FINAL PROJECT REPORT FOR INVENTORY CONTROL SYSTEM FOR THE CALCULATION AND ORDERING OF AVAILABLE AND PROCESSED RESOURCES

GROUP 9

- ✓ SIMANT PUROHIT
- ✓ AKSHAY THIRKATEH
- ✓ BARTLOMIEJ MICZEK
- ✓ ROBERT FAIGAO

December 7, 2012

ACKNOWLEDGEMENTS

We would like to thank Ms. Kimberly Harmon for her professional input, feedback, and support, as well explaining the need required to make our product a successful one. Her experience in the restaurant industry proved fruitful and extensive when it came to the project requirements and development. We all eat at restaurants, but no one realized that the amount of work that chefs do goes beyond simply cooking the meal.

We would also like to thank Professor John Bell and his teaching assistant, Munavvar Khan, for their continued guidance and feedback throughout the course of the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

At the end of their day, chefs and managers in the restaurant industry spend a couple of hours counting inventory and placing orders for the following week. The Restaurant Inventory Control System is designed to not only assist in this problem, but also automate many of the tedious tasks associated with it. The system keeps track of current inventory levels for recipes at the ingredient level, predicts how much inventory is needed for the upcoming week, and generates order forms to that can be automatically sent to vendors.

After meeting with a chef for Guckenheimer, an on-site corporate restaurant management company, we were very easily able to pinpoint issues in the maintenance of resource requirement lists. To keep track of their inventory levels, staff had to calculate a list of groceries utilized during a course of time, calculate and analyze the requirements for the future, and place their next order to multiple vendors if needed. This process takes up a lot of time and human effort, and is also prone to human error. The same chef used to be the head chef at Vintage 338, a privately owned Chicago wine bar, where they had the same issues.

It became our goal to develop a program that can be used by both large corporations as well as small businesses. This meant the system had to provide an efficient and simple user interface that at the same time is capable of more precise changes and inputs. The system had to also be accurate and reliable in terms of the database design. Since all of the data and data objects are stored in a database, it was imperative that these requirements were met.

The basic functionality underlying the system is as follows: chefs can add recipes to the database, which are then broken down to their ingredient level. These ingredients are then tracked by the system and updated with each sale of certain items. Should they reach a predetermined threshold level, the manager is notified and given the option to place an order with the respective vendor. Through the use of a prediction algorithm, the system uses data such as previous sales, future dining events, and special requests to determine order quantities. The manager has control over all factors associated with the system, should they require a change.

Certain functional requirements that were brought up during our case study by the chefs included allowing the user to be able to create, delete, and update recipes, ingredients, and vendors as these changed frequently. They also stated that the system must include mechanisms for the manager to approve any outgoing orders in case manual changes needed to be made, as well as allow changes to be made to inventory levels in case of an error.

The system offers very precise control over the database, allowing the manager to add, remove, and update the recipes, ingredients, vendors, and future events. It also include important functionalities of predicting future inventory needs by accounting for thing such as past sales, upcoming events, and unique ingredients that may be needed for a special occasion or recipe. Once the manager confirms and if necessary, updates order requests, forms are generated to specific vendors that can be easily mailed out.

The Restaurant Inventory Control System was originally designed to be a Windows application developed in Visual Basic (for the user interface and logic) to store data in a SQL Server, but a decision to switch to Java and Java Database Connectivity (JDBC) was made during the development phase due to simpler and more versatile deployment.

Testing was completed to ensure that incorrect user inputs weren't added to the database. Any incorrect information in the database would cause a trickle effect of issues throughout the entire system, which is heavily dependent on the data. We also tested each subsystem individually to ensure that the requirements set for the project were achieved.

The system was successful in accurately maintaining the inventory levels, predicting the requirements of the next order, relating recipes to their respective ingredients, and provided a simple and effective user interface to update inventory levels and place orders to vendors.

