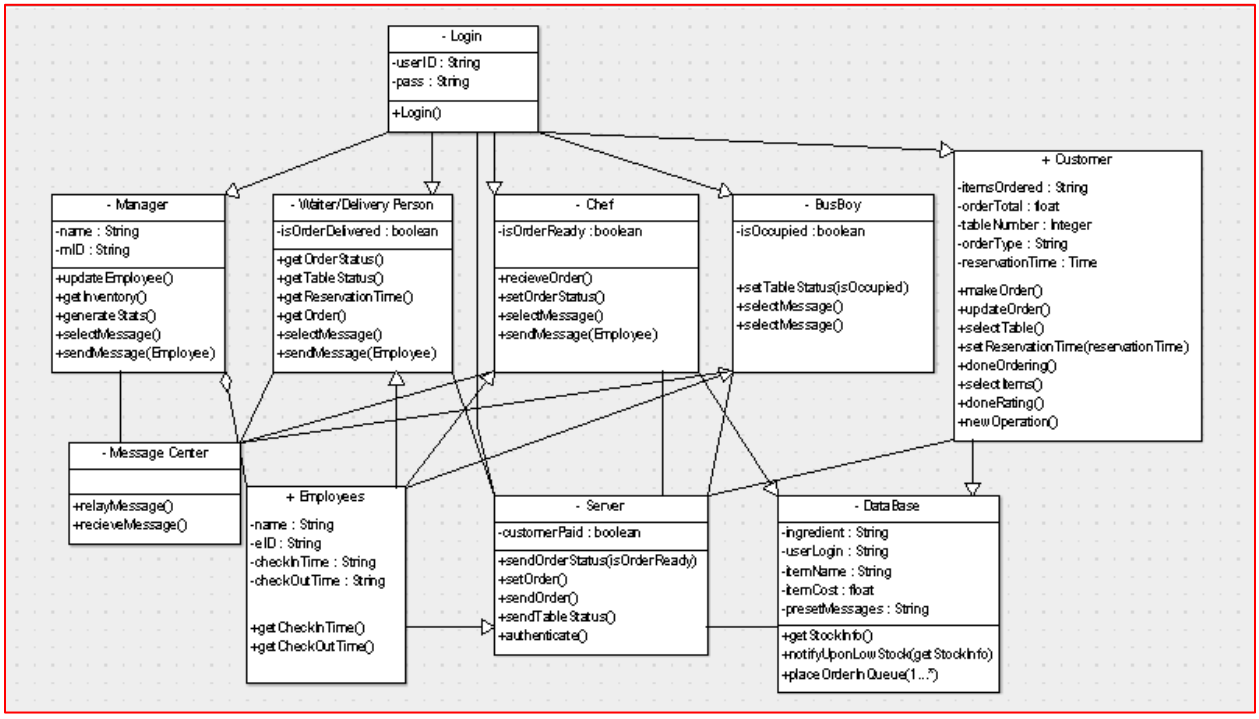


Figure: Complete Class Diagram

Class diagram depicts the various “actors” in our system, and how they interact with each other. Minus sign indicates private visibility, while plus sign indicates public visibility. Manager, waiters/delivery person, busboy, employees, and customer all constitute client-side, while the server relays information to/gets information from client and database.

The above class diagram illustrates how all the classes connect to each other and their inner workings. Upon opening the application, the user will be prompted to login in with his/her username and password. The Login Activity will contact the server to authenticate() the login to add an extra layer of security. All of the logins of the users will be stored in the database to ensure that the user logging in exists. From there, the user will be greeted upon his/her appropriate activity screen and can perform whatever actions may be required. For instance, any of the employees would be able to send preset messages to each other, which gets relayed to the other user(s) via the message center. Those preset messages are hard-coded in the database for ease of access, which would also reduce the Employee’s chatting time (no time would be wasted). Most importantly, any change made in the classes by the actors (e.g. Customer selected an order) will be reflected upon in real-time in the database.

8.2) Data Types and Operation Signatures

- Login:
  - Attributes:
    - -userID: A string containing the user's email used to login
    - -pass: A string containing the user's password used to login
  - Operations:
    - Login(): This method will log the user in and direct them to the appropriate screen.
- Message Center:
  - Operations
    - receiveMessage(): This method will receive the message from the user
    - relayMessage(): This method will send the message that was received from the user to the appropriate user.

- Class - Manager
  - Attributes:
    - -name: 'name' is stored in a private String, containing the personal name of the user of this class.
    - -mID: 'mID' is stored in a private String, containing the manager identification number of the user of this class.
  - Operations:
    - -updateEmployee(): This operation serves as a way for the Manager class to update the information in the Employee class concerning anything about the employee - such as name, SSN, payroll, etc.
    - -getInventory(): This operation serves as a way for the Manager class to obtain information on the current status of restaurant's inventory. This should bring up the data relating to inventory and present it nicely in tables/charts.
- Class - Employee
  - Attributes:
    - -name: 'name' is stored in a private String, containing the personal name of the user of this class.
    - -eID: 'eID' is stored in a private String, containing the employee identification number of the user of this class.
    - -checkInTime: 'checkInTime' is stored in a private String, containing the employee's time of checking into their work shift.
    - -checkOutTime: 'checkOutTime' is stored in a private String, containing the employee's time of checking into their work shift.
  - Operations:

        - -getCheckInTime(): This operation serves as a way for the Manager class to obtain information on the Employee's time of checking in.
        - -getCheckOutTime(): This operation serves as a way for the Manager class to obtain information on the Employee's time of checking out.
- Class - Waiter/Delivery Person
    - Attributes:
        - +isOrderDelivered: The attribute is a boolean that will be used to indicate whether or not an order has been delivered to a table.
    - Operations:
        - +getOrderStatus(): This operation serves as a way for the Waiter class to obtain the value of the isOrderDelivered attribute.
        - +getTableStatus(): This operation serves as a way for the Waiter class to obtain information on the current status of a table.
        - +getReservationTime(): This operation serves as a way for the Waiter class to obtain information regarding the time a reservation was made by the Customer class.
        - +getOrder(): This operation serves as a way for the Waiter class to obtain information regarding the meal that was ordered from Customer class.
- Class - Chef
    - Attributes:
        - +isOrderReady: The attribute is a boolean that will be used to indicate whether or not an order has been fulfilled.
    - Operations:
        - -setOrderStatus(): Sets the boolean isOrderReady to 'true', indicating an order has been fulfilled.
- Class - Server
    - Attributes:
        - +customerPaid: The attribute is a boolean that will be used to indicate whether or not an order has been paid for by the customer.
    - Operations:
        - +sendOrderStatus(isOrderReady): Sends the order status to the Waiter class.
        - +setOrder(): Uses the information gathered from the Customer class to pass along the order as well as dining/reservation time/status.
        - +sendOrder(): After the order has been set, it is then sent to the Chef class.
        - +sendTableStatus(): Gathers the required data on table implicitly specified, and then sends it out to the requester.
- Class - BusBoy
    - Attributes:
        - +isOccupied: A boolean indicating whether or not a table is still occupied, so that the BusBoy actor knows when to clear off the table.
    - Operations:
        - +setTableStatus(isOccupied): Updates the status of to indicate whether or not a table is available to seat occupants.
- Class - Customer
    - Attributes:

