Auto-Serve

Restaurant Automation

Getting Served! Now Automatic.

Group 1

All members contributed equally.

Prem Patel
Sai Kotikalapudi
John Bartos
Scott Xu
David Shen
Joshua Devasagayaraj

Table of Contents

1. Customer Statement of Requirements

Many restaurants today still use the same basic methods from years ago to handle orders from customers. Often times this leads to complicated coordination of activities between chefs and waiters. This also means that other tedious tasks, such as inventory checks, fall to the managers to perform. This project seeks to introduce automation in privately-owned restaurants to alleviate some of the problems associated with current restaurant management practices.

1.1 Problem

After researching current restaurant practices and work from previous groups, we narrowed our focus to these problems:

1.1.1 Keeping Inventory and Determining When to Restock

Managers have the incredibly tedious role of keeping inventory and deciding whether or not to order more supplies for the restaurant. A survey taken by a previous group who interviewed the manager at a Buffalo Wild Wings says,

“[The manager] 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.” - Group 2, 2011

It was evident that the burden of the accounting work fell on the manager. We know that this burden could be partially reduced, if not completely eliminated, through automation.

Furthermore, after accounting for the current inventory, the manager must decide whether to restock or not. This decision requires the manager to estimate how long the restaurant’s remaining supplies will last. This estimation requires knowledge of past usage rates and other information about the restaurant’s past performance, and is not a trivial calculation; it has to take into account past usage rates and predict factors which will affect future usage, such as seasonal changes in demand and upcoming holidays.

Lastly, managers would need to know if their inventory was getting too low. If it falls below a certain threshold and they do not catch it, it would lead to costly results for the restaurants. At the very least, it would lead to a loss in profits and poorer customer ratings.

Software assistance could alleviate many of the above issues and boost restaurant efficiency by handling the inventory system autonomously, with minimal user input after the system is fully operational.

1.1.2 Keeping Diners Informed About Their Wait Time

One common thing that we noticed was that customers rarely know how long their food will take to finish. This was evident from our personal experience: waiters can give rough estimates based on what they know about the kitchen's current state, but at best those are still rough guesses.

Common consequences of being left in the dark are feelings of boredom or uncertainty, leading to the customer thinking, "is there time to step outside for a cigarette, or to the bathroom? Should I order an appetizer to make the wait more bearable?" Customers become upset when their food arrives later than expected since they could have ordered appetizers to soothe their hunger.

Another potential problem is the delay between individual dishes being finished in the kitchen and then being sent to a table. What if one dish takes much longer than the rest? If a diner wants his food as soon as possible -- that is, without waiting for the table's other orders -- he should be informed of when it's finished as well as given the ability to request its early delivery.

1.1.3 How to Cook and Deliver Menu Items that Customers Order

Currently, many restaurants have chefs cook orders on a first come first serve basis. Many members of our group have seen the classic pen and paper method of organizing incoming orders at restaurants they have gone to.

Typically, the waiter takes an order and hangs the description of the order at the chef counter, where the chefs take and prepare orders one by one. After cooking, the food is placed in a "ready area" with the order description.

Waiters additionally have the job of searching for their respective orders and checking that all the food is ready; with no system to directly notify waiters, this forces them to periodically return to the kitchen for essentially no reason.

After looking at this process, we knew there was a lot of room for improvement. The overall efficiency of the restaurant and consequently customer satisfaction can be improved.

1.1.4 Tracking Menu Item Popularity and The Menu Item Rating System

Restaurant managers can sometimes have a difficult time determining what menu items are popular and what items are infrequently ordered. Customers too want to know what dishes are the best and what dishes to avoid. For most small restaurants, advertising is achieved through word of mouth. New customers of the restaurant usually hear reviews from their peers. Their peers usually recommend the restaurant because of a certain menu item that they liked or because the service provided by the restaurant is excellent. This influences what choices the customers have.

When designing or updating the menu, restaurant managers have the difficult dilemma of adding great new dishes and removing stale, unpopular old dishes. By making customer satisfaction information available in the form of popularity and ratings numbers, the manager has a source of hard data and a unified view of the customer preferences to base his decisions off of.

A list of menu items sorted by popularity and/or rating would serve everyone well: instead of asking waiters (who have a smaller role in our automated restaurant) for recommendations, customers can view the popular dish list directly. Instead of trying to guess what dishes are most and least popular, managers can view this information directly and easily and make better informed decisions.

1.1.5 Things to Note before Reading

When developing our solutions to these problems, we came across a few issues.

1.1.5.1 Menu Items vs. Dishes

When deciding the names of menu items that customers order, we were debating between calling them “menu items” or “dishes”. In the customer’s point of view, they can be seen as menu items. However, in the chef’s point of view, they can be seen as dishes.

There are times in this document when the terms “menu item” and “dish” are both used interchangeably. Both terms refer to the same concept; in most places, “menu item” will be used, but in cases where “dish” is more appropriate it is used instead.

1.1.5.2 Table Order vs. Table

When we designed the queue for waiters who had to deliver dishes back to the customers, we initially decided to represent the table as a “table” that represented the people who ordered many items from the same table. We realized that this was a poor decision since tables might order again. Therefore, for clarity, we changed the naming convention from tables to table orders (or just order) where the table order just represents the composite order for all people at the table. To represent the actual table that the customers are seated at, we just use a table ID number since the actual table need not be a concept, just an attribute.

1.2 Glossary of Terms

Many of the terms you see here can be understood as you read the report. They are also listed here for clarity and formality.

1.2.1 Technical Terms

Order Queue - The queue of menu items, ordered by customers that are ready to be cooked by the chefs.

Ready Queue - The queue of menu items that are ready to be delivered to the customers.

Order - A concept that represents the list of items that the customer ordered.

Table - concept that represents the physical table that customers are seated at.

Table Order - concept that represents the actual composite order of all the customers seated at a certain table.

Inventory System - an electronic book keeping of the current inventory in the restaurant. This includes its raw ingredients, current menu items, etc.

Scheduling Policy - a predefined set of rules to determine where in the queue or line the next item should go.

1.2.2 Non-Technical Terms

Chef - Cooks all the food in the restaurant.

Manager - Manages the inventory, orders more supplies, and deals with overall management and finances of the restaurant.

Waiter - Handles delivering food and

Customer - the person who the service provided by the restaurant is being given too.

Ingredient - A food that is used to create a menu item. e.g lettuce, carrots, etc.

Menu Item - The food that that is listed on the menu given to customers to choose from. e.g various burgers, pasta, etc.

Dish - Equivalent to the menu item, but used when describing queuing related to chefs and waiters for simplicity and is more appropriate.

2. System Requirements

2.1 Proposed Solution

Our solution focuses on the problems we highlighted in the first section. We have left out trivial elements of the system such as login, ordering items, indicating dirty tables, etc. These items were already done by previous groups and their reports are applicable. We have focused on our core ideas that make our product genuine and worthwhile.

2.1.1 Inventory System

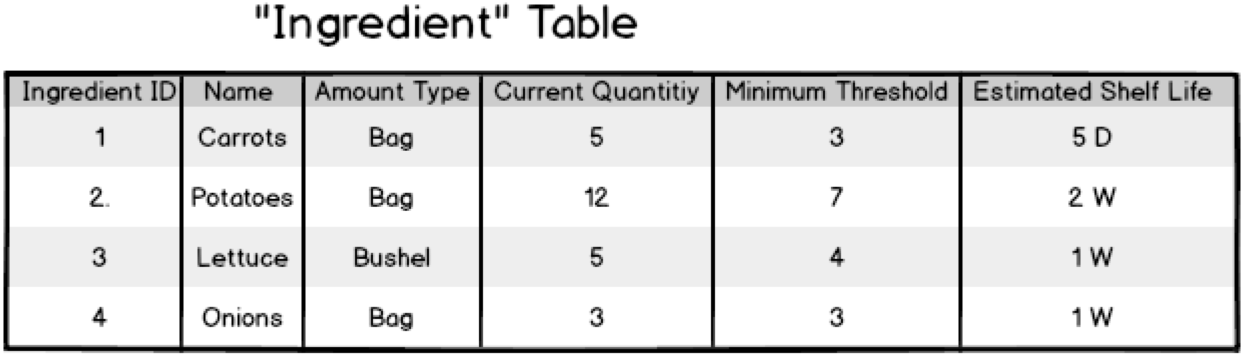
To combat the problems involved with keeping inventory and determining when to restock, we set out to design a system that takes as much of the burden as possible off of the manager.

With our smart inventory system and prediction algorithms, we can effectively reduce the amount of times the manager has to physically check to stockroom for ingredients and partially automate the restocking process.

To implement this inventory system, we chose to use a database. MYSQL seemed the most cost effective choice since it is the most popular and freely available. Since it is a medium sized restaurant, it will suit our needs. Not only can the database hold information about individual ingredients, but we can create tables in the database to reference these ingredients so that we can store menu items to represent containers.

In database terminology, a **table** represents a matrix of r rows and columns that is serialized and stored as data depending on the type of database.

In the database, our table for raw ingredients can be visualized like this:



"Ingredient" Table

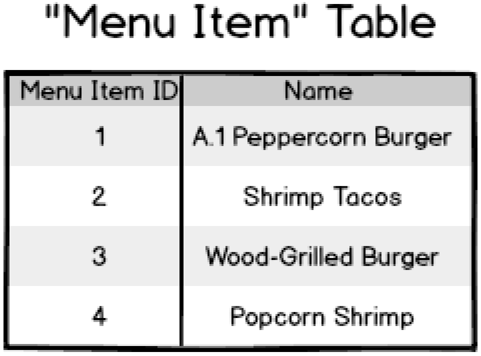
Ingredient ID	Name	Amount Type	Current Quantitiy	Minimum Threshold	Estimated Shelf Life
1	Carrots	Bag	5	3	5 D
2.	Potatoes	Bag	12	7	2 W
3	Lettuce	Bushel	5	4	1 W
4	Onions	Bag	3	3	1 W

Figure 1: This is Ingredient table as represented in the database.

Ingredient ID represents the identification of the ingredient inside the table to relative to the other ingredients. **Name** is the name of the ingredient. **Amount Type** is the type of quantity that the ingredient is measured in. **Minimum Threshold** is the manager specified minimum amount of this items that should be in the inventory. **Estimated Shelf Life** is the estimated shelf life of the ingredient or the amount of time that the ingredient can be stored before it goes bad.

When the manager issues an order for raw ingredients to the supplier, this table can be automatically updated when the order is verified by the manager. As long as the supplier is a trustworthy source, the inventory will be correctly updated without manager intervention.

Furthermore, a table for menu items will hold all the menu items available to the customer.

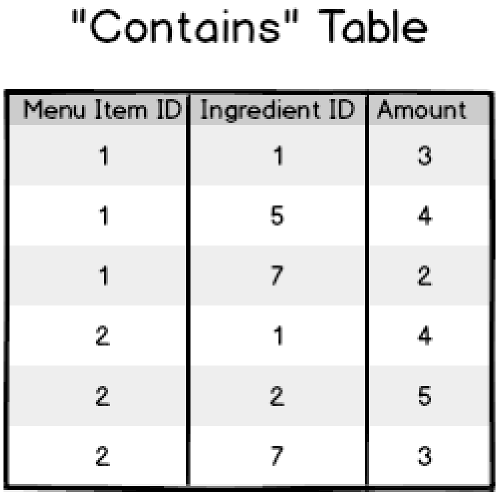


"Menu Item" Table

Menu Item ID	Name
1	A.1 Peppercorn Burger
2	Shrimp Tacos
3	Wood-Grilled Burger
4	Popcorn Shrimp

Figure 2: This is the Menu Item table as represented in the database.

To store mapping between menu items and ingredients, we create an additional table called “Contains” to store the **one-to-many relationship** that menu items have with ingredients.



"Contains" Table

Menu Item ID	Ingredient ID	Amount
1	1	3
1	5	4
1	7	2
2	1	4
2	2	5
2	7	3

Figure 3: This is Contains table as represented in the database.

When chefs cook menu items from the menu, the inventory system can be automatically updated to show the remaining inventory after cooking each dish. Of course, when making each dish, there is going to be some error in measuring by the chefs and the total amount of inventory usage will have a small percent error. We discuss this issue later.

Since the system is aware of each menu item in the menu, the amount of times the menu item is ordered, and the contents of each menu item, the system can keep a real-time measure of the amount of ingredients in the inventory.

Having the system keep track of inventory leads to a plethora of features that we elaborate on later on such as future predictions of inventory requirements and knowing how much of an item we can produce.

One thing we must not forget is that the manager will always have access to the inventory system in the event that he must manually change the stock.

2.1.2 Inventory Usage Prediction

To streamline the management process, our system has the ability to predict ingredient usage rates. This feature solves the issue of the manager having to use historical data about the restaurant in his decision of whether to restock.
This is a data mining problem in which we take usage data over a lengthy period of time and develop predictions of how much inventory will be used and how much should be ordered based on the season, holidays, weekday, etc.

As we lack experience in this area, we decided to start out with a relatively simple approach to the problem. At the suggestion of our advisor, we also investigated more complex autoregressive, moving-average, and autoregressive-moving-average or ARMA models, but without the proper background in signal processing and statistics we found these models too difficult to implement.

A detailed description of our algorithm is included further in this document, under "Mathematical Models".

2.1.3 Inventory Alerts

Another feature of the inventory system that introduces novelty are alerts. This feature alleviates the issue involved with managers having to determine when inventory falls below a certain threshold.

Alerts are triggered when inventory of items fall below certain limits. These limits though obvious it may seem, are not just the bare minimum to cook a menu item but rather is threshold set by the manager. Again when group 2 in 2011 interview Buffalo Wild Wings and asked the manager how often he restocks, he said,

"Every Wednesday and Saturday regardless of demand. Always have a surplus."

After reading this, we found it appropriate to improve this procedure and not only to automatically determine when it is best to restock, but also send the appropriate demand to the supplier himself (with manager approval, of course).

Our design of the alert system follows:

We use the predictive ingredient usage model to estimate when an ingredient falls below its critical stock threshold. The manager can set these thresholds based on demand and perishability, but in general it should be no less than 10% of the usage in the time period between restocks or the amount of the restocking order.

By recursively applying the equation we can estimate ingredient usage for an arbitrary day. First we calculate usage for day n+1 and subtract it from our current stock level. If it is below the threshold, we send the alert. If it is not, we apply the formula for day n+2 using our estimate for day n+1 as an input; if the total usage on day n+1 and n+2 cause stock levels to fall below the critical threshold, we send the alert. If not, we iterate once more; this process continues until it reaches the day when the stock falls beyond the threshold.

In the final system, we will most likely estimate usage extremely conservatively to prevent any item from going out of stock. With time, the restaurant manager can manually update/lower the thresholds and other parameters.

Customer Wait Time Estimation

Our system will estimate the wait time for a table order using the existing menu item queue that exists in the kitchen terminal. Each dish in the queue is associated with a table; by checking each dish for a given table, we can determine the expected finish time of the last dish and use it to determine the total wait time for the table. For tables which request dishes be delivered as they are finished, we will display the estimated wait for each dish instead of for the entire table.

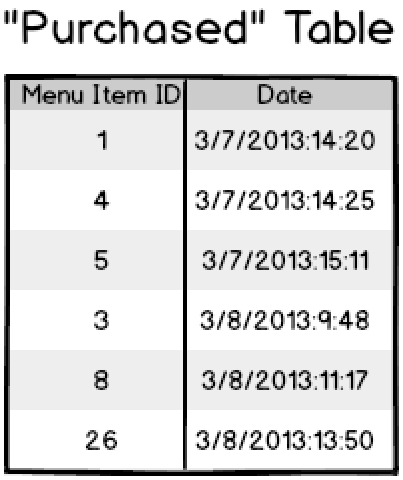
The detailed algorithm is written in the mathematical model.

2.1.4 Food Popularity

To solve the problem of determining what items are popular on the menu and what items are not, we have designed a new way so that both the manager and the customer will be able to know what the "hot items" on the menu are.

Since, we are using an inventory system; the task of providing popular items to the manager becomes trivial. By knowing which menu items customer's order, and the number that are ordered every day, the manager can be provided a clear cut overview of the most popular items in the current day, week, month, season, etc.

To determine food popularity we need keep track of what items are being bought every day. As we are using a database, we can add another table to keep track of this.



"Purchased" Table

Menu Item ID	Date
1	3/7/2013:14:20
4	3/7/2013:14:25
5	3/7/2013:15:11
3	3/8/2013:9:48
8	3/8/2013:11:17
26	3/8/2013:13:50

Figure 4: This is Purchased table as represented in the database.

This “Purchased” Table will be updated with the date of every order that is purchased. This way, it is simple to find the amount of times a menu item is ordered within a certain time frame.

For instance, if we want to find the most popular items last week. We just need to tabulate the number of times each item was purchased during that week.

Since this is just a simple search and count procedure, it is not needed to be described, and therefore not needed to be shown in the mathematical model.

Similarly, the system can also provide popular items to customers; however some customers may not trust statistics that the restaurant generates. The customer may think that the restaurant may be trying to sell more of one menu item over another.

Therefore, we designed a rating system so that customers can rate various menu items that they have eaten and write their own comments about items. Similar to Amazon.com’s rating system, the rating system is designed so that customers can “like” other customers’ review and rating so that these higher quality reviews will be higher up in the list of reviews that the customer may read for each menu item.

Since, the rating system is designed similar to Amazon.com, there is no need to go further into detail about it.

2.1.5 Chef’s Interface and Menu Item Queuing

As we focused on efficiency and time saving in the automation process, one realization was that chefs can prepare foods faster if they are given similar foods that can be cooked together or in parallel. One example of this is having a chef cook two burgers and then a cheesecake rather than have a chef cook a burger, then a cheesecake, and then back to a burger. This essentially cuts the time by the length of one burger. When the chef gets the ingredients for one burger, he may also get the ingredients for the other burger and cook them in parallel.

Similar to operating system design, this problem becomes similar to that of scheduling and prioritizing processes, however in this case the processes are menu items to be cooked, and our cpu is the chef.

Keeping this in mind, we designed an optimal scheduling algorithm based on a priority queue where the new menu item will be given a priority based on what's currently cooking, and what's currently on the queue. Of course, menu items cannot be preempted because no chef would stop halfway in making a menu item as that would be wasteful and sometimes ruin the menu item in certain cases. Thus we focused on non-preemptive scheduling of menu items.

Our solution to the problem can be visualized as follows:

Each menu item on the queue for chefs can be modeled as a compound data structure. There are many parts to the menu item structure but I will only lay out the parts that pertain to the chef’s queue.

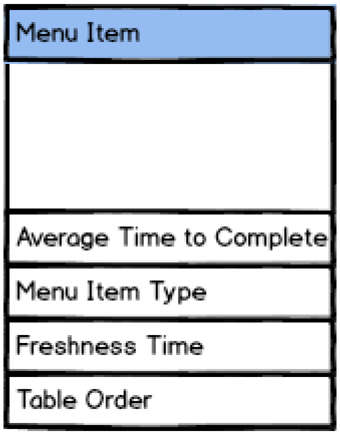


Figure 5: These are the attributes of the Menu Item that pertain to queuing.

