Group 2

Restaurant Automation

Report 2

March 5, 2017

Project Team

Mit Patel

Raj Patel

Nill Patel

Dylan Herman

Prabhjot Singh

Moulindra Muchumari

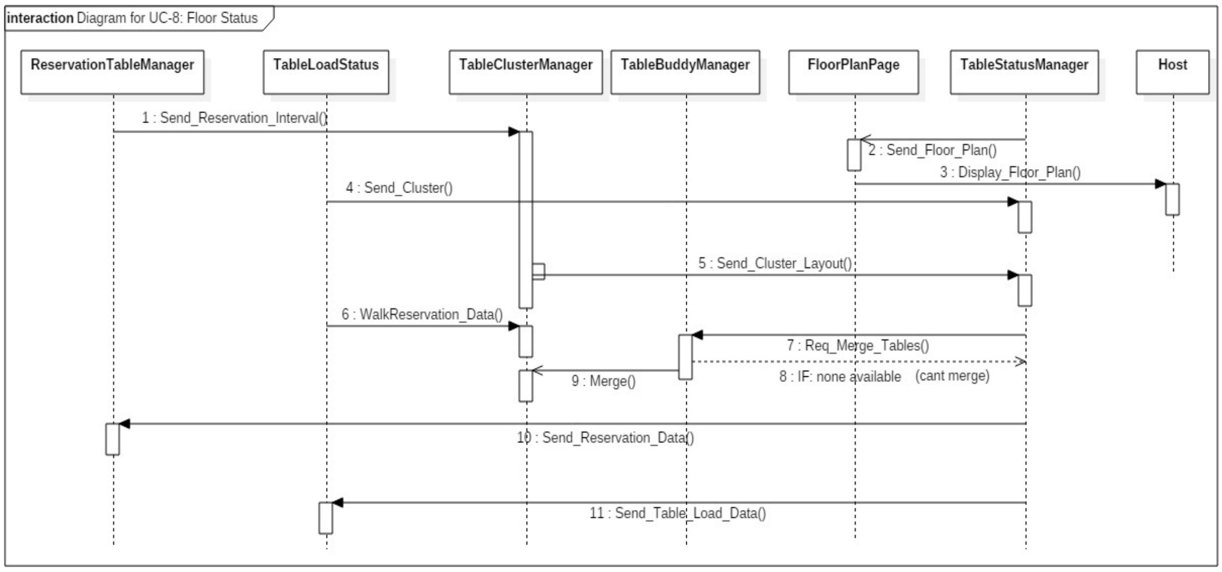
All team members contributed equally!

Table of Contents

Interaction Diagrams

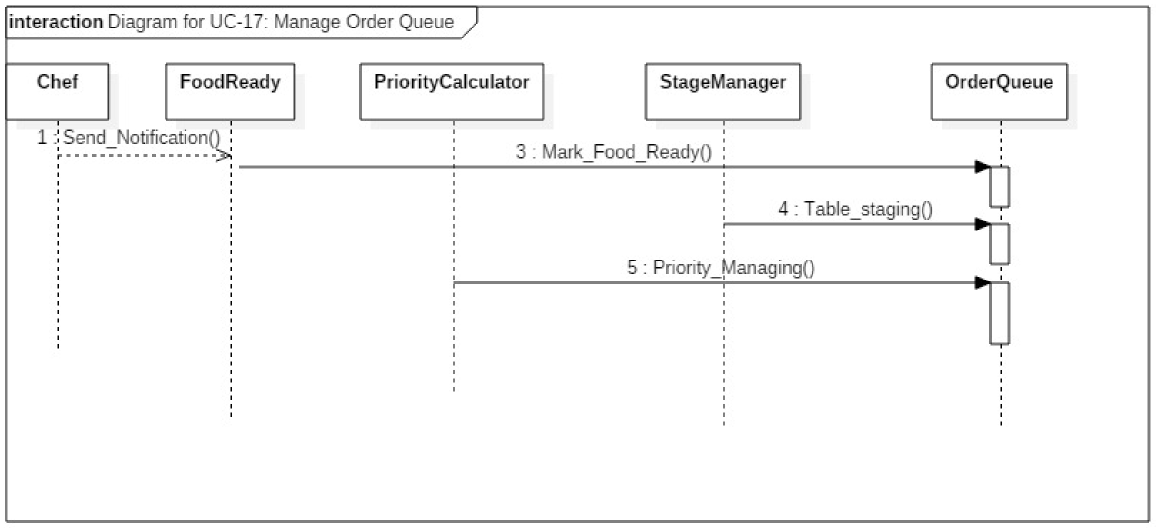
Based on *Learning UML 2.0*, a sequence diagram should be used to represent a flow of messages, while a communication diagram is better for focusing on the connections between different participants within an interaction. The fully-dressed use cases we chose lean towards a flow of messages rather than a communications link because our use cases are more focused on how messages are passed between different modules.