        - +itemsOrdered: A string containing the items ordered by the Customer actor, to be later passed by the operation makeOrder.
        - +orderTotal: A float representing the total cost of the order, to be displayed at the end of the Customer actor's order.
        - +tableNumber: An integer that keeps track of the table selected by the Customer actor.
        - +orderType: A string that holds the type of dining service the Customer actor wants (Dine-in, Take-out, Delivery).
        - +reservationTime: If the Customer actor wishes to make a reservation, the reservation time is stored in this attribute.
    - Operations:
        - -makeOrder(): Uses all of the Customer attributes to create an order that is to be passed on the restaurant staff classes.
        - +updateOrder(): Updates any aspect needed of the Customer's order.
        - -selectTable(): Attempts to select the table requested by the Customer using tableNumber.
        - -setReservationTime(reservationTime): Sets the reservation time on the staff side of this with the reservationTime specified by the Customer.
- Class - DataBase
    - Attributes:
        - #ingredient: A string that holds the list of ingredients of a certain item on the menu.
        - #userLogin: A string that holds the user's login, to be compared against when a user attempts to log in.
        - #itemName: A string that holds the name for an item located on the menu.
        - #itemCost: A float that will hold the cost of an item off of the menu.
    - Operations:
        - #getStockInfo(): Sends the inventory stock of an item when requested by the appropriate class.
        - #notifyUponLowStock(getStockInfo): Allows the appropriate class to be notified when a certain item has a low number in stock.

8.3) Traceability Matrix

concept#	Manager	Employee	Waiter/Delivery Person	Chef	Server	BusBoy	Customer	dataBase
1			x		x			
2					x		x	
3	x		x	x			x	x
4	x				x			x
5		x			x			x
6	x	x			x			x
7				x	x			
8			x			x		x
9			x		x		x	x
10							x	x
11	x							x
12	x	x			x			
13			x		x			
14							x	x
15			x	x	x			
16					x			
17	x							x

Each of the concept numbers to the left of the matrix (#1-17), described in detail in the corresponding section within Report #1, map to one or more of the classes found in the class diagram in this document. For example, the first concept states “Populating the reservation list with customers that have already ordered and selected a seat”. We’d need to keep track of customers as they makeOrders()*. That information would then be sent to the server and then relayed to the waitress. Similar mapping can be done for the other concepts.

*see description under either class diagram “operations” or “list of methods and test inputs”

8.4) Design Patterns

The notification system uses the publisher-subscriber pattern to deliver notifications to all of the appropriate parties. When a notification is generated and sent to notificationHandler, the handler can parse the type of notification, its content, and its source and determine which interfaces need to receive the notification. It then publishes the notification out to those interfaces. For example, when the system detects that the stock level of an item in the inventory is empty, it would pass the name of the item whose stock has run out. It reads the input information and constructs the full notification message, then updates the rest of the interfaces. The notificationHandler will know each notification type and choose which subscribers should be notified. While notifications could have been implemented using direct communication, using the publisher subscriber pattern makes the system easier to maintain and update. If another user interface must be added for a new employee class, it is simple to include it in the subscribers list and include it in the notification system.

8.5) Object Constraint Language

Important contracts (invariants, preconditions, postconditions) for classes and their operations.

waiterInterface:placeOrder() : bool
//The Waiter is the one placing the order
Invariants: self.Waiter -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Succesful and pushed into Queue
Post-Condtions: placeOrder == true

waiterInterface:cancelOrder(orderinfo* o) : bool
//The Waiter is the one placing the order
Invariants: self.Waiter -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Deleted and removed from Queue
Post-Condtions: cancelOrder(orderinfo* o) == true

busboyInterface:cleantable() : bool
//The busboy is the one cleaning the table
Invariants: self.busboy -> True
//The table is dirty on it and not occupied
Pre-Conditions: table.occupied -> False
table.dirty -> True
//Table Cleaned and updates table status
Post-Conditions: cleantable == true

chefInterface::addToMenu(string itemName, double Price): bool
//The chef is the one updating the menu
Invariants: self.chef -> True
//Item is not available on Menu
Pre-Conditions: MenuItem == NULL
//Update item in menu
Post-Conditions: addToMenu(string itemName, double Price) == true

chefInterface::removeFromMenu(itemInfo* i): bool
//The chef is the one updating the menu
Invariants: self.chef -> True
//Item is available on Menu
Pre-Conditions: MenuItem != NULL
//Removing Item successful from menu
Post-Conditions: removeFromMenu(itemInfo* i) == true

managerInterface::fire(eInfo* e): bool
//The manager is the one firing the employee
Invariants: self.manager -> True
//employee is in database pre: employeeinfo != NULL
//Removing employee succesful from database post: fire(eInfo* e) == true

managerInterface::hire(eInfo* e): bool
//The manager is the one hiring the employee
Invariants: self.manager -> True
//employee is not in database
Pre-Conditions: employeeinfo == NULL
//Adding employee successful from database
Post-Conditions: hire(eInfo* e) == true

managerInterface::addToMenu(string itemName, double Price): bool
//The manager is the one updating the menu
Invariant: self.manager -> True
//Item is not available on Menu
Pre-Conditions: MenuItem == NULL
//Update item in menu
Post: addToMenu(string itemName, double Price) == true

managerInterface::removeFromMenu(itemInfo* i): bool
//The manager is the one updating the menu
Invariant: self.manager -> True
//Item is available on Menu
Pre-Conditions: MenuItem != NULL
//Removing Item succesful from menu
Post-Conditions: removeFromMenu(itemInfo* i) == true

managerInterface:placeOrder() : bool
//The manager is the one placing the order
Invariants: self.manager -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Succesful and pushed into Queue
Post-Condtions: placeOrder == true

managerInterface:cancelOrder(orderinfo* o) : bool
//The manager is the one placing the order
Invariants: self.manager -> True
//The table has customers on it
Pre-Conditions: table.occupied -> True
//Order Deleted and removed from Queue
Post-Condtions: cancelOrder(orderinfo* o) == true

9. System Architecture and System Design:

9.1) Architectural Styles

The function of a system architecture is to provide mechanisms and an abstraction of the underlying machine. By grouping our architectural styles into specific categories, we can more logically express the general schema of the project as well as going into some depth about how each portion is implemented. The general schema are as follows, with the specific nature detailed underneath:

1.) Communication - This category includes the methods used to communicate between the client and server, as well as client-client based communication.

- Client/Server - "Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures." - Microsoft Developer Network (Reference 10).

  a.) As explained below in further detail, the server acts as a mediator between the client and database. The server will take in any necessary information and convert that information into data necessary for the use of clients.

