Report #2 - 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:

	Project Category	Team Member Name						
		Jimmy	Tim	Jose	Pawel	Husen	Ram	Vam
Responsibility Levels	**Sec 1. *Interaction Diagrams***	*20%*	*20%*			*60%*		
	Sec 2. *Class Diagram and Interface Specification*			*20%*	*40%*		*40%*	
	Sec 3. *System Architecture and System Design*	*20%*						*80%*
	Sec 4. *Algorithm and Data Structures*			*5%*		*20%*	*25%*	*50%*
	Sec 5. *User Interface Design and Implementation*		*90%*			*10%*		
	Sec 6. *Design of Tests*				*50%*	*30%*	*20%*	
	Sec 7. *Project Management and Plan of Work*	*90%*				*10%*		

Table of Contents:

1. Interaction Diagrams

Sequence Diagrams:

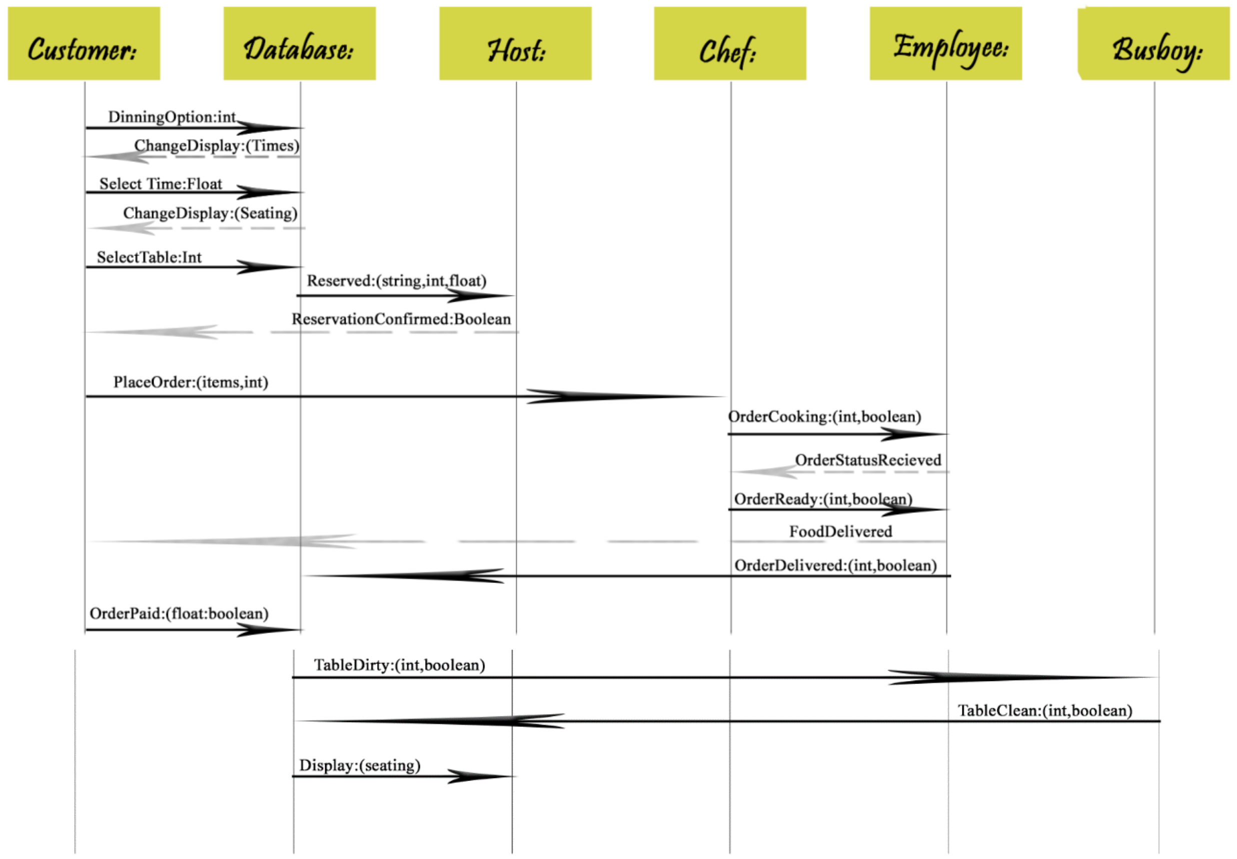
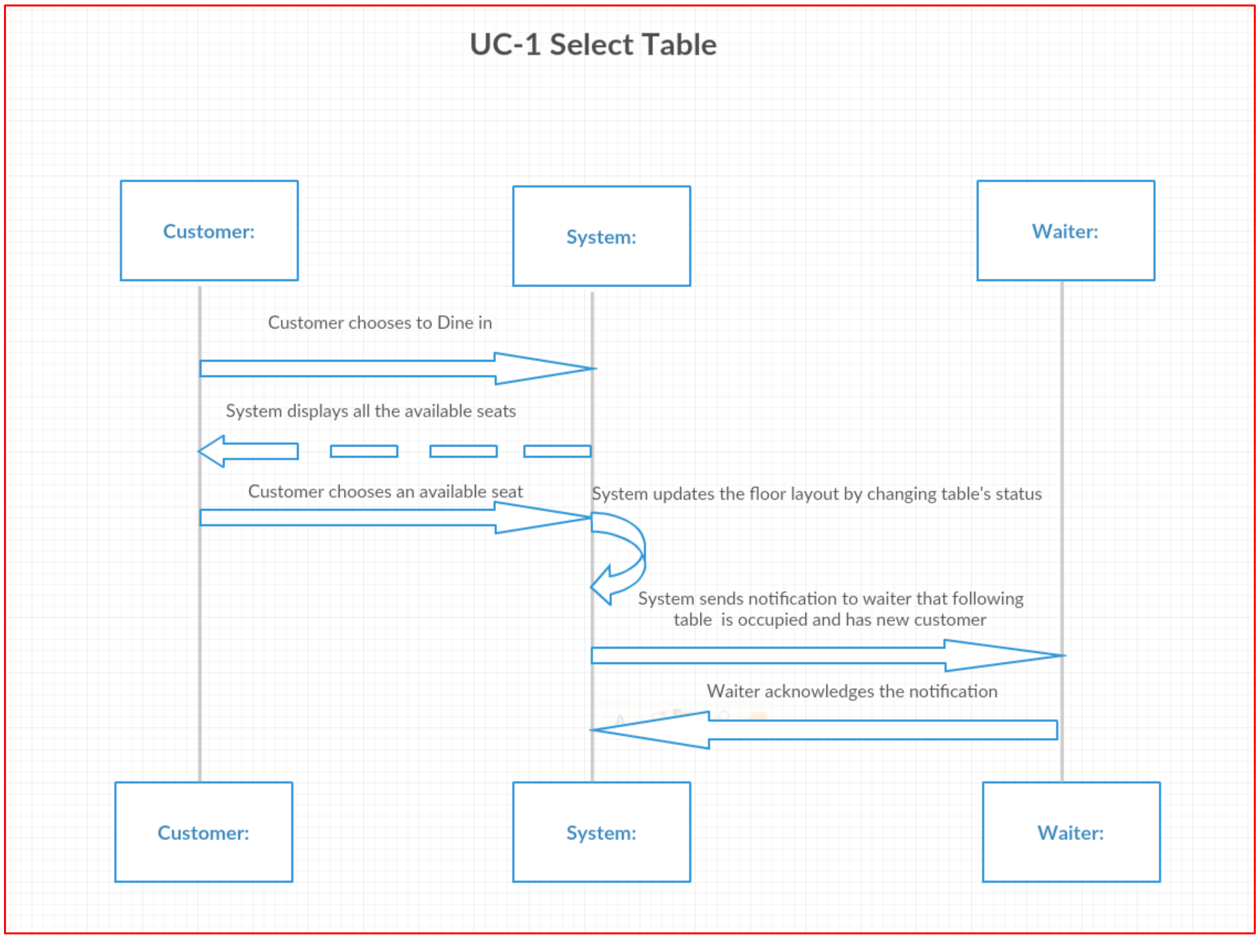
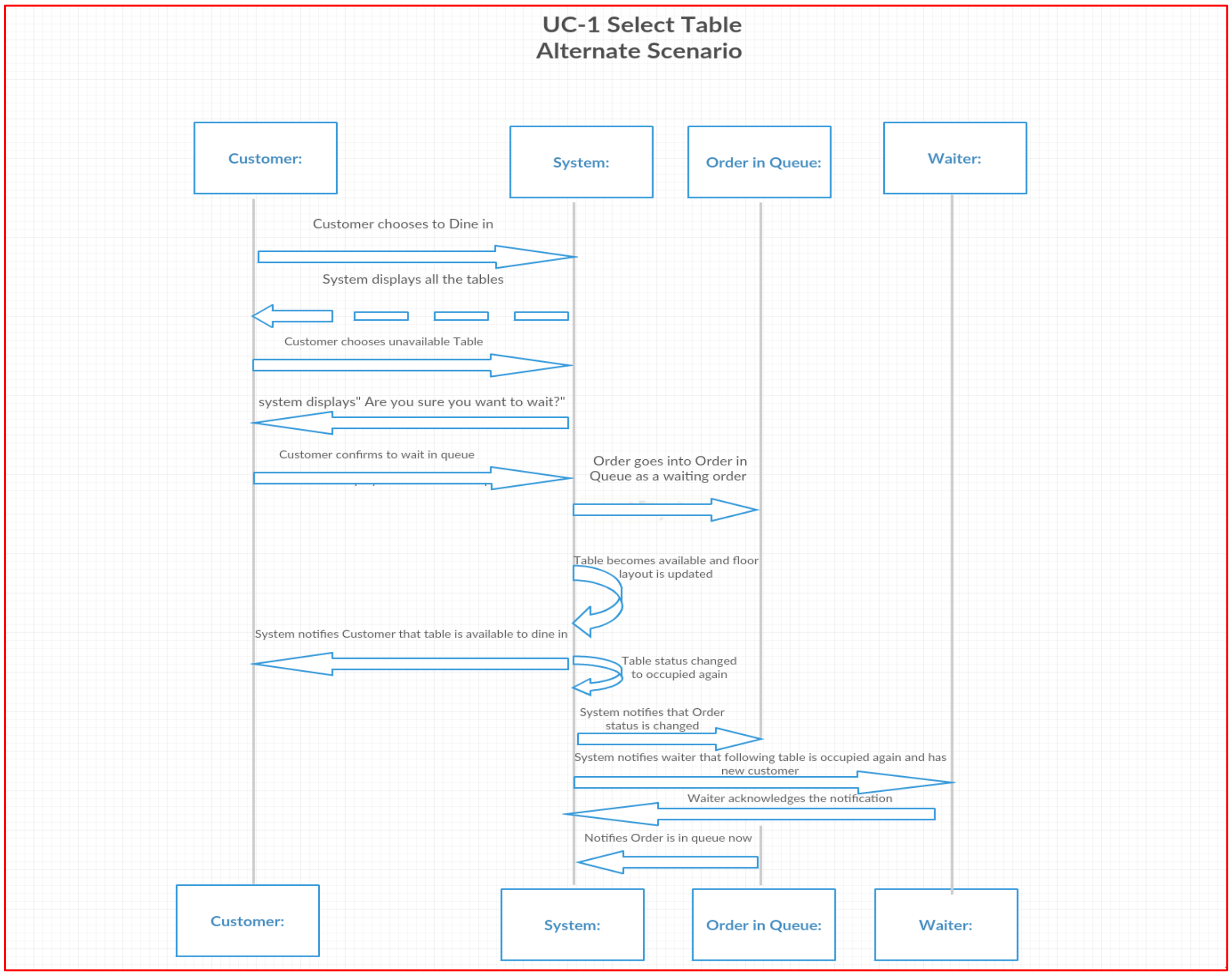