UC-8 Floor Status


interaction Diagram for UC-8: Floor Status
ReservationTableManager
TableLoadStatus
TableClusterManager
TableBuddyManager
FloorPlanPage
TableStatusManager
Host
1 : Send_Reservation_Interval()
2 : Send_Floor_Plan()
3 : Display_Floor_Plan()
4 : Send_Cluster()
5 : Send_Cluster_Layout()
6 : WalkReservation_Data()
7 : Req_Merge_Tables()
8 : IF: none available (cant merge)
9 : Merge()
10 : Send_Reservation_Data()
11 : Send_Table_Load_Data()

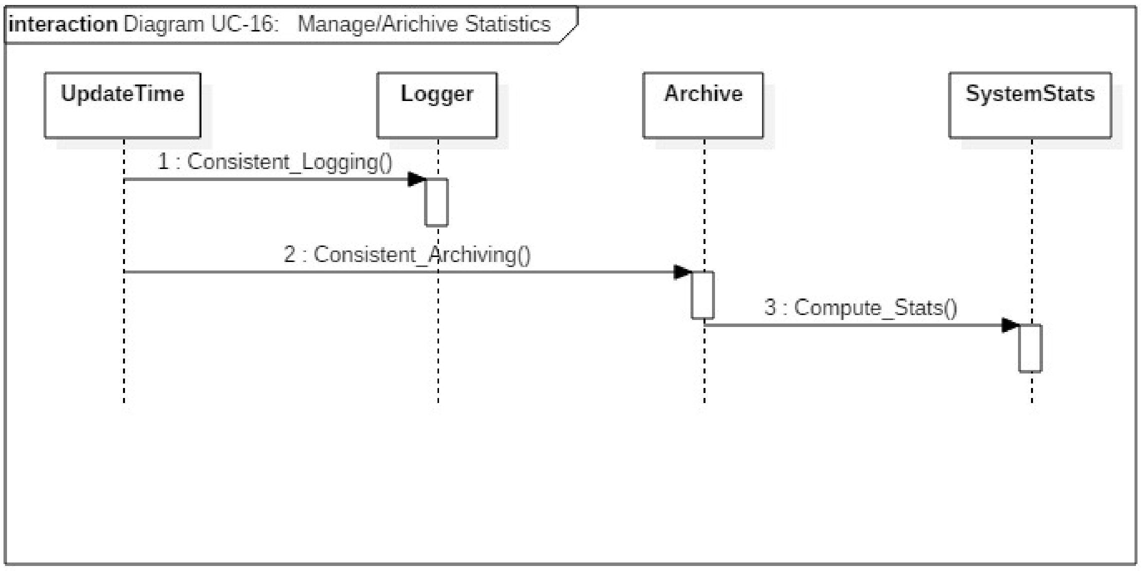
The Floor Status Use Case describes the interactions that take place for the host to be able to view an accurate display of the dining area floor plan. There are two types of tables: Reservation and Walk-in tables. Reservation tables can only be reserved within periodic intervals; this ensures there is enough time for each person to eat. However, a person can take a Reservation table through walkin if no one has reserved it at the start of the current interval; once the interval ends, the table is up for reservation again. Walkin tables are exclusive to people "walking in" only. Data is collected from past days and throughout the current day via the ReservationTableManager to decide the optimum interval for reservations, and data is collected via the TableLoadStatus to decide the optimum number of Reservation to Walk-in tables per Cluster. A Cluster is a grouping of tables of a given size. The status of each table: whether it is reserved, dirty (needs to be cleaned), taken, or free is tracked in real time. In addition, if there aren't enough tables of a certain size and type to satisfy a request, tables of a smaller size can be "buddied" up to form larger tables. All of this information is then forwarded to the floor plan display so the host can see the status of all tables in real time.

UC-17 Manage Order Queue



The Manage Order Queue use case describes the interactions that take place to order items within the queue. Items in the queue are prioritized based on the stage the items are assigned (i.e. appetizers, entrees, and desserts). In addition, a second priority calculation is computed based on the average service time it takes to process a table's order. Then, the chef marks each food item as completed.