- Service-Oriented Architecture - "Refers to applications that expose and consume functionality as a service using contracts and messages." - Microsoft Developer Network (Reference 10).

  a.) The server will act as the interpreter between multiple clients. The communication methods between multiple clients will be on a click based system in order to efficiently get messages across, that way the time to convey the message is much faster for clients and the communication time between the clients will be minimized.

  b.) The server will manage any data pertaining to the user. That data will be combined for the client's needs in a statistical format. By combining the needed information, the client will be able to more easily use the data for any of the client's needs. The data will be collected over a period of time, and the server will also keep track of the time which pertains to a specific data.

- Message Bus - "An architecture style that prescribes use of a software system that can receive and send messages using one or more communication channels, so that applications can interact without needing to know specific details about each other." - Microsoft Developer Network (Reference 10).

a. For a certain event, the client will send messages only through one communication channel (which is the click-based system as mentioned above). Because of the clicked-based communication system, the clients do not need to be aware of who receives this message. The client must only be aware of the functionalities of the application.

2.) Structure - This category is based around the forms used to store, send and retrieve data.

- Component-Based - "Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces." - Microsoft Developer Network (Reference 10).

  a. List: When an item on the menu is selected, that item will be automatically placed in the order list. When the order is complete, the order list will be sent to the appropriate recipient. At the same time, the reservation list is also populated with the client's information.

- Layered Architecture - "Partitions the concerns of the application into stacked groups (layers)." - Microsoft Developer Network (Reference 10).

  a. When a user is displayed a list of objects, those objects will lead to instructions being sent to the server, possibly along with information. These instructions are then parsed by the server, which are then is executed by the database - taking into account any possible data that was transmitted alongside the instructions. Thus, this project follows a layered architecture, with the user interface being the first tier, the server parsing commands being the second, and the database's retrieval/relinquishment of data being the third.

9.2) Identifying Subsystems

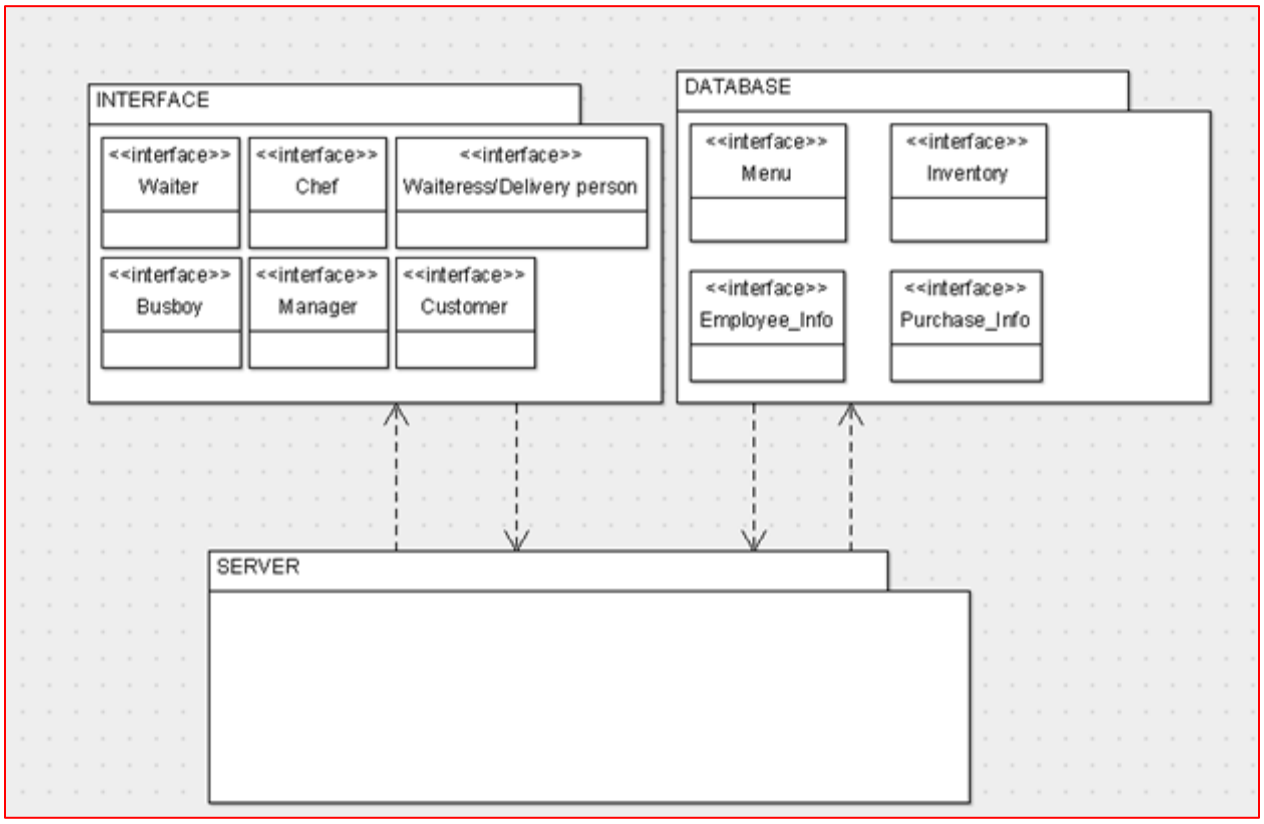


Figure: UML Package Diagram

This diagram depicts our subsystem into a “component diagram” that creates singular objects that work together to make our subsystem visually easier to understand. As can be seen, the server package is what will be communicating with both the database and the people. It can receive information as well as relay information when needed. The database package will hold all employee & user data, every purchase made, menu items, and inventory. The manager will be able to use the database to edit employee information as well as keep tabs on the inventory. The interface package will host the main graphical user interface (GUI) and will display all relevant information such as menu items, table availability, order status, etc. It will act as the “client”. Of course, Customer, Waitress, etc. are all sub-packages that fall under the Interface super-package.

9.3) Mapping Subsystems to Hardware

This system will run on different machines. The subsystems are the following: database, server, and the client. The back-end of the system consists of the database and the server, and the front-end is the client. The database will be MongoDB, which will run on either a desktop or a portable computer, such as a laptop. The server will ideally run of the same machine as the database. If the server and the database runs of different machines, then it is required that they are under the same connection. The client side of the system will run on a mobile device, which must also be under the same connection as the database and the server. All subsystems are required to be under the same connection because in order for the system to run, they must be able to communicate with each other. Each instance of the client (or each user interface) will be on different mobile devices, and all the mobile devices will communicate with the back-end of the system.

9.4) Persistent Data Storage

For our system, all of the data will be stored permanently. This is possible because the instance of the server and the database will be running continuously. MongoDB regularly writes to a file. Even if the database instance is terminated, the data will still be saved. The persistent objects that will need to be stored are the rating system, customers' order items, and the menu. The storage will be managed using a non-relational database (MongoDB). This is because the data can be accessed more easily. Non-relational databases are faster than relational databases, and it is more compatible with Node JS (the language that the server is written in).

