# Restaurant Automation

Group 2

Praveen Chekuri
Pradnya Pisal
Zac Brown
Kartik Bhatnagar
Bill Fung

# Table of Contents

## Summary of Changes:

We haven't really changed our specification too much since we would like to keep working on the project as our senior design. Below are some of the key edits performed:

- Fully Dressed Use Cases: We have only specified 2 in great detail as we implemented those 2.
- Use Case Diagram: Changed this up with a few of the new modifications.
- Domain Analysis: Better defined domain analysis along with individual domain analysis of use cases we implemented.
- Added Sequence Diagrams
- Unit Tests and Integration Tests were added.
- Also check Appendix for minor use case changes.

## 1. Customer Statement of Requirements

In order to effectively run a restaurant, cost and time saving are essential. Minimizing time by a few seconds for each table can speed up order processing, increase efficiency and boost profits. The biggest hurdle most restaurants face is the migration from a paper-pencil system to a completely automated touch-screen system. Most retail establishments choose to use a point-of-sale system (herein referred to as POS) to handle their transactions. For a restaurant, a POS system can greatly change the following problems plagued by typical 'pen & paper' establishments:[1]

- Keeping track of empty, clean and reserved tables within a restaurant. This requires a notebook or whiteboard which is constantly changing.
- Waiters making mistakes with customer's orders. At times, a waiter can forget to add a specific item, make a change due to a customer's allergies, or forget to give the order to the kitchen.
- Busboys need to keep track of which tables need clearing. This means that they must be always checking for tables. Waiters need to usually alert them. This takes extra time from other staff.
- Waiters need to constantly check with cooks to determine when food is ready. Conversely, cooks need to make sure waiters know that food is ready. This can create cold food (potential food-poisoning), wrong orders and an unsatisfied customer.
- Managers need to maintain record-keeping of employee hours for payroll. They must also re-print menus when food is not available or a price needs to be changed. This can be costly and time-consuming to a restaurant.
- Managers need to analyze hundreds of paper receipts to determine best-selling items, popular hours and customer satisfaction.
- Customers, when seated, have to wait for a waiter to order. They must rely on the waiter to remember their order and specific details. Their food may take longer to be prepared if the waiter has multiple tables. They may be billed incorrectly since they cannot see their check until their meal is complete.
- Typical reservation systems rely on host/hostess to mark down reservation requests using a notepad and marking the table during the time. This could lead to reservation discrepancies.
- Impatient customers also call over the waitress to find out the status of their order several times during their visit, wasting the waiter's service time.

To remedy these issues, a POS software solution is the best option. The POS system should have these basic high-level features[2]:

- Allow the restaurant to operate faster (faster turnaround on food, faster seating, faster order preparation)
- Reduce employee error (thereby increasing customer happiness). This also reduces waste (when the wrong item is ordered, food must be discarded).
- Operate on two hardware platforms:
    - Desktop touch-screen computers for employees. Employees will use the touch-screen computers to interface directly with the POS system.

[1] These are a variety of typical problems which the software should tackle. The software solution should at a minimal tackle these problems.

[2] A more detailed remedy is further prescribed in the CSR.

- Android tablets running a version of the POS system. The tablets will be provided to customers, at their tables, allowing them to directly order from the computer system installed in the restaurant. The tablets are the property of the establishment and are kept at each table.

The architecture of the system shall appear as follows (connected lines illustrate a wired network connection, while curved linear near a device represent a wireless network connection):

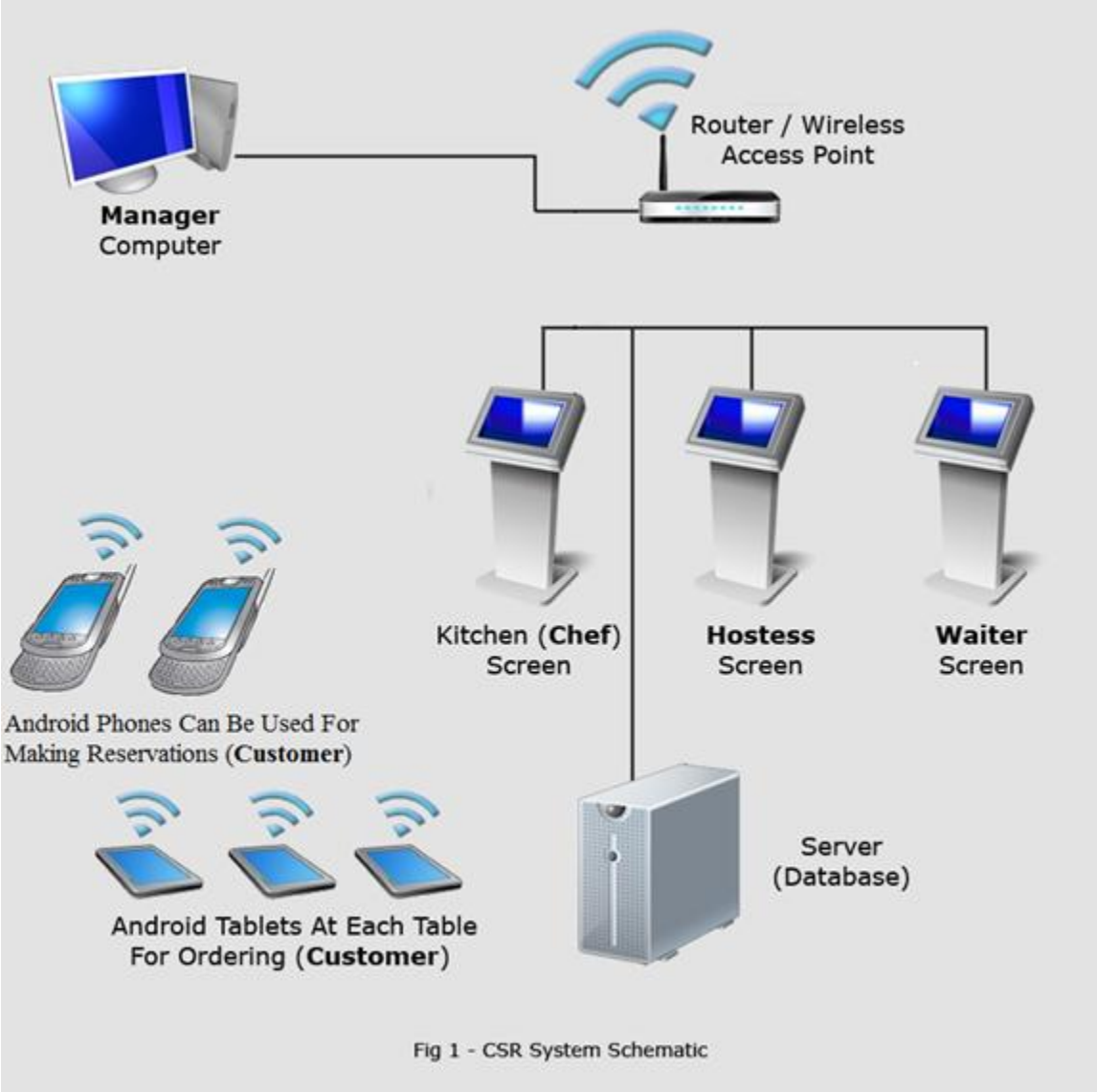



**Fig-1: Architecture of the System**

**Detailed Requirements**:

The system (architecture seen above), should provide the following specific functional requirements:

- Provide six users and abilities for each user role: Manager (Administrator), Chef, Hostess, Waiter, Customer.

Each of the user roles should meet the following functional requirements:
Manager**:**

- Create users which fill in each role. Manage users in each role (delete them, modify their pay, add comments, etc.)
- Manage users (view the time sheet of each employee and determine based on their hours their pay rate)
- Modify the menu of the restaurant. Add items, remove items, change their description, and update prices.
- Run reports on the restaurant (these can be static or graphical as necessary):
    - Determine which items are the most popular
    - Determine the most popular time of day for sales
    - Determine how long it takes for food to be prepared.
- Leave notes on employees (payroll, behavioral, etc. for the employee record)
- Modify the table layout of the restaurant (floor plan). This change will be reflected on the screens of all other restaurant users who have layout access abilities.
- Manage any administrative computerized aspects of the restaurant.
- Edit the advertisements displayed by the tablets during inactive periods.
- Has the authority to control customer's bills in order to provide discounts due to inconvenience.
- Login interface to prevent un-authorized access of restaurant management.

Hostess:

- View a layout of the restaurant
- Using a color coded system, red (occupied table OR being cleaned), yellow (reserved), green (open), be able to quickly identify tables where guests may be seated at
- Change the status of a table to "occupied (red)" when a customer sits at a table.
- Ideally, show the customer the restaurant layout, thereby allowing the customer to sit at his/her chosen table
- Ability to use 'timecard' functions for payroll purposes (e.g. clock-in and clock-out of shifts)
- Assign waiters to tables using their judgment based on waiter's experience and party size.

Chef:

- Queue of upcoming orders, organized by order most-recently submitted.
- Ability to determine the table number of the customer who ordered, as well as the waiter who is assigned to the table.
- When an order is prepared, a feature to "Complete" the order. The appropriate waiter will then be notified at their terminal.
- When an order is completed, the order will be removed from the queue. If there are additional orders (not on screen), the order will be fetched from the system and added to the screen.
- Chef's screen will have a login in order for manager to track chef's orders in case customers complain about poorly cooked food. There will be several terminals which will depend on

the number of chefs working in the kitchen and feasibility of installing additional terminals in the kitchen.
- Interface should include a 'timecard' function for payroll

Waiter:

- Ability to access the tab of a table they are assigned to. Within the tab the waiter should be able to:
    - View the cost of specific items
    - Modify the tab of the table (e.g. provide complementary food in case the customer is not satisfied), add specific extra charges, or reduce the bill
    - Close the tab of a customer. When a customer is ready to pay, the waiter should be able to charge payment
        - ✓ Provide support for entering a credit card number (stored in database for processing later)
        - ✓ Open a cash drawer for cash
- Alerts when a table is seated, when a table 'calls them over' , and when a tab is requested to be closed.
- Notification system when food for one of their tables is ready.
- Quick login system. There may be multiple waiters sharing the same terminal. A drop down box of waiters, as well as a pin number for secure login (4 digits).
- Ability to use 'timecard' features, similar to other employees (payroll purposes).
- **In addition**, busboy duties will be replaced by waiters in order to save money for restaurant operation. The additional duties shouldn't be too demanding for the waiters as their duties will be reduced by the automation.
- Alerted when a table tab has been closed and a table needs to be cleaned.
- When a table is cleaned, change the status of a table. This in turn shall show on the floor map that a table is ready for use by the next customer.

Customer:

- Simple, easy to use, Android-based application on restaurant provided touch screen tablet. Use the table to place order in the restaurant computing system.
- View the menu of the restaurant and add menu items to cart
    - As items are added to 'cart', see total price, add notes (e.g. "Steak rare", or "no sauce")
    - Apply coupons or promotions to purchases.
    - Order 'cart' of food when ready. This order is sent to the chef.
- Call over waiter for question / comment on food.
- Request to close the bill for purchase and pay.
- Using their android device, customer should be able to reserve a table for a party of upto twenty people within a three-day time frame, atleast 24hrs prior to the period of reservation.

Due to the fast-paced nature of restaurant environments, the number of keystrokes, clicks, and touches on computer screens should be kept to a minimum. This will ensure an easy to used, efficient system. It is key to remember, the goal of the project is to create a system which is to replace an inefficient, antiquated paper and pen system. If the new system is less efficient, it has failed its design purpose. Functionality, security of the system must not be overlooked.

## 2. Glossary of Terms

1. **Manager** – Controls and oversees all of the business. In charge of editing the menu items available to the customers. The manager is also responsible for assessing restaurant performance.
2. **Customer** – Orders Food and services from the restaurant.
3. **Chef** – Cooks food ordered by the customer.
4. **Waiter** – Assists customer upon request. Waiter is responsible for serving food.
5. **Hostess** - Receives customers and handles reservations.
6. **Clock In/Clock Out** – A system function that allows employees to enter their hours for manager to run payrolls.
7. **Reservation System** – Customers can reserve a table using an android application.
8. **Payroll** – The amount of money each employee will receive for their services. The payroll differs for each employee.
9. **Menu** – List of dishes available in the restaurant. Will be displayed on an android tablet.
10. **Floor Layout** – Shows all tables in the restaurant along with their respective status.(ex: Occupied, Reserved etc.)
11. **User Interface** – The visual on the computer and tablet that allows user interaction with the system. Allows touch screen interaction to order, pay bill, call waiter and view order.
12. **Table Status** – Shows the availability of a table as well as the waiter in charge of serving that table.
13. **Order Status** – Shows whether the order of a particular table is ready to be "served" or "cooking".
14. **Payment** – Customers have the option of payment through cash or credit.
15. **Ads** – Appear on the menu when idle. Manager has options to edit the slides.
16. **Wireless Access Point** – Access points used in home or small business networks are generally small, dedicated hardware devices featuring a built-in network adapter, antenna, and radio transmitter. Access points support Wi-Fi wireless communication standards.
17. **Terminals** – These are terminals used by hostess, manager, chef and waiter running Windows 7.
18. **POS** – Point of Sale System

## 3. System Requirements

### Functional Requirements

| Requirements | PW | Description |
|---|---|---|
| REQ1 | 3 | The system should allow employees to login given a unique identification number to access their respective interfaces. |
| REQ2 | 5 | The system should be able to modify (add, edit, remove) the employee personnel list. |
| REQ3 | 5 | The system should allow orders requested from customer to be displayed on the chef's interface punctually. |
| REQ4 | 2 | The system should handle a customer's request to reserve a table. |
| REQ5 | 3 | The system should display a real time floor plan of the entire restaurant by allowing the hostess and waiters to update the table status. |
| REQ6 | 5 | The system should allow a customer to notify a waiter for assistance through the android tablet instantly. |
| REQ7 | 5 | The system should store the working hours of all employee through a clock-in/clock-out system which consists of a timestamp of when the employee clocked in/out. |
| REQ8 | 1 | The system should allow the chef to update the estimated time of completion of each order once he starts cooking it. |
| REQ9 | 2 | The system should store the menu items on a database with item name, price, description, sides etc. |
| REQ10 | 4 | The system should keep track of all sales income. |

**Table-1: Functional requirements with priority values**

Functional requirements are necessary for developing systems. Table-1 discusses the functional requirements of our system with descriptions and priority weights of each requirement. These requirements state what the system should be able to accomplish. Subset to these requirements is non-functional requirements that are discussed next.

### Non-Functional Requirements

| Requirements | PW | Description |
|---|---|---|
| REQ11 | 4 | The cook's user interface should notify the waiter on completion of an order with 1 screen touch of the root window. |
| REQ12 | 2 | The host's screen should prompt for the reservation confirmation code when a reserved table is selected. |
| REQ13 | 5 | Managerial tasks should be limited to the manager's user interface to prevent potential security breaches. |
| REQ14 | 2 | The customer should be able to make a reservation within 6 screen touches through their android interface. |
| REQ15 | 2 | The login interface should be consistent for all restaurant personnel to maintain consistency in case an employee changes role. |
| REQ16 | 4 | The cook's user interface should have reliability so not more than one |

| | | |
|---|---|---|
| | | order fails on a given day due to system failure. |
| REQ17 | 5 | The menu (Android Interface) should contain text and graphics that describe each item to an average customer. |
| REQ18 | 3 | The host's interface should predict waiters for incoming customers so each waiter has the same workload. |
| REQ19 | 2 | The reservation system should be compatible with any android phone provided it is connected to the internet. |
| REQ20 | 4 | The waiter's interface should be able to distinguish between alerts from chef, customer and host clearly from a distance of 1 meter provided he/she has average eyesight. |

**Table-2: Non-Functional Requirements with priority values**

Non-functional requirements are also essential in system development. There is not a significant distinction between functional and non-functional requirements. Non-functional requirements relates to the superficial conditions needed for the system, such as number of clicks needed to complete a task for individual actors. Table 2-discusses the non-functional requirements of our system.

## Appearance Requirements

| Requirements | PW | Description |
|---|---|---|
| REQ21 | 5 | The host's screen should display the floor layout along with different colors to represent if the table is free, occupied or needs cleaning. |
| REQ22 | 4 | The menu should have a brief description of the item when selected. |
| REQ23 | 5 | The chef's screen should display two-three orders simultaneously so the chef knows what to expect once the current order is closed. |
| REQ24 | 3 | The waiter's screen should have the tables he/she is serving along with order details and chef updates. |
| REQ25 | 2 | The tablets (menus) should run a slideshow of attractive ads to make it more attractive. |

**Table-3: Appearance Requirements with priority values**

Table-3 discusses some of the appearance requirements of our system, which are not as important compared to the functional requirements. Appearance requirements generally mention about the basic appearance of the system, for ex, should the actor be able to read his/her interface 2 feet away from the screen.

# 4. Functional Requirements Specification

## a. Stakeholders

There are six key stakeholders involved in this system: customers, end users, owners of the system, project managers, system analysts, system architects and developers. In our system we, Group 2, will be the system analysts, project managers, system analysts, system architects and developers since we will be performing all the tasks required starting from system design until it is completely implemented and functional. The rest of the stakeholders have different stakes according to their role.

The end users will be those interested in the system's functionality. In our system, the manager, host, waiter and chef are all involved in the daily functions performed by the system. The customers also have comparable stakes in the system as they will be interacting with the system to the same degree as the end-users. Customers also have The owners of the restaurant are important stakeholders in the system as well since they will be investing in our system and any sort of failure of the product will cause them a loss of revenue.

## b. Actors & Goals

| Actors | Goal | Use Cases |
|---|---|---|
| *Manager* | To oversee all restaurant operations. Responsible for all employee payroll performance records. In charge of updating the menu through the addition and removal of menu items. | UC-1, UC-2, UC-3, UC-4, UC-5, UC-6, UC-7, UC-8, UC-9, UC-10, UC-11, UC-12, UC-13, UC-14 |
| *Customer* | To order and pay for food and services. | UC-16, UC-19, UC-20, UC-21, UC-24 |
| *Chef* | To prepare and update the status of food ordered by the customer | UC-1, UC-2, UC-18 |
| *Waiter* | To assist the customer upon request. Responsible for serving food, cleaning tables, and updating the status of a table from unavailable to available | UC-1, UC-2, UC-17,UC-21, UC-22 |
| *Hostess* | To oversee the activity of the restaurant. Responsible for seating customers, assigning waiters to designated areas and updating the status of a table. | UC-1, UC-2, UC-9, UC-18 |

**Table-4: Actor's job goals and the use-cases they utilize**

It is important to understand Actors' roles and goals in order to design a system. Table-4 shows different actors (who will be interacting with the system) along with their goals and the use-cases that are associated with each actor. Goal descriptions explain each actor's role within the system along with the use-cases each actor utilizes.

## c. Use Cases

### i. Casual Description

| Use Case | Description |
|---|---|
| 1. Login(UC-1) | Staff enters unique ID and password information which is verified by the system.<br>Fulfills REQS: 1 |
| 2. EnterHours(UC-2) | Employees record the number of hours they have worked each day.<br>Fulfills REQS: 7 |
| 3. ManageEmployees(UC-3) | Manager adds, remove, or modifies each employee in the database.<br>Fulfills REQS: 2 |
| 4. AddEmployee(UC-4) | Manager adds a new employee in the database.<br>Fulfills REQS: 2 |
| 5. RemoveEmployee(UC-5) | Manager removes an existing employee from the database.<br>Fulfills REQS: 2 |
| 6. EditEmployee(UC-6) | Manager edits employee information.<br>Fulfills REQS: 2 |
| 7. ManageMenu(UC-7) | Manager adds, remove, or modifies items on the menu.<br>Fulfills REQS: 9 |
| 8. AddMenuItem(UC-8) | Manager adds a new item in the menu.<br>Fulfills REQS: 9 |
| 9. RemoveMenuItem(UC-9) | Manager removes an existing item from the menu.<br>Fulfills REQS: 9 |
| 10. EditMenuItem(UC-10) | Manager edits menu item information.<br>Fulfills REQS: 9 |
| 11. ManageLayout(UC-11) | Manager adds/removes tables and edits the floor plan of the restaurant.<br>Fulfills REQ: 5 |
| 12. ManageAds(UC-12) | Manager adds, remove, or modifies ads of the slide show while the menu is idle.<br>Fulfills REQS: 9 |
| 13. ViewPayroll(UC-13) | Manager views hours logged by each employee and the wages owed based on these hours.<br>Fulfills REQS:7 |
| 14. ViewSales(UC-14) | Manager views sales records by month, week, and day.<br>Fulfills REQS:10 |
| 15. TableStatus(UC-15) | Hostess assigns waiters to tables based on availability.<br>Fulfills REQS: 18 |
| 16. OrderFood(UC-16) | Customer views menu and orders desired meals.<br>Fulfills REQS: 17 |
| 17. CancelOrder(UC-17) | Waiter cancels an item in Order already sent to chef.<br>Fulfills REQS: 6 |
| 18. ServeFood(UC-18) | Chef notifies waiter when food is ready to be served.<br>Fulfills REQS: 11 |
| 19. CallWaiter(UC-19) | Customer calls waiter for assistance. |

| | |
|---|---|
| | Fulfills REQS: 6 |
| **20. PayBill(UC-20)** | Customer views total cost of ordered meals and calls waiter to proceed to bill payment.<br>Fulfills REQS: |
| **21. CashPayment(UC-21)** | Waiter pays for ordered meals using cash payment.<br>Fulfills REQS: |
| **22. CardPayment(UC-22)** | Waiter pays for ordered meals through card payment.<br>Fulfills REQS: |
| **23. SendReservation(UC-23)** | Customer views available reservation times and then sends a request for a reservation.<br>Fulfills REQS: 4 |
| **24. CancelReservation(UC-24)** | Customer cancels reservation.<br>Fulfills REQS: 4 |

**Table-5: Use-Cases and their Description**

## ii. Use Case Diagram

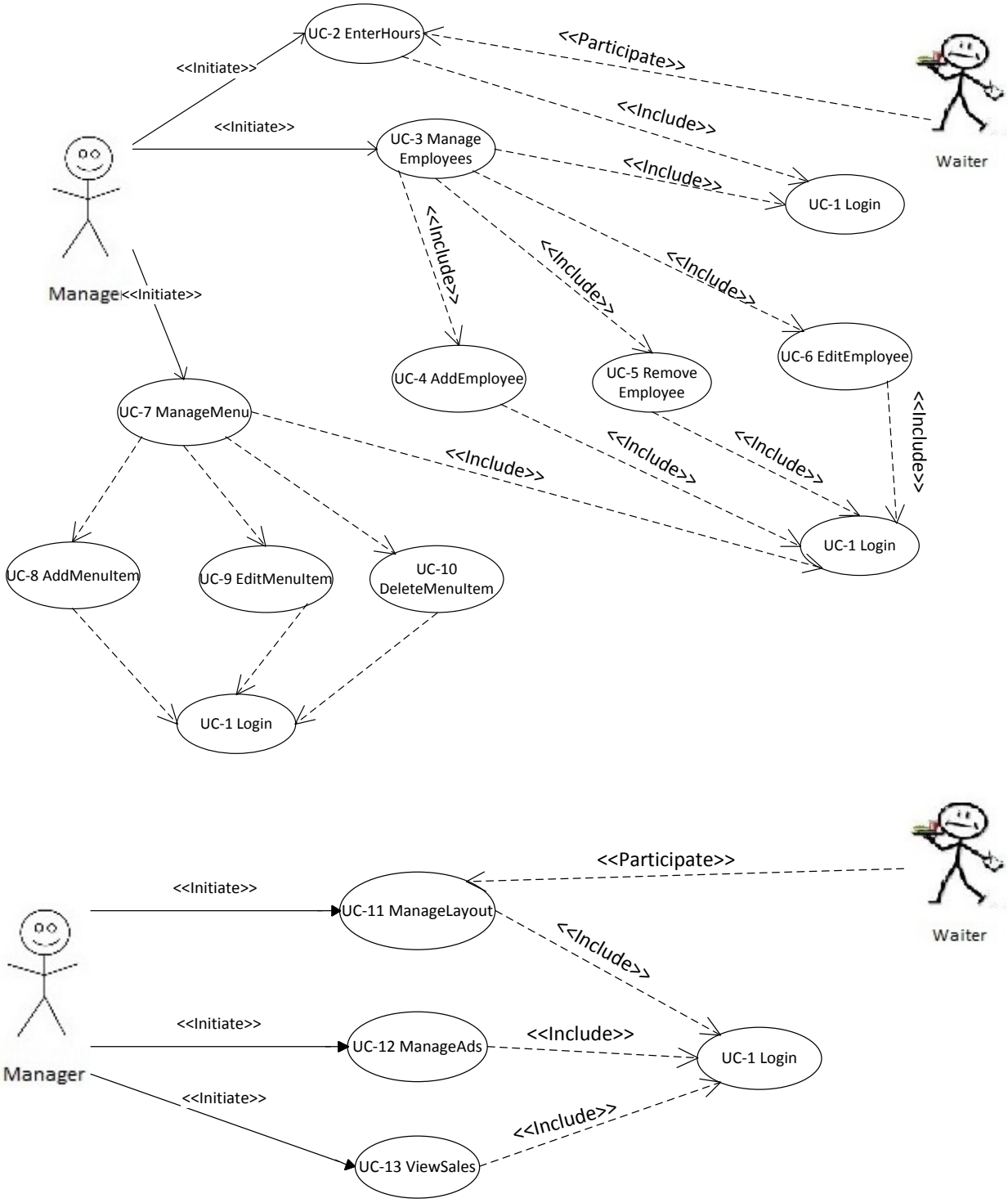



**Fig-2: Manager's Use-Case Diagrams**

**Note: Although the associations are not shown for "UC-7: Manage Menu", it should be clearly visible that the associations are repetitive throughout the image, that is, ManageMenu <<includes>> UC-8, UC-9, and UC-10, and UC-1. UC-8, 9, and 10 also include UC-1: Login**