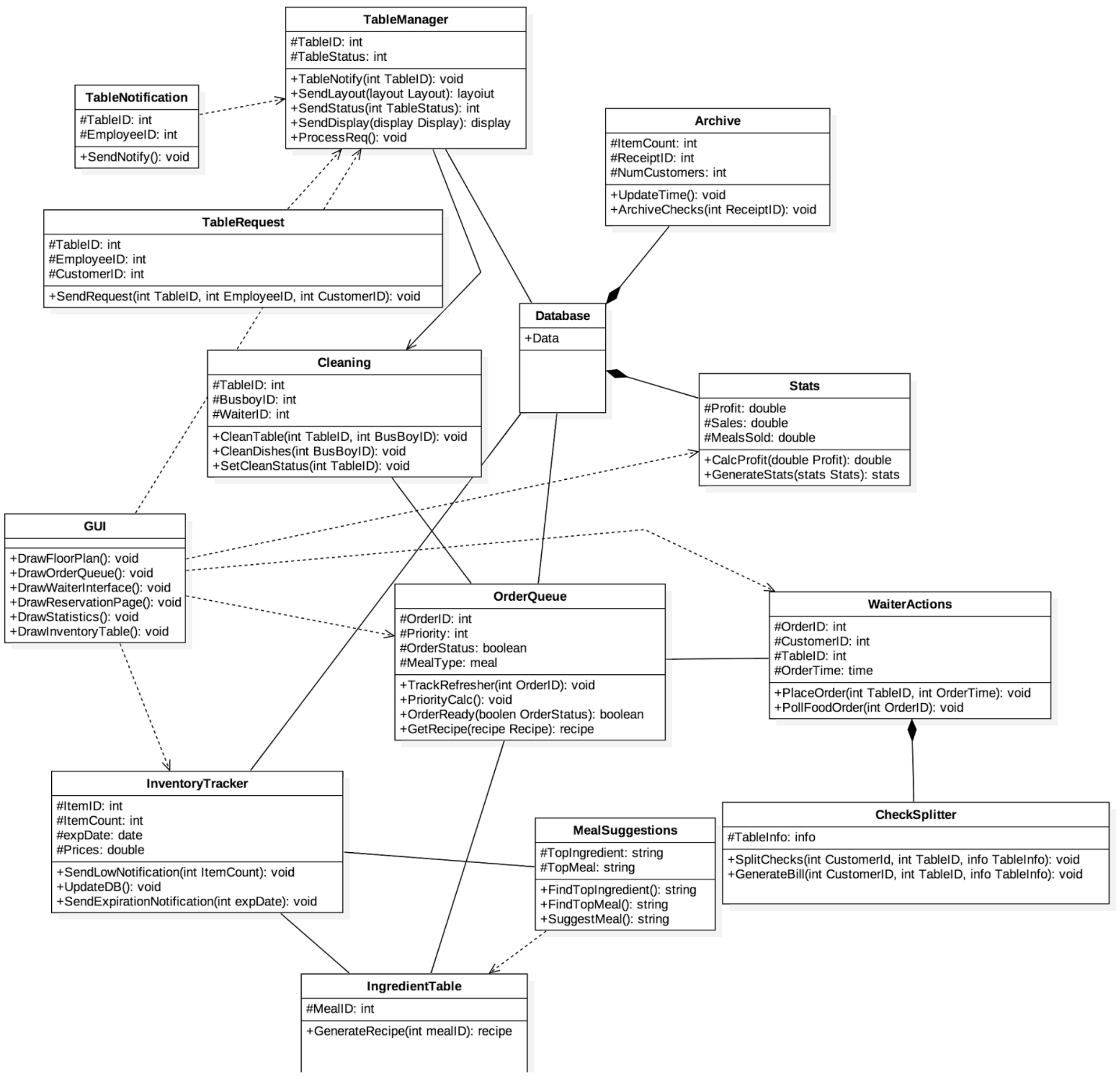
UC-16 Manage/Archive Statistics


interaction Diagram UC-16: Manage/Arichive Statistics
UpdateTime
Logger
Archive
SystemStats
1 : Consistent_Logging()
2 : Consistent_Archiving()
3 : Compute_Stats()

The Manage/Archive Statistics use case describes the interaction that takes place whenever there is a new transaction. The system will get automatically updated once these transactions occur, As these values are updated, the system will periodically compute several statistics such as total profit, number of items sold, most popular dish, etc. All this information is available upon request through the system.

Class Diagram and Interface Specification

Class Diagram


TableManager
#TableID: int
#TableStatus: int
+TableNotify(int TableID): void
+SendLayout(layout Layout): layout
+SendStatus(int TableStatus): int
+SendDisplay(display Display): display
+ProcessReq(): void
TableNotification
#TableID: int
#EmployeeID: int
+SendNotify(): void
Archive
#ItemCount: int
#ReceiptID: int
#NumCustomers: int
+UpdateTime(): void
+ArchiveChecks(int ReceiptID): void
TableRequest
#TableID: int
#EmployeeID: int
#CustomerID: int
+SendRequest(int TableID, int EmployeeID, int CustomerID): void
Database
+Data
Cleaning
#TableID: int
#BusboyID: int
#WaiterID: int
+CleanTable(int TableID, int BusBoyID): void
+CleanDishes(int BusBoyID): void
+SetCleanStatus(int TableID): void
Stats
#Profit: double
#Sales: double
#MealsSold: double
+CalcProfit(double Profit): double
+GenerateStats(stats Stats): stats
GUI
+DrawFloorPlan(): void
+DrawOrderQueue(): void
+DrawWaiterInterface(): void
+DrawReservationPage(): void
+DrawStatistics(): void
+DrawInventoryTable(): void
OrderQueue
#OrderID: int
#Priority: int
#OrderStatus: boolean
#MealType: meal
+TrackRefresher(int OrderID): void
+PriorityCalc(): void
+OrderReady(boolen OrderStatus): boolean
+GetRecipe(recipe Recipe): recipe
WaiterActions
#OrderID: int
#CustomerID: int
#TableID: int
#OrderTime: time
+PlaceOrder(int TableID, int OrderTime): void
+PollFoodOrder(int OrderID): void
InventoryTracker
#ItemID: int
#ItemCount: int
#expDate: date
#Prices: double
+SendLowNotification(int ItemCount): void
+UpdateDB(): void
+SendExpirationNotification(int expDate): void
MealSuggestions
#TopIngredient: string
#TopMeal: string
+FindTopIngredient(): string
+FindTopMeal(): string
+SuggestMeal(): string
CheckSplitter
#TableInfo: info
+SplitChecks(int CustomerId, int TableID, info TableInfo): void
+GenerateBill(int CustomerID, int TableID, info TableInfo): void
IngredientTable
#MealID: int
+GenerateRecipe(int mealID): recipe

Class Diagram Description

The class diagram above includes the following classes:

TableManager is a class that contains TableID, TableStatus and uses those to notify customers of their table opening, sending the layout of the restaurant to the interface, it also processes reservation requests.
TableNotification is a class that sends the notifications to clean table, serve table, etc. to the respective employees.
TableRequest is a class that sends a request to the respective employees to complete a task for the respective tables.
Cleaning is a class that pertains to the busboy that will notify him/her when to do a certain task.
OrderQueue is a class that manages the orders and tracks refreshers along with retrieving recipe’s from the ingredient table.
WaiterActions is a class that pertains to the waiter that is in charge of customer orders.
CheckSplitter is a class that automatically
IngredientTable is a class that generates the recipe for the order queue when it's requested by the order queue.
InventoryTracker is a class that keeps track of the inventory and sends notifications when items are below a certain number or if they are about to expire.
MealSuggestions is a class that will find both top ingredients and top meals to suggest meals popular and profitable meals to the manager.
Stats is a class that will calculate statistics such as profit and meals sold using information from the database
Archive is a class that consistently logs receipts along with the system time.
Database is a class that holds all the data in the system and interacts with the rest of the classes.
GUI is a class that depends on multiple classes to draw the user interface of the system.