The following is the MongoDB database schema:

```
Customer :: {
        name:String,
        username:String
        password:String
        orders:[Order]
    }

Order :: {
          customer:Customer
          items:[Item]
          date:Date
   }

Item :: {
          name:String
          quantity:Number
          price:Number
   }
```

```
Menu :: {
        items:[Item]
   }
```

Every schema will hold the data for different parts of the system. Objects of one schema are attributes of another schema. For example, a customer has a list of orders. Customer and Order are two different schemas. The history of the orders will be placed under the Order Schema. In order for the chef's side of the application to access the client's (customer's) current order, the system must access the most recent order, which is listed by date.

9.5) Network Protocol

This system runs on multiple machines and requires a communication protocol. The communication protocol being used is HTTP. The reason for this is that it is the most widely used protocol, and allows multiple platform experience. Because of the fact that it is widely used, there are many documentations for it. The API service being used is REST. REST services provides a specific pattern on which the HTTP communication protocol is being used. REST service is a stateless web service tool; therefore, it will not depend on any of the old requests.

9.6) Global Control Flow

Executive orderness: The system, SpeedByte, executes is procedure driven. The application will be executed in a linear fashion, and the user will generally have to go through the same step every time he/she is using the system.

Time dependency: There are timers in the system. The first timer is a countdown to when food will be done. The second timer is also a countdown to when the seat is reserved for dine-in. Both of these times are event responsive.

Concurrency: Node JS (specifically Express JS) uses a different thread per HTTP request. It uses the handler as the function for the thread of control. Synchronization is enforced through by the database, MongoDB, because there is a level of mutual exclusion that occurs when the data is manipulated.

9.7) Hardware Requirements

This system depends on certain hardware requirements. It requires a server that will act as messenger between the database side and the client side of the system.

Listed below is the specifications of the systems:

Mobile:

API	23 (Android Marshmallow 6.0.1)
Device	SAMSUNG Galaxy S6
RAM	4 GB
Storage	32 GB

Server:

RAM	4 GB
Hard drive	1 GB (minimum)
Network Card	66 Kb/s

10. Algorithms and Data Structures:

10.1) Algorithms

Customer: The customer has the option of choosing if s/he wants to dine in, order takeout, or order delivery. Next, the customer will choose the items from the menu. To maximize efficiency, items will be added into an expandable data structure as they are selected by the customer. If the customer unchecks an item, then that item will be deleted from that data structure. The expected big O notation for this algorithm is O(n) time and O(n) space. If the customer chooses to dine in, he/she will select a table and choose the time that s/he wants to eat. A simple algorithm is needed in order to accomplish this. When it gets closer to the scheduled time, the customer will get alerted. The database will contain information regarding the tables (number of people for each table), menu, and a temporary storage of customer's information. The menu items order will be stored permanently, and a graph representation will be made for the manager's use. The customer will also be given an option to take a survey of the restaurant. When the customer is done with the survey, the data will be stored into the database in the order that it is presented. The big O notation for this algorithm is O(1). No data structure will be required in order to implement this. This algorithm is yet to be implemented.

Chef: The chef will be able to access orders that the customers placed. The customer's orders will be placed in a data structure, which the chef will be able to access. When the food is cooked, it will be deleted from the data structure. The algorithm will be implemented by the use of an infinite loop, where items will be constantly added to the data structure and when a particular item is prepared, that item will be removed from the data structure. The worst case for this algorithm is expected to be O(n*n). This algorithm is also yet to be implemented.

Manager: The manager side of the application will have one of the simplest algorithms. The statistics of various information will need to be presented in the format of a graph. In order for this to happen, the database will be queried for the needed information. Accessing the particular table will only require an O(1) time. However, extracting the information and placing it into the required data structure will be expected to be an O(n) time. Therefore the total time required is O(n). The algorithm is also yet to be implemented.

10.2) Data Structures

The data structures used in this application depend on various factors. These factors depend on the situation (problem to be solved), efficiency (how long the process takes and how easy it is to access and store information), and adaptability with the application (how well it fits in with the functionality of the application).

Customer: When the customer selects items of the menu, an expandable data structure is required to store the information. An array list is the best data structure in this case. This is because accessing an item in the array list is an O(1) operation, and adding an item to an array list is also an O(1) operation. Therefore, this is the most efficient data structure to use.

Chef: The chef's side of the application will require a data structure that will allow easy access of the items. A queue is the data structure which will fit perfectly into this. The items will be put in order of the customers' orders which will be added to the queue appropriately.

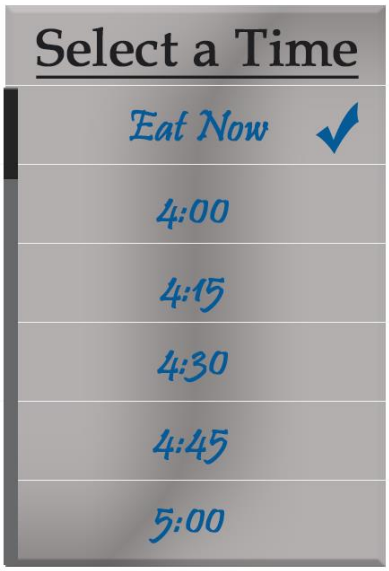
Manager: The manager side of the application will only be presented with graphs and tables of the statistics. A linked list will be used in this case because a linked list can be easily expanded upon. A linked list is inefficient regarding search; however, the search is not necessary when making a graph.