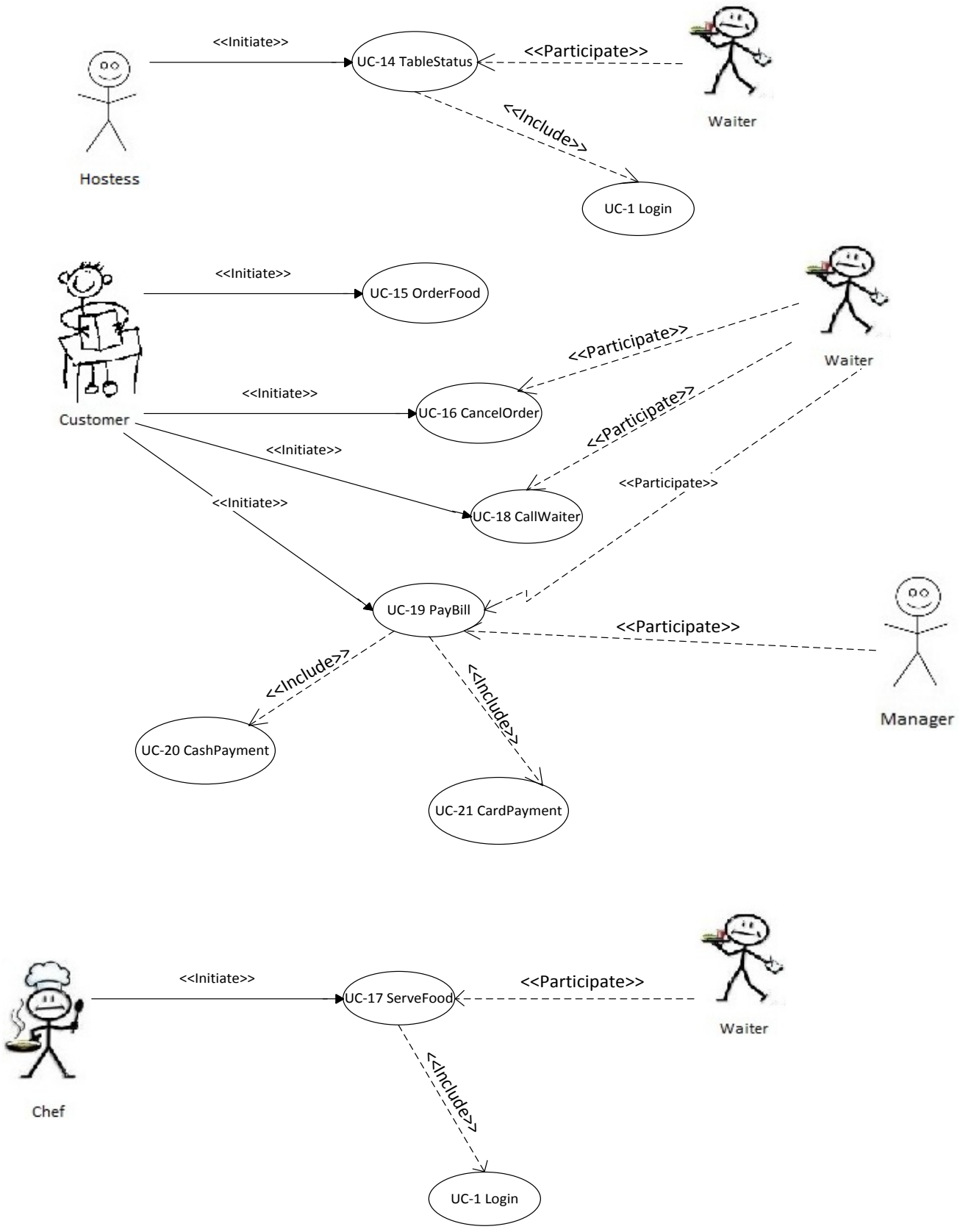



**Fig-3: Use-Case Diagram of Individual actors: Hostess, Chef, and Customer**

Use-Case diagrams are an essential part of software engineering. It provides an overview of the interaction between the actors and the system parts. Fig-2 and Fig-3 displays the use-case diagrams involving different actors. Fig-2 shows utilities the manager is associated with, that is, different actions can be initiated by the actor which leads to accomplishing tasks. The solid lines at the beginning tell us different actions that can be initiated by the particular actor. The dashed lines usually display different associations, such as <<participation>> and <<inclusions>>. The action verbs used in the diagrams are self-explanatory; the reader can understand what actors can initiate the particular tasks and which tasks require participation of other actors. All the use cases cannot be initiated because they are sub-sections of the main use-case, which is why they are considered <<include>>, which proposes plausible tasks an actor can complete. Most of the use-cases <<include>> “UC-1: Login”, which means that the actor must login to complete any tasks they can initiate. Other “included” use-cases are just sub-sections of the main use-case as mentioned before; in order to complete tasks utilizing the sub-sections, the actor must login into the system. UC-1 is shown in multiple times in Fig-2 and Fig-3 to make it easy for the reader to understand the entire interaction of the actor with the system; all the use-cases include only one “Login” and therefore should not be confused with its presence in multiple locations. Three use-cases have been eliminated from the previous report because one of the functions was allowing too much power to the manager (UC-13: payroll) and the other two related to the reservation system have been eliminated since they were not implemented and are not as essential compared to the other use-cases.

**It is difficult to present the two images combined in one, as it would create confusion in understand the diagram**

### iii. Traceability Matrix

| Req't | PW | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | X | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 5 | | | X | X | X | X | | | X | X | | | | | | | | | | | | | | |
| 3 | 5 | | | | | | | | | | | | | | | | X | X | | | | | | | |
| 4 | 2 | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 3 | | | | | | | | | | | X | | | | X | | | | | | | | X | X |
| 6 | 5 | | | | | | | | | | | | | | | | | | | X | X | | | | |
| 7 | 5 | | X | | | | | | | | | | | X | X | | | | | | | | | | |
| 8 | 1 | | | | | | | | | | | | | | | | | | X | | | | | | |
| 9 | 2 | | | | | | | X | X | | | | X | | | | | | | | | | | | |
| 10 | 4 | | | | | | | | | | | | | | | | | | | | X | X | X | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| Max PW | | 5 | 5 | 5 | 5 | 5 | 5 | 2 | 2 | 5 | 5 | 3 | 2 | 5 | 5 | 3 | 5 | 5 | 1 | 5 | 5 | 4 | 4 | 3 | 3 |
| Total PW | | 5 | 5 | 5 | 5 | 5 | 5 | 2 | 2 | 5 | 5 | 3 | 2 | 5 | 5 | 3 | 5 | 5 | 1 | 5 | 9 | 4 | 4 | 3 | 3 |

### iv. Fully Dressed Description

We have only done 2 major fully dressed descriptions of use cases due to time constraints. But if time permitted we would expand each use case in such a way.

| Use Case UC-1 | Login |
|---|---|
| **Related Requirements:** | REQ1 |
| **Initiating Actor:** | Hostess, Waiter, Manager, Chef |
| **Actor's Goal:** | To successfully login to the system and perform restaurant operations |
| **Participating Actors:** | |
| **Preconditions:** | Access to a computer connected to system network |
| **Postconditions:** | User successfully logged in to system |
| **Failed End Condition:** | Login Failed |

**Flow of Events for Main Success Scenario:**

→ 1. User selects Login button
← 2. System prompts for a unique identification number
← 3. User enters information into designated field
← 4. System validates user input and displays proper interface of that user

**Flow of Events for Extensions (Alternate Scenarios):**

4 a. System could not validate log information and sends a login failed error

| Use Case UC-8 | AddMenuItem |
|---|---|
| **Related Requirements:** | REQ9 |
| **Initiating Actor:** | Manager |
| **Actor's Goal:** | To successfully add a new menu item to Menu |
| **Participating Actors:** | |
| **Preconditions:** | Manager receives a new item from corporate to be added to the menu.<br>User is authorized as Manager and has logged in successfully: Include Login (UC-1) |
| **Postconditions:** | System successfully updated Menu |
| **Failed End Condition:** | Menu failed to update |

**Flow of Events for Main Success Scenario:**

→ 1. Manager selects Add Menu Item option from ManageMenu interface.

← 2. System prompts manager to fill out information required for creating a new menu item (Name, description, Side Dishes, Cooking Time, URL where chef can find the recipe, date that item will be added to the menu)

→ 3. Manager enters all required information and hits the next button.

← 4. (a)System validates the entered data fields. (b)System prompts manager to choose ingredients required for the meal along with quantities.

→ 5. Manager enters all required information and hits the Next button.

← 6. (a)System confirms all information is valid. (b)System confirms that all ingredients are in inventory.

←7. (a)System retrieves the stored data fields (i.e. name, description etc.). (b)System looks up cost of each ingredient and calculates food cost for the item and displays to the manager. System verifies the price of the item through the "Uniform System of Accounts for Restaurants" and prints suggested price along with the profit margin. System also prompts manager to edit the price if necessary.

→8. Manager confirms price and hits the Add Item button.

←9. (a) System validates new menu item information. (b) Updates Menu on the projected date and removes item from potential list.

**Flow of Events for Extensions (Alternate Scenarios):**

3 a. End-User selects Cancel option

← 1. (a) System does not add menu item to the potential item list. (b) returns to previous Interface

4 a. System detects a letter element in price data field

← 1. System notifies end-user that data field is invalid or missing

b. System detects a data field is empty

← 1. System notifies end-user that data field is invalid or missing

5 System recognizes a required ingredient is not in the restaurant inventory.

→ 1. System sends a warning to manager saying that ingredient is not in inventory and make sure to add it to the next shipment list.

| Use Case UC-16 | OrderFood |
|---|---|
| **Related Requirements:** | REQ3 |
| **Initiating Actor:** | Customer |
| **Actor's Goal:** | To select desired meal items from the food menu |
| **Participating Actors:** | Chef, Waiter |
| **Preconditions:** | Customer is seated and ready to order |
| **Postconditions:** | Customer has placed the order. |
| **Failed End Condition:** | Customer decides to leave without completing the order. |

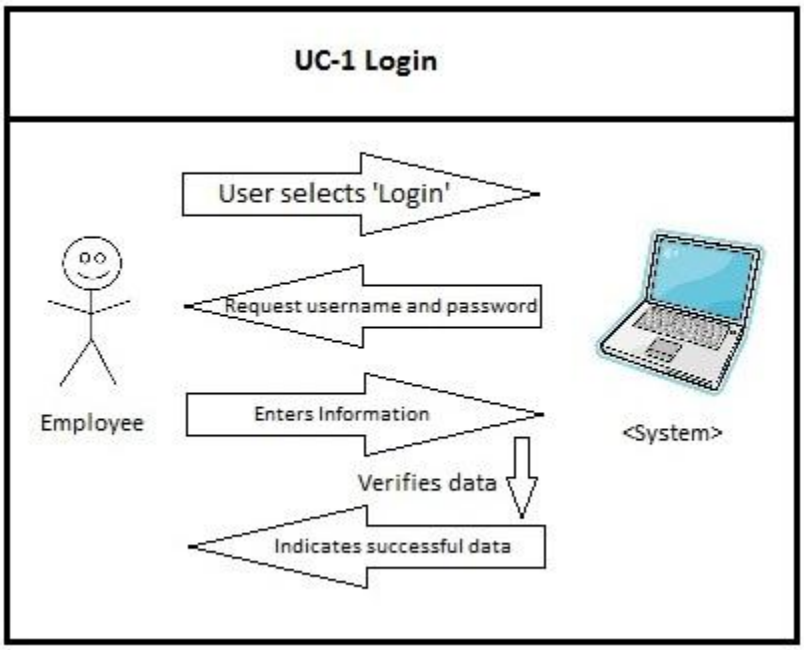
**Flow of Events for Main Success Scenario:**

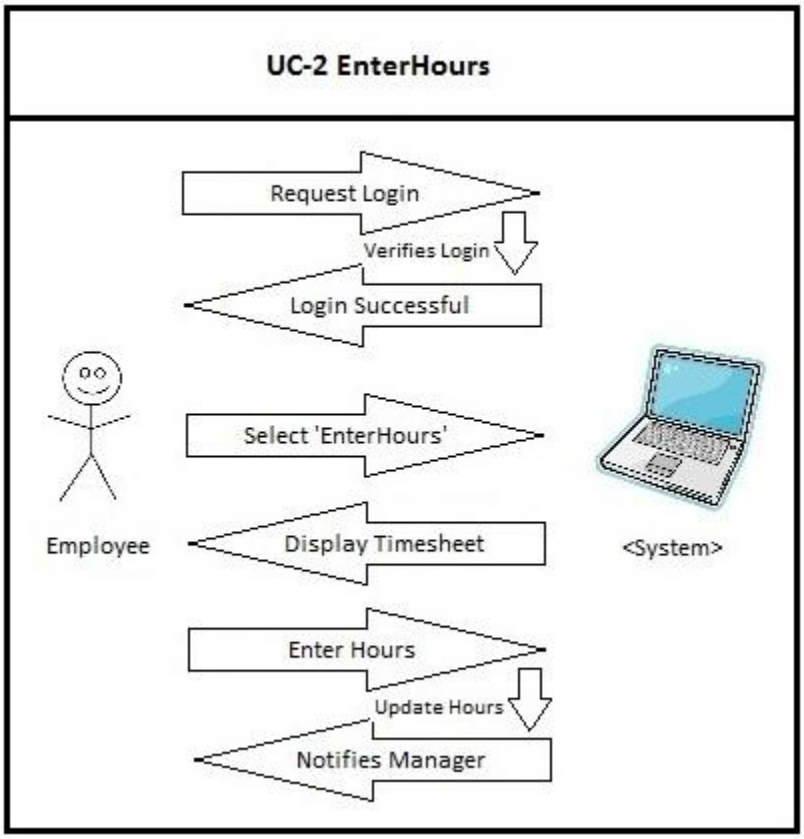
→ 1. Customer selects "View Menu" from the main screen
← 2. System displays a categorized list of food items available to order based on ingredients available.
→ 3. Customer (a) selects food item of his/her choice and specifies the desired side meal as well as any Notes for the Chef (b) Selects "Add To Order" option
← 4. (a)System check availability of ingredients of the order. (b) System adds customer's meal to order list, if ingredients are available.
→ 5. When finished adding meals, Customer selects "View Order" option.
← 6. System displays a list of all meals selected in Customer's Order with options to Remove individual items from the Order.
→ 7. Customer verifies Order and selects "Order Now" option.
← 8. (a)System confirms the order and sends the Table Number, Time Stamp, and each Meal Name, Side and Chef Notes from the Order to the Chef's Order Queue. (b) System subtracts the ingredients required by the order from inventory.
←9. (a)System stores order information on a database for future payment processing. (b) System notifies user that the order was placed successfully.

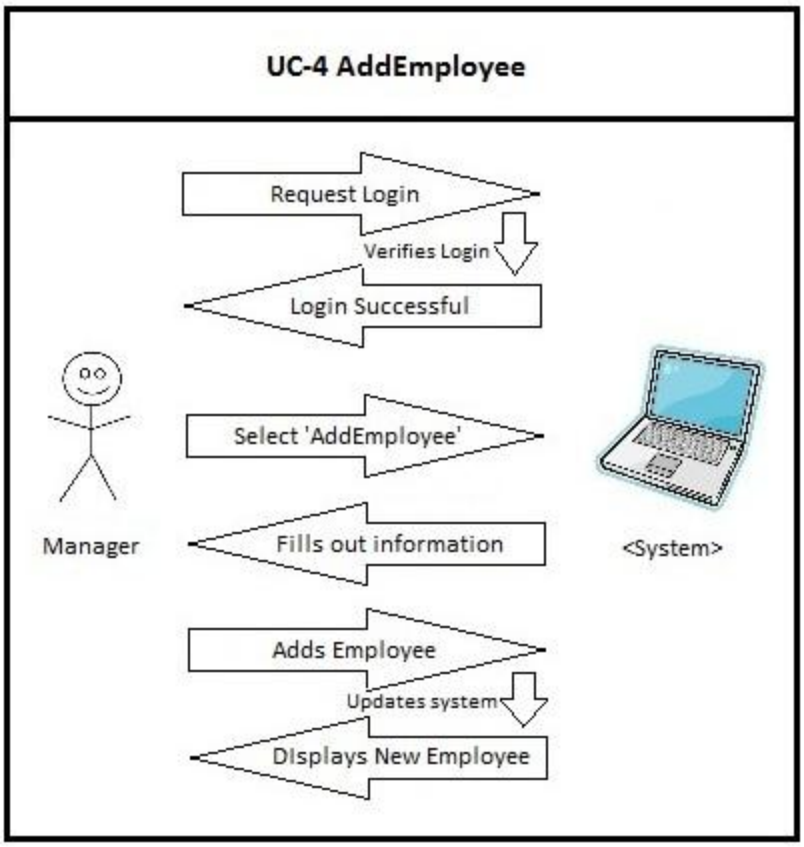
**Flow of Events for Extensions (Alternate Scenarios):**

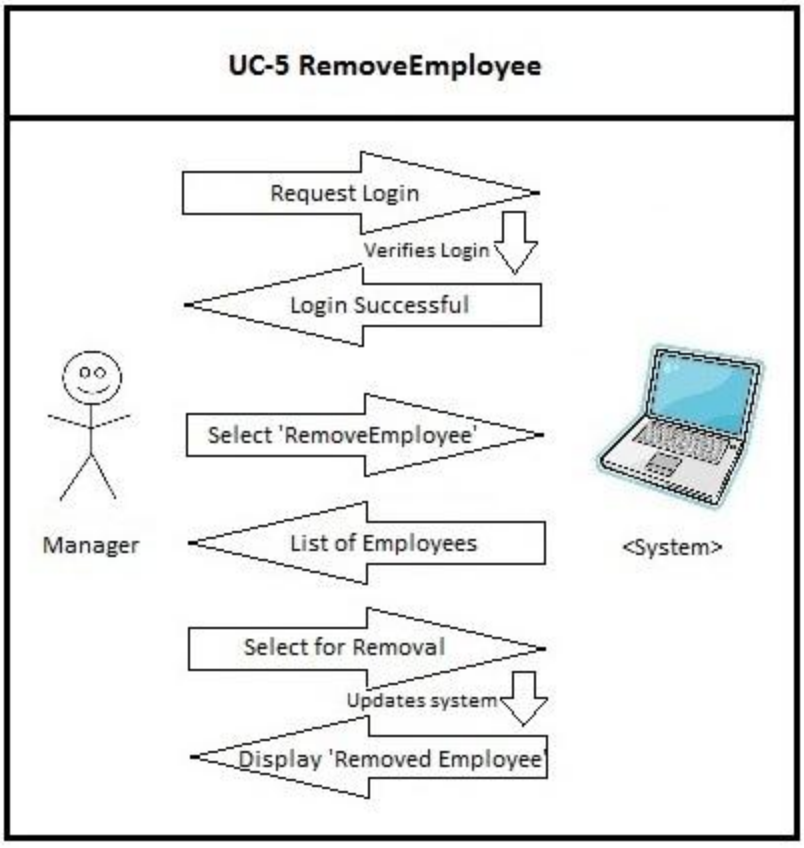
3/5 Customer can selects the cancel order option.
← 1. System does not send order and returns customer to main screen.
4 (a) Ingredients for the items ordered are unavailable.
← 1. System notifies user that item selected cannot be ordered due to a shortage in inventory.
← **2**. System does not add item to the order.
4 (b) System recognizes that customer did not specify a side.
← 1. System notifies user to select a side before the item can be added to the order.
7 Customer decides to remove an item from the order by selecting the remove button next to the item.
← 1. System removes the item from the order.

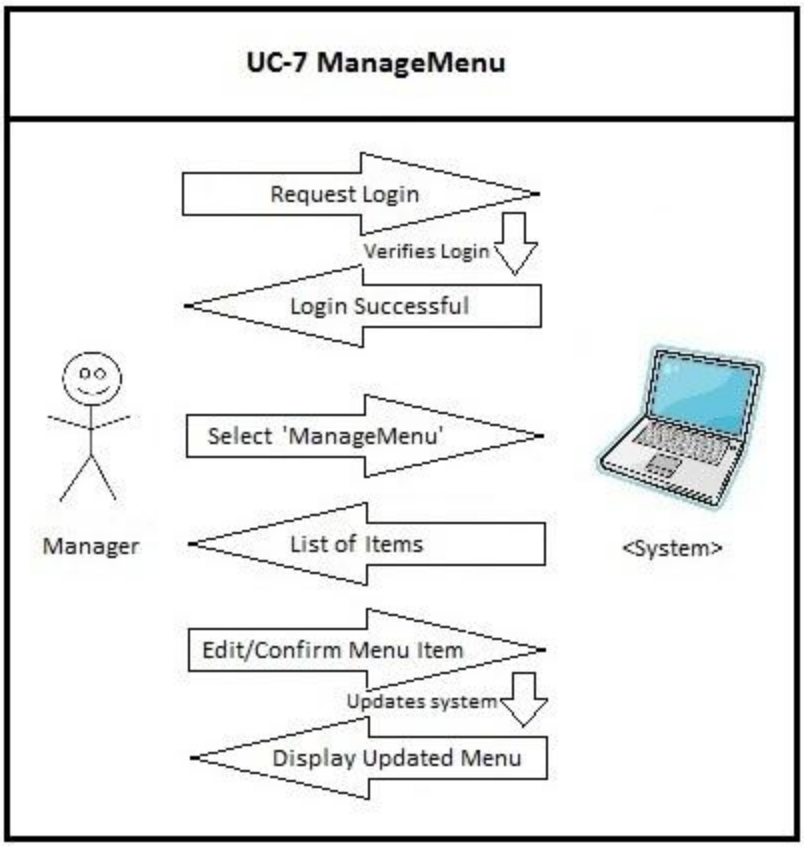
## d. System Sequence Diagrams

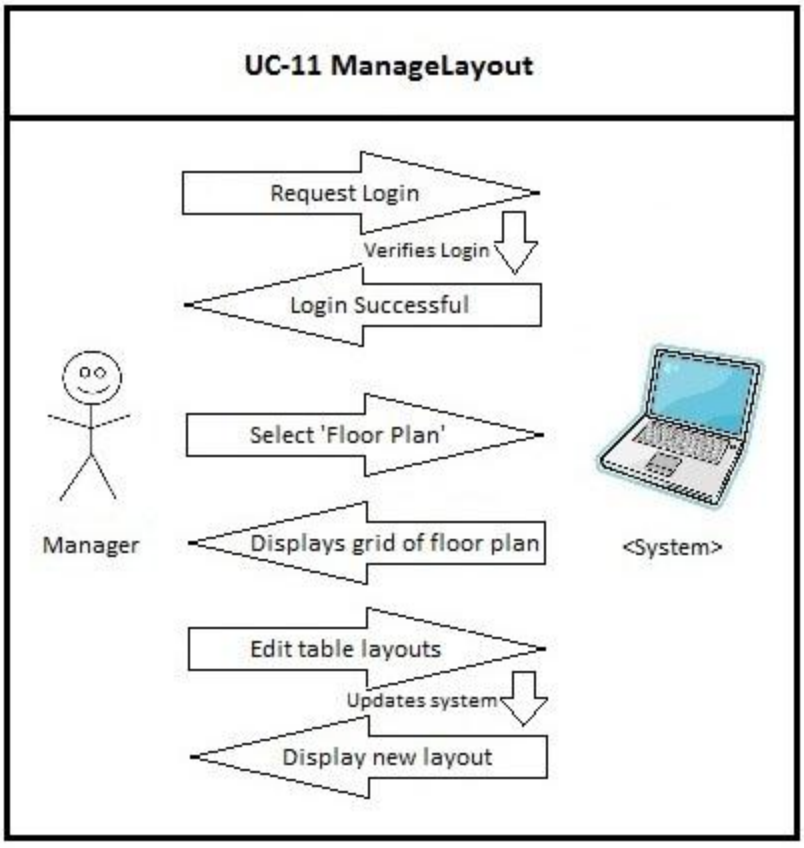


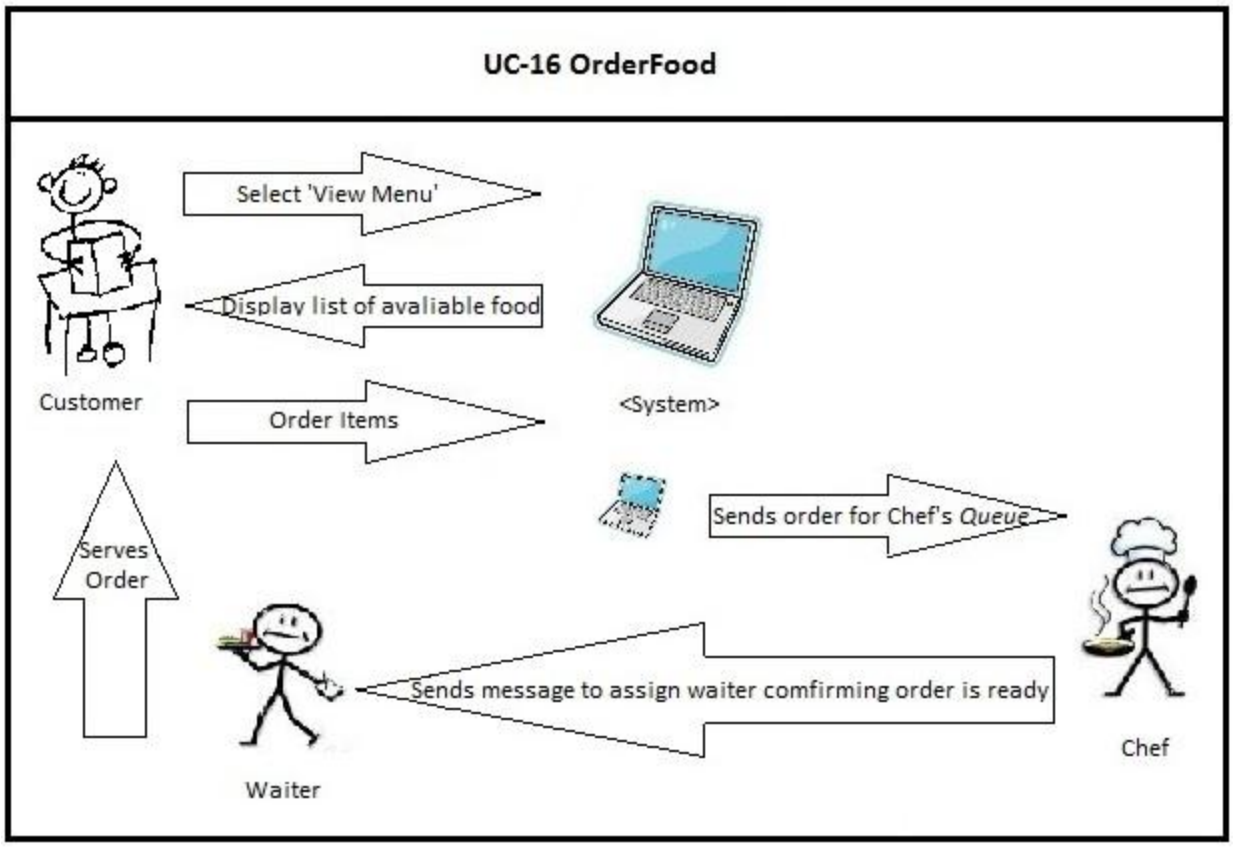



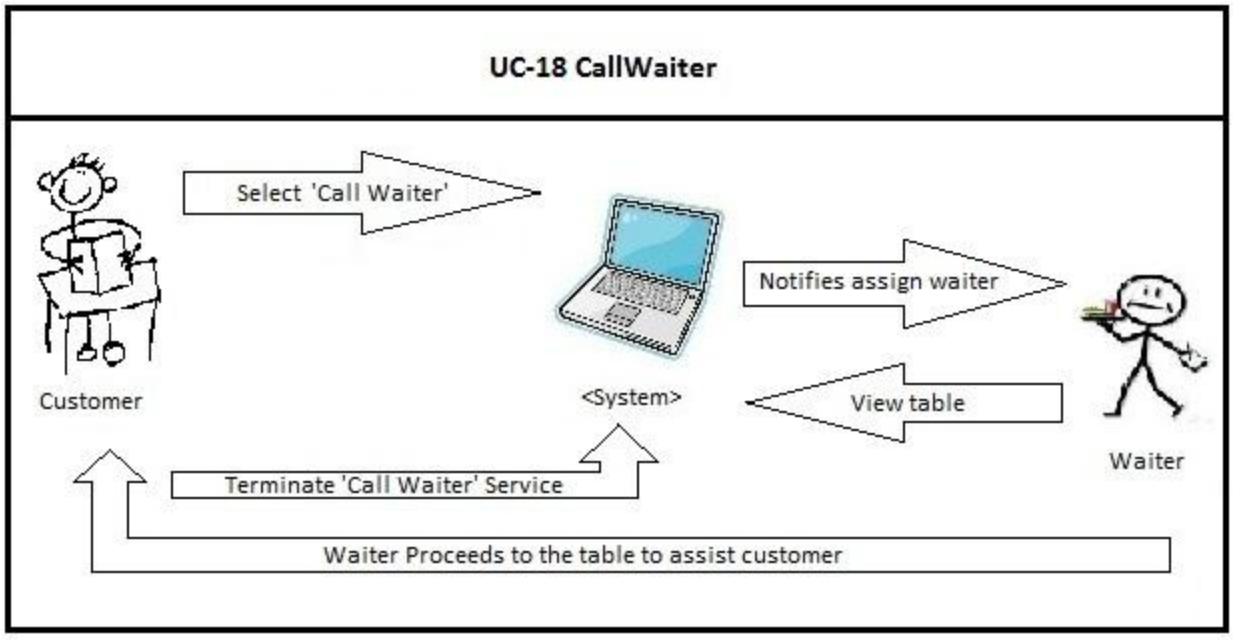

UC-4 AddEmployee
Request Login
Verifies Login
Login Successful
Select 'AddEmployee'
Fills out information
Adds Employee
Updates system
DIsplays New Employee
Manager
<System>