Here we focus on three properties of the menu item,

Average time to complete: The average total time that the chef takes to cook the menu item in questions, this time must be calculated based on real data that the restaurant takes when operating under normal conditions. In our system, we assume that this data is available for us.

Menu item Type: The category of menu item that the menu item falls under.

Freshness Time: This is the time that the dish can be kept warm and still retain its freshness

Table Order: This is a reference to the table order that this menu item belongs to. The table order is described in detail in the section involving queuing with waiters. A table order of zero means that the item does not belong to a table order. This is used later on in queuing for waiters.

Here is the basic scenario of when a burger will move up the queue.

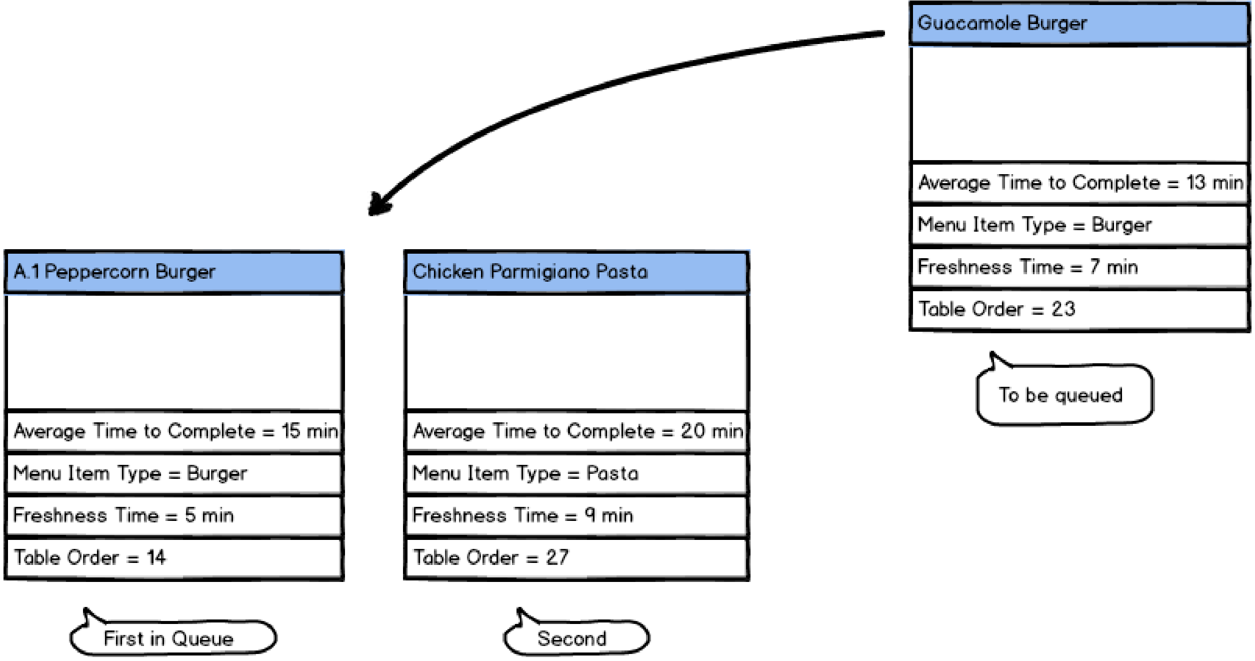


Figure 6: The guacamole burgers get queued with the A.1 peppercorn burger.

In this situation, since the guacamole burger is a burger, it can be queued together with the peppercorn burger, allowing the chef to make these burgers in parallel and essentially cutting the total turnaround time by the time to complete of the Guacamole Burger. One restriction to this is that the Guacamole Burger is only queued together because it average time to complete is less than the average time to complete of the

Peppercorn Burger. If the Guacamole Burger had an average time to complete greater than that of the Peppercorn Burger, it would not be queued since that would add time additional time to the queue that would increase the wait time of the pasta. This is not desirable since that would increase the wait time of a customer that ordered the pasta caused by someone who ordered after him. This "customer-first" approach is the key among restaurants and it's maintained in this queuing policy.

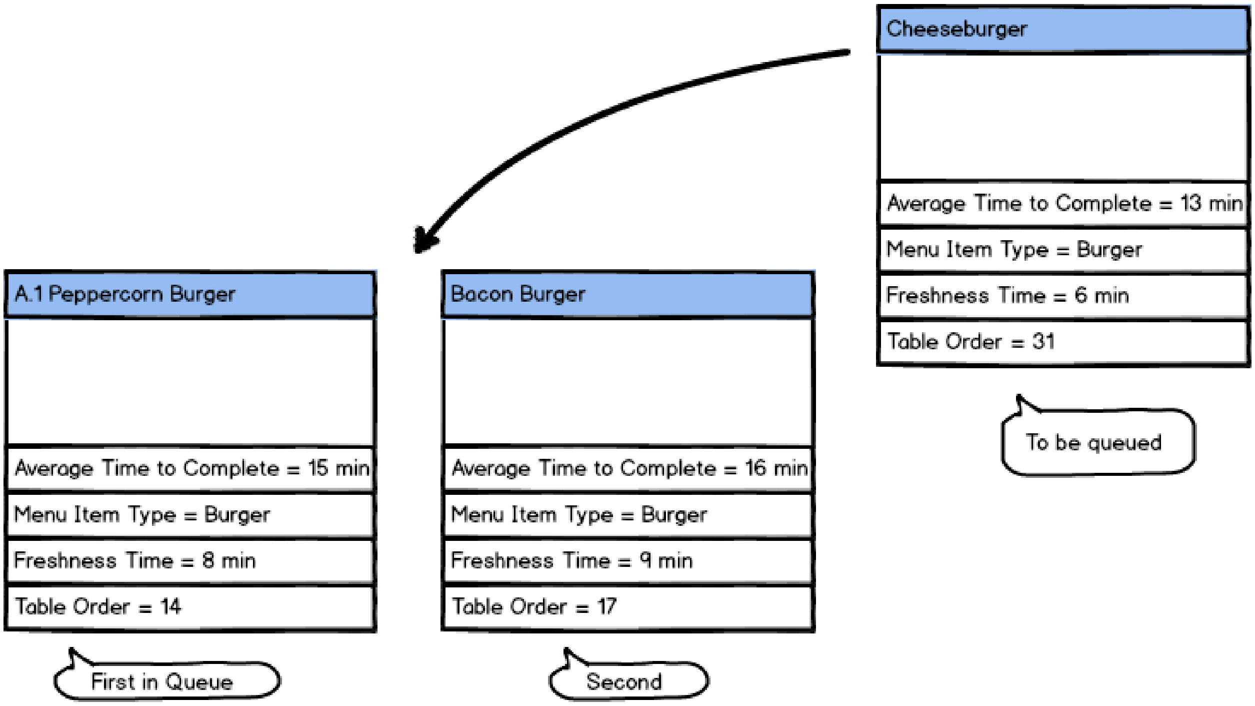


Figure 7: The Cheeseburger is being queued ahead of the Bacon Burger because it has a lower average time to complete.

In this situation, we have multiple burgers that already on the queue and we are adding another burger. In this case, we can find the first burger that has an average time that is greater than the cheeseburger's average time to complete. Thus, we when we are scheduling the Cheeseburger, there is more room for other burgers to be scheduled in the future.

In the previous two scenarios we have neglected the fact of what table the burgers are being ordered from. Suppose we have a group of people that are ordering from the same table. In our current scheduling policy, some members of the group will have food that will be cooked far ahead of others in the table. Therefore, some of the food will not be as fresh as others. Even if the food is still kept warm, certain foods rely on freshness for their taste and consequently customer satisfaction.

Therefore, we added a certain freshness factor to the policy whereby dishes cannot be put ahead of other dishes from the same table by this freshness time. Thus dishes cannot be queued earlier than the freshness time away from the rest of the table order.

Here's is an example of when freshness time comes into play:

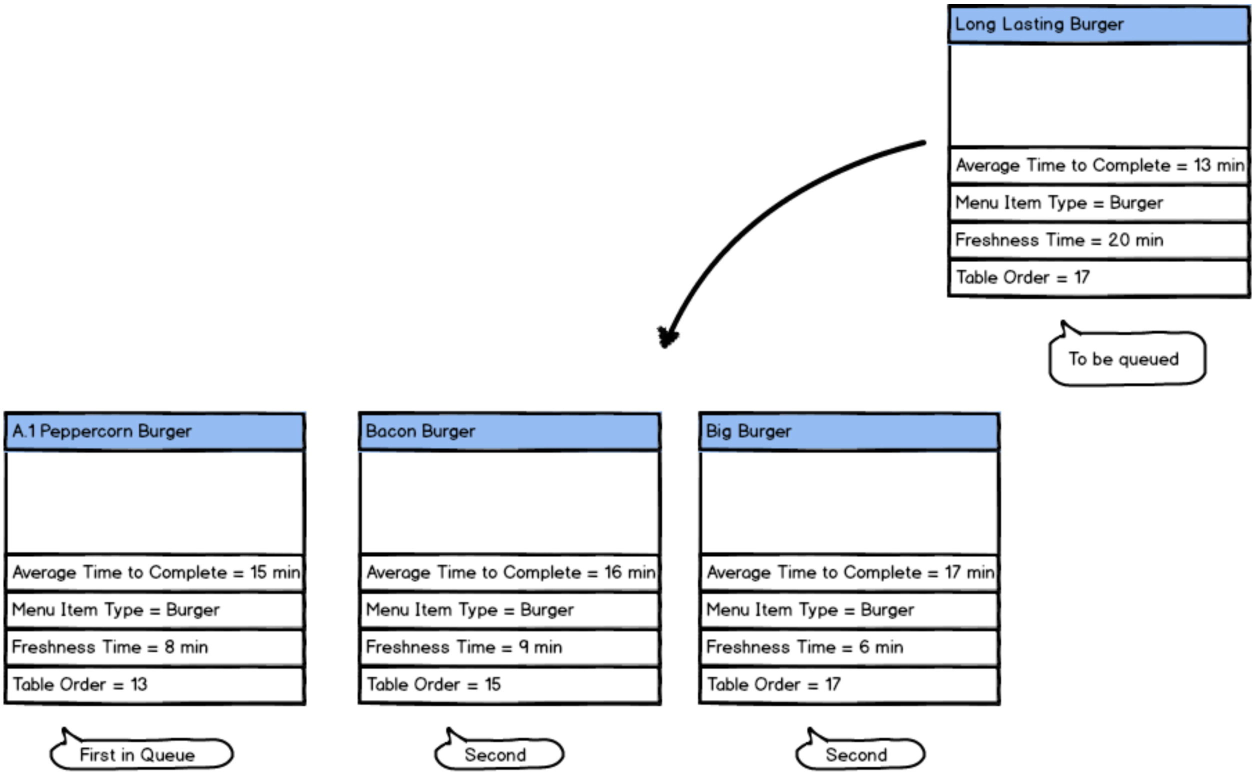


Figure 8: The long lasting burger is being queued with the Bacon burger and not the peppercorn burger because of freshness time constraints.

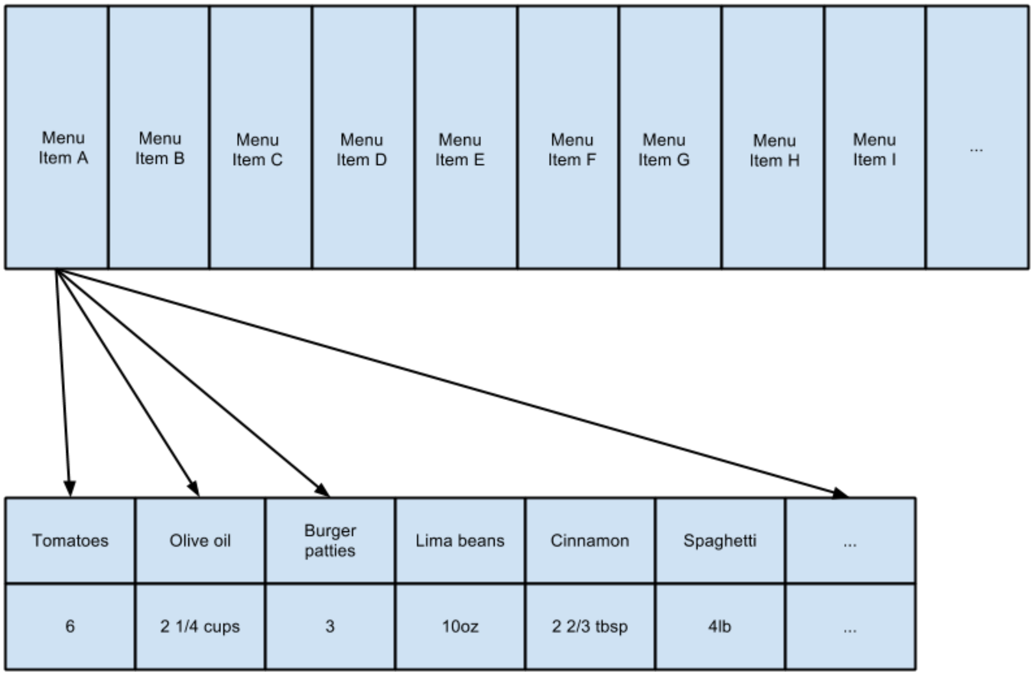
The first thing you may notice is that "Table Order" is another property of the menu item. The table order will be described in a later section, but this attribute is just a reference to the table order that contains this menu item when it was ordered.

What happens in this situation is that the long lasting burger cannot be queued with the peppercorn burger because the time that it would stay out would be larger than its freshness time. Therefore it will be queued with the next best item, which is queuing with the Bacon burger.

The formal algorithm is described in the mathematical model.

2.1.6 Shared Ingredient Display

The chef's interface will show the menu item queue. To further increase the efficiency of the kitchen, our system will determine the total ingredient usage of each ingredient for all menu items currently residing in the queue. By displaying this data directly to the chef and sous-chefs, it will be possible for the kitchen to prepare ingredients and therefore dishes more quickly.



Tomatoes	Olive oil	Burger patties	Lima beans	Cinnamon	Spaghetti	...
6	2 1/4 cups	3	10oz	2 2/3 tbsp	4lb	...

Figure 9: The Chef will be able to see a list of all the ingredients needed for the the current dishes in the order queue.

Originally, we intended to show which ingredients were needed for every dish in the queue; however, doing this quickly caused an unacceptable amount of UI clutter. The current design only displays these relationships for the dish that is up next to be cooked. This allows the chef to quickly understand what ingredients must be retrieved immediately, while the aggregate ingredient usage list shows what ingredients will be needed in the medium and long term. This enables the chef and kitchen staff to have a clear view of what must be done now and what can be done later.

2.1.7 Queuing of Orders for Waiters

Similar in design to the chef's menu item queue, this queue represents finished menu items grouped by table and ready to be delivered to the diner. However, the items of this queue are table orders instead of menu items; whenever an entire table order becomes finished in the kitchen, that table's order is placed on the table queue. The waiter checks this queue from the floor terminal and delivers the menu items in a first come first serve manner. However, most restaurants won't deliver items to tables if all the items in the table is ready. We decided to provide the option to the customer whether to wait out till all the items are ready, or just deliver them on a first come first serve basis.

Therefore, we designed the queuing for table order as two queues. One is the wait queue and one is the ready queue. The ready queue will hold either tables that are ready, when the customer chooses to deliver items when all of them are ready, or menu items that are ready to be delivered to tables where the customer chooses to deliver items on a first come first serve basis. The wait queue is a structure of tables and each table will hold the menu items that belong to that table.

Each table structure can be seen as this:

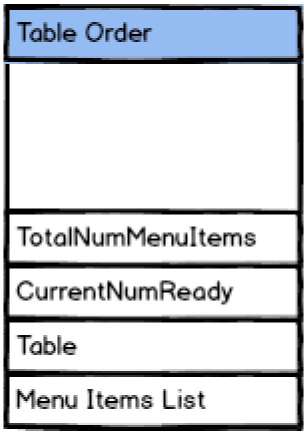


Figure 10: The Table Order structure and its properties that are used in queuing for waiters.

TotalNumMenuItems: The total number of menu items associated with this order..

CurrentNumReady: The category of menu item that the menu item falls under.

Table: This represents the table that the order is coming from. It will just be a number in our case.

Menu Items List: This is the list of menu items that belong to this order.

If the customer chooses to deliver items by table, then a table structure is created when he orders as a group. For each menu item that he creates, he adds a reference to it in the Menu Items List. The total number of menu items and the current number that are ready are also set so that one can determine when the table is ready. Table represents

Our Wait queue can then be described as a list of tables as such:

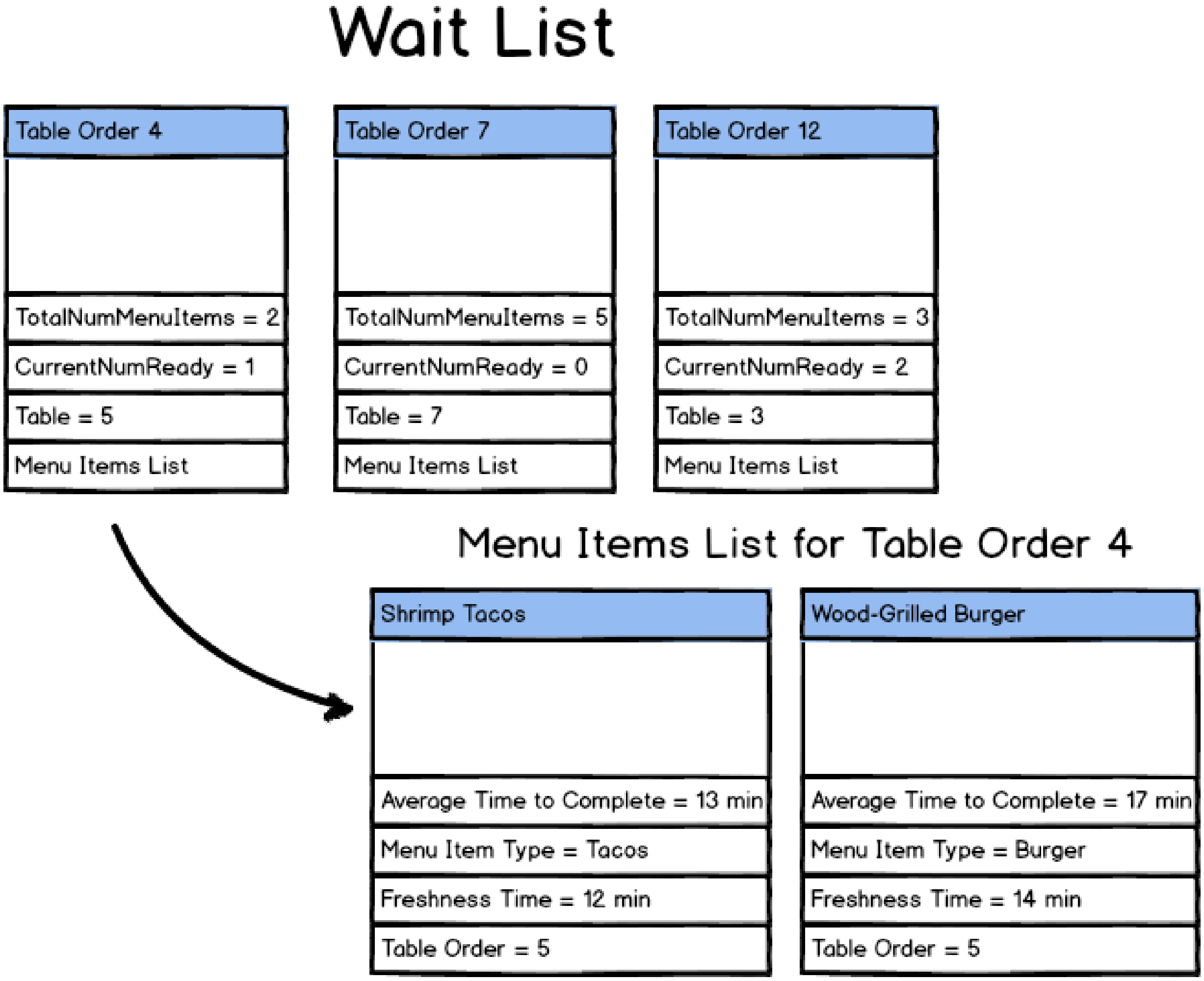


Figure 11: An example of the wait list and an expansion of the menu items list for Table Order 4.