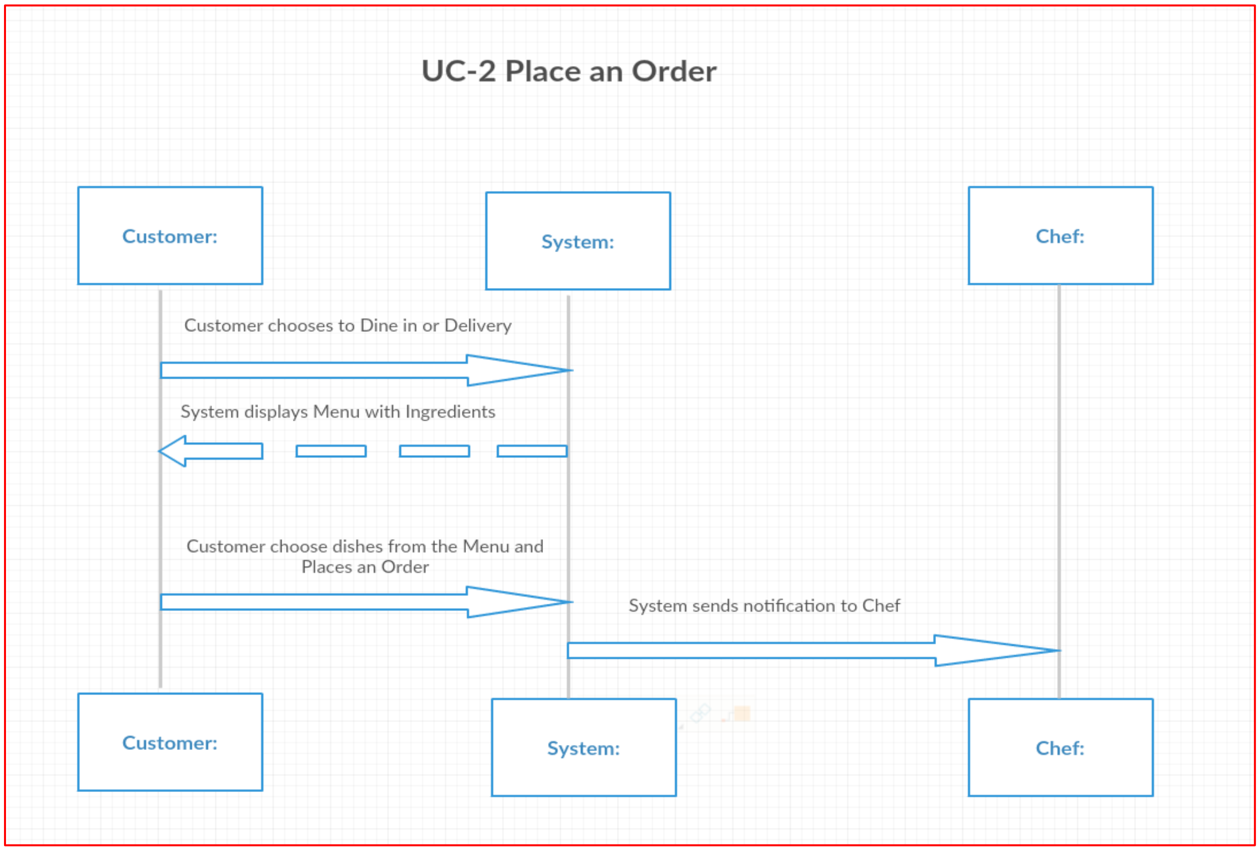
Diagram 1: The above diagram represents the communication throughout all the different users for a typical customer transaction. The customer is the one who sets the stage for the rest of the actors which makes it a perfect example to show the interactions amongst all of the actors.


UC-1 Select Table
Customer:
System:
Waiter:
Customer chooses to Dine in
System displays all the available seats
Customer chooses an available seat
System updates the floor layout by changing table's status
System sends notification to waiter that following table is occupied and has new customer
Waiter acknowledges the notification
Customer:
System:
Waiter:

<u>Diagram 2:</u> 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.


UC-1 Select Table
Alternate Scenario
Customer:
System:
Order in Queue:
Waiter:
Customer chooses to Dine in
System displays all the tables
Customer chooses unavailable Table
system displays" Are you sure you want to wait?"
Customer confirms to wait in queue
Order goes into Order in Queue as a waiting order
Table becomes available and floor layout is updated
System notifies Customer that table is available to dine in
Table status changed to occupied again
System notifies that Order status is changed
System notifies waiter that following table is occupied again and has new customer
Waiter acknowledges the notification
Notifies Order is in queue now
Customer:
System:
Order in Queue:
Waiter:

<u>Diagram 3:</u> 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.