Data Types and Operation Signatures

TableManager

Attribute	**Description**
#TableID: int	Holds the ID of the respective table
#TableStatus: int	Indicates the status of a table using numbers such as free table is 1, busy is 0

Operations	**Description**
+TableNotify(int TableID):void	Sends a notification when a table is ready for customers
+SendLayout(layout Layout): layout	Sends the table layout and any clustered groups of tables that shows the status of each table upon request
+SendStatus(int TableStatus): int	Sends the status of the table so it can be updated on the interface
+SendDisplay(display Display): display	Updates the display based on any updates to the system
+ProcessReq(): void	Process incoming reservation requests along with walk in requests

TableNotification

Attributes	**Descriptions**
#TableID: int	Holds the ID of the respective table
#EmployeeID: int	Holds the ID of the respective employee

Operations	**Descriptions**
+SendNotify(): void	Sends notifications to the respective employees.

Cleaning

Attributes	**Descriptions**
#TableID : int	Holds the ID of the respective table
#BusboyID: int	Holds the Busboy's employee ID
#WaiterID: int	Holds the Waiter’s employee ID

Operations	**Descriptions**
+CleanTable(int TableID,int BusboyID): void	Sends Busboy a notification to clean the table using the TableID
+CleanDishes(int BusboyID): void	Sends Busboy a notification to clean the dishes in the kitchen
+SetCleanStatus(int TableID): void	Busboy is able to set the status of the table to clean

TableRequest

Attributes	**Descriptions**
#TableID : int	Holds the ID of the respective table
#EmployeeID: int	Holds the ID of the respective employee
#CustomerID: int	Holds each customer's ID

Operations	**Descriptions**
+SendRequest(int TableID,int EmployeeID,int CustomerID): void	Sends a request to the system to fulfill a reservation or walk in request

InventoryTracker

Attributes	**Descriptions**
#ItemID: int	The ID for an item in the inventory
#ItemCount: int	The number of items corresponding to each ItemID
#expDate: date	The expiration date for the respective ItemID
#Prices: double	The price of each item for the respective ItemID

Operations	**Descriptions**
+SendLowNotification(int ItemCount): void	Sends notification to the manager if the item count falls below a predefined threshold
+UpdateDB(): void	Updates database with new item count every time anything from the inventory is used
+SendExpirationNotification(int expDate): void	Sends notificiation to the manager if the item gets closer to the expiration date

Database

Attributes	**Descriptions**
+data	Holds all the data used for the system.

OrderQueue

Attributes	**Descriptions**
#OrderID: int	Holds the ID of the order that has been placed
#Priority: int	Holds the priority of the order
#OrderStatus: boolean	Holds the status of the order, ready or not ready
#MealType: meal	Holds the type of meal each table is on, such as appetizers, entrees etc.

Operations	**Descriptions**
+TrackRefresher(int OrderID): void	Tracks the refreshers for the given OrderID
+PriorityCalc(): void	Calculates the priority of the order based on factors such as time, size etc.
+OrderReady(boolean OrderStatus): boolean	Marks order as ready when it is done
+GetRecipe(recipe Recipe): recipe	Contacts the recipe book and retrieves the recipe for the requested meal(s)

IngredientTable

Attributes	Descriptions
#MealID: int	Holds the ID of the meal

Operations	Descriptions
+GenerateRecipe(int mealID): recipe	Generates the recipe for the requested mealID

Archive

Attributes	Descriptions
#ItemCount:int	The number of items corresponding to each ItemID
#ReceiptID: int	Holds the ID for each receipt generated at the restaurant
#NumCustomers: int	The total number of customer in a requested time period

Operations	Descriptions
+UpdateTime(): void	Gets the system time to display when the application was last updated
+ArchiveChecks(int ReceiptID): void	Saves the checks for future archiving use along with using the data for statistics

Stats

Attributes	**Descriptions**
#Profit: double	The total profit made in a requested time period
#Sales: double	The total money from sales, not including costs, over a requested time period
#MealsSold: double	The total number of meals sold over a requested time period

Operations	**Descriptions**
+CalcProfit(double Profit): double	Calculates the profit by using the data from the system.
+GenerateStats(stats Stats): stats	Generate several statistics summarizing the restaurant's performance

WaiterActions

Attributes	**Descriptions**
#OrderID: int	Holds the ID of the order that has been placed
#CustomerID: int	Holds each customer's ID
#TableID: int	Holds the ID of the respective table
#OrderTime: time	The time each order was placed

Operations	**Descriptions**
+PlaceOrder(int TableID,int OrderTime): void	Sends the order to the order queue along with the TableID and the time of the order
+PollFoodOrder(int OrderID): void	Sends a request to order queue to send a notification back if the food is ready