When the customer creates an order, if he selects for the order to be grouped for the table, a table structure will be created and references to the menu Items will be stored in the Menu Item List. Then, when items are done cooking, and are ready to be delivered, they go through each table if the wait list, and if they find themselves in one of the Menu Item lists, then they add the one to the current number ready attribute. When the current number ready is equal to the total number of menu Items, then the table can be queued to the ready queue.

Now the ready queue can contain either tables or individual menu items, again depending on whether the customer chooses to hold out for the group or deliver the item on a first come first serve basis.

The ready queue can be described as follows:

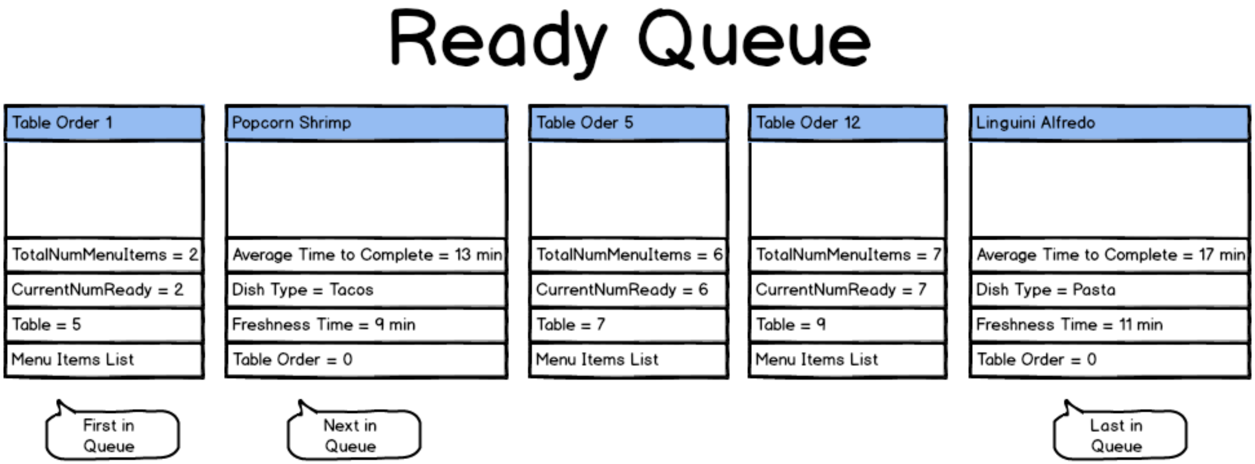


Figure 12: The ready queue is based on a first come first serve basis where whatever is added into the queue will get delivered to the appropriate customers. It consists of both Table orders and individual menu items.

The ready queue has a simple scheduling policy, its a first come first serve basis. We can see that it consists of both table structures and individual menu items. This way, depending on whether the customer decided to wait for the group or get his food as soon as its ready, it will be queued accordingly. A table order of zero represents that the menu item does not belong to a table order.

2.1.8 Deployment of The System

After designing our solutions to the problems, we were left with the decision of determining how to deploy it in a restaurant environment. The main focus of our deployment was put into making the system simple, high scalability, easy to setup and cost efficiency.

When thinking about how to deploy the system, We noticed that today's restaurant systems are deployed using specific machinery. These machinery are usually built by companies that contributed to the construction of the restaurant. Therefore, they are meant to be static and last the lifetime of the restaurant and consequently they are quite expensive and are rarely expandable.

When we designed our system, we tried to improve on these aspects. We targets a platform for the system that is both cost efficient and easily expandable, namely the android operating system.

Our system will run on numerous android tablets. The manager will have a single tablet that will act as his console. The chefs will have a number of tablets depending on the number of chefs that are at the restaurant. These tablets may be mounted to station so that the chefs may cook and handle the tablet easier. The same goes for the waiters. There will be a tablet at every table so that the customer can interact with the system.

Since Android tablets have become well integrated with society and many people know how to use them well. Therefore, having them instead of built in consoles are incredibly better for a number of reasons.

First, having a tablet system will allow us easily scale the system to fit the needs of the restaurant. A larger restaurant will just need to purchase more tablets than a restaurant of smaller size.
Second, the system is very cost efficient. Android tablets are relatively very cheaper than the custom equipment created to hold the system . Custom equipment like the consoles currently in restaurants.

Third, the system is highly expandable. When the system is deployed on android tablets. Providing updates to the system becomes simple. For every update that is created for the system, it can be pushed to the tablets.

2.2 Enumerated Functional Requirements

We felt it was best to use requirements over user stories since some of our ideas were strictly system based. For instance, it was difficult to describe any queuing policy through a user story since the actual scheduling wasn't trigger.

Identifier	Requirement	PW
REQ - 1	The system shall store a database of ingredients and the following information for each ingredient: • Name • Menu items it is used in • Current stock level • Estimated shelf life	5
REQ - 2	The system shall store information on the raw ingredients of the menu items such as the estimated shelf life, and the menu items that the ingredient is used in.	3
REQ - 3	The system shall store the following information for each menu item: • Ingredients required • Amount of each ingredient • "Freshness" value representing the maximum time this dish should be allowed to wait after being prepared (used in dish queue scheduling)	3
REQ - 4	The system shall predict the usage rate of each ingredient and predict the day that the ingredient is expected to fall below a predefined restocking threshold using historical information gathered from the restaurant.	4
REQ - 5	The system shall alert the manager when an ingredient's stock level falls below a certain threshold.	2
REQ - 6	When an ingredient's stock level falls below its restock threshold, the system shall prepare a restock order and send it to the manager for verification. When it is verified, the system places the order.	3
REQ - 7	The system shall predict inventory usage for the next seven days using the previous seven days and show it to the manager.	2
REQ - 8	The system shall provide a prediction to the manager when the restaurant will run out of food or fall below a certain threshold in the future.	5

REQ - 9	The system shall give the customer a choice of delivering menu items all at once (by default) or deliver each item to the table on a first come first serve basis.	3
REQ - 10	The system shall queue menu items of the same type together so that chefs can cook them in parallel. However, if the customer wants items to be delivered as a table, then items cannot be queued too far ahead of the rest of the table to maintain freshness. The system shall maintain a log of each table order that was placed/edited	5
REQ - 11	The system shall show the sous chefs shared ingredients between menu items on the chef's queue so that the sous chef can prepare ingredients beforehand for upcoming menu items.	2
REQ - 12	The system shall predict the wait time for menu items that are on the menu and display that information to the customer.	5
REQ - 13	The system shall queue orders on a first come first serve basis for waiters based on table, if the customer chooses for orders to be delivered when all orders belonging to the table are ready or individually if he chooses to deliver orders as soon as they are ready.	5
REQ - 14	The system shall rank dishes by rating and popularity and display lists of the most popular and highest rated dishes on the menu.	3
REQ - 15	The system shall use a menu system to keep a list of all the menu items offered at the restaurant.	5
REQ - 16	The system shall allow the manager to manage the menu items on the menu.	3
REQ - 17	The system shall allow the information of the menu item to be viewed by the customer.	5
REQ - 18	The system shall allow the manager to add, remove, update, and disable menu items on the menu. The system shall also keep a log of the information that is edited on the database.	4

2.3 Enumerated Non-Functional Requirements

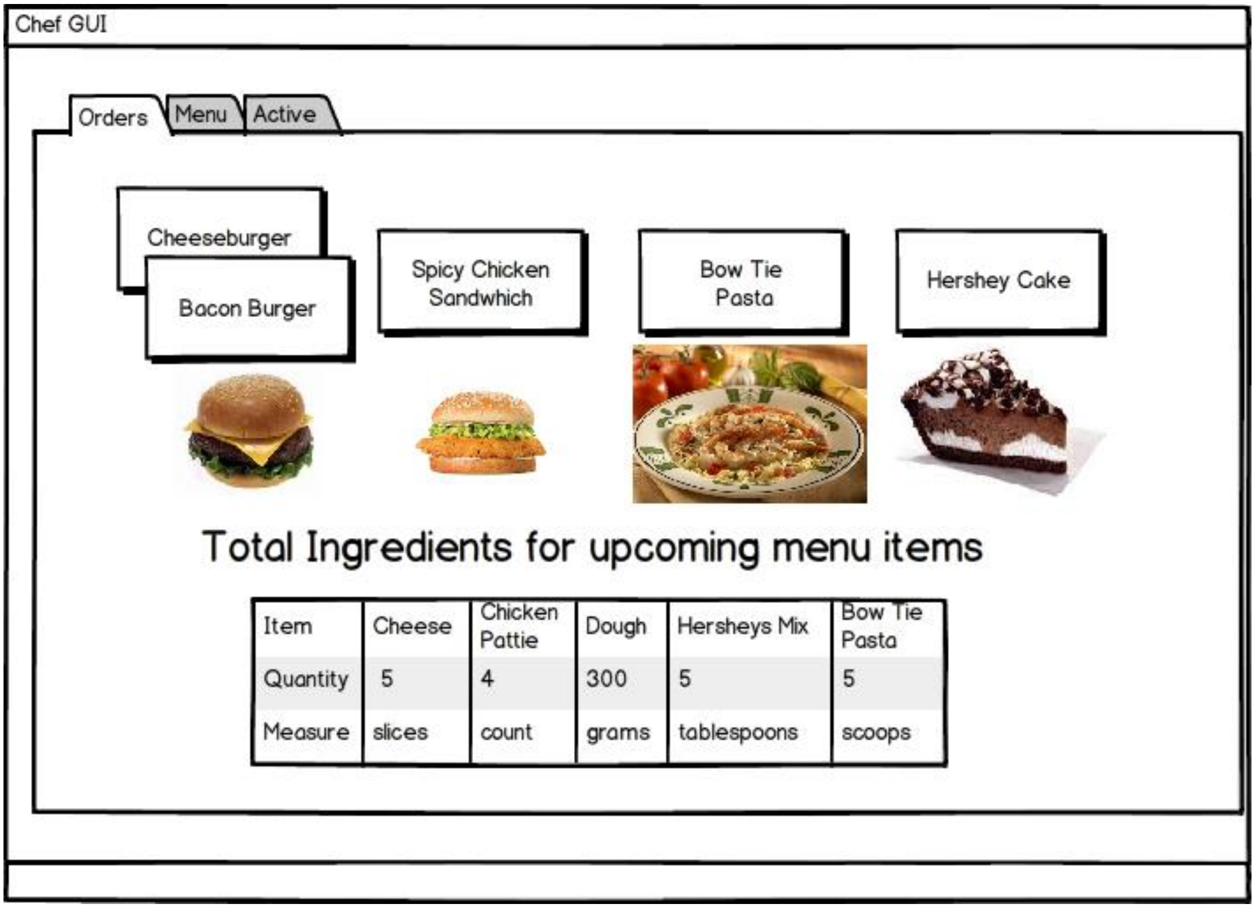
One thing to note is that we do not have any non-functional requirements. The reason for this is that our report details only our ideas rather than a complete system that incorporates requirements that may have been done by another group or trivial requirements such as login, register, display, etc. Since our ideas were completely functional, there were no requirements that were nonfunctional.

2.4 On-Screen Appearance Requirements

Identifier	Requirement	PW
REQ - 19	The system shall display the menu and make the menu items selectable to view to the users.	5
REQ - 20	The system shall display the average wait time for orders to customer next to the menu items.	3
REQ - 21	The system shall display all the ingredients in the inventory so that the manager can view them.	2
REQ - 22	The system shall display the ready queue which is list of tables or individual menu items from which users (usually waiters) can select the next item to deliver to the table.	3
REQ - 23	The system shall display the order queue to the chef as well as options to select which dish will be prepared. If the dish is unable to be prepared the chef will have the option to disable the menu item.	4
REQ - 24	The system shall be able to be able to display an option to the customer allowing him to choose between delivering items on a first come first serve basis or holding out until the table's items are ready.	1
REQ - 25	The system shall display the inventory usage for the next seven days to the manager.	5
REQ - 26	The system shall be able to show a notification to the manager when it needs to alert the manager in the form of a pop up notification and email.	5
REQ - 27	The system shall be able to create a request to the supplier in the form that the supplier specifies.	3

REQ - 28	The system shall be able to send notification to the proper interface.	4

2.5 User Interface Mock-Up for On Screen Requirements



Total Ingredients for upcoming menu items

Item	Cheese	Chicken Pattie	Dough	Hersheys Mix	Bow Tie Pasta
Quantity	5	4	300	5	5
Measure	slices	count	grams	tablespoons	scoops

Figure 13: The chef will be able to select the dish of the order queue by touching on the button which indicates the dish has been selected to be prepared. Notice that the first item is actually two items that were queued together.

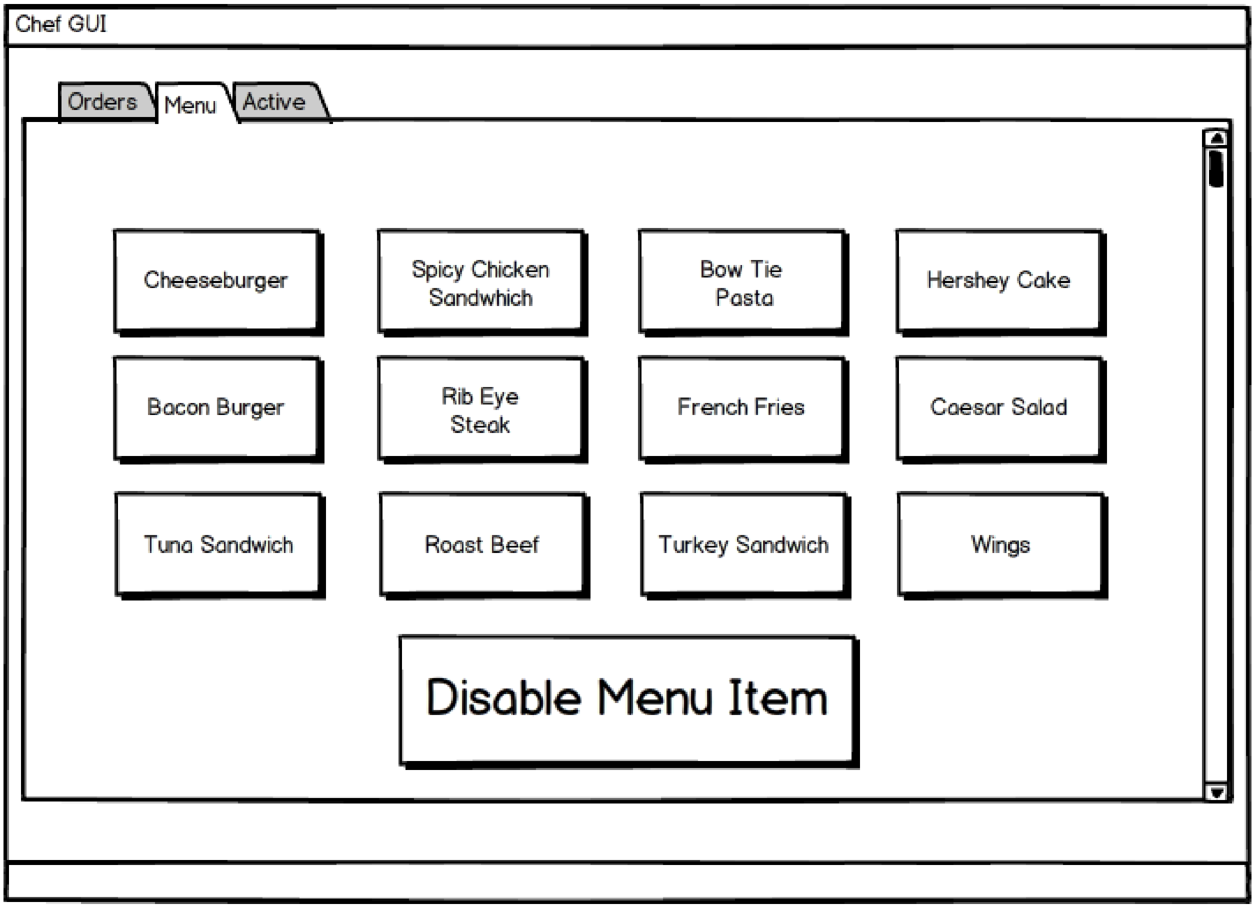


Figure 14: The chef will be able to disable the menu item on the menu if the dish is not able to be prepared by taping on which items he wants to disable (he can disable more than one at a time) and then tapping the disable menu item button.