UC-2 Place an Order
Customer:
System:
Chef:
Customer chooses to Dine in or Delivery
System displays Menu with Ingredients
Customer choose dishes from the Menu and Places an Order
System sends notification to Chef
Customer:
System:
Chef:

<u>Diagram 4:</u> 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).

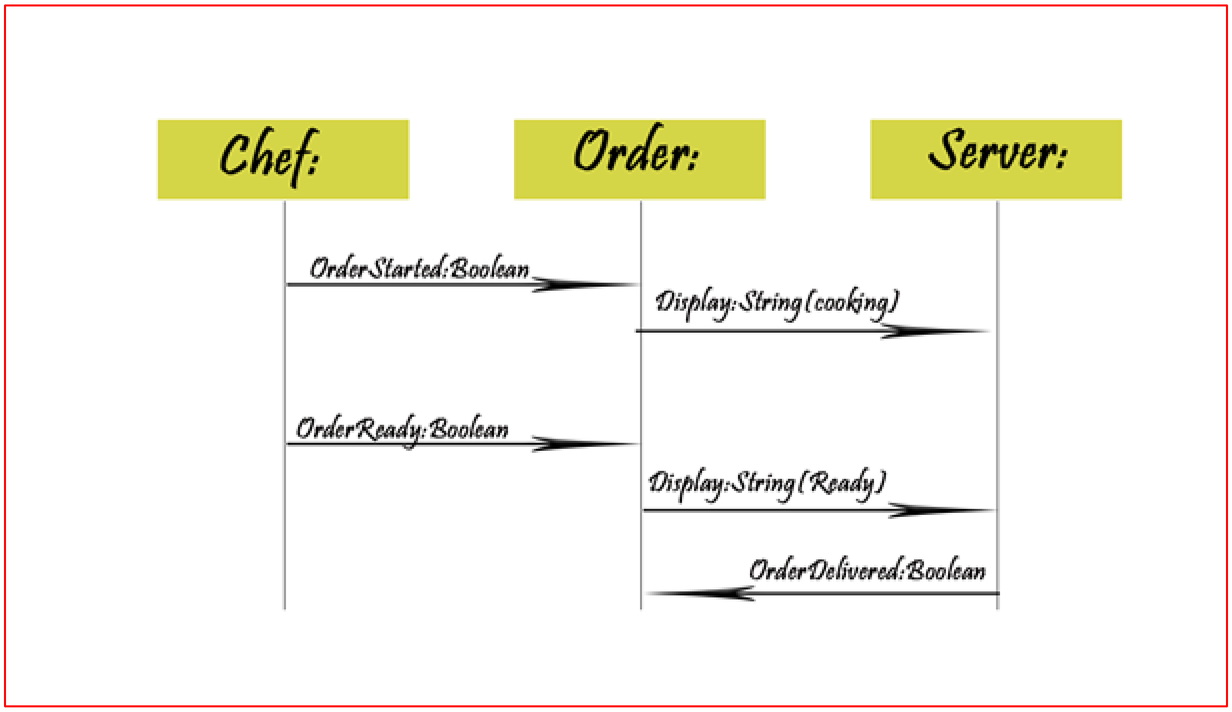
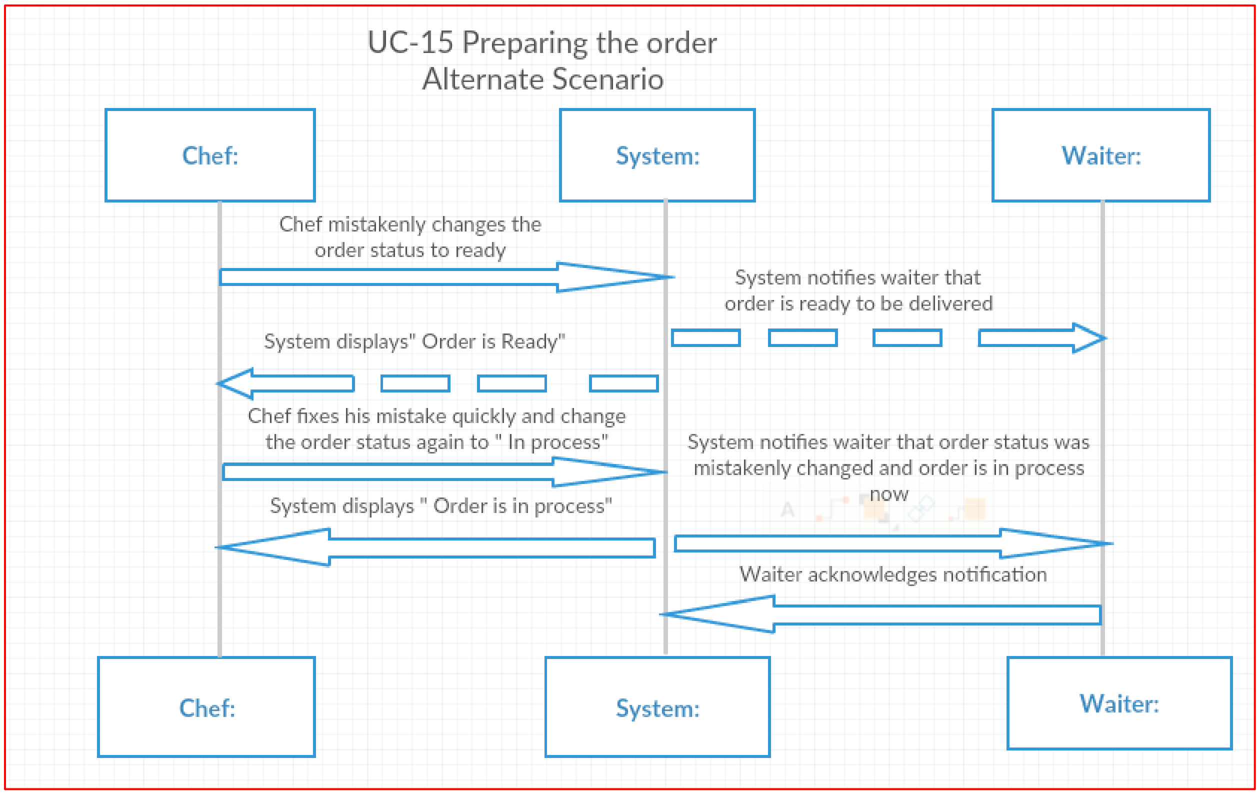
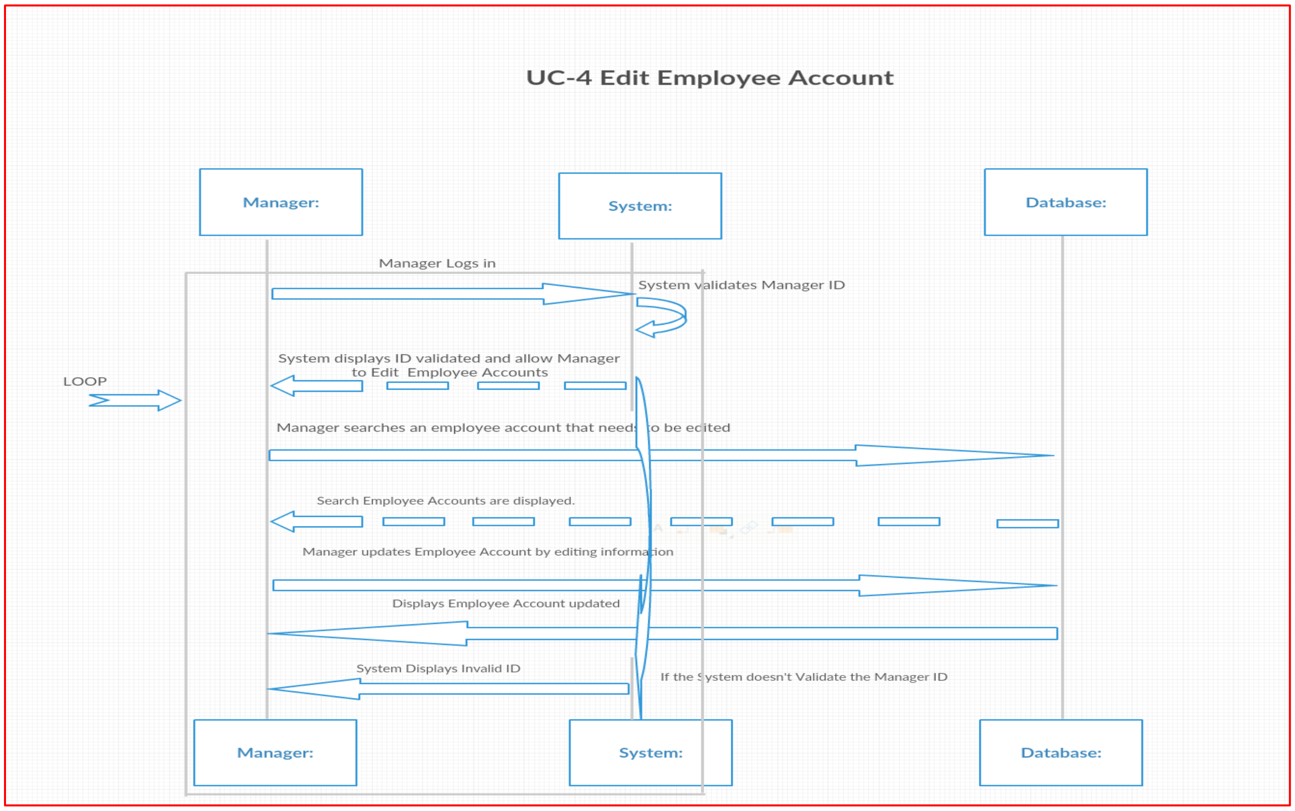