11. User Interface Design and Implementation:

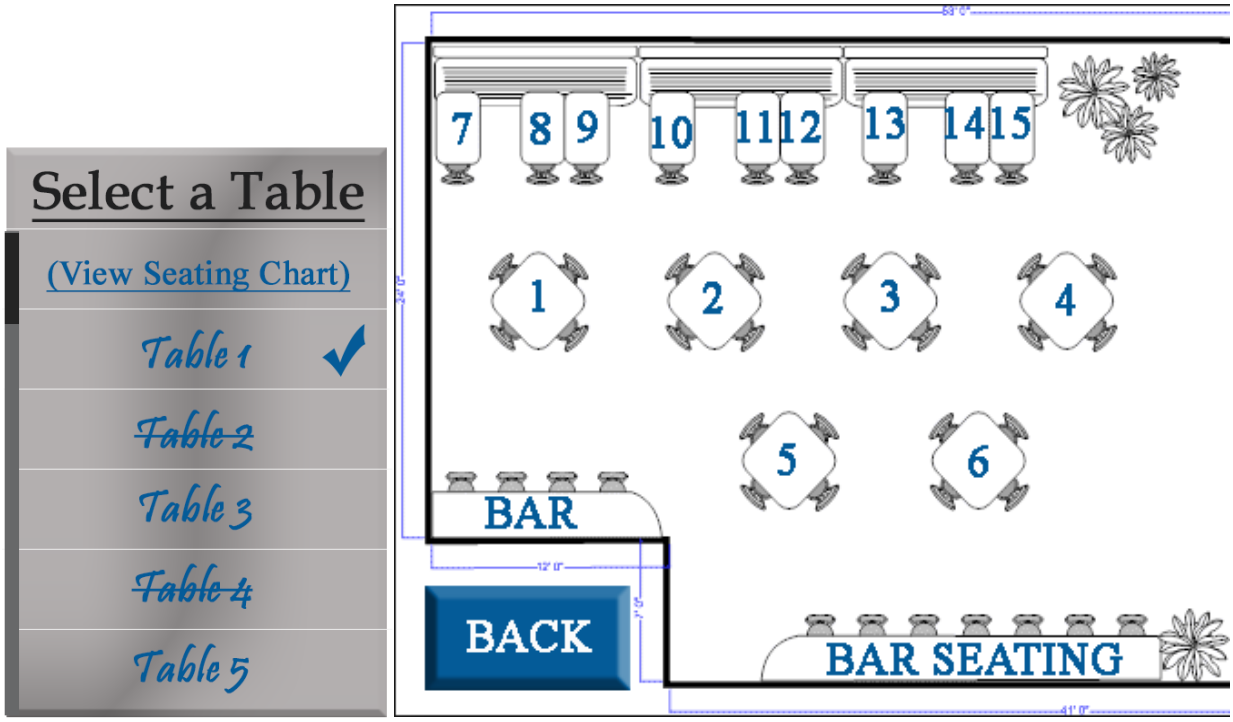
Customer Interface

The Customer is first asked to select a one of three dining options, Dine In, Takeout, and Delivery. If Takeout or Delivery are chosen the user is brought directly to the ***Menu Page***. On the other hand if they choose to dine in they are directed to the ***Reservation Page***.

If the Customer selects Dine in they are brought to the ***Reservation page*** which is an interactive scroll menu that is set in fifteen minute intervals. Once the user selects a time they a directed to the ***Table Selection page***.



After selecting a time the user is brought to the table selection page, this is an interactive scroll menu listing all the tables in the restaurant. If a table is crossed out that means it is unavailable at the selected time. If the customer is not a regular and wishes to see a seating plan of the restaurant they may click the “View Seating Chart” button which will bring them to a layout of the restaurant, otherwise they may simply click on the corresponding table number. The table will then be reserved for the corresponding time.



Reference for Image: http://www.restaurantfurniture.net/restaurant-design/

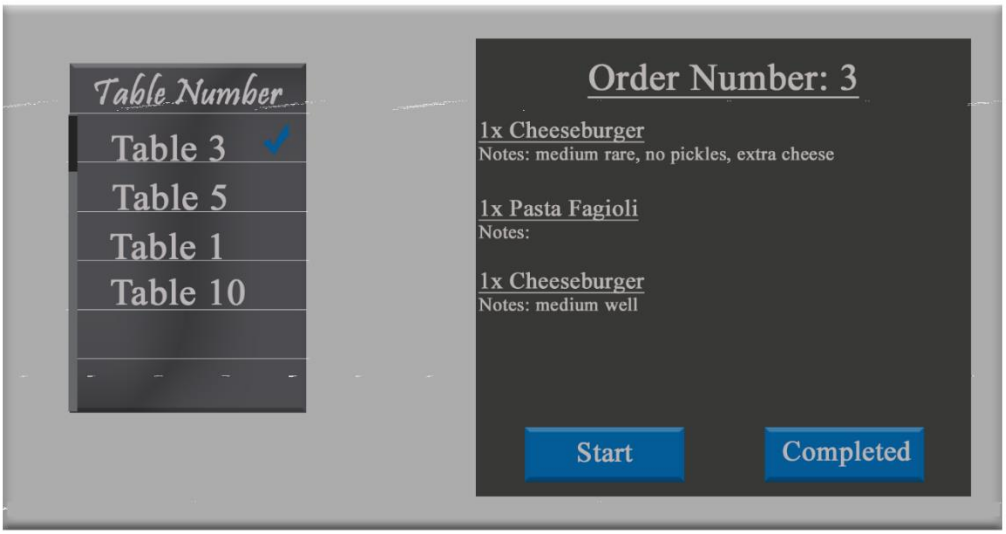
Once the Host brings the Customer to the table they will be brought to the menu page. The menu Page will be yet another scroll menu listing each of the menu items, each item will have a rating, a wait time, and an adjustable quantity. If the customer wishes to get more information on the item they may click on the item which will bring them to a description page. The user will also have an option to create notes to the chef specifying any allergies or request they may have. Once the customer is satisfied with their order they may hit place order, which will send their order to the chef.

Cheeseburger menu item: https://millersalehouse.com/menu/

After paying, the customer will be brought to a ratings page where, if they chose, can provide valuable feedback on each of the ordered dishes by simply clicking on the corresponding number of stars they wish to rate the dish.

Chef Interface

The Chefs Interface will always remain on the same page but will be updated accordingly. The ***Table Number*** display shows the tables orders in the order they were received. The chef has the option to choose a table, once the table is selected the order will be displayed on the right portion of the screen. The chef clicks ***start*** once the order begins cooking which notifies the server. Once the order is complete the Chef clicks ***completed*** which removes the completed table off of the list and notifies the server to pick up the food.

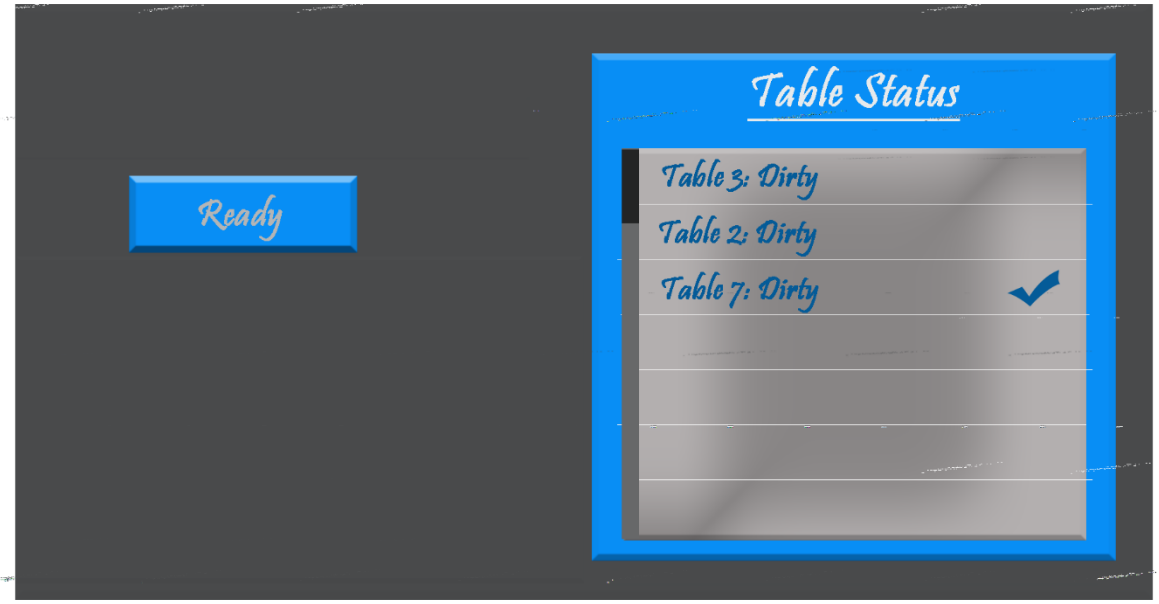


Server Interface

The Server Interface is one of the simpler displays. The server ***Order Status*** is derived from the ***Chef Page.*** When the chef starts cooking an order it appears on the ***Order Status*** as "Cooking" this acts as a warning to the server that an order will be done soon. Once the Chef completes the order the Order Status changes from *Cooking* to *Ready.* The Server then delivers the food to the customer, once the food is delivered the status switches to waiting for payment. Once the Customer pays their bill the server clicks paid which removes the table from the list and sends a message to the busboy.