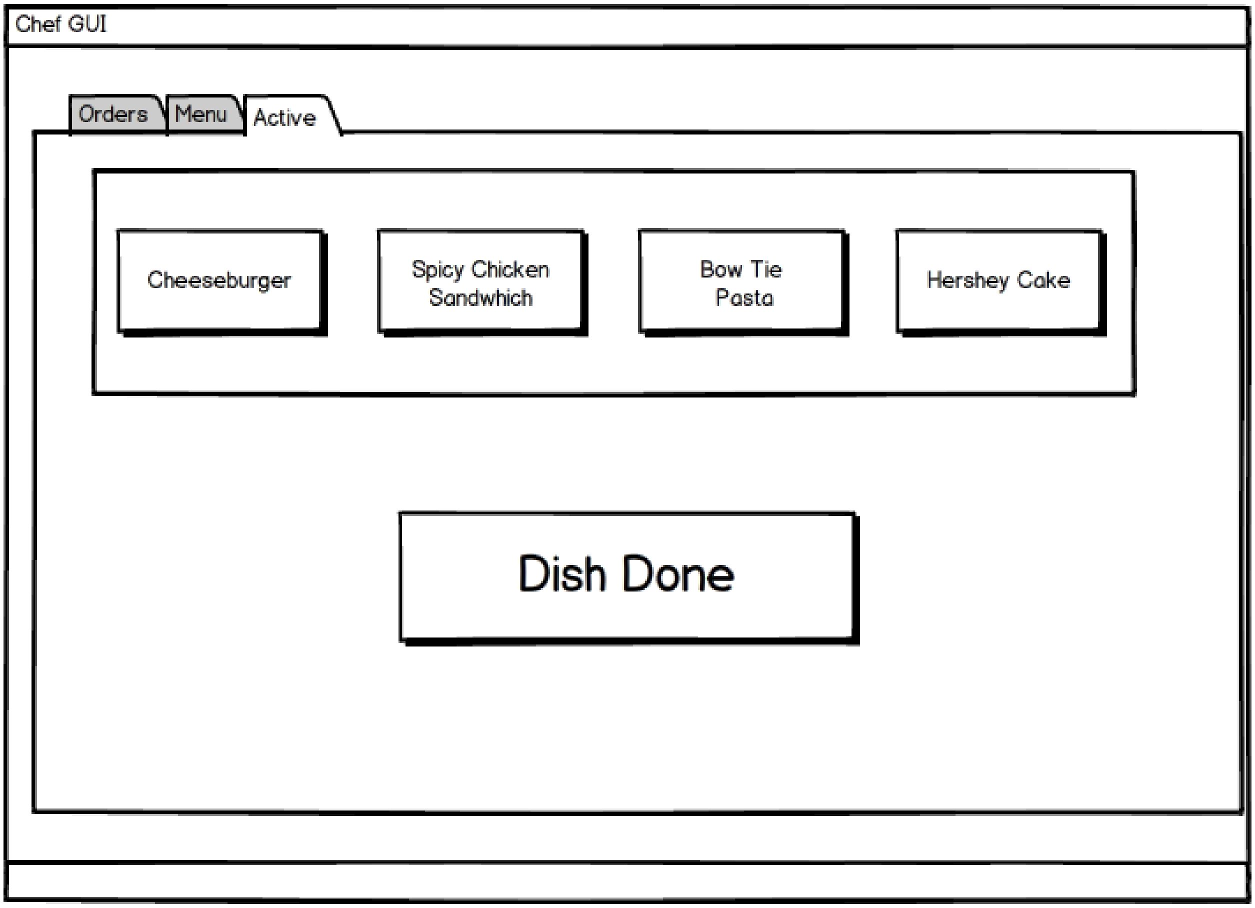
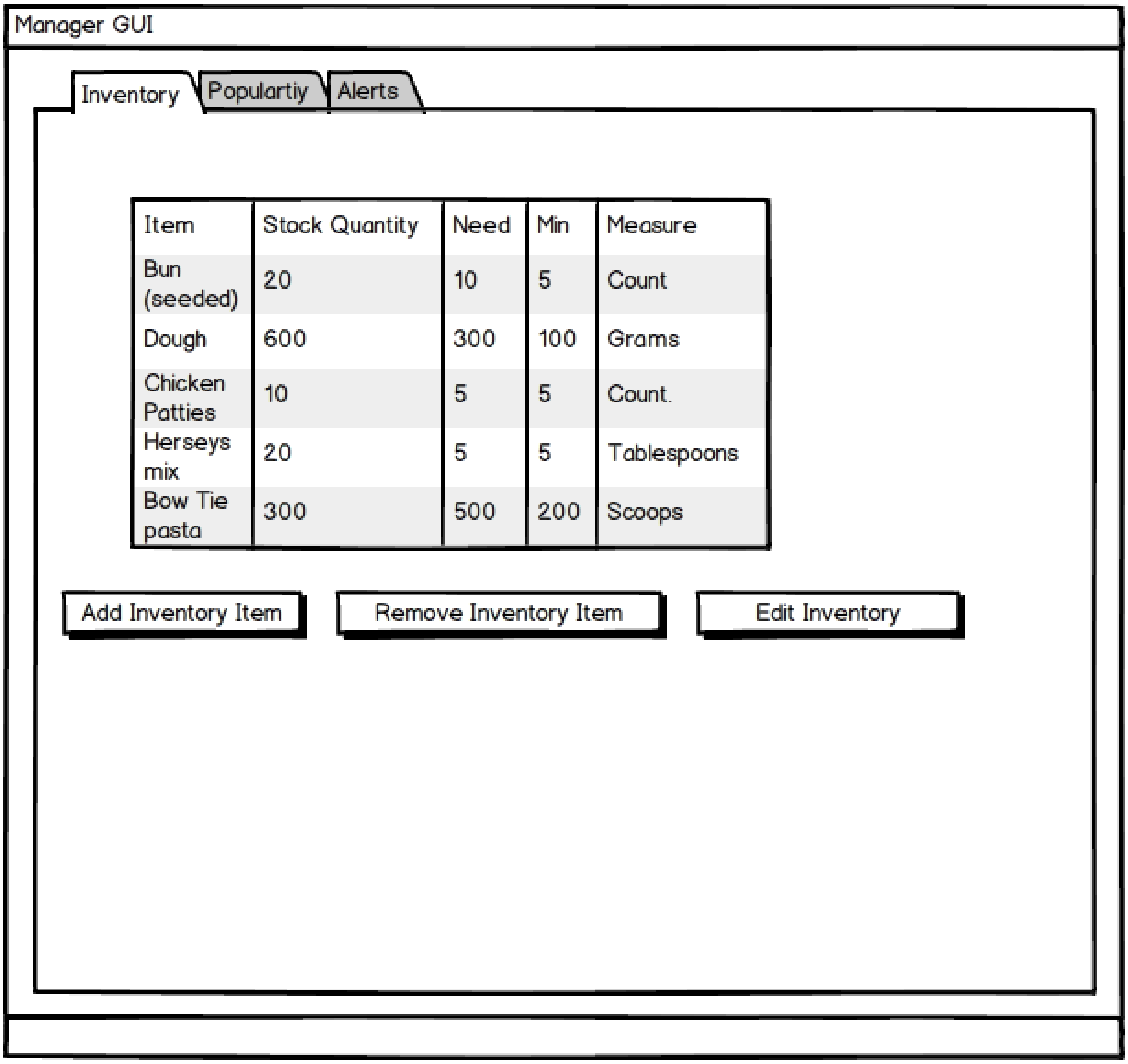


Figure 15: The chef can select which dishes are done being prepared by tapping on the dish and selecting the Dish Done. This will notify the waiter that the dish is ready to be delivered.



Item	Stock Quantity	Need	Min	Measure
Bun (seeded)	20	10	5	Count
Dough	600	300	100	Grams
Chicken Patties	10	5	5	Count.
Herseys mix	20	5	5	Tablespoons
Bow Tie pasta	300	500	200	Scoops

Figure 16: The manager will be able to view a list of the inventory item. The manager will also be able to add, remove, or edit the inventory item.

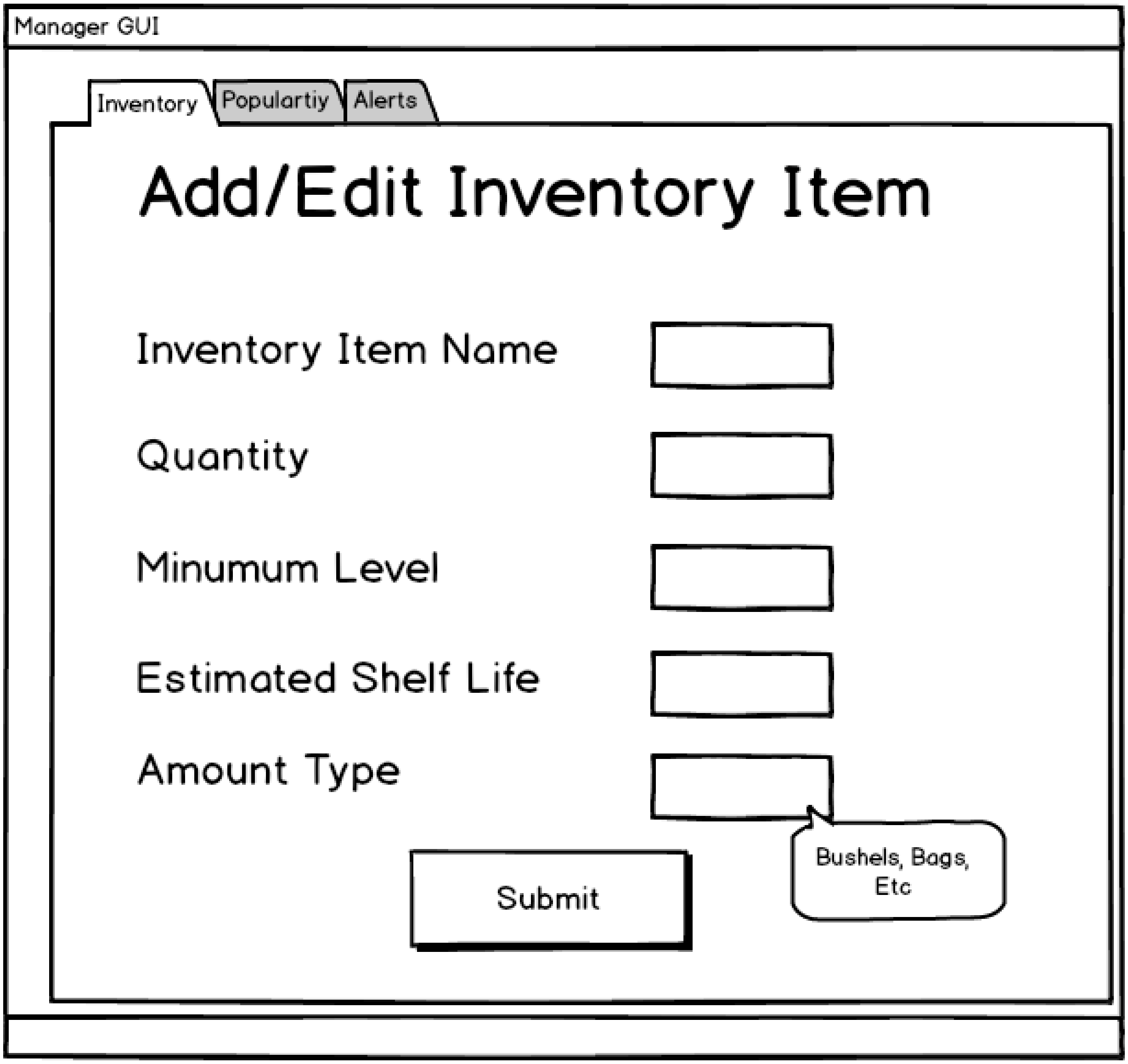
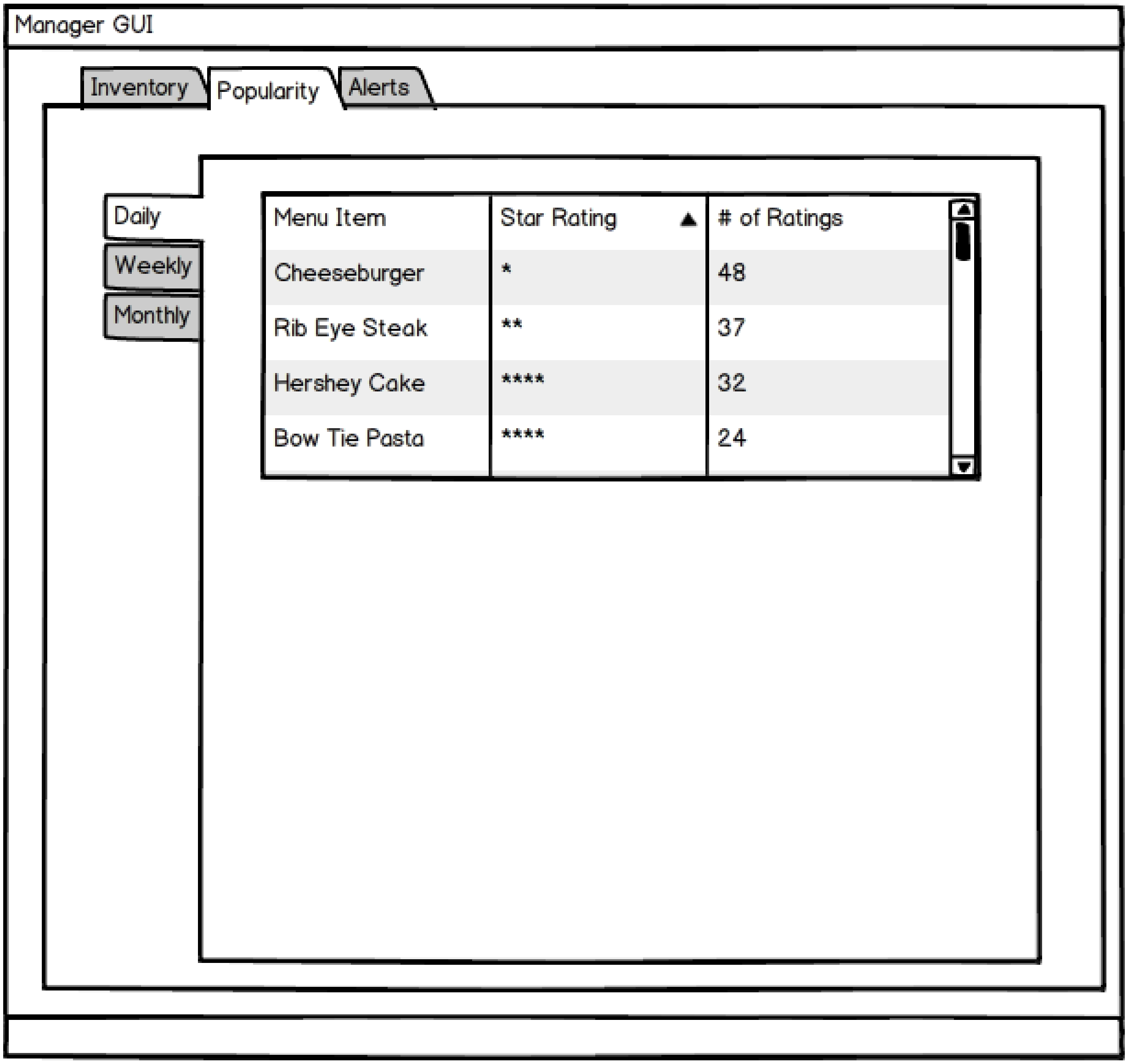
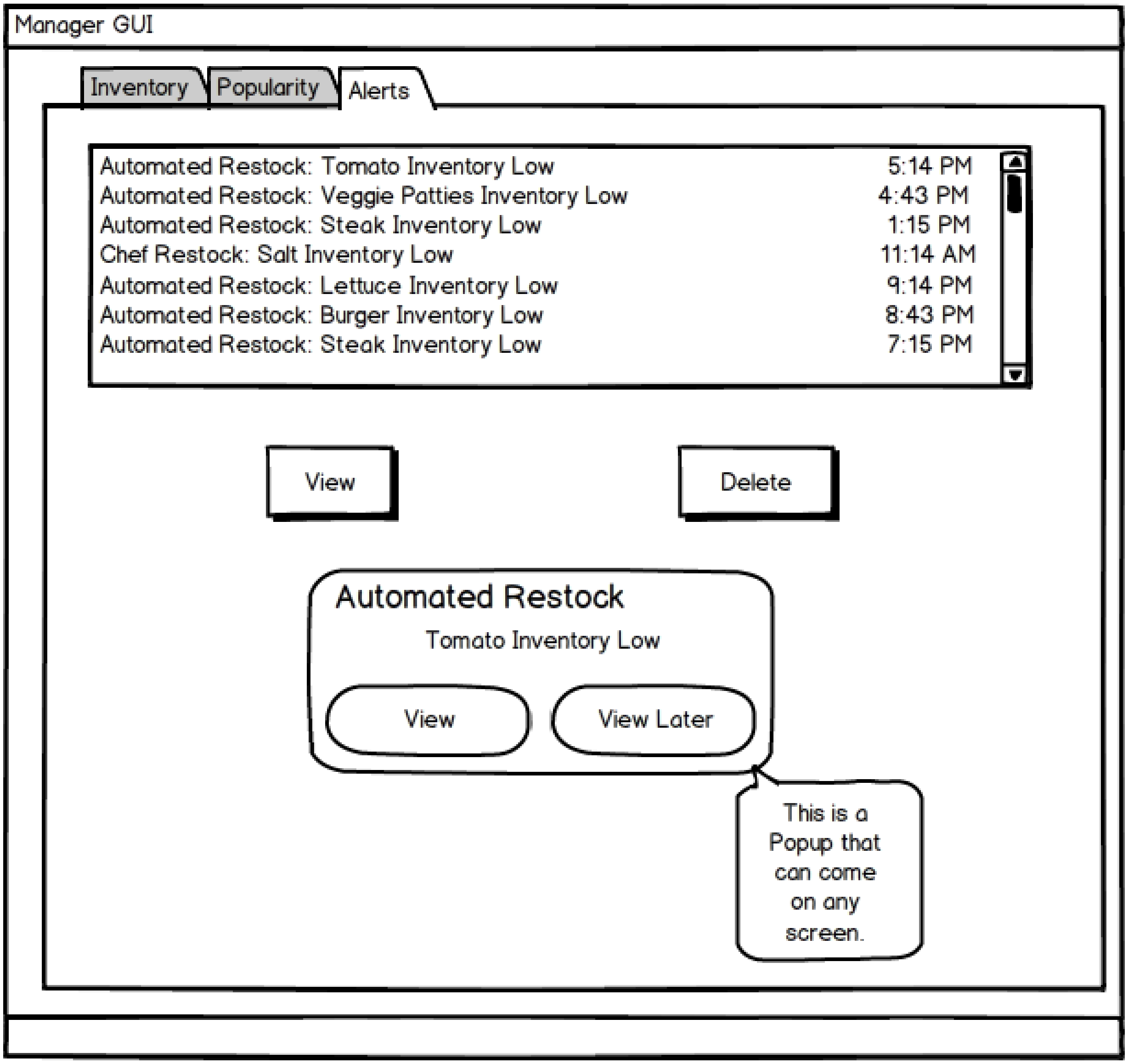


Figure 17: The manager will be able to add and edit inventory items by typing the inventory item name, quantity, etc.



