Auto Order 2.0

Group 2

Report #1: System Design

Website: https://sites.google.com/site/restautomation/

"The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

– Fred Brooks

Group Members:
Praveen Chekuri

Pradnya Pisal

Zachary Brown

Kartik Bhatnagar

Bill Fung

Individual Contribution Breakdown

Task/Group Member	Praveen	Pradnya	Bill	Kartik	Zac
Project Management (15 points)	6	4	3		2
Sec 1: Interaction Diagrams (30 points)		6	10	14	
Sec 2: Class Diagram & Interface Specification (10 Points)	3				7
Sec 3: System Architecture & Design (15 Points)	3	4	2	6	
Sec 4: Algorithms & Data Structures (4 Points)			4		
Sec 5: User Interface Design & Implementation (11 Points)	6				5
Sec 6: Design of Tests (12 Points)		6			6
Sec 7: Plan of Work (2 points)	2				
Sec 8: References (1 Point)			1		

Individual Point Allocation

Praveen Chekuri	20
Pradnya Pisal	20
Bill Fung	20
Kartik Bhatnagar	20
Zachary Brown	20

Table of Contents

Re-Iteration of Report 1

Interview

- done by Zac Brown

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.

Use Cases

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. 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.

←10. (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.

→11. Manager confirms price and hits the Add Item button.

←12. (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.

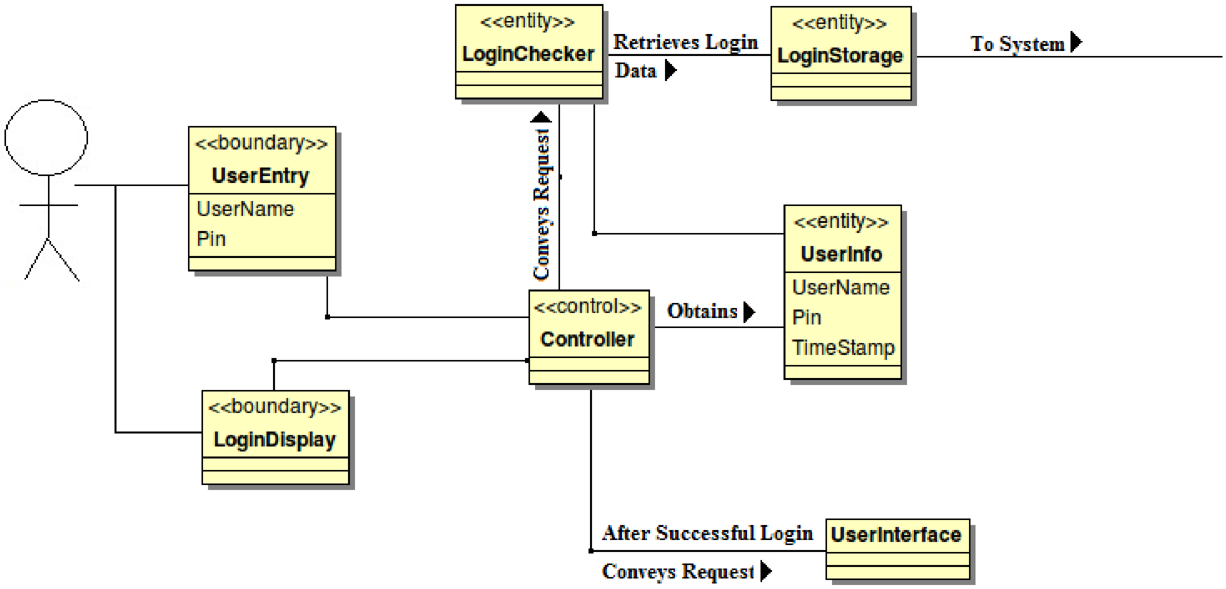
Flow of Events for Extensions (Alternate Scenarios):

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.

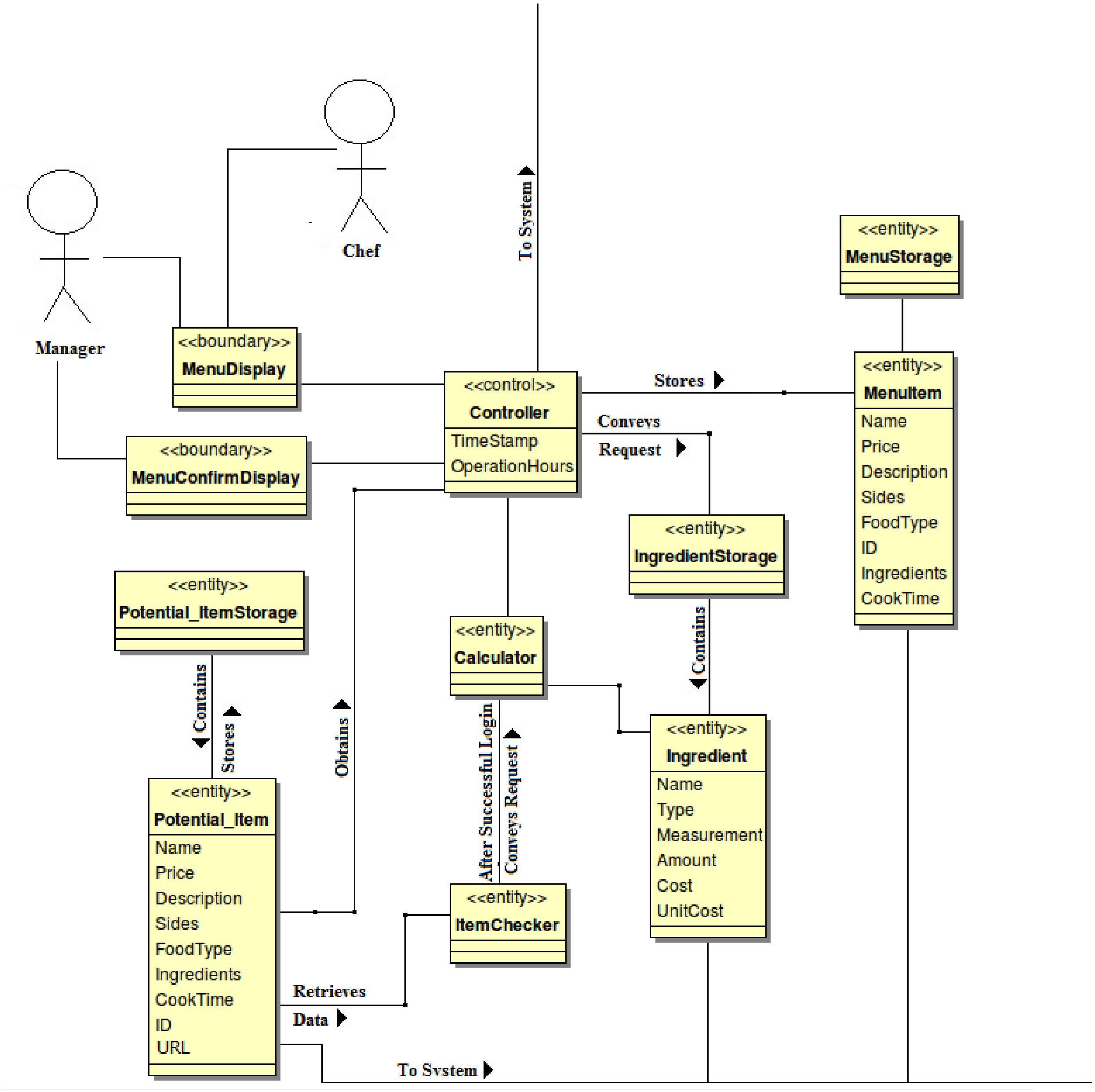
Domain Models

This section contains the detailed domain models which are part of the system. These detailed domain models help us better understand how the specific use cases perform.

