A PROJECT PROPOSAL

FOR THE

INVENTORY CONTROL SYSTEM

FOR

CALCULATION AND ORDERING OF

AVAILABLE AND PROCESSED RESOURCES

GROUP 9

- **SIMANT PUROHIT**
- **BART MICZEK**
- **AKSHAY THIRKATEH**
- **BOB FAIGAO**

Executive Summary

Our proposed project is a real time implementation of an inventory control system for an on-site corporate restaurant management and catering company. One such company is Guckenheimer (www.guckenheimer.com) which builds, staffs, and upkeeps corporate kitchens as well as provides catering services to corporate companies. This project is specific in that it applies to the dining domain of restaurants, but is flexible enough to be applied to many different kitchens and restaurants. In the case of Guckenheimer, they can use the software in their kitchens across the nation. The scope of this project will primarily focus on Guckenheimer's kitchen and inventory located at the Groupon Chicago Office.

Currently at Groupons kitchen, and the food industry in general, restaurant staff and managers are forced to keep track of inventory by hand. This means that they must count what they have sold and what they have left at the end of each day. They must also fill out order forms to be sent to vendors so that they can restock their inventory in preparation for the next week. This wastes valuable man hours and is a rather simple task to automate using our software.

We propose a solution to this issue by developing software that keeps track of inventory in the "back of house", or kitchen, and updates it according to daily sales. Each food item is linked to respective resources (or ingredients) and as each product is sold the ingredients utilized in making that product are also utilized. These changes in inventory are kept track of through utilizing a database.

We propose to keep track of each and every ingredient by dynamically linking it to the product and as a result create a dependent relationship to that product. At a specific time period (typically the end of the week); if the inventory is below the threshold level, order forms to the specific vendors are generated in order to restock the required items for the next week. The project also makes smart predictions on required inventory for the following week based upon the predicted climate and possible occasions or events that may influence near future sales. At the end of the week, the software takes into account all threshold levels, predictions, and other factors to generate an order form, which after being verified by the manager is sent out to the vendors.

The Purpose of the Project

A case study at 'Guckenheimer' (an on-site corporate restaurant management and catering company) cited issues regarding a basic resources requirement list that has to be maintained manually by the staff. To keep track of their inventory levels they have to calculate a list of the groceries utilized during a course of time, calculate and analyze the requirements for the future, and place their next order to the vendors if needed. This process takes up a lot of time and human effort, and is also prone to human error.

This poses a problem of a situation that the staff at 'Guckenheimer,' as well as many other restaurants faces. It takes up a lot of time to manually keep track of sales and place correct orders to vendors, wasting useful labor in trivial works. A product which would assist in tackling the above mentioned problems would prove to be fruitful to clients such as 'Guckenheimer' and similar enterprises as this product would help convert the unproductive time to something more useful, by removing the unnecessary error prone complications and efforts.

Goals of the Project

The project aims at providing an efficient interface to the restaurants for managing their grocery inventory based on each item sold. The basic idea involved here is that each item is linked to its atomic ingredients which are stored in a database. At the end of each day, the system analyzes the total sale of menu items and proportionately deducts appropriate amount from the resource database. Then it compares the current available resources with the threshold level of each ingredient. If it finds that certain ingredients are below the threshold, it will generate a purchase order for those item(s) and send it to the manager (admin) for approval.

We also propose to include a special feature "Prediction". This feature keeps track of any upcoming occasions, climatic changes and special events that may influence inventory needs for the upcoming week. The system will then predict the required resources for these events based on previously accumulated information/knowledge. It will now generate an updated purchase order in accordance with the predictions.

The product also aims to keep track of the shelf life of resources. If any resource nears the end of its shelf life, it would intimate to the manager (admin) the details of the quantity that is near its expiration date. The restaurant must function efficiently, the groceries must be tracked correctly, timely orders must be sent out to the vendors, and the inventory must be maintained and updated at all times.