Menu Item	Star Rating ▲	# of Ratings
Cheeseburger	*	48
Rib Eye Steak	**	37
Hershey Cake	****	32
Bow Tie Pasta	****	24

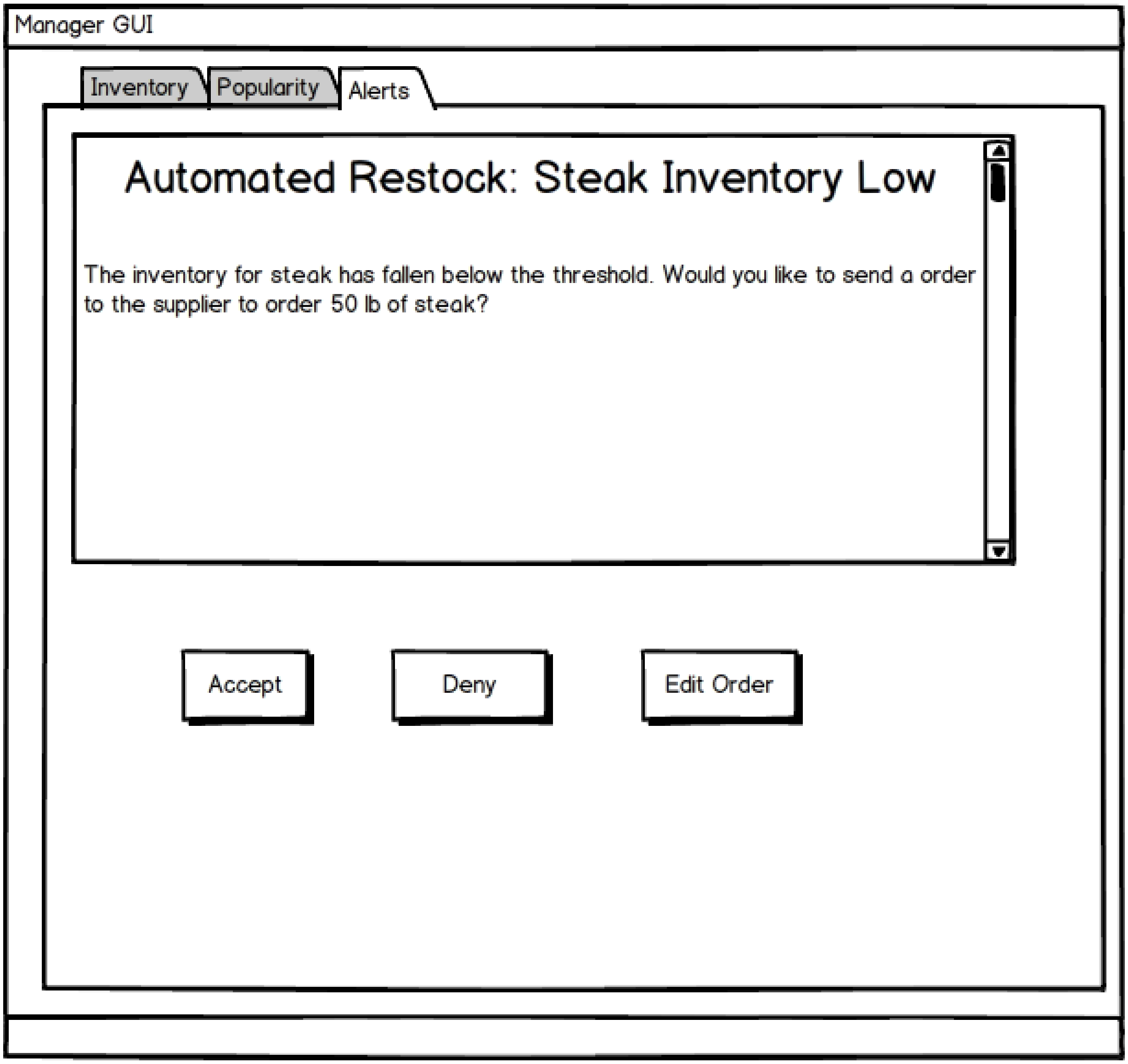
Figure 18: The manager will be able to view a list of popular menu items in the restaurant according to any time reference.



Automated Restock: Tomato Inventory Low	5:14 PM
Automated Restock: Veggie Patties Inventory Low	4:43 PM
Automated Restock: Steak Inventory Low	1:15 PM
Chef Restock: Salt Inventory Low	11:14 AM
Automated Restock: Lettuce Inventory Low	9:14 PM
Automated Restock: Burger Inventory Low	8:43 PM
Automated Restock: Steak Inventory Low	7:15 PM

This is a Popup that can come on any screen.

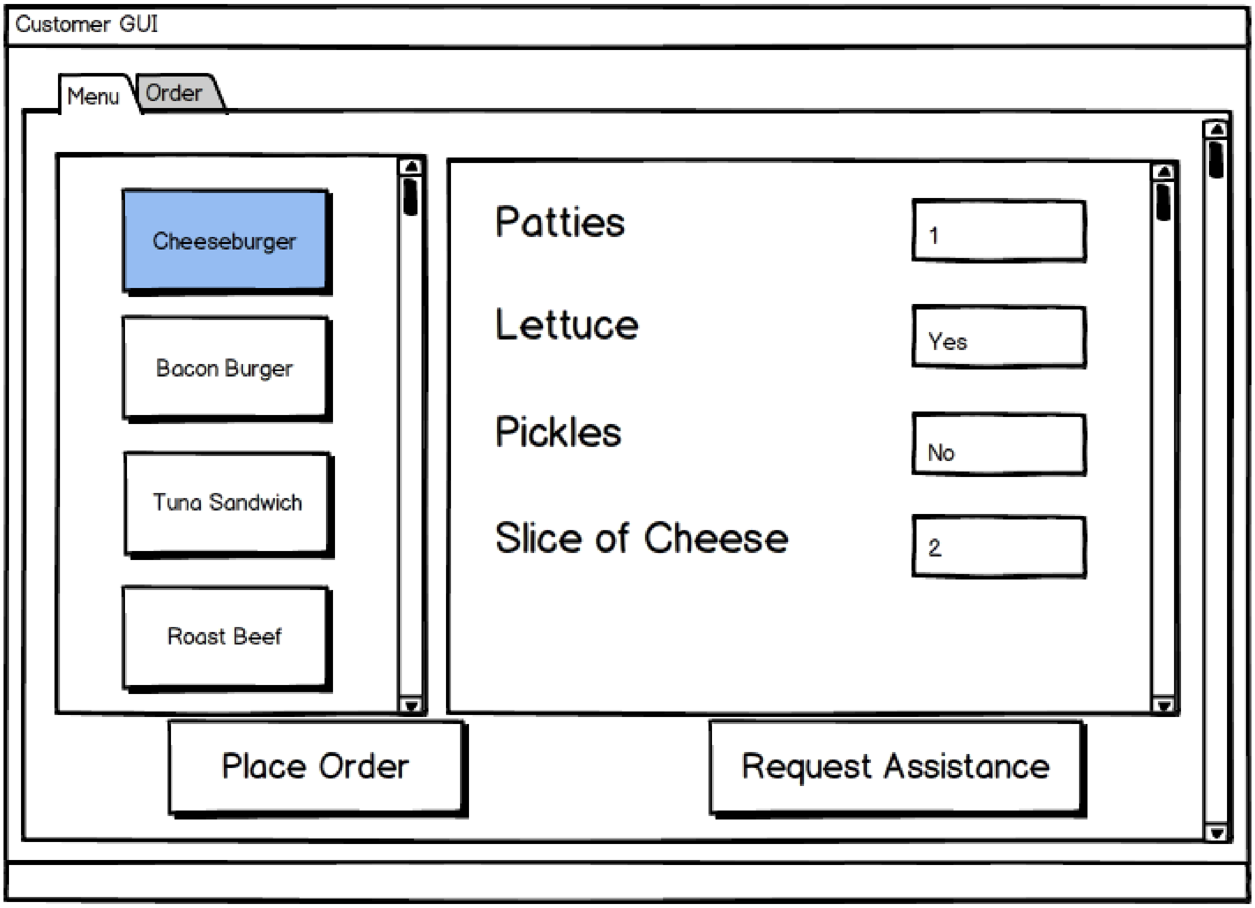
Figure 19: The manager will be able to view a list of all the automated alerts sent by the automated inventory system. The manager will also be alerted in the form of a pop up notification on any screen.



Automated Restock: Steak Inventory Low

The inventory for steak has fallen below the threshold. Would you like to send a order to the supplier to order 50 lb of steak?

Figure 20: The manager will be able to view the alerts and can approve, deny, or edit the request to restock.



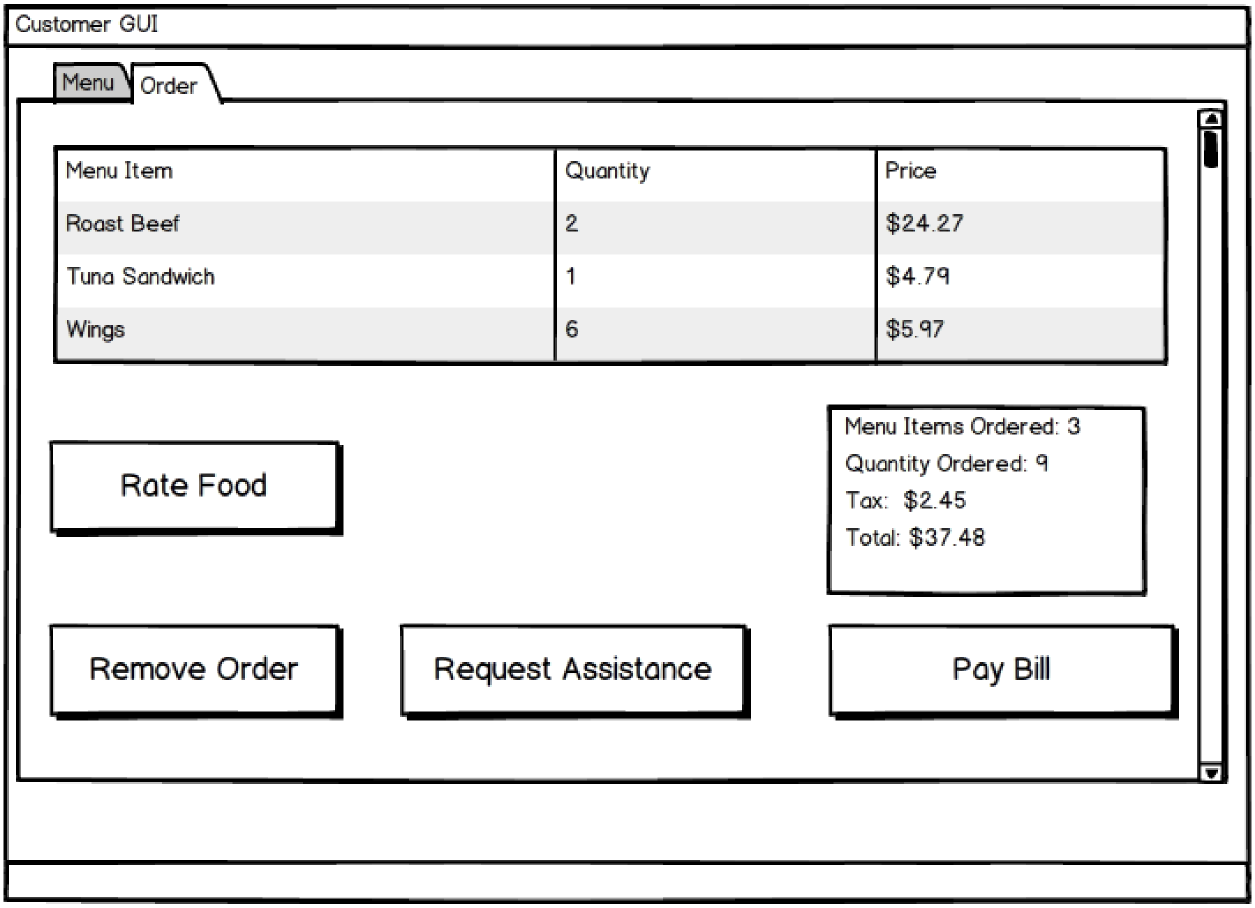
Patties 1

Lettuce Yes

Pickles No

Slice of Cheese 2

Figure 21: The customer will be able to select the menu item and edit the ingredients within it and be able to place the order. The customer will also be able to request assistance of the waiter if needed.



Menu Item	Quantity	Price
Roast Beef	2	$24.27
Tuna Sandwich	1	$4.79
Wings	6	$5.97

Menu Items Ordered: 3
Quantity Ordered: 9
Tax: $2.45
Total: $37.48

Figure 22: The customer will be able to view the menu item that were ordered along with the price and the total cost so far. The customer will also be able to rate the food, pay the bill, and request assistance by tapping on the appropriate button.

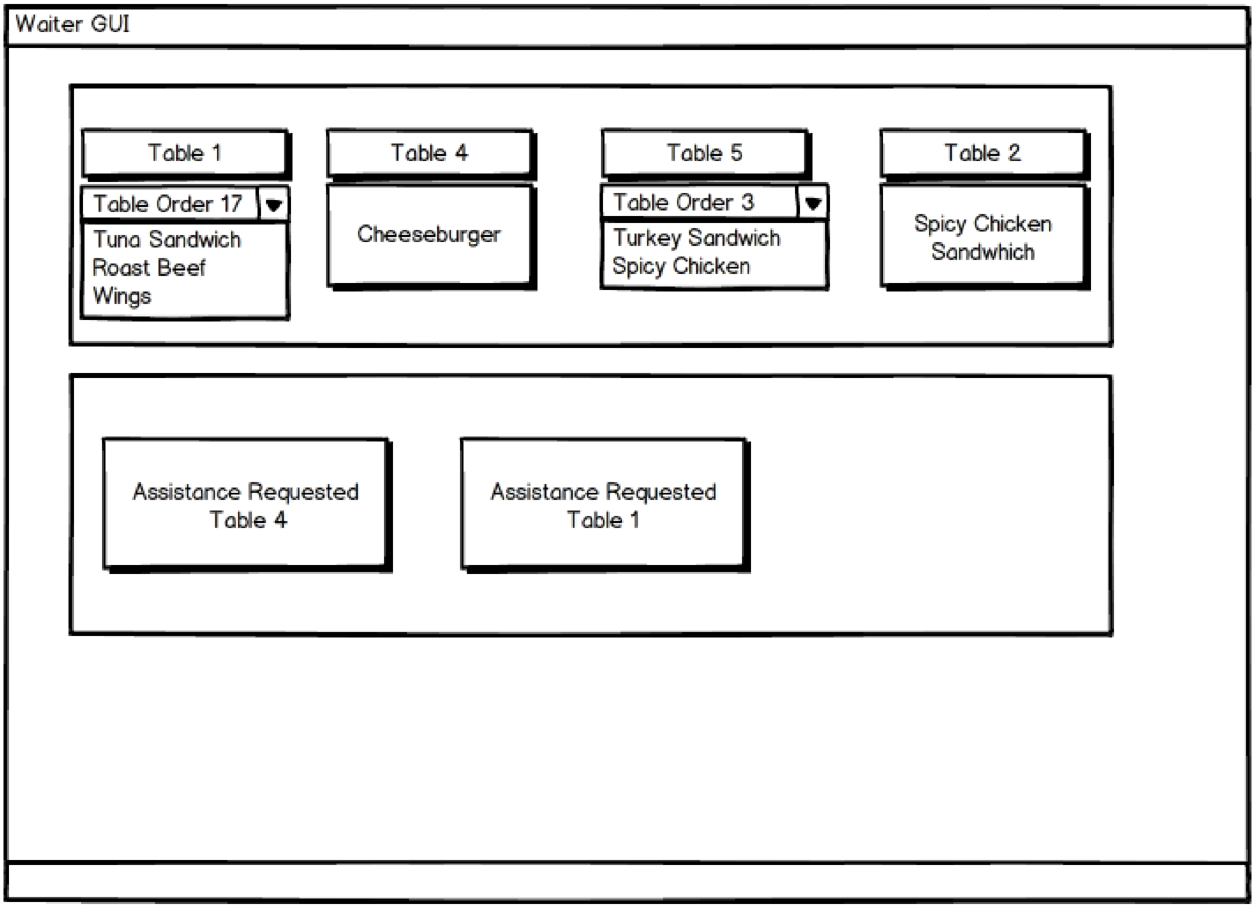


Figure 23: The waiter will be able to see which table requires assistance as well as the dishes that are in the ready queue.

2.6. User Effort Estimation

2.6.1 Scenario 1: Customer Wishes to Place Order

1. Navigation: Selects the Menu Item (2 Taps)
    A. Customer selects menu item with finger by tapping on it.
    B. After completing the Data Entry as shown below Click Place Order

2. Data Entry: Selects which Ingredients are wanted on the Menu Item (2 or more Tap)
    A. Tap the ingredients that you wish to adjust.
    B. Enter amount or yes/no to adjust number or remove ingredient (max 3 taps)

    C. Tap additional ingredients to add and then tap the add button.

2.6.2 Scenario 2: Chef Selecting Dish to Prepare

1. Navigation: Selects the Dish from the order queue to prepare (4 Taps)
    A. Chef taps on the dish that will be prepared and starts preparing it.
    B. After the dish is done being prepared the chef switches to "Active" Tab.
    C. Chef taps on the dish that is prepared and then Dish Done to notify waiter.

2.6.3 Scenario 3: Waiter selects order or menu item to deliver

1. Navigation: Selects the next order or menu item on ready queue(1 Tap)
    A. Waiter taps on the menu item and goes to kitchen to pick it up.
    B. Waiter picks up prepared menu item and delivers it to the appropriate table.

2.6.4 Scenario 4: Manager wishes to manually add inventory item

1. Navigation: Selects to Add inventory item in the inventory.
    A. Manager taps on the inventory tab.
    B. Manager Selects "Add Inventory Item"
2. Data Entry: Enters appropriate information in the fill in boxes.
    A. Manager enters the Inventory Item Name.
    B. Manager enters the Quantity.
    C. Manager enters minimum level (Threshold).
    D. Manager enters estimated shelf life (Expiration).
    E. Manager enters Amount Type.

3. Functional Requirements Specification

3.1 Stakeholders

End Users:

Restaurant Employees

These are the end users who hold the major interest in the system, as they expect to use it to simplify their life and improving the time efficiency of the restaurant.

Customers

The end user, who will also be using the system and have some interest in the system as it could possibly lessen their wait time and the need for constant interaction for the waiter.

Manager

The manager is essentially the administrator of the system and like the restaurant employees has a major stake in the success of the system as he is able to make his restaurant more efficient and easier to manage.

The Software Team

The software team is the group responsible for the design, implementation and manufacturing of the software and hold the highest interest in the success of the system. In this case, our group is the software team.

3.2 Actors and Goals:

Initiating Actors

Manager
The restaurant owner who is responsible for managing the system.

Chef
The chef is the one who cooks all the food.

Waiter
The waiter is the one who attends customers.

Customer
The customer of the restaurant who places orders.

Participating Actors

Timer
The timer responsible for keeping track of time (When making logs or billing) or clocking the time of an order.

Database
The storage system used to hold the data (usually local).

