SYSTEM REQUIREMENTS SPECIFICATIONS

FOR THE PROJECT

INVENTORY CONTROL SYSTEM

FOR

CALCULATION AND ORDERING OF

AVAILABLE AND PROCESSED RESOURCES

GROUP 9

- **SIMANT PUROHIT**
- **BARTLOMIEJ MICZEK**
- **AKSHAY THIRKATEH**
- **ROBERT 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.

INDEX

1. Introduction
    - 1.1 Purpose of the system…….. 4
    - 1.2 Scope of the system……. 4
    - 1.3 Objectives and success criteria of the project……. 5
    - 1.4 The Domain of the Project……..... 5
    - 1.5 The Client……. 6
    - 1.6 The User…….. 6
    - 1.7 Definitions, acronyms, and abbreviations…….. 7

2. Current system…… 8

3. Proposed system
    - 3.1 Overview…… 8
    - 3.2 Functional requirements…….. 9
    - 3.3 Nonfunctional requirements
        - 3.3.1 Usability…… 9
        - 3.3.2 Reliability……. 9
        - 3.3.3 Performance……..... 10
        - 3.3.4 Supportability……..... 10
        - 3.3.5 Packaging…….. 10
        - 3.3.6 Implementation…… 10
        - 3.3.7 Interface…… 11
        - 3.3.8 Legal…… 11
    - 3.4 System models
        - 3.4.1 Scenarios…….. 12
        - 3.4.2 Use case model…… 19
        - 3.4.3 Object model
            - 3.4.3.1 Generalization and Class Hierarchy Diagram……. 41
            - 3.4.3.2 Multiplicity Diagram……. 42
            - 3.4.3.3 Association Diagram…… 43
        - 3.4.4 Dynamic model (Sequence Diagrams)…… 44
        - 3.4.5 User interface—navigational paths and screen mock-ups…….. 50
        - 3.4.6 Architecture Breakdown……. 56

4. Glossary…… 58

1. Introduction

1.1 Purpose of the system

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.

1.2 Scope of the system

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.

1.3 Objectives and success criteria of the project

The objective of the project is to provide an efficient inventory control whose main functionality apart from calculating the inventory include predicting the requirement for the next order and also if there is a "Special Occasion" then accordingly the manager selects the particular occasion and extra requirements is added to the next issuing order to the vendors which needs to be approved by the manager. 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 success criteria depends on

- The accuracy in maintaining the inventory levels
- The accuracy in predicting the requirements of the next order
- The accuracy in relating recipes to their respective ingredients
- Ease of use when it comes to updating inventory levels and placing orders to vendors

1.4 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 dining 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.

1.5 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.

1.6 The User

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.

1.7 Definitions, acronyms, and abbreviations

- Manager: The manager implies the manager of the restaurant/company who handles all the administrative works.

- Recipe: This is the menu item that the restaurant/company provides to its customers.

- Ingredient: This is the entity that the recipe is composed of.

- Vendor: This is the company that provides the restaurant/company with the required ingredients.

- Order: Order is the list of ingredients and the quantities that is or is to be requested from the vendor.

2. Current system

"Guckenheimer" (an on-site corporate restaurant management and catering company) follows a system where the basic resources list needs to be manually calculated at the end of a certain time period by the staff. They must accordingly check the inventory levels for determining if they are below the threshold level then orders are processed to the vendors. This sort of system not only leaves a lot of room for human error, but is also incredibly time consuming. The lack of a centralized database also creates an issue when it comes to keeping track of inventory levels as well as past trends in ingredient requirements. The system also relies on human intuition and guesswork to place the correct orders for the following week, which will not be as precise as an algorithm designed for this purpose.

3. Proposed system

3.1 Overview

We propose to develop 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.

3.2 Functional requirements

The System aims at providing an efficient interface to the user for managing of inventory, it shall also provide the user varied options for managing the inventory through various functions at hand. The ingredient levels are continuously monitored based on their usage and are checked for the threshold levels in the inventory and accordingly the user is alerted about low levels of certain ingredients. The design is such that the user does not have to manually update the inventory every time, the System does if for the user.

The System calculates and predicts the amount of usage for specific set days that are pre-set by the user(admin) , it also alerts the user of an impending action to order ingredients before the specific day set by the user. Therefore the user never has to worry about manually calculating the estimated usage of the ingredients as the System does it for the user.

The simple interface of the System has functions like adding a recipe, removing or updating the recipe. It also extends to functions such as adding a vendor for an ingredient,, removing the vendor, checking threshold levels, processing orders, altering processed orders etc.

3.3 Nonfunctional requirements

- **3.3.1 Usability**
  - The system must be easy to use by both managers and chefs such that they do not need to read an extensive amount of manuals.
  - The system must be quickly accessible by both managers and chefs.
  - The system must be intuitive and simple in the way it displays all relevant data and relationships.
  - The menus of the system must be easily navigable by the users with buttons that are easy to understand.

- **3.3.2 Reliability**
  - The System must give accurate inventory status to the user continuously. Any inaccuracies are taken care by the regular confirming of the actual levels with the levels displayed in the system.
  - The System must successfully add any recipe, ingredients, vendors or special occasions given by the user and provide estimations and inventory status in relevance with the newly updated entities.

- The system must provide a password enabled login to the user to avoid any foreign entity changing the data in the system.
- The system should provide the user updates on completion of requested processes and if the requested processes fail, it should provide the user the reason for the failure.
- The system should not update the data in any database for any failed processes.

- **3.3.3 Performance**
  - The system must not lag, because the workers using it don't have down-time to wait for it to complete an action.
  - The system must complete updating the databases, adding of recipe, ingredient, vendor and occasions successfully every time the user requests such a process.
  - All the functions of the system must be available to the user every time the system is turned on.
  - The calculations performed by the system must comply according to the norms set by the user and should not vary unless explicitly changed by the user.

- **3.3.4 Supportability**
  - The software is designed such that it works even on systems having the minimum configuration.
  - The system is adaptable even if additional plugins or modules are added at a later point.
  - The data can be exported to the manager so as to make the system more portable.

- **3.3.5 Packaging**
  - The system must be able to run on the Windows operating systems beginning with Windows XP, and must be able to run on future releases such as the upcoming Windows 8
  - The software must incorporate a license key authentication process.
  - The packaging must come with a manual that details the use of the system, and also the instructions on how to use the program. This manual may be included either in a booklet that comes with the software, or on the disc that the software itself is on.

- **3.3.6 Implementation**
  - The System User Interface is built on Microsoft Visual Studio 2010.
  - The Programing is done in Microsoft Visual Studio 2010.
  - The Database is implemented on the Microsoft Access 2010.

- The connection between the Database and the System is achieved using ODBC connection available at hand in Visual Studio 2010.

- **3.3.7 Interfacing**
  - The system must offer an easy and simple way of viewing the current inventory.
  - The system must be able to display the relationships between vendors, ingredients, and recipes in an intuitive manner.

- **3.3.8 Legal**
  - The software must be licensed on an individual basis for smaller companies, as well as through a multi-license deal for larger corporations.
  - The client should agree to EULA before using our software.

3.4 System models

3.4.1 Scenarios

- Add Pizza Recipe Scenario

Scenario Name	addPizzaRecipe
Initiating Actors	*manager*:Manager
Flow of events	1. There is a new menu item 'Pizza' offered by the restaurant and the item is not on the recipe database. Thus the manager initiates adding a new recipe to the database by pressing the add recipe button. 2. The *System* brings up the addRecipe form in front of the manager. 3. The manager fills up the details of the pizza recipe such as the bread used, toppings that will be used and the quantity of each one of the resource used is also entered by the manager. The manager also adds any new ingredient not already on the list of ingredients by invoking the add ingredient function on the system. 4. The manager submits the changes and the system adds the new Pizza recipe to the recipe database and any new ingredients to the ingredients database. 5. The Manager receives an acknowledgement of the completion of the process.

- Remove Pizza Recipe

Scenario name	removePizzaRecipe
Initiating actor	*manager:*Manager
Flow of events	1. The manager is going through the sales and notices that pizza sales are down and decides to remove Pizza as a menu item. 2. The manager activates the removes recipe function on the system. 3. The system brings out the list of recipes that are currently on the database. The manager selects the Pizza recipe to remove and submits the change. The System confirms if manager is sure to remove the pizza recipe from the menu items. 4. The manager confirms the change and the system acknowledges the removal of the recipe from the database.

- Add Vegetable Vendor Scenario

Scenario Name	addVegetablesVendor
Initiating Actors	*manager:*Manager
Flow of events	1. The *manager* wishes to add a new vegetables vendor to the list of vendors. To do this, the *manager* activates the addVendor function on the system. 2. The System presents a addVendor form to the *manager*. 3. The *manager* now adds the details of the vendor from whom vegetables will be obtained. The *manager* now adds the ingredients that can be obtained from this vendor, for example, potatoes, tomatoes, onions etc. 4.The *manager* submits the changes and an acknowledgement is received about the adding of the new vendor to the database

- Add Thanks Giving Scenario

Scenario Name	addThanksGivingOccasion
Initiating actors	*manager*:Manager
Flow of events	1. The *manage*r knowing that during the period of ThanksGiving the sales of certain menu items will go up the manager invokes the addSpecialOccasion function on the system. 2 The System now displays the addOccasion form to the *manager*. 3. The *manager* fills out the form by adding the name i.e. Thanks Giving and the corresponding dates in the form. 4. The *manager* now fills list of recipes that will be utilized more on Thanks Givings. 5. The *System* takes the data from the form and calculates the amount of ingredients that may be used up for the given dates based on past data and adds the data to the ingredient database. 6. The System now checks the current levels of resources against the new thresholds that are set after adding the new occasion and if the System finds that the some resources are below required levels, it will present to the manager a list containing the ingredients that are below threshold and their predicted usage of the ingredient. 7. The manager now agrees to process the order and thus the system processes the order and presents it to the *manager*. 8. If the *manager* is satisfied with the order quantity quoted by the system, *manager* will press activate the send out orders to vendors to send out the orders. 9.If the *manager* is not satisfied with the order, changes can be introduced in the order quotation before confirming the orders.

- Update Pizza Recipe Scenario

Scenario name	updatePizzaRecipe
Initiating actors	*manager*:Manager
Flow of events	1. The *manager* activates the updateRecipe function on the system. 2. The system now brings out a list of recipes to be selected by the *manager*. 3. The *manager* now selects Pizza from the list of recipes and chooses to update it. 4. The *manager* is now presented with a form that presents the current properties of recipe. The *manager* now add/removes ingredients, their usage etc. and then submit the changes. 5. The system now updates the recipe database with the new properties and acknowledges the changes to the *manager*.

- Update Resource Database Scenario

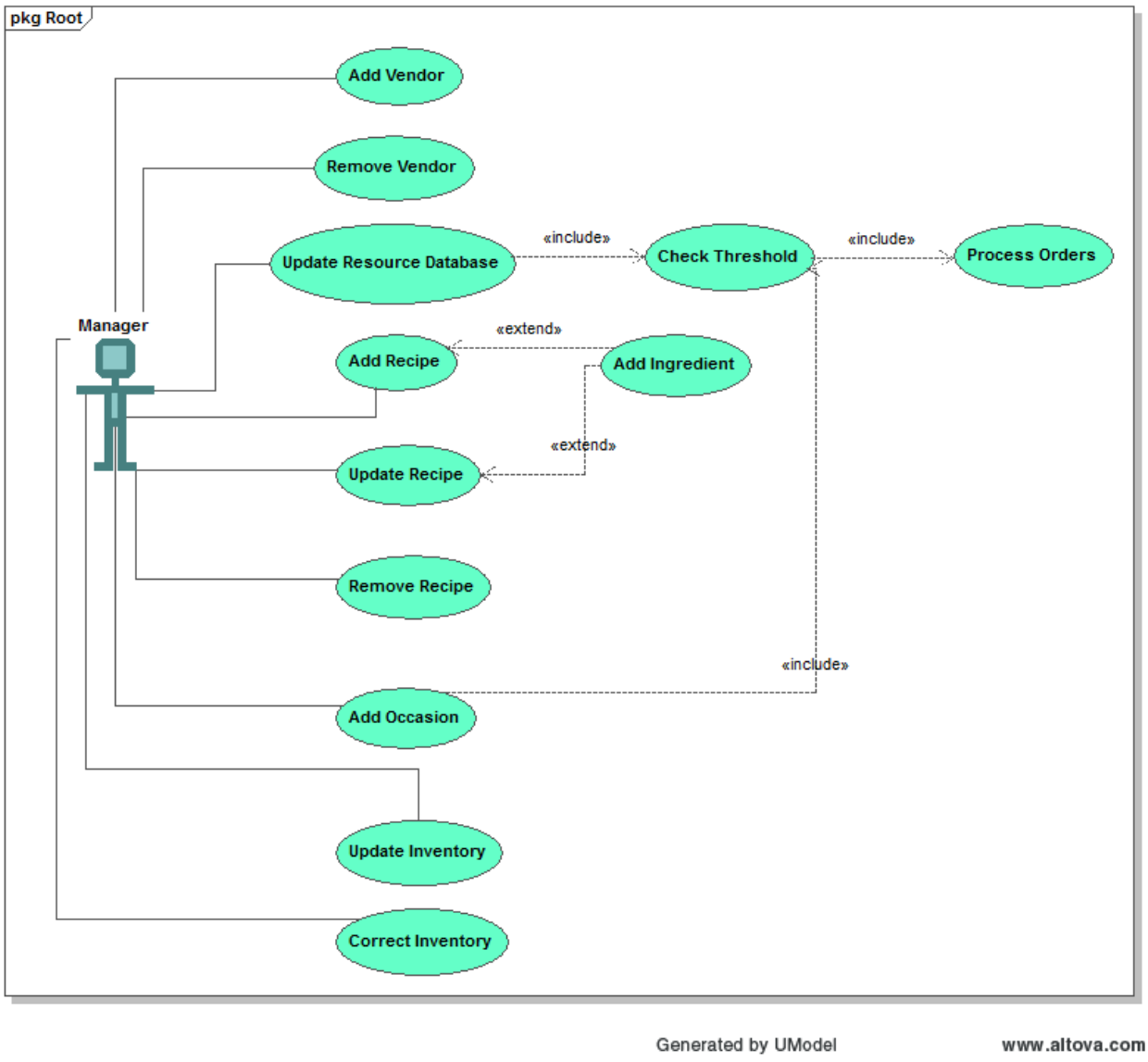
Scenario name	Update Resource Database at the end of day.
Initiating actors	*manager:*Manager
Flow of events	1. The manager activates the update resource database function. 2. The manager is presented with a sold food form and the manager enters the menu items that were sold and also the corresponding quantities that were sold in the day. 3.The manager submits the form and the system deducts the quantities of ingredients that were used up in making those menu items. The system now checks for threshold levels and if it finds them to be above threshold levels, tells the manager that all the ingredients are above threshold and sufficient resources are available for usage.