Busboy Interface

Once the ***Busboy*** receives the message from the Server that a customer has paid, the corresponding table appears on the ***Table Status*** section, notifying the Busboy to clean the table. After cleaning the table the Busboy simply selects the table that was cleaned and clicks the ready button. This will remove the table from the list and set the table as available to the Host and Customers.



Host Interface

The Host will have the option to select a specific time to view the status of those tables. This is done by simply clicking on the specified time on the interactive scroll menu.

After selecting a time the Host will be brought to the ***Table Availability Page.*** This page shows the current status of each table. Green tables mean the table is available. Red tables mean that the corresponding table is occupied, and Blue tables mean they are reserved but the person has not been seated yet. If the Host chooses to seat a customer they will simply click on the corresponding table, this will turn the table to the occupied status.

Management Interface

The Opening Menu for the manager has three options ***Edit Item, Edit Employee,*** and ***Statistics.*** If the manager chooses to edit an item or employee they will be directed to an information page, if the statistics option is chosen the manager will be brought to ***Statistics Selection Page.***

Add/Delete Item

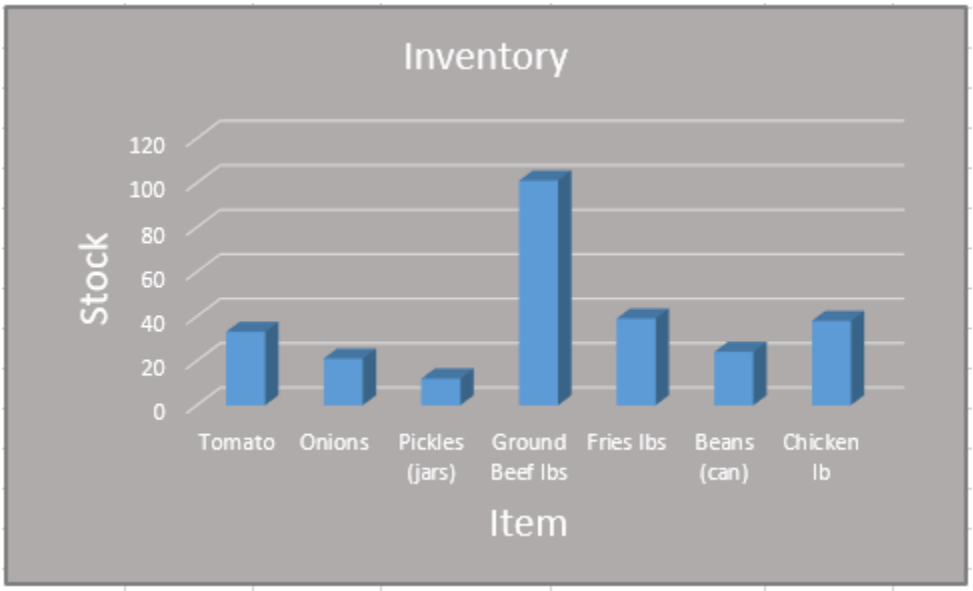
The Manager simply writes in the item's information and clicks ***Add*** or ***Delete***, this will add or delete the corresponding item to the system.

Add/Delete Employee

The Manager simply writes in the employee's information and clicks ***Add*** or ***Delete***, this will add or delete the corresponding employee to the system.

If the manager selects ***Statistics*** they are brought to the ***Statistics Selection Page,*** here they have the option to select what statistics they would like to view.

Inventory



Traffic

Profits

Message Center

SpeedByte has an integrated messaging system that will display notifications and messages. The messages will be preprogrammed which will create a quick and easy way to communicate important information. Having the messages preprogrammed will also prevent improper use of the messaging software. Each of the user main screens will have a button to access this feature. Employees will have more capabilities such as sending messages compared to customers who will only receive notifications. Notifications can be used to notify customers if an item is out of stock, if the restaurant will be closed on a specific day, or any other messages that the manager wishes to set.

Sending a message is very simple, the user will be able to simply select who the recipients of the message will be from a scroll down menu (Customers, Employees, Manager, or All). After selecting who the message will be sent to the user selects which message will be sent from another scroll down menu. Lastly, the user will click confirm which will send the message to all the designated recipients.

Message Sending Screen

Notifications Screen

Additional Navigation Notes

In any of the interfaces if the user wishes to go back to a previous page they can simply click the back button that comes standard with every android device, this will bring them to the previous page.

The Design has come a long way from the initial stages of development. We have stream lined the design, the design guides the user through the process which reduces the confusion of most apps. This also keeps the user on task and moving along which results in a faster transaction time which will benefit customers and employees alike.

The new design rewards regular customers with a faster interface but at the same time has options to make it easier for first time users.

The Overall interface is very simple and user friendly. Each interface was constructed specifically to reduce the amount of user effort. This creates an overall easy to use app that will appeal to all aspects of a restaurant.

12. Design of Tests:

12.1) Test cases

- Base Design of General Tests
    - Physically select each item on the interface, regarding user selections such as the buttons available for customers, including ‘Take-Out’, ‘Dine-In’, and ‘Delivery’. Such selections should transition the screen to an appropriate area, such as the menu, or reservation selection screen, for the aforementioned options. If the screen does not change to the appropriate page, that would be considered a failed test.
    - Following the above scenario, each possible button will be pressed, the results will be noted and compared to what the actual result should have been. Again, if the expected outcome is not the experimental outcome, the test has failed. Specifically, if the button press is registered, but the result consists of no action - that is a failed test. If the button is pressed, and the outcome is unexpected, that is a bug along with being a failed test.
    - Due to much of the program being able to be tested by the abovementioned base tests, it is not entirely possible to create in-depth tests, unless it is for a standalone method that is able to be tested without need a manual key-press.
- In-depth Testing for Methods
    - General Form
        - Following the same idea of the base design, each method will undergo a remote call, from which the input will be varied and the output will be noted as well as compared to the proper output.
    - List of Methods and Their Test Inputs
        - selectItems()
            - While this method does not take in any input, it is called upon an item press for objects in the customer menu. Thus, this method will be tested through the general test aforementioned.

doneOrdering()

- This method takes in an arraylist populated with objects selected from the customer menu, and the output should be the items from the arraylist put into a queue, to be sent to the actor Chef.
    - Inputs will follow a mixture of objects put into the arraylist, including an empty array list, a list filled with invalid objects, and a list with a mix of the two.
    - If the program fails to account for inappropriate input, or if the output that is in the queue is not accurate, then the test results in a failure.

doneRating()