3.3 Use Cases

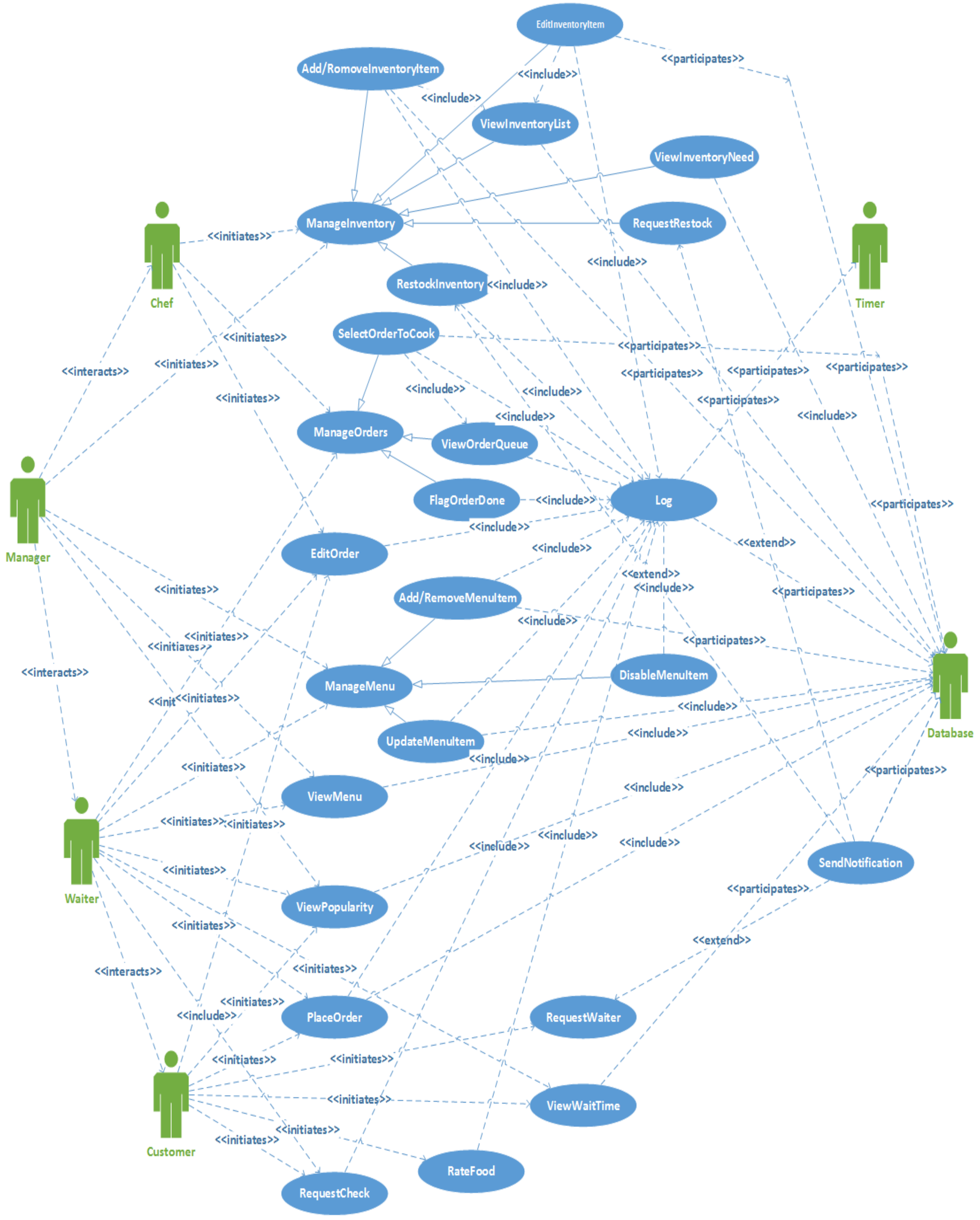
3.3.1 Casual Description

Use Case	Name	Description
UC - 1	ManageInventory	Allows the user to manage the inventory. To review more detail on sub use cases refer to UC - 1, 2, 3/4, 5, 6, 7, 27.
UC - 2	ViewInventoryList	Allows the user to view the list of inventoried items along with the each items estimated amount left. (sub use case for UC -1)
UC - 3/4	Add/RemoveInventory Item	Allows the user to add/remove Inventory Item. When item is being added user also has to add specific information concerning the added inventory item which includes, current amount, and quantity measure(what it is measured in), and the shelf life. The updated information is stored as a log(see UC- 20). (sub use case for UC -1)
UC - 5	ViewInventoryNeed	Allows the user to view food trend specific for the past 4 weeks and a list of inventoried Items needed for the coming 7 days. Optional Implementation: system makes use of RequestRestock(UC -6) to automatically make the requests to order inventory needed for the next 7 days. (sub use case for UC -1)
UC - 6	RequestRestock	Allows the system to send a notification (see UC - 26) to the manager to approve a Inventory restock.The updated information is stored as a log(see UC- 20). (sub use case for UC -1)
UC - 7	RestockInventory	Allows user to either send an order notification for the inventory item to the specified supplier or update information on it manually(see UC - 27).(using sendNotification, reference UC - 26) .A log is created for the information on the order or action. (using log, reference UC - 20)(sub use case for UC -1)

UC - 8	ManageMenu	Allows user to manage the menu items on the menu. To review more detail on sub use cases refer to UC - 9/10, 11, 12 .
UC - 9/10	Add/RemoveMenuItem	Allows the user to Add/Remove menu items on the menu. When the chef adds the menu item, he is responsible for adding the information about the menu item, which includes: Inventory items (need to make the menu item) along with quantity of each items for the menu item along with the estimated cook/ready time, and the freshness time. The updated information is stored as a log(see UC- 20). (sub use case for UC -8)
UC - 11	UpdateMenuItem	Allows the user to update information on menu items.(reference UC - 9/10 for description on menu item information) The updated information is stored as a log(see UC- 20). (sub use case for UC -8)
UC - 12	DisableMenuItem	Allows the user to disable menu items that he cannot cook or is unable to and updates the menu information. The updated information is stored as a log(see UC- 20). (sub use case for UC -8)
UC - 13	ViewMenu	Allows the user to view menu containing list of menu items and information on each menu item. (reference UC - 9/10 for description on menu item information)
UC - 14	ManageOrders	Allows the user to manage the Orders that are on the order queue. To review more detail on sub use cases refer to UC - 15, 16, 17.
UC - 15	ViewOrderQueue	Allows the user to view the order queue so that he may select an order to cook/make. (sub use case for UC -14)
UC - 16	SelectOrderToCook	Allows the user to choose an order from the order queue that matches his specialty/skills and removes it from the order queue. The updated information is stored as a log(see UC- 20). (sub use case for UC -14)
UC - 17	FlagOrderDone	Allows the user to flag the menuItem as ready when the menu item is ready to be served and notifies the waiter. The updated information is stored as a log(see UC- 20). (sub use case for UC -14)

UC - 18	PlaceOrder	Allows the user to place orders on the menu items that they want to eat. The order information is stored as a log(see UC- 20).
UC - 19	EditOrder	Allows the user to remove an order or edit its information. The updated information is stored as a log(see UC- 20).
UC - 20	Log	Allows the user/system to log the information of any changes onto the database along with a timestamp.
UC - 21	ViewWaitTime	Allows the user to view the wait time of the menu Items they have ordered or approximate arrival time for a selected a menu item.
UC - 22	RequestWaiter	Allows the user to request the waiter to tend to his or her table. This is done using SendNotification(UC-26). This action is stored as a log(see UC - 20).
UC - 23	RequestCheck	Allows user to request the check after they have finished eating. This is done using SendNotification(UC-26). This action is stored as a log(see UC - 20).
UC - 24	RateFood	Allows user to rate menu items that they have eaten. This action is stored as a log(see UC - 20).
UC - 25	ViewPopularity	Allows the user to view popularity of menu items.
UC - 26	SendNotification	Allows the user/system to send either email notification, with a predefined message body and recipient specific to the type of email, or a notification to their GUI. This action is stored as a log(see UC - 20).
UC - 27	EditInventory	Allows the user to edit information on inventoried item(see UC - 3/4 for more information on editable information). This action is stored as a log(see UC - 20). (sub use case of UC-1).

3.3.2 Use Case Diagrams



3.3.3 Traceability Matrix

	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22	UC23	UC24	UC25	UC26	UC27
REQ - 1	5	X	X	X	X			X													X							X
REQ - 2	3	X	X																									
REQ - 3	3								X					X														
REQ - 4	4	X					X																					
REQ - 5	2	X				X	X																					
REQ - 6	3	X				X	X	X																				
REQ - 7	2	X					X																					
REQ - 8	5	X		X	X	X																						X
REQ - 9	3																		X	X	X							
REQ - 10	5														X		X	X			X							
REQ - 11	2														X	X												

REQ - 12	5																				X							
REQ - 13	5																		X		X							
REQ - 14	3																								X	X		
REQ - 15	5								X																			
REQ - 16	3								X	X	X	X																
REQ - 17	5								X					X														
REQ - 18	4								X	X	X		X															
REQ - 19	5																											
REQ - 20	3																											
REQ - 21	2	X	X			X																						
REQ - 22	3														X			X										
REQ - 23	4														X		X											
REQ - 24	5																		X	X								
REQ - 25	5	X	X																									
REQ - 26	5																									X		

REQ - 27	3																											
REQ - 28	4																					X	X			X	
MAX PW		5	5	5	5	5	4	5	5	4	4	3	4	5	5	2	5	5	5	5	5	5	4	4	3	5	4	5
TOTAL PW		31	15	10	10	12	11	8	20	7	7	3	4	8	14	2	9	8	13	8	16	5	4	4	3	8	4	10

UC 1 > UC 8 > UC 20 > UC 2 > UC 14 > UC 18 > UC 5 > UC 6 > UC 3 = UC 4 = UC 27 > UC 16 > UC 7 = UC 13 = UC 17 = UC 19 > UC = 9 = UC 10 > UC 21 > UC 12 = UC 22 = UC 23 = UC 26 > UC 11 = UC 24 > UC 15

Therefore, we elaborate UC 1, 8, 14, 18(We chose to Ignore UC 2 it is a sub use cases of 1 and will be elaborated in it) as they have the highest priority as they not only enclose most of the sub - use cases while also involving most of the functionality presented for the restaurant. These use cases, ManageInventory, ManageMenu and ManageOrders and PlaceOrders, are the backbone features of our system and so have the highest priority as they take and make up the most significant amount of our system requirements. Although UC-20 (Log) has a high priority weight, we will not be describing this in our fully-dressed or considering it of importance as it is a trivial aspect of our system and only aids in keeping track of the ever-changing database. Also, as this functionality has no influence or interaction with the user, a back-end feature aptly put, but rather a sub use case, or a system use case to be more descriptive, we do not wish to describe it in detail. However, we chose to focus on UC – 21, ViewWaitTime, as it represents one of our systems most essential features that we feel not only is an frontal feature, one to market our product, but also an elaborate use-case that requires description and very involved with the user.

3.3.4 Fully Dressed Description and System Sequence Diagrams:

From now on we will focus on these five use cases and elaborate them since they pertain to our ideas.

Use Case UC - 1: ManageInventory
Use Case UC - 8: ManageMenu
Use Case UC - 14: ManageOrders
Use Case UC - 18: PlaceOrder
Use Case UC - 21: ViewWaitTime

Other use cases may be included as sub-use-cases in these five, or will be elaborated at a future time.

Use Case UC - 1: ManageInventory

Initiating Actor: Manager, Chef
Actor's Goal: To manipulate the information on the current inventoried items.
Participating Actor: Database, Timer
Related Requirements: REQ - 1, 25
Sub - Use Cases: UC - 2, 3/4, 5, 6, 7, 27
Precondition: The user is either a chef or manager or someone who has privilege to view inventory. The database is up and functional. For UC - 6,7, the send notification (UC - 26) is functioning.
Postcondition: The desired manipulation has been made to the database and a log has been stored of the action that took place.

Flow of Events:
1. -> **User** selects the Inventory Management Option
2. <- **System** shows an interface that displays selectable options which include: View Inventory, Add/Remove Inventory, View Inventory Need,Request Restock, Restock Inventory
3. User either

a. -> Selects View Inventory (UC - 2)
1. <- **System** either
a. displays a list of inventory items from the database and goes to 6
b. is unable to contact database and goes to alt: 4

b. -> Selects Add/Remove Inventory Item (UC - 3/4)
1. <- **System** displays an option of either add or remove

User either

a. -> Selects Add option (UC - 3)

1. <- **System** displays a list of information required to be filled(view UC-3/4 for this list)
2. -> **User** fills in all the information and selects done after he completes it.
3. <- **System** either

a. enters new Inventory Item into the database and updates the database with the user filled information and and goes to 4

b. is unable to contact database and goes to alt: 4

b. -> Selects Remove option (UC - 4)

a. do 3. a. 1.
b. -> **User** selects the inventory item he wants to remove
c. -> **System** requests confirmation of the removal of the inventory item
d. -> **User** confirms action
e. System either

a. <- removes new Inventory Item from the database and goes to 4

b.<- is unable to contact database and goes to alt: 4

c. -> Selects View Inventory Need (UC - 5)

2. -> **System** either

a. Shows the Inventoried Item usage of the last 4 weeks along with the inventoried items and amount of each inventoried items needed for the next 7 days.

b. is unable to contact database and goes to alt: 4

d. -> Selects Request Restock (UC - 6)

1. do 3. a. 1.
2. -> **user** selects the Inventory Item to restock
3. <- **System** uses SendNotification with information of the inventoried item and a predefined message.
4. include:: Send Notification (UC - 26) (may be unable to send message and goes to Alt: 4)
5. go to 4.

e. -> Selects Restock Inventory (UC - 7)

1. -> **User** either

a. selects to send an order request

1. ->**System** either
    - a. if User come from an Request Inventory Alert selects the Inventoried Item on the alert
    - b. does 3. a. 1.
        - **1.** -> **User** selects the inventory item to order
2. <- **System** queries how much to order

3. -> **User** selects amount to order

4. <- **System** uses SendNotification with information on amount of inventoried item to order and a predefined message.

5. include:: Send Notification (UC - 26) (may be unable to send message and goes to Alt: 4)

6. go to 4.

b. selects to update inventory manually

1. ->**System** either

a. if User come from an Request Inventory Alert selects the Inventoried Item on the alert

b. does 3. a. 1.

1. -> **User** selects the inventory item to update

2. <- **System** queries how much to update

3. -> **User** selects amount

4. <- **System** either

a. <- removes the Inventory Item from the database and goes to 4

b.<- is unable to contact database and goes to alt: 4

f. -> Selects Edit Inventory Item (UC - 27)

1. do 3. a 1.

2. -> **User** selects Inventory Item to edit

3. <- **System** either

a. displays list of information of the select Inventory Item from the database

b. is unable to contact database and goes to alt: 4

4.-> **User** updates information and selects done.

5. <- **System** either

a. updates the database with the user filled information and and goes to 4

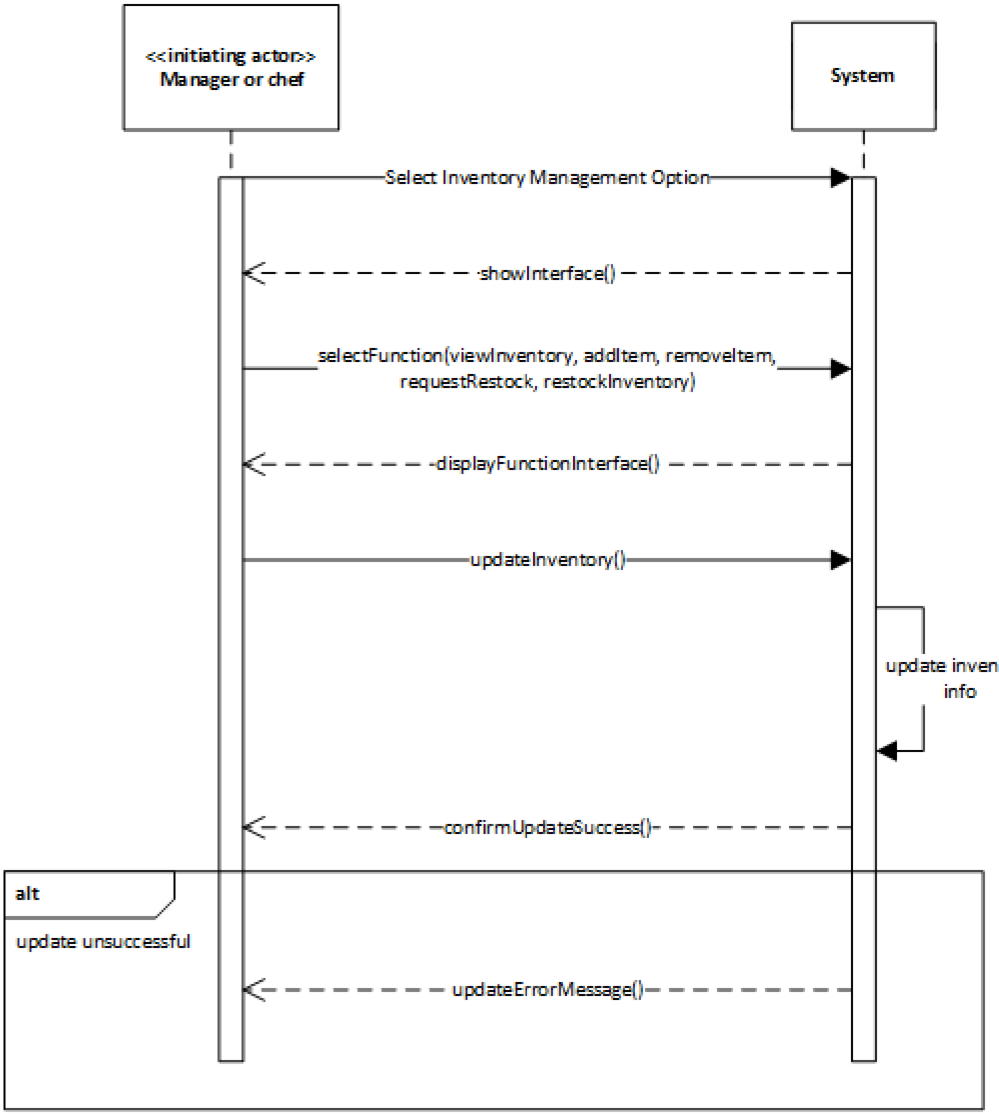
b. is unable to contact database and goes to alt: 4

4. <- **System** confirms with a success message.

5. Include::log (UC - 20) (Makes a log of any updated information)
6. <end>
Alt:
4. <- **System** returns an error message
5. Include::log (UC - 20)
6. <end>

Flow of Events for Main Success Scenario:
any flow of events that pass through all the way to 6 and not through the Alt sequences are success scenarios.
Flow of Events for Fail Scenario:
Any flow of events that lead up to Alt sequences 4-6 will be our fail scenarios.