CheckSplitter

Attributes	**Descriptions**
#TableInfo: info	Holds the information about the table such as the size of the table and the number of people paying

Operations	**Descriptions**
+SplitChecks(int CustomerId, int TableID,info TableInfo): void	Splits the checks evenly between the number of people paying or by each respective order, given the TableInfo
+GenerateBill(int CustomerID,int TableID,info TableInfo): void	Generated the bill(s) based on the TableInfo, CustomerID and the TableID

MealSuggestions

Attributes	**Descriptions**
#TopIngredient: string	Holds the ID of the most used ingredient in the restaurant
#TopMeal: string	Holds the ID of the most ordered meal in the restaurant

Operations	**Descriptions**
+FindTopIngredient(): string	Looks in the database to find the top ingredient
+FindTopMeal(): string	Looks in the database to find the top meal
+SuggestMeal(): string	Uses the top ingredient and top meal along with other statistics to suggest more profitable meals

GUI

Operations	**Descriptions**
+DrawFloorPlan(): void	Uses information from TableManager and uses it to draw the floor plan of the restaurant
+DrawOrderQueue(): void	Uses information from the OrderQueue and uses it to draw the interface of the orders for the chefs in the kitchen
+DrawWaiterInterface(): void	Uses information from WaiterActions and draws the interface on the waiter's tablet
+DrawReservationPage(): void	Draws the reservation page depending on the information in TableRequest
+DrawStatistics(): void	Draws the statistics page and displays the information from the Stats class
+DrawInventoryTable(): void	Draws the inventory table using information from the InventoryTracker

Traceability Matrix

	TableManager	TableNotification	Archive	TableRequest	Cleaning	Stats	GUI	OrderQueue	WaiterActions	InventoryTracker	MealSuggestions	IngredientTable	CheckSplitter
ReservationTableManager	X												
TableLoadStatus	X												
TableClusterManager	X												
TableBuddyManager	X												
TableStatusManager	X												
TableStatusNotifier		X											
FloorPlanPage							X						
ReservationPage							X						
CleanRequest				X	X				X				
WalkinRequest				X									
SetCleanedRequest				X	X								
CleanNotify		X			X								
ReadyNotify		X							X				
Logger			X										
SystemStats						X							
Archive			X										
UpdateTime			X			X							
CheckSplitter									X				X
PlaceOrder								X					
OrderQueue								X					
StageManager								X					
PriorityCalculator								X					
FoodChecker								X	X				
MealStageTracker							X	X	X				
FoodReady								X					
FoodPollRequest								X	X				
OrderReady								X	X				
InventoryContainer										X			
InventoryTracker										X			
IngredientTable												X	
IngredientPopularity													
MealPopularity											X		
MealSuggester											X		
MealRequest										X			

The traceability matrix describes how each the domain concepts map to the corresponding class. The explanations for the mappings are:

TableManager : The table manager is responsible for deciding what interval "Reservation" tables can be reserved in, from collected data. This is the job of the ReservationTableManager domain concept. The TableManager is also responsible for collecting data regarding the number of reservations and walkins and deciding the ratios of "Reservation" to "Walkin" tables per cluster. Therefore, the TableLoadStatus domain concept maps to the TableManager. The table Manager is also responsible for managing all the table “Clusters” per size; this is the domain concept TableClusterManager. It is also responsible for combining tables of different sizes to form larger tables when needed; this is the concept of TableBuddyManager. Lastily, it is responsible for managing all the statuses of the tables: reserved, dirty, available and in-use. This is concept of the TableStatusNotfier. We realized that all of the domain concepts that dealt with managing table weren’t distinct enough to be considered separate classes; they interconnected a lot and communicate with one-another. Thus we combined them into one class, TableManager.

TableNotification : TableNotification is responsible for managing all notifications for tables, whether a table needs to be cleaned or a waiter needs to go to that table. Thus, TableNotification takes on the concept TableNotifer. The two notifications it sends, as described above, are CleanNotify and ReadyNotify. None of these concepts are different enough that they justify splitting them into classes. The Notifications are simply messages.

Archive: We realized the Archive and Logger domain concepts were very similar, if not almost the same thing. Thus it is justified making them one class. The UpdateTime domain concept is utilized heavily by the Archive, and thus is mapped to it.

TableRequest: All of the table requests have been mapped to the TableRequest. We realized generalizing them made sense and the requests could simple be functions. They weren’t complex enough to justify them as being classes.

Cleaning: The cleaning related operations were distinct enough from everything else to justify there own class. Thus the clean related requests and notifications are mapped to another class too, Cleaning.

Stats: Statistics is a very important concept for the restaurant. The Stat class is mainly built of the SystemStatistics concept. In addition, the stats directly depend upon the time for timestamps; thus it uses UpdateTime.

GUI: The three main interfaces in the system are the FloorPlanPage, Reservation page for customers, and the WaiterActions interface for making table requests. Thus the system GUI is made up of these three.

OrderQueue: The order queue is made up lots of components that are relatively small and thus don't need to be their own classes. It manages the order in which orders are made, their priority. It also maintains state for when food and orders are ready to be picked up by the waiter. Thus it is directly connected to FoodPollRequest and OrderReady.