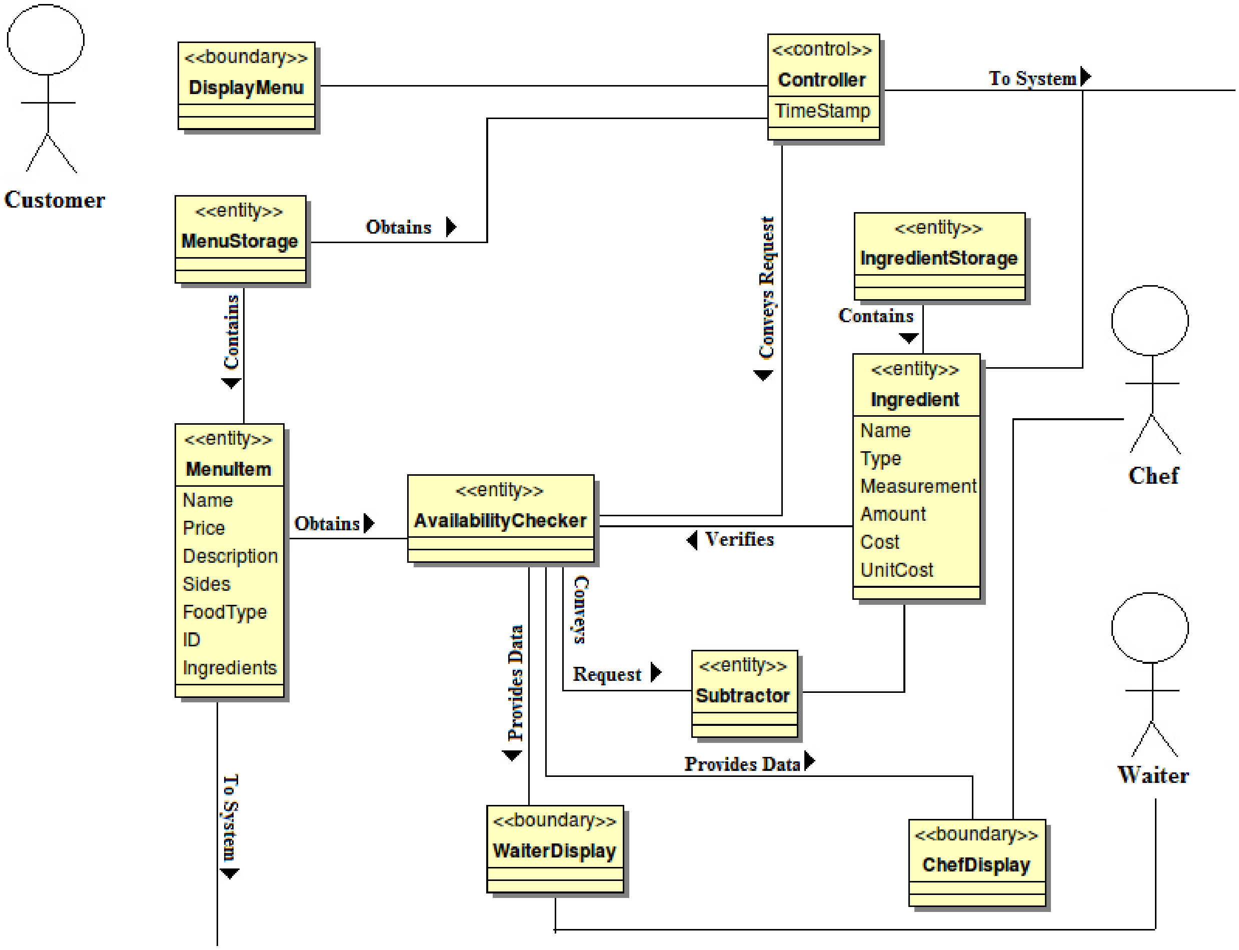
Use Case UC-1: Login



Use Case UC-8: AddItem

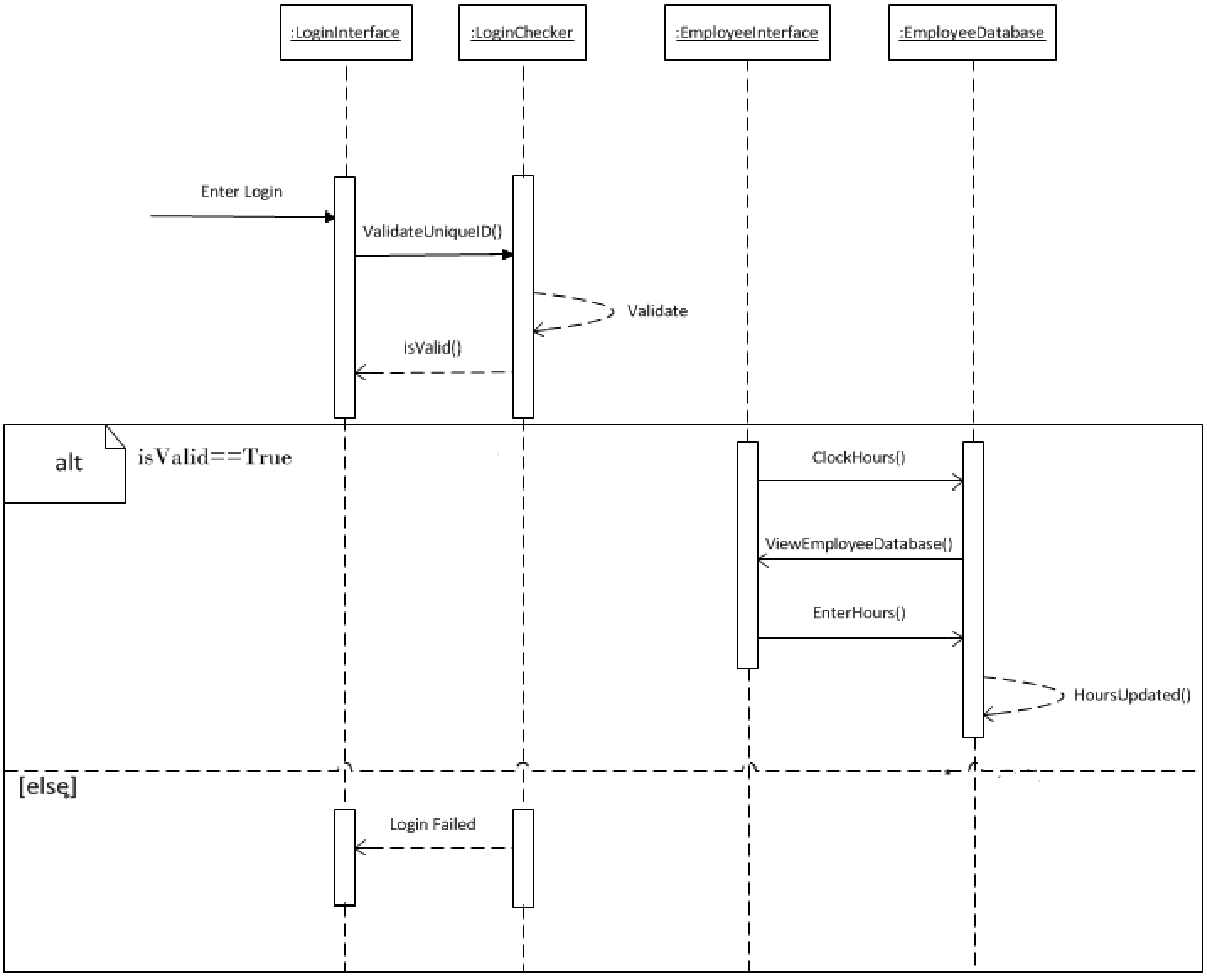


Use Case UC-16: Order Food



Interaction Diagrams

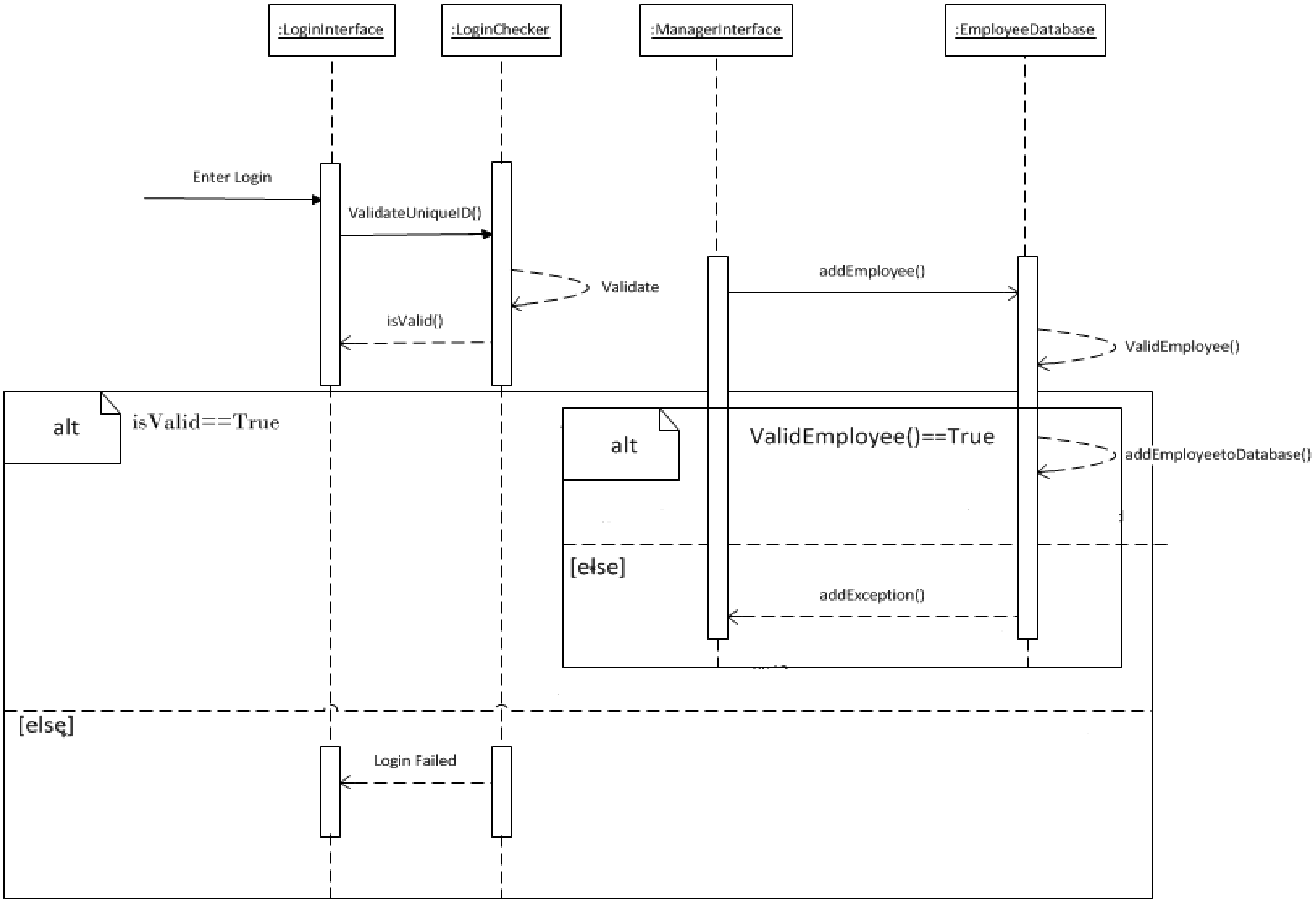
Use Case-2: Enter Hours



Responsibilities:

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.

Use Case-4: Add Employee

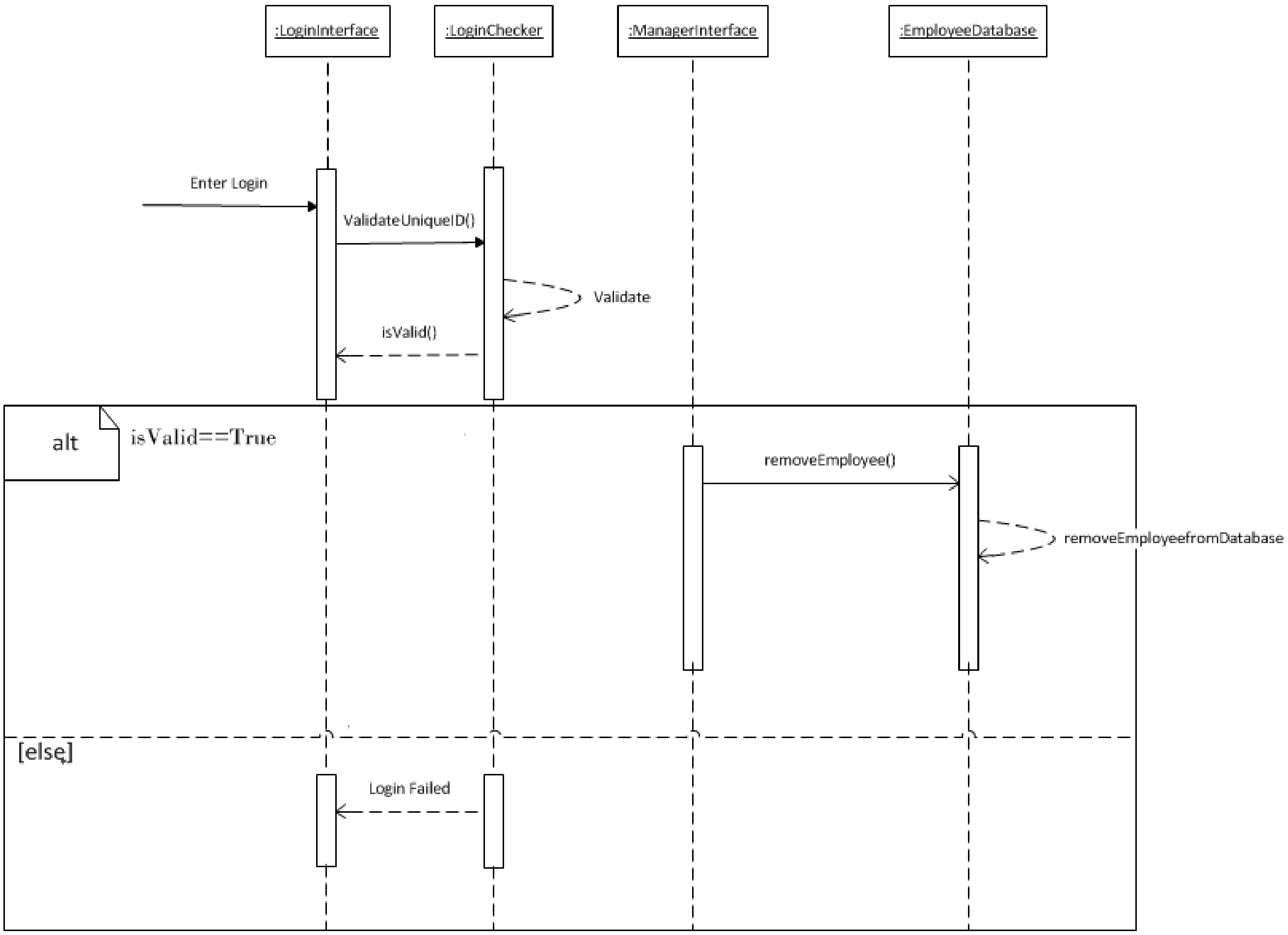


Responsibilities

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 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

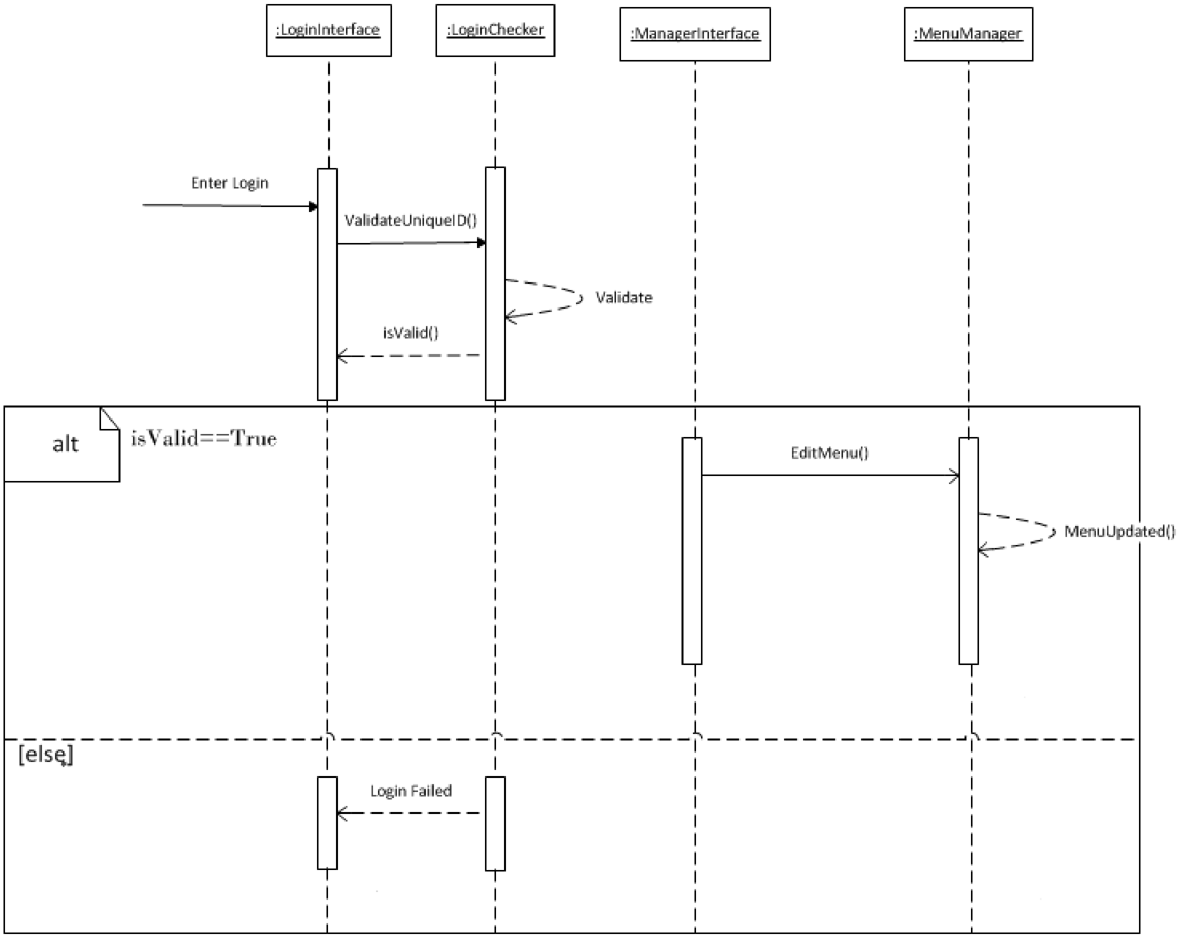


Responsibilities

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.