- Correct Inventory Scenario

Scenario name	Correct Inventory
Initiating actor	*manager*:Manager
Flow of events	1. The *manager* feels that there is some difference between the inventory levels shown by the system and the actual inventory levels. The *manager* initiates the check inventory function on the system. 2. The *manager* is presented with a form listing all the ingredient and their corresponding levels. 3. The *manager* now conducts a survey of the actual inventory and matches them with the levels shown by the system. if any discrepancies are found, the manager updates the new quantity next to the box that is presented next to the ingredient name on the form. 4. After updating any inventory discrepancies, the manager submits the changes and the inventory is updated accordingly. 5. The threshold levels are also updated keeping in mind the errors that were incurred in calculations earlier and the manager receives an acknowledgement of process completion.

3.4.2 Use case model

Use case diagram



Use cases:

- Update Resources Database Use Case

Usecase name	**UpdateResourceDatabase**
Participating Actors	Initiated by *Manager(admin)*
Flow of events	1. The *Manager* activates the update resource database function. 2. The System presents a form to the *Manager*. The form asks for details of the sold food items during the course of the week and the corresponding quantity of the food sold. 3. The *Manager* inputs the data of the sold food for the week and the quantity that was sold and presses Ok button. 4. 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. 5. The *System* now calculates the amount of resources used and will deduct the amount of ingredients that were used up from the resource database. 6. The *System* now invokes the CheckThreshold usecase.
Entry condition	The *Manager(admin)* is logged on to the *System*
Exit condition	If the process was successful, the *Manager* 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.

Entity Objects

1. Manager: The user who initiates and manages the whole use case. The Manager enters the sold food data into the form that is presented to him and also controls what actions are to be taken once all the calculations are over.

2. Resources: These are the ingredients that are available for use in making of recipe.

Boundary Objects

1. Update Resource Database Button: This is the Button the manager presses to initiate the processes in the use case

2. Sold Food Form: This is the form Manager fills to give details about the food that was sold in the course of the week.

3. Process Order Form: In this form manager enters the quantities corresponding to each ingredient that manager wants to update.

Control Objects

Update Control Object: This object is created when the manager activated the UpdateResourceDatabase Function. The Object is then responsible for creating the SoldFood Form and requesting resource levels from the database, comparing new values with the threshold values and also sending the new values back to database.

The Object also sends out acknowledgements to the Manager once the requested processes are completed.

- Check Threshold Use Case

Use case name	**CheckThreshold**
Participating actors	Initiated by UpdateResourceDatabase usecase Or by AddOccasion usecase
Flow of Events	1. The *System* now compares the current levels of the resources with the threshold levels of the resources. It now lists all the ingredients that are below the threshold level, along with the predicted usage of the ingredients and presents it to the *Manager*. 2. The *Manager* can now send out orders by pressing the Process order button. This action invokes the processOrder usecase. 3. If *Manager* presses cancel, no orders are processed.
Entry Conditions	The manager is logged on to check the inventory levels at the interval of a certain time period.
Exit Conditions	The inventory levels are checked and the appropriate action is taken.
Quality Requirements	The system correctly calculates the correct threshold differences

Entity Objects

1. Resources: These are the ingredients that are available for use in making of recipe.

2. ThresholdLevels: These are the ingredient levels below which an alert is generated for the manager to generate new orders.

Boundary Objects:

processOrder: This button invokes the processing of orders for the ingredients that are below threshold levels.

Control Object:

Update Control Object: This object is created when the manager activated the UpdateResourceDatabase Function. The Object is then responsible for creating the SoldFood Form and requesting resource levels from the database, comparing new values with the threshold values and also sending the new values back to database. The Object also sends out acknowledgements to the Manager once the requested processes are completed.

- Process Order Use Case

Use case name	**ProcessOrder**
Participating Actors	Initiated by the *Manager*
Flow of events	1. The *System* now gathers a list of vendors from whom the corresponding ingredients are ordered. The *System* now matches the corresponding ingredients with the vendors from whom ingredients are available. Loop 2. The *System* now creates an order and then presents order summary form to the *Manager*. The form has three options on it, one to approve the order, the other to revise the order and one to cancel the order] 3. The *Manager*, at this point can approve the generated order. The *Manager* can do this by pressing the approveOrder button on the order summary form. This will send the generated orders to the vendors. The *Manager* receives the acknowledgment of the reception of the order and the process ends.] 4. The *Manager* can also choose to revise the order and enter the quantities to be ordered manually for every corresponding ingredient. The *Manager* can do so by choosing the reviseOrder button. An updated order summary form is presented to the *Manager* and the flow returns back to point 2]. 5. The *Manager* can also choose to cancel the order by pressing the cancelOrder button. In this case the generated order is cancelled and no orders are sent out.]
Entry Condition	The user is logged into the system

Exit Conditions	1. The *Manager* approves the order and receives an acknowledgement of the orders being sent. 2. If the orders are not sent out successfully after pressing the approveOrder button, the *Manager* receives a message that the orders were not sent out. 3. The *Manager* presses the cancelOrder button.
Quality Requirements	The order is sent to the correct vendor

Entity Objects:

1. Manager: The manager activates the use case to add a new recipe to the database. He is responsible for adding ingredients to the recipe.

2. Ingredient: The ingredients which are below the threshold level are procured as a list and the same needs to be approved by the manager.

3. Vendor: The generated order is sent to the vendor as a request for consignment.

Boundary Objects:

Process Order: This button is used when the generated order form is approved by the manager and the following is sent to the vendor.

Cancel Order: Cancel order button is used if the orders are not generated properly. If the manager is not satisfied by the order form then cancel order is used.

Revise order: The revise order enables changes to be made in the current generated order form. The manager then edits the quantities accordingly.

Control Objects:

Process Order Control: This control object manages the whole process of producing a purchase order, revising the purchase order and cancelling the current purchase order. It contains a list of ingredients that are needed from the vendor .Only when the manager approves the final list the vendor shall receive the ordered form.

- Add Recipe Use Case

Usecase name	**AddRecipe**
Participating Actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates the "Create New Recipe" function on his/her terminal 2. The System responds by presenting a form to the *Manager*. The form asks for details associated with the recipe. 3. The *Manager* completes the form by inserting ingredients to be used in the new recipe. It also adds any new ingredient used in the recipe by executing the addIngredient usecase which extends this usecase. The *Manager* also inputs the amount of ingredient to be used in a single order of the recipe. After the form has been completed the *Manager* 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 and any new ingredient to the ingredient database.
Entry condition	The *Manager* is logged into *System*
Exit condition	The *Manager* has received an acknowledgment from the *System*. OR The *Manager* has received an explanation of why the process couldn't be completed.
Quality Requirements	This use case is extended by the **AddIngredient** use case. The process must complete successfully with the new recipe added to the recipe database without any errors.

Entity Objects:

1. Manager: The manager activates the use case to add a new recipe to the database. He is responsible for adding ingredients to the recipe.

2. Recipe: This is a new recipe that is being added to the database by the manager. The recipe is defined by the items that are used in making of the recipe.

3. Ingredients: The ingredients are the items that the recipe is made of.

Boundary Objects:

Add Recipe button: This button initiates the add recipe use case which helps in adding a new recipe to the database.

Add Recipe form: This form asks the manager for details of the recipe in terms of the ingredients that are required and the quantity that is required per ingredient.

Add Ingredient Button: This button initiates the addIngredient use case. This is initiated in between the addRecipe usecase if any ingredient is not available to the manager while adding the recipe and the manager wishes to add that specific ingredient to the recipe.

Control Objects:

Add Recipe Control: This control object manages the whole process of adding the new recipe to the database. It manages the process of adding the new recipe to the database, the new ingredient to the ingredient database and also acknowledgement of the process completion to the manager.

- Update Recipe Use Case

Usecase name	**UpdateRecipe**
Participating Actors	Initiated by *Manager*
Flow of events	1. The *Manager* activates "Update Recipe" on system. 2. *System* responds by bringing up list of recipes. 3. The *Manager* selects a recipe to change. 4. *System* now shows a updateRecipe form with the list of ingredients in the recipe and corresponding amount. 5. The *Manager* changes the recipe by adding/removing ingredients or updating the amount of ingredients used in the recipe. The *Manager* can also add new ingredients that are not available currently by invoking the AddIngredient usecase. 6. The *Manager* then finishes the update by pressing the finish button on the system. 7. The *System* confirms that the change has been made and updates the databases.
Entry condition	The *Manager* is logged into *System*
Exit condition	The *Manager* receives an acknowledgment from the *System*, OR The *Manager* has received an explanation of why the process couldn't be complete.
Quality Requirements	1) This usecase is extended by the **AddIngredient** use case. 2) The update process must be complete successfully without any errors.

Entity Objects:

Manager: The manager activates the use case to update a recipe. The manager is responsible for adding/removing ingredients to and from the recipe.

Recipe: This is an entity that is being updated by the manager. The recipe is defined by the items that are used in making of the recipe.

Ingredients: The ingredients are the items that the recipe is made of.

Boundary Objects:

Update Recipe Button: This button initiates the add recipe use case which helps in adding a new recipe to the database.

Update Recipe Form: This form asks the manager to input any changes that are to be made to the recipe.

AddIngredient: This button initiates the addIngredient use case. This is initiated in between the add recipe usecase if any ingredient is not available to the manager while updating the recipe and the manager wishes to add that specific ingredient to the recipe.

Control Objects:

Update Recipe Control: This control object manages the whole process of updating recipe. It manages the process of adding/removing the ingredients from the recipe. It also acknowledges the process status to the manager.

- Remove Recipe Use Case

Usecase name	**RemoveRecipe**
Participating actors	Initiated by *Manager*
Flow of events	1. The *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 *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 *Manager* if he/she wants to delete the recipe. 5. The *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.
Entry condition	The *Manager* is logged in System
Exit condition	The *Manager* has received an acknowledgment that the recipe has been deleted. OR The *Manager* has received an acknowledgment that the recipe has not been deleted. OR The *Manager* has received an explanation of why the process couldn't be completed.
Quality Requirement	The removed recipe should reflect in any other list or connected database.

Entity Objects:

Manager: The manager activates the use case to remove a recipe.

Recipe: This is an entity that is being removed by the manager. The recipe is defined by the items that are used in making of the recipe.

Boundary Objects:

RemoveRecipe: This button initiates the add recipe use case which helps in adding a new recipe to the database.

Remove Recipe Form: This form presents a list of recipes currently in the database and asks the manager to select which recipe is to be deleted.

Control Objects:

Remove Recipe Control: This control object manages the whole process of updating recipe. It manages the process of adding/removing the ingredients from the recipe. It also acknowledges the process status to the manager.

- Add Occasion Use Case

Usecase 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 *Manager* now fills list of recipes that will be utilized more on the selected day. 5. The *System* takes the data from the form and calculates the amount of ingredients that may be used up for the given dates based on past data and adds the data to the ingredient database. 6. The *System* now invokes the CheckThreshold use case.
Entry condition	The *Manager* is logged into System.
Exit condition	The *Manager* receives a notification of successful completion of the process 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.

Entity Objects:

Manager: The manager activates the use case to add an occasion to the database.

Recipe: This is an entity that is being added here for special use on the day of the occasion. The recipe is defined by the items that are used in making of the recipe.

Ingredients: Ingredients are the entities that the recipe is made up of.