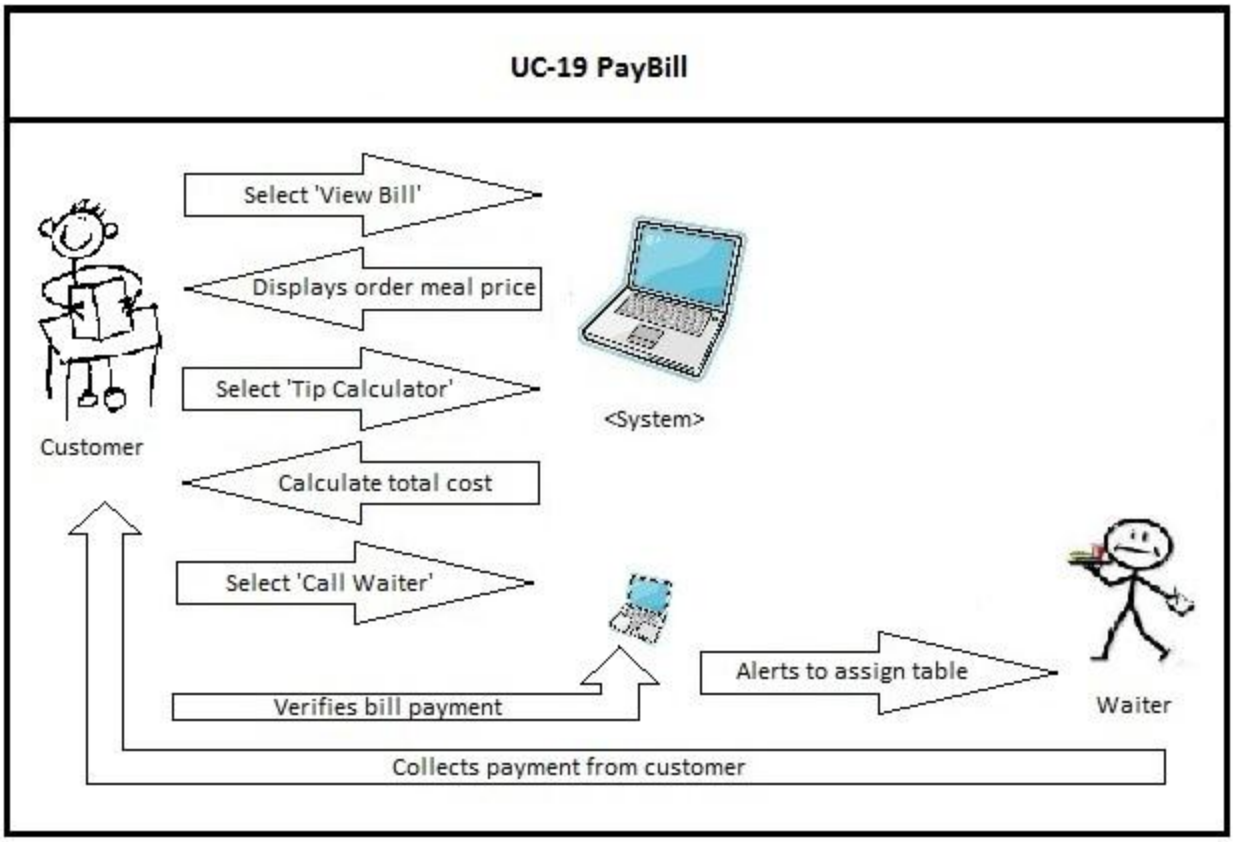

UC-5 RemoveEmployee
Request Login
Verifies Login
Login Successful
Select 'RemoveEmployee'
List of Employees
Select for Removal
Updates system
Display 'Removed Employee'
Manager
<System>

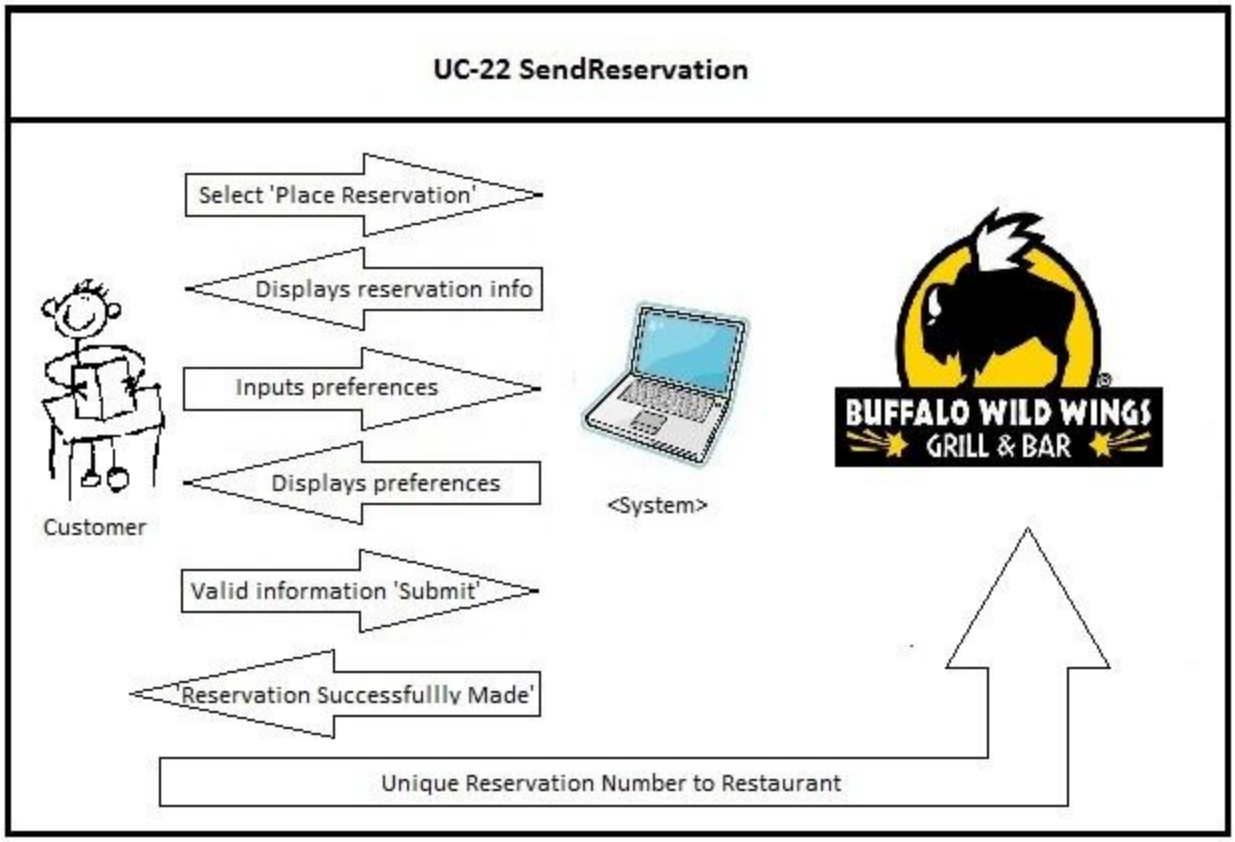

UC-7 ManageMenu
Request Login
Verifies Login
Login Successful
Select 'ManageMenu'
List of Items
Edit/Confirm Menu Item
Updates system
Display Updated Menu
Manager
<System>


UC-11 ManageLayout
Request Login
Verifies Login
Login Successful
Select 'Floor Plan'
Displays grid of floor plan
Edit table layouts
Updates system
Display new layout
Manager
<System>


UC-16 OrderFood
Select 'View Menu'
Display list of avaliable food
Customer
Order Items
<System>
Sends order for Chef's Queue
Serves Order
Sends message to assign waiter comfirming order is ready
Chef
Waiter


UC-18 CallWaiter
Select 'Call Waiter'
Notifies assign waiter
Customer
<System>
View table
Waiter
Terminate 'Call Waiter' Service
Waiter Proceeds to the table to assist customer


UC-19 PayBill
Select 'View Bill'
Displays order meal price
Select 'Tip Calculator'
Calculate total cost
Select 'Call Waiter'
Verifies bill payment
Alerts to assign table
Collects payment from customer
Customer
<System>
Waiter


UC-22 SendReservation
Select 'Place Reservation'
Displays reservation info
Inputs preferences
Displays preferences
Valid information 'Submit'
Reservation Successfullly Made'
Unique Reservation Number to Restaurant
Customer
<System>
BUFFALO WILD WINGS
GRILL & BAR

## 5. Effort Estimation

In order to calculate the user effort required we would first need to give an approximate size estimate/relative size on a 1-10 scale to each use case. We will do so in Table 0.1.

| Use Case | Use Case Points/ Size Estimate |
|---|---|
| 1. Login(UC-1) | 2 |
| 2. EnterHours(UC-2) | 5 |
| 3. ManageEmployees(UC-3) | 5 |
| 4. AddEmployee(UC-4) | 5 |
| 5. RemoveEmployee(UC-5) | 5 |
| 6. EditEmployee(UC-6) | 5 |
| 7. ManageMenu(UC-7) | 5 |
| 8. AddMenuItem(UC-8) | 5 |
| 9. RemoveMenuItem(UC-9) | 5 |
| 10. EditMenuItem(UC-10) | 5 |
| 11. ManageLayout(UC-11) | 6 |
| 12. ManageAds(UC-12) | 2 |
| 13. ViewPayroll(UC-13) | 4 |
| 14. ViewSales(UC-14) | 4 |
| 15. TableStatus(UC-15) | 7 |
| 16. OrderFood(UC-16) | 10 |
| 17. CancelOrder(UC-17) | 1 |
| 18. ServeFood(UC-18) | 3 |
| 19. CallWaiter(UC-19) | 2 |
| 20. PayBill(UC-20) | 1 |
| 21. CashPayment(UC-21) | 1 |
| 22. CardPayment(UC-22) | 1 |
| 23. SendReservation(UC-23) | 10 |
| 24. CancelReservation(UC-24) | 4 |

**Table-0.1: Assigned Use Case Points**

From the above table (Table-0.1) we can see that we have about 102 use case points with the current specification.

Next we know that we have completed use cases 7, 8, 9, 10, 16, 17. And we have partially completed use case 19. From this we can see we have implemented about 36 use case points.

The amount of time it took us to do the implementation was approximately 750 hours. This was also due to the fact that we didn't have an idea about Android SDK or how databases work. So if we cut about 150 hours (30 hours per person) in learning the languages. It took us about 600 hours to implement 36 use case points.

Next we can use the formula to find productivity factor:

Duration = Use Case Points * Productivity Factor

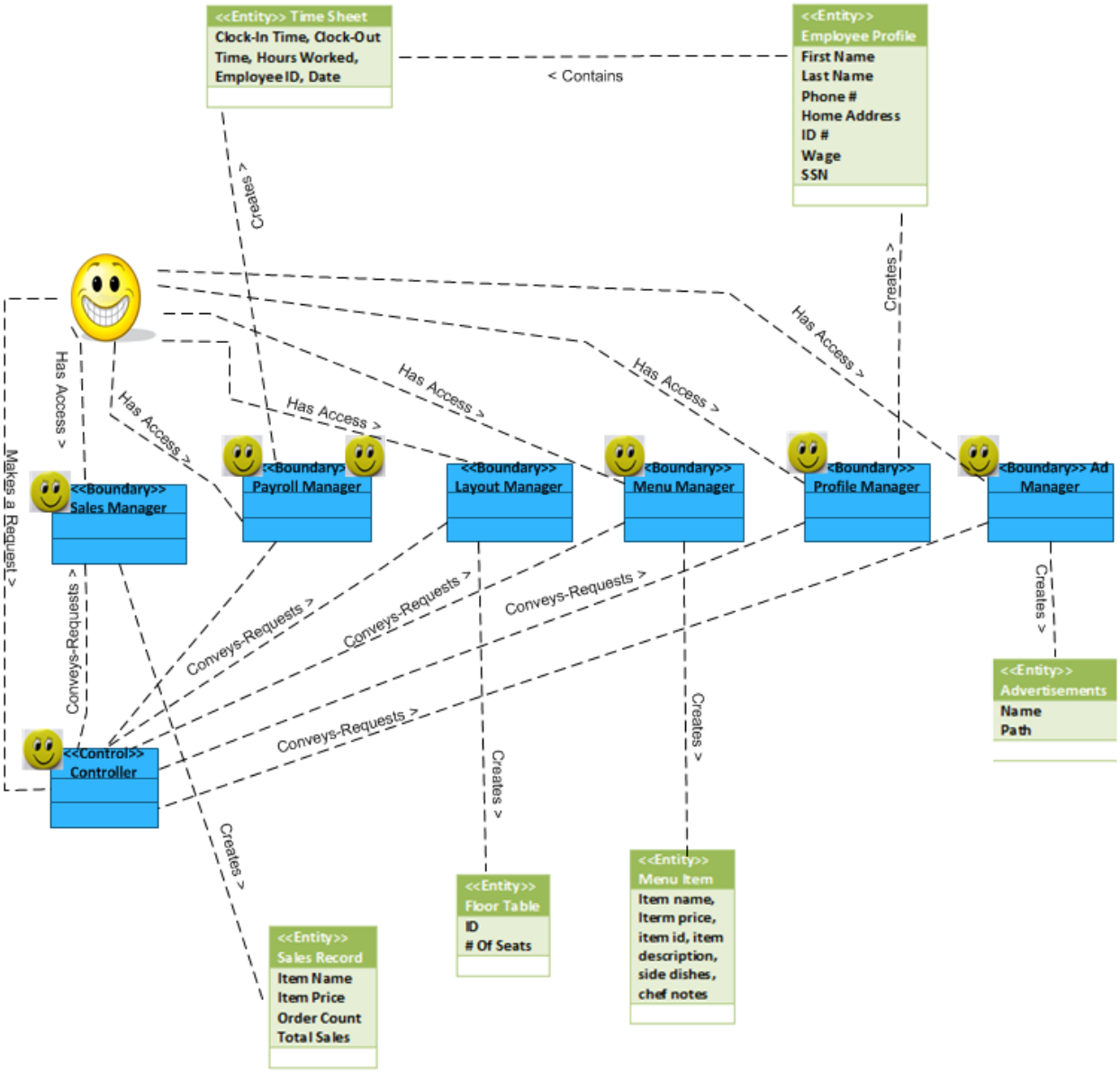
600 = 36 * Productivity Factor

Productivity Factor = 16.67hours

So the duration of the whole specified project would come out to:

102*16.67 = **1700hours**

# 6. Domain Analysis

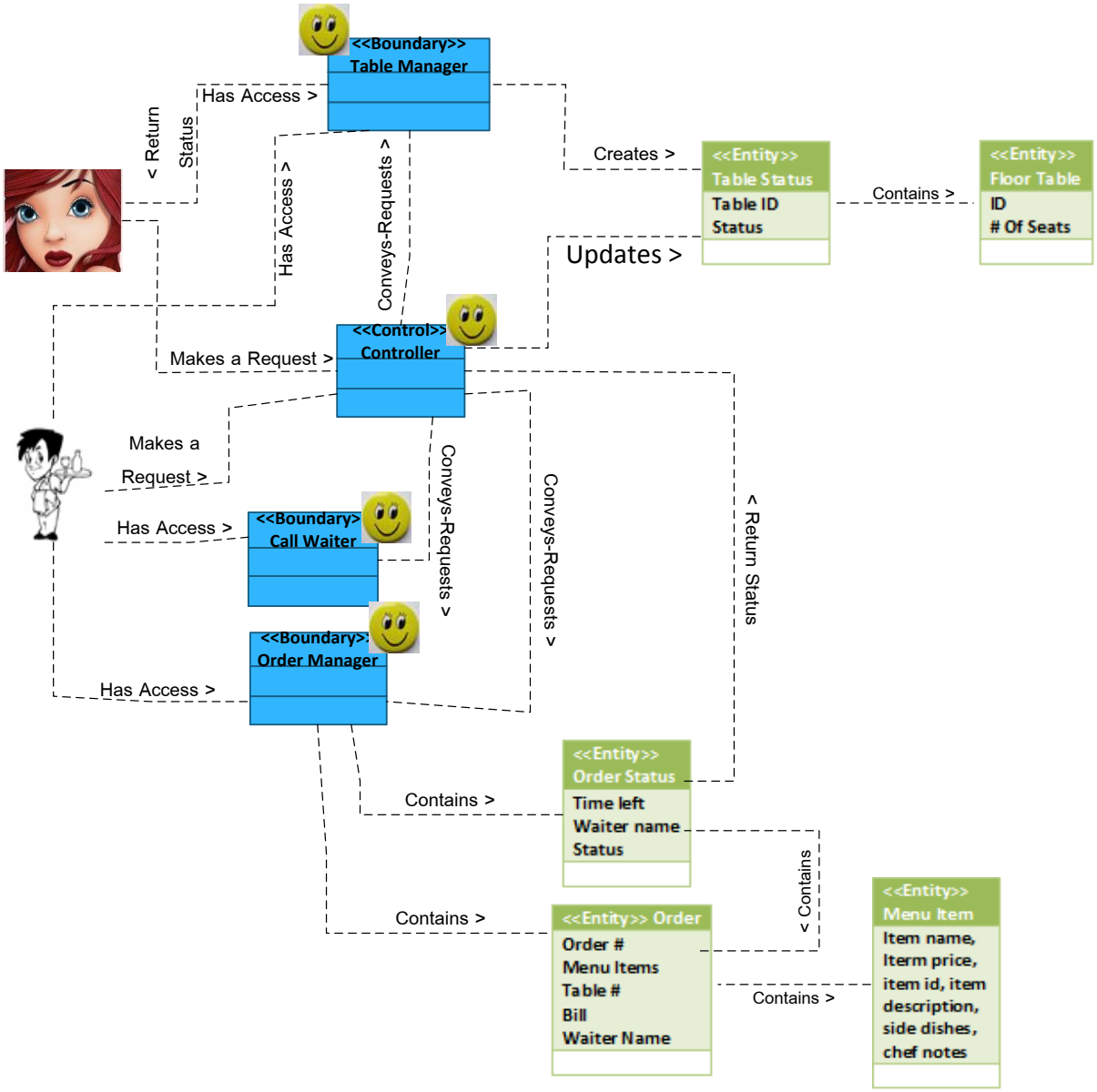



7.

8. **Fig-14: Detailed Manager's Domain Analysis**

9. Domain is quite complex for this particular project, which is why it has been broken up to display the inner working of the domains clearly. Fig-14 shows part of the domain that consists of Manager as the actor and the boundaries the actor has access to. Boundaries in this case are inner workers that help accomplish different actions. The actor (Manager) has access to these boundaries in order to achieve goals of editing and checking. All requests are processed through the controller, which is the main unit (server) that provides access to different components (boundaries) of the system. The controller also allows easy interaction between components. As requests are processed, different entities are created through boundaries that contain a list of information and

are updated constantly through the server as well as the interconnections of different components. As you can see (Fig-14), the actor “Makes a Request” that passes through the controller. The controller then “Conveys the Request” to the appropriate worker (Boundary), which in result creates the appropriate entity, which pops up on the actor’s interface since that is what they requested access to. This entire process is internal; a person cannot see this happening. Workings of hostess and waiter are next.

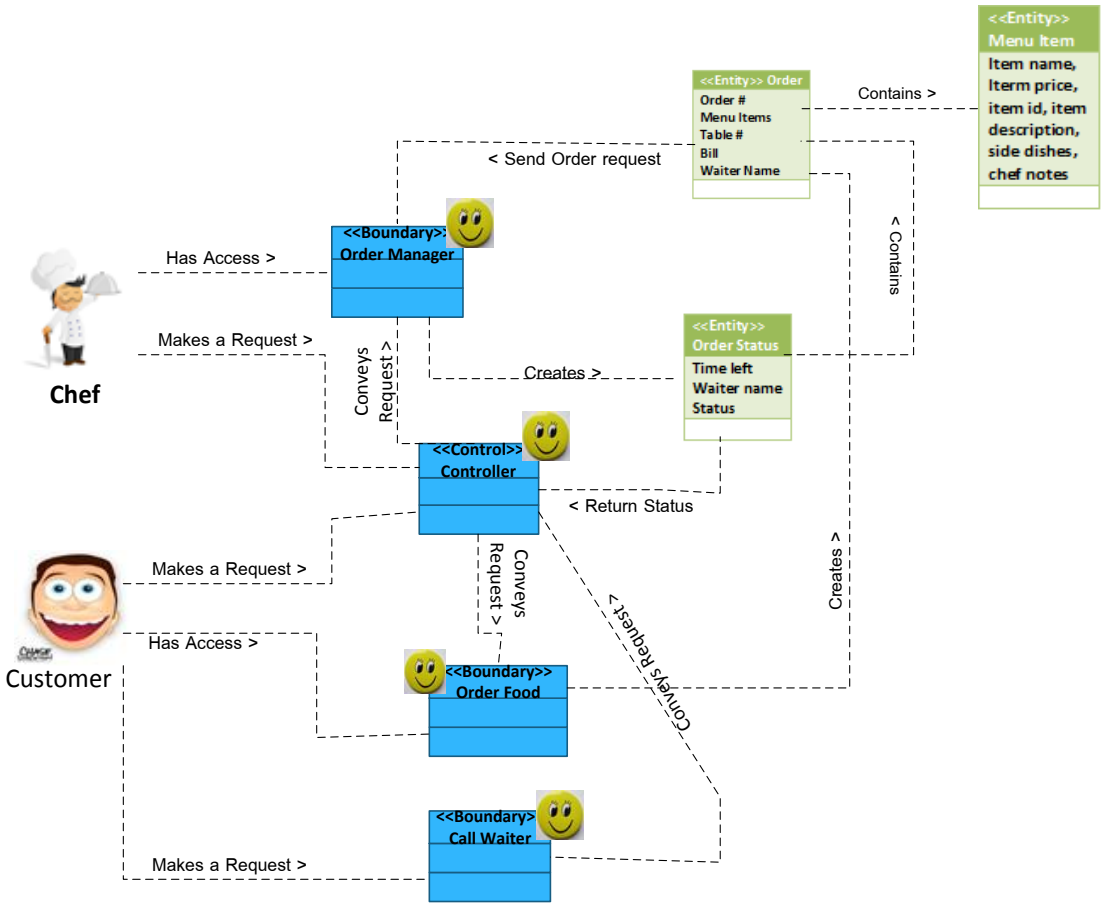



**10.**

**11. Fig-15: Domain Analysis of two actors working together**

12. Concepts of boundaries and entities have been explained in the description of Fig-14. Keep in mind, the controller shown in different pictures is actually just one controller (server). It has been shown in different places so the reader can understand

how different domains are interacting with the controller. In Fig-15, you can see the interaction of the hostess and the waiter in the system. Note that every actor has access to different workers (boundaries), which are connected to the controller (server) that allows the entire system to be interconnected, but with restrictions. Waiter cannot access any other actor's workers and vice-versa. Fig-15 shows that the hostess "makes a request" to access "Table Manager" where she can the "table status" and "floor Table". Although the hostess has access to the table manager, she cannot use it to edit table status; it is editable only by the waiter. The table status is edited by the waiter, which is updated through the controller, which in turn is updated in table manager so the hostess can see the status of different tables. Note that having access to a worker does not mean that the actor can always use the worker to edit information. Sometimes workers are present to just deliver the information. As you can see, the waiter has access to "Call Waiter" and "Order Manager" that contain different entities; the sole purpose of these are to deliver the updated information to waiter, which is provided by the chef's interface (not included in Fig-15). Workings of chef and the customer is next.