WaiterActions: The waiter's action are very related. The waiter asks for a table to be cleaned and is notified when one is ready and is being taken by customers. In addition, the waiter utilizes the food poll request system and its notifications to keep track of when his/her orders are ready.
InventoryTracker: The inventory tracker a concept with the same name as the class mapped to it, and also utilizes the InventoryContainer and takes in MealRequests

MealSuggestions: The meal suggestions system utilizes the different popularities to come up with suggestions and thus has those concepts mapped to it to form a class.

IngredientTable: The ingredient table mapped directly from its concept.

CheckSplitter: The check splitter mapped directly from its concept.

System Architecture and System Design

Architectural Styles

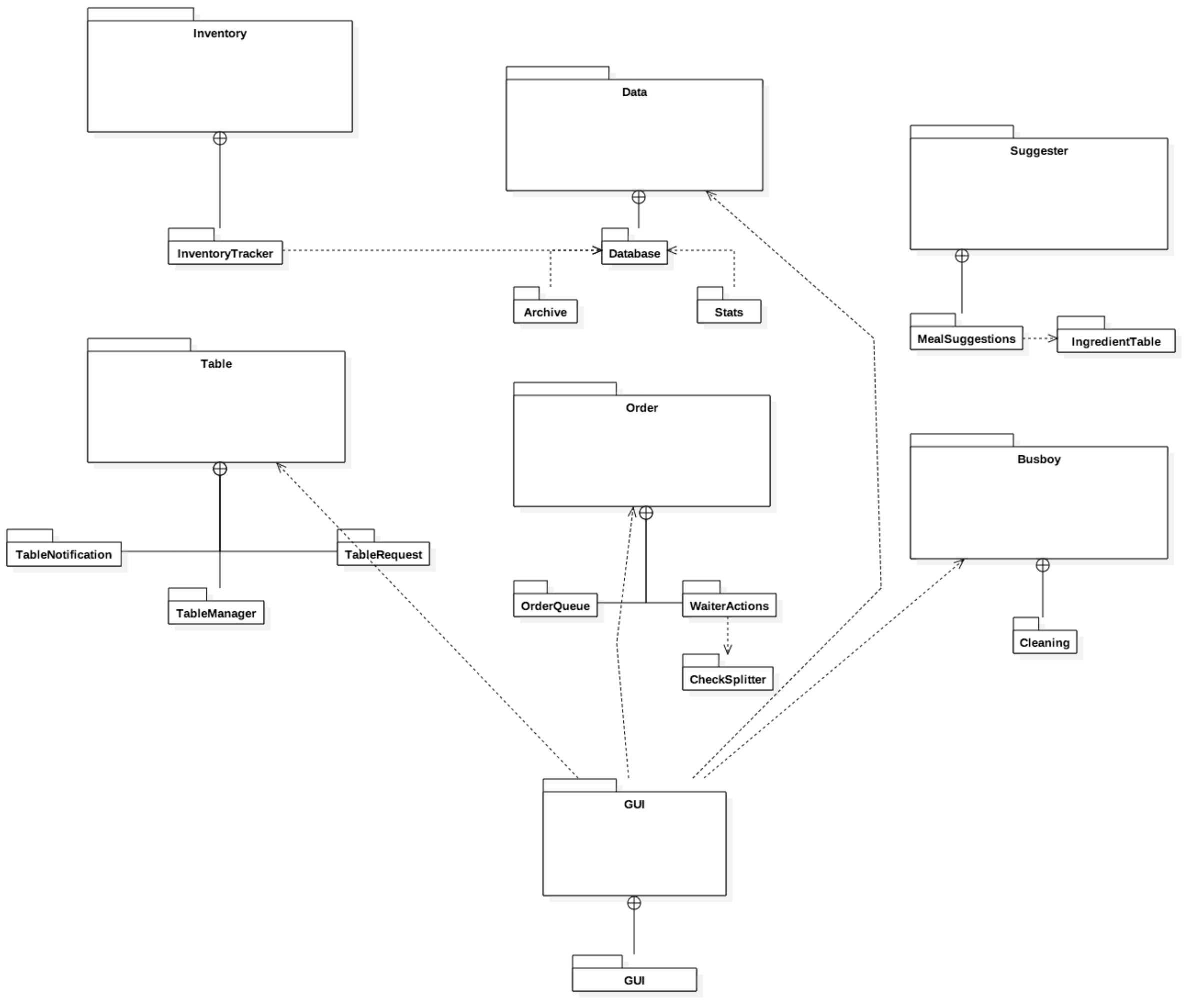
Our project follows a three-tier model which is composed of a presentation tier, a logic tier, and a data tier. Each tier is divided into smaller components based on the user interaction.

The presentation tier represents the various user interfaces depending on the user interacting with the system. The user interfaces are divided into three categories; customer, employee, and manager. The customer will be limited to viewing the reservation system, while the employee will only have access to the system specific to his or her role. In addition, the manager will have access to all features of the system, which can be configured by the application.

The logic tier deals with all of the computational aspects of our system. For instance, there will be various components that handle the floor plan to manage tables, inventory system, billing calculations and statistics, etc. The logic tier is also responsible for user authentication to allocate the appropriate permissions for a user. In an overall aspect, the logic tier handles all of the processing details of the entire system.

The data tier will represents the storage system of our application and will hold all pertinent data needed for different computational functions of the logic tier. Some types of data we will store and collect includes order history, ingredient usage, billing statistics, etc.