As always, there do exist improvements for the system, given that a system of this scale would still be considered in early stages of development. The prediction algorithm can be enhanced further, but that would only be possible with large sets of data analysis that would be unique to each company using the product. We have to keep in mind that although we have encompassed the restaurant industry as a whole in the scope of this system, that industry itself can be broken down into multiple layers. Each of these layers would have its own specific requirements of dealing with inventory control. Also considering the large technological movement, access to the program through a web application would be ideal for remote access to the program and database. This would require a dedicated server to host the database and dedicated web development and therefore has been considered as an optional enhancement.

The program completes a task that some may deem trivial, but many chefs would greatly appreciate to have in their own work environments. Not only does it reduce the workload on chefs that need to keep track of every ingredient used, it also automates a task as simple as sending a food item order. Although the final product is not yet complete for wide distribution, we are confident that we have successfully fulfilled an important need of data management in the restaurant industry. What was once the manual labor of counting and ordering, as well as the mental labor of memorizing all ingredients used in a recipe was digitalized and streamlined into a process that can be used efficiently and reliably.

1 PROJECT OVERVIEW

1.1 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.

1.2 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.

1.3 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.

1.4 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. 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.

1.5 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.

1.6 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

2 System Architecture Overview (Development Environment)

2.1 FRONT END

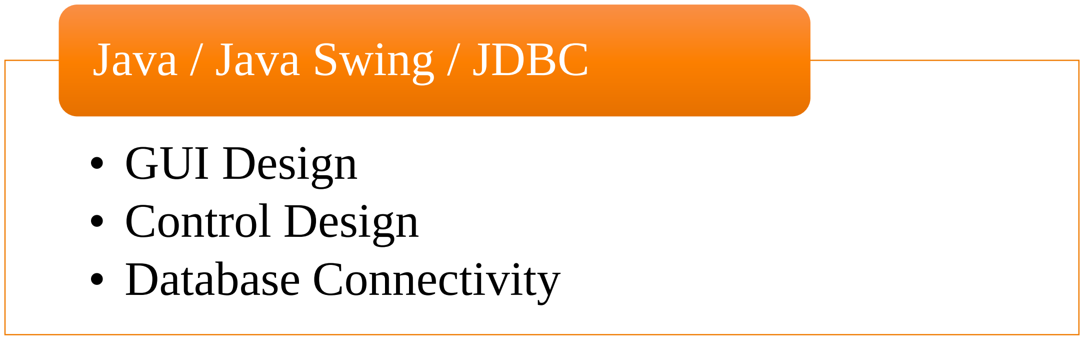


Figure 1: Front End

2.2 BACK END

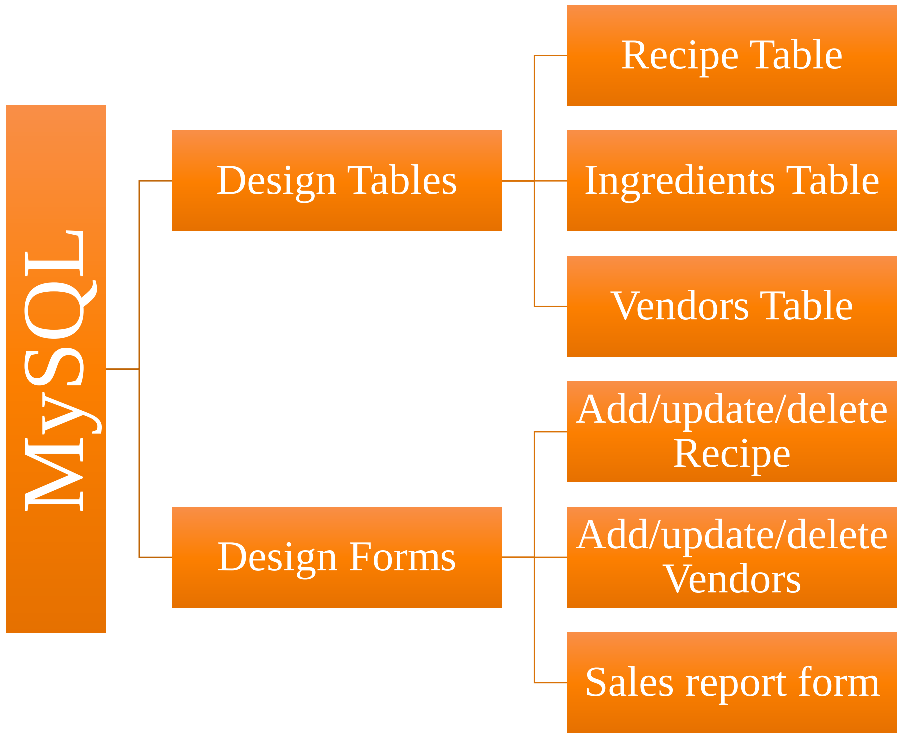
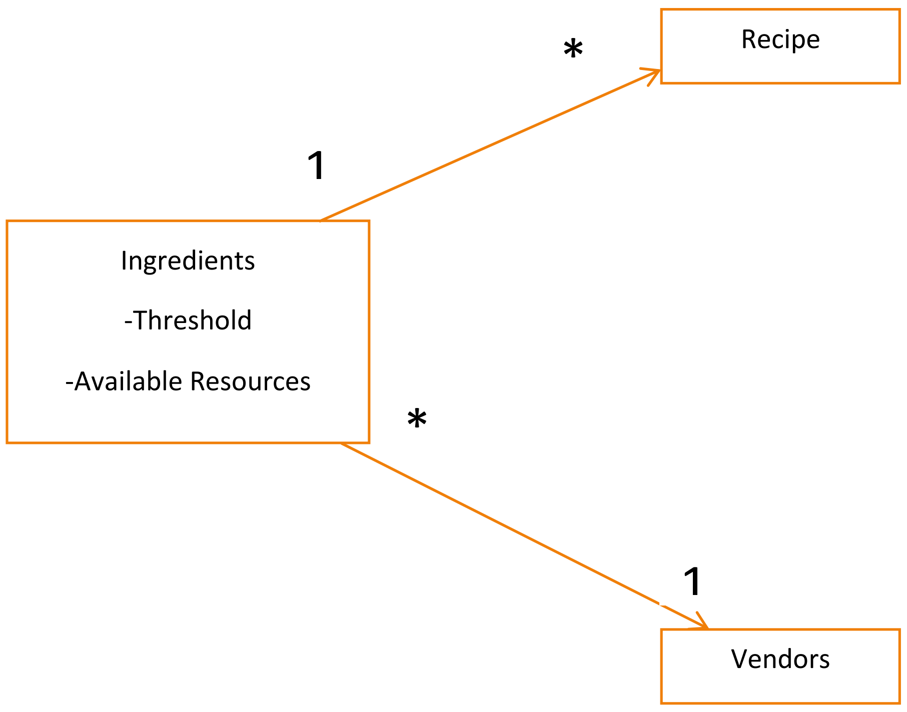