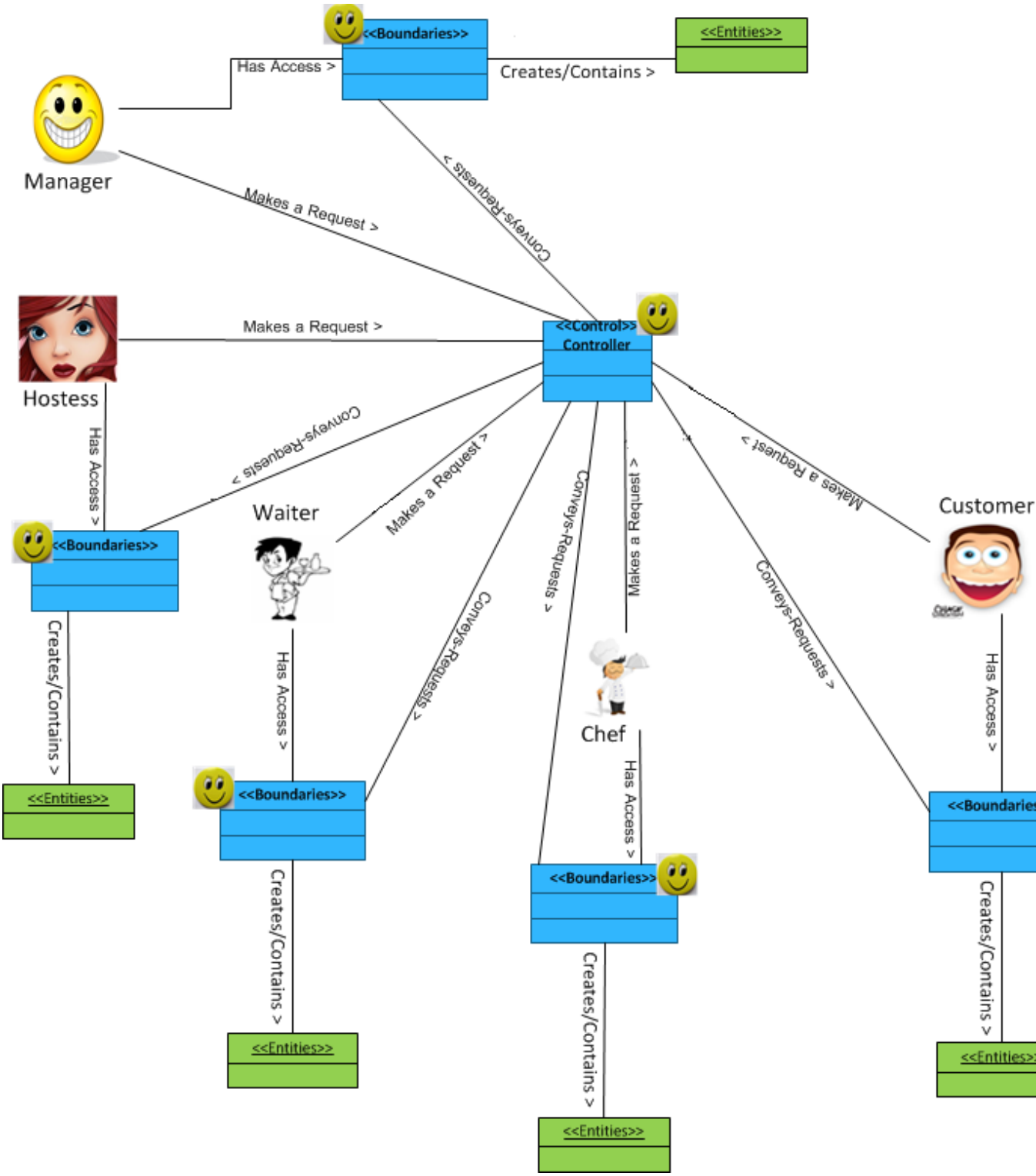

13.

14. **Fig-16: Chef and customer Domain Analysis**

15. Here is another concept where two different actors are working together, but also involve actions where information is sent to another actor's workers. Although two

actors have access to the same worker, their functions are different for each actor. Fig-16 shows that the chef has access to “order Manager”, where the chef sends a request to its worker through the controller to create/update an “order status”, which contains the entity “order” and “Menu Item”. The entity “order status” is transferred to the waiter’s worker named “order manager”, where the waiter can see the order status to serve the food in timely manner. Same concept applies to the customer as an actor. The customer orders food using the server, where the “order food” worker sends the request by creating an entity (contains list of information). The entity can be seen by the chef as well as the waiter (not shown in the picture). Here you see an example of workers having different functions for different actors. A different approach is required when the customer needs to use the “Call Waiter” worker. A request if directly sent to the worker, which conveys the request through the controller, transferring the request to the waiter’s worker, notifying him of his required services. Such processes are internal, and are easily understood through the domain analysis diagrams.

16. **** Reservation concept has been eliminated from these diagrams since it was not implemented. Due to the complexity of the diagrams, it was not possible to present all possible connections legibly, which is why the workings are explained in the text to avoid confusion. An overall picture of the system connections will be provided further in this section****




17.

**18. Fig-17: General overview of the Domains and their associations**

19. Fig-18 provides a general overview of the connections (the associations, such as Has Access, Creates, Makes Request), which would be difficult to show if every connections was displayed. Fig-14, 15, and 16 have been consolidated into Fig-17, that

is, all the workers (Boundaries) are compressed into “<<Boundaries>>” and all the information lists (Entities) are compressed into “<<Entities>>”. This helps reduce the complication of showing all the workers and information lists. The actions are shown right above the connection lines and the “>” shows which way the action is performed. These are the general and uniform actions performed by every actor; detailed information has been mentioned in the descriptions of Fig-14, 15, and 16.

| Description | Type | Concept Name |
|---|---|---|
| **Coordinate actions of concepts associated with this use case and delegate the work to other concepts.** | D | Controller |
| **Coordinates payroll actions, where the manager can check to see the amount of hour worked by an employee on a particular day.** | D | Payroll Manager |
| **Coordinates actions of editing the Menu, where the manager can add, edit, or remove a menu item.** | D | Menu Manager |
| **Provides management of the floor plan, where the manager can add or remove a table.** | D | Layout Manager |
| **Manages the editing of advertisements and organizes them to be displayed in slideshow.** | D | Ads Manager |
| **Manages the list of current employees and their personal information.** | D | Profile Manager |
| **Sends the customized customer order to the chef for processing and to the waiter to receive order status updates.** | D | Order Food |
| **Notifies the waiter of the customer who is in need of assistance.** | D | Call Waiter |
| **Manages the status of a specific order being processed by the chef (began, 5 min, 10 min, done, etc.)** | D | Order Manager |
| **Coordinates the availability status of a specific table (reserved, occupied, etc.)** | D | Table Manager |
| **Organizes sales information of each menu item for the manager to improve services.** | D | Sales Manager |
| **Records the amount of time an employee worked.** | K | Time Sheet |
| **List of all the food items on the menu** | K | Menu Item |
| **List of upcoming menu items and places around the neighborhood.** | K | Advertisements |
| **List of all revenue earned from a particular menu item** | K | Sales Record |

| | | |
|---|---|---|
| **Contains each table's ID number and number of seats** | K | Floor Table |
| **Holds each employee's personal information** | K | Employee Profile |
| **Maintains/Holds each table's availability status** | K | Table Status |
| **Maintains each table's food order status** | K | Order Status |
| **Contains a list of customer's desired menu items** | K | Order |

**20. Table-6: Concept Name and Definitions**

Identifying concepts is part of the domain model and Table-6 lists the responsibilities and the worker titles (concept names) to whom the responsibilities are assigned. In this case, it happens that a single responsibility is assigned to a single worker, but this is not necessarily the case. Complex responsibilities may be assigned to multiple workers and vice versa a single worker may be assigned multiple simple responsibilities. Responsibilities are divided into Doing type and knowing type. Concepts that perform functions are usually under the "Doing" category and concepts that keep track of different information are in the "knowing" category.

| Concept Pair | Association Description | Association Name |
|---|---|---|
| **Controller ↔ AD Manager** | Controller passes requests to AD Manager and displays a list of all Ads on the menu application | Conveys a request |
| **Controller ↔ Profile Manager** | Controller passes requests to Profile Manager and displays a list of profile of all current employees | Conveys a request |
| **Controller ↔ Layout Manager** | Controller passes requests to Layout Manager and displays the restaurant's floor plan | Conveys a request |
| **Controller ↔ Menu Manager** | Controller passes requests to Menu Manager and displays a list of all menu items | Conveys a request |
| **Controller ↔ Payroll Manager** | Controller passes requests to Payroll Manager and displays a timesheet | Conveys a request |
| **Controller ↔ Sales Manager** | Controller passes requests to Sales Manager and displays sales record | Conveys a request |
| **Controller ↔ Table Manager** | Controller passes requests to Table Manager and displays the status of each table | Conveys a request |
| **Controller ↔ Order Manager** | Controller passes requests to Order Manager and displays the | Conveys a request |

| | | |
|---|---|---|
| | status of each table's order | |
| **Controller ↔ Call Waiter** | Controller passes requests to Call Waiter to alert waiter of a customer in need of assistance | Conveys a request |
| **Controller ↔ Order Food** | Controller passes requests to Order Food to send the orders to the Chef | Conveys a request |
| **AD Manager ↔ Advertisements** | Ad manager generates the updated advertisements | Generates/Creates |
| **Profile Manager ↔ Employee Profile** | Profile manager generates the updated employee information | Generates/Creates |
| **Menu Manager ↔Menu Item** | Menu manager creates the updated menu list provided from the manager. | Generates/creates |
| **Layout Manager ↔ Floor Table** | Generates updated floor plans for the waiter and the hostess. | Generates/Creates |
| **Payroll Manager ↔ Timesheet** | Generates and saves employee working hours matching them with employee profile. | Generates/Creates |
| **Sales Manager ↔Sales record** | Generates a record of the revenue for analysis for each menu item. | Generates/Creates |
| **Table Manager ↔Table Status** | Provides the update table availability information by the waiter/hostess. | Generates/Creates |
| **Call Waiter ↔Table Status** | Updates the customer assistance status of a particular table. | Generates/Creates |
| **Order Manager ↔Order** | Sends the customized order to the chef to begin processing and for the waiter to see. | Generates/Creates |
| **Order manager ↔ Order Status** | Updates the status of the customized order by the chef (time remaining). | Generates/Creates |
| **Order food ↔Order** | Provides the new customized order by the customer. | Generates/Creates |

**21. Table-7: Concept Pair and Association Definitions**

Associations describe *who* needs to work together and *why*, not *how* they work together. Table-7 lists the association name and description of each concept pair, which tells the reader about the plausible functions that can be completed for each association.

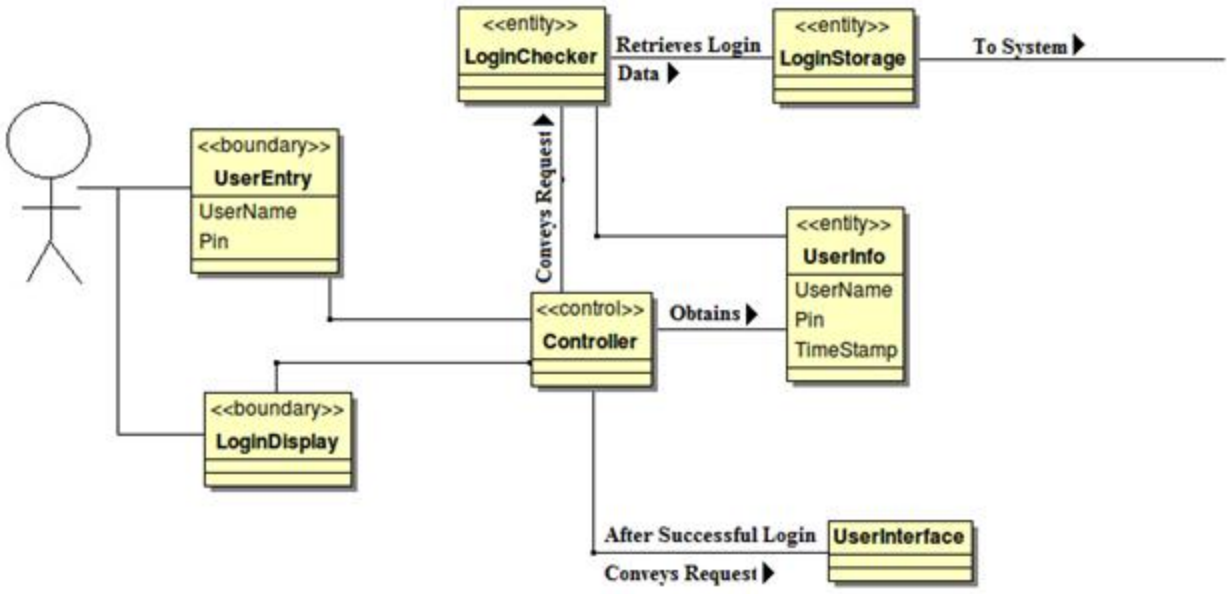



**Fig-18: UC-1: Login Domain Model**

## *Use Case UC-8: AddItem*

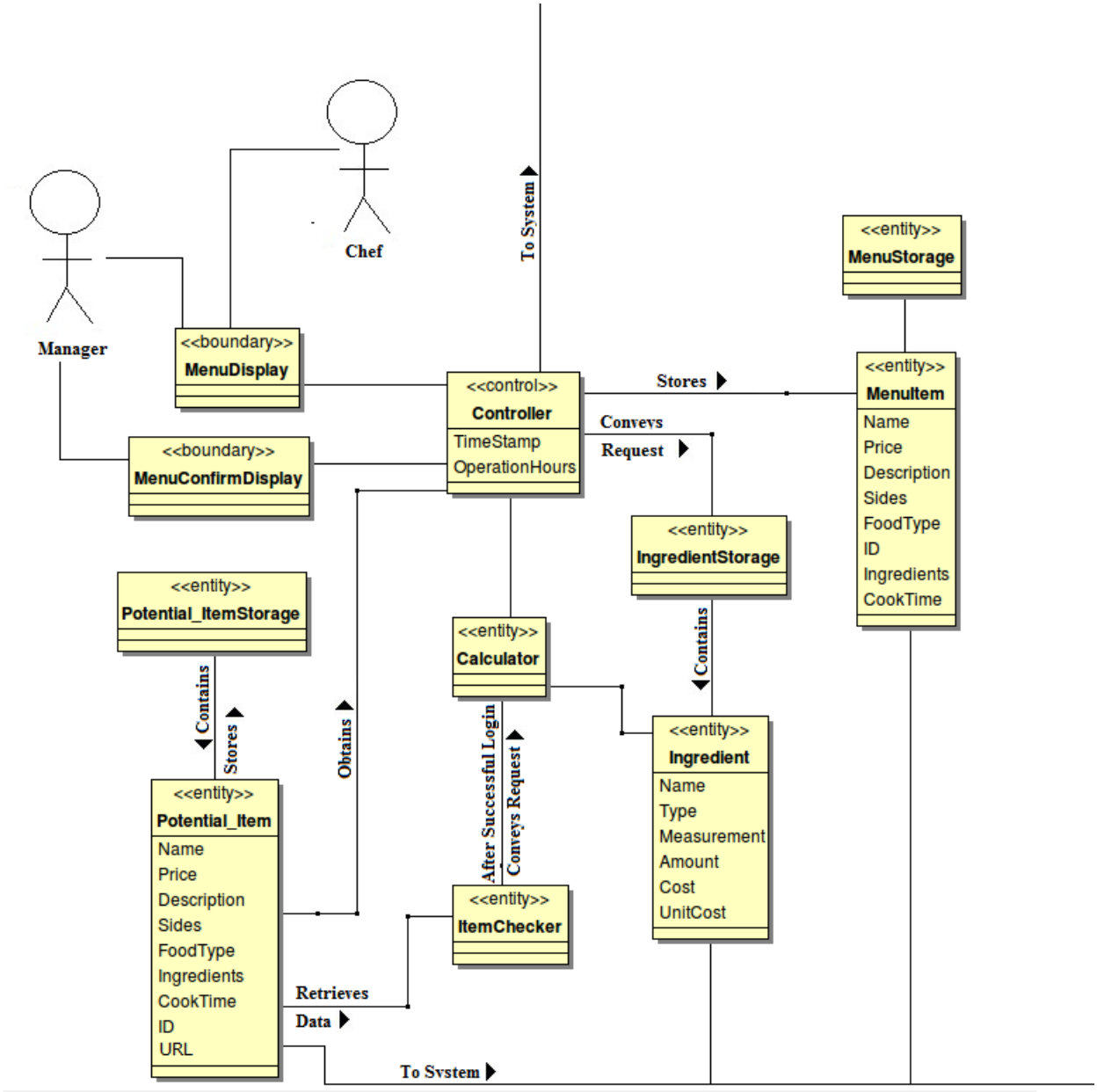



**Fig-19: UC-8: Add Item Domain Model**

## *Use Case UC-15: Order Food*

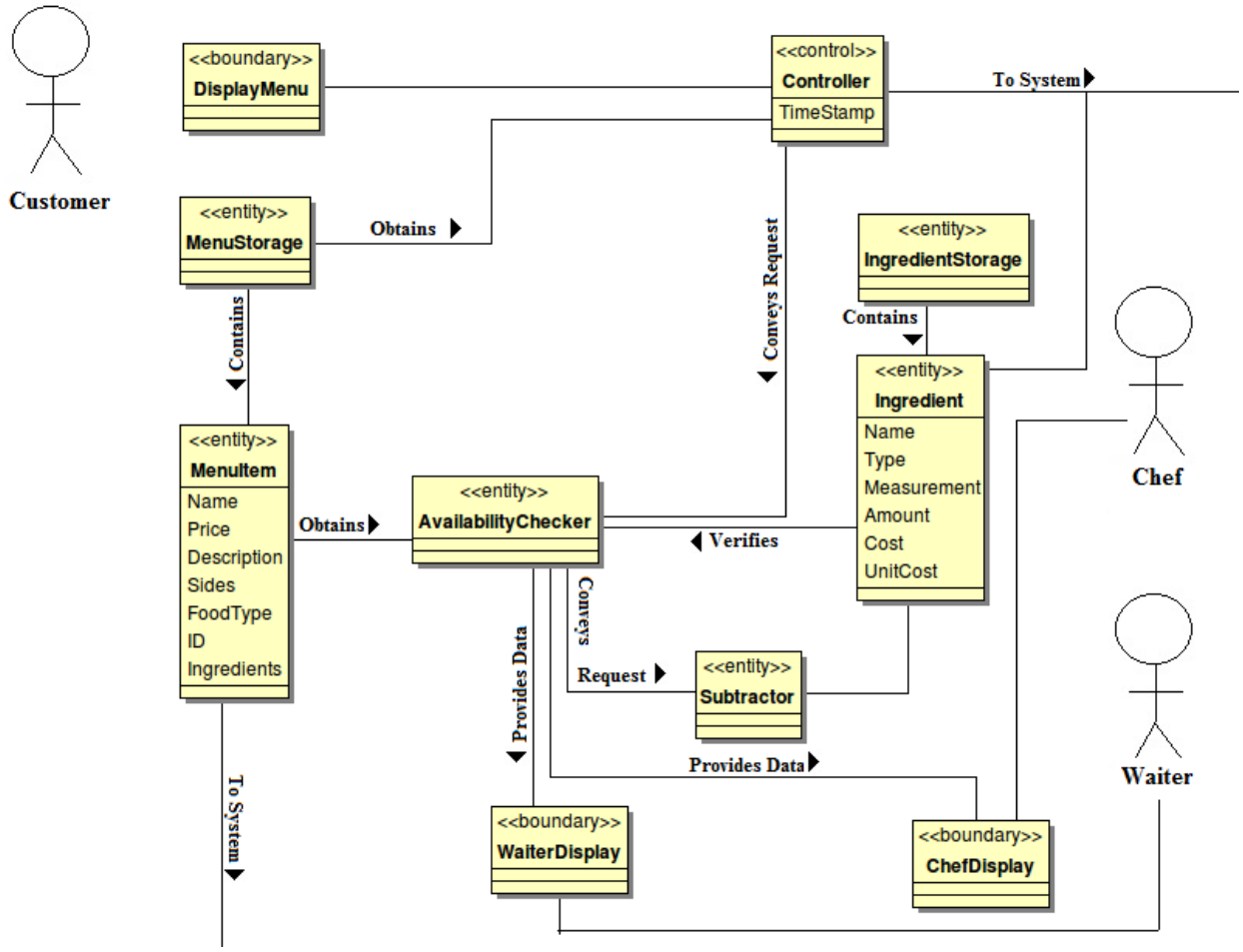



**Fig-20: UC-15: Order Food Domain Model**

## *Traceability Matrix*

Domain Concepts

| | | Controller | Payroll Manager | Menu Manager | Layout Manager | Ads Manager | Profile Manager | Order Food | Call Waiter | Order manager | Tables Manager | Sales Manager | **Time Sheet** | Menu Item | Advertisements | Sales Records | Floor Table | Employee Profile | Table Status | Order Status | Order |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Use Case** | PW | | | | | | | | | | | | | | | | | | | | |
| **UC-1** | 15 | X | X | X | X | X | X | | | | X | X | X | | | | | | | | |
| **UC-2** | 13 | X | X | | | | | | | | | | X | | | | | | | | |
| **UC-3** | 8 | X | | | | | X | | | | | | | | | | | X | | | |
| **UC-4** | 8 | X | X | | | | X | | | | | | X | | | | | X | | | |
| **UC-5** | 8 | X | X | | | | X | | | | | | X | | | | | X | | | |
| **UC-6** | 8 | X | X | | | | X | | | | | | X | | | | | X | | | |
| **UC-7** | 13 | X | | X | | | | | | | | | | X | | | | | | | |
| **UC-8** | 10 | X | | X | | | | | | | | | | X | | | | | | | |
| **UC-9** | 10 | X | | X | | | | | | | | | | X | | | | | | | |
| **UC-10** | 10 | X | | X | | | | | | | | | | X | | | | | | | |
| **UC-11** | 4 | X | | | X | | | | | | | | | | | | X | | | | |
| **UC-12** | 2 | X | | | | X | | | | | | | | | X | | | | | | |
| **UC-13** | 10 | X | | | | | | | | | | X | | | | X | | | | | |
| **UC-14** | 12 | X | | | | | | | | | X | | | | | | | | X | | |
| **UC-15** | 15 | X | | | | | | X | | X | | X | | | | X | | | | | |
| **Uc-16** | 13 | X | | | | | | | | X | | | | | | | | | | X | X |
| **UC-17** | 15 | | | | | | | | X | | | | | | | | | | | | X |

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **UC-18** | 13 | X | | | | | | | X | | | | | | | | | | | |
| **UC-19** | 13 | X | | | | | | | X | | | | | | | X | | | | |
| **UC-20** | | | | | | | | | | | | | | | | X | | | | |
| **UC-21** | | | | | | | | | | | | | | | | X | | | | |

**Table-8: Use-cases-to-domain-model traceability matrix. (PW=Priority Weight)**

# 7. Interaction Diagrams

## *Use Case-2: Enter Hours*

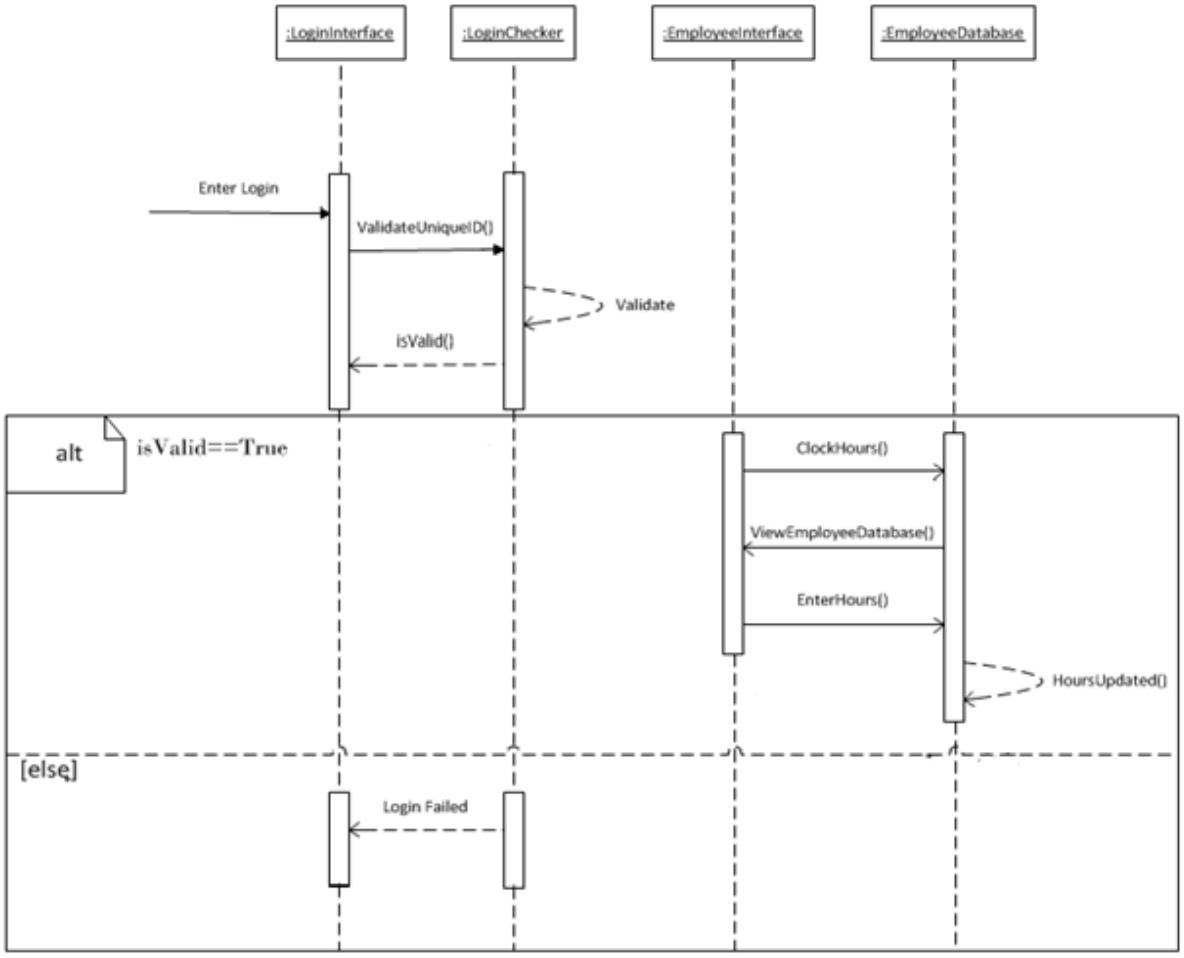



**Fig-21: UC-2: Enter Hours Interaction Diagram**

Responsibilities in Fig-21:

1. Employee begins to login.
2. The system checks the validity of the login by the Employee. If isValid()==True then moves to next step.
3. Alternative: System failure. Employee login is denied access to the system.
4. Employee selects to input hours for employee using the clockHours() method.
5. The system displays the employee database with the method veiwEmployeeDataBase().
6. Employee enters time information using the method EnterHours() to the complete the timesheet.
7. The hours enter from the employee will be saved and viewed by the manager so he knows how much each employee will be getting paid.