- This method does not take an input directly, but it registers the selection of the customer regarding the rating of the meal. This method will be tested by the aforementioned base test design.

selectTable()

- This method does not take in direct input, but is instead a method relating to the table UI. This method falls under the general, base test.

updateOrder()

- This method does not take in a direct input. The changes made by the customer will be updates on the database. If those updates are not the same as the input, then the test results in a failure.

setReservationTime()

- This method is called upon a button click as well, thus it follows the general test.

getTableStatus()

- Instance method that will be occasionally called by the server to update local table status. Will be tested by setting a status for a table, and calling this method to read it. (Do we really need this? Table will be updated by people, so we should only wait for that input)

setOrderStatus()

- A method that updates the status of a meal to ‘done’ when called upon. Testing will be done through the initial general tests.

getOrderStatus()

- Method that will obtain the status from the meal instance. Testing will be done through the initial general tests.

getInventory()

- This method takes no input and outputs the local inventory levels for the callee. Will be tested by means of the general test.

generateStats()

- Takes in no input and outputs visuals concerning customer orders based on an hourly, daily, and weekly basis. If no stats are output, or the numbers are incorrect, the test result is a failure.

getCheckInTime()

- Set/Obtains the check in time, depending on whether a check in time was already set. This method will be checked by the general tests.

getCheckOutTime()

- Set/Obtains the check-out time, depending on whether a check in time was already set. This method will be checked by the general tests.

12.2) Test Coverage

The tests specified should cover all normal operation by typical users. Covers only slightly abnormal input, in the case of data corruption upon the transmission of data. Thus, this coverage does not include any prevention of possible intruders attempting to circumnavigate the system to access personal data.

12.3) Integration and Testing

Following the exact same procedure as the general test expressed during the initial stages of 'Design of Tests', the correct combination of modules will be tested in series. As one module passes the test, it will then be paired with its corresponding modules (which has also been tested solo), and they will be tested together. Following this pattern, all modules will be systematically grouped and tested.

Automation Testing

Testing was done automatically using automation testing tool called QTP/UFT. QTP is a product provided by HP, it works for all mobile applications, web applications, and mobile applications. Unlike selenium which only test web applications.

Procedure of testing:

- Run QTP
- Record F3 and invoke the application (SpeedByte)
- Enter the login credentials
- Click Login
- Select Standard Check Point from QTP f12
- Added "Dine In" as first check point
- Click Enable, text area X and Y parameters
- Click Ok
- Selected synchronization point
- Went to Run Parameters
- Changed Synchronization time from 10 seconds to 25 seconds (Time Efficient) Allows application to synchronize with QTP as application takes approximately 15 seconds to Login.
- Exit the application normally (Not multitasking and swipe out the app from task manager in android tablet)
- Stopped recording
- Went to the expert view window
- Edited "GetRoProperty()" to get the confirmation message of successful login

Set obj = Application("name:=Login"). AppEdit("User ID:=tim@mail.com")
msgbox obj.GetTOProperty("User ID")
'Would retrieve the object User ID from the test object description, whether it's in the OR or DP defined
SetSecureobj = Application("name:=Login"). AppEdit("Password:=tim_pass")
msgbox obj.GetTOProperty("Password")
"Supposed to retrieve the object Password from the test object description, whether it's in the OR or DP defined". But as the Password is secure, it won't retrieve the password and the msgbox will be ***empty.***
(Continued below)…
Now we set the name property
obj.SetTOProperty "name", "Login Successfully "*And retrieve it*
msgbox obj.GetTOProperty("name")
Msgbox will pop up a Message Box with "Login Successfully"
All the points mentioned above confirms that automation checkpoint for Dine In, synchronization for Login, Black Box Testing for retrieving password (Results= Empty) , Plus testing for confirmation of database query is outputted in result set.

- Playback recording
- Results: 8 Passes and 1 Failed

Result set	**Passed/Failed**
System.util	Passed
Login	Passed
Confirmation Message	Passed
Dine In	Passed
Check Point	Passed
GetRoProperty("tim@mail.com")	Passed
GetRoProperty("tim_pass")	Failed
Confirmation Message ("Login Successfully")	Passed
Exit	Passed

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

13.1) Merging the Contributions from Individual Team Members

All information is first gathered on a Google document and each member contributes their individual parts of the project onto the document. This assures each member can contribute their parts with ease. Jimmy then takes each part and finalizes the report on a separate Word document to ensure consistency throughout formatting and appearance. After this, the document is then exported as a PDF file to keep everything consistent.

Issues and how they were tackled: At first, it was a bit difficult to fully coordinate which members would do which parts of the report. As well as having a lot of other classes to worry about, we wanted to schedule weekly meetings when we were all free to assure us each understood the goals and how we would reach them. After a few moments, we eventually got a weekly scheduled meeting and were able to be productive. Each member was appointed a specific part, with there being some members who were floaters and helped other members finish their parts if time permitted. Overall, our teamwork has been pretty great. Another problem may be the inconsistencies in fonts, appearances, and bolding/underlining when having multiple members do multiple parts. To fix this, we had one member copy all of the information from the Google doc and finalize a version on his own. This really helped with the consistency in formatting, appearance, and everything else.

Changes in report 1 were also made and are ready to submit along with the future changes in this report for the report 3 submission in the near future.

13.2) Project Coordination and Progress Report

Managing shared resources via Github, Google Doc, and other means of communication have allowed us to stay organized and complete our work in a timely manner.