Figure 2: Back End

2.3 Basic Database Relationship Diagram



2.4 Assignment of responsibilities

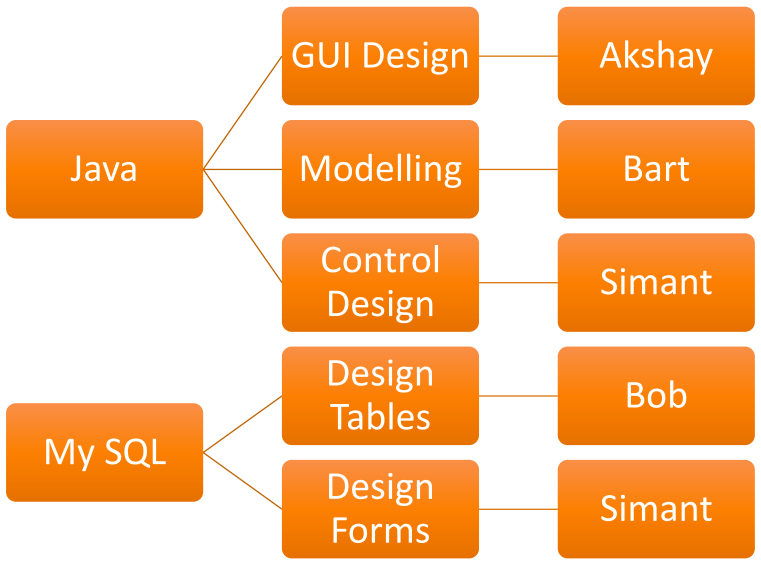


Figure 3: Responsibilities

3 Requirements Analysis

3.1 Functional Requirements

- ✓ The user must have, at disposal, functions for managing the inventory efficiently.
- ✓ The functions for inventory management should allow the user to know which ingredients in the inventory are below their threshold levels and need attention.
- ✓ The system must include functions that will allow the user to add a recipe, ingredient, vendor to the database.
- ✓ The user should also be able to delete any recipe from the database when not needed.
- ✓ The system must allow the user to create orders for the ingredients that are below threshold.
- ✓ The system must include a mechanism for the user so that the user can just update the sales of the day in the system and the system deducts the corresponding amount of ingredient quantity from the inventory. Thus keeping a track of ingredients.
- ✓ The system must also include functions for the user to add special days in the system when the inventory usage will be more than usual or less than usual and thus provide a way to alert the user of the possibility of over usage or under usage or certain ingredients.
- ✓ The system also must provide a prediction function to the user where the system will give the user the predicted usage of inventory of certain pre-set days.
- ✓ The system must have a password protected access system such that only people with authenticated credential are allowed to access the function of the system.

3.2 Non-functional requirements

- ✓ Usability

  i. 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.

  ii. The system must be quickly accessible by both managers and chefs.

  iii. The system must be intuitive and simple in the way it displays all relevant data and relationships.

  iv. The menus of the system must be easily navigable by the users with buttons that are easy to understand.

*****Note: For detailed Requirements, refer to the Requirement Analysis Document [9]***.

- ✓ Reliability

  i. 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.

  ii. 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.

  iii. The system must provide a password enabled login to the user to avoid any foreign entity changing the data in the system.

  iv. 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.

  v. The system should not update the data in any database for any failed processes.

- ✓ Performance

  i. The system must not lag, because the workers using it don't have down-time to wait for it to complete an action.

  ii. The system must complete updating the databases, adding of recipe, ingredient, vendor and occasions successfully every time the user requests such a process.

  iii. All the functions of the system must be available to the user every time the system is turned on.

  iv. 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.

- ✓ Supportability

  i. The software is designed such that it works even on systems having the minimum configuration.

  ii. The system is adaptable even if additional plugins or modules are added at a later point.

  iii. The data can be exported to the manager so as to make the system more portable.

- ✓ Packaging

    i. 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

    ii. The software must incorporate a license key authentication process.