## *Use Case-4: Add Employee*

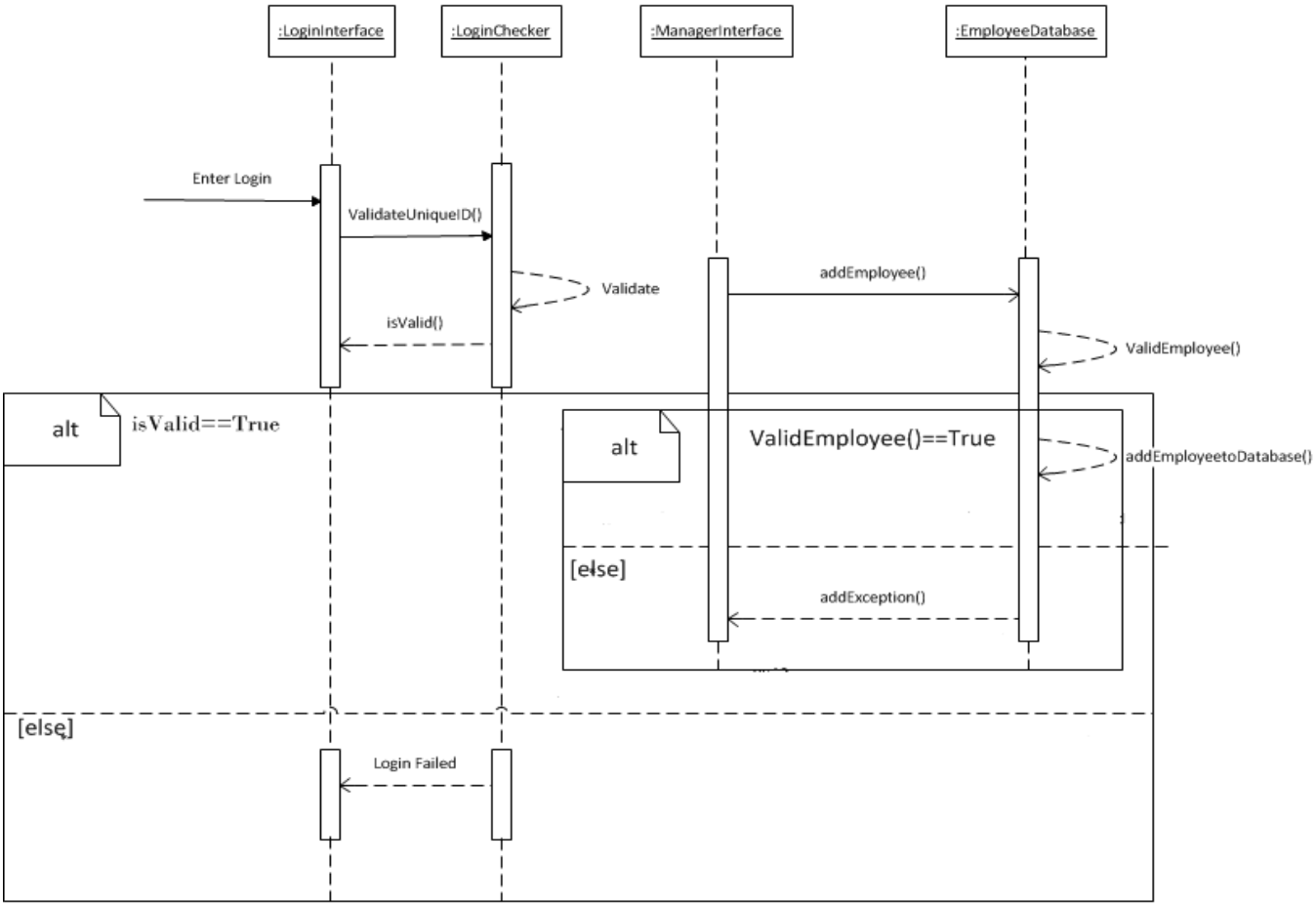



**Fig-22: UC-4: Add Employee Interaction Diagram**

Responsibilities in Fig-22:

1. Manager begins to login.
2. The system checks the validity of the login by the Manager. If isValid()==True then moves to next step.
3. Alternative: System failure. Manager login is denied access to the system.
4. Manager selects to add employee using the addEmployee() method.
5. The Manager will fill out the form for the new employee.
6. The system will validate the employee by using the method ValidEmployee().If ValidEmployee()==True the system will add employee to the database by using the method addEmployeetoDatabase().

Else if ValidEmployee()==False, then addException() method will be need to be complete before the manager can add the employee.

## *Use Case-5: Remove Employee*

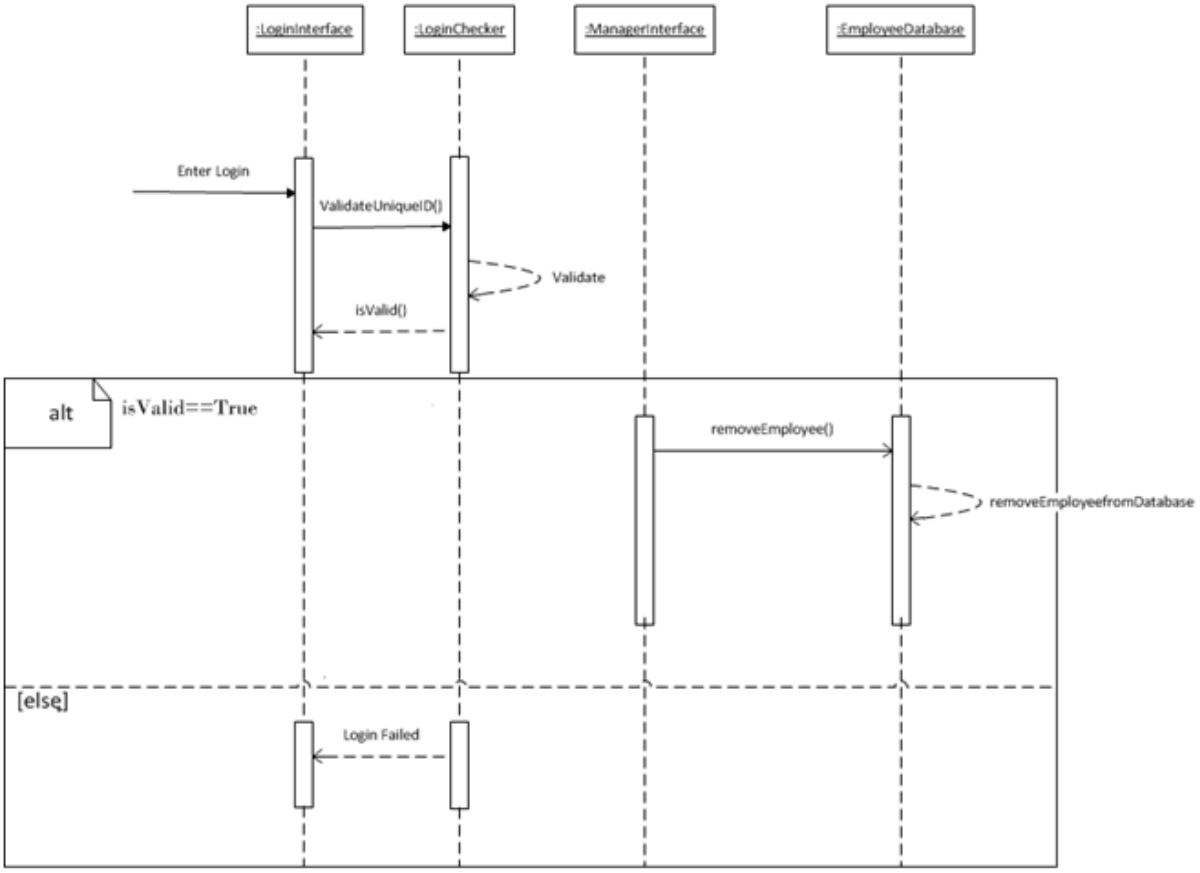



**Fig-23: UC-5: Remove Employee Interaction Diagram**

Responsibilities in Fig-23:

1. Manager begins to login.
2. The system checks the validity of the login by the Manager. If isValid()==True then moves to next step.
3. Alternative: System failure. Manager login is denied access to the system.
4. Manager selects to remove employee using the removeEmployee() method.
5. When manager removes the employee all information for that employee will be erased from the database.

The system will remove the employee from the database by using the method removeEmployeefromDataBase().

## *Use Case-7: Manage Menu*

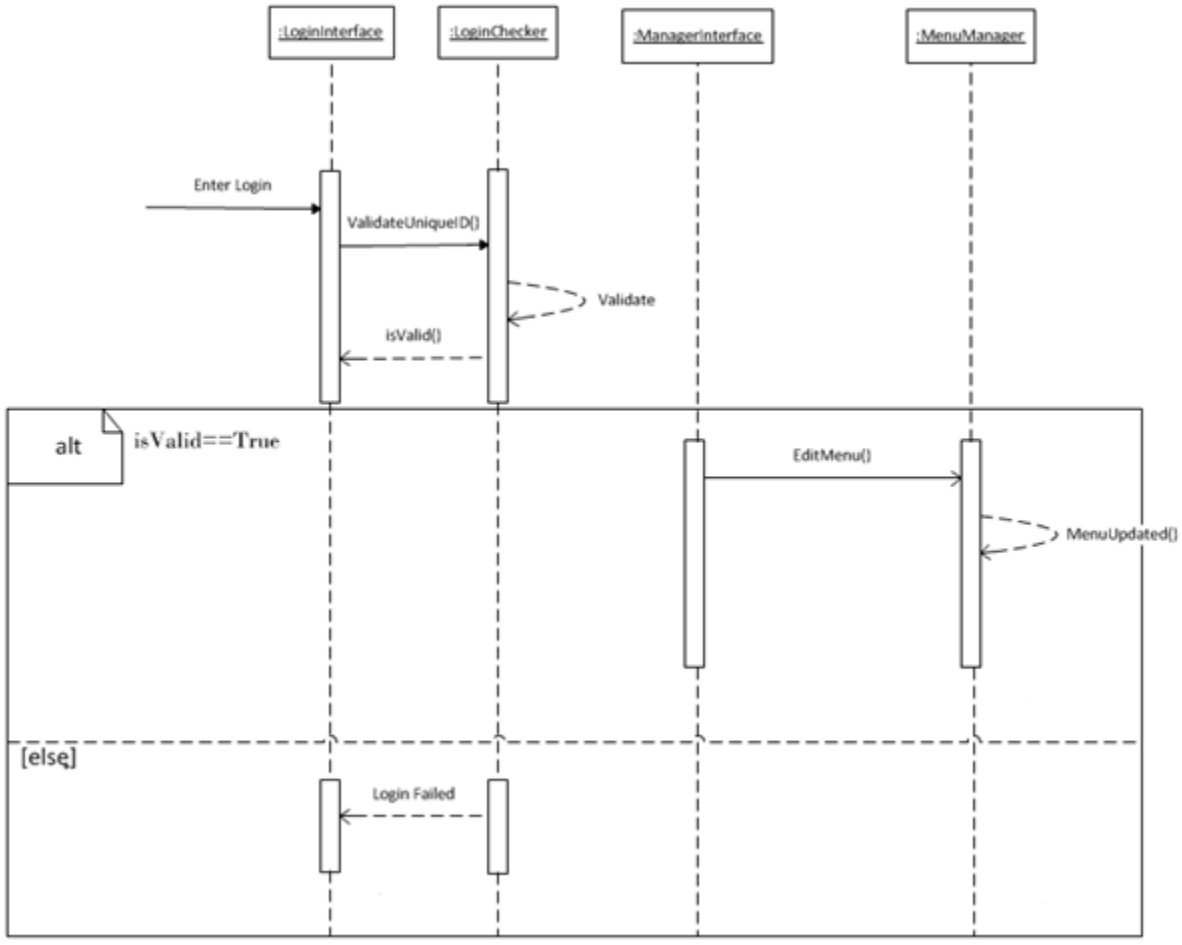



**Fig-24: UC-7: Manage Menu Interaction Diagram**

Responsibilities in Fig-24:

1. Manager begins to login.
2. The system checks the validity of the login by the Manager. If isValid()==True then moves to next step.
3. Alternative: System failure. Manager login is denied access to the system.
4. Manager selects the edit menu items using the EditMenu() method.
5. When Manager selects EditMenu() the manger wants to make a change within the database like changing the name of a item, changing the price of items, or changing the description of an item.
6. Manager will save the changes after EditMenu() is complete.

The system will save changes to the menu by using the MenuUpdated() method.

## Use Case-11 Manage Layout

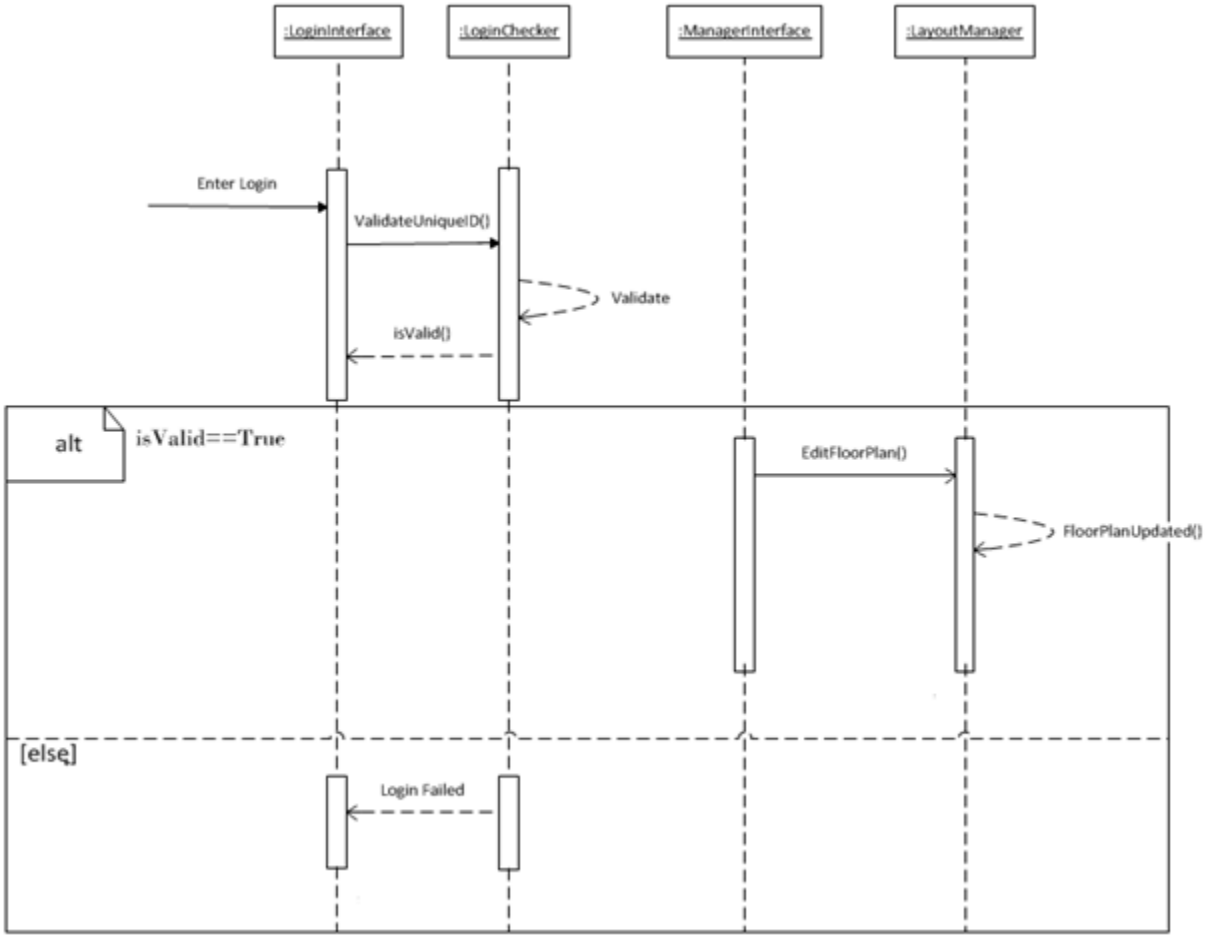



**Fig-25: UC-11: Manage Layout Interaction Diagram**

Responsibilities in Fig-25:

1. Manager begins to login.
2. The system checks the validity of the login by the Manager. If isValid()==True then moves to next step.
3. Alternative: System failure. Manager login is denied access to the system.
4. Manager selects to edit floor by using EditFloorPlan() method.
5. The manager will now have a layout of the floor plan shown on his computer screen. He will then make changes and will update the floor plan which will be sent to the hostess android phone notifying that the floor plan needs to be changed.

The system will update with FloorPlanUpdated() method.

## Use Case-15: Order Food

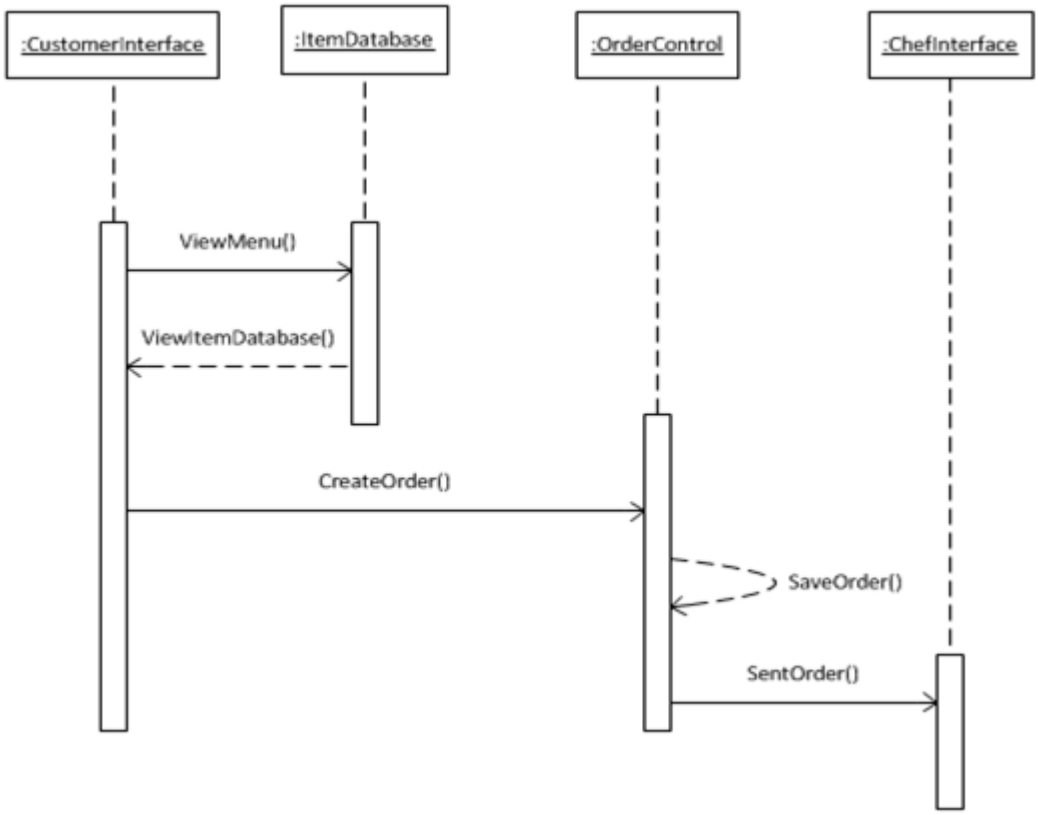



**Fig-26: UC-15: Order Food Interaction Diagram**

Responsibilities in Fig-26:

1. The Customer begins by viewing the menu by using the method ViewMenu().
2. While viewing the menu the items that are shown are display from ViewItemDateBase() .
3. The Customer will create an order of the items that they prefer with CreateOrder() which will be done on an android tablet.
4. The order will be saved and sent to the chef android tablet notifying the chef for cooking. This is done with the method SentOrder() to the chef.

## Use Case-18: CallWaiter

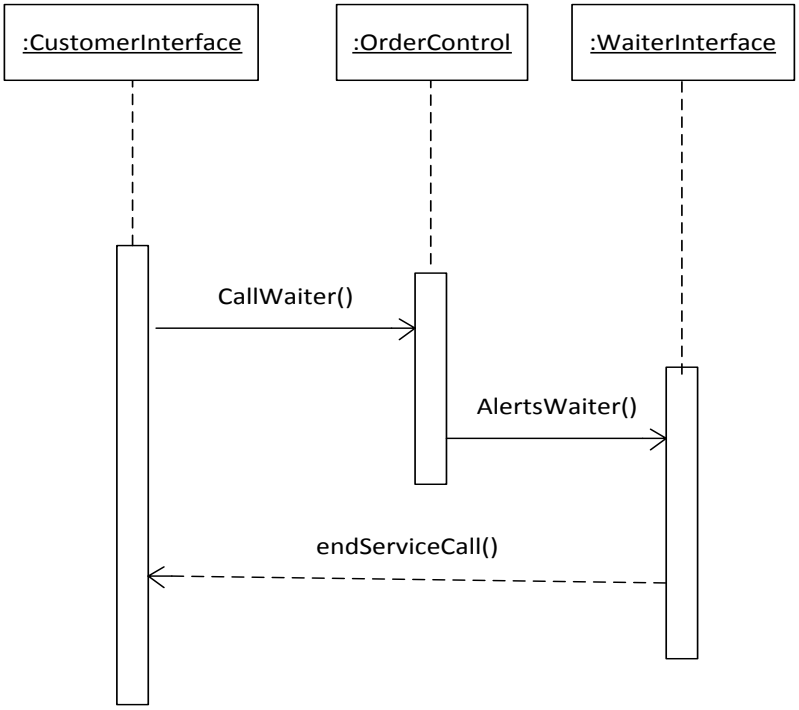



**Fig-27: UC-18: Call Waiter Interaction Diagram**

Responsibilities in Fig-27:

1. The customer wants to call the waiter for help by using the method CallWaiter() thru the OrderControl system.
2. When the CallWaiter() button is used on the android tablet from the customer it will send a message thru the database.
3. The system will then notify the waiter android phone to assist the customer in need.

The waiter will assist all needs of the customer and then end the service call on their android phone using the method endServiceCall() so the system knows that the CallWaiter() alert has been handled.

## Use Case-19: Pay Bill

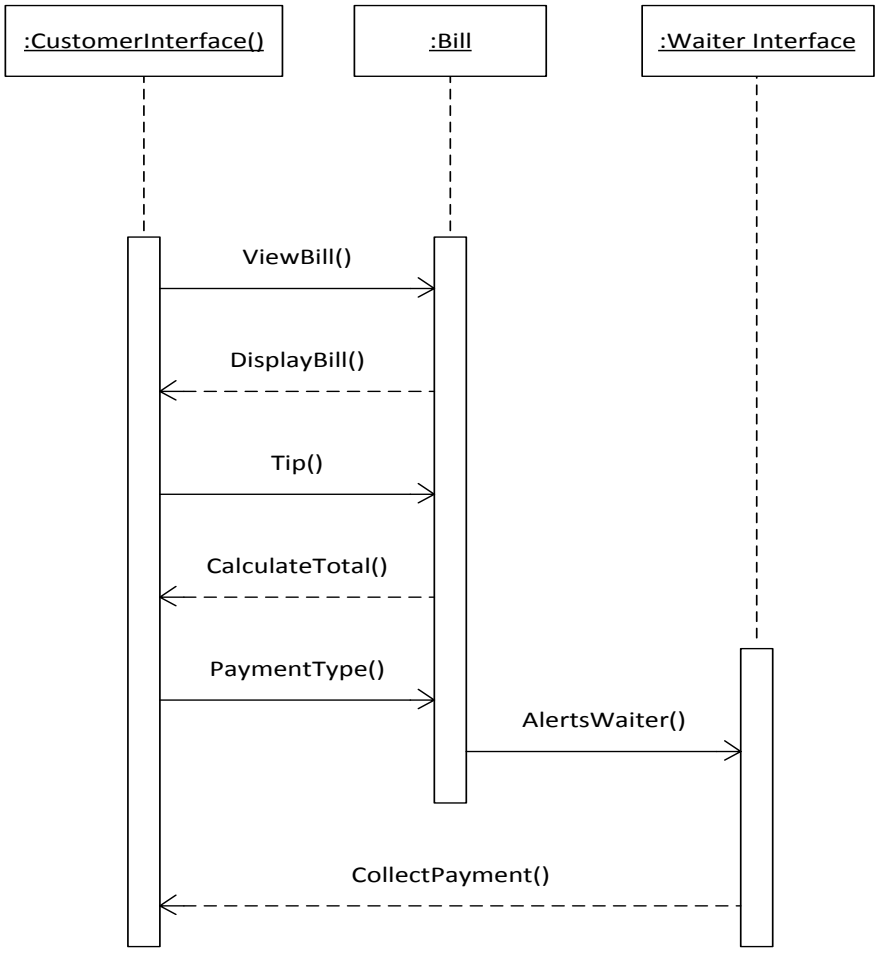



**Fig-28: UC-19: Pay Bill Interaction Diagram**

Responsibilities in Fig-28:

1. During check out the customer will need to view the bill using the method viewbill().
2. The system will display the bill with displaybill().
3. The customer will use tip() method to calculate the tip amount for their meal.
4. The system will then calculate the total amount with the tip with calculatetotal().
5. Customer will select payment type.

The system will notify the waiter to collect the payment from customer by using collectpayment() method.