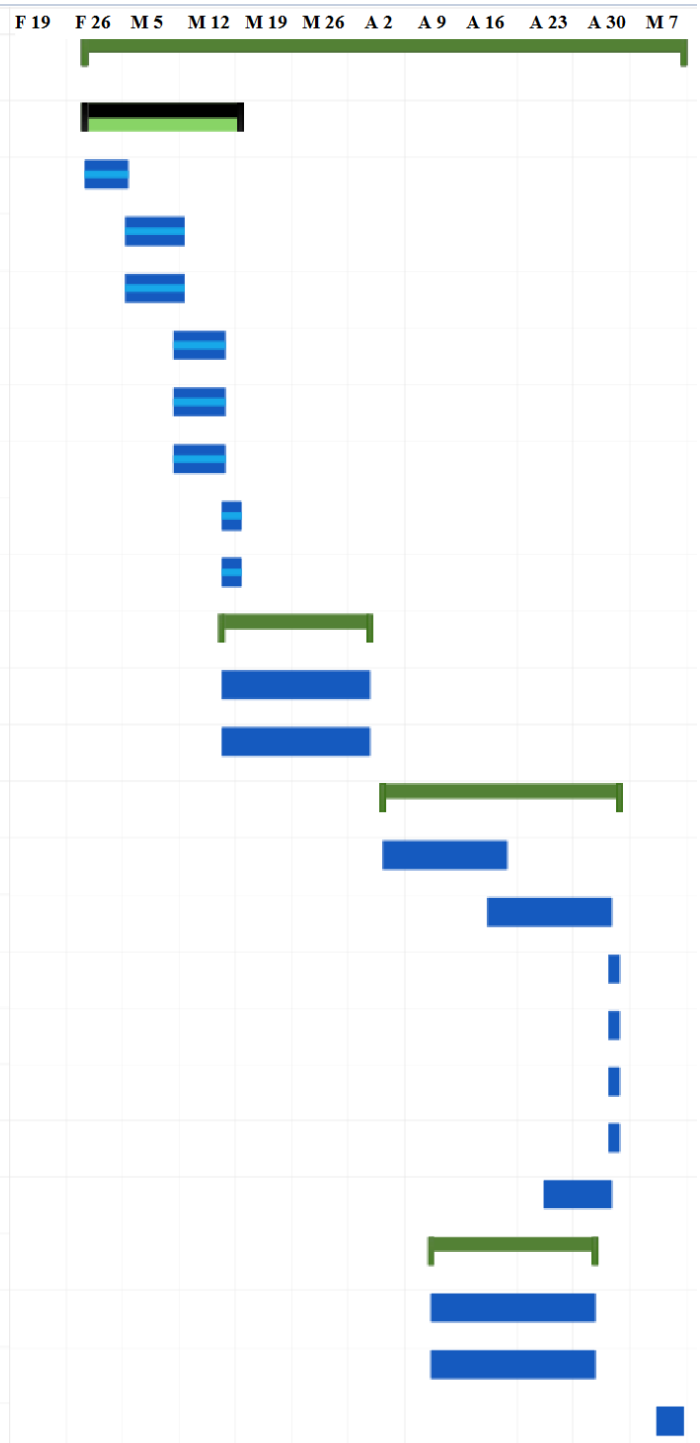
Current status: The team is currently using NodeJS for the server of the app and MongoDB for the database. This allows us to have communication throughout the app. We are also using Java and working on the GUI of the app to ensure that it is pleasing to the eye of the customers.

Future status: We worked on most of our main functions and solidly prepared for Demo 2. Also, having an easy to use and efficient interface will be one of our goals, while keeping top communication between our database and application.

13.3) History of Work

Functional Feature and Description	Start Date	End Date
Rating System - collecting information from customers on how well the food was prepared, and their rating of it from 1 to 5.	3/13/2017	3/20/2017
Payment System - allow customers to pay through the app, and give them the ability to add tip and split the bill.	2/20/2017	3/20/2017
Seating Chart - have an interactive chart that shows available seats to customers, and “dirty” seats to busboys.	2/27/2017	3/20/2017
Menu - an interactive menu so that customers can view items in greater detail, seeing ingredients, calories, estimated cook time, rating, etc.	2/20/2017	3/13/2017
Food Wait Time - implementing an active system that updates how long the customer has before their food is finished.	3/1/2017	3/20/2017
Inventory Tracker - implementing a system that reduces the amount of raw food in stock based on what has been ordered. A manager can update the inventory through the app as well.	2/27/2017	3/13/2017
Food / Customer Tracker - implementing a system that keep track of the flux of app using customers on a daily to hourly basis, and what they have ordered.	2/27/2017	3/20/2017
Reservation List - App using customers that wish to dine in will be placed on a reservation list, so that they may be easily seated. The hostess will have access to this list to confirm the reservation.	2/27/2017	3/6/2017
Table Alerts - alerting busboys that a table has been recently vacated so that they may come in and clean it up.	2/27/2017	3/6/2017
Waiter Related Alerts - alerting waiters to make sure to check up on tables and confirming the customer is satisfied, and about any food that is ready to be delivered to a table.	2/27/2017	3/13/2017
SpeedByte Server - backend for the app, will have to receive, send, and manipulate data.	2/20/2017	3/20/2017

ID	Task Name
0	**Project1**
1	**Report # 2(Complete)**
2	Interaction Diagrams
3	Class Diagrams
4	System Architecture
5	Algorithms
6	User Interface
7	Design of Tests
8	Plan of Work
9	References
10	**Demo #1**
11	Implementation
12	Presentation
13	**Report #3**
14	Revise Report 1
15	Revise Report 2
16	History of Work
17	Current Status
18	Conclusions
19	Future Work
20	**Reflective Essays**
21	**Demo #2**
22	Implementation
23	Presentation
24	**E-Project Archive**



13.4) Breakdown of Responsibilities

Outlined below are the teams, and the proposed work plan over the course of the next few weeks.

TEAM	CODE NAME	MEMBERS
Administration/Management	Team α	Ram, Jose
Employees/Waiters	Team β	Vam, Tim
Customers	Team ¢	Jimmy, Pawel, Husen

SHORT TERM PLAN OF WORK	TEAM
Create a communication bridge between manager and company material via a server	Team β, Team ¢
Create an interactive customer order menu	Team α, Team β
Implement the available table seating chart	Team ¢
Administration information and data trends	Team α
Customer reservations via data capture and storage will populate a list	All

Work done, current work, and work to do continued on next page.

Team Member	Work Done	Current Work	Work To Do
Team α	Functional Requirement and Domain Model	Create an interactive customer order menu	Work on payment system for employees and inventory tracker
Team β	User Interface Specification, Mathematical Model, and UI diagrams	Create a communication bridge between manager and company material via a server	Finalize User Interface and customer menu
Team ¢	CSR, System Requirements, Functional Specification, Plan of Work	Implement the available table seating chart	Finalize waiting order in queue, table availability, and bill payment

The integration will be coordinated by Vam and Jimmy to assure everything is properly integrated and each part of the application works and communicates properly. Integration will be performed by each member who works on a specific unit, but mainly teams β (Vam, Tim) and ¢ (Jimmy, Pawel, Hussain) will be doing the testing and assurance tests.

Key Accomplishments:

- Server and application created from scratched
- Fully linked server to database and application
- User-friendly interface
- Key features like message center, rating system, and food wait times
- Finished all work in time frame

Future Work:

Although the second demo is over and done with, we look forward to improving our application as a side project to continue to improve our skillset as developers and software engineers. Some features include:

- QR Code for signing in when at restaurant to increase efficiency
- Have a central operating database system
- Restaurant Finder for finding nearest local restaurant

14. References:

1. http://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html - used for concise details for each part of our report

2. http://www.ece.rutgers.edu/~marsic/Teaching/SE/syllabus.html - used for scheduling our meetings and work to be done

3. http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/ - used for clear explanation of the Restaurant automation project choice

4. http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf - used for specific terminology usage

5. www.openclipart.org - used for creation of logo and other diagrams

6. Adobe PhotoShop - used for all User Interface diagrams

7. Argo application - used for each UML diagram included

8. http://www.restaurantfurniture.net/restaurant-design/ - restaurant image used in User Interface

9. https://millersalehouse.com/menu/ - cheeseburger image used in User Interface

10. Microsoft Developer Network. "Chapter 3: Architectural Patterns and Styles." *Chapter 3: Architectural Patterns and Styles*. Microsoft, n.d. Web. 12 Mar. 2017. - Used in System Architecture - Part A

When specific references were used, they were specifically labeled in the section. For the other references, we used them as a general basis for different aspects, such as the report layout and grading breakdowns.