Sequence Diagram:



Use Case UC - 8: MangeMenu

Initiating Actor: Manager, Chef, Waiter
Actor's Goal: To manipulate the information on the Menu which consists of information on each menu item.
Participating Actor: Database, Timer
Related Requirements: REQ - 3, 15,16,17,18
Sub - Use Cases: UC - 9/10, 11,12
Precondition: The user is either a chef or waiter or someone who has privilege to view inventory. The database is up and function
Postcondition: The desired manipulation has been made to the database and a log has been stored of the action that took place.
Flow of Events:

1. -> **User** selects the Menu Management Option
2. <- **System** shows an interface that displays selectable options which include: Add/Remove Menu Item, Update Menu Item, Disable Menu Item
3. User either
    a. -> Selects Add/Remove Menu Item (UC - 9/10)
    1. <- **System** displays an option of either add or remove
2. **User** either
a. -> Selects Add option (UC - 9)
    a. <- **System** displays a list of information required to be filled(view UC-9/10 for this list)
    b. -> **User** fills in all the information and selects done after he completes it.
    c. <- **System** either
a. enters new Menu Item into the database and updates the database with the user filled information and and goes to 4
b. is unable to contact database and goes to alt: 4

b. -> Selects Remove option (UC - 10)
    a. include::View Menu (UC - 13)
    b. -> **User** selects the Menu item he wants to remove
    c. -> **System** requests confirmation of the removal of the Menu item
    d. -> **User** confirms action
    e. System either
a. <- removes the Menu Item from the database and goes to 4
b.<- is unable to contact database and goes to alt: 4

b. -> Selects Update Menu Item (UC - 11)

1. include::View Menu (UC - 13)
   2. -> **User** selects Inventory Item to edit
   3. <- **System** either
      a. displays list of information of the selected Menu Item from the database

b. is unable to contact database and goes to alt: 4

   4.-> **User** updates information and selects done.
   5. <- **System** either

a. updates the database with the user filled information and and goes to 4

b. is unable to contact database and goes to alt: 4

c. -> Selects Disable Menu Item (UC - 12)

1. include::View Menu (UC - 13)
2. -> **User** selects the Menu item he wants to disable
3. -> **System** requests confirmation for disabling of the Menu item
4. -> **User** confirms action
5. **System** either

a. <- disables the Menu Item and flags it disables in the database and goes to 4

b.<- is unable to contact database and goes to alt: 4

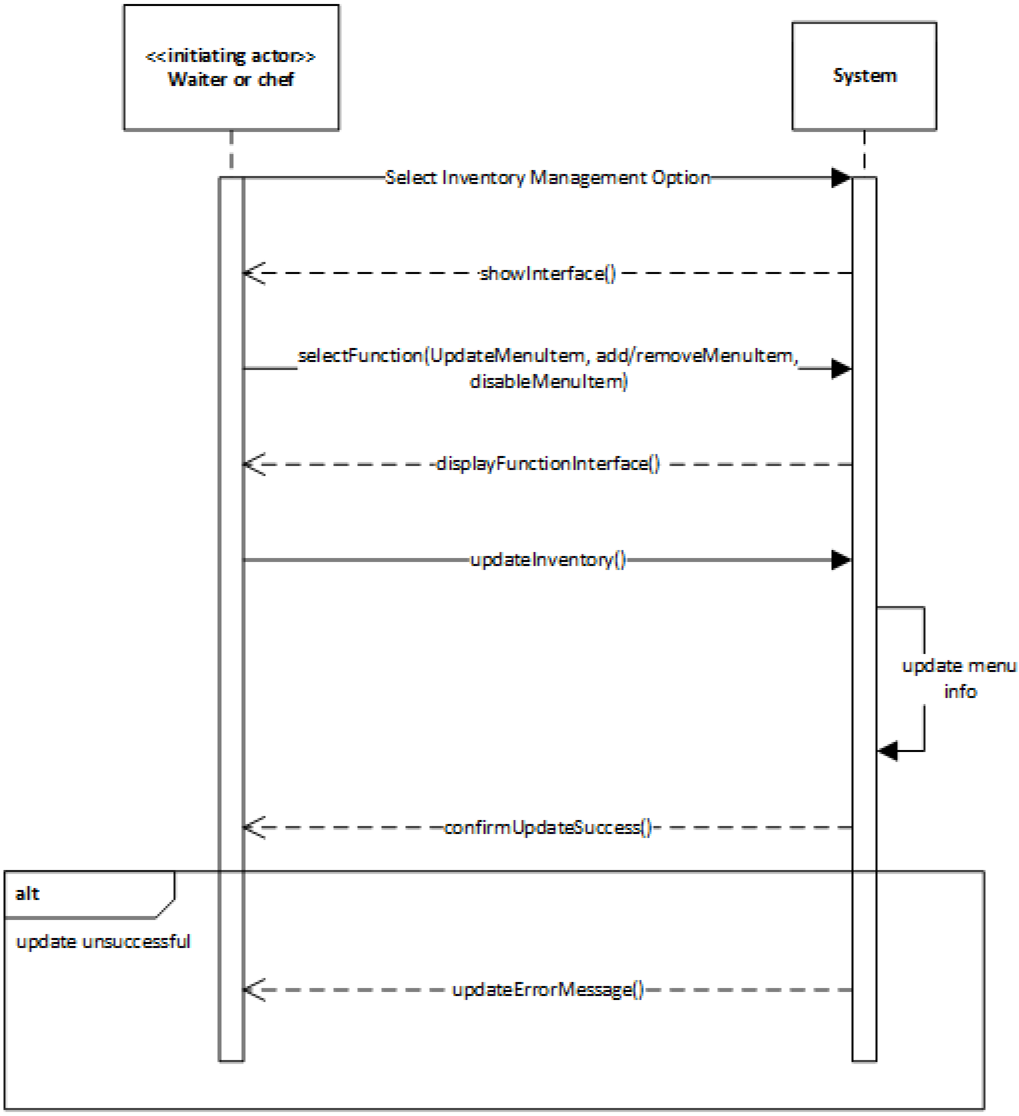
4. <- **System** confirms with a success message.
5. Include::log (UC - 20) (Makes a log of any updated information)
6. <end>

Alt:
4. <- **System** returns an error message
5. Include::log (UC - 20)
6. <end>

Flow of Events for Main Success Scenario:
Any flow of events that pass through all the way to 6 and not through the Alt sequences are success scenarios.
Flow of Events for Fail Scenario:

Any flow of events that led up to Alt sequences 4-6 will be our fail scenarios.

Sequence Diagram:



Use Case UC - 14: MangeOrders

Initiating Actor: Chef, Waiter
Actor's Goal: To manipulate information on the orders placed
Participating Actor: Database, Timer

Related Requirements: REQ - 10
Sub - Use Cases: UC - 15, 16, 17
Precondition: The user is either a chef or waiter or someone who has privilege to view inventory.The database is up and running.
Postcondition: The desired manipulation has been made to the database and a log has been stored of the action that took place.
Flow of Events
1. -> **User** selects the Menu Management Option
2. <- **System** displays interface for managing customer orders with options to view the order
   queue, select an order to cook, and flag an order as done.
3. -> **User** chooses to either
   a. select view order queue (UC -15)

1. <- **System** either
   a. displays order queue and wait queue from the database and goes to 6
b. is unable to contact database and goes to alt: 4

   b. select an order to cook (UC - 16)
      1. do 3. a. 1.
      2. ->**User** select order to cook.

3. **System** either
a. <- removes the Menu Item/Items from the Order Queue and puts them
   on wait queue and updates the database and goes to 4
b. <- is unable to contact database and goes to alt: 4

   c. flag an order as done (UC -17)
      1. do 3. a. 1.
      2. ->**User** select order to flag as done.
      3. **System** either

a. <- removes the Menu Item/Items from the wait Queue and puts them
   on the ready queue and updates the database and goes to 4
b. <- is unable to contact database and goes to alt: 4

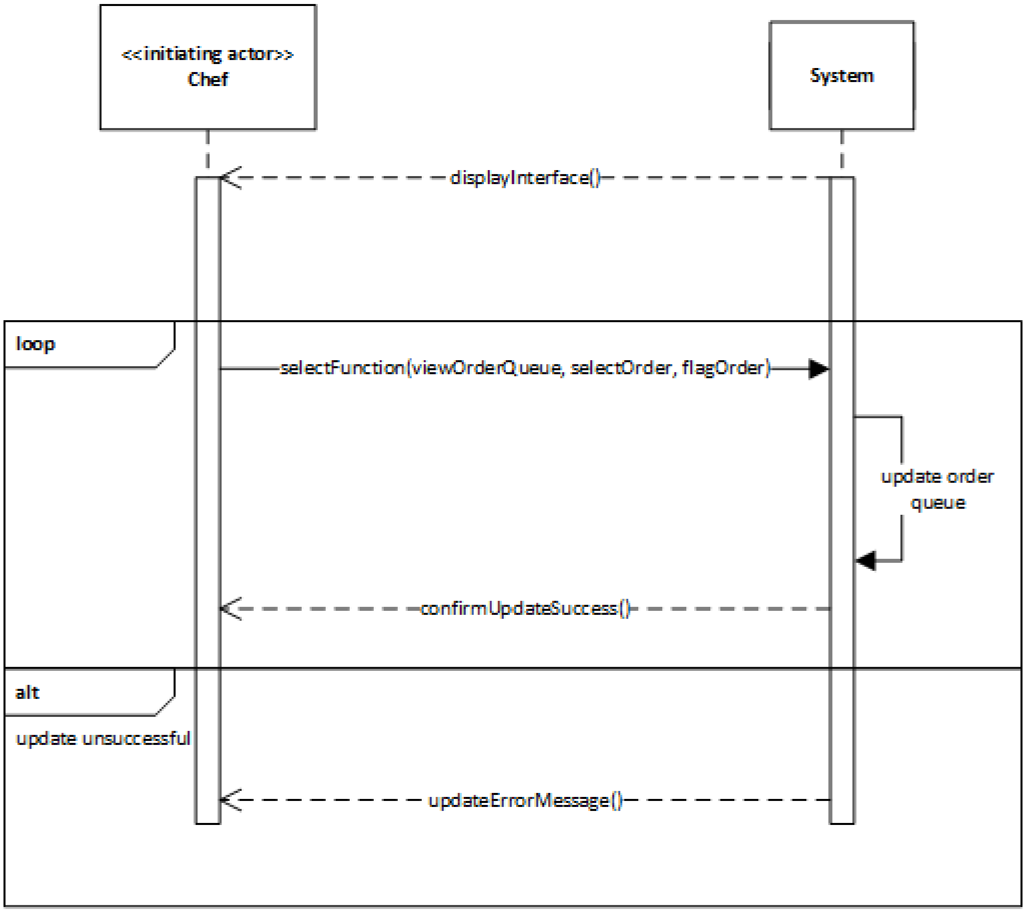
4. <- **System** confirms with a success message.
5. Include::log (UC - 20) (Makes a log of any updated information)
6. <end>

Alt:

4. <- **System** returns an error message
5. Include::log (UC - 20)
6. <end>

Flow of Events for Main Success Scenario:
any flow of events that pass through all the way to 6 and not through the Alt sequences are success scenarios.
Flow of Events for Fail Scenario:
Any flow of events that lead up to Alt sequences 4-6 will be our fail scenarios.

Sequence Diagram:



Use Case UC - 18: PlaceOrder

Initiating Actor: Waiter, Customer
Actor's Goal: To place an order of menu item/items.
Participating Actor: Database, Timer
Related Requirements: REQ - 13, 24

Precondition: The user is either a or waiter or someone who has privilege to view inventory. The database is up and running. The selected menu item/items are not disabled or unable to be cooked.
Postcondition: The selected menu item/items are put on the order queue.
Flow of Events

1. include::ViewMenu(UC - 13)
2. -> **User** selects menu item/items and selects done.
3. <- **System** queries it user wants to make delivery of orders together or individually.
4. -> **User** selects option.
5. **System** either

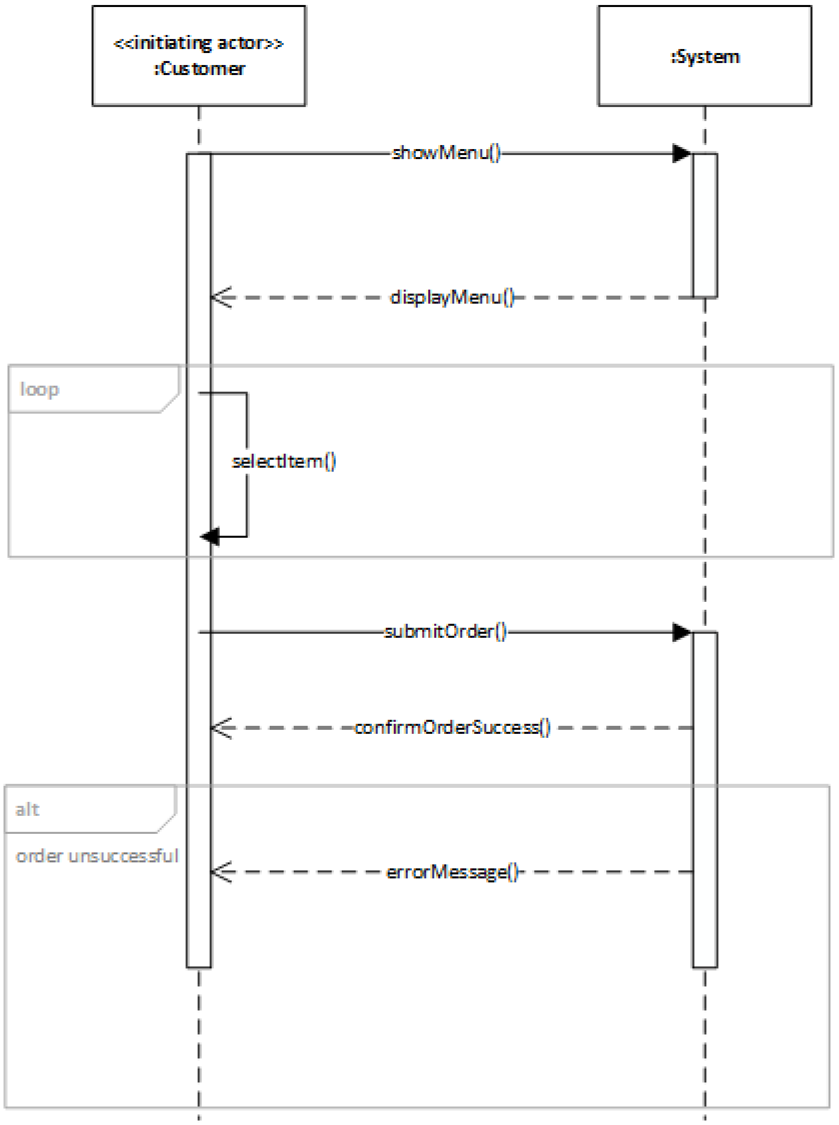
a. <- puts the menu item/items selected on the order queue and updates the database and goes to 6
b. <- is unable to put menu item/items on the order queue as menu item/items has been either disable or is unable to be queued due to low amount of inventory item needed for the the menu item/items and goes to alt: 6
c. <- is unable to contact database and goes to alt: 6

6. <- **System** confirms with a success message.
7. Include::log (UC - 20) (Makes a log of any updated information)
8. <end>

Alt:
6. <- **System** returns an error message
7. Include::log (UC - 20)
8. <end>

Flow of Events for Main Success Scenario:
any flow of events that pass through all the way to 8 and not through the Alt sequences are success scenarios.
Flow of Events for Fail Scenario:
Any flow of events that lead up to Alt sequences 6-8 will be our fail scenarios.

Sequence Diagram:



Use Case UC - 21: ViewWaitTime

Initiating Actor: Waiter, Customer
Actor's Goal: To view the estimated wait time of either an orders placed (either entire order or specific menu items of the order) or the estimated wait time of a menu item on the menu.
Participating Actor: Database, Timer
Related Requirements: REQ - 12
Precondition: The database is up and running.
Postcondition: The display shows the wait time of the selected item
Flow of Events :

2. -> **User** selects view wait time and chooses to either
    - a. View Wait Time of Order
        - 1. -> **System** either
            - a. shows wait time of item with the most wait time as the time of order if customer chooses to have all menu items on the order to be delivered together.
            - b. shows the wait time of each menu item ordered, if the customer chose to have the menu items order to be delivered individually.

    - b. View Wait Time of Selected Menu Item
        - 1. -> **System** displays information on select menu item if it was placed on the order queue and references the database to do so and goes to 3
    - 2. <- is unable to put menu item/items on the order queue as menu item/items has been either disable or is unable to be queued due to low amount of inventory item needed for the the menu item/items and goes to alt: 3

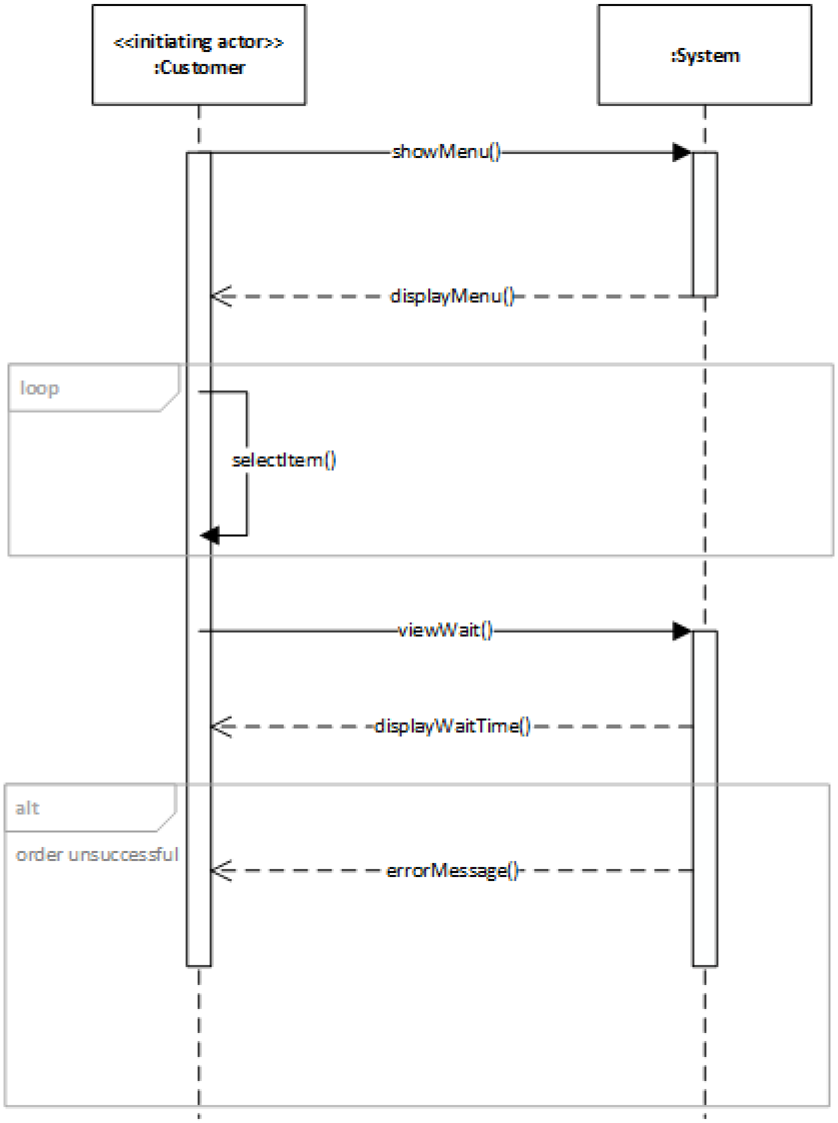
3. <- is unable to contact database and goes to alt: 3

3. <- **System** confirms with a success message.
4. Include::log (UC - 20) (Makes a log of any updated information)
5. <end>

Alt:
3. <- **System** returns an error message
4. Include::log (UC - 20)
5. <end>

Flow of Events for Main Success Scenario:
any flow of events that pass through all the way to 3 and not through the Alt sequences are success scenarios.
Flow of Events for Fail Scenario:
Any flow of events that lead up to Alt sequences 3-5 will be our fail scenarios.