Boundary Objects:

Add Occasion Button: This button initiates the addOccasion use case which helps in adding a new occasion to the database.

Add Occasion Form: This form asks for details about the occasion that is to be entered in the database. Details such as date, recipes that are sold more etc. are added as occasion details

Control Objects:

Add Occasion Control: This control object manages the process of adding the new occasion to the database and thus re calculating new threshold levels and resources requirements for the manager to consider.

- Update Inventory Use Case

Usecase 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* now presents a form to the *Manager* asking for details of the received amount of ingredients. 3. The *Manager* enters the ingredients and the corresponding quantity received and presses the submit button. 4. The *System* adds the corresponding amount to the resources database and acknowledges the completion of the process.
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.

Entity Objects:

Manager: This entity initiates the use case to update the stock inventory once the new order is received.

Ingredients: The ingredients are the basic building blocks of the recipe. Here the ingredient delivery is accepted in accordance with the order issued for the ingredients and updated in the ingredients database.

Boundary Objects:

Update Inventory Button: This button when pressed by the manager brings up a form for the manager to add the quantity of ingredients that are received in the order.

Update Inventory Form: This form asks the details about the ingredients that were received and the quantity that was received of the respective ingredient.

Control Objects:

UpdateInventory Control: This object manages the whole updating ingredient database after reading the sold received food data from the form.

- Correct Inventory Use Case

Use case name	**CorrectInventory**
Participating actors	Initiated by *Manager*
Flow of Events	1. The *Manager* presses the "Correct Inventory" button on the console. 2. The *System* presents a CorrectInventory form to the *Manager* with the list of ingredients and corresponding remaining quantity. 3. The *Manager* now enters the corrected quantity (if any corrections exists) corresponding to each ingredient and presses the submit button. 4. The *System* now updates the resources database with the correct quantity.The *System* calculates the errors that were existing in the original and the corrected values of the resources and accordingly adjusts the value of quantity of ingredients used per recipe. It then prints out the correct inventory to the screen. 5. The *Manager* then acknowledges that the calculated inventory is accurate.
Entry Conditions	The *Manager* is logged into the system
Exit Conditions	The Inventory level is correctly updated
Quality Requirements	The data taken from the *Manager* is accurately stored into the database.

Entity Objects:

Manager: This entity initiates the use case to correct the inventory.

Ingredient: This is the entity whose quantity is updated in the process of the correcting.

Boundary objects:

Correct inventory button: This is the button that the manager presses to initiate the correction process.

Control Objects:

UpdateInventory Control: This object manages the whole process of correction. It updates the database with the correct data that is provided by the manager.

- Add Vendor Use Case

Usecase 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* for 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. 4. 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 added.
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

Entity Objects:

Manager: This entity initiates the use case and adds details of the vendor to be added.

Vendor: This is the entity that supplies certain ingredients to the restaurant.

Boundary Objects:

Add Vendor Button: This is the button the Manager presses to initiate the adding of vendor to the database.

Add Vendor Form: This is the form that the manager fills out to add the details of the vendor to the database. Details such as the vendor name, the ingredients available from the vendor and the shipping time required by the vendor for the requested ingredients is mentioned in the details.

Control Objects:

Add Vendor Control: This control object manages the process of presenting the add vendor form to the manager, reading the entered data and updating the database with the new data.

- Remove Vendor Use Case

Usecase 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

Entity Objects:

Manager: This entity initiates the use case to remove the vendor from the list of vendors.

Vendor: This is the entity that supplies certain ingredients to the restaurant.

Boundary Objects:

Remove Vendor Button: This is the button the manager presses to initiate the use case.

Remove vendor form: This is the form which presents the list of vendors that are currently in the database and asks the manager to select the vendor to be removed from the database.

Control Objects:

Remove Vendor Control: This is the control object that facilitates the process of removal of the selected vendor from the vendor database.

- Add Ingredients Use Case

Use case name	**AddIngredient**
Participating actors	Initiated from the AddRecipe use case
Flow of Events	1. The *System* presents a form to the *Manager* for adding the new ingredient. 2. The *Manager* inputs the details of the ingredient including the vendor from whom the recipe is available. If the vendor is not currently part of the current database, the *Manager* has to add a new vendor via the AddVendor use case. He/She then confirms the details of the ingredient and presses the "Add" button 3. The *System* now makes available the new ingredient to the *Manager* for including it in the recipe.
Entry	The AddRecipe function is currently running
Exit	The Ingredient is successfully added to the database
Quality Requirements	The details for the ingredient are correctly added to the correct database.

Entity Objects:

Manager: The manager activates the use case to add an ingredient.

Ingredient: This is an entity that is being added by the manager. These are considered as atomic substances which are used up in clusters in recipes.

Vendor: The vendor from whom we are purchasing the particular ingredient also needs to be inputted.

Boundary Objects:

Add Ingredient: This button initiates the add ingredient use case which helps in adding a new ingredient to the database. This is linked to the vendor providing the consignment.

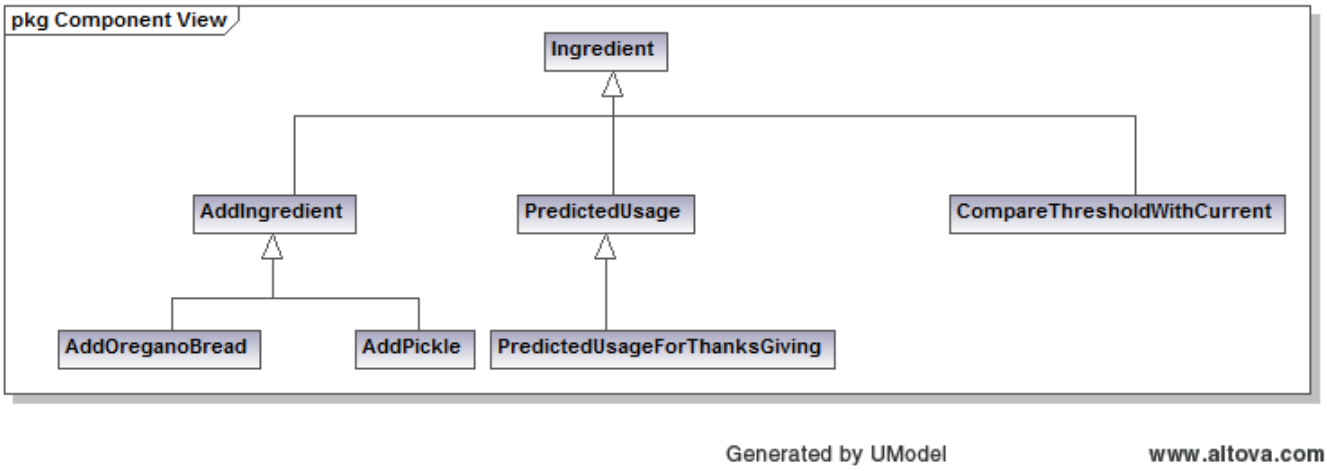
Control Object:

Add ingredient control: This control object enables the manager to input the ingredient and also the providing vendor; this is used to link it with the vendor so as to when the threshold levels are reached then they can be replenished.

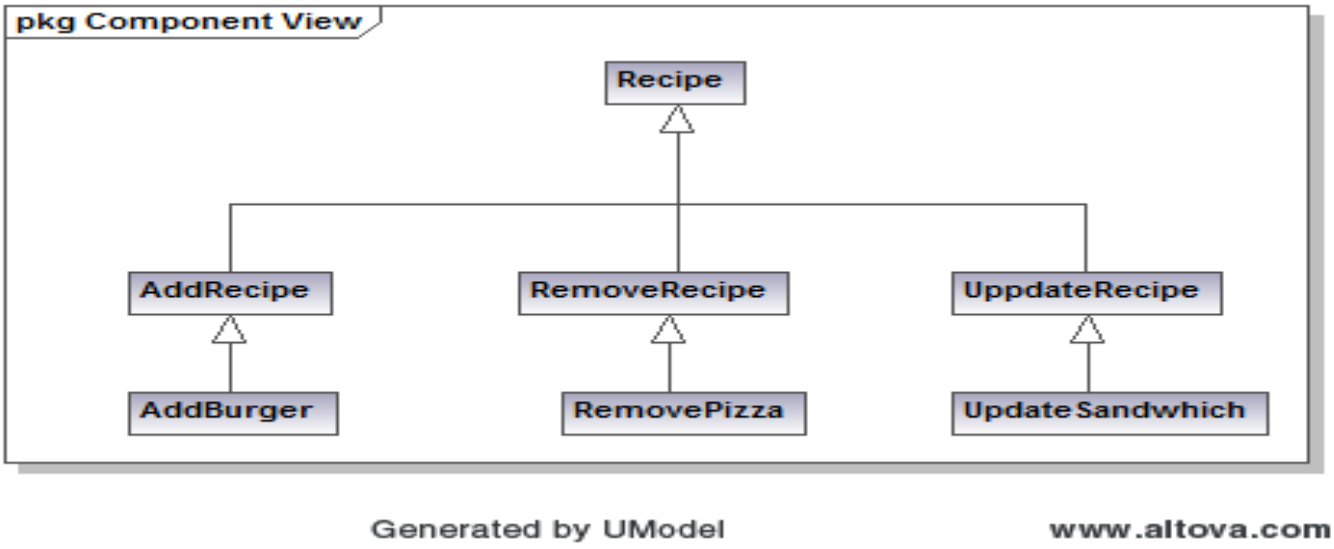
3.4.3 Object model

3.4.3.2 Class Generalization and hierarchy diagrams

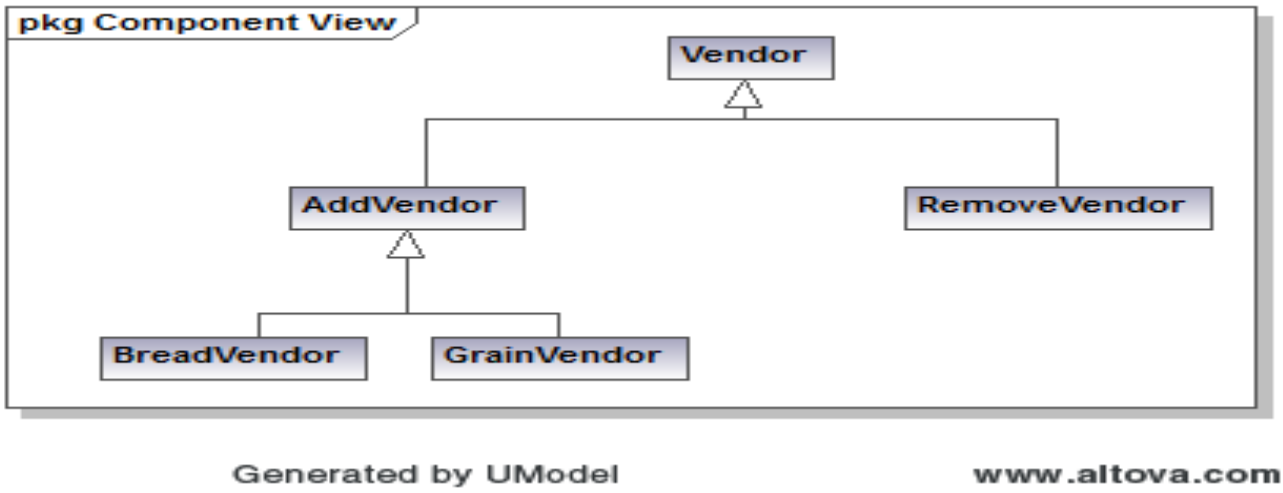
- Ingredient Class Generalization and hierarchy diagram



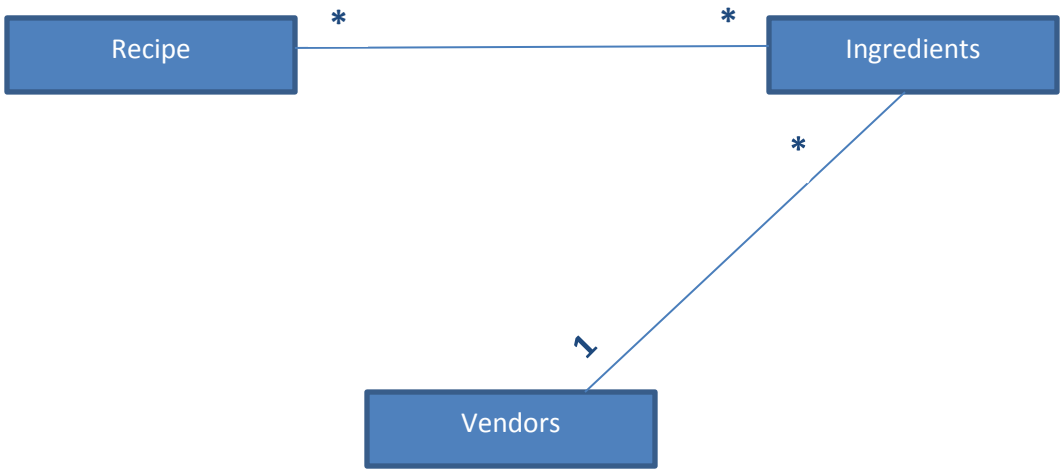
- Recipe Class Generalization and hierarchy diagram