# 8. Class Diagram & Interface Specification

## a. Class Diagrams

### *Use Case UC-1: Login*

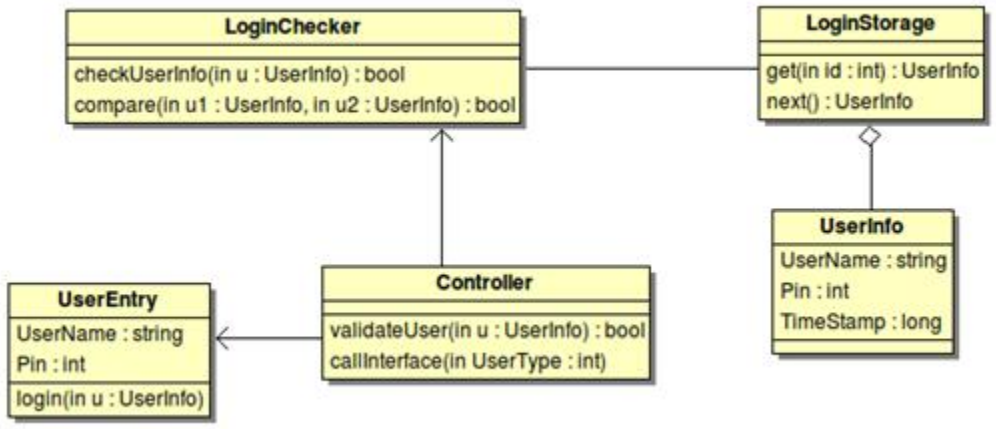



**Fig-29: UC-1: Login Class Diagram and its interface specifications**

### *Use Case UC-8: AddItem*

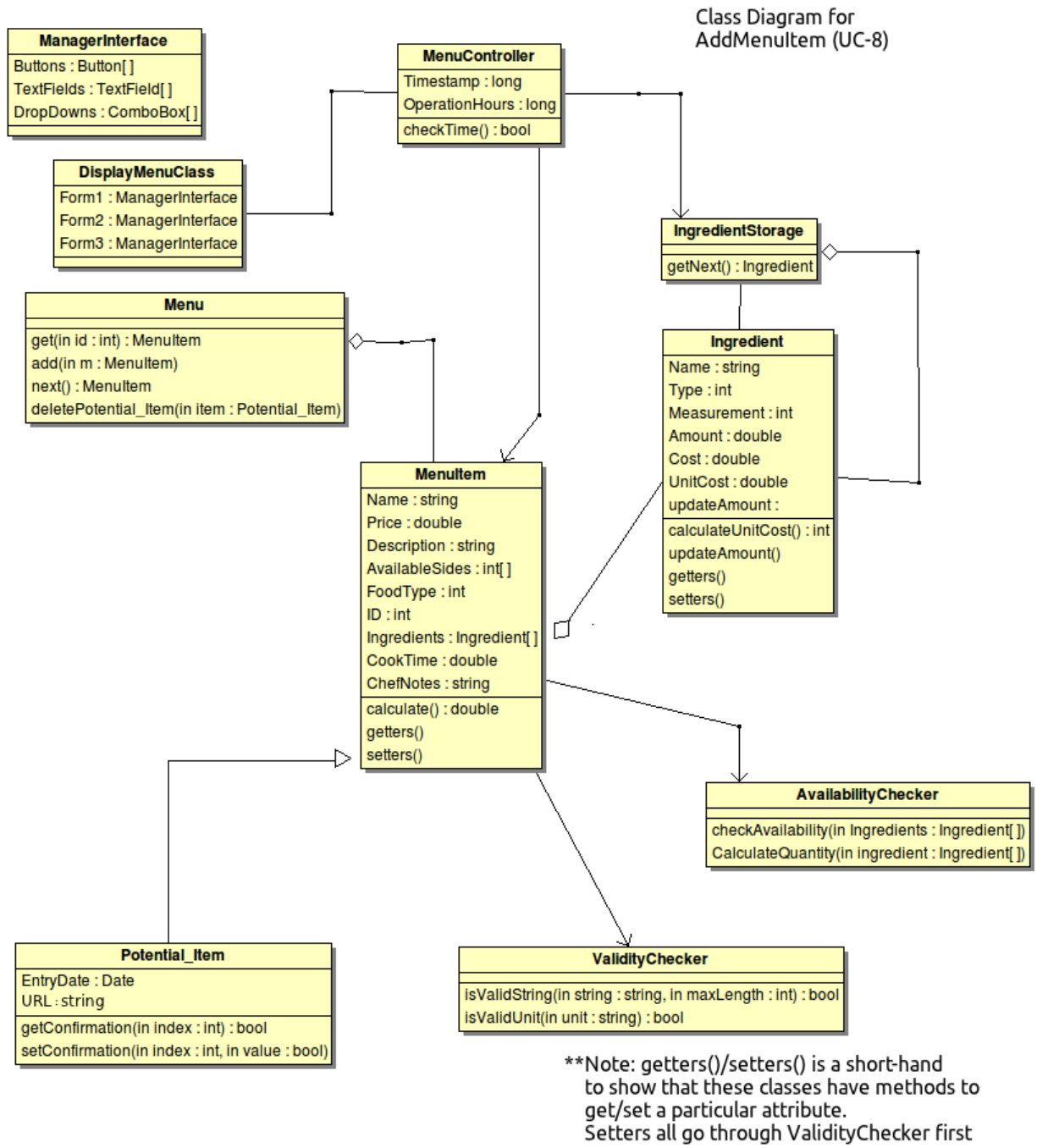



**Fig-30: UC-8: Class Diagram and its Interface Specifications**

### *Use Case UC-15: Order Food*

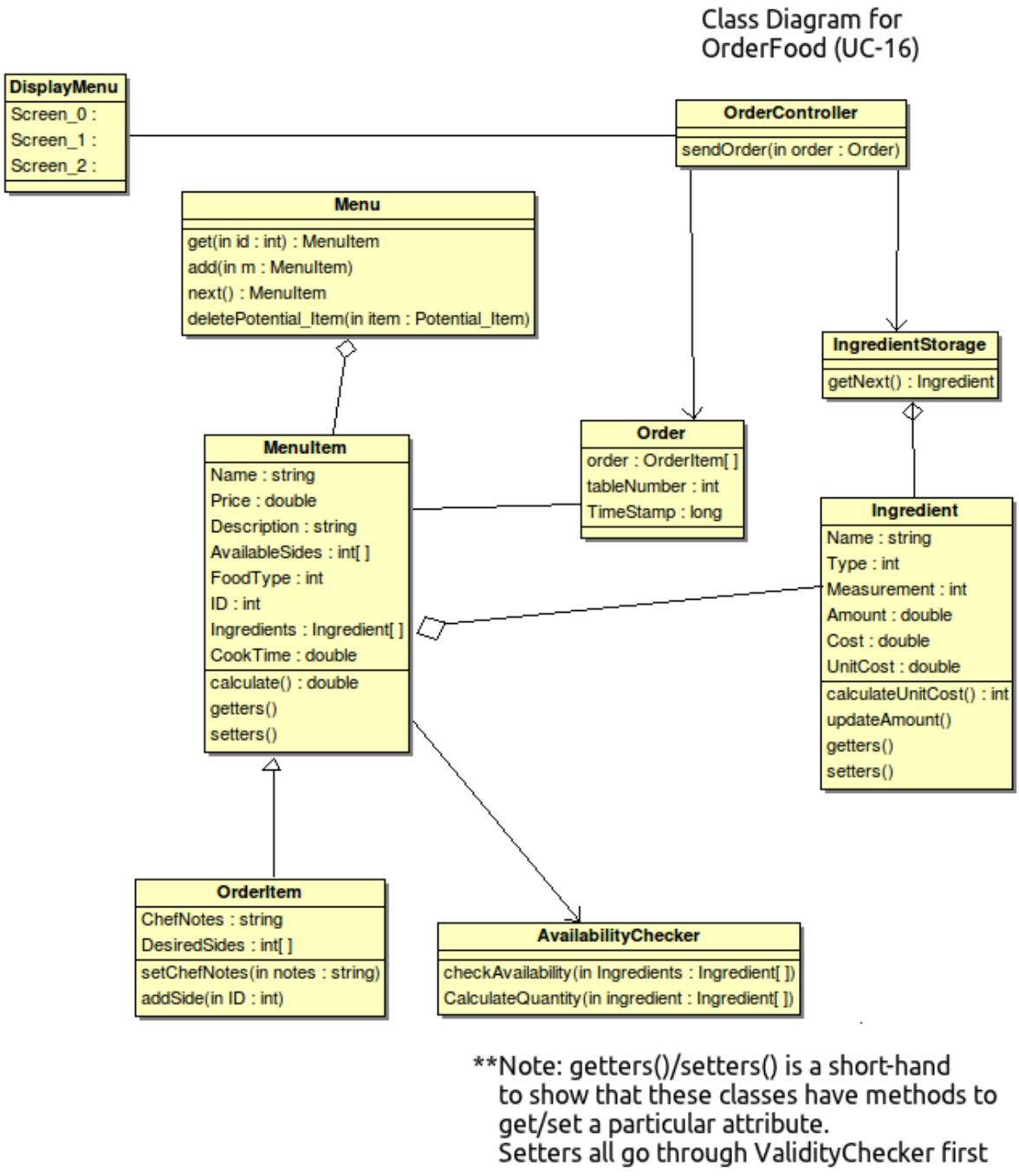



**Fig-31: UC-15: Order Food Class Diagram and its Interface Specifications**

## b. Data Types & Operation Signatures

**ManagerInterface**

- Buttons : Button[]
- TextFields : TextField[]
- DropDowns : ComboBox[]

**DisplayMenuClass**

- Form1 : ManagerInterface
- Form2 : ManagerInterface
- Form3 : ManagerInterface

**MenuController**

- Timestamp : long
- OperationHours : long

+ checkTime() : bool

**Menu**

+ get(id : int) : MenuItem
+ add(m : MenuItem) : void
+ next() : MenuItem
+ deletePotential_Item(item : Potential_Item) : void

**MenuItem**

- Name : String
- Price : double

- Description : String

- Sides : int[]

- FoodType : int

- ID : int

- Ingredients : Ingredient[]

- CookTime : double

+ calculate() : double

+ getName() : String

+ getPrice() : double

+ getDesc() : String

+ getID() : int

+ getIngredient(index : int) : Ingredient

+ getCookTime() : double

+ CalculateQuantity() : double

- checkAvailability() : bool

**Potential_Item (extends MenuItem)**

- Confirmed : bool[2]

- EntryDate : Date

+ getConfirmation(index : int) : bool

+ setConfirmation(index : int, value : bool) : void

**IngredientStorage**

+ getNext() : Ingredient

**Ingredient**

- Name : string
- Type : int
- Measurement : int
- Amount : double
- Cost : double
- UnitCost : double
+ getName() : String
+ getType() : int
+ getMeasure() : int
+ getAmount() : double
+ getCost() :double
+ calculateUnitCost() : double
- updateAmount() : void

**OrderController**

+ sendOrder(order : Order) : void

**Order**

- order : MenuItem[]
- tableNumber : int
- TimeStamp : long

**UserInfo**

- UserName : string

- Pin : int

-TimeStamp : long

**Controller (From Login Class Diagram)**

+ validateUser(u : UserInfo) : bool

- callInterface(UserType : int) : void

**LoginChecker**

+ checkUserInfo(u : UserInfo) : bool

- compare(u1 : UserInfo, u2 : UserInfo) : bool

**UserEntry**

- UserName : String

- Pin : int

+ login(u : UserInfo) : void

**LoginStorage**

+ get(id : int) : UserInfo

+ next() : UserInfo

## c. Traceability Matrix

| Domain/Class | Order Controller | Order | Display Menu | Display Menu Class | Menu Controller | Validity Checker | Availability Checker |
|---|---|---|---|---|---|---|---|
| **Menu Display** | | | x | | | | |
| **Menu Confirm** | | | | x | | | |
| **Calculator** | | | | | | | x |
| **Item Checker** | | | | | | x | |
| **Availability Checker** | | x | | | | | |
| **Subtractor** | | x | | | | | |
| **Controller** | x | | | | x | | |

**Table-9: Use-Case UC-8: AddItem & Use-Case UC-15: OrderFood Traceability Matrix**

Table-9 shows the traceability matrix for part of the class diagram that is part of our first demo. Since we will be implementing UC-1, UC-8 and UC-16 we emphasized these use cases in both our class diagrams as well as our traceability matrix. Most of the domain concepts are their own class, except for a certain few such as the controller which had to be split into multiple classes; otherwise a single class would have too much to do.

# 9. System Architecture & System Design

## *a. Architectural Styles*

**1) Client-Server**

The style describes system that involves a separate client-server system, and a connecting network. The client application will run a program that will communicate with a centralized server. The server application will be accessed by multiple clients, also referred to as a 2-tier architecture style. The client software will initialize a communication session, while the server will wait request queries from the client.

Our system will implement the Client-Queue-Client systems. Every user will have a terminal, which will contain a graphical UI establishing communication with the database server containing much of the operation logic. Each user terminal will be constantly connected to the database server, where the server waits for the request queries from the client. Each request query from the client will go through the server and the server directs the request to the appropriate processing unit. This approach will allow clients to communicate with other clients through a server-based queue. Clients will be able to read data and send data to a server that acts simply as a queue to store the data. Figure 1 shows an example of how the client server architecture is utilized in our system.

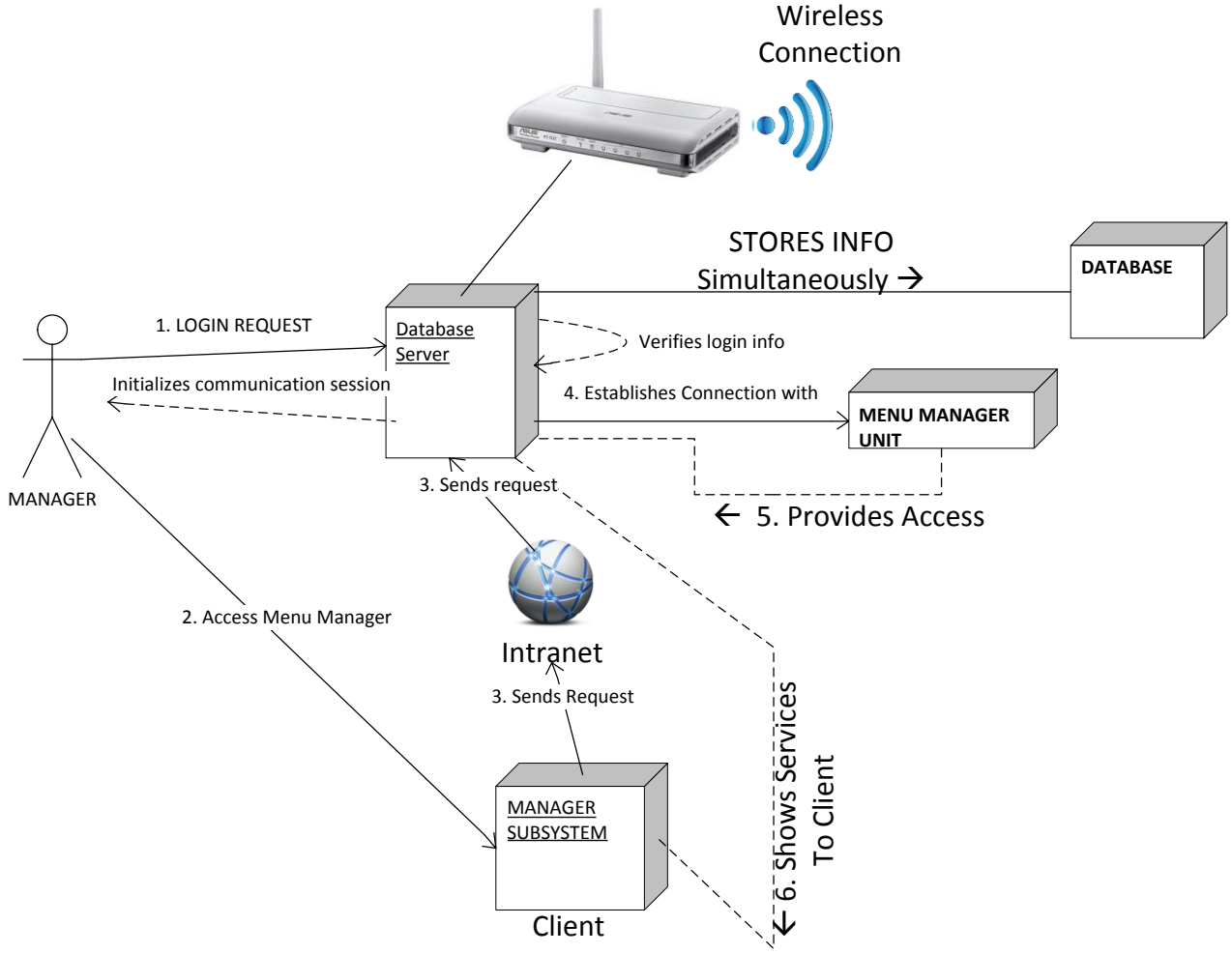



**Figure32: Client Server communication process**

Fig-32 displays the interaction process of the manager with the system utilizing Client-Server method; the process if as follows:

1. All the clients (users) except for customers will have to login first.
    a. The server gets the request to login, where the login information is verified and communication session is established.
2. The client can then use their subsystem to access provided services: in this case the manager will have access to menu manager, view employee info, inventory control and floor layout.
    a. The client accesses menu manager.
3. The Manager subsystem relays the request to the server.
4. The server establishes connection with the appropriate processing unit, in this case: Menu Manager Unit
5. The Menu manager unit provides access to the server.

6. The server provides access to Menu Manager's services such as addNewItemRequest, removeItem, updateItem and so on.

Successful login is required to establish a communication connection with the server and then client can use any of the services from the unit. The appropriate function unit is basically in the server, the Menu Manager Unit is shown explicitly to make it easy for the reader to understand the functioning of the system. Connection between the Unit and the server is through the Enterprise Service Bus (ESB); and the units use subscribe and publish method of Message Bus to communicate with each other, which is discussed later in this section. All the information is saved and sent through the server, which is finally stored into the database simultaneously for backup purposes.

The main benefits of client-server architecture are:

- Higher security – all the data will be stored on the server, offering a greater control of security than client machines.
- Centralized data access – Since data is stored only on the server, access and updates to the data are easier to administer than in other architectural styles.
- Ease of maintenance – The roles and responsibilities of the system are distributed among several aspects of our server and are known to each other through a network. This ensures that a client remains unaware and unaffected by a server repair, upgrade, or relocation.

A major disadvantage to this system is the tendency for application data and operation logic to be closely combined on the server, which can negatively impact system extensibility and scalability, and its dependence on a central server, which can negatively impact system reliability.

**2) Message Bus**

This particular architecture design describes the principle of using 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. Our system will implement ESB and Subscribe/Publish message bus for designing applications where interaction between applications is accomplished by passing messages asynchronously over a common bus. In Figure 2, ESB provides commodity services in addition to translation and routing of a client request to the appropriate answering service. The request from the Database Server is sent to the ESB, which then routes the request to the appropriate service unit. The service units utilize another aspect of message bus, that is, the subscribe/publish method to communicate amongst each other.

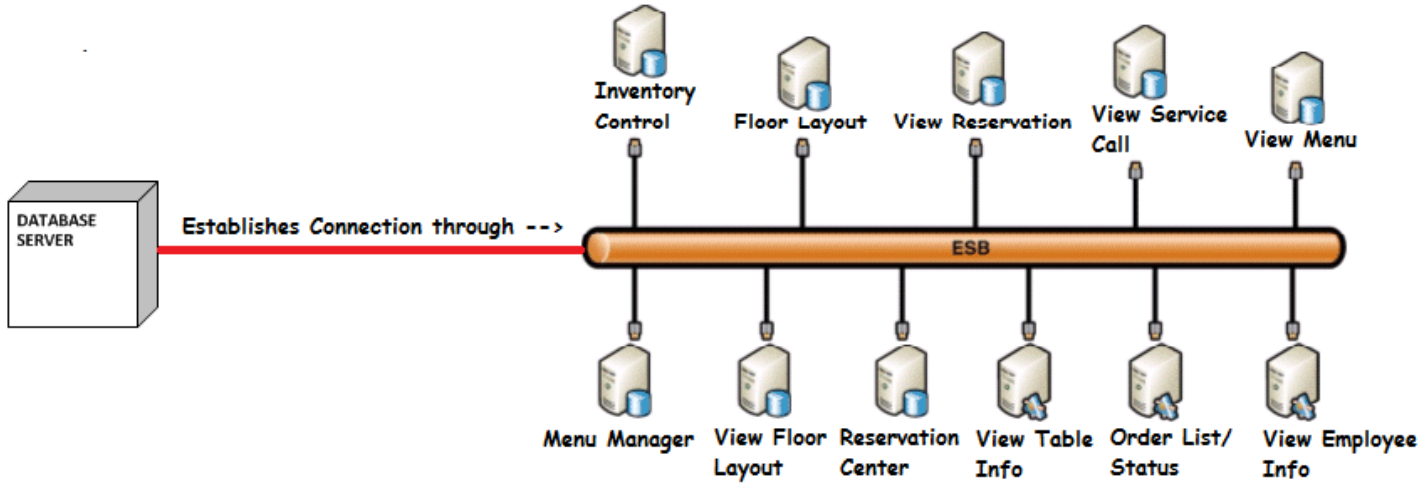



**Figure 33: Implementation of ESB (Message Bus)**

Figure 33 is an example of service units utilizing subscribe/publish method to read, write, and update information amongst each other, without having to go through the server every time. The method provides quicker results especially with the information that does not need to be stored in the database. The Message Bus bar in the image is the subscribe/publish passage. The conditions shown in the image are: “view floor layout” service unit subscribes to the “view table info” unit and vice versa; when the waiter updates the table status in “view table info” the update message is carried through the message bus and published to the hostess’ “view floor layout” unit. Second condition shown is that “view reservation” unit subscribes to the “Reservation center” unit; when a customer makes a reservation using the “Reservation Center” unit, the message is transferred through the message bus with all the information and published in the “view reservation” for the hostess.

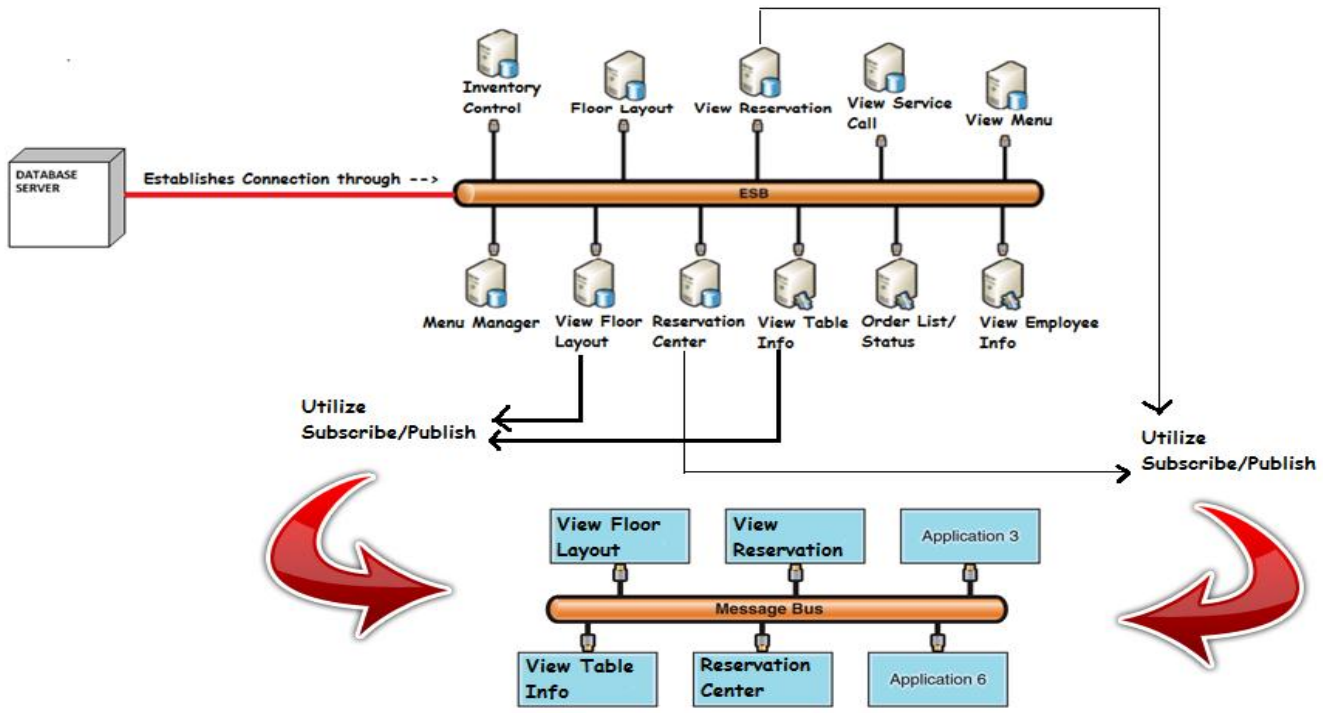



**Figure 34: System utilizing Subscribe/Publish method (Message Bus)**

As you can see in Fig-34, the client-server and the message bus implementation will be integrated together to make the system work efficiently. The client-server model will help establish communication sessions that will allow clients to make requests and the server to process them utilizing Message Bus methods, which include ESB and Subscribe/Publish. Message Bus will be the medium that completes requests and provides the desired results. Therefore, the two architecture styles define our system operation and a network, where clients and server can communicate with each other to generate results.