The Domain

This proposed project aims at inventory control in the restaurant and catering Industry. Such a large domain would result in an equally as large scope of development. As a result we narrow our software down to our case study of an outlet of Guckenheimer concentrating only on the basic resources utilized in inventory control of the outlet. Although the software will be developed keeping in mind the needs of Guckenheimer and available data at first, then applying it to the larger domain of the entire restaurant industry can be achieved with ease.

Our target domain is full of software to track sales of food items, but lacks in this area of inventory management. Our software can be scaled from large corporate dinings all the way to small privately-owned restaurants. It is also fairly domain specific: the database runs off recipes which generate the necessary ingredients. It also updates the inventory based off of the sale of those recipes. This requirement focuses our product to our domain and makes it more appealing to those looking for a solution to this specific problem.

The Client

The client can vary from private restaurant owners to corporate restaurant management companies, such as Guckenheimer (www.guckenheimer.com). A corporate restaurant management company that starts up, staffs, and oversees the everyday workings of a corporate restaurant, such as the one in the Groupon Chicago office. As stated above, while our product can be applied to the entire domain of the restaurant and catering business, focusing on a specific business provides us with more precise and consistent data. A company such as Guckenheimer would be an ideal client, as they staff multiple corporate kitchens across the nation, including kitchens for Groupon and even Google. A large scale company such as this this can apply our software to each and every kitchen, cutting down costs on a very large scale.

Our software will allow our client to customize the database to suit the needs of each kitchen individually. They can vary in recipes, vendors from which they order their products, and threshold levels. This provides a uniform product that can be customized at a smaller scale. Our client would need to purchase multiple licenses, or more likely a corporate subscription that would allow them to use the software in multiple kitchens. We would also offer single use licenses to appeal to restaurants that only need to manage a single inventory of goods.

User of the product

The main users of the product would be kitchen management and staff. The management would approve the orders that would be sent out, provide vendor information, upload recipes, and set threshold levels. Many of these tasks, such as the information regarding vendors, recipes, and threshold levels would need to be set only once. Of course, the option to add, remove, or update this data would be implemented as well. Once this initial step has been taken, our software will require nothing more than a weekly approval for the orders being sent out, minimizing the work that management has to complete in order to insure the correct amount of inventory is available.

Kitchen staff would be responsible for updating the amount of product sold at the end of the day. Each day, the register prints out the products sold and the quantity of each product sold. Instead of manually subtracting that amount from the inventory, they input the amounts sold into our software which will do the number crunching for them. This data is also stored into the "predictions" feature for future use.

Domain Expert

Ms. Kimberly Harmon, employee of Guckenheimer, chef at Groupon Chicago Office
Phone: 801.361.6597
Email: kharmon128@hotmail.com