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

Use Case-7: Manage Menu

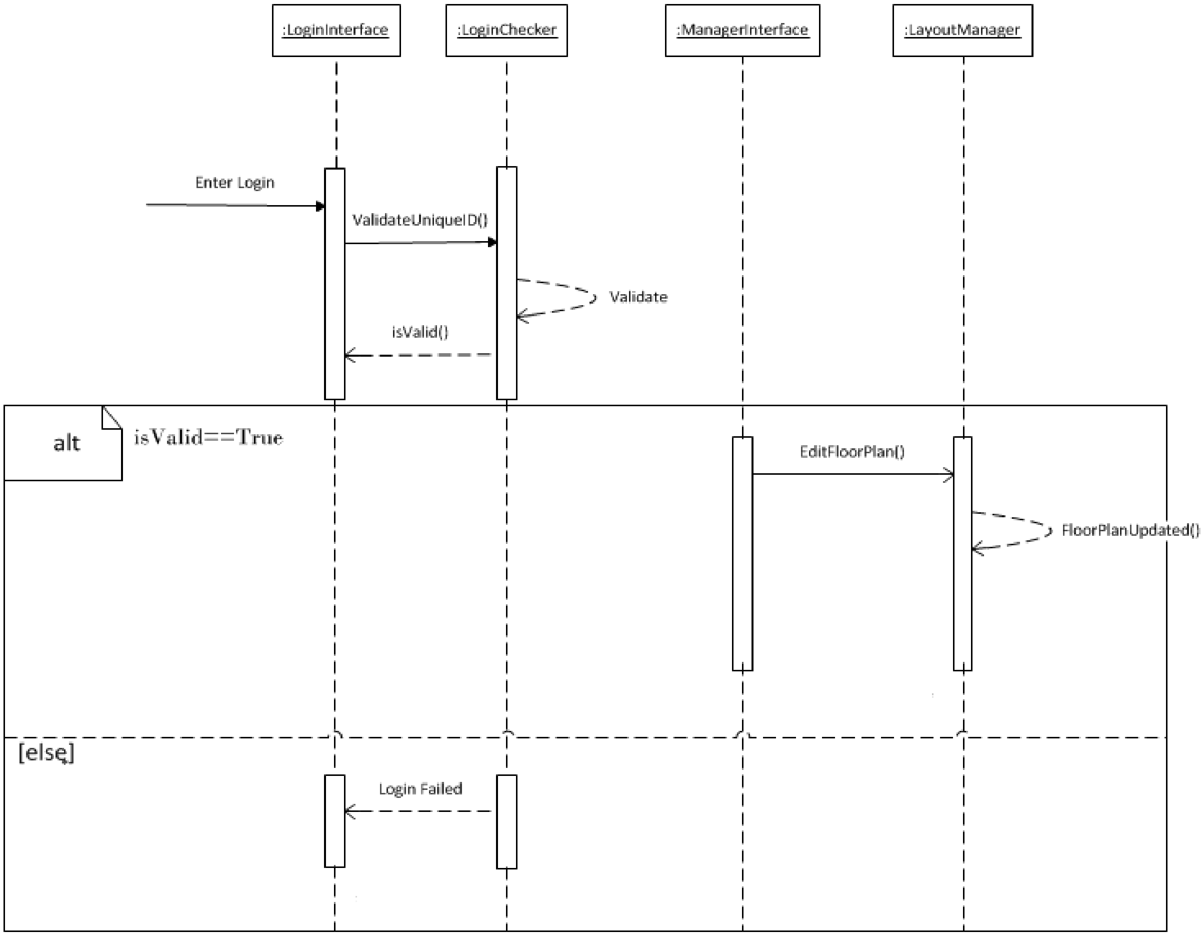


Responsibilities

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.

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

Use Case-11 Manage Layout

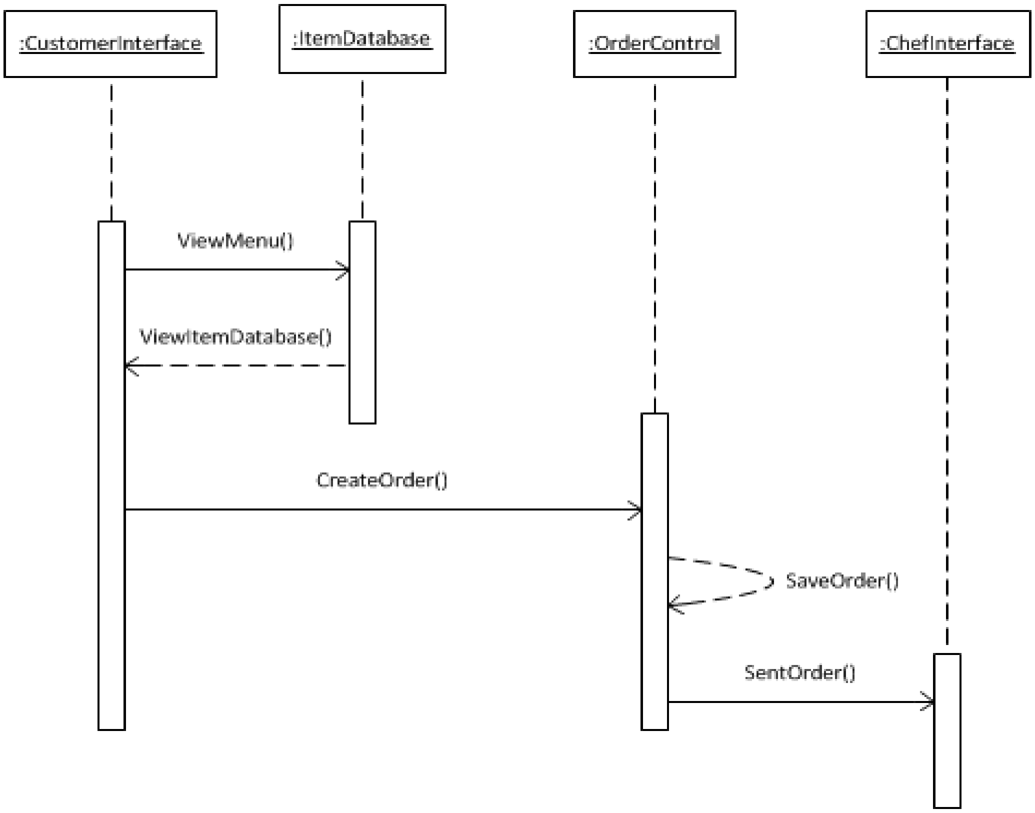


Responsibilities

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.

The system will update with FloorPlanUpdated() method.

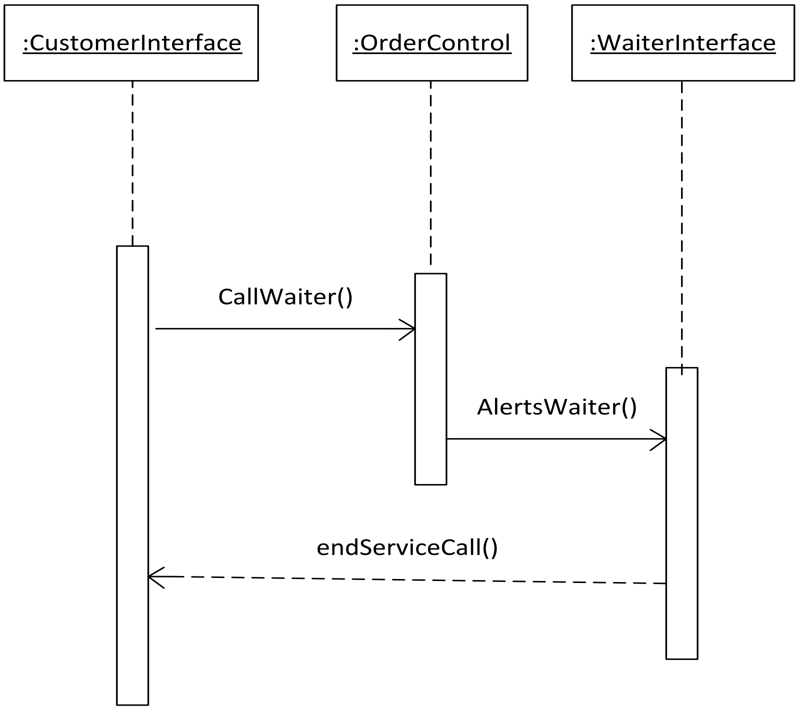
Use Case-16: Order Food



Responsibilities

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. Tthe Customer will create an order of the items that they prefer with CreateOrder().
4. The order will be saved and sent to the chef for cooking. This is done with the method SentOrder() to the chef.

Use Case-18: CallWaiter

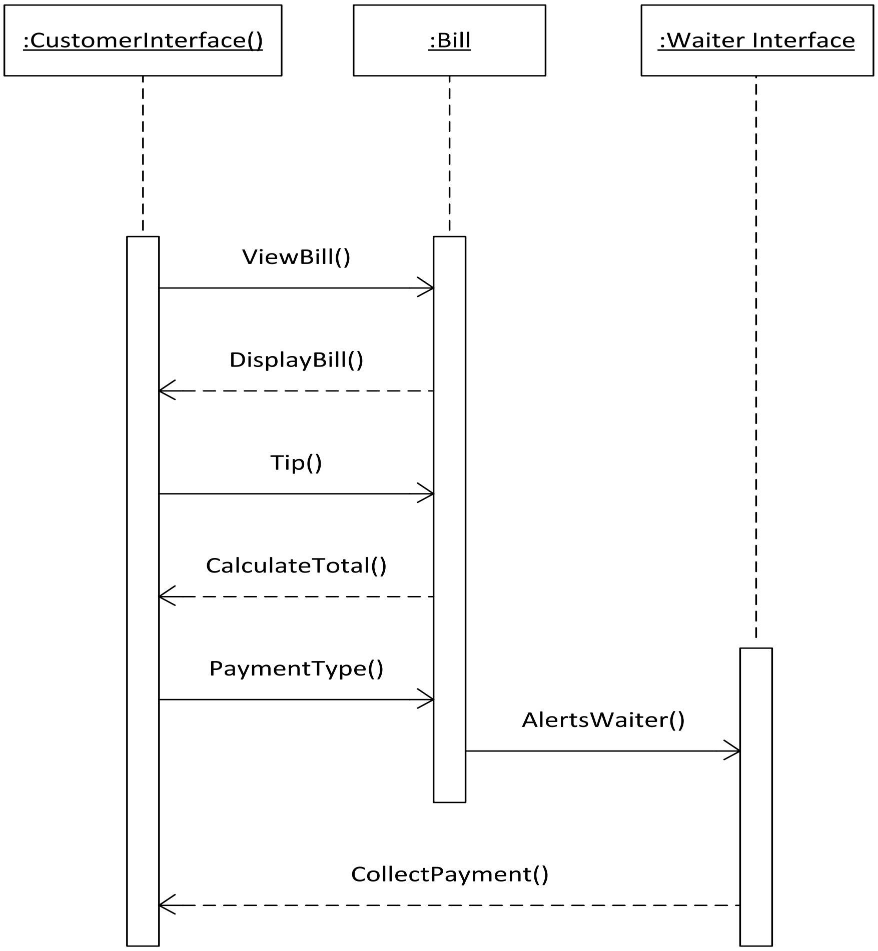


Responsibilities

1. The customer wants to call the waiter for help by using the method CallWaiter() thru the OrderControl system.
2. The system will then notify the waiter to assist the customer in need.

The waiter will assist all needs of the customer and then end the service call on the system by using the method endServiceCall().

Use Case-20: Pay Bill

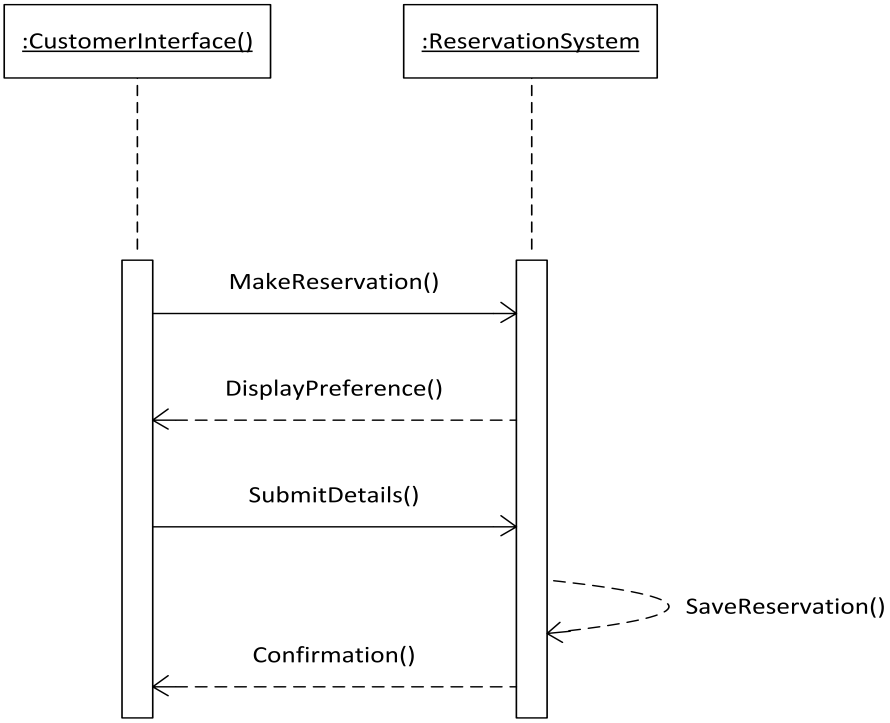


Responsibilities

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.