Identifying Subsystems



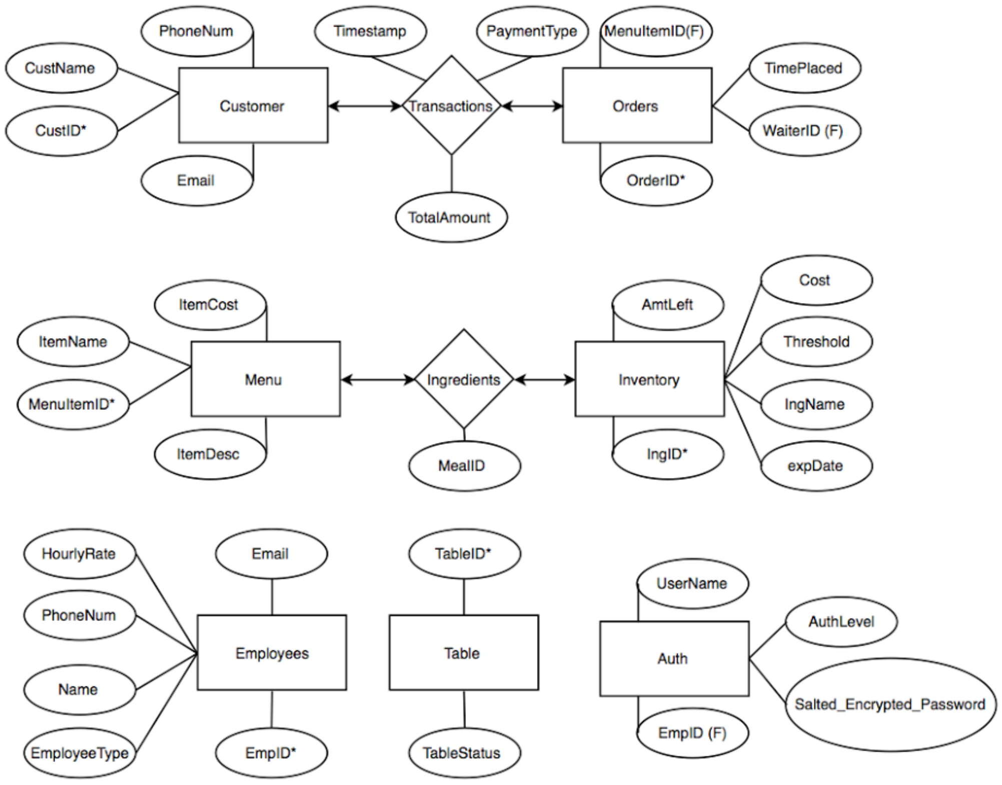
Mapping Subsystems to Hardware

Subsystems	Server	Tablet/Phone/PC
Inventory	X	
Data	X	
Suggester	X	
Table	X	X

Busboy	X	X
Order	X	X
GUI		X

Persistent Data Storage

Since our system is data driven, we are required to have persistent data storage. We intend to utilize a SQL central database for all subsystems. The following is the ER diagram.



Network Protocol

Our application uses the widely used HTTP protocol because the application is web-based. Our user interface is accessed via an HTTP request that will display the application. The logic functions a will also be accessed via HTTP requests made by the application. All remaining forms of communication are related to the database, which are hidden away by the framework we are using. Although the exact database communication protocol is unknown, it is known that sockets are used.

Global Control Flow

Execution order: Our server is an event-driven system because a customer may place reservation request at a random type or a waiter could place on order at any time. The server responds to each request by appropriately selecting the next action to process the event. Also, an employee could be processing multiple actions at once. For example, a waiter may check the order for table 1 and take the order (submit) for table 2. So, all of the users are not restricted to a linear sequence of events.

Time dependency: Our system is both real-time and event response type. For example, all orders tracked in real-time so that customers receive their food as quick as possible. Every event between taking order and delivering it is however event response type. Each event’s completion results in the start of the next event. For instance, when a customer is seated, a waiter is sent to take the order.

Concurrency: We will not directly need concurrency because each subsystem operates as a separate process. However, the underlying libraries could possibly use multiple threads (ex. server). This is not our concern because they are abstracted and a black box in our perspective.

Hardware Requirements

Tablet/Phone/Computer (Client-side)

- Modern web browser with javascript support (i.e. Google Chrome, Mozilla Firefox, Safari).
- Wireless Network Card
- High Resolution Display