    iii. 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 USE CASE MODEL

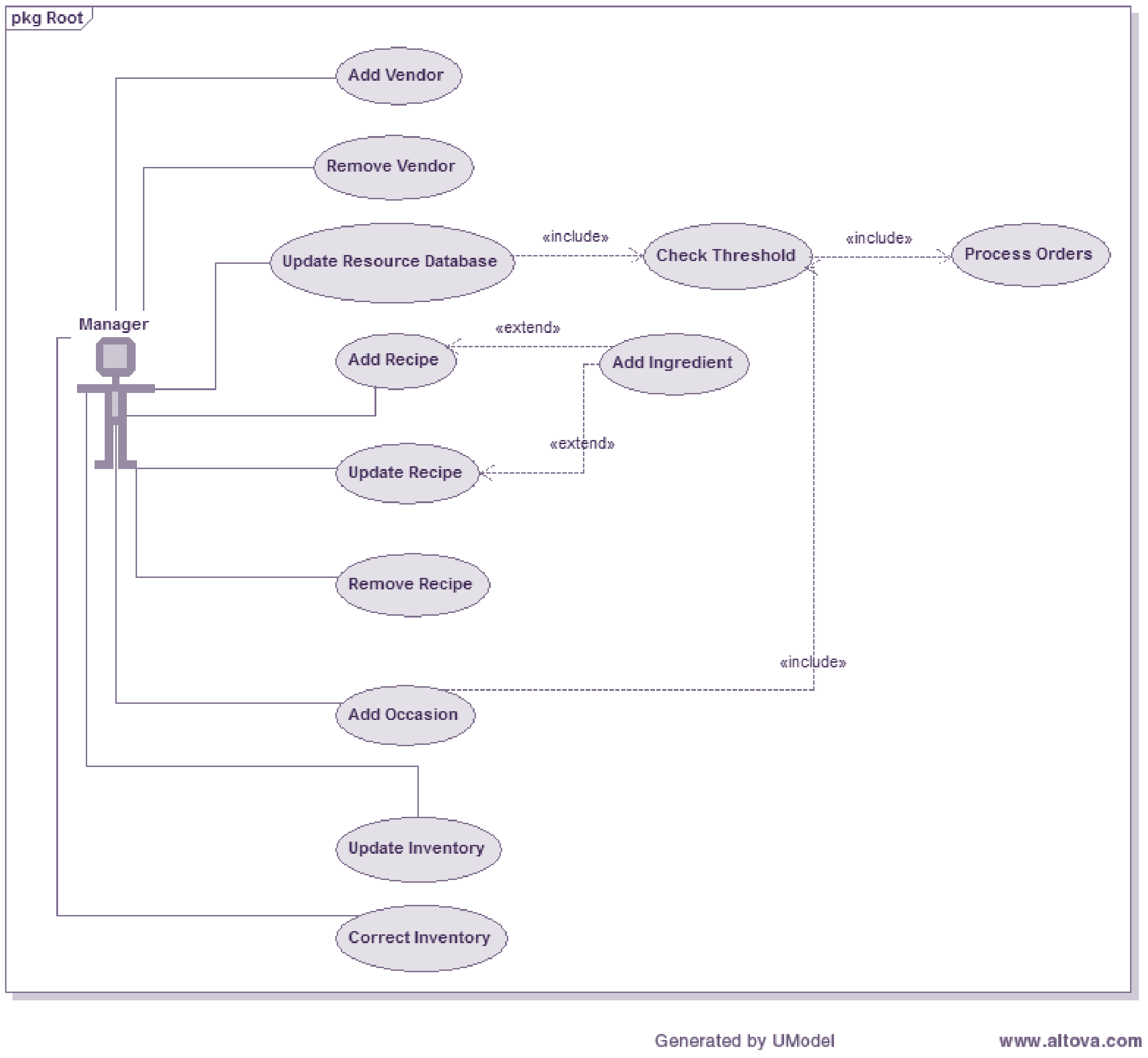


Figure 4: Use Case Model

3.4 USE CASES

3.4.1 Update Resource Database

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.

Table 1: Update Resource Database

3.4.2 Check Threshold Use Case

Usecase 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

Table 2: Check Threshold Use Case

3.4.3 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

Table 3: Process Order Use Case

3.4.4 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.

Table 4: Add Recipe Use Case

3.4.5 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.

Table 5:Update Recipe Use Case

3.4.6 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.

Table 6:Remove Recipe Use Case

3.4.7 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.

Table 7: Add Occasion Use Case

3.4.8 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.

Table 8: Update Inventory Use Case

3.4.9 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.

Table 9: Correct Inventory Use Case

3.4.10 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

Table 10: Add Vendor Use Case

3.4.11 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

Table 11: Remove Vendor Use Case

3.4.12 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.