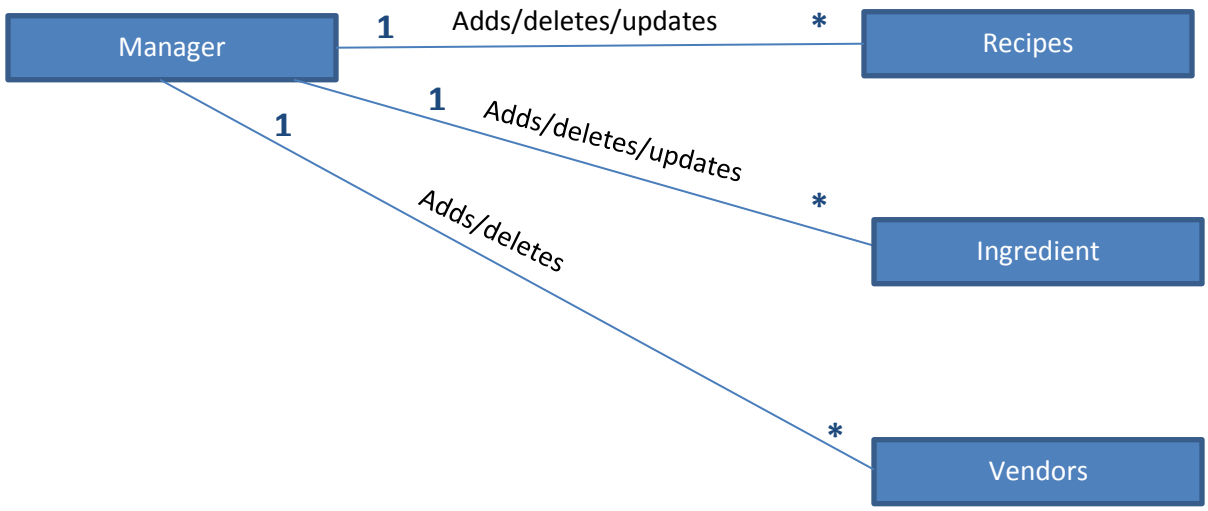
- Vendor Class Generalization and hierarchy diagram.

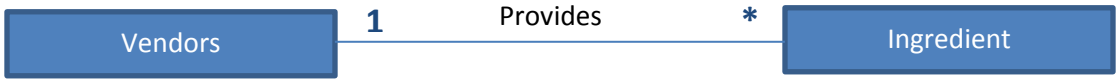


3.4.3.2 Multiplicity Diagram



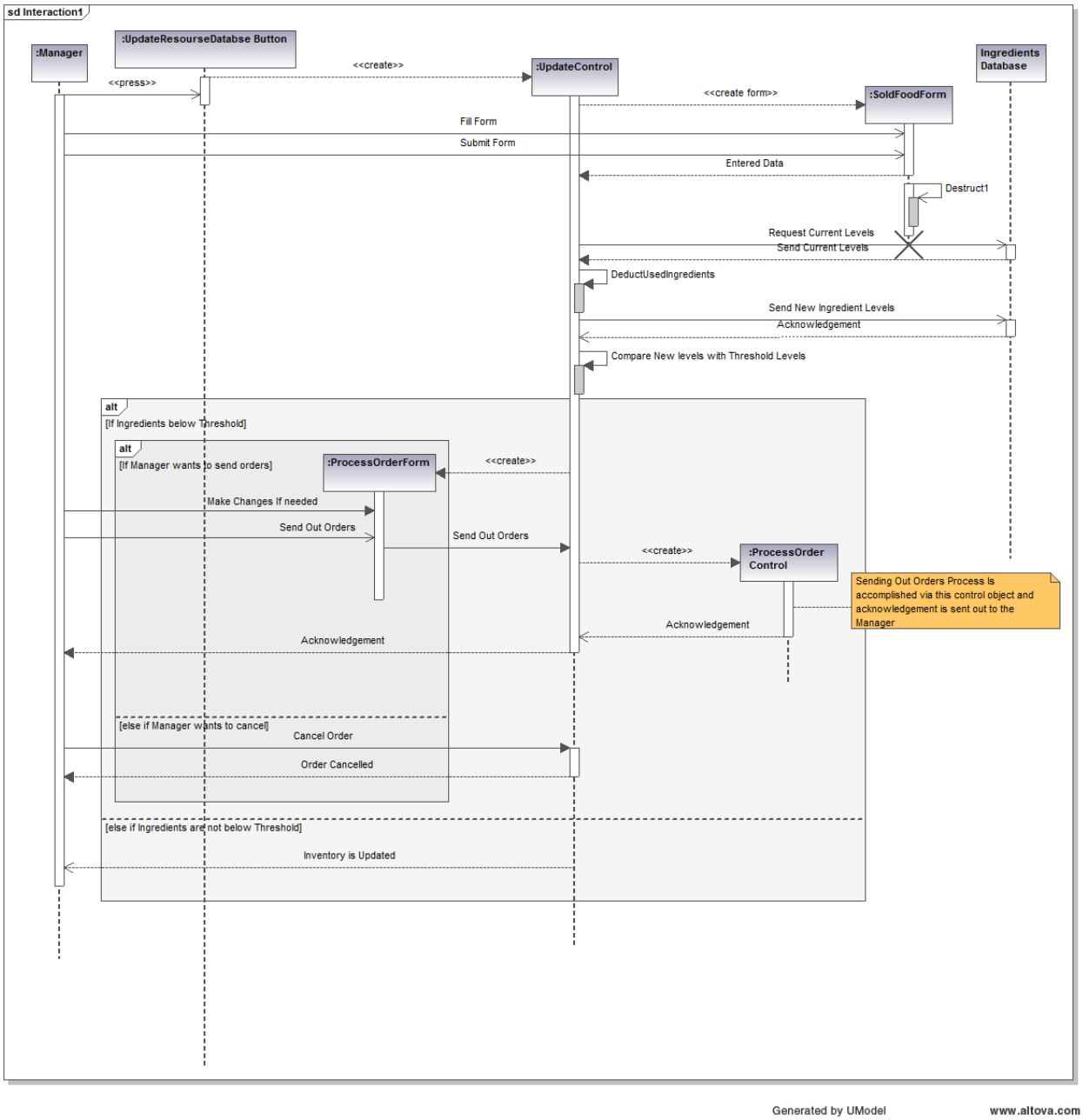
3.4.3.3 Association Diagram



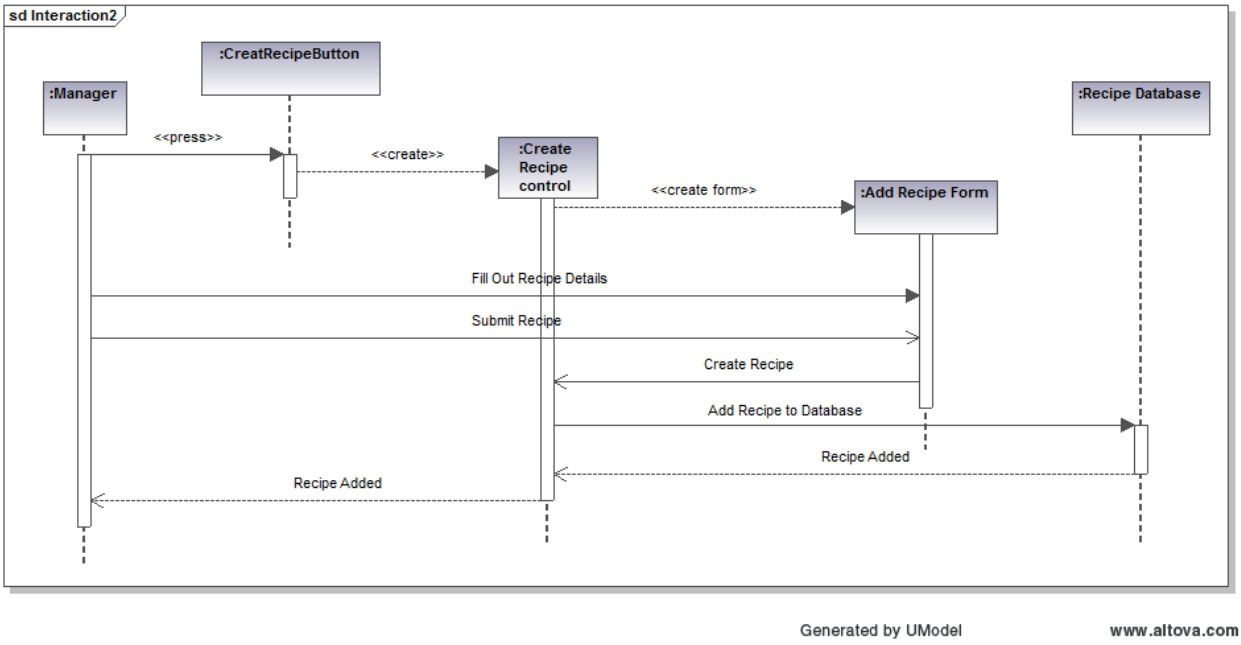


3.4.4 Dynamic model

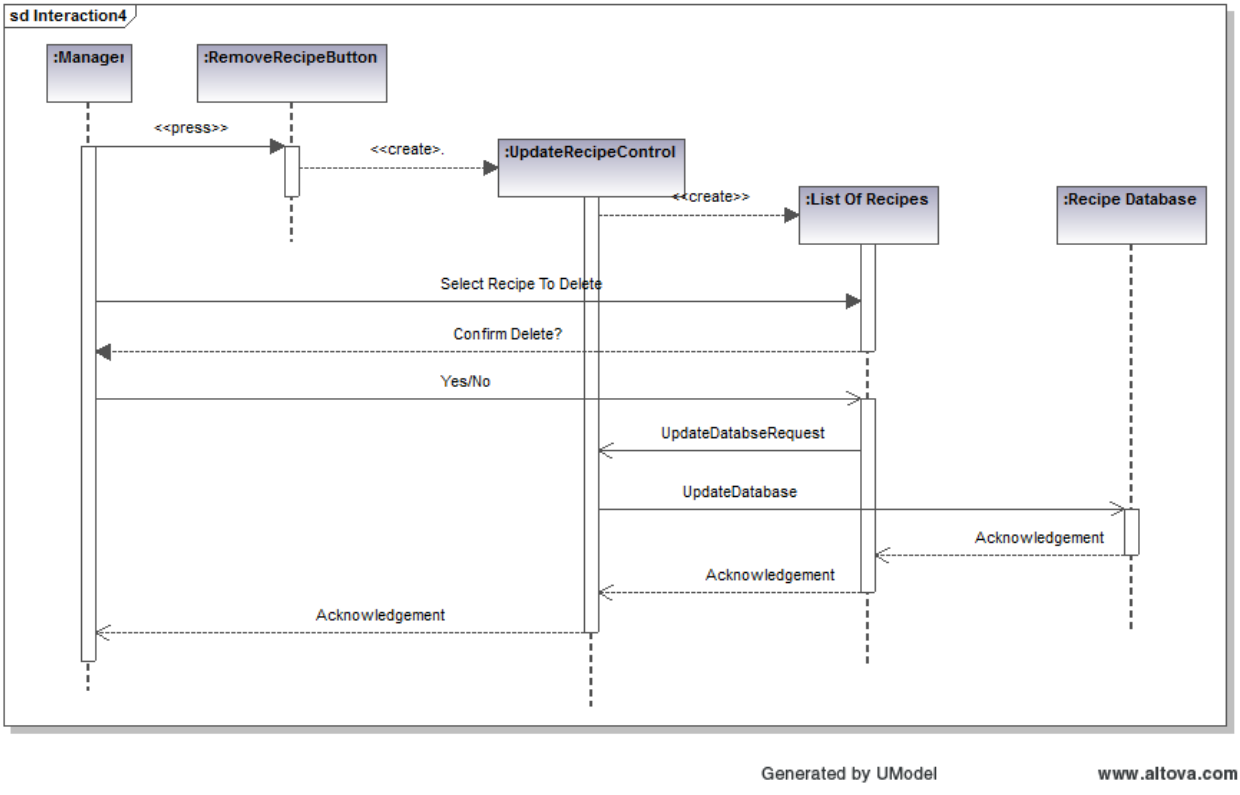
- Update Resource Database Sequence Diagram