## b. Identifying Subsystems

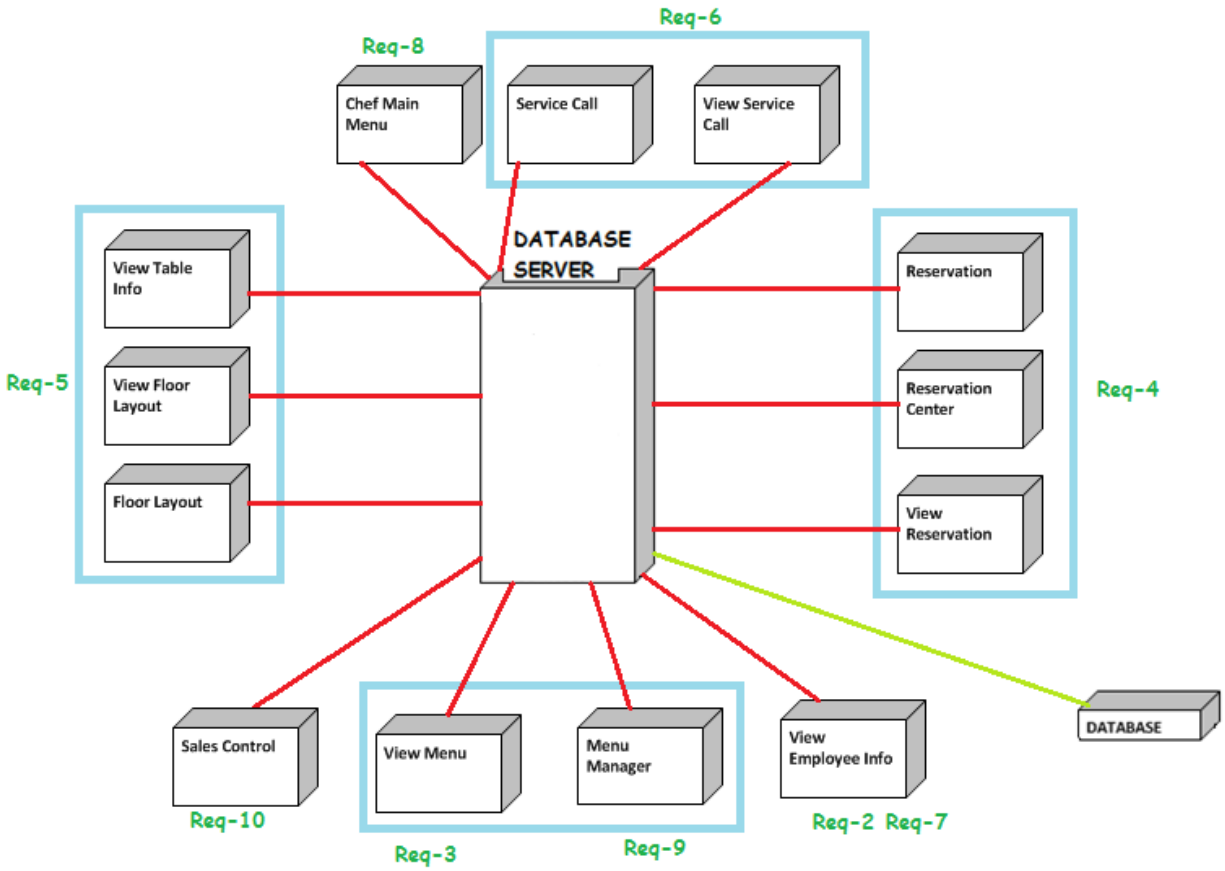



**Figure 35: Identified Subsystems.**

Subsystems are the key components of a "Software-to-be" that describes what the software is all about. Identifying subsystems is part of the analysis that is required to design software. Subsystems are made to fulfill requirements that are stated at the beginning of every software design project. Such analysis is visually presented in Fig-35. Some of the subsystems are grouped together because integrating multiple subsystems together allows fulfillment of a requirement that is stated in "Green". For example, grouping is signified by the blue border and the "View Table Info", "View Floor Layout", and "Floor Layout" implement different functions that work together to fulfill Req-5 completely. Following the figure, "View Employee Info" satisfies Req-2 and Req-7, "View Menu" and "Menu Manager" satisfy two separate requirements, Req-3 and Req-9 respectively. "Sales Control" fulfills Req-10, "Reservation", Reservation Center", and "View Reservation" work together to fulfill Req-4, "Chef Main Menu" fulfills Req-8, and "Service Call" and "View Service Call" satisfy Req-6. Grouping does not mean that those subsystems can only satisfy one requirement, they can fulfill multiple Reqs and each individual subsystem can also fulfill a particular Req. After identify the crucial subsystems, it is advisable to map them to the hardware, which is discussed next.

## c. Mapping Subsystems to Hardware

The restaurant automation system is set to run on multiple terminals that are also the identified subsystems. Terminals such as android tablets are going to be use for customers and chef to place orders and receive orders. The other subsystem such as the waiter and hostess will have an android phone that can receive messages when they get an alert. The manager subsystem will be run thru a computer allowing him/her to add/edit/remove items from the menu database. The restaurant subsystems (terminals) will be hardwired to a database server, which is finally connected to a database where all the information is saved for backup purposes. The subsystems are also connected to Internet and the database server is connected to a wireless router to convey information to the tablets. The database itself will be located in its own private room so employees/customers can't just enter and mess around with it. The image below provides an overview of the overall system.

**The subsystems (terminals)**, although not shown in the image below, will incorporate a Windows 7 Professional operating system processing through Intel Core i3 CPU @ 2.10 GHz or higher with 2GB or higher RAM and 1GB Hard drive. The subsystems will establish communication with each other and the server through intranet and will also have access to the internet to access information online. **The database server** will be operating on SQL Server 2008 R2 as a minimum requirement, where all the information is stored to be accessed concurrently. The server will process requests at a speed of 1GHz or higher with a minimum of 2GB RAM and a 100 GB hard-drive. Installing software on the server will allow distributing it to the other subsystems if needed. The server is also connected to a wireless router to provide services on the tablets. The server is constantly storing information into another database for backup purposes, which also has a minimum storage requirement of 100GB. **The android tablets** will incorporate Android Froyo (2.2)and will be connected wirelessly to the server processing request queries at a speed of 1GHz or higher with a 512 MB RAM at the minimum and a 1GB hard-drive. Therefore, the communication methods used by different components of the system allow the subsystems to work together as a system.

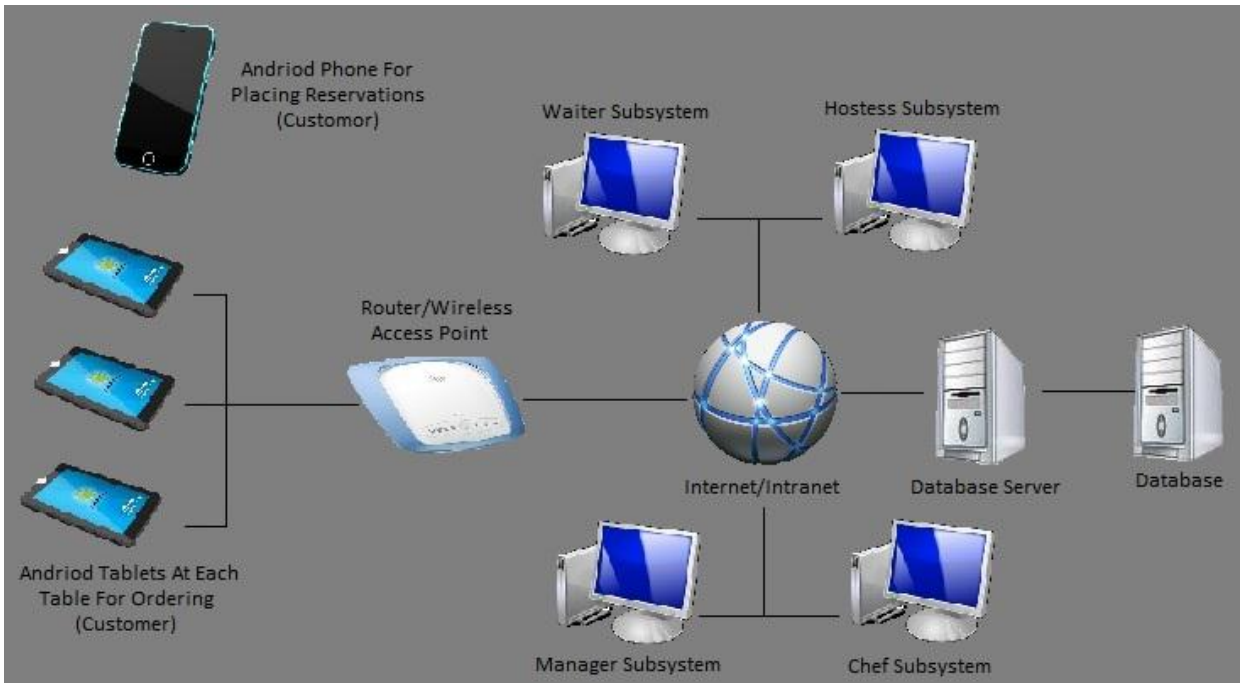



**Figure 36: A high level network model along with hardware allocations**

## d. Persistent Data Storage

Our system will use a central database to store and manage the persistent data. Persistent data is data that is required to be stored permanently in our database. For this sole reason, we decided to go with the relational database model. All our data will be partitioned into tables, organized appropriately based on a set of rows and columns. Each column consists of data attributes, with each row storing different records. Not only does this provide easy access for our system to retrieve data, but all operations on data will simply be implemented on the tables. This will work in our favor when writing SQL commands since all the information will be presented in a systematic manner. It is, also, important to note that each data broken into smaller pieces is related to one another in one form or another.
Furthermore, we decided to have our relational database be built on a client/server paradigm. The software applications act as the clients to the server and do not have to deal with the manipulation of the database directly. Their only task is to make requests for the server to perform the assigned operations. This holds several advantages as supposed to a database built without a server. Having a database with a built in server allows the server to maintain a backup of the data and add sophisticated features.

A list of the persistent objects required for our database is presented below.
**Employee Data** will hold personal information of each employee
**MenuItem** will maintain a record of all the menu items within the restaurant
**Table** will keep track of each table displayed on the floor layout, table status, and waiter currently assigned to that table
**Order** will keep track of each customer's orders, its current status, the table that requested the order and the date the order was made
**Receipt** will store each food item's revenue
**CustomerID** will keep track of each customer's personal information that makes a request for reservation
**Below is a sample of each table within the database.**

**Employee Data:**

| Last Name | First Name | PIN | Birthdate | Age | Hire Date | Last Day Worked | Wage | Street Address | City | Zip | State | Home # | Cell # |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Smith | John | 9820 | 01/15/91 | 21 | 06/02/11 | ----- | $8.50 | 21 | North Brunswick | 08902 | NJ | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**Menu Item:**

| Item ID | Item Name | Price | Food Type | Description |
|---|---|---|---|---|
| 101003 | Chicken Tenders (6) | $10.55 | Tenders & Shrimps | Original all-white chicken Buffalo Tenders ™ lightly breaded and cooked until crispy. |
| 202004 | Jerk Chicken Sandwich | $9.56 | Sandwiches | Blackened grilled chicken breast topped with our signature Carribean Jerk™ sauce and bleu cheese dressing. |
| 300003 | Grilled Chicken Salad | $9.12 | Salad | Seasoned, grilled chicken served over fresh greens with tomatoes, onions, a blend of cheeses and croutons. |

**Table:**

| Table Number | Status | Name of Assigned Waiter |
|---|---|---|
| 03 | Occupied | Smith, John |
| 01 | Reserved | ----- |
| 08 | Open | ----- |

**Order:**

| Order Number | Status | Table Number | Order Date |
|---|---|---|---|
| 003 | Ready | 03 | 07/03/11 |
| 004 | Processing | 01 | 07/03/11 |
| 005 | Processing | 05 | 07/03/11 |

**Receipt:**

| Item ID | Item Name | Price | Quantity | Total Cost |
|---|---|---|---|---|
| 100103 | Chicken Tenders (6) | $10.55 | 2 | $21.10 |
| 202004 | Jerk Chicken Sandwich | $9.56 | 1 | $9.56 |
| 300003 | Grilled Chicken Salad | $9.12 | 1 | $9.12 |

**Customer ID (reservation system):**

| Last Name | First Name | Phone # | Email Address | Comments |
|---|---|---|---|---|
| Jordan | Mike | xxx-xxx-xxx | xxxxxx@gmail.com | Birthday Celebration |
| | | | | |
| | | | | |

## e. Network Protocol

We will be using the Java JDBC protocol to implement our project. Java JDBC provides us with the interface for accessing relational databases. Via JDBC we can maintain the database connection, issue database queries and updates and receive the results.

In our project our primary tasks are to store and update data related to restaurant automation. This can be done via a SQL database. JDBC allows us to construct SQL statements and embed them inside Java API calls. JDBC lets us smoothly translate between the world of database and world of Java application which is why Java JDBC is the perfect match for our software implementation. We will be installing the Java JDBC driver on our platform in order to transition between Java and MySQL.

The other communication protocols are well developed and functional. However, they are not the best match for our software. The reason Java sockets wouldn't fit in is because there is multiple communication links required for programs. It is not as simple as client and server sockets. HTTP protocol might be necessary for the reservation system depending on how we plan on implementing the reservation system. As of now, we are not sure as to how Android treats applications. Therefore we will definitely be centering our network communications around Java JDBC.

## f. Global Control Flow

**Execution Orderness:**

The system will follow an event-driven execution system where request can be made by any user at any point in time. Once a request is made, the system will process it using an appropriate function, managed by the control structure. Certain requests will have to follow a given procedure once they are initiated. For ex: when the customer touches submit order, it will follow a procedure – where it will go through the server and gets recorded, and is then passed on to the chef and the waiter for further actions. When no actions are being taken, the system will be idle until user-interaction occurs. Multiple users can make requests simultaneously and each request will be handled accordingly by the control structure.

**Time Dependency:**

Our system will consist of multiple timers for various employees on the system. Our system is more of an event-response type, with no concern for real time. For example, in terms of response time, consider the case when the customer touches the "submit order" button on the android tablet; on average the chef will receive the order list from the customer in less than a minute. If the order list does not display on the chef's screen, then the chef will need to refresh the interface, resulting in a minute delay on average to complete an order. Another example, when the waiter cleans a dirty table and updates the table status, the response time for the status to show up on the hostess' display can take up to a minute. This time delay applies to all the users on the system. The chef is also provided with a feature where he can post an approximate time he will need to prepare an order. The objective of this feature is to assist the chef with cooking times, which will help in maximizing kitchen output. The waiter is also provided with a preset time limit to deliver an order when the status turns to ready. This helps the manager in evaluating employee performance as well as the overall service experience of the customers.

The event-response is derived from the fact that the system will only need to react to input from multiple interfaces. This particular system also utilizes a unique technique, which specifies the syntax of multi-threaded dialogues. They compactly represent the concurrency needed to implement multi-threaded dialogues. The support allows interfaces to be structured differently compared to the existing dialogue specification systems based on state transition specifications or grammar. The flexibility allows many interfaces, especially direct manipulation interfaces, to be specified with a more modular structure than most existing systems allow.

The system is not concerned with real-time response because the required response does not need to occur instantaneously. Since there are no particular time constraints in the overall system, time delay of a minute or two would not make a difference in functionality.

Concurrency:

The system will contain multiple threads, which involves multiple subsystems running independently of each other. All interactions between subsystems are controlled through a database or through the "control structure". For example, in terms of multiple threads operating at once, there are two waiters changing the status of different tables to "ready", both are contacting the system to change the certain tables to ready which will show up on the hostess interface showing that the two different tables are ready for more customers.

From the example above, you can see that no direct communication is done between subsystems. The entire communication between the subsystems is completed through a check of the information in the database or "controller", which also provides means for synchronizing data between subsystems.

## g. System Requirements

The following hardware requirements were decided based on the testing environment we will be using. Some of the requirements do not necessarily have to be met since the software does not require all the resources listed below. However, we would rather exceed the expectations for future software upgrades, rather than replace hardware every time a software upgrade is performed. There are four key hardware devices in our system:

- Windows Terminals
- Android Tablets
- Android Phones
- Datacenter

### Windows Terminals

- Processor: Intel Core i3-2310M CPU @ 2.10GHz or higher.
- Random Access Memory – 2GB or Higher
- Hard Drive: 1GB should be sufficient.
- Operating System: Microsoft Windows 7 Professional
- Display: Screen Resolution of 1366x768 is preferred.
- Mouse & Keyboard for user input.
- Ethernet (10/100) port to support network interaction.
- Universal Serial Bus (USB 2.0 or higher) will be necessary to place and install required files to enable respective end user interfaces.
- Speakers preferred for future voice commands and playback.

### Android Tablets

- Processor: Dual-Core 1 GHz or higher.
- RAM: 512MB or higher preferred.
- Hard Drive: 1GB preferred.
- Micro-USB port to place and install the respective software.
- Operating System: Android Froyo (2.2) or higher.

- WLAN: Wi-Fi a/b/g/n required to transfer data between the server and the tablet.
- Mobile Broadband: Not required since data will be transferred via Wi-Fi.
- Display: 768x1024 pixels, 9.7 inch screen (~132 ppi pixel density).
- Multi-touch capability.
- Adobe Flash capability to play the interactive advertisements.
- Speakers preferred for future voice commands and playback.

## Android Phone

- Processor: Dual-Core 512 Mhz or higher.
- RAM: 512MB or higher preferred
- Hard-Drive: 100MB or higher as the software for this device is quite meek.
- Access to Android Store in order to download the application.
- Operating System: Android Froyo (2.2) or higher.
- WLAN: Wi-Fi a/b/g/n required to transfer data between the phone and restaurant's server.
- Mobile Broadband is required in the absence of Wi-Fi hotspots for the end-user.
- Multi-touch capability
- Display: May vary according to end-user's preference. As long as above requirements are met, display size will not affect the functionality of the software.

## Datacenter

- Operating System: SQL Server 2008 R2 or higher.
- Processor: Recommended 1.0 GHz or faster.
- RAM: Minimum of 2GB. 4GB Recommended.
- Software: Microsoft Windows Installer 4.5 or later; MySQL
- Hard-Disk: A minimum 100GB Hard-Drive to start-off. Size may vary based on restaurant usage.
- CD/DVD Drive for installing software.
- Universal Serial Bus (USB) Driver to place, install and transfer required software/data.
- Display: VGA Display: At least 800x600 pixel resolution.
- Mouse and Keyboard for user input.

# 10. Algorithms & Data Structures

## a. Algorithms

The algorithms of our sale system will provide help for all the simple usage of our use cases between the actors and the system. These use cases will require relatively easy algorithms such as adding, removing, editing, updating items, or calculating bill totals of all different time frames into one easy complete frame that shows the final results.

The system also includes some complex algorithms like search/sort items. The search/sort algorithms will be most likely use when the administrator is in need of an important business conclusion that needs to search/sort for a certain item, previous records, restaurant data, sales records, etc.

Our system contains many algorithms from basic. For example one of the basic algorithms is on implementing the orderfood use case where we had to limit number of sauces based on number of wings the customer ordered. Algorithms are the most important thing in the system because it does the task that the all the employees need. Without algorithms simple things will become harder and more time consuming which will cost the business more money and time.

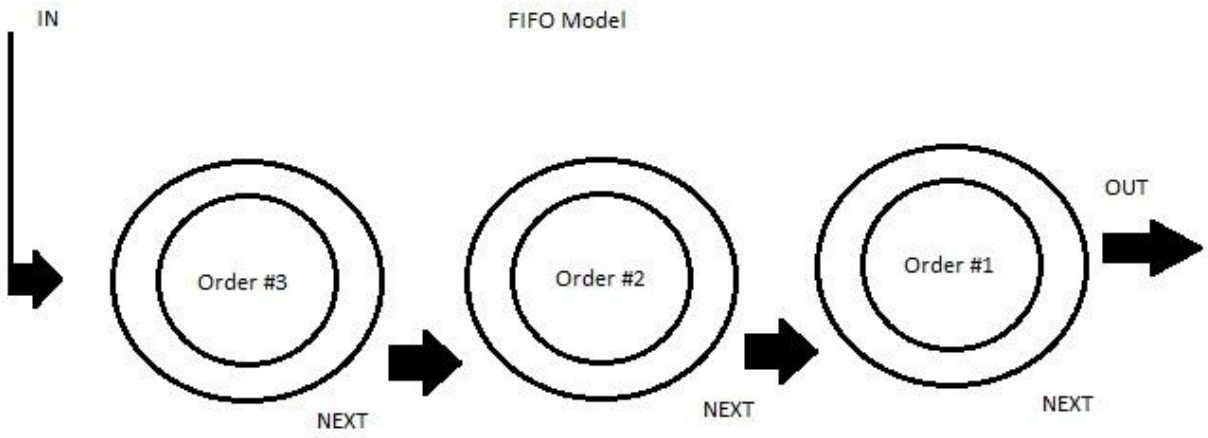
## b. Data Structures

Data structures are necessity for an efficient software system. Our system has all different types of complex data structures like arrays, linked lists, stacks, hash tables, and queues. Our system uses arrays because of their short processing time. Since we are using android devices which are limited in hardware performance, we concentrate on the performance for all the functions involving all our android devices. Arrays will help the performance rather than using another data structure causing the devices to run slower.

Linked lists are other data structures that our system can implement. Linked lists are an excellence way for great flexibility performance within the system. We all are familiar with the advantage that linked lists provide in the dynamic use of memory. Hash table is a data structure that uses a hash function to map identifying values that is input into the system, known as keys (person's name), to their associated values (person's telephone number). Thus, a hash table implements an associative array. The hash function is used to transform the key into the index of an array element where the corresponding value is to be sought. The main reason of hash tables is the ability to search on average with a constant-time just like arrays regardless of the number of items in the table. Hash tables also allow us to perform different searches over the same database tables giving us flexibility to provide changes to our statistical data information at any time as might be needed by the customer's requests.

Our system will contain the FIFO (first in first out) method. This applies that the value of the highest priority in the queue. An example, when deciding which order to place first in the queue when sent to the chef, the time at which the order was sent is the deciding factor for FIFO. Queues are necessary so the order will be processed first and removed from the queue after it has

been prepared and ready to serve as shown in the figure below. There are many other data structures that we might implement but as of now these are the main data structures that the system is running.



# 11. User Interface and Design Implementation

## CUSTOMER INTERFACE

The Customer will be in communication with the System through an Android Tablet Application representing the restaurant menu. Upon seating the Customers, the Hostess will provide the table with an Android Tablet displaying the Menu Home Page (Fig-4). The Menu Home Page provides everything for the Customers to best enjoy their dining experience including options to Call the Waiter, Order Food, View the Order, and Pay the Bill.

Once the Order Option is selected, the Android Tablet displays the Menu Interface (Fig-5) which lists all Menu Items organized by categories such as Appetizers, Burgers, Desserts, etc. All categories are presented as Buttons in the top-left corner of the screen. Because of sizing issues, only six Category Buttons are presented at a time. The first five buttons will be Menu Categories and the sixth button will be a 'More' Option that displays the remaining Menu Categories if there are More than six.

After selecting the desired category, all Menu Items within the category are displayed on the left side of the tablet. This list is clickable and will be scrollable if all items do not fit on the screen. The right side of the screen will contain a message prompting the Customer to select a Menu Item from the List. Once an item is selected, the right side of the screen displays everything needed for a user to fully customize any order. The Customer will see a description of the Menu Item selected, including certain toppings or ingredients as well as the price of the item. Beneath the description is a text box for the Customer to leave requests for the Chef about the order. A typical request may be to cook a burger to medium rare, or for an extra sauce or dressing on the side. The Customer will also be able to choose the side to accommodate the meal. These options are presented as radio buttons for the Customer to choose from. Next to the Meal Sides is a button to add the Meal to the Order List. If the Customer does not want to add this order, then he or she can click either a different Menu Category or a different Menu Item and repeat the process. When the Customer adds the Meal to the Order List, the Android Tablet will display a notification of the successful addition and will also say that unwanted items can be removed from the View Order Menu.

The View Order Interface (Fig-6) can be selected from the Menu Interface by selecting the button in the top right corner of the screen. The Menu Interface also provides options for the Customer to call the Waiter for help and to also cancel the entire Order. The Cancel Order option will warn the Customer of the action before committing. When the Customer has finished selecting from the Menu, the View Order interface will display a scrollable list of all items ordered. Next to each item will be a button to remove that item of the Order List. The Customer will be warned once again before the Android Application commits to the deletion. The View Order Interface will provide options for the Customer to Call the Waiter, to Cancel the Order, to Send the Order, and to Add to the Order. Selecting Add to Order will return the Customer to the Menu Interface and the above steps can be repeated. The Customer can finalize the order by sending it to the chef through the Send Order Option.

After selecting the Send Order Option the Customer will be returned to the Main Interface where the user can once again Order, View Order, Pay Bill, and Call Waiter. If the Customer attempts to View the Order, the same View Order Interface will appear once again but items already sent to the chef will not have Delete Buttons next to them. A waiter will need to override any items sent to the chef that are no longer desired.

The Customer will also be able to pay the bill from the Main Interface by selecting the Pay Bill Option. Selecting so will present the Customer with the Pay Bill Interface (Fig-7). The screen will display a scrollable List of all items ordered. Each item in the List will display the name of the Menu Item selected, the selected Side Item, the Note for Chef, and the Price of the item. Beneath the Order List is a Text Box for the Customer to specify a tip for the waiter. The Customer can also use the Easy Tip option located next to the Text Box, which will automatically add a ten, fifteen, eighteen, or twenty percent tip (excluding tax). Next to the Tip Options, the Bill Total will be presented. This Total will be the sum of all items ordered, all drinks, tax, and tip. The Customer cannot pay from the Tablet and must call the Waiter who will accept the payment and return a receipt and change.

Any time the Android Application is idle for 3 minutes, the Customer will see the Advertisement Interface. From here, the Customer will see a slide show of restaurant related images (food, drinks, logos, etc.) selected by the Manager. These images will change every 30 seconds until the Customer touches the picture. There will also be arrow options for the Customer to manually change the current image of the slide show. Clicking the image will return the Customer to the Main Interface.

Fig-8 displays a flow chart of how all Android Interfaces Interact. Several things are worth noticing: The Pay Bill Interface can only be reached from the Main Interface. This is because the Customer has no need of paying until an order has been sent. It is also important to note that the Customer cannot return to the Main Interface once the Order Process has begun until either the Customer cancels the order or sends it to the chef. Lastly, the Main Interface always has options to View Order and Pay Bill, even before an initial order has been placed. The Android Application will handle empty Orders by displaying a note on the screen saying no orders have been placed.