Sequence Diagram:



4. Domain Analysis

4.1.1 Concept Definition

Requirement	Concept	Responsibility	Type
R1	OrderTaker	Takes the diner's order and sends it to the kitchen. Also capable of updating/editing orders after they are made.	D
R2	Menu	Displays restaurant menu to diner	K
R3	AssistButton	Allows diner to request aid and notifies waiters of the request	D
R4	FoodTimeDisplay	Allows diner to view time until their food arrives	K
R5	ReadyOrderQueue	Contains knowledge of what orders are done waiting and are ready to be delivered	K
R6	TableStatusView	Displays to the waiter the status of each table in the restaurant: occupied, ready, or needs attention	K
R7	DishQueue	Contains knowledge of dishes that must be made	K
R8	DishCompleteNotifier	Notifies the system that the current dish is complete	D
R9	InventoryDatabase	Contains the total list of ingredients stored in the inventory	K
R10	InventoryNotifier	Notifies manager of the predicted date when an ingredient will run out, when stock for an item falls below a predefined threshold, when an item goes bad, and when restock orders need to be made	D

R11	InventoryChanger	Allows restaurant staff to make changes to the inventory	D
R12	MenuChanger	Allows restaurant staff member to make changes to the menu	D
R13	Logger	Logs order information and restaurant operational information	D

4.1.2 Attribute definitions

Concept	Attributes	Attribute Description
Dish	Price	The price of the dish
	Ingredients	The ingredients that make up the dish
	Popularity	The popularity of the dish
Ingredient	Price	The price of the ingredient
	Amount	The amount of the ingredient currently present in the inventory
	Type	The type of ingredient: vegetable, meat, dairy, grains, fruits, spices, oils, and other. The other type is a catch-all for items that do not fit into the regular categories.
	EstimatedDepletionTime	The estimated time, based on our depletion prediction algorithm, until the ingredient is depleted
	EstimatedExpirationTime	The estimated time, based on our expiration prediction algorithm, until the ingredient expires
OrderTaker	TableNumber	The number of the table where the order was placed

	Dish	The dish that the customer had ordered
Menu	Dish	A dish that is currently on the menu
AssistButton	TableNumber	The number of the table where assistance was requested
FoodTimeDisplay	TableNumber	The number of the table where the request for the arrival time was placed
	Dish	The dish the customer requested the arrival time for
	Time	The time until the requested dish arrives
ReadyOrderQueue	TableNumber	The number of the table where the dish is to be delivered to
	Dish	The dish to be delivered
TableStatusView	TableNumber	The number of the table displaying its status
	TableStatus	The current status of the table: occupied, ready, or needs attention
EditOrder	TableNumber	The table number where the updated dish is to be delivered
	CurrentDish	The current dish that is to be edited
	EditedDish	The updated dish that will be delivered to the customer. If the dish was deleted, this field will indicate it
DishQueue	TableNumber	The table number corresponding to where the dish was ordered
	Dish	A dish currently present in the queue
	DishQueuePosition	The position of the dish in the queue
DishCompleteNotifier	TableNumber	The table number corresponding to where the

		dish was ordered
	Dish	The dish that has been completed
InventoryDatabase	Ingredient	An ingredient in the inventory database
InventoryNotifier	Ingredient	An ingredient in the inventory that has been alerted to the manager by the system
	NotificationType	The type of notification sent out by the system to the manager. These notifications can be a predicated date when the ingredient will run out, when stock for an item falls below a predefined threshold, when an item goes bad, and when restock orders need to be made
InventoryChanger	Ingredient	The ingredient to be added, changed, or removed from the inventory
MenuChanger	Dish	The dish to be changed

4.1.3 Association definitions

Concept pair	Association description	Association name
OrderTaker <-> DishQueue	OrderTaker takes orders from tables and sends them to DishQueue for queueing	sends order information
OrderTaker <-> Menu	OrderTaker reads menu item information from the menu to construct the object it sends to the DishQueue	reads dish information
AssistButton <-> TableStatusView	AssistButton sends the assistance request to TableStatusView for display	sends assistance request
FoodTimeDisplay <->	FoodTimeDisplay receives information from	receives dish

DishQueue	DishQueue in order to calculate food wait time	information
ReadyOrderQueue <-> DishCompleteNotifier	ReadyOrderQueue receives finished dishes from DishCompleteNotifier and queues them for delivery to tables	receives finished dish events
DishQueue <-> DishCompleteNotifier	DishQueue sends a dish complete event to DishCompleteNotifier	sends finished dish events
DishQueue <-> InventoryChanger	When new items are added to DishQueue, DishQueue updates the inventory with the ingredients used in that item	sends inventory update information
InventoryDatabase <-> InventoryNotifier	When a warning event happens (restock confirmation, no stock remaining, etc), InventoryDatabase sends the type of event to InventoryNotifier	generates inventory warnings
InventoryDatabase <-> InventoryChanger	InventoryChanger sends information on what ingredients and fields to update to InventoryDatabase	receives inventory update information
MenuChanger <-> Menu	MenuChanger sends information on what dishes and fields to update to the Menu	sends menu update information

4.1.4 Traceability Matrix

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
UC1									X		X		
UC2									X				
UC3									X		X		
UC4									X		X		
UC5									X				
UC6									X	X			
UC7									X	X	X		
UC8		X										X	
UC9		X										X	
UC10		X										X	
UC11		X										X	
UC12		X										X	
UC13		X											

UC14							X						
UC15							X						
UC16							X						
UC17					X		X	X					
UC18	X	X					X						
UC19	X	X					X						
UC20													X
UC21				X									
UC22			X										
UC23			X										
UC24		X											
UC25		X											
UC26								X		X			

4.2 System Operation Contracts

4.2.1 Use Case UC - 1: ManageInventory

Preconditions:

- IsLoggedIn = True
- IsAuthorized = True
- IsDatabaseOperational = True
- IsNotifierOperational = True

Postconditions:

- DatabaseChange = True
- Database and Inventory have been updated
- The changes made have been saved in the log

4.2.2 Use Case UC - 8: MangeMenu

Preconditions:

- IsLoggedIn = True
- IsAuthorized = True
- IsDatabaseOperational = True

Postconditions:

- DatabaseChange = True
- Database and Menu have been updated
- The changes made have been saved in the log

4.2.3 Use Case UC - 14: MangeOrders

Preconditions:

IsLoggedIn = True

- IsAuthorized = True
- IsDatabaseOperational = True
- IsQueueEmpty = False

PostConditions:

- DatabaseChange = True
- Database and OrderQueue have been updated

5. Mathematical Model

5.1 Scheduling Algorithm for Chefs

Since this algorithm is for placing a new menu item into the queue for chefs to cook, it must be run every time a customer places an order.

Our algorithm for scheduling is described as follows:

ItemToBeQueued - The next item that needs to be added to the queue

OrderQueue - The current queue of menu items ready to be cooked by a chef

CurrentItem - First Item in Queue, it is not the item that the chef is currently checking. When the chef takes an item from the queue to be cooked, it is removed from the queue.

Freshness time - Menu Item "freshness" time or the length of time it can be sitting out.

```
while CurrentItem's Average Time to complete > ItemToBeQueued's Average Time to Complete

if ItemToBeQueued's  menu item type is the same as the  currentitem's menu item type
    for each menu item at position until end of queue
        if( menu item belongs to same order)
            length + = menu item's average wait time
        if length < Current Item's freshness factor
        position = CurrentItem's position
Until all menu items in ChefQueue
if( !position )
    position = end of queue.
ItemToBequeued's position = position
```

The scheduler will loop through the current queue and find the best place to place the item to be queued. It will check if the item is of the same type as the item to be queued, then it will determine if the average time to complete is greater than that of the item to be queued and also it will check to see if the freshness time rule is kept. If these things are true, it will store that position and at the very end check if a position was found. If it wasn't found then we know that none of these were met and the item will be added at the end of the queue.

5.2 Inventory usage rate estimation and run-out date estimation

Our recursive algorithm estimates ingredient usage for a single ingredient i for day n (today) based on estimated past daily ingredient usage (day n-1, n-2 … n-7) and real-world weekly ingredient usage (week w-1 … w-4). Four weeks was chosen because the window excludes long term seasonal weighting; that is, if a menu item is extremely popular in the winter but not in spring its past popularity won't affect ingredient usage estimates for spring.

Daily ingredient usage will be estimated by adding the usage per menu item of ingredient i for all menu items ordered on that day. Weekly ingredient usage will be real-world data obtained from manual inventory updates by restaurant employees.

In the beginning, there will be no data for past weeks or days. Because weekly information isn't estimated but is based on real-world data, the algorithm will have to take the first few weeks to gather data. At minimum, the algorithm could be run after a single week using only one week's real-world data instead of the average of four weeks. After the first four weeks, the algorithm will be running optimally. Alternatively, the restaurant manager can manually input his own estimate of when an ingredient will run out; until enough real data is gathered, the rate of usage based on this estimate can be used to tune the initial predictions and allow an experienced restaurant manager to guide the system until it has enough data.

First, we determine the relative usage of ingredient i on day n-7 compared to total estimated usage that week, aka the proportion of that ingredient used on the same day of the week last week. That proportion is multiplied by the averaged real-world weekly usage, and yields the estimated ingredient usage for today.

Day n proportional usage: $U(n-7) / (U(n-1) + U(n-2) + U(n-3) + U(n-4) + U(n-5) + U(n-6) + U(n-7))$

Weekly average = $U(w-1) + U(w-2) + U(w-3) + U(w-4)) / 4$

$U_i(n) = [(U(w-1) + U(w-2) + U(w-3) + U(w-4)) / 4] * [U(n-7) / (U(n-1) + U(n-2) + U(n-3) + U(n-4) + U(n-5) + U(n-6) + U(n-7))]$

where U(week) is the actual weekly usage taken from the inventory system's data, and NOT a value returned by the recursive function. This shorthand was used to increase readability.

To take into account holidays and other special occasions, we propose a table of dates and "usage modifiers" which can either multiply an entire day's overall ingredient usage (for nonspecific busy days) or multiply only certain ingredients, such as cranberries on Thanksgiving. With this modification, the final equation is:

$$U_i(n) = M * \{[(U(w-1) + U(w-2) + U(w-3) + U(w-4)) / 4] * [U(n-7) / (U(n-1) + U(n-2) + U(n-3) + U(n-4) + U(n-5) + U(n-6) + U(n-7))]\}$$

5.3 Wait time estimation

Basic design for a function WaitTime which returns the wait time for a single item/dish is shown below:

```
WaitTime_D(Dish currentDish)
{
        Time waitTime;
        Time now = currentTime;

        waitTime = currentDish.getArrivalTime() + currentDish.getCookTime() - now;
        return waitTime;
}
```

Finding wait time for an entire table is almost equally as simple: the finish time for a table is equal to the finish time of the dish that is finished last. In simple pseudocode, it could be modeled as such:

```
WaitTime_O(Order currentOrder)
{
Time maxWaitTime = currentTime;
Time temp;
Time now = currentTime;

for this Dish = each dish belonging currentOrder
{
        temp = WaitTime_D(thisDish);
        if temp > maxWaitTime
                maxWaitTime = temp;
}
return now + maxWaitTime;
}
```

5.4 Queueing of table orders for waiters

Before we describe the algorithm for queueing, we need to describe how the wait list gets built.

Necessary Precondition: Customer chooses to hold menu items until entire table's menu items are ready

```
Create table order with id equivalent to customers table
set total number of menu items
set current number ready to zero
set table = number equivalent to customer's table
create empty MenuItemList

for each menu item in customers order
    add reference to item in MenuItemList

add table order to wait queue
```

When the customer creates a composite order and specified that he wants to wait until the the entire order is ready, a corresponding table order will be created. If the customer creates a singular order or specifies that he wants the items on a first come first serve basis, then a table order will not be created. Finally this table order is added to the wait queue. This is important for how the queuing algorithm works.

Now we can describe our final algorithm for queueing table orders for waiters.

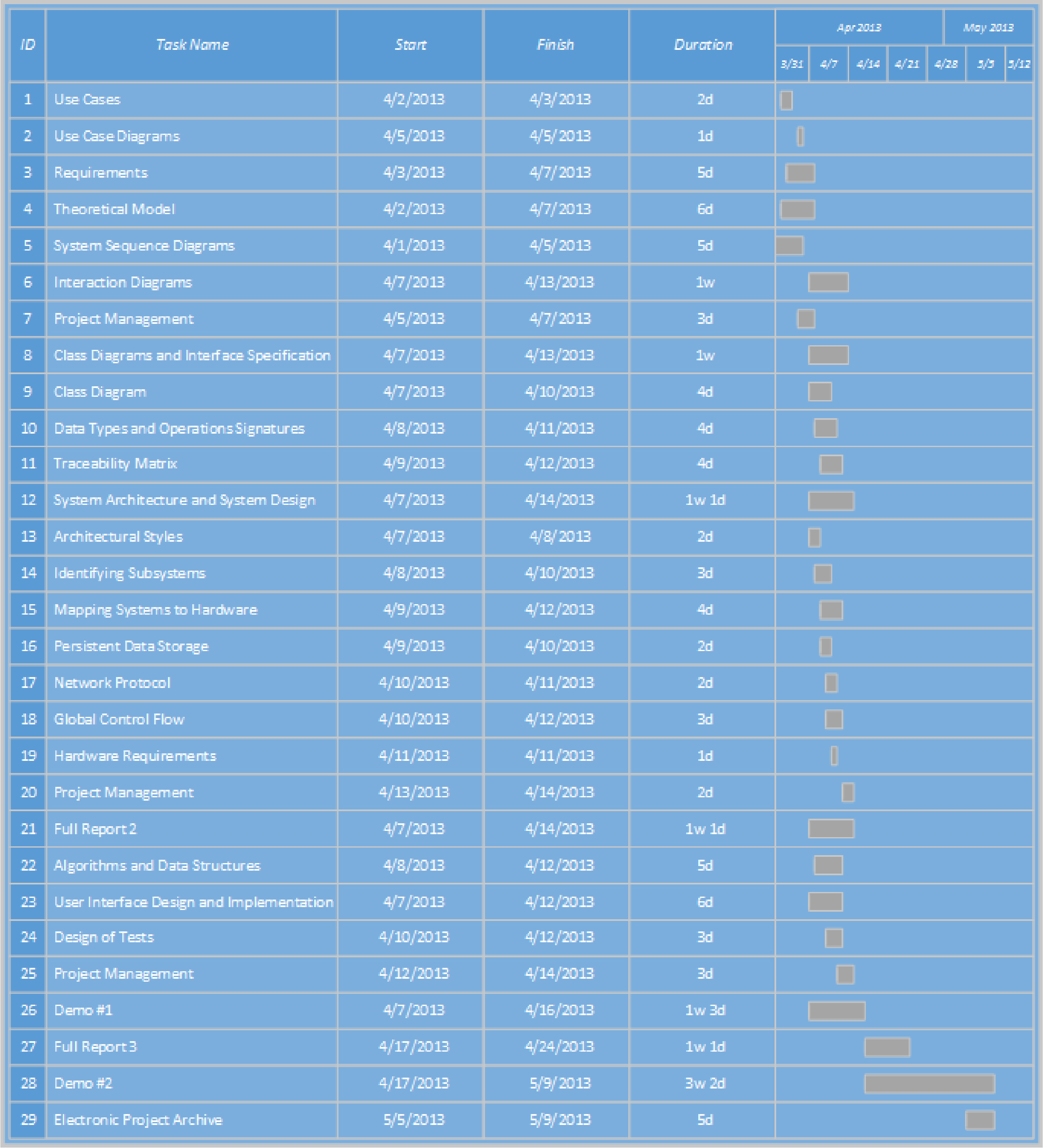
ReadyMenuItem = menu item that has been cooked and needs to be queued.
Wait List = The current wait list of table orders
ready queue = the current ready queue of either table order or menu items that are ready to be delivered.

```
boolean foundTable = false
for each table order in wait list
    for each menu item in table order's menu item list
        if readyMenuItem == menu item
            table order.currentNumReady++
            if table order.currentNumReady == table order.totalNumItems
            move table order to ready queue
            foundTable = true


if ( ! foundTable )
add readyMenuitem to Ready Queue
```

When a menu item is finished cooking, the chef flags that it is done cooking and the scheduling algorithm places it either in the ready queue or the wait queue. It will first search every menu order that is in the wait queue to see if the item corresponds to any of the table orders in the wait queue. If it finds the item in a menu order, then it will increment the number of ready of items that are ready for that table order. If all the items are ready after this one has been added then the table order can be added to the ready queue. if the menu item could not be found in any of the table through the boolean foundTable, then we know what the item either belongs to a singular order or the customer requested his items to come in a first come first serve fashion and therefore can be directly placed onto the ready queue.

6. Project Management - Projected Work Flow


Apr 2013
May 2013
3/31
4/7
4/14
4/21
4/28
5/5
5/12

ID	Task Name	Start	Finish	Duration
1	Use Cases	4/2/2013	4/3/2013	2d
2	Use Case Diagrams	4/5/2013	4/5/2013	1d
3	Requirements	4/3/2013	4/7/2013	5d
4	Theoretical Model	4/2/2013	4/7/2013	6d
5	System Sequence Diagrams	4/1/2013	4/5/2013	5d
6	Interaction Diagrams	4/7/2013	4/13/2013	1w
7	Project Management	4/5/2013	4/7/2013	3d
8	Class Diagrams and Interface Specification	4/7/2013	4/13/2013	1w
9	Class Diagram	4/7/2013	4/10/2013	4d
10	Data Types and Operations Signatures	4/8/2013	4/11/2013	4d
11	Traceability Matrix	4/9/2013	4/12/2013	4d
12	System Architecture and System Design	4/7/2013	4/14/2013	1w 1d
13	Architectural Styles	4/7/2013	4/8/2013	2d
14	Identifying Subsystems	4/8/2013	4/10/2013	3d
15	Mapping Systems to Hardware	4/9/2013	4/12/2013	4d
16	Persistent Data Storage	4/9/2013	4/10/2013	2d
17	Network Protocol	4/10/2013	4/11/2013	2d
18	Global Control Flow	4/10/2013	4/12/2013	3d
19	Hardware Requirements	4/11/2013	4/11/2013	1d
20	Project Management	4/13/2013	4/14/2013	2d
21	Full Report 2	4/7/2013	4/14/2013	1w 1d
22	Algorithms and Data Structures	4/8/2013	4/12/2013	5d
23	User Interface Design and Implementation	4/7/2013	4/12/2013	6d
24	Design of Tests	4/10/2013	4/12/2013	3d
25	Project Management	4/12/2013	4/14/2013	3d
26	Demo #1	4/7/2013	4/16/2013	1w 3d
27	Full Report 3	4/17/2013	4/24/2013	1w 1d
28	Demo #2	4/17/2013	5/9/2013	3w 2d
29	Electronic Project Archive	5/5/2013	5/9/2013	5d

These start and finish dates are estimates and might change in the future.

References

[1] http://www.enewsbuilder.net/peoplereport/e_article000657699.cfm?x=b11,0,w (Robot Picture)
[2]http://www.ece.rutgers.edu/~marsic/books/SE/projects/Restaurant/2012-g11-report3.pdf (Group 2, 2011 report)