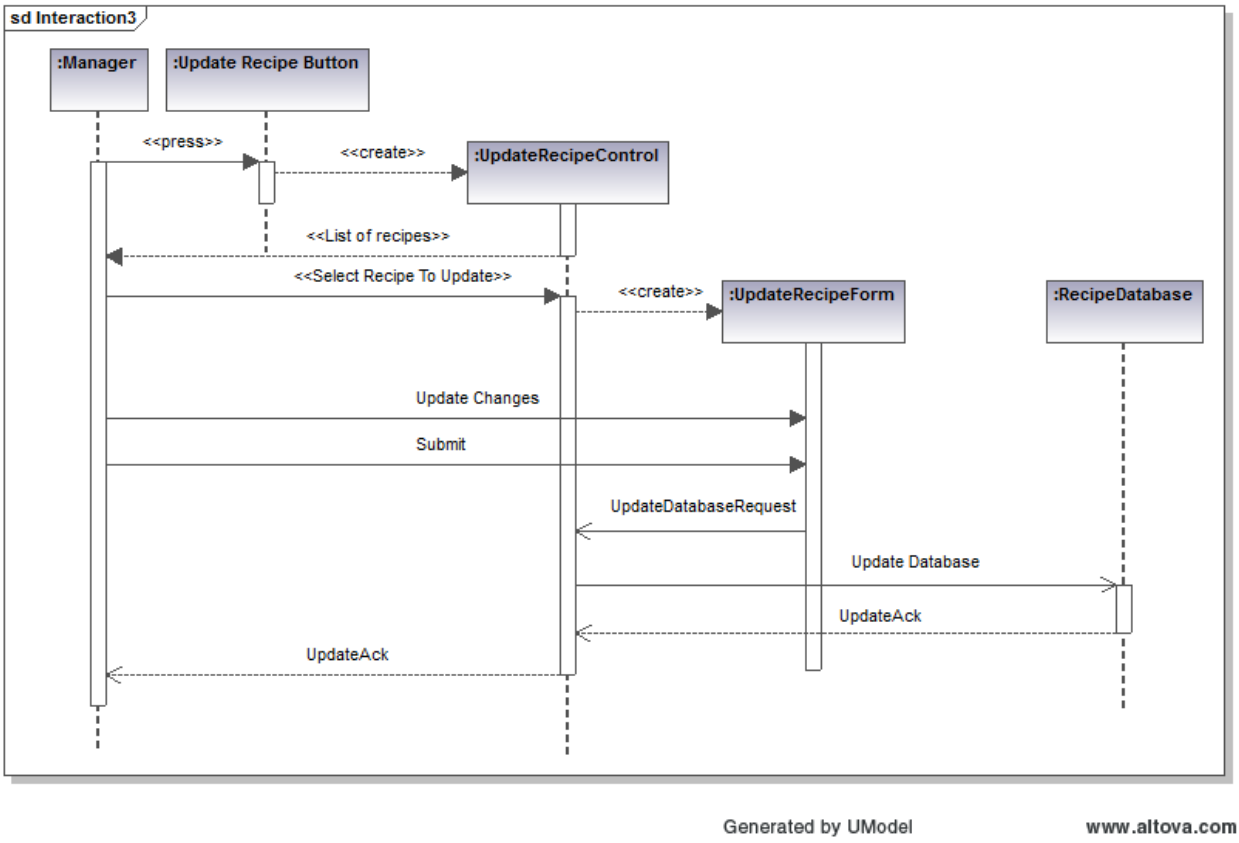
- Add Recipe Sequence Diagram



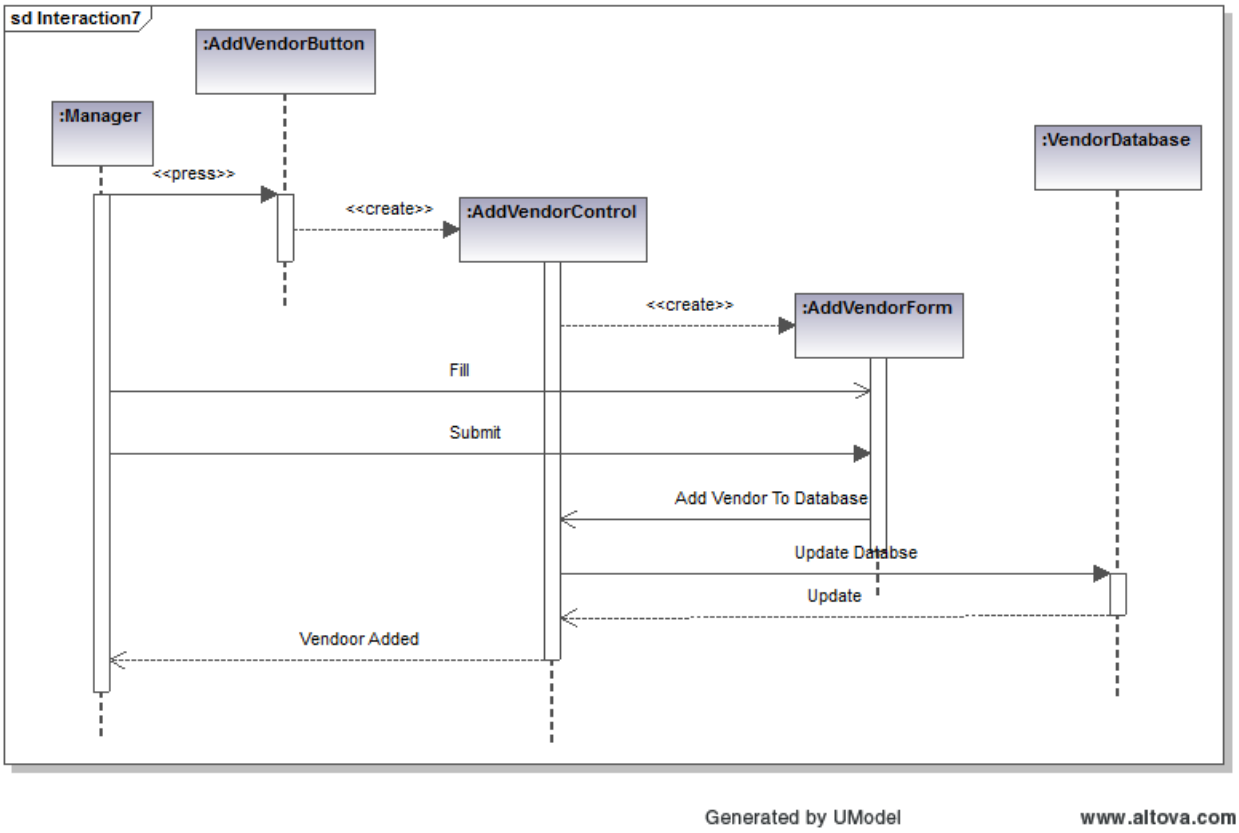
- Remove Recipe Sequence Diagram



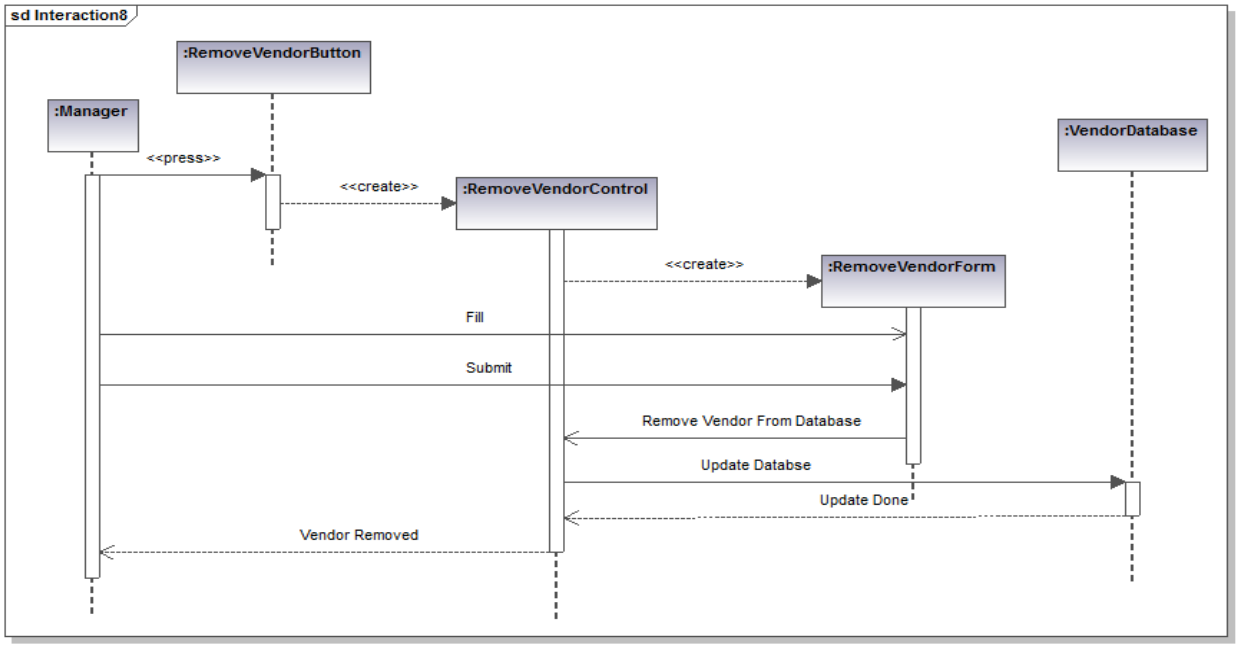
- Update Recipe Sequence Diagram



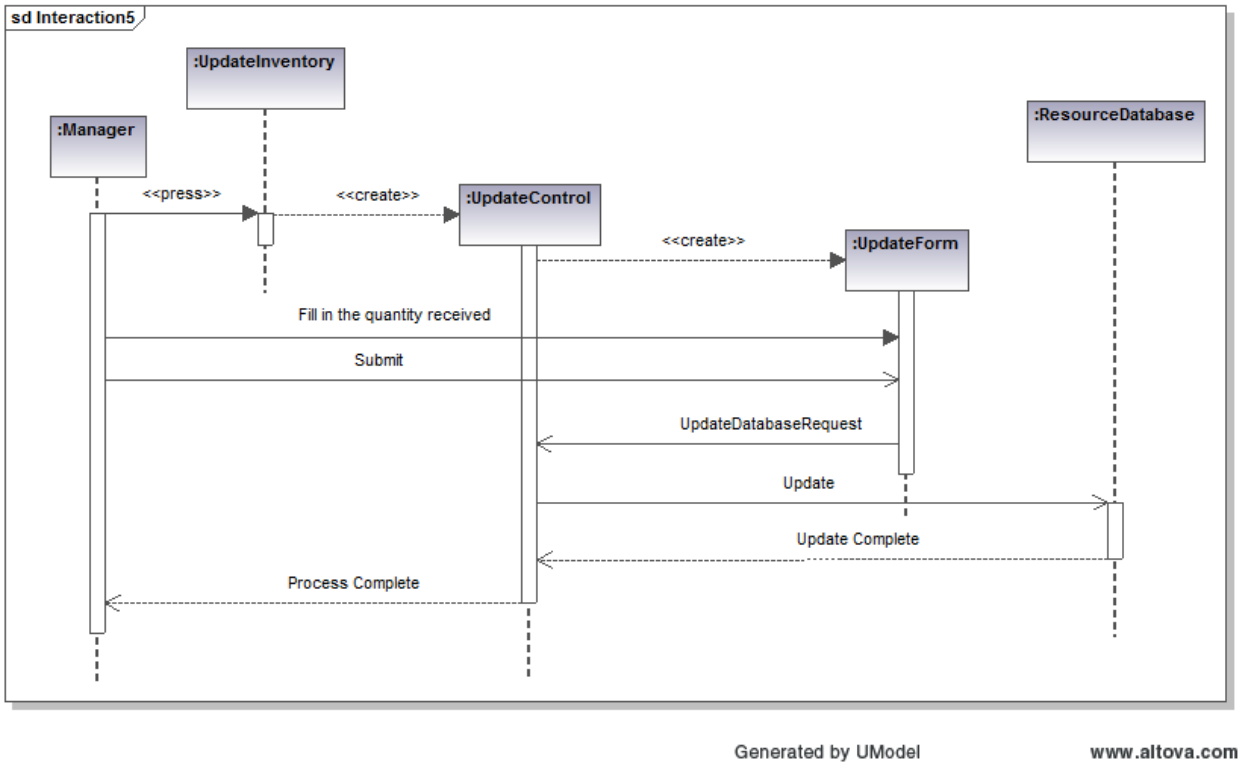
- Add Vendor Sequence Diagram



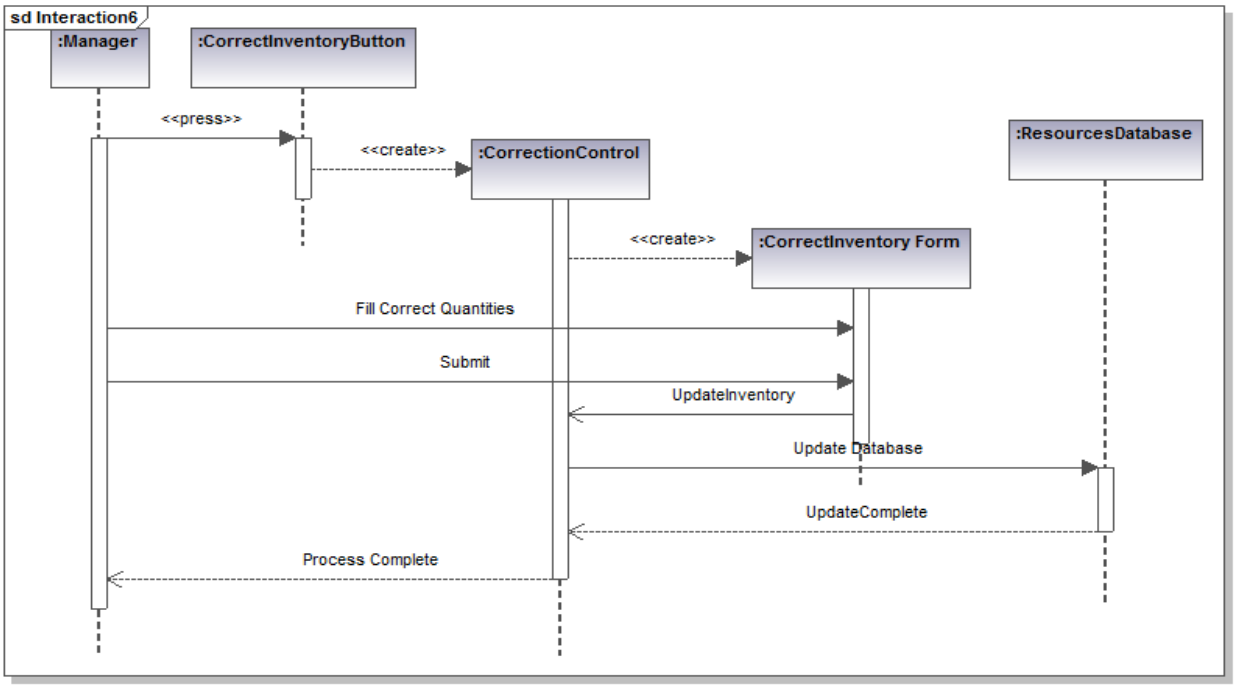
- Remove Vendor Sequence Diagram



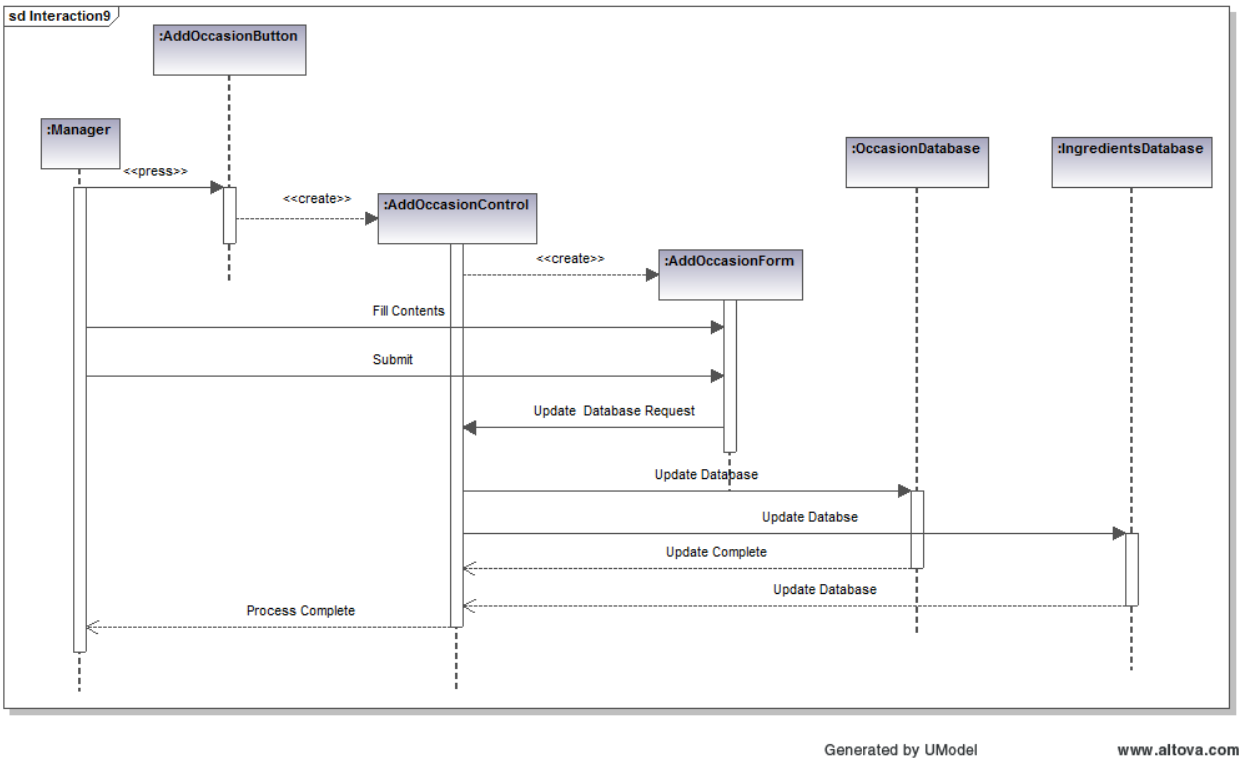
- Update Inventory Sequence Diagram



- Correct Inventory Sequence Diagram

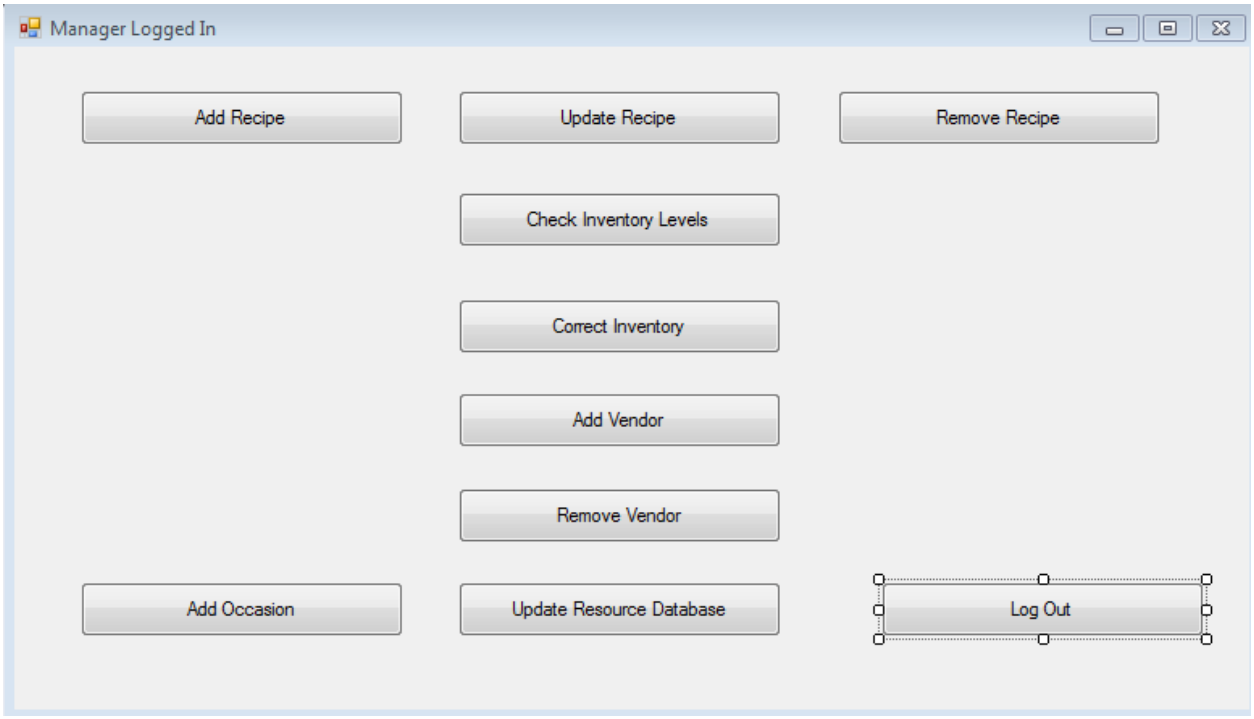


- Add Occasion Sequence Diagram

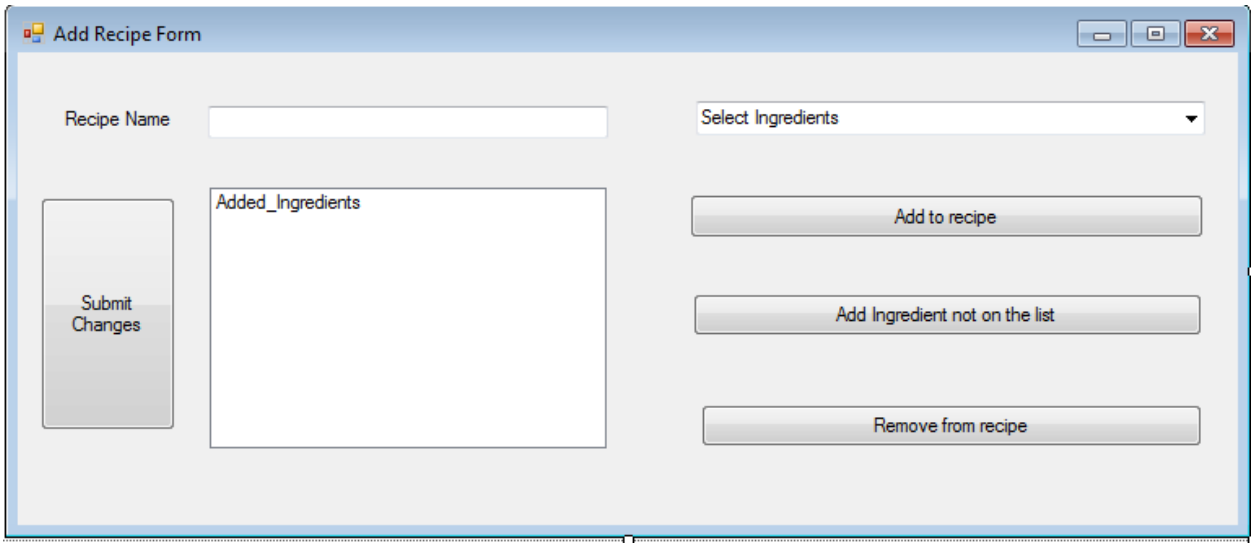


3.4.5 User interface—navigational paths and screen mock-ups

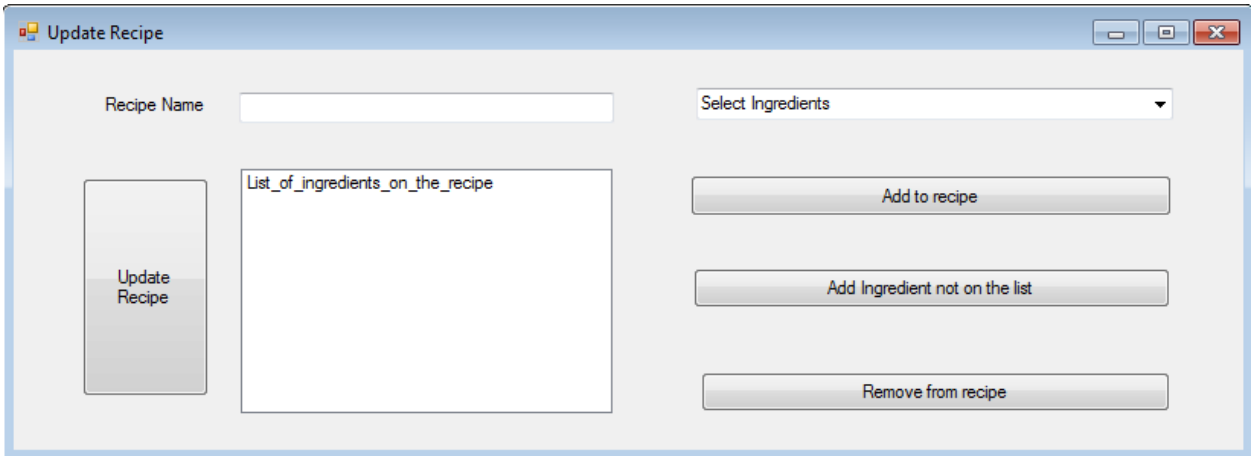
- Main Screen



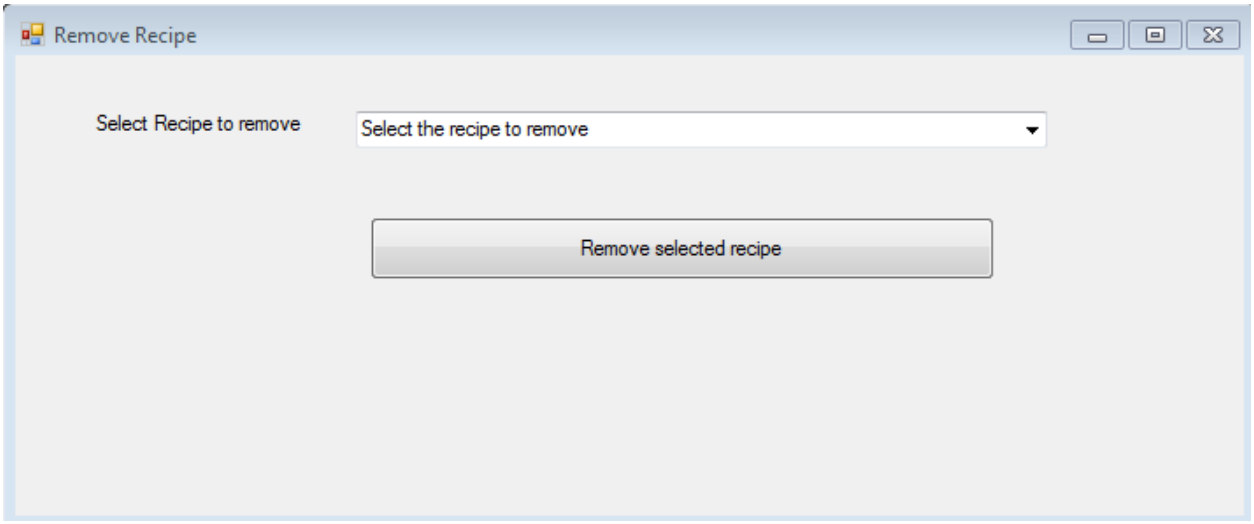
- Add Recipe Screen



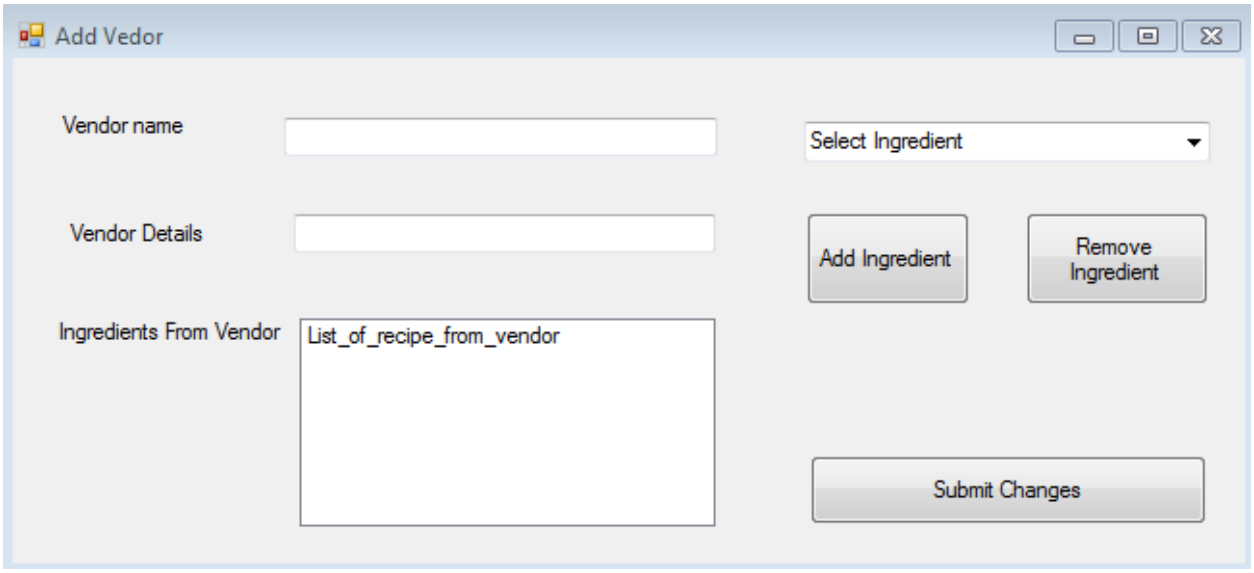
- Update Recipe screen



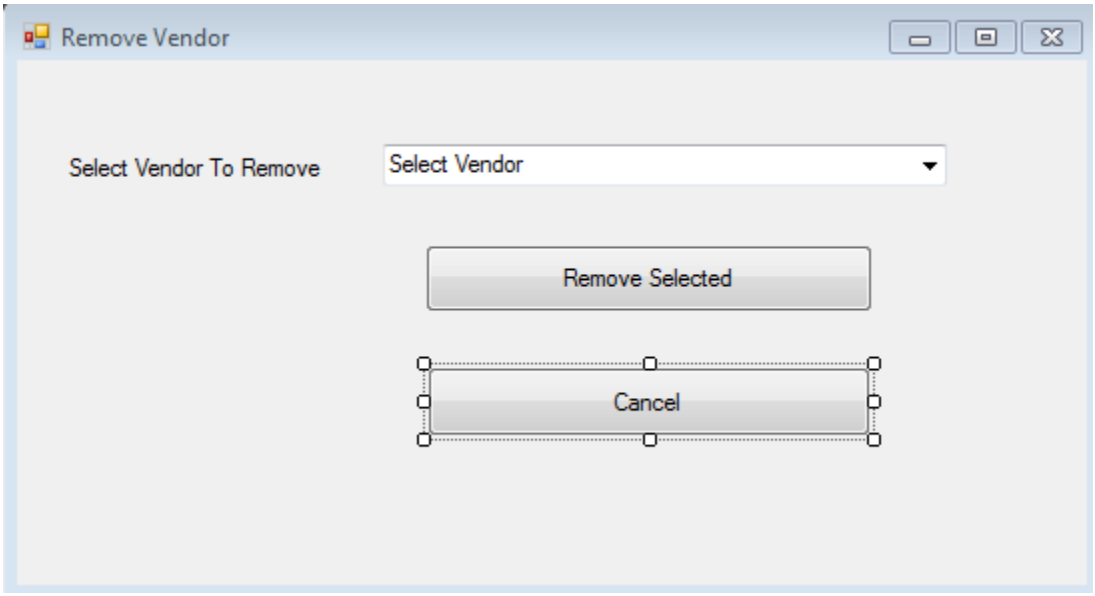
- Remove Recipe Screen



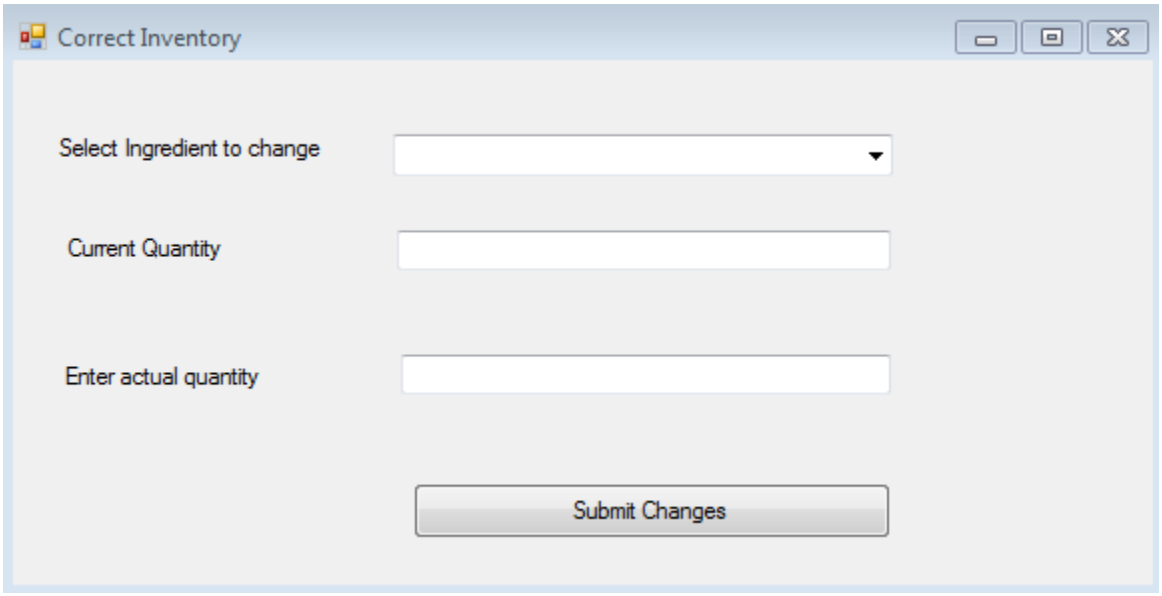
- Add Vendor screen



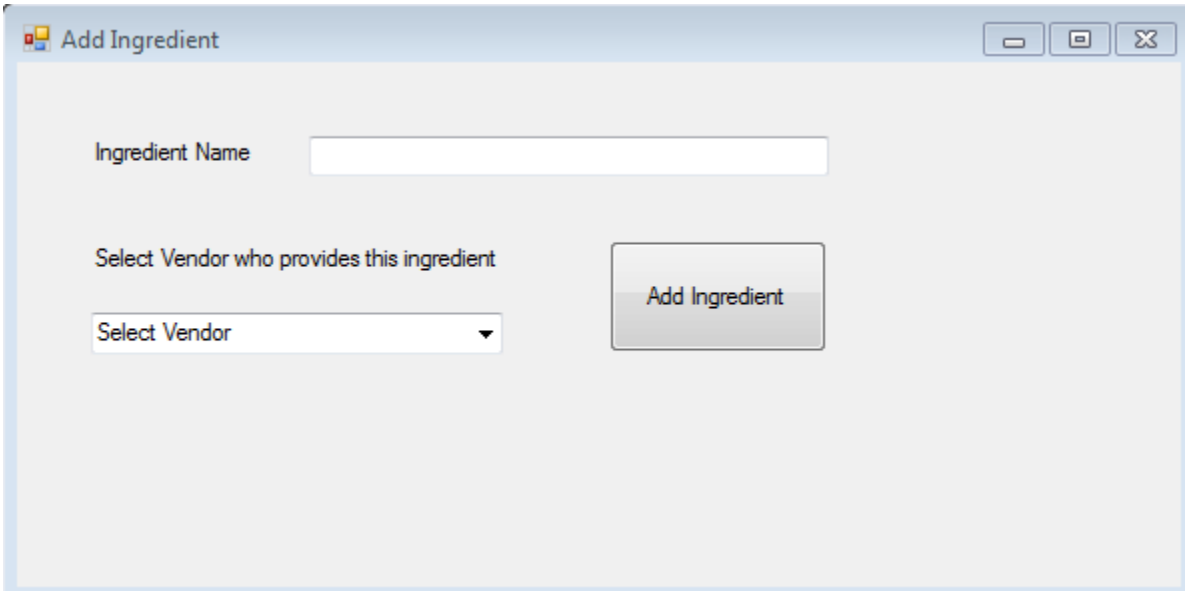
- Remove Vendor screen



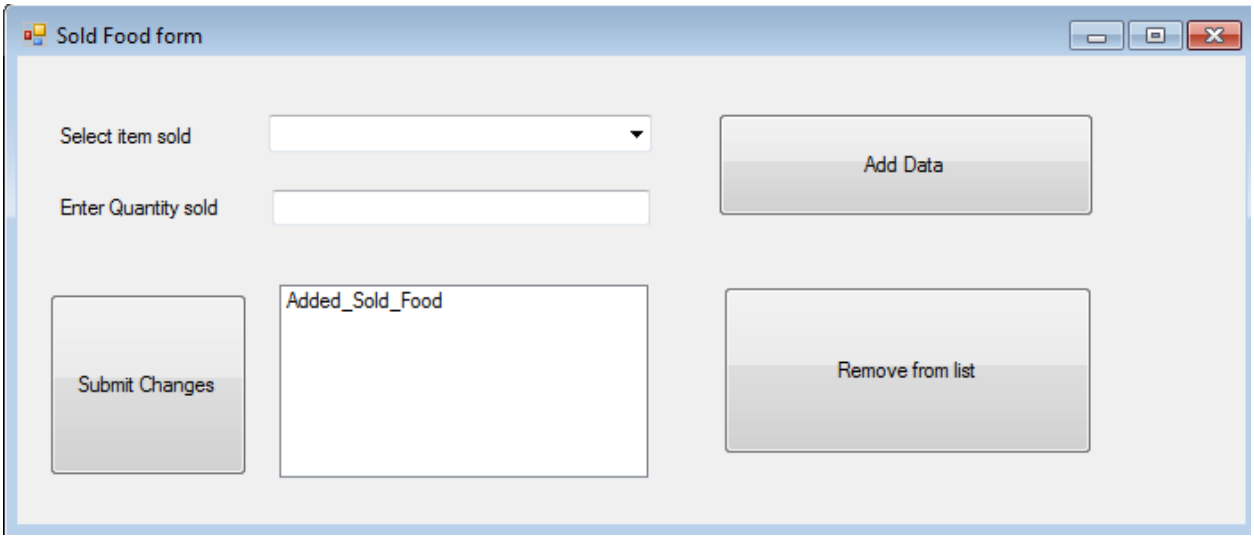