Server

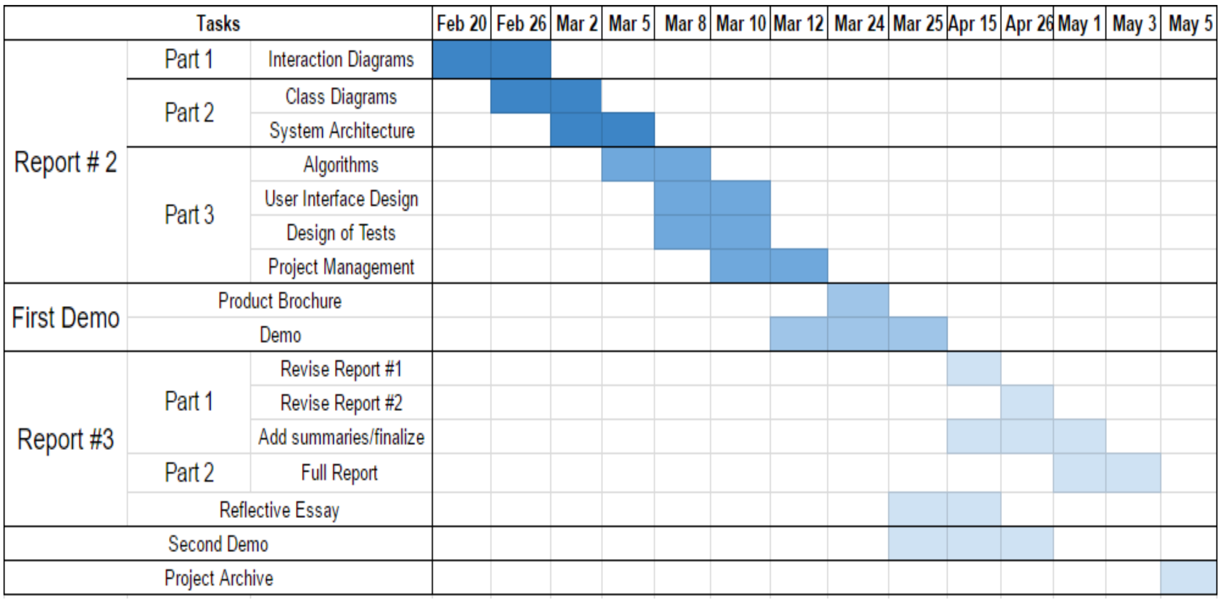
- 8GB RAM
- Quad-core Processor
- Network Card

Project Management

Project Coordination and Progress Report

Our team has not started development yet, so no use cases have been implemented. We plan to start development after submitting Report 2 - Part 1 since we are now getting into the system design. The team will be deciding on which technologies to use in order to accomplish our tasks most efficiently based on the requirements. Our plans for development are described below in the breakdown of responsibilities along with what we have already accomplished.

Plan of Work



Tasks			Feb 20	Feb 26	Mar 2	Mar 5	Mar 8	Mar 10	Mar 12	Mar 24	Mar 25	Apr 15	Apr 26	May 1	May 3	May 5
Report # 2	Part 1	Interaction Diagrams	■	■												
	Part 2	Class Diagrams		■	■											
		System Architecture			■	■										
	Part 3	Algorithms				■	■									
		User Interface Design					■	■								
		Design of Tests					■	■								
		Project Management						■	■							
First Demo	Product Brochure									■						
	Demo								■	■	■					
Report #3	Part 1	Revise Report #1										■				
		Revise Report #2											■			
		Add summaries/finalize										■	■	■		
	Part 2	Full Report												■	■	
	Reflective Essay										■	■				
Second Demo											■	■	■			
Project Archive																■

Note: The different colors indicate that there is a deadline for submission is between them.

Breakdown of Responsibilities

To make sure that our team is productive and efficient we divided up into 3 groups of two. Since we have completed an initial design of our system together, our next few weeks would be to build prototype of the system. Since we have not completely created detailed algorithms, we will try to build the UI and use basic/naive algorithms according to the concepts in the domain model. We created the tasks by grouping similar use cases in a subsystem and assigning each subsystem to a subteam. As each feature is developed by the sub-teams they will be performing tests on it so its not left until last minute.

Teams:
Dylan Herman and **Moulindra Muchumari** are responsible for Cleaning, Billing, and Manager subsystems.

Completed Tasks:
- Created preliminary UI design of the Manager's data and statistics view
- Planned out the use cases and domain models for the management subsystem

Current Tasks:
- Design what data points are to be collected from the restaurant and possible statistical measurements that can be analyzed from the data.
- Keep attending weekly meetings

Future Tasks:
- Implement the UI design for the manager's console and make changes accordingly
- Create/list out possible settings that the manager should manage
- Create the split check interface
- Implement the archiving and statistics system

Mit Patel and **Prabhjot Singh** are responsible for the Ordering subsystem.

Completed Tasks:
- Created the UI design for the chef's Order Queue
- Planned out the use cases and domain models for the Order Queue subsystem
- Created the mathematical models for managing order queue

Current Tasks:
- Implement the UI design for the chef's Order Queue
- Keep attending weekly meetings
-

Future Tasks:
- Create UI design for the waiter's tablet to place orders and receive notifications
- Create a sample menu/ menu categories
- Implement the Order Queue subsystem using the mathematical model and domain analysis

Nill Patel and **Raj Patel** are responsible for Seating and Inventory Management subsystems.

Completed Tasks:

- Created an interactive UI design for the floor plan using different colors for the status
- Keep attending weekly meetings
- Planned out the use cases and domain models for the Floor Plan subsystem
- Created the mathematical models for suggesting meal plans

Current Tasks:

- Implement the floor plan UI (only the non-interactive part)
- Keep attending weekly meetings

Future Tasks:

- Create the reservation interface frontend
- Design the UI for raw materials (ingredients) interface that displays the list of materials and their amounts
- Implement the food suggestion subsystem using the mathematical model and domain analysis

All team members managed the project equally!

References

- "What's is the difference between include and extend in use case diagram?" N.p., n.d. Web. 5 Feb. 2017. <http://stackoverflow.com/questions/1696927/whats-is-the-difference-between-include-and-extend-in-use-case-diagram>.

- Marsic, Ivan. "Software Engineering Project Report." Software Engineering Project Report - Requirements. N.p., n.d. Web. 19 Feb. 2017. <http://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html>.

- StarUML
  -Program used to create Diagrams

- Axure
  -Program used to create wireframes for the UI

- Russ Miles and Kim Hamilton: *Learning UML 2.0,* Reilly Media, Inc. 2006.