Diagram 5: The above interface shows the simplicity of the Chef's interface. The chef simply tells the system when an order is started which will notify the server accordingly. The chef then finishes preparing the order, and signals the employees that the food is ready for pickup.


UC-15 Preparing the order
Alternate Scenario
Chef:
System:
Waiter:
Chef mistakenly changes the order status to ready
System notifies waiter that order is ready to be delivered
System displays" Order is Ready"
Chef fixes his mistake quickly and change the order status again to " In process"
System notifies waiter that order status was mistakenly changed and order is in process now
System displays " Order is in process"
Waiter acknowledges notification
Chef:
System:
Waiter:

<u>Diagram 6:</u> 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.


UC-4 Edit Employee Account
Manager:
System:
Database:
Manager Logs in
System validates Manager ID
System displays ID validated and allow Manager to Edit Employee Accounts
LOOP
Manager searches an employee account that needs to be edited
Search Employee Accounts are displayed.
Manager updates Employee Account by editing information
Displays Employee Account updated
System Displays Invalid ID
If the System doesn't Validate the Manager ID
Manager:
System:
Database:

<u>Diagram 7:</u> 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.

2. Class Diagram and Interaction Specification

Part a.) Class Diagram

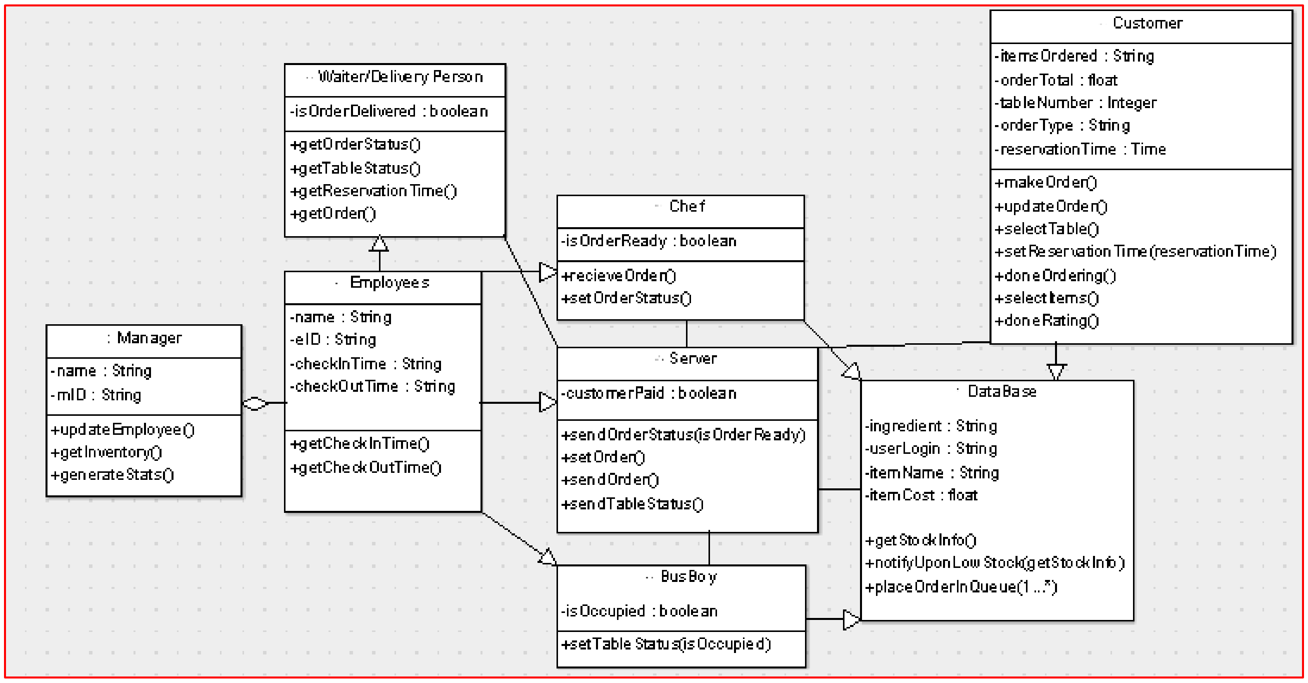


Figure: Complete Class Diagram

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.

Part b.) <u>Data Types and Operation Signatures</u>

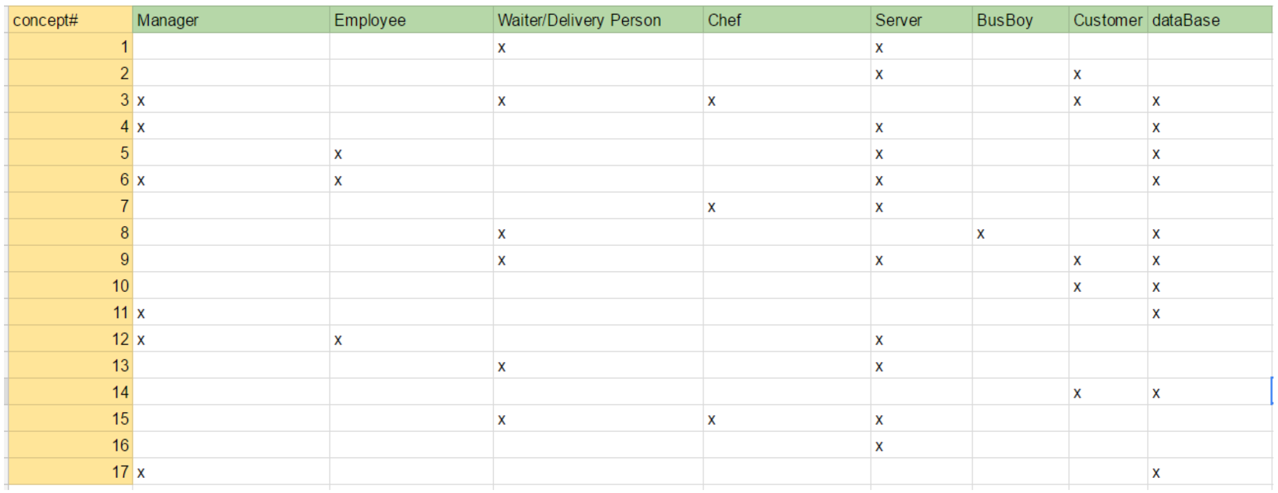
- Class - Manager
  - Attributes:
    - -name: ‘name’ is stored in a private String, containing the personal name of the user of this class.
    - -mID: ‘mID’ is stored in a private String, containing the manager identification number of the user of this class.
  - Operations:
    - -updateEmployee(): This operation serves as a way for the Manager class to update the information in the Employee class.
    - -getInventory(): This operation serves as a way for the Manager class to obtain information on the current status of restaurant’s inventory.
- 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 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.