- Correct Inventory Screen



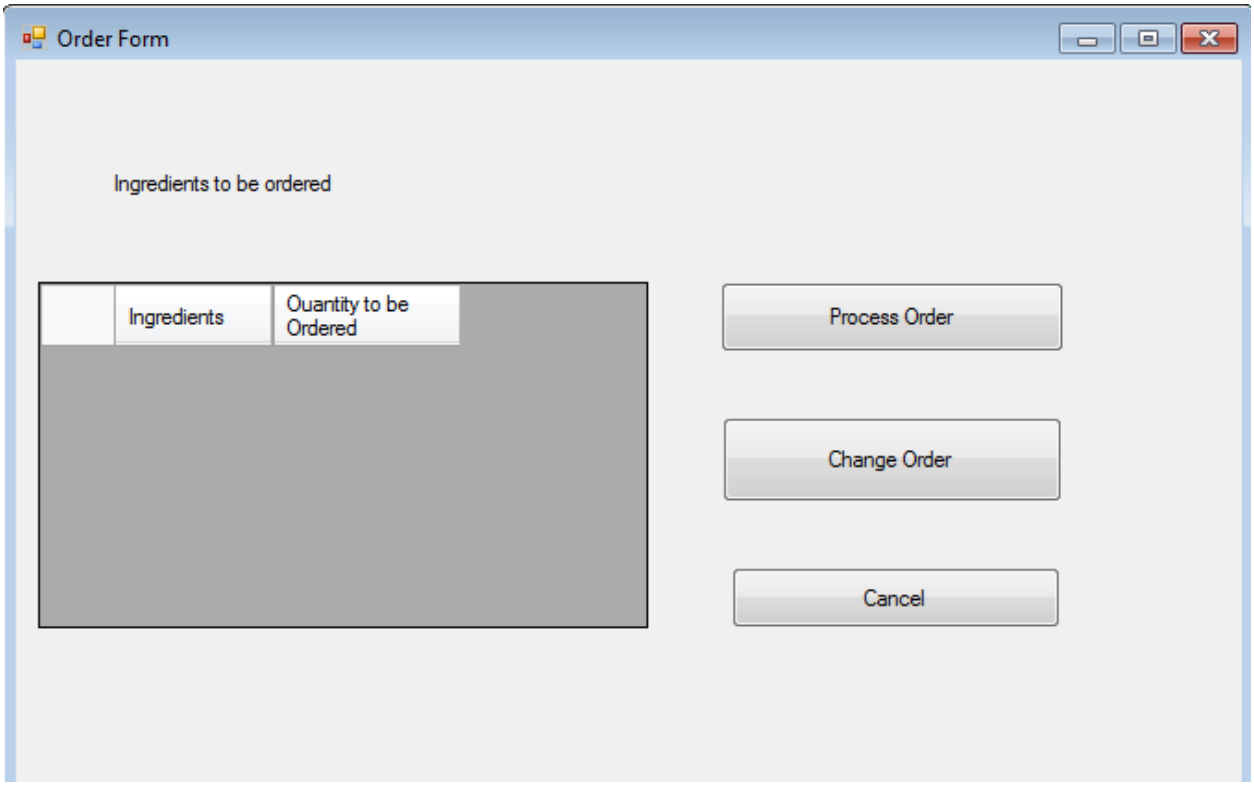
- Add Ingredient Screen



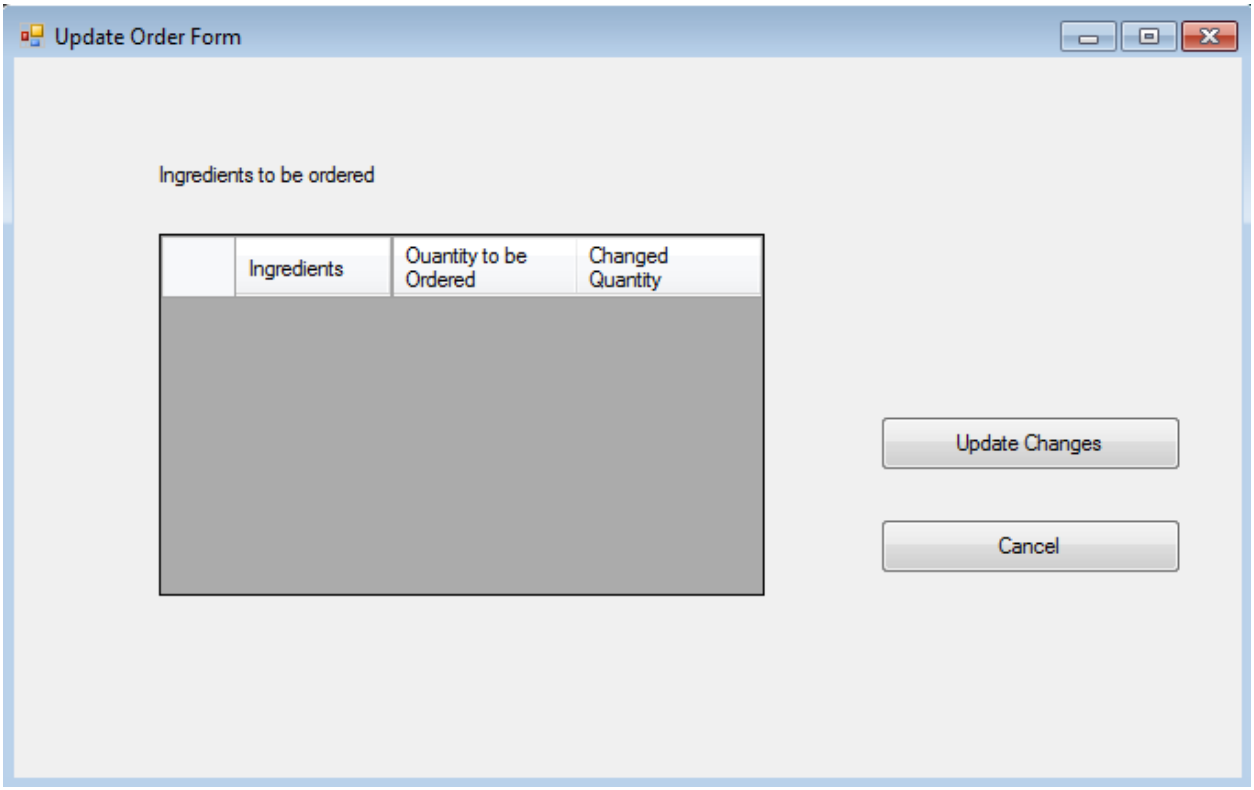
- Sold food form screen



- Order Form

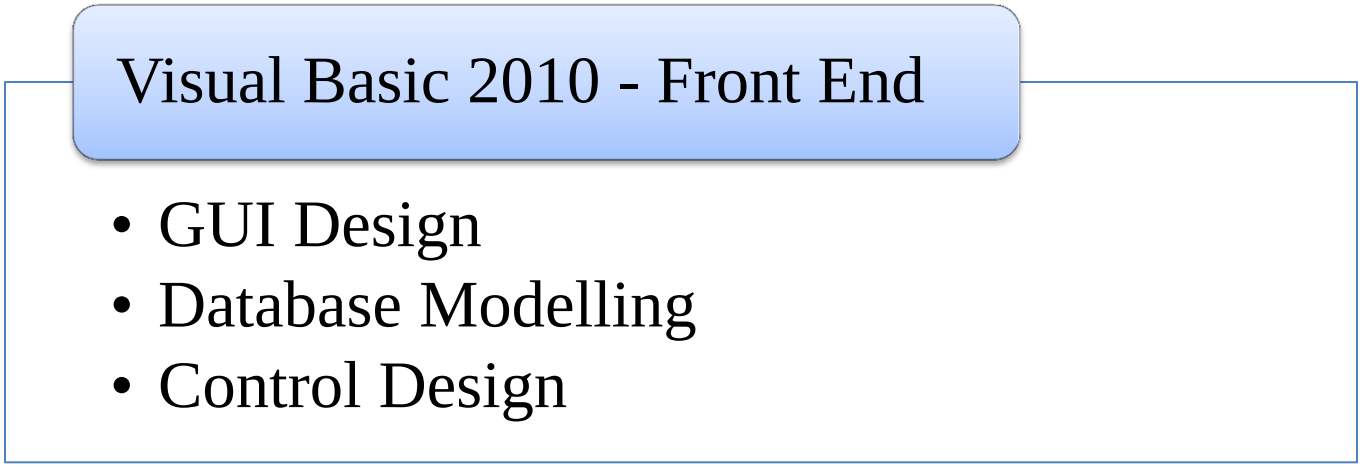


- Update Order Form screen

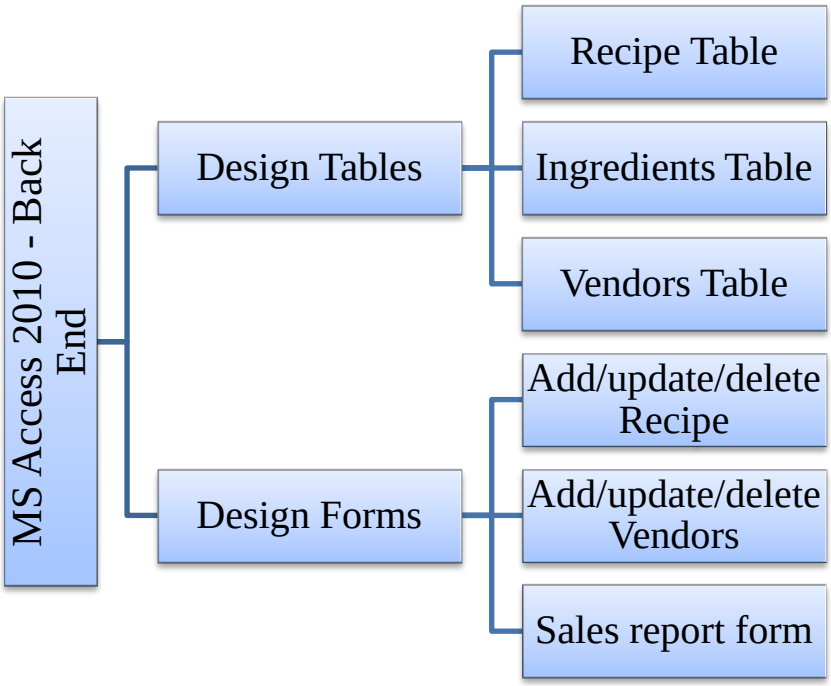


3.4.6 Architecture Breakdown

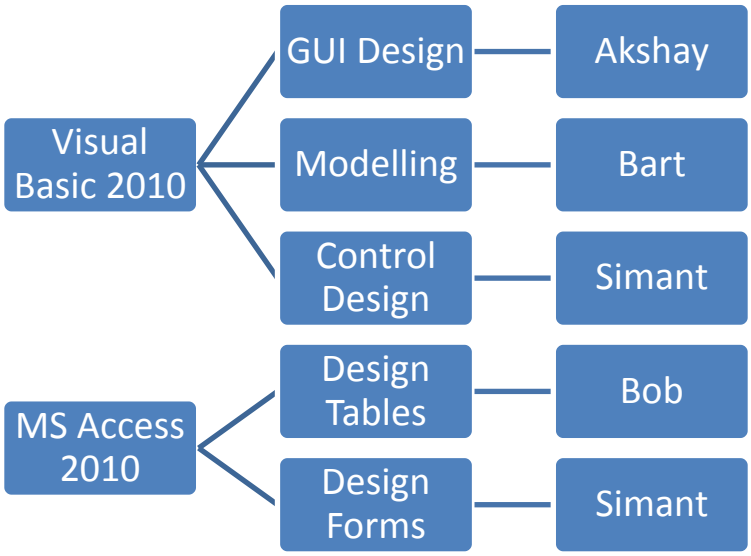
Front End:-



Back End:-



Assignment of responsibilities



4. GLOSSARY

Manager	A manager is the person who is here considered to be the person in charge. He undertakes responsibility of maintaining, operating the store and keeps information of each and every specifics. He is also responsible for taking decisions regarding issuing orders to vendors , add new vendors, create new recipes ,etc.
Recipe	A recipe is a dish that is made using the ingredients from the store. The recipes and ingredients are inter-related. A recipe is usually made up of two or more ingredients.