Table 12: Add Ingredients Use Case

3.5 Multiplicity and Association Diagrams

3.5.1 Multiplicity Diagram

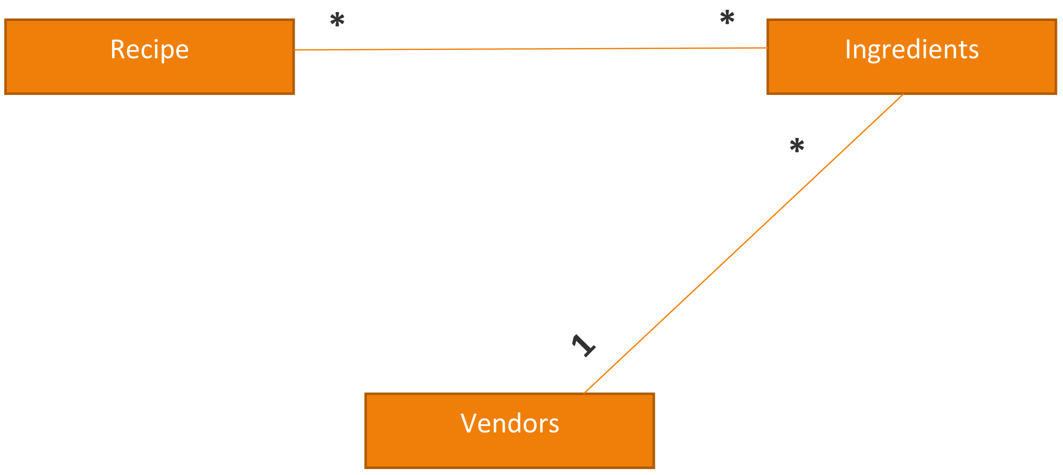


Figure 5: Multiplicity Diagram

3.5.2 Association Diagram

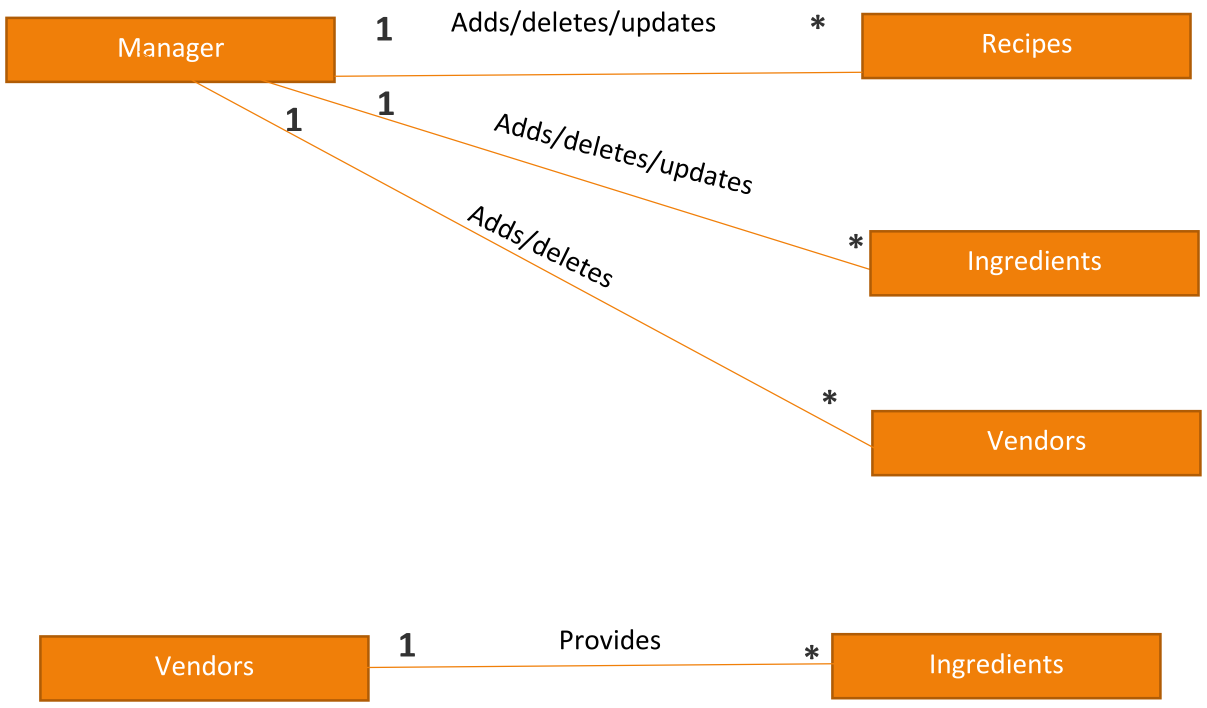


Figure 6: Association Diagram

3.6 Dynamic Model

3.6.1 Update Resource Database Sequence Diagram