Part c.) <u>Traceability Matrix</u>



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”

3. System Architecture and System Design

Part a.) <u>Architectural Styles</u>

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

    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

    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

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

  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

  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.

Part b.) Identifying Subsystems

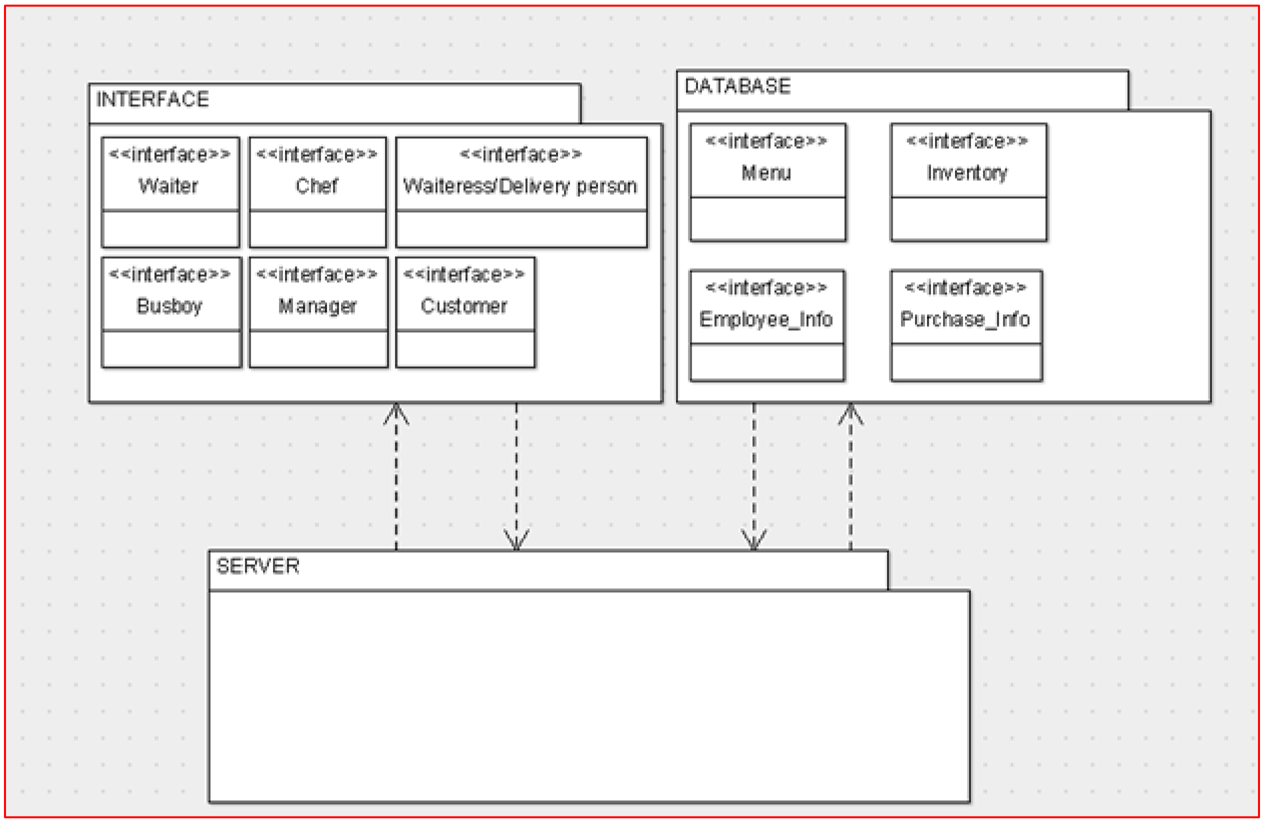


Figure: UML Package Diagram

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.

Part c.) 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.

Part d.) 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.

Part e.) 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.

Part f.) 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.

Part g.) 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

4. Algorithms and Data Structures

Part a.) 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.

Part b.) <u>Data Structures</u>

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.

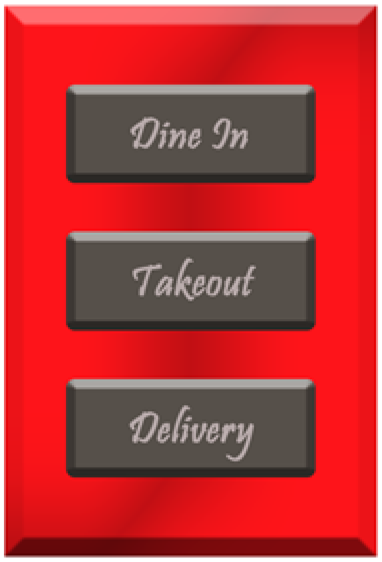
Chef: The chef's side of the application will require a data structure that will allow easy access of the items. A queue is the data structure which will fit perfectly into this. The items will be put in order of the customers' orders which will be added to the queue appropriately.

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.

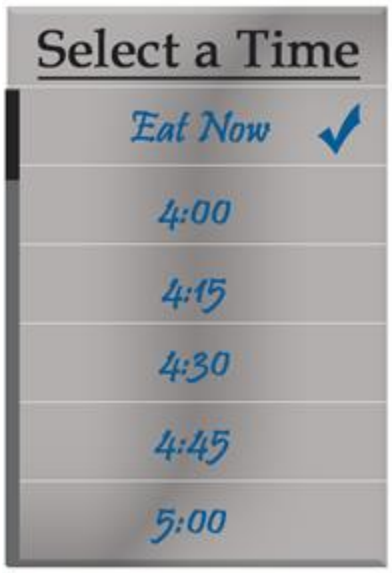
5. User Interface Design and Implementation

Customer Interface

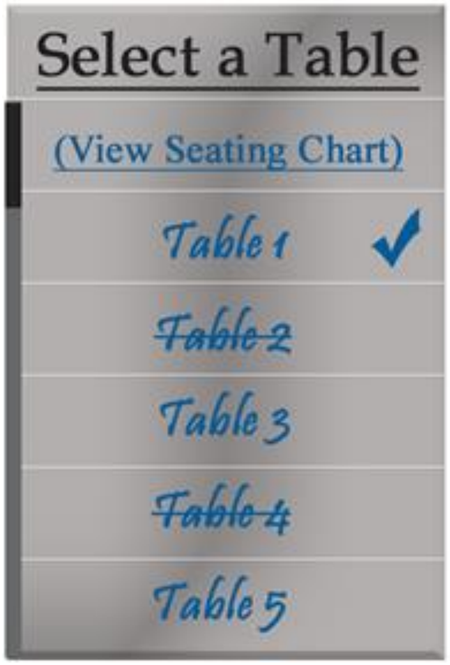
The Customer is first asked to select a one of three dining options, Dine In, Takeout, and Delivery. If Takeout or Delivery are chosen the user is brought directly to the ***Menu Page***. On the other hand if they choose to dine in they are directed to the ***Reservation Page***.

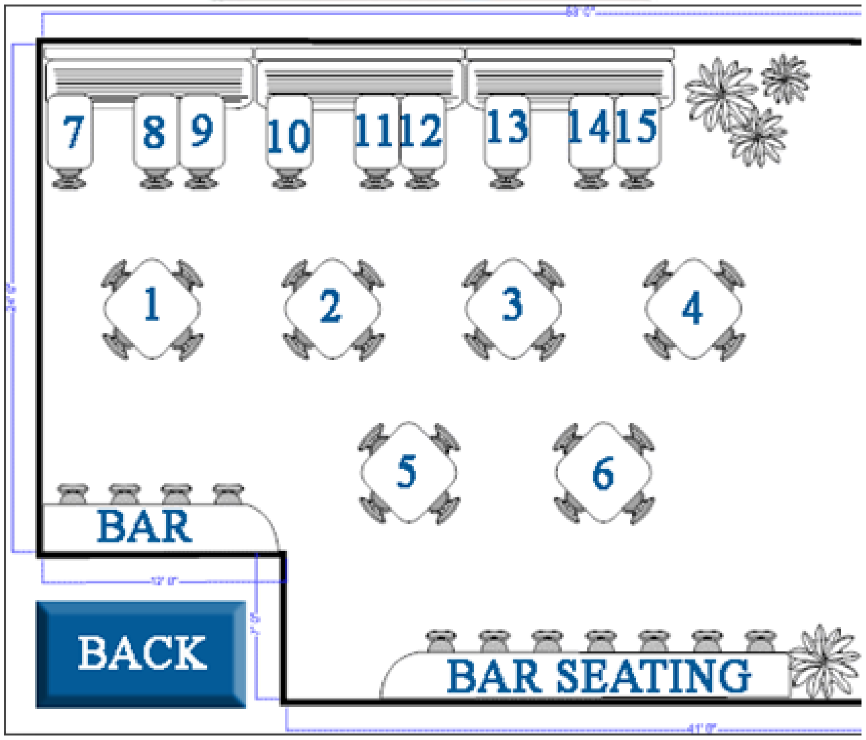


If the Customer selects Dine in they are brought to the ***Reservation page*** which is an interactive scroll menu that is set in fifteen minute intervals. Once the user selects a time they a directed to the ***Table Selection page***.



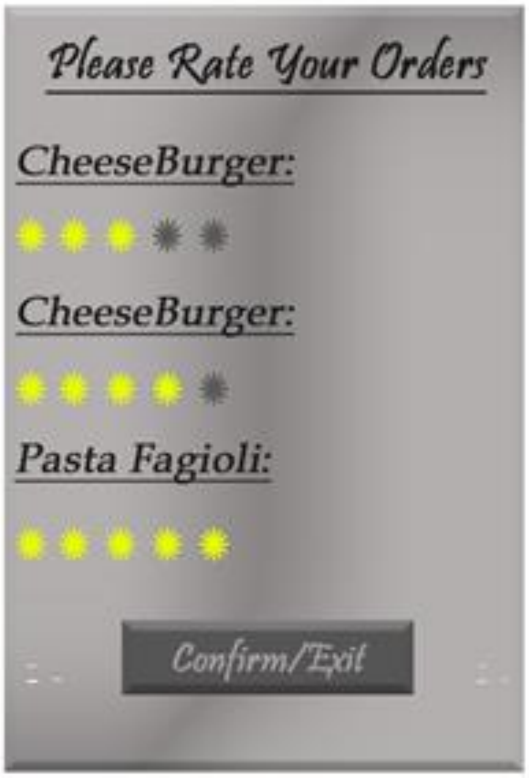
After selecting a time the user is brought to the table selection page, this is an interactive scroll menu listing all the tables in the restaurant. If a table is crossed out that means it is unavailable at the selected time. If the customer is not a regular and wishes to see a seating plan of the restaurant 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.





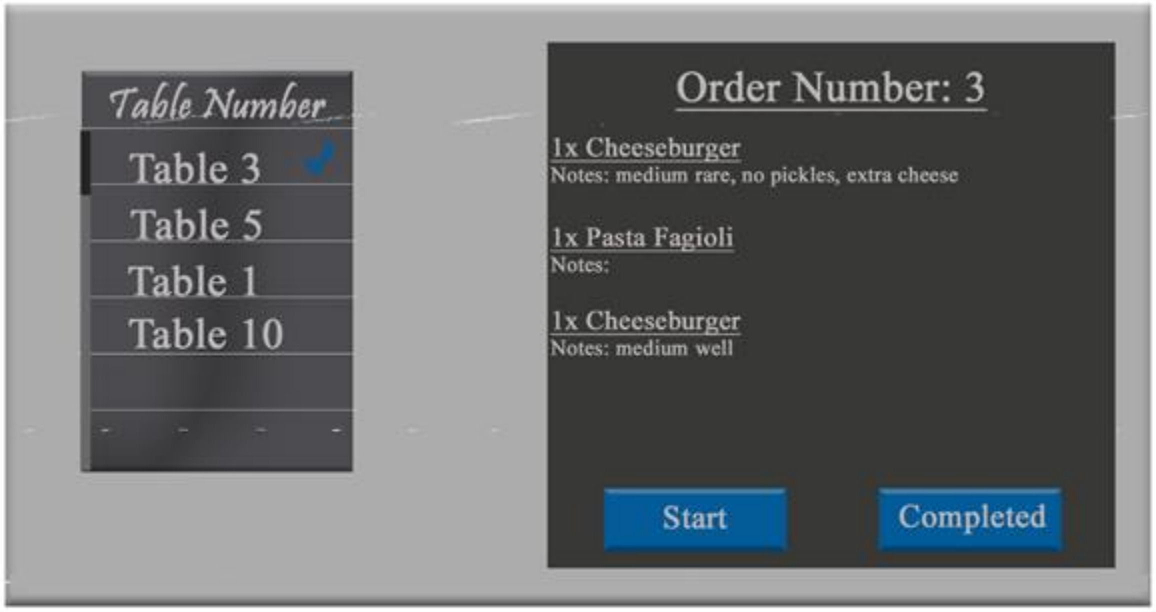
Reference for Image: http://www.restaurantfurniture.net/restaurant-design/

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 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.



Order Number: 3

1x Cheeseburger
Notes: medium rare, no pickles, extra cheese

1x Pasta Fagioli
Notes:

1x Cheeseburger
Notes: medium well

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.

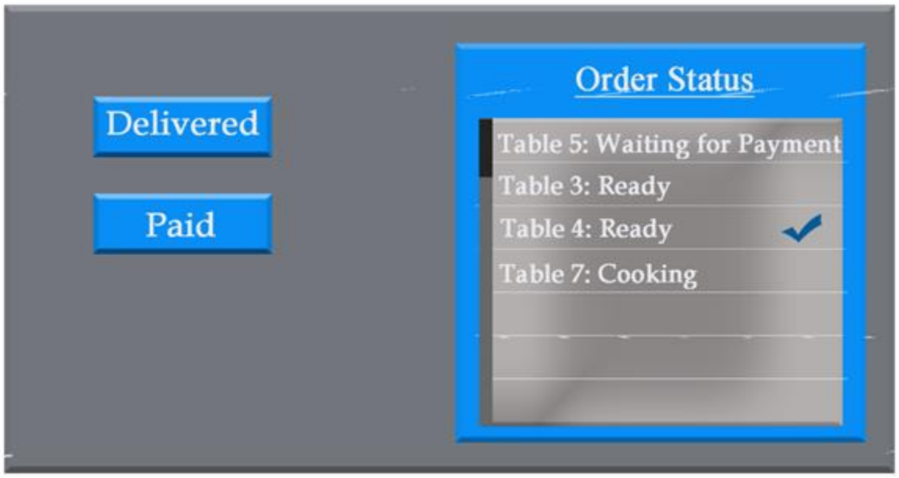


Table 5: Waiting for Payment
Table 3: Ready
Table 4: Ready ✓
Table 7: Cooking

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.

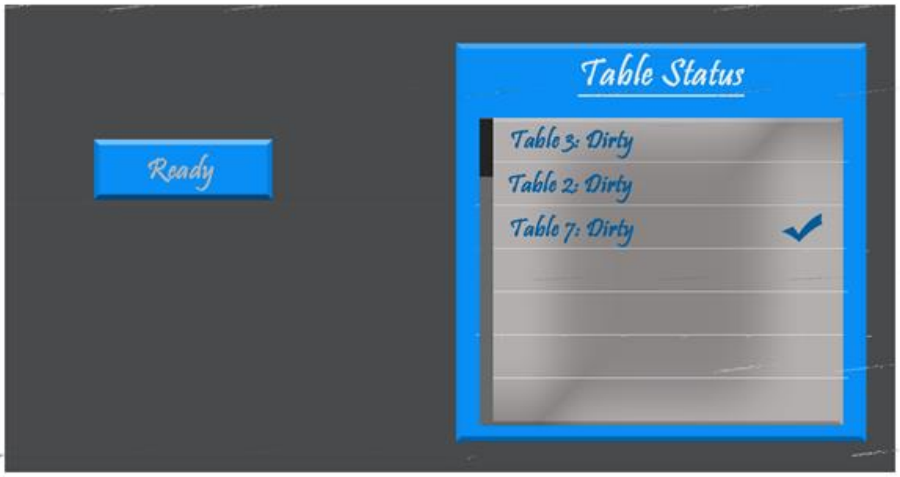
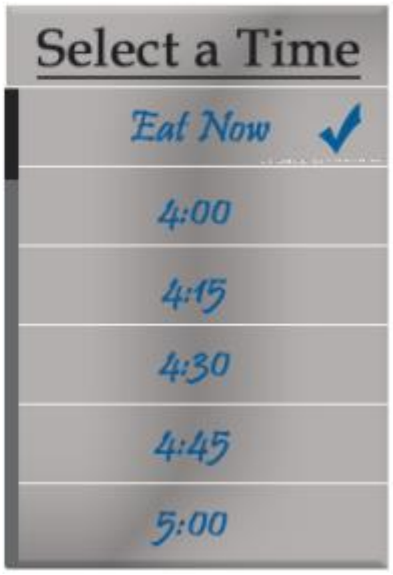


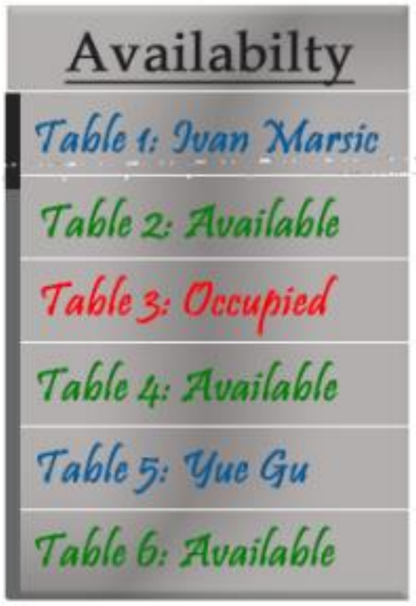
Table 3: Dirty
Table 2: Dirty
Table 7: Dirty ✓

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.

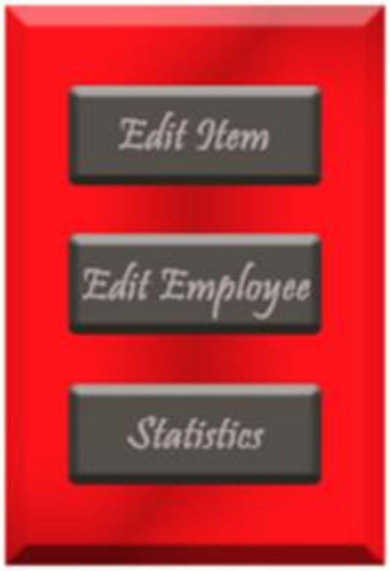


After selecting a time the Host will be brought to the ***Table Availability Page***. This page shows the current status of each table. Green tables mean the table is available. Red tables mean that the corresponding table is occupied, and Blue tables mean they are reserved but the person has not been 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.



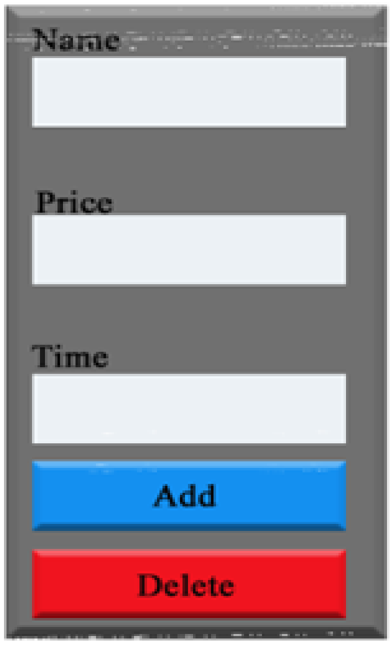
Management Interface

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***.



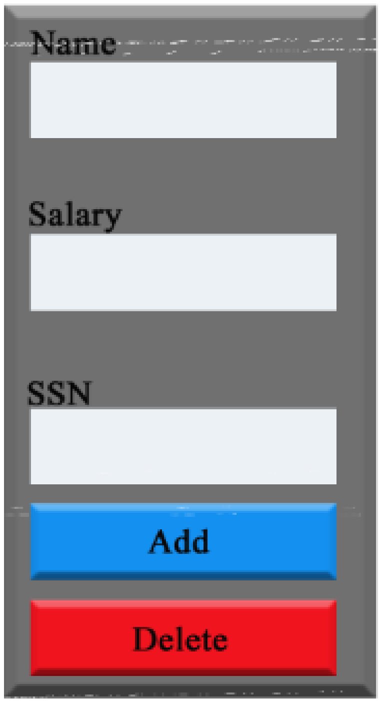
Add/Delete Item

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.

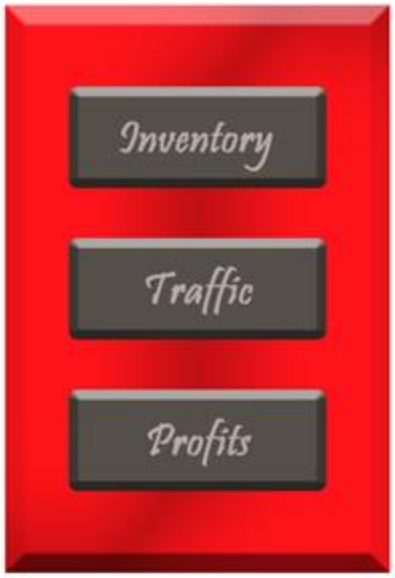


Add/Delete Employee

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.

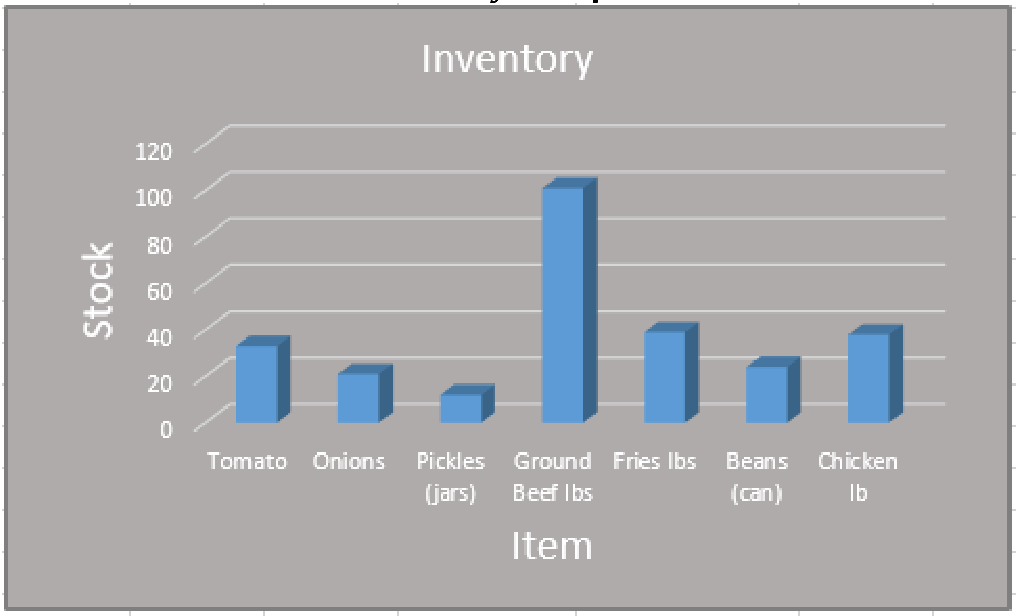


If the manager selects ***Statistics*** they are brought to the ***Statistics Selection Page***, here they have the option to select what statistics they would like to view.

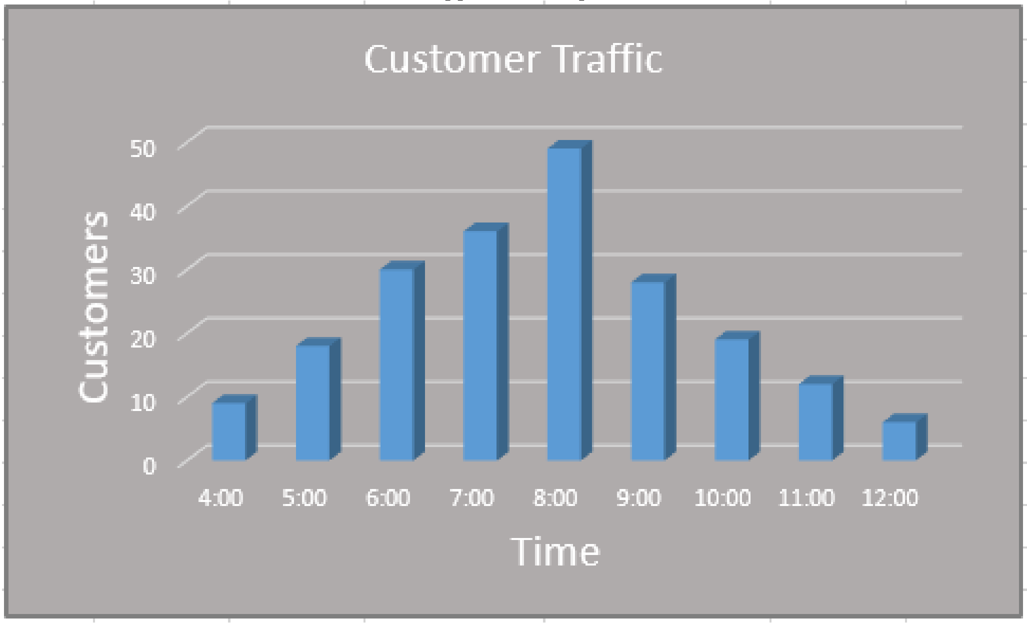


Statistics continued below.

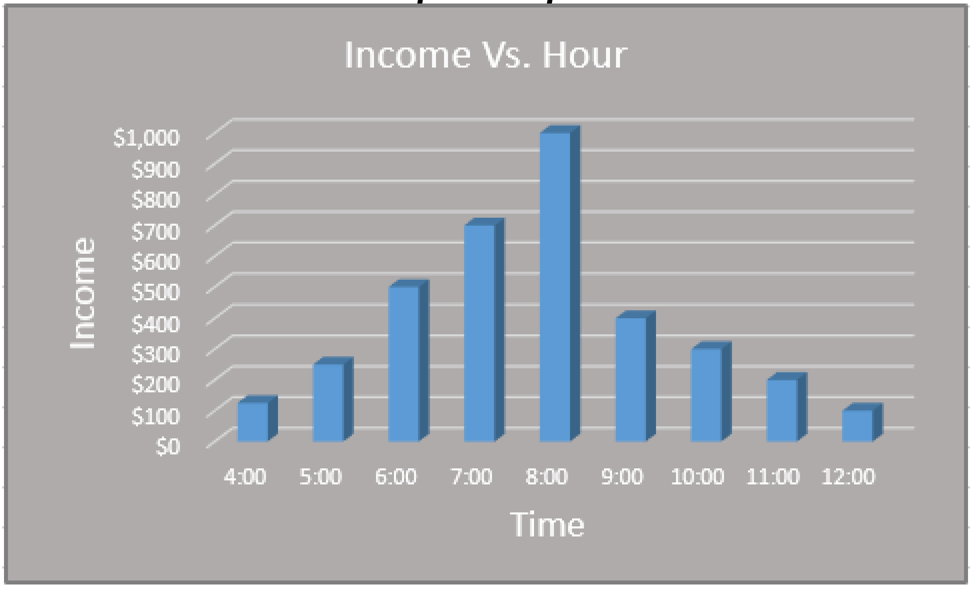
Inventory interface



Traffic interface



Profits interface



In any of the interfaces if the user wishes to go back to a previous page they can simply swipe right on their screen, this will bring them to the previous page.

The design has come a long way from the initial stages of development. We have streamlined 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. Efficiency at its finest.

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.

6. Design of Tests

Part a.) 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.
    - 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.
- 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.
        - 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.
        - 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.
- 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.

Part b.) Test Coverages

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.

Part c.) 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.

7. Project Management and Plan of Work

Part a.) 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 we 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.

Part b.) 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.

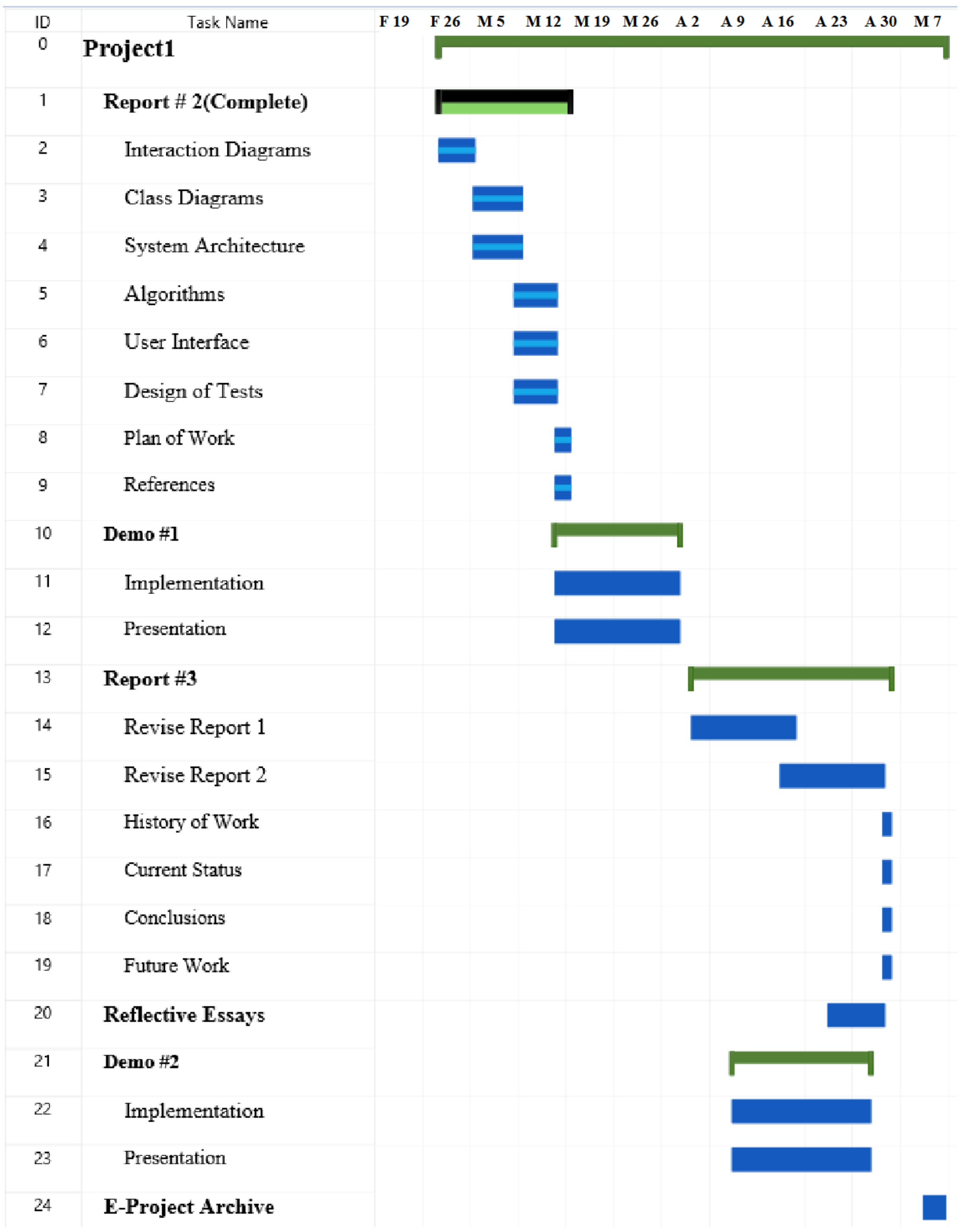
Current status: The team is currently using NodeJS for the server of the app and MongoDB for the database. This allows us to have communication throughout the app. We are also using Java and working on the GUI of the app to ensure that it is pleasing to the eye of the customers.

Future status: We will begin to work on the main functions and test cases to prepare ourselves for Demo 1. Also, having an easy to use and efficient interface will be one of our goals, while keeping top communication between our database and application.

We are currently on track with our near future goals.

Part c.) <u>Plan of Work</u>

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**

Part d.) 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, Husein) will be doing the testing and assurance tests.

8. 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