Use Case-23: Send Reservation



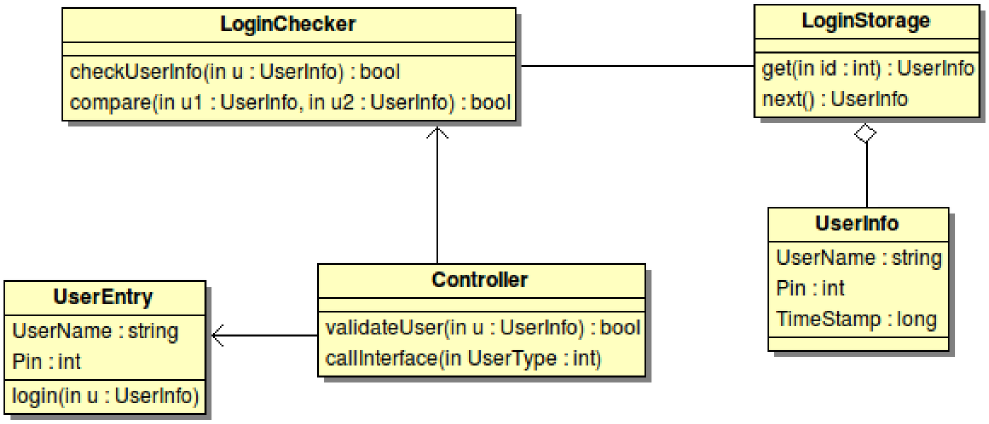
Responsibilities

1. The customer wants to make a reservation by using the PlaceReservation() method.
2. PlaceReservation() is completed, the system will display the preference with DisplayPerference().
3. Customer views the preferences and submits it by using the Submit() method.
4. The reservation made by the customer will be saved with the SaveReservation().

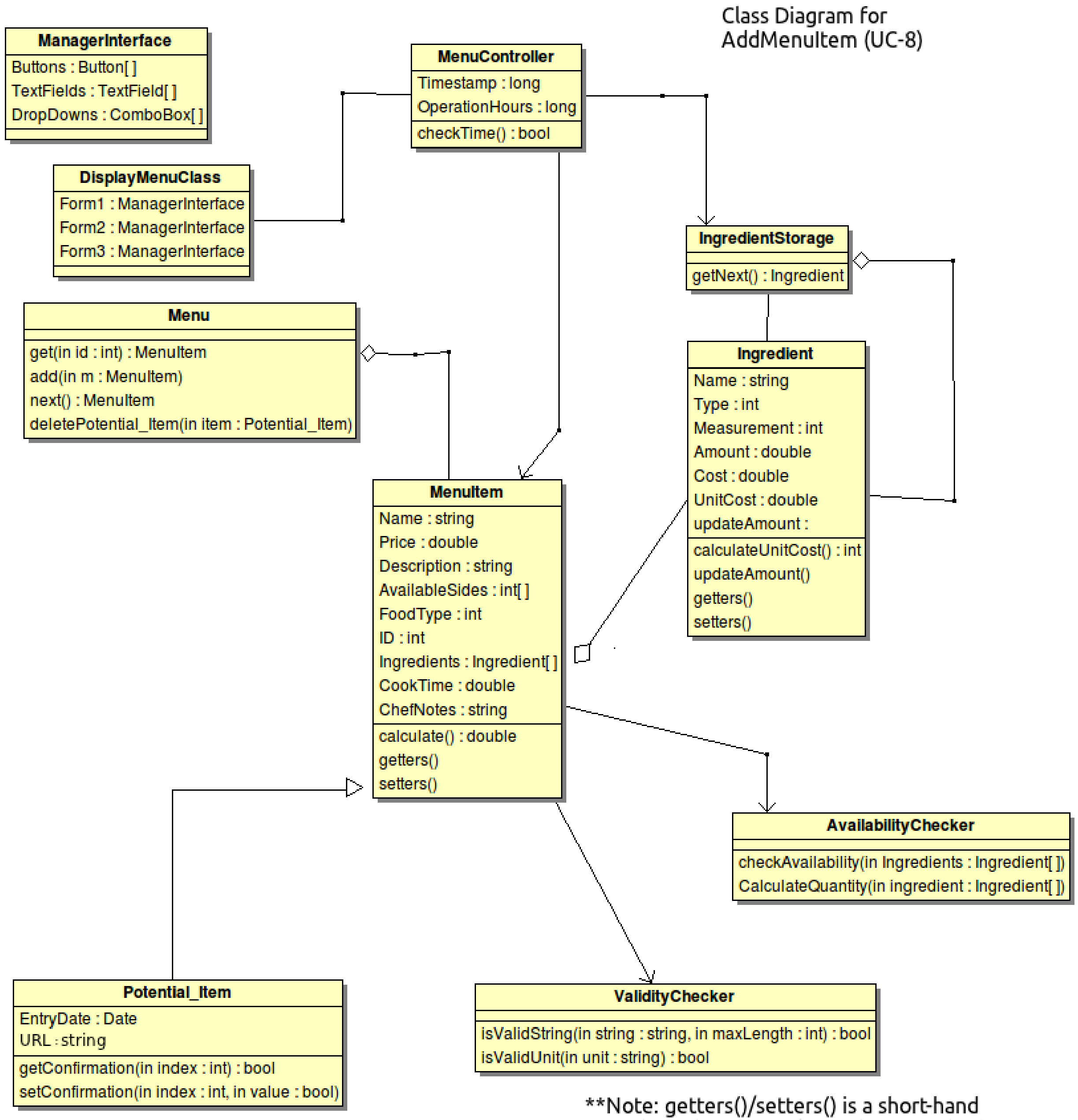
Class Diagram & Interface Specification

Class Diagrams

Use Case UC-1: Login


LoginChecker
checkUserInfo(in u : UserInfo) : bool
compare(in u1 : UserInfo, in u2 : UserInfo) : bool
LoginStorage
get(in id : int) : UserInfo
next() : UserInfo
UserInfo
UserName : string
Pin : int
TimeStamp : long
UserEntry
UserName : string
Pin : int
login(in u : UserInfo)
Controller
validateUser(in u : UserInfo) : bool
callInterface(in UserType : int)

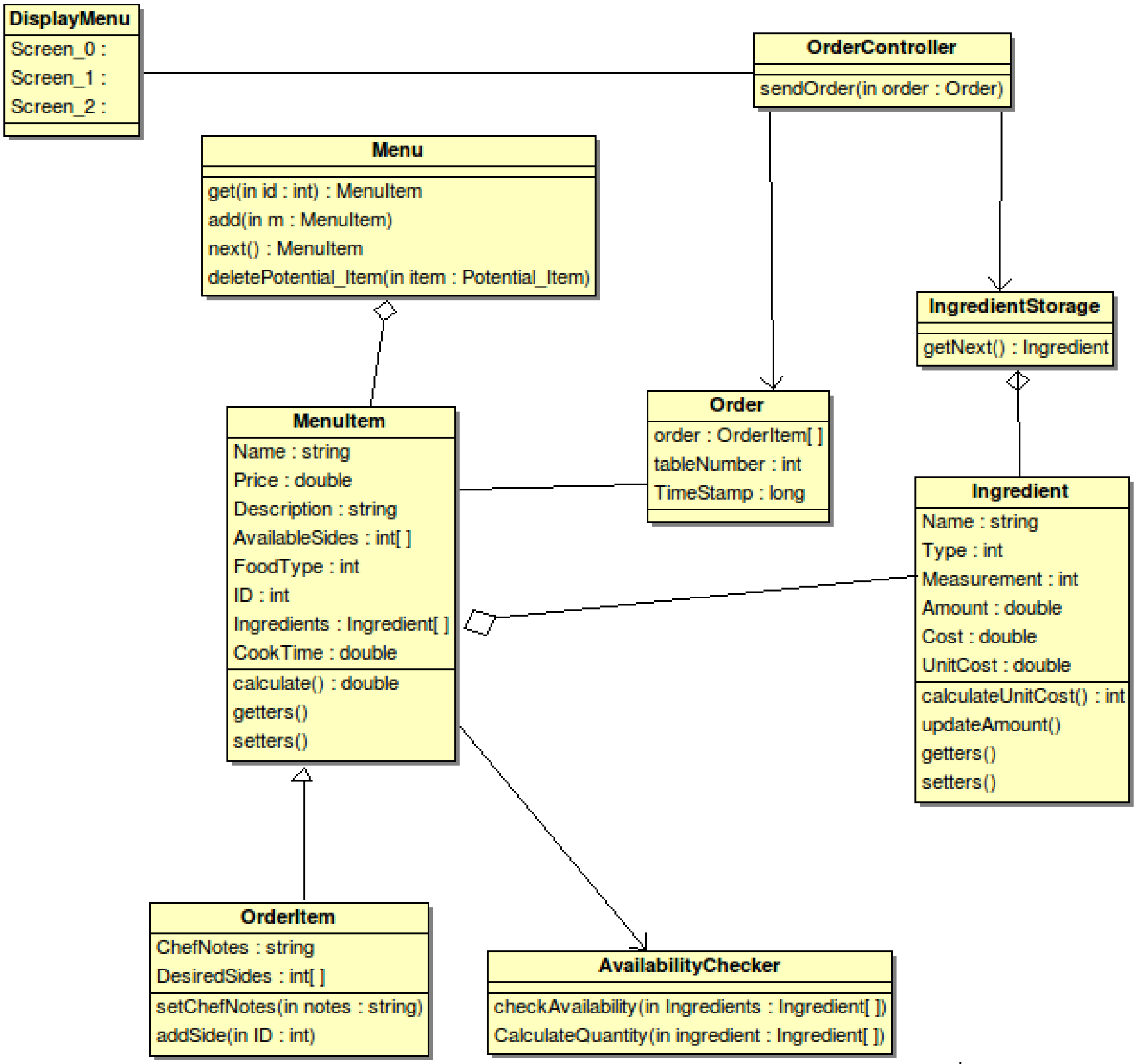
Use Case UC-8: AddItem


Class Diagram for AddMenuItem (UC-8)

**Note: getters()/setters() is a short-hand
to show that these classes have methods to
get/set a particular attribute.
Setters all go through ValidityChecker first

Use Case UC-16: Order Food

Class Diagram for
OrderFood (UC-16)



**Note: getters()/setters() is a short-hand
to show that these classes have methods to
get/set a particular attribute.
Setters all go through ValidityChecker first

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

Traceability Matrix

Use-Case UC-8: AddItem & Use-Case UC-16: OrderFood

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		

Above is 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.

System Architecture & System Design

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.

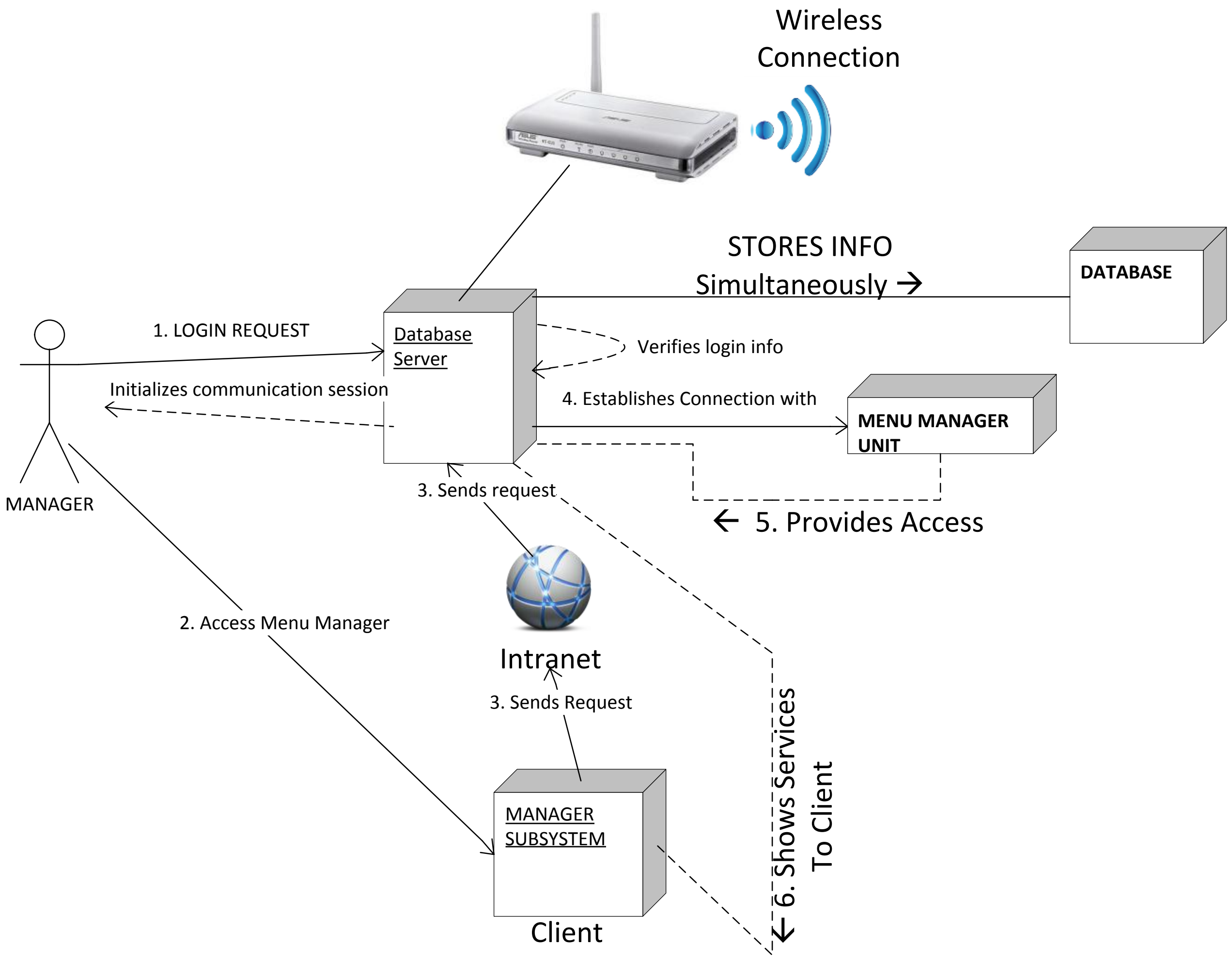


Figure1: Client Server communication process

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.

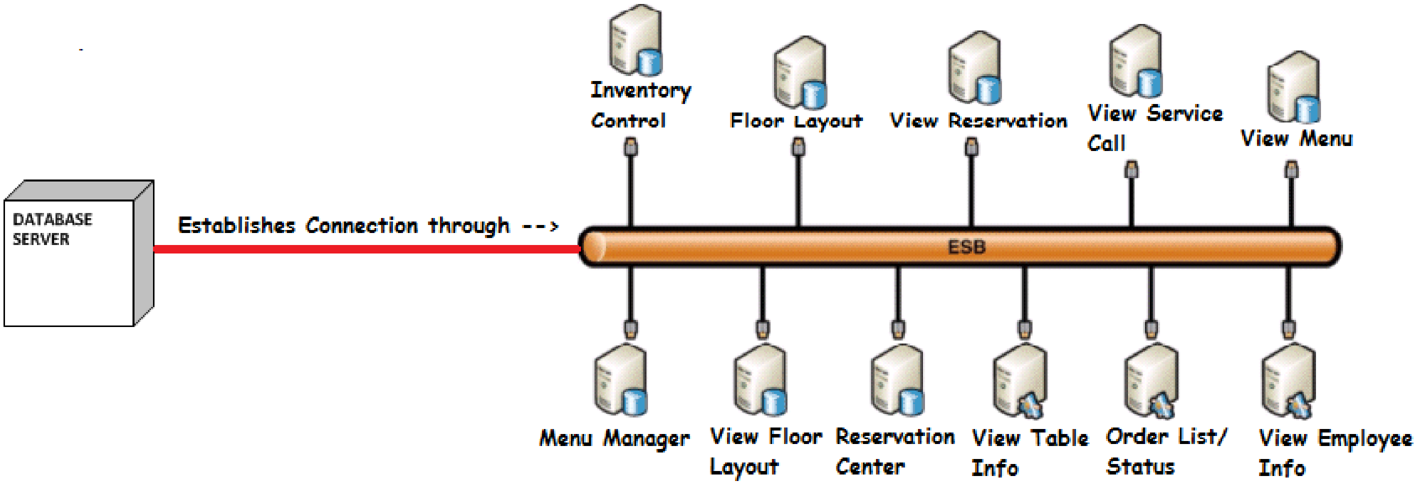


Figure 2: Implementation of ESB (Message Bus)

Figure 3 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.

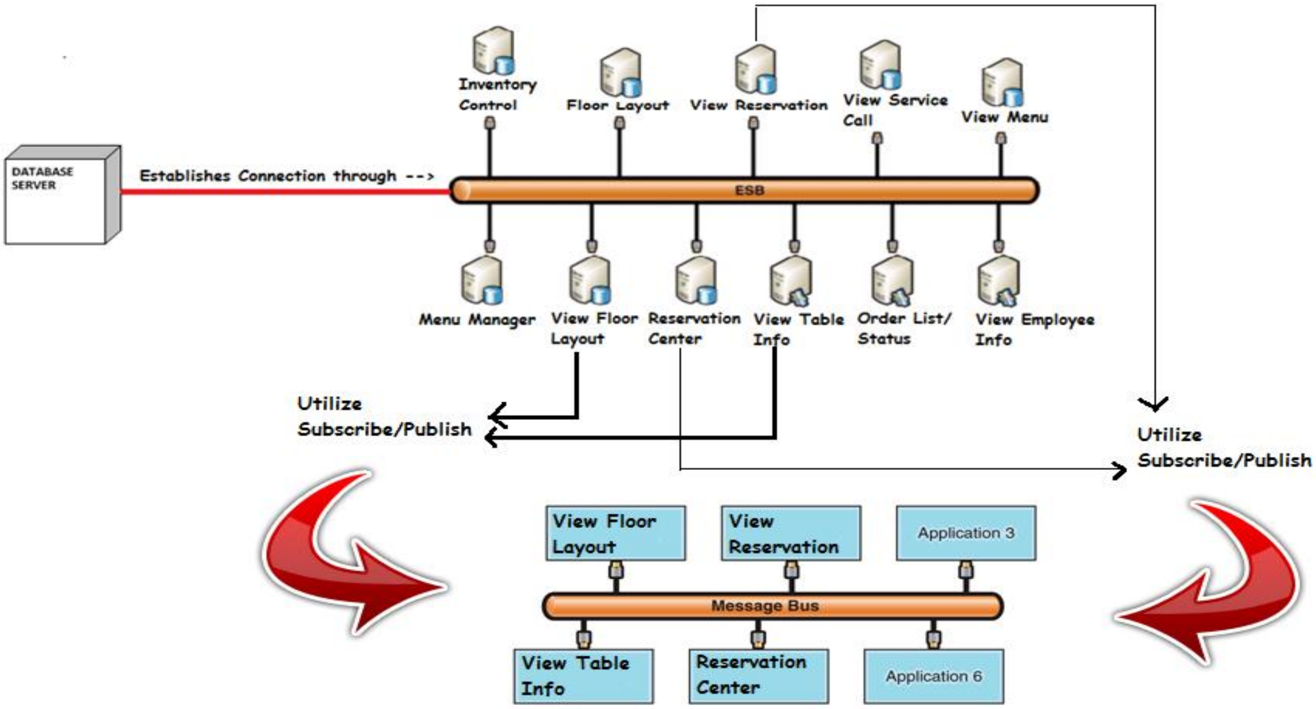


Figure 3: System utilizing Subscribe/Publish method (Message Bus)

As you can see, 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.

Identifying Subsystems

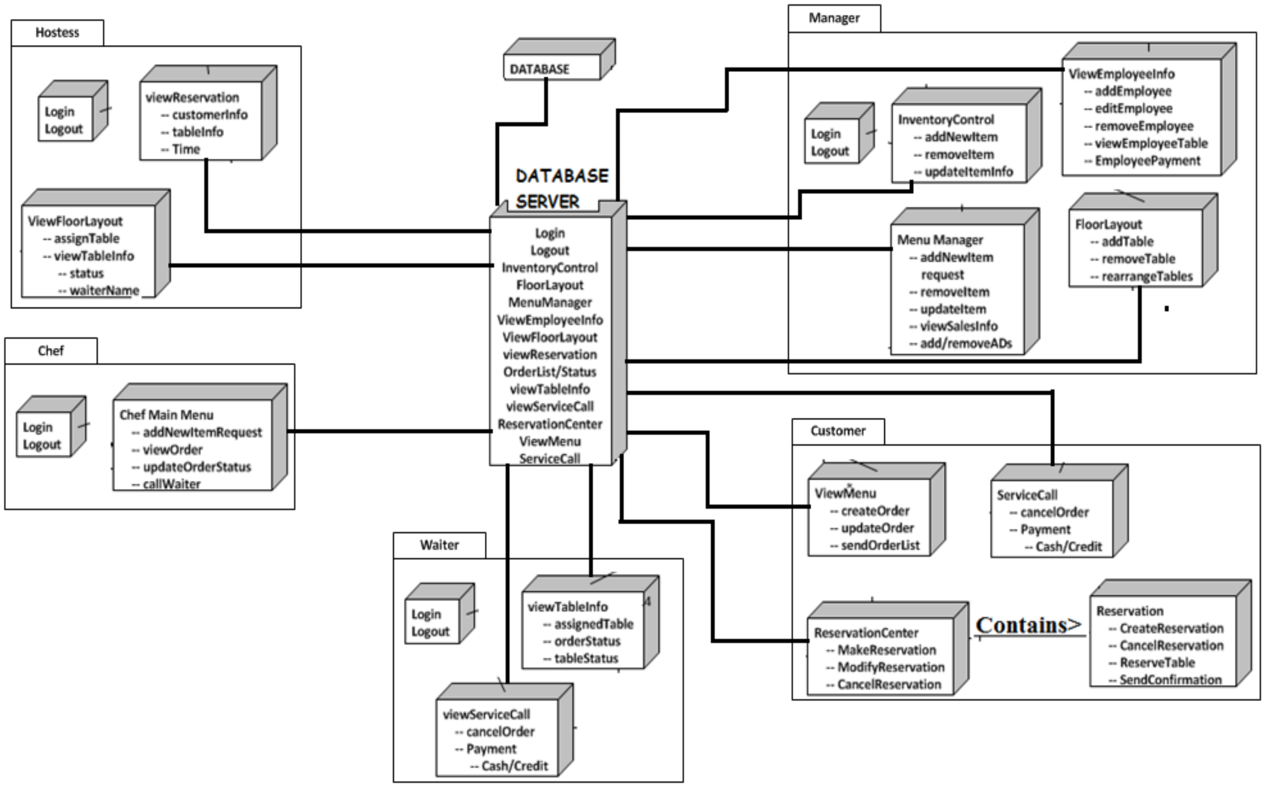


Figure 4: Identified Subsystems.

Identifying subsystems requires careful extraction of information from the use cases as well as the class diagrams. A particular criterion was considered to identify the subsystems, such as type of services each subsystem will provide and most of the interaction should be within a subsystem. Based upon this analysis, all the subsystems are classes, which provide access to specific functions. For ex: in the "Menu Manager" class, the appropriate personnel will have access to functions such as "addItemRequest", "removeItem", "updateItem", etc. Each of these is a separate class because of different functionality. It is easy to break up the system into smaller pieces for better organization and it will help avoid overlapping of functions that are similar in their working, but provide different services. Our system requires loosely-coupled dependency, which is mostly required for service calls and status updates of tables and orders. The higher the dependency rate, more difficult it is to organize it into simpler systems. Higher dependency also provides increased risk of complications because if one of the subsystems is malfunctioning, then the dependent subsystem will have low functionality, resulting in poor performance of the entire system. Figure 4 depicts the dependency of subsystems as well as the interaction with the server.

The folders are named customer, manager, hostess etc. to show who will have access to those subsystems. It is important to show accessibility because specific services are provided by specific personnel. The reader should also be able to understand who can access what subsystems. The idea of controlled access is portrayed here, which helps the system operate in an organized manner. The server has access to each subsystem in order to process request queries; user clients request access to certain functions and the server processes them by directing the request to the appropriate subsystem via ESB. Important data is stored in the server as well as in another database, which is available for backup purposes only.

Mapping Subsystems to Hardware

The restaurant automation system is set to run on multiple terminals that are also the identified subsystems. 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 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.

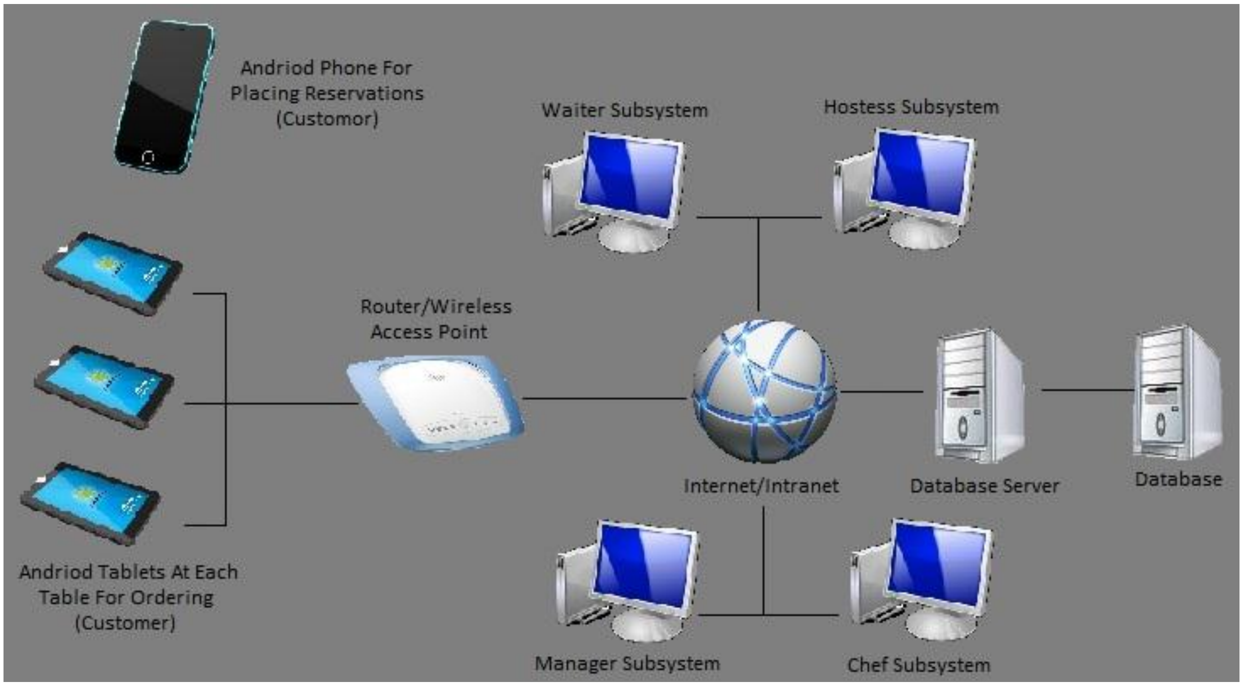


Figure 5: A high level network model along with hardware allocations is shown is the figure below

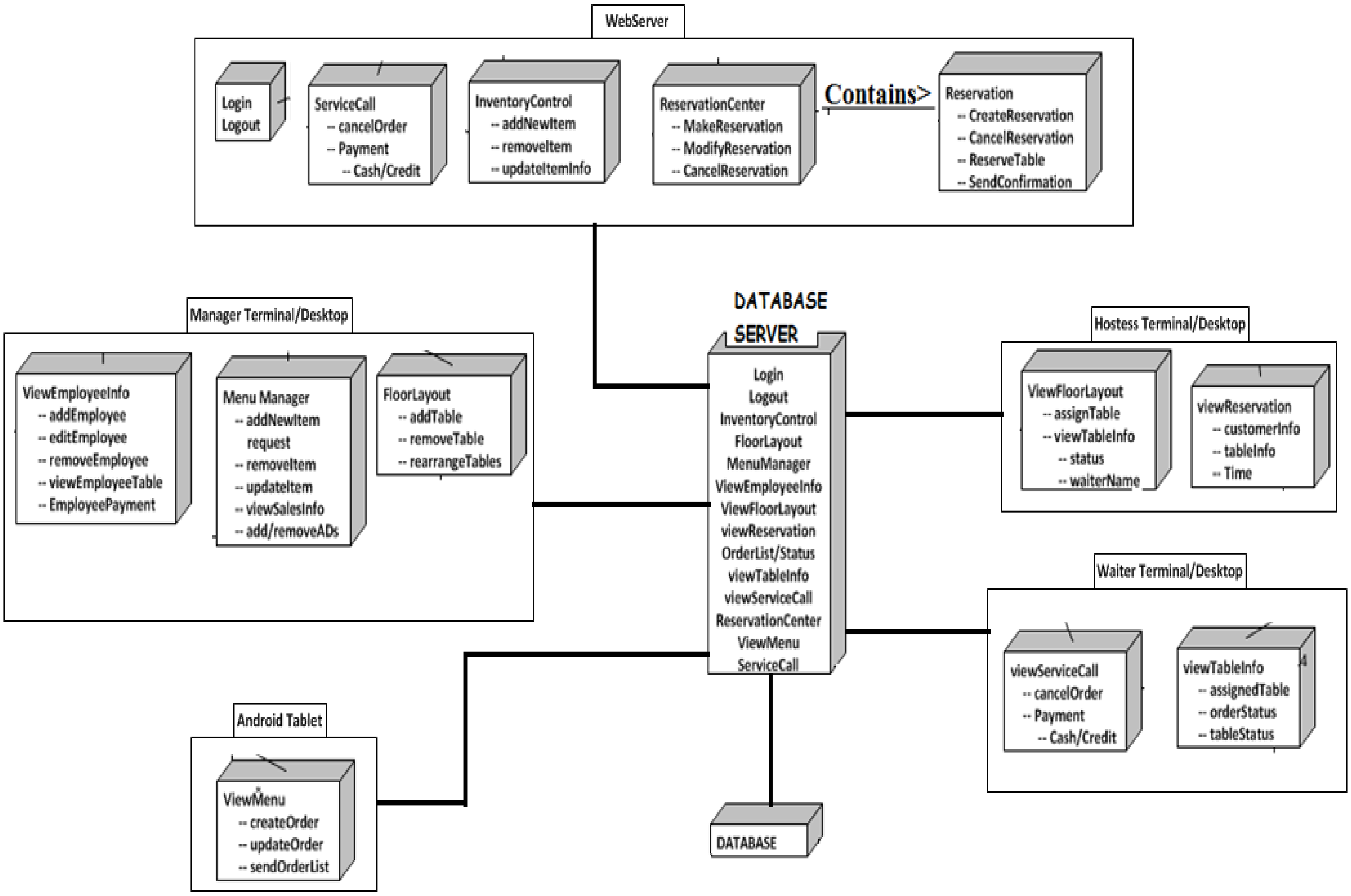


Figure 6: Subsystems Mapped to Hardware

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*

EmployeeData
firstName
lastName
PIN
Birthdate
Age
hireDate
lastDayWorked
streetAddress
City
zipCode
State
homePhone
cellPhone

MenuItem
itemID
itemName
Price
Type
Description

Table
Table#
tableStatus
assignedWaiter

Order
Order#
orderStatus
Table#
orderDate

Receipt
itemID
itemName
Price
Quantity
totalCost

(for reservation system)

CustomerID
firstName
lastName
phone#
emailAddress
sideNote

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 ^{TM} lightly breaded and cooked until crispy.
202004	Jerk Chicken Sandwich	$9.56	Sandwiches	Blackened grilled chicken breast topped with our signature Carribean Jerk^{TM} 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

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.

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.

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.

Algorithms & Data Structures

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.

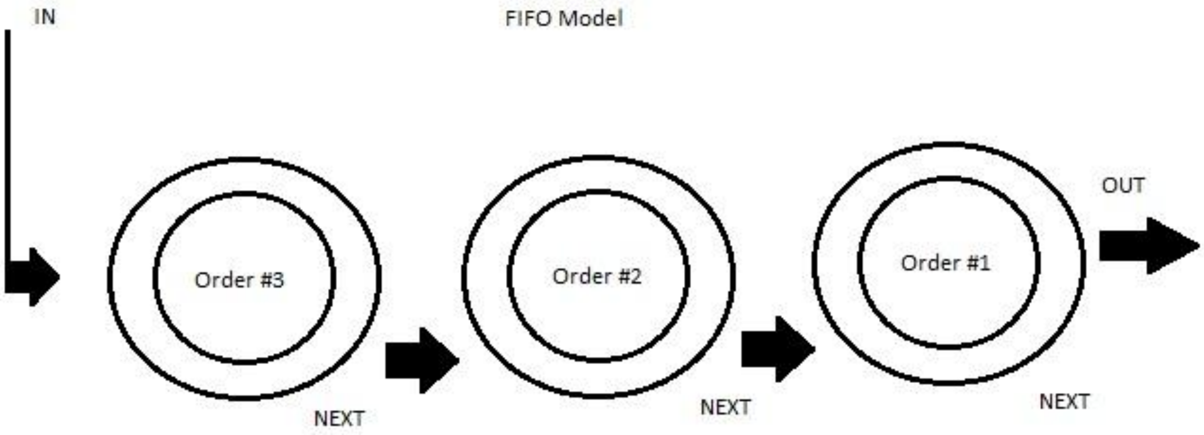
Our system contains many algorithms from easy to complex. Algorithms are the most important thing in the system because it does the task that the all the employees need. Without algorithms simple things will become harder and more time consuming which will cost the business more money and time.

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.

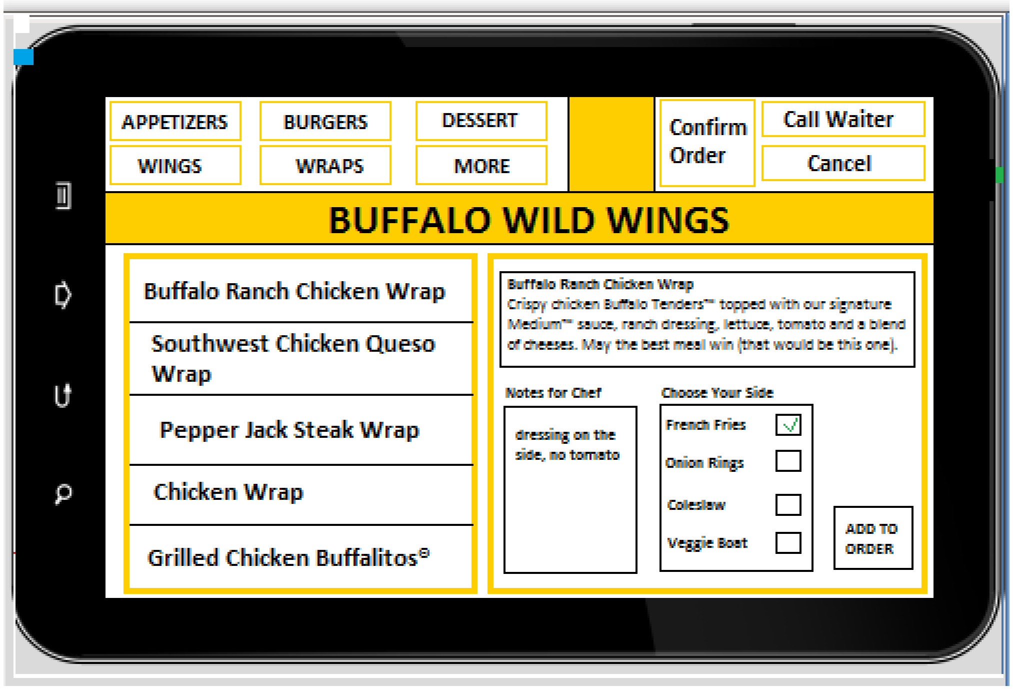
Our system will contain the FIFO (first in first out) method. This applies that the value of the highest priority in the queue. An example, when deciding which order to place first in the queue when sent to the chef, the time at which the order was sent is the deciding factor for FIFO. Queues are necessary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



User Interface Design & Implementation

Android Interface

One aspect of the user interface that may cause confusion when ordering from the menu is how to go from selecting items off the menu to actually sending the order to the chef. In the original design, to go from the menu interface to the order confirmation interface, the customer must select the "View Order" button from the top right corner. The tablet will now have a "Confirm Order" button instead of the "View Order" button. The new interface has also increased the size of the "Confirm Order" button to emphasize its importance.



Buffalo Ranch Chicken Wrap
Crispy chicken Buffalo Tenders™ topped with our signature Medium™ sauce, ranch dressing, lettuce, tomato and a blend of cheeses. May the best meal win (that would be this one).

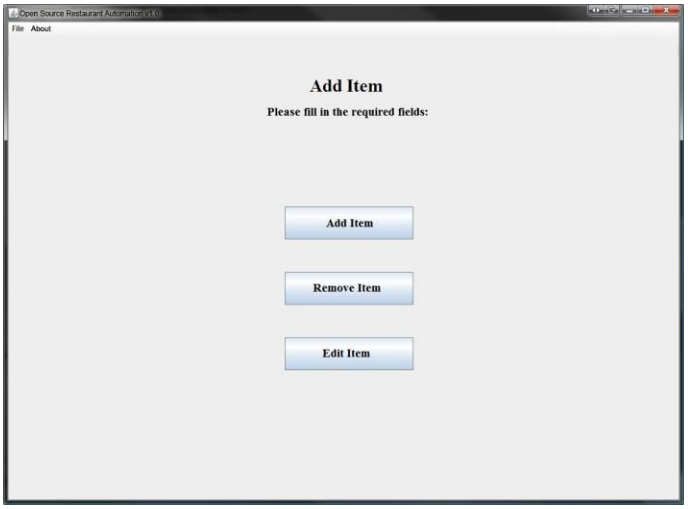
Notes for Chef
dressing on the side, no tomato

Choose Your Side
French Fries ☑
Onion Rings ☐
Coleslaw ☐
Veggie Boat ☐

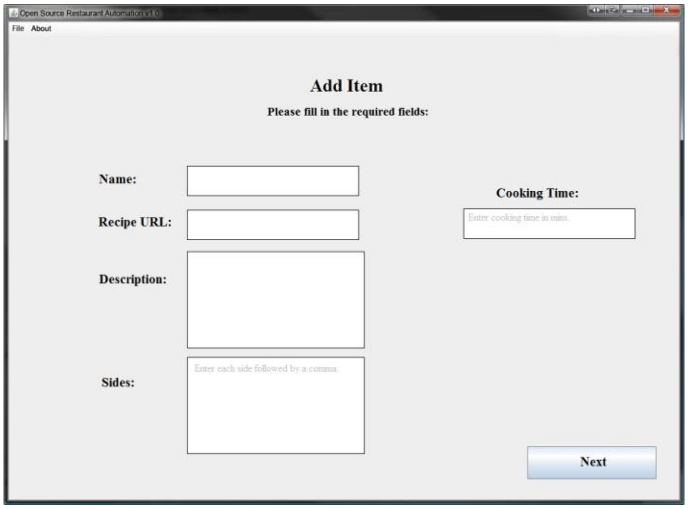
New Android Interface

Windows Interface Specification

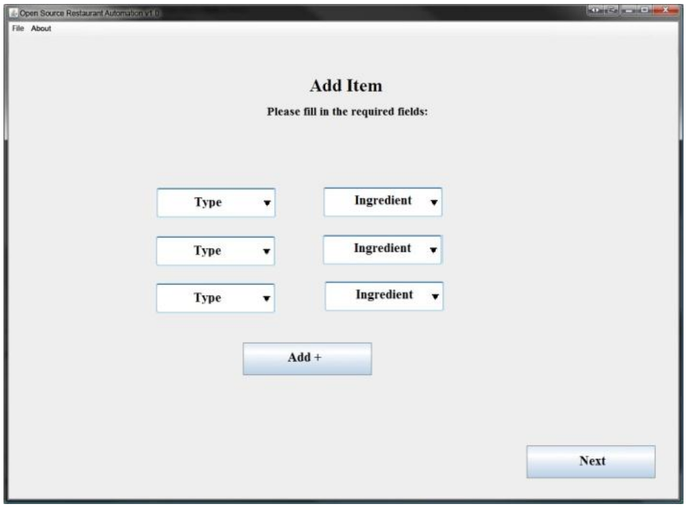
The following interface specification is that of Use Case 8: Add Item. Most of our other windows interfaces that we specified in report 1 stay valid. This use case has changed drastically therefore we have decided to demonstrate this one. The new implementation has improved the usability by reducing the number of clicks required to perform the task.


Open Source Restaurant Automation v1.0
File
About
Add Item
Please fill in the required fields:
Add Item
Remove Item
Edit Item

This above is the first screen for performing the task. The user starts by clicking the add item button. This screen requires 1 click.


Open Source Restaurant Automation v1.0
File
About
Add Item
Please fill in the required fields:
Name:
Recipe URL:
Description:
Sides:
Enter each side followed by a comma.
Cooking Time:
Enter cooking time in mins.
Next

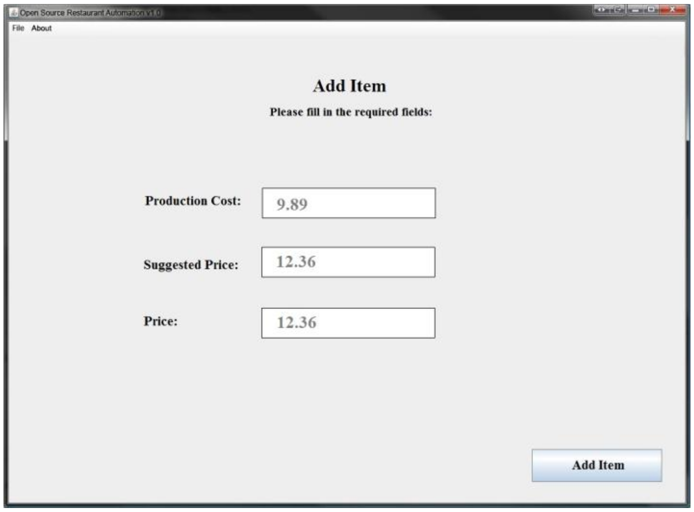
The above screen is the first form the manager has to fill. Four text fields need to be filled and a single click. The user effort on this screen is highly dependent on the amount of text that needs to be entered.



Add Item

Please fill in the required fields:

The above screen is the second form which asks the manager to enter the required ingredients. By default it has upto 5 ingredients ready to be entered. If the manager wishes he/she may add more ingredients by clicking the “Add” button. We have reduced user effort by eliminating text boxes for ingredients.



Add Item

Please fill in the required fields:

Production Cost: 9.89

Suggested Price: 12.36

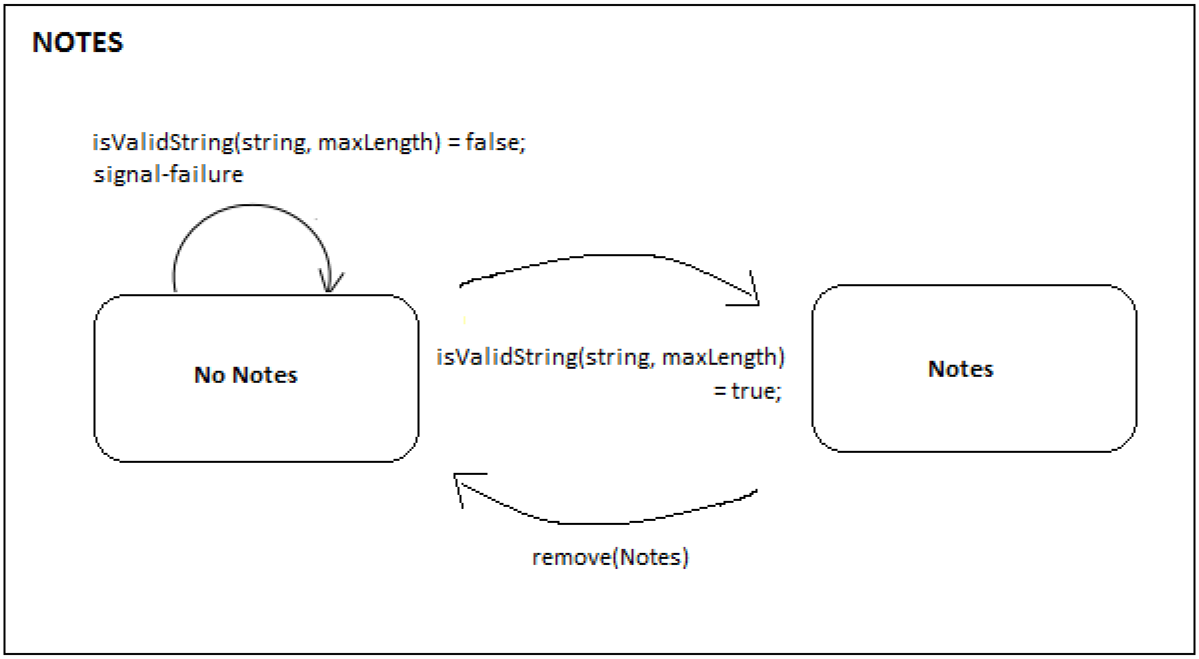
Price: 12.36

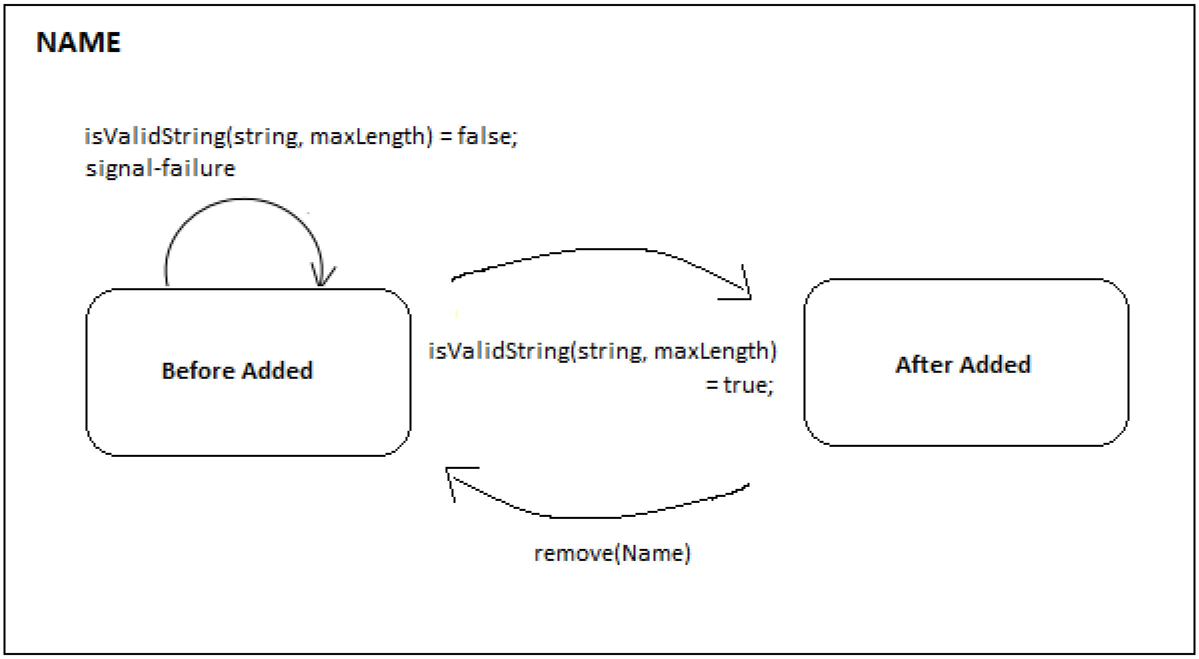
The above screen is the final screen required from the manager to add an item. The first box shows production cost which is calculated based on the ingredients the manager has entered. The second box shows the price based on a 25% profit. The third box is editable by the manager as to what he/she wants the price to be. By default the price box shows the suggested price to reduce user effort but it can be edited. Finally the manager has to click “Add Item” to complete the task.

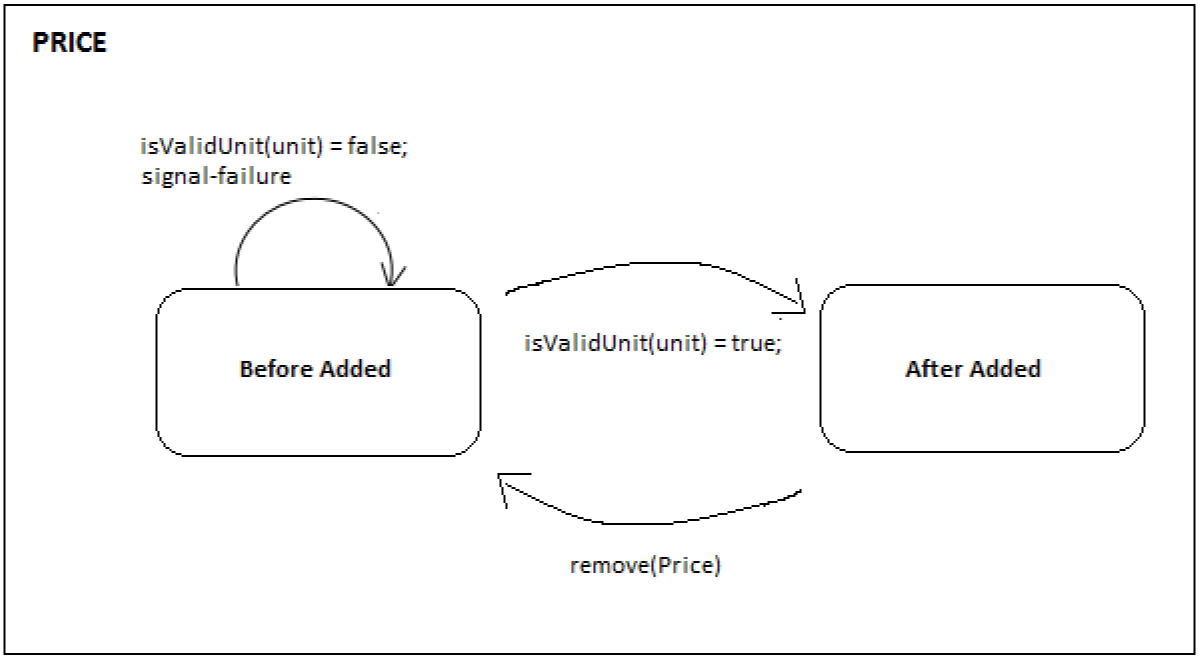
Design of Tests

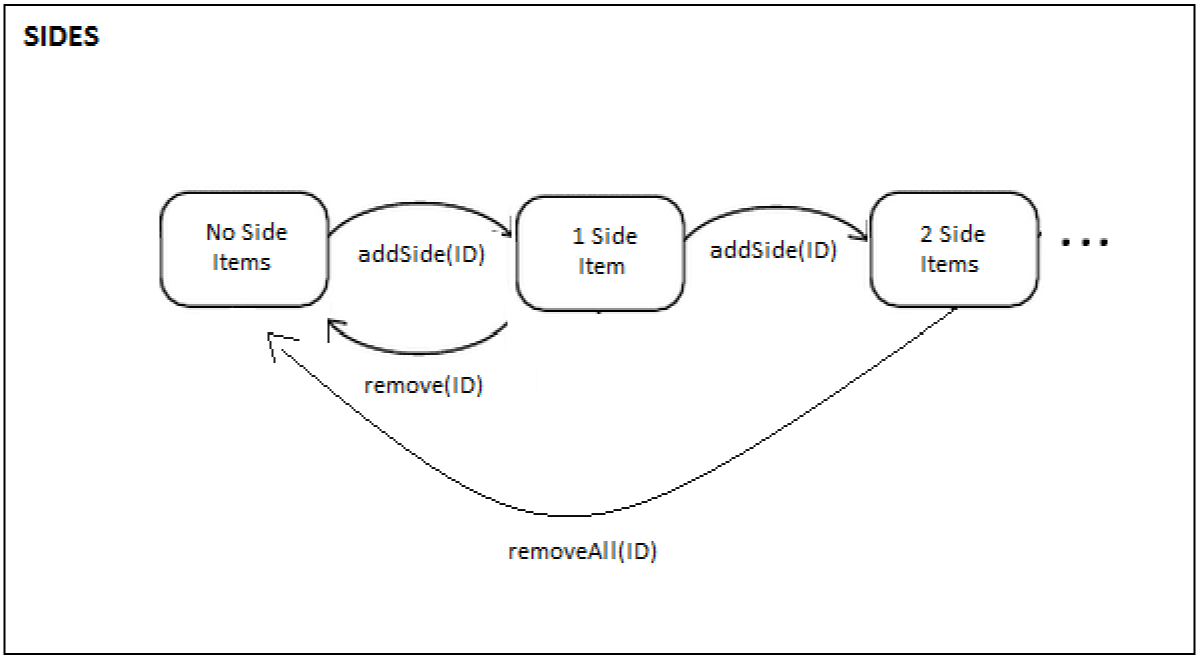
State Diagrams

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.


NOTES
isValidString(string, maxLength) = false;
signal-failure
No Notes
isValidString(string, maxLength) = true;
Notes
remove(Notes)


NAME
isValidString(string, maxLength) = false;
signal-failure
Before Added
isValidString(string, maxLength) = true;
After Added
remove(Name)


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 **Use Case Tested:** Login (UC-1), main success scenario **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. **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

Test Coverage

The unit tests develop will implement a State-Based Test Coverage Plan. The unit tests were designed so that dynamic attributes of every class will enter every state possible. Because many of the attributes in these use cases are TextFields that can be edited by the Manager, most attributes only have 2 states (Empty-String/Valid-String). Other attributes can also have an infinite number of states such as the order attribute (OrderItem[ ]) in the Order Class. Because each state is defined as the number of items in the array the number of states varies with each customer. The unit test will show that the number of items in the order is truly dynamic and that the order size can vary to various amounts.

Project Management

Progress Report

Our group has already begun setting up the database. As of now, we are dealing with a VB.net platform and MySQL and hopefully progress from there. It may occur to us to later on to code either in PHP or Java, whichever will make the task more user friendly for us. For the time being, we want to experiment with different languages and get a feel of which environment we are more comfortable in.

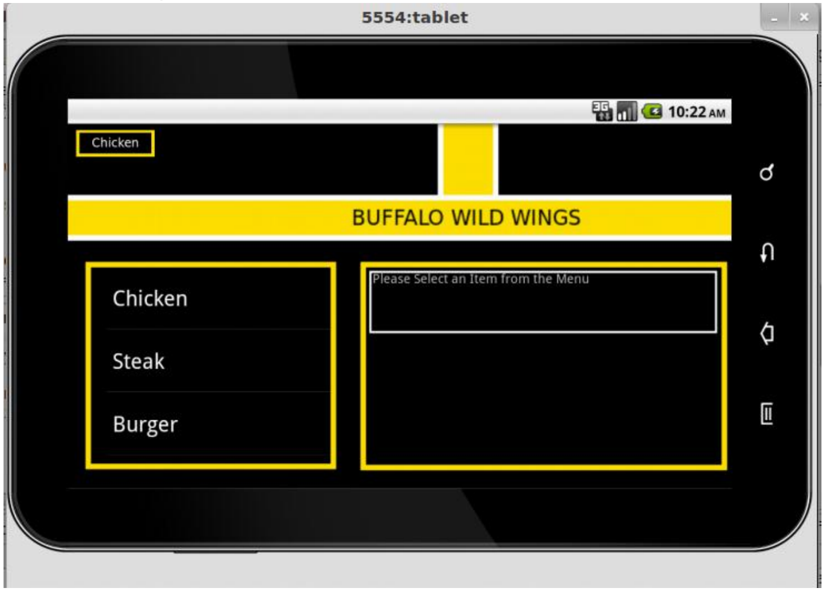
A simple table has been created to hold Employee Data along with each employee's username, PIN number and other personal information. Next, a Login interface was created to allow an employee to log in successfully. A sample of employees' information was recorded in our database and from there we used the username and PIN to successfully log in from the Login interface. All that was required was now to connect to server and database. After we were able to figure out how to do that, our database was tested and worked efficiently.

Below is a portion of the written function code:

```
Friend Function getLoginData() As DataTable

        Dim strUsername = txtUsername.Text
        Dim strPassword = txtPassword.Text
        Dim myConnection As SqlConnection = New SqlConnection("Data Source=PISALS-PC\PRADNYASQLDB;Initial Catalog=User_Login;Integrated Security=True")

        Dim myCommand As SqlCommand = New SqlCommand("", myConnection)
        Dim sqlParam As System.Data.SqlClient.SqlParameter
        Dim ds As New DataSet
        Dim da As New SqlDataAdapter(myCommand)
        myConnection.Open()
        myCommand.Parameters.Clear()
        myCommand.CommandType = CommandType.Text

        myCommand.CommandText = "select * from User_users where Username = @Username and password = @Password"

        da.SelectCommand.Parameters.Add(New SqlParameter("@Username", SqlDbType.VarChar))
        da.SelectCommand.Parameters("@Username").Value = strUsername

        da.SelectCommand.Parameters.Add(New SqlParameter("@Password", SqlDbType.VarChar))
        da.SelectCommand.Parameters("@Password").Value = strPassword
        Dim dt As New DataTable
        da.Fill(dt)

        myCommand.CommandTimeout = 120
        myConnection.Close()
        myConnection = Nothing
        myCommand = Nothing

        Return dt

End Function
```

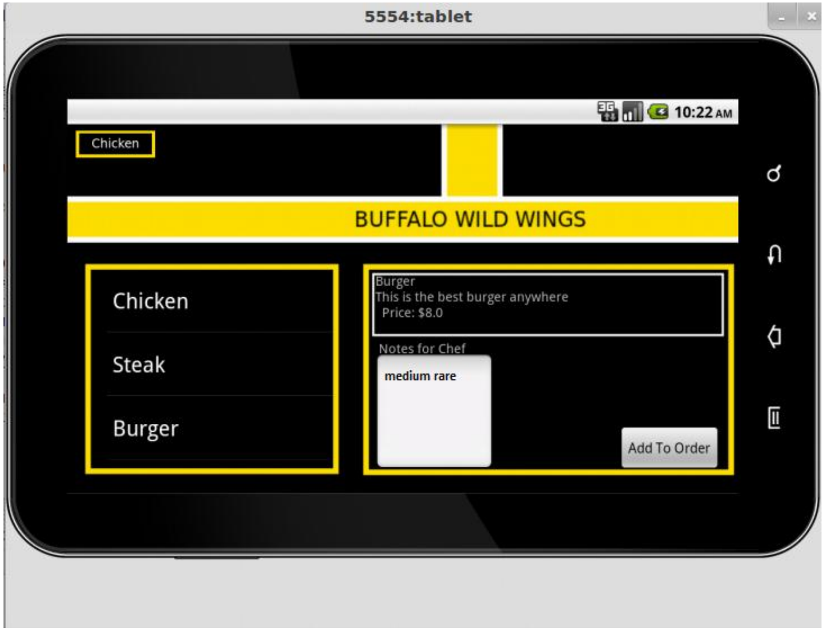
Coding for the Android Menu Application has already begun. The layout scheme for the app is running in the android emulator. Because the database for the menu has not yet completed, there are three dummy menu items currently that can be selected. Looking at Android_Image1 below shows the three menu items and the tablet prompting the user to select an item before continuing. Once the user has selected an item, the tablet displays the name, description, and price appropriately (Android_Image2). Beneath the description is a Text Field where the user can add comments for the chef when preparing the food. There is also an Add button visible at this point, but it does not have any implementation.

Android Image 1



Please Select an Item from the Menu

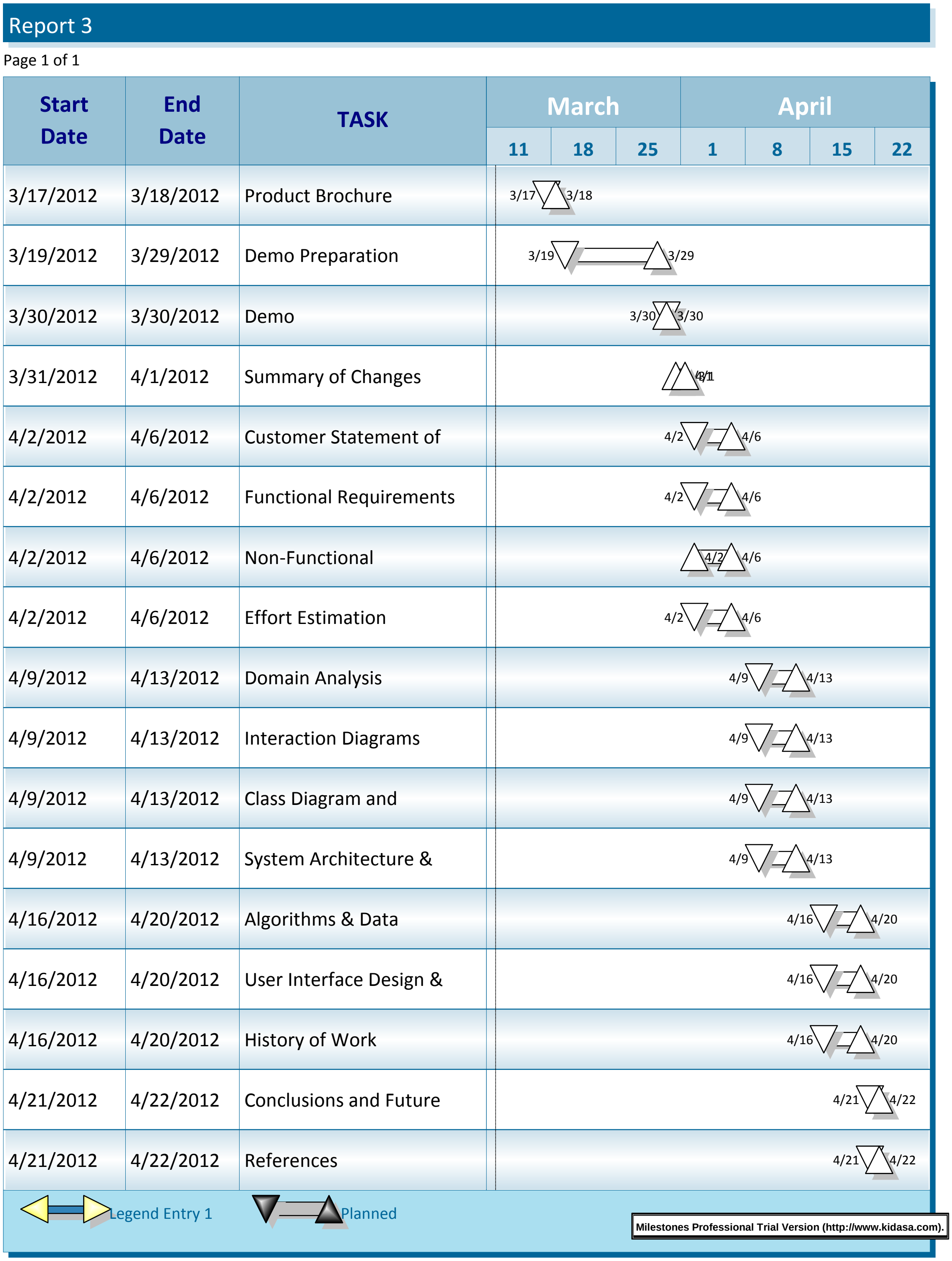
Android Image 2



Burger
This is the best burger anywhere
Price: $8.0

Notes for Chef
medium rare

Plan of Work



Report 3

Page 1 of 1

Start Date	End Date	TASK
3/17/2012	3/18/2012	Product Brochure
3/19/2012	3/29/2012	Demo Preparation
3/30/2012	3/30/2012	Demo
3/31/2012	4/1/2012	Summary of Changes
4/2/2012	4/6/2012	Customer Statement of
4/2/2012	4/6/2012	Functional Requirements
4/2/2012	4/6/2012	Non-Functional
4/2/2012	4/6/2012	Effort Estimation
4/9/2012	4/13/2012	Domain Analysis
4/9/2012	4/13/2012	Interaction Diagrams
4/9/2012	4/13/2012	Class Diagram and
4/9/2012	4/13/2012	System Architecture &
4/16/2012	4/20/2012	Algorithms & Data
4/16/2012	4/20/2012	User Interface Design &
4/16/2012	4/20/2012	History of Work
4/21/2012	4/22/2012	Conclusions and Future
4/21/2012	4/22/2012	References

Breakdown of Responsibilities

Here is a temporary breakdown of responsibilities for report 3. Note that this breakdown is temporary and subject to change.

Task/Group Member	Praveen	Pradnya	Bill	Kartik	Zac
Summary of Changes (5)	3		2		
Customer Statement of Requirements (6)		5	1		
Glossary of Terms (4)			4		
Functional Requirements (37)	7	7	7	8	8
Non Functional Requirements (6)	6				
Domain Analysis (25)			9	12	4
Interaction Diagrams (30+10)		10	10	20	
Class Diagram & Interface Specification (10+10)	5				15
System Architecture & System Design (22)	8	14			
Algorithms & Data Structures (4)			4		
User Interface Design & Implementation (8)					8
History of Work (5)					5
Conclusions & Future Work (5)	5				
References (3)			3		
Project Management (10)	6	4			

References

Useful Information From:

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

Pictures From:

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

Links that help with the report:

6. http://en.wikipedia.org/wiki/Intranet
7. http://en.wikipedia.org/wiki/Data_structure