Ingredients	Ingredients here considered to be atomic substances which are used as a component of a recipe. These basic ingredients cluster to form a recipe. The availability of these help in creating new recipes and regularly checking the inventory levels and prediction analysis.
Vendor	The necessary consumable items (ingredients) which are used up in preparing the recipes are provided by the vendor on a regular basis. The manager is responsible for checking the current level of inventory with the threshold levels and if any particular ingredient is found to be in the danger zone then an order is processed by the manager to the vendor.
Order	Order is a generalized function used for replenishment of used up ingredients. The order usually consists of an order form which is supplied by the manager to the vendors.
Add Recipe	This action leads to the addition of a new recipe into the existing list of recipes. When a new recipe is made it also links up to the ingredients being used. So whenever a particular recipe is ordered it ends up using its required ingredients.
Remove Recipe	The manager initiates the remove the recipe action. Once a recipe is removed it is also cleared from the RECIPE table.
Add Vendor	Add vendor is the case when a new ingredient is being added to the database and the present vendor isn't supplying that particular item.
Remove Vendor	If a vendor is removed from the list of vendors then the recipe table reflects only the recipes that are not linked with that particular vendor.
Check Threshold	Check threshold level refers to checking the inventory levels of all the ingredients. This provides information pertaining to the next order (vendor) and also the recipe selection.
Process Order	A process order issues a purchase/required list of ingredients and accordingly generates a purchase order for the vendor.
Update Resource Database	Updating the resource gives a crystal clear idea to the manager regarding the usage of the inventory from the quantity sold in a particular time period. This leads to the checking the threshold values.
Add Occasion	Occasion refers to something special and this leads to extra needs from the vendor. So accordingly when an occasion is added the specials recipes are kept in mind and a purchase order to satisfying the special needs is generated.