Hardware Requirements

Processor - Intel Dual-core processor, 2.0 GHz or higher

RAM - Minimum 2 GB of RAM

Network interface chip

Hard Drive - 500 GB

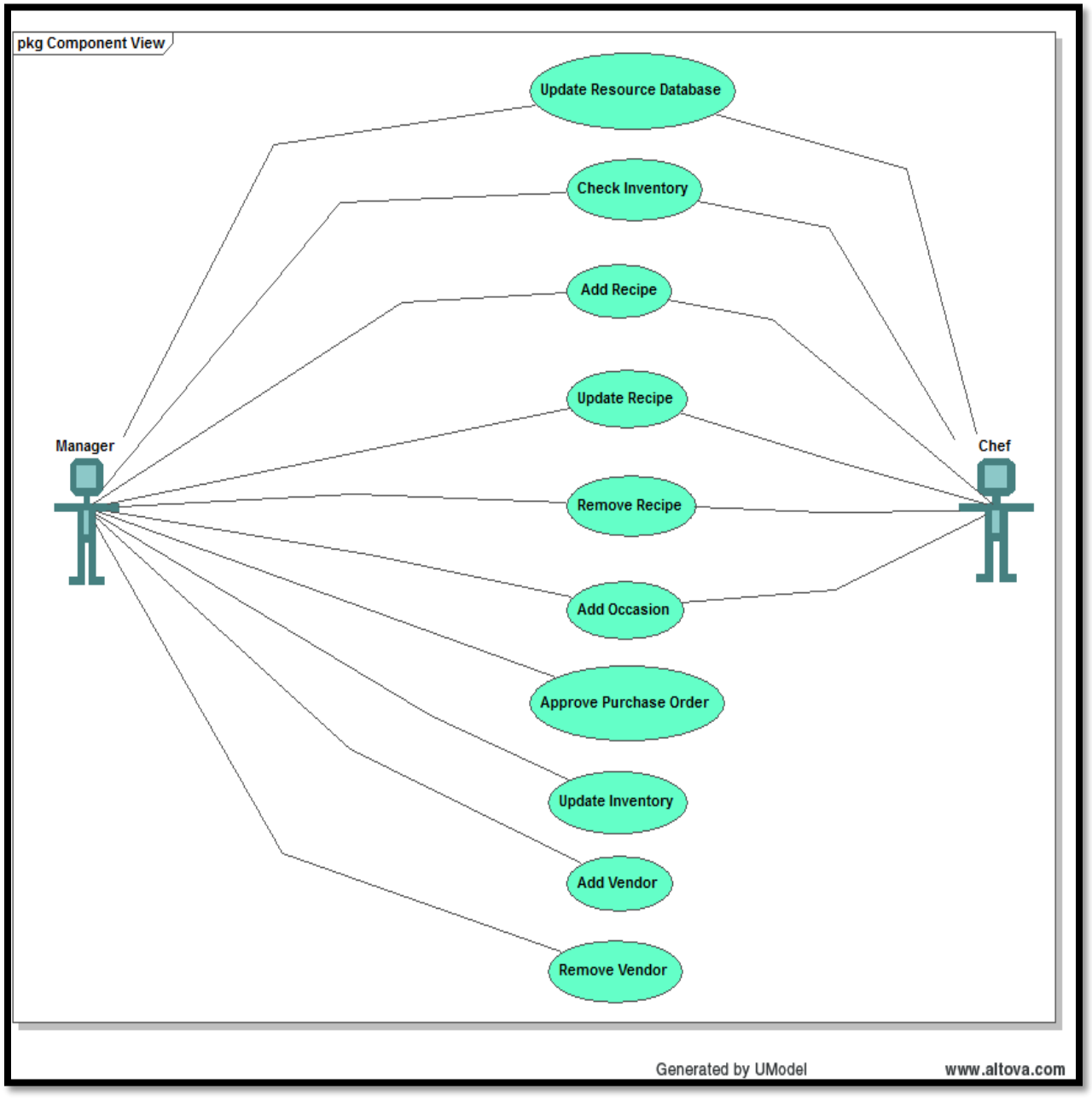
Software Requirements

Operating System: Windows XP or higher.

Front End: VB.NET (Visual Studio 2010)

Back End: Microsoft Access 2010

Overall UML Diagram



Typical Use Cases

1) Update resource database

Use case name	**UpdateResourceDatabase**
Participating Actors	Initiated by *Manager(admin) or Chef*
Flow of events	1. The *Manager or Chef* activates the update resource database function 2. The *Manager or Chef* inputs the amount of each item sold. 3. The *System* reads the sold food data and then further reads, from the ingredients database, the ingredients that were used in making of the food items that were sold. The *System* now calculates the amount of resources used and will deduct the amount of ingredients that were used up from the resource database.
Entry condition	The *Manager(admin)* or *Chef* is logged on to the *System*
Exit condition	If the process was successful, the *Manager/Chef* receives an acknowledgement that the process was completed successfully. OR If the process was not successful, the *Manager* will receive an explanation of what error had occurred during the process.
Quality Requirements	The update process must complete successfully and without errors.

2) Check inventory

Use case name	**CheckInventory**
Participating Actors	Initiated by *Manager(admin) or Chef*
Flow of events	1. The *Manager/Chef* activates the "Check Inventory' function on his/her terminal. 2. The System displays the current estimated inventory of ingredients to the *Manager/Chef*. 4. The System will compare the current levels of ingredients with the pre-set threshold levels. 5. If the levels of ingredients are found to be below threshold, it will create orders for purchase and sends it to the *Manager* for approval. 6. The *Manager/Chef* is notified of the process completion
Entry condition	The *Manager/Chef* is logged into System.
Exit condition	Successful Acknowledgement.
Quality Requirements	The function accurately calculates the inventory that should be left when all of the orders have been calculated

3) Add Recipe

Use case name	**AddRecipe**
Participating Actors	Initiated by *Chef/Manager*
Flow of events	1. The *Chef/Manager* activates the "Create New Recipe" function on his/her terminal 2. The System responds by presenting a form to the *Chef/Manager* 3. The *Chef/Manager* completes the form by inserting ingredients to be used in the new recipe. The *Chef* also inputs the amount of ingredients to be used in a single order of the recipe. After the form has been completed the *Chef* submits the form to the *System*. 4. The *System* acknowledges that the new recipe has been created. It also adds it to the recipe database.
Entry condition	The *Chef/Manager* is logged into *System*
Exit condition	The *Chef/Manager* has received an acknowledgment from the *System*. OR The *Chef/Manager* has received an explanation of why the process couldn't be completed.
Quality Requirements	The process must complete successfully with the new recipe added to the recipe database without any errors.

4) Update recipe

Use case name	**UpdateRecipe**
Participating Actors	Initiated by *Chef/Manager*
Flow of events	1. The *Chef/Manager* activates "Update Recipe" on system 2.*System* responds by bringing up list of recipes 3. The *Chef/Manager* chooses a recipe to change 4. *System* updates by showing list of ingredients in chosen recipe 5. The *Chef/Manager* changes the recipe by choosing new ingredients or updating the amount of ingredients used in the recipe. The *Chef* then finishes the update by selecting the finished command on the system. 6. The *System* confirms that the change has been made and updates the database.
Entry condition	The *Chef/Manager* is logged into *System*
Exit condition	The *Chef/Manager* receives an acknowledgment from the *System*, OR The *Chef/Manager* has received an explanation of why the process couldn't be complete.
Quality Requirements	The update process must be complete successfully without any errors.

5) Remove recipe

Use case name	**RemoveRecipe**
Participating actors	Initiated by *Chef/Manager*
Flow of events	1. The *Chef*/*Manager* activates the "Remove Recipe" function on his/her terminal 2. The System responds by showing the current list of recipes saved on the System. 3. The *Chef*/*Manager* chooses which recipe(s) to remove and removes them by selecting a delete button through the terminal window. 4. The System confirms with each deletion with the *Chef/Manager* if he/she wants to delete the recipe. 5. The *Chef*/*Manager* confirms his/her decision with a yes/no 6. The System acknowledges the decision by either removing the recipe if responded with "yes" or by canceling the delete if responded with "no". It then displays an acknowledgment of the decision by displaying a delete successful or a canceled request. 7. The System notifies the *Chef/Manager* about the change and requests new threshold levels for ingredients from deleted recipes.
Entry condition	The *Chef/Manager* is logged in System
Exit condition	The *Chef/Manager* has received an acknowledgment that the recipe has been deleted. OR The *Chef/Manager* has received an acknowledgment that the recipe has not been deleted. OR The *Chef/Manager* has received an explanation of why the process couldn't be completed.
Quality Requirement	The removed recipe should not reflect in any other list or connected database.

6) Adding an occasion/busy days/weekends etc.

Use case name	**AddOccasion**
Participating actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates the "Add Occasion or Event" function on his/her terminal. 2. The System displays a form to be filled out by the manager. 3. The *Manager* fills out the form by adding a name of the event or occasion and selecting the date(s) the event is to be held. 4. The System takes the data from the form and calculates the amount of ingredients that may be used up for the given dates. 5. The *Manager* is notified of the calculations and accordingly orders requests are initiated. 6. On successful completion of this process, an acknowledgement is sent to the *Manager*.
Entry condition	The *Manager* is logged into System.
Exit condition	The *Manager* receives a notification of successful completion of the project OR The *Manager* is notified that the process was not complete with a valid explanation of the error that had occurred during the process.
Quality Requirements	The Occasion is accurately added to the database.

7) Approve Purchase Order

Use case name	**ApprovePurchaseOrder**
Participating actors	Initiated by *Manager*
Flow of events	1. The *Manager* reviews the purchase order created by the System. 2. After successful review, the manager either approves the purchase order right away or makes changes to the order and then provides approval. 3. After receiving approval from the Manager, the System will send out purchase orders to respective Vendors. 4. *Manager* receives an acknowledgment of the process completion.
Entry condition	*Manager* is logged on to the System
Exit condition	Orders are sent successfully
Quality requirements	The purchase orders are sent successfully to designated vendors.

8) Updating inventory once order is received

Use case name	**UpdateInventory**
Participating actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates the "Update Stock inventory" function on his/her terminal. 2. The System updates the inventory database based on the amount ordered from the vendors. The System then displays the amount that it calculated to the vendor and waits for approval from the *Manager*. 3. The *Manager* either approves the amount calculated or updates amounts in the inventory. 4. If the *Manager* updates the inventory, the System updates its values of the inventory.
Entry condition	The *Manager* is logged into the System
Exit condition	The Inventory levels are successfully updated
Quality Requirements	The number shown to the manager accurately shows the actual amount of ingredients stored

9) Add vendor

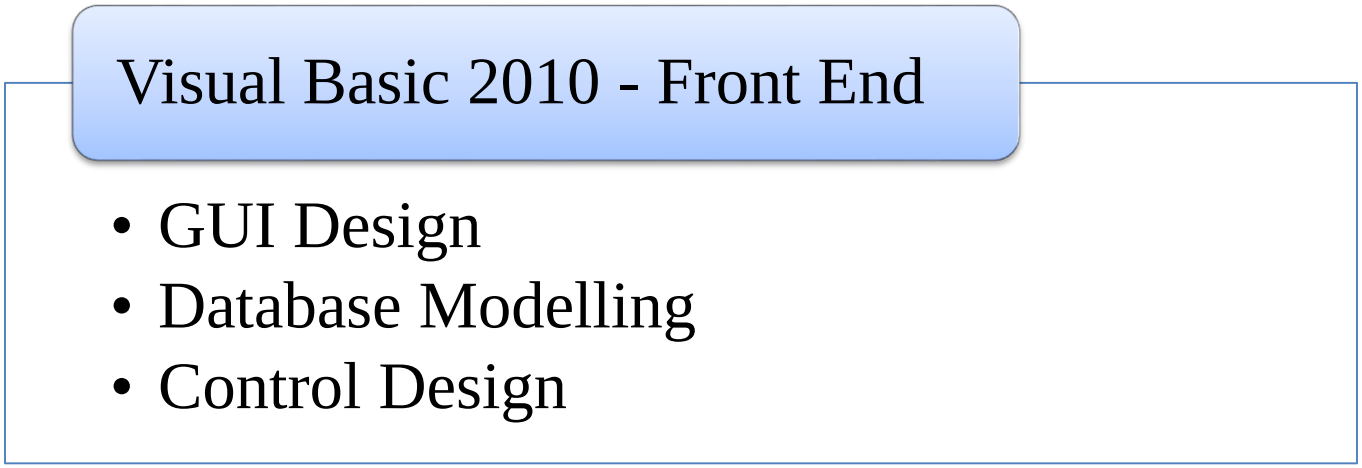
Use case name	**AddVendor**
Participating actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates the "Add Vendor" function on his/her terminal 2. The System responds by displaying a form to be completed by the *Manager* of the vendor to be created. 3. The *Manager* completes the form by filling the information of the vendor to be created and also the ingredients that will be ordered from that vendor. After all of the information has been filled in, the Manager then submits the form. 3. The System takes the information from the form and adds the vendor the database of vendors. It then displays an acknowledgment to the *Manager* that the Vendor has been created.
Entry condition	The *Manager* is logged in System.
Exit condition	The *Manager* has received an acknowledgment that the vendor has been created. OR The *Manager* has received an explanation of why the process couldn't be completed.
Quality Requirements	The Vendor has been accurately stored into the database

10) Remove Vendor

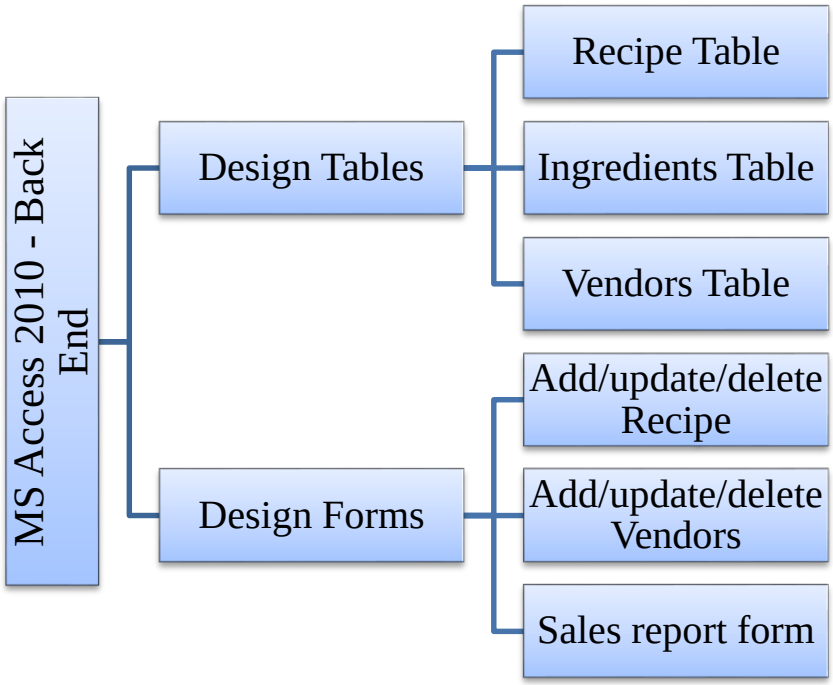
Use case name	**RemoveVendor**
Participating actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates the "Remove Vendor" function on his/her terminal 2. The System responds by showing the current list of Vendors saved to the System. 3. The *Manager* chooses which vendor(s) to remove and removes them by selecting a delete button through the terminal window. 4. The System confirms with each deletion with the *Manager* if he/she wants to remove the vendor. 5. The *Manager* confirms his/her decision with a yes/no 6. The System acknowledges the decision by either removing the vendor if responded with "yes" or by canceling the delete if responded with "no". It then displays an acknowledgment of the decision by displaying a delete successful or a canceled request.
Entry condition	The *Manager* is logged into the System
Exit condition	The *Manager* has received an acknowledgment that the vendor has been removed. OR The *Manager* has received an acknowledgment that the vendor has not been removed. OR The *Manager* has received an explanation of why the process couldn't be completed.
Quality Requirements	The Vendor should not appear in the list of active vendors or any other database

Architecture Breakdown

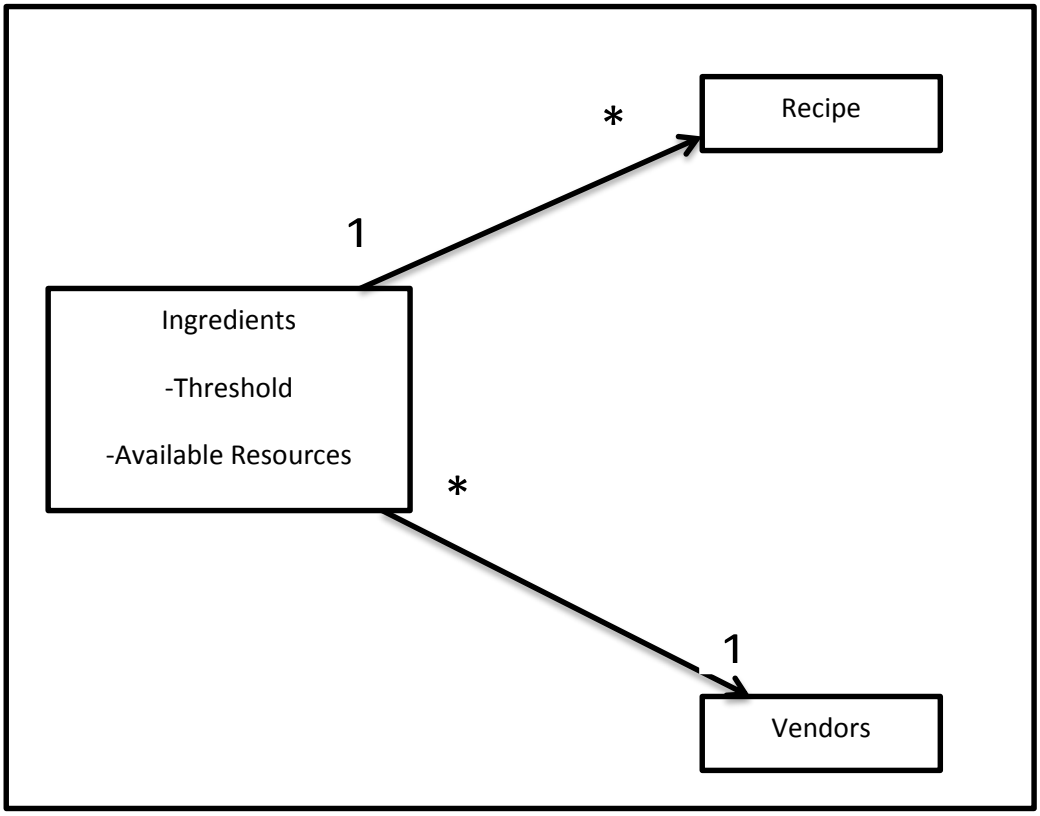
Front End:-



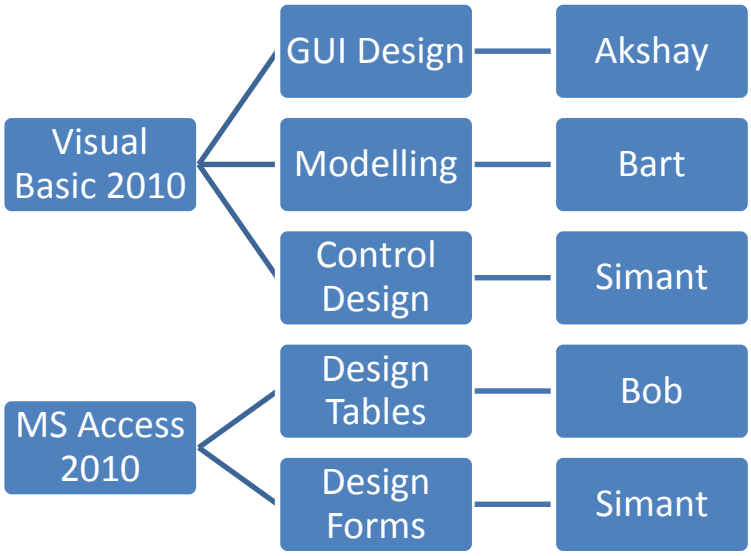
Back End:-



Basic Database Relationship Diagram



Assignment of responsibilities



Project Work Plan (Gantt Chart)