**Fig-4: Main Menu Interface (1)**

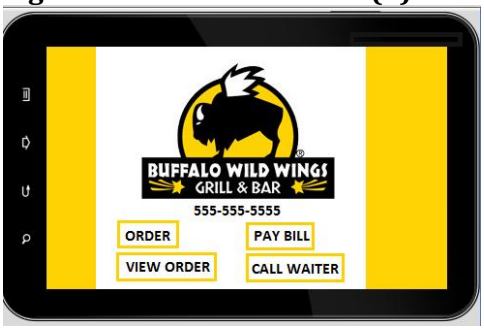



**Fig-5: Menu Interface (2)**


**Fig-6: View Order Interface (3)**

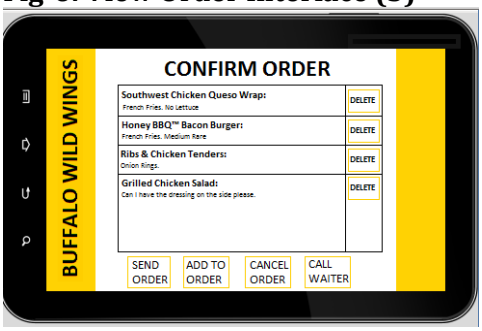
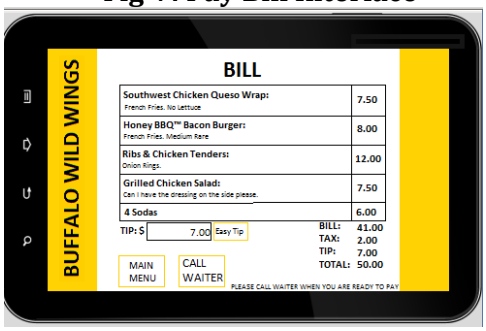



**Fig-7: Pay Bill Interface**



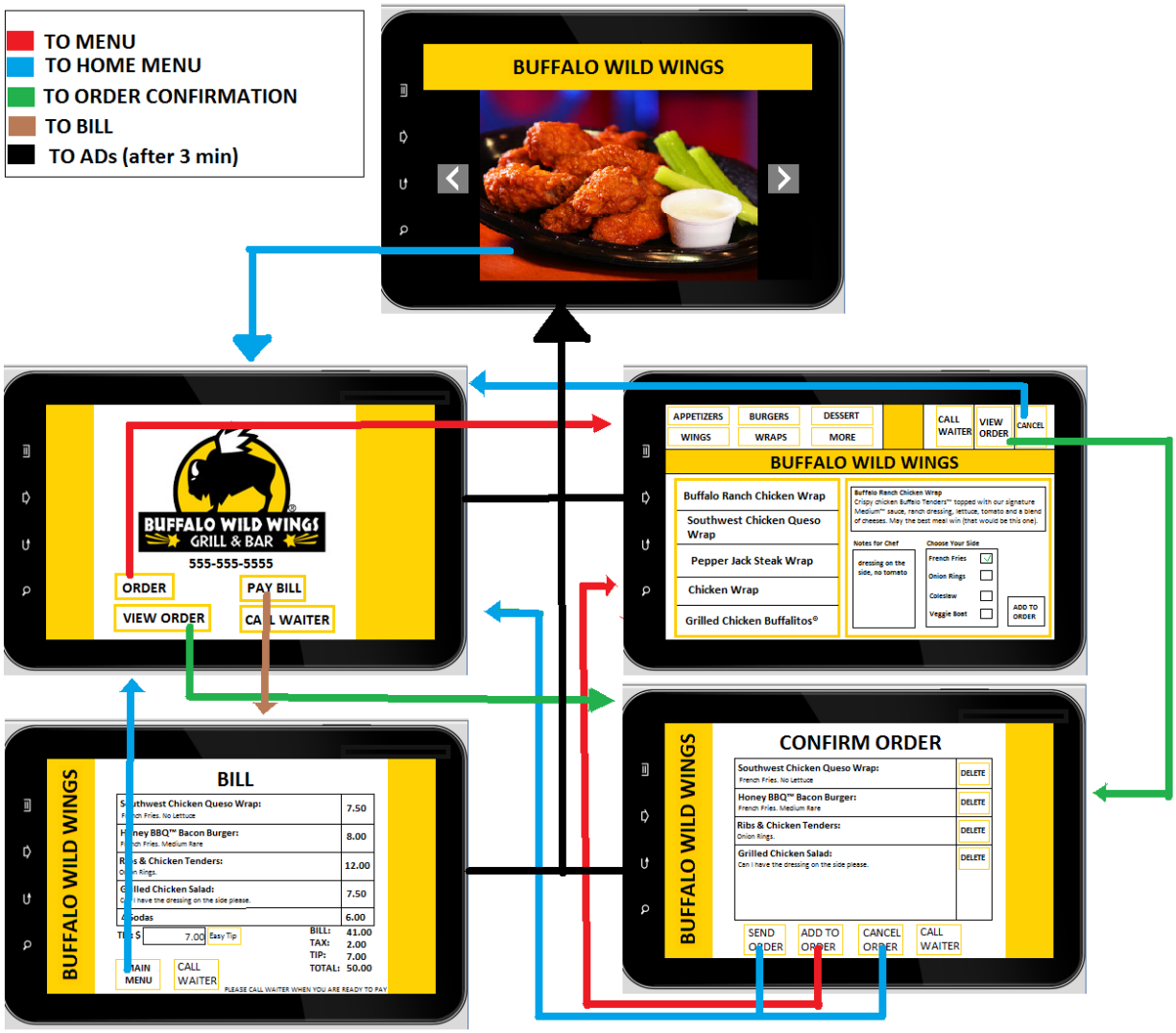
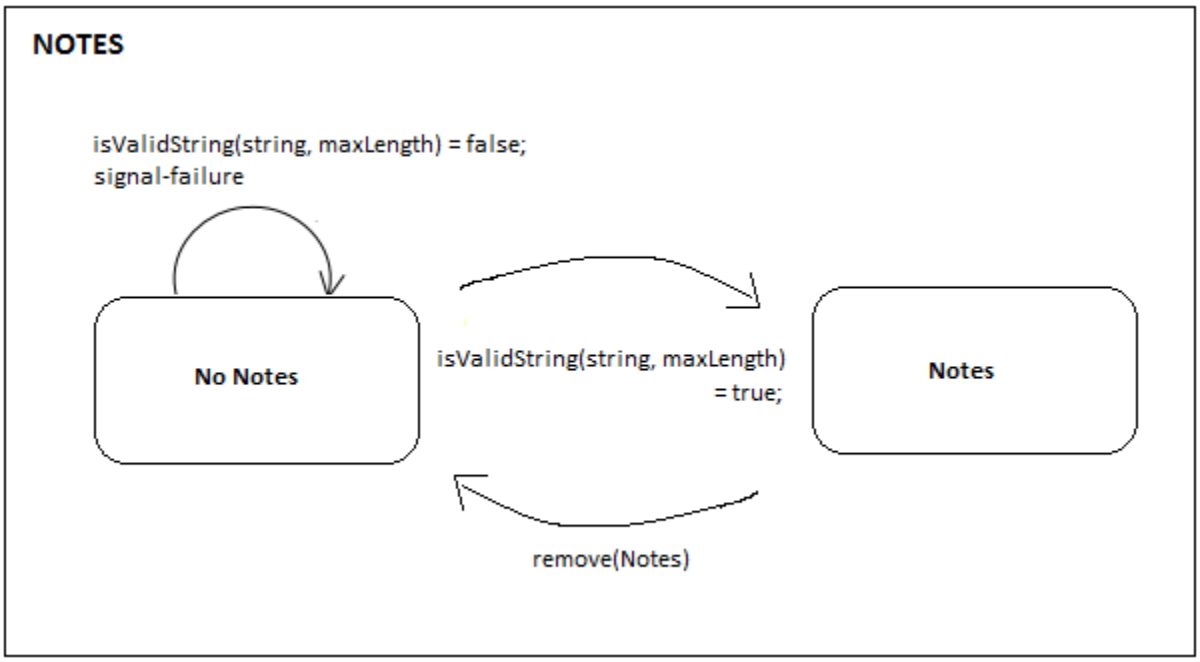


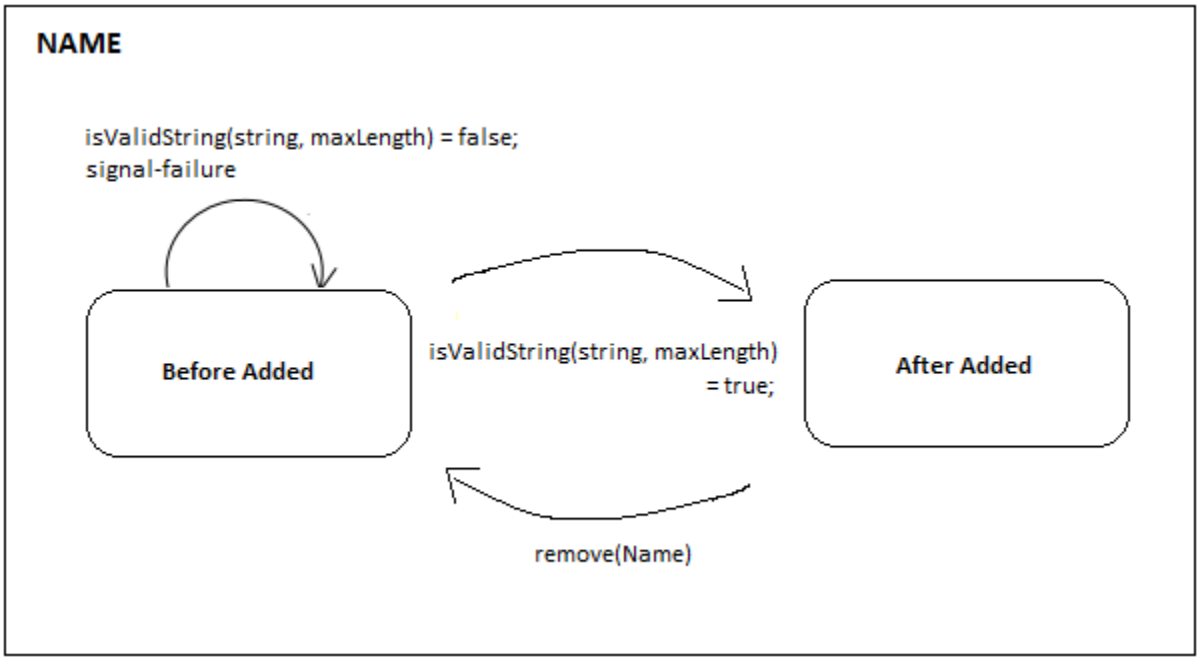

**Fig-8: Flow-Chart of Menu Interface: We can see that REQ17 will be addressed through this analysis. REQ25 and REQ22 are also completed through the above interfaces.**

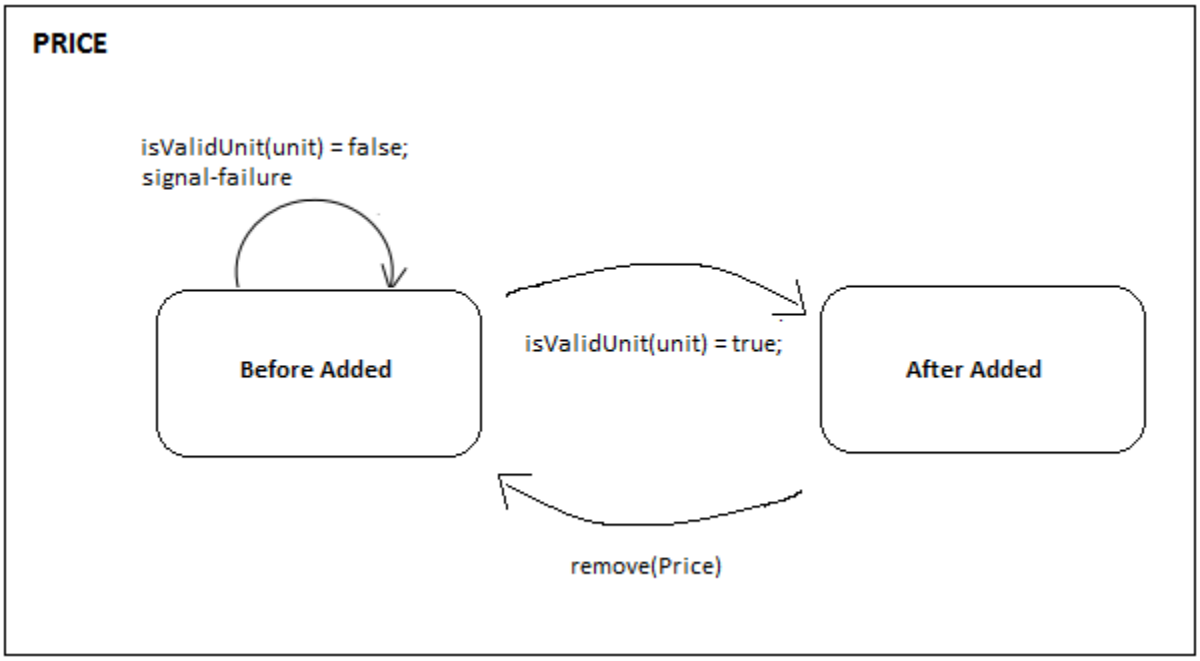
# 12. Design of Tests

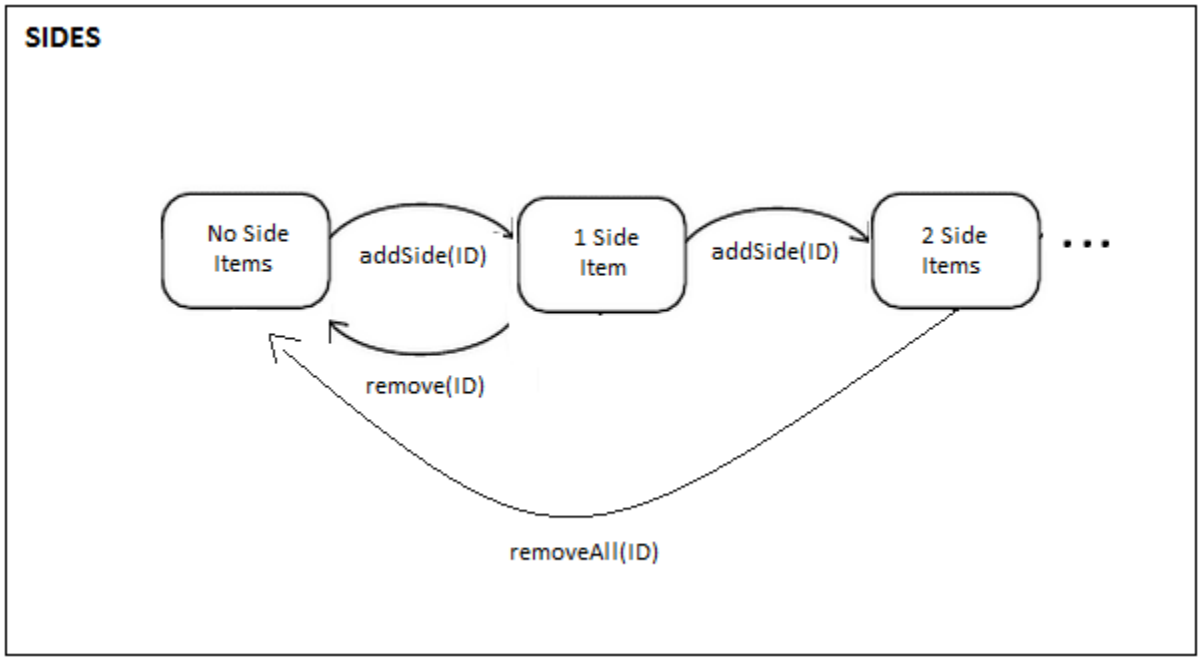
## State Diagrams

Below are four state diagrams drawn to give us a visual of how one class transitions from one state to another. Many of these states are similar to one another except for a slight difference in the attributes.




PRICE
isValidUnit(unit) = false;
signal-failure
Before Added
isValidUnit(unit) = true;
After Added
remove(Price)


SIDES
No Side Items
addSide(ID)
1 Side Item
addSide(ID)
2 Side Items
remove(ID)
removeAll(ID)

## Test Cases

| | |
|---|---|
| **Test-case Identifier:** TC-1<br>**Use Case Tested:** Login (UC-1), main success scenario<br><br>**Pass/fail Criteria:** The test passes if the user enters a pin that is contained in the database and system displays the correct interface for user.<br><br>**Input Data:** Keyboard with mouse | |
| **Test Procedure** | **Expected Result** |
| Step 1. Type in an incorrect PIN | System will display a message: “Invalid Pin.” |
| Step 2. Type in a correct PIN | System will direct user to the correct interface |

| | |
|---|---|
| **Test-case Identifier:** | TC-2 |
| **Use Case Tested:** | AddMenuItem (UC-8), main success scenario |
| **Pass/fail Criteria:** | The test passes if the manager adds a menu item to the potential item list. |
| **Input Data:** | Keyboard with mouse |
| **Test Procedure** | **Expected Result** |
| Step 1. Select Add Menu Item | System will prompt user to enter in new menu item information |
| Step 2. Leave a text field blank and hit Next | System will display an error message: “Fill in the missing text fields” |
| Step 3. Enter in new menu item information and hit Next | System will prompt manager to choose ingredients required for the meal along with quantities and the URL and the date that item will be added to the menu |
| Step 4. Leave ingredient text field blank | System will display an error message: “Must have at least one ingredient” |
| Step 5. Enter in an ingredient not stored in the inventory | System will display a warning: “Ingredient not in inventory. Add to shipment list” |
| Step 6. Enter in ingredients required for the meal along with quantities and the URL and the date that item will be added to the menu and hit Next | System will display the expected cost for each ingredient with the option to change price |
| Step 7. Enter a letter character in the price text field | System will display an error message: “Enter in a number” |
| Step 8. Enter valid price and hit Add Item | System will display food item on the potential item list |
| Step 9. Check potential item list | System will display list of potential food items plus recently added food item |

**Test-case Identifier:** TC-3
**Use Case Tested:** OrderFood (UC-16), main success scenario

**Pass/fail Criteria:** The test passes if the customer selects desired meal items from the food menu

**Input Data:**

| **Test Procedure** | **Expected Result** |
|---|---|
| Step 1. Select view menu from main screen | System will display a list of food items available to order |
| Step 2. Select a particular food category (“chicken”) | System will display a list of all food items for the selected category |
| Step 3. Select a particular food item from the chicken category | System will display item name, item price, list of available sides, and text field for “notes for chef” |
| Step 4. Hit add to order | System will display a message: “Food item added” |
| Step 5. Hit view order | System will display a list of ordered food items |
| Step 6. Hit the remove item button next to the item just added | System will removes item and refreshes view order list |
| Step 7. Hit add to order | System will display menu again |
| Step 8. Hit cancel | System will clear order list and display main menu screen |
| Step 9. Hit view order | System will display a message: “No items in order” |
| Step 10. Hit add to order and repeat steps 2-5 | System will display list of items ordered |
| Step 11. Hit send order | System will send order to chef and to the assigned waiter and returns to the main menu screen |

## Simple Unit Tests

Unit test deals with each component separately to ensure that a function works correctly. When the object under each test calls a method on dependency then it will change the state of the dependency as shown in our state diagrams. Since our main interest is in the state change, our concern is not whether or not a method is called. Below is a set of standard step procedure for each component to be tested with its expected result.

For Login:

loginUserName = {valid username, invalid username, empty}
PIN = {valid, invalid, empty}

| **Step** | **Expected Result** |
|---|---|
| Enter in a valid username and an invalid PIN | Failed to log in |
| Enter in a valid username and an empty PIN | Failed to log in |
| Enter in an invalid username and a valid PIN | Failed to log in |
| Enter in an invalid username and an empty PIN | Failed to log in |
| Enter in an empty username and an empty PIN | Failed to log in |
| Enter in a valid username with an incorrect valid PIN | Failed to log in (PIN does not match its username) |
| Enter in a valid username with a correct valid PIN | Logged in successfully |

For Name, when one enters the Name of a food item the test should confirm that 1) Name is a string and 2) the string Name does not exceed the maxNameLength. If any of these requirements are not met, the test will signal a failure. This step procedure applies for many similar components such as 'Description', 'URL', and 'ChefNotes' with a slight change in name for the call methods.

| **Step** | **Expected Result** |
|---|---|
| Call setName(string Name) where Name.length > maxNameLength | Name not added |
| Call setName(string Name) where Name.length <= maxNameLength | Name successfully added |

| **Step** | **Expected Result** |
|---|---|
| Call setDescription(string Description) where Description.length > maxNameLength | Description not added |
| Call setDescription(string Description) where Description.length <= maxDescriptionLength | Description successfully added |

Price will be tested in the same format. Several tests will be made:
1) Price will not be added if an empty field is called.
2) Price will not be added if Price is set to a string of characters or any other unrecognizable symbols.
3) Price will not be added if the price range exceeds a certain erroneous range.
4) Price will be added successfully if a call to setPrice has a double int Price as its argument.
Once again, this step procedure will be followed thoroughly for other components as well (such as 'CookTime', 'Amount', 'Cost', 'UnitCost', etc.)

| **Step** | **Expected Result** |
|---|---|
| Call setPrice(double Price) where Price.unit is blank | Price not added. Must insert a double int |
| Call setPrice(double Price) where Price.unit is | Price not added (Invalid) |
| Call setPrice(double Price) where Price.unit > maxPricelength | Price not added (Price set to an unreasonable price) |
| Call setPrice(double Price) where Price.unit is a double int | Price successfully added |

## Integration Testing

We focused on the incremental, top-down approach for the Manage Menu design. This was a feasible approach for us since it organized the modules into separate entities and allowed us to check for data flow between each module. We focused on the testing of highest level modules first and worked our way down with the use of stubs for low-level modules.

The main manage menu interface contains three buttons:
1) Add Item 2) Print Menu 3) Remove Item

If the user selects "Add Item", the link navigates to another screen. There are several text fields for the user to fill in (item name, item description, item price). To test this part of the module, we enter data in individual text fields (Ex. Cheeseburger, Tasty, $3.50). We check for correctness of functionality by referring to the database. The item "Cheeseburger" and its properties appear in the database, since that is what the 'Add Item' function was implemented to do. Later on, we hope to insert more fields such as "Food Type" and create a drop down of "Ingredients List". We are in the middle of implementation for both of these functions except they are not fully functioning yet.

Separate modules were designed, also, to automatically generate an item ID and a URL for each item. While our implementation of these methods does not take into account of adding this information into the database, we did manage to test for its functionalities.

When we ran our “Add Item” module on an Eclipse platform, an item ID of 100007 was created for item “Cheeseburger”. At the moment, food type is initialized to “Burgers” for testing purposes. Our goal is for the system to automatically generate the correct output by referring to the database and doing a quick sort to locate the specific food type and do a count of the items listed under that food type. In our case, we had already stored six other items under food type “Burgers” with item IDs from 100001-100006.

Also, for “Create URL” a separate function is called to automate a URL based on the food item added. Basically, the URL is to generate a “www.google.com/search?q=” LINK where the name of the food is inserted shortly after. For instance, when item “Cheeseburger” was added to the list, we test for the functionality of the code by checking that the URL appears (which it does). It creates a URL for cheeseburger: “www.google.com/search?q=Cheeseburger”. Our next goal for demo 2 is to have the URL and item ID inserted directly into the database.

Another module that we designed is “Print Menu” where its screen displays a list of items contained in the database. While this is the main functionality of this module, our implementation for the time being only focuses on printing out the recently added menu items on the screen. To test data correctness, the item “Cheeseburger” does successfully appear on the screen. The user, then, has the option to 1) “Edit Item” information or 2) “Remove Item” from database. For demo one, the edit module has not been implemented. However, a separate module is designed for “Remove Item”. When the user selects “Remove Item” button next to the item “Cheeseburger”, this should call method “Remove Item”. This can easily be tested by referring to the database; likewise the item “Cheeseburger” does no longer exist. One thing that we are still working on is to get the “Print Menu” page to refresh each time an item is removed from the database. At the moment, item “Cheeseburger” is still being displayed on the list even though it is no longer in the database. However, if user tries to

remove the same item twice, the system will prompt a message saying that item “Cheeseburger” is no longer in the database.

Lastly, the third button on the main Manage Menu interface is also the “Remove Button”. During the actual integration testing, we happened to encounter bugs that still need to be fixed. However, the procedures for integration testing are as follows. If the user chooses to remove item right after the item has been added, it can do so without having to go to the print menu. Please keep in mind that our main purpose of “Print Menu” is to display ALL items stored in the database. So, this separate button for “Remove Item” will come in handy when the user does not have to scroll through all the items when it can instead remove the most recently added item. This functionality was tested shortly after item “Pizza” was added. While “Print Menu” displays (Cheeseburger, Pizza) and the option to remove either item, the “Remove Item” button found on the main page will know to remove item “Pizza” from the database. Also, if user tries to remove “Pizza” again from “Print Menu”, the system will prompt a message saying that item “Pizza” is no longer in the database.

# 13. History of Work, Current Status, & Future Work

Well we have successfully implemented about 6 out of 24 use cases compared to Report 1. There were a lot of additions in ideas since Report 1 which changed use cases here and there. However starting Report 2 we fell on the right track as we started implementation. As per future work, we will be continuing the project as our senior design Spring (2012) the same team with the exception of Bill Fung as he will be graduating. We have not changed the project specification much since we would like to implement it and maybe add more ideas in the future when we start on it again.

## Key Accomplishments

- Learned Android SDK.
- Learned a lot about Database interaction.
- Learned how a software project is implemented in real world scenario.
- Learned a lot about team work.

# 14. References


**Useful Information From:**

1. 1. Software Engineering by Ivan Marsic, Rutgers University
   http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf

**Pictures From:**

1. http://www.bostonbakesforbreastcancer.org/on-the-chopping-block/
2. http://www.chasestockton.com/2010/10/free-vector-cartoon-face/
3. http://vi.sualize.us/inguyenvu/red%20head/
4. http://www.principalspage.com/theblog/archives/you-never-forget-your-first-love-especially-if-he-is-a-waiter
5. http://www.picturesofsmileyfaces.info/
6. http://www.bww.com/

7. 2. http://www.itechnews.net/2010/04/01/cisco-valet-wireless-routers-with-flips-simplicity-in-design/
8. http://www.softicons.com/free-icons/system-icons/mac-icons-by-artua.com/intranet-icon
9. http://www.iconarchive.com/show/vista-hardware-devices-icons-by-icons-land/Computer-icon.html
10. http://androidcommunity.com/toshiba-10-1-inch-android-tablet-gets-fully-detailed-20110318/

**Links that help with the report:**

11. http://en.wikipedia.org/wiki/Intranet
12. http://en.wikipedia.org/wiki/Data_structure

# 15. Appendix

## Interview

We called the manager of Buffalo wild wings because our use case 8 was confusing and we wanted to know how it is actually done. And here were the questions we asked (His name was Matt and his phone number is 732-297-9413):

- How do you add items to the menu?
  - All items are chosen by corporate, so manager doesn't have control of what items go on the menu.
- What happens if a chef doesn't know how to cook an item?
  - As a manager, he is responsible for knowing everything about the menu including how to cook each item. He goes through training for this. If a chef does not know how to cook, then the manager has to step in. From corporate, he also gets a recipe book about how to cook each item.
- What do you look for in a sales report? How often do you run a sales report?
  - Sales reports are run each night at closing. You do a weekly report with each nightly report to check if it adds up.
- Do you look at each item when you run a sales report?
  - No, I just look at net profit and gross profit.
- How often do you restock your inventory?
  - Every Wednesday and Saturday regardless of demand. Always have a surplus.
- How do you manage inventory?
  - He hates the fact that he has to go in and manually check each ingredient and see what you need more of for the next shipment by paper and hand.
- How do you make sure you have enough inventory?
  - Compares it with last years and depending on that orders. Maintains a surplus.
- Is there anything you would suggest to make your job easier?
  - Wants to use tablet menus.
  - Automate inventory checking.
  - Hardest part about being manager is dealing with customers.

## Changes made to the project

Based on the manager's feedback we have modified our project to better serve the end-user's needs. Firstly looking at his suggestions, we already have two of the three suggestions specified under our system. For the third part, dealing with customers cannot really be automated at this point. It is something we cannot really help the manager with. Also using his suggestions, we had to make changes to Use Case 8 and Use Case 16. For Use Case 8 – Add Item, the chef had to be removed as an initiating actor since the manager has complete power over this. And for Use Case 16 the addition of the inventory system to our project has caused a major change to the system side of Use Case 16 as the system automatically takes into account availability of ingredients and subtracts them as well. As for the sales report we will be changing how that is implemented in our system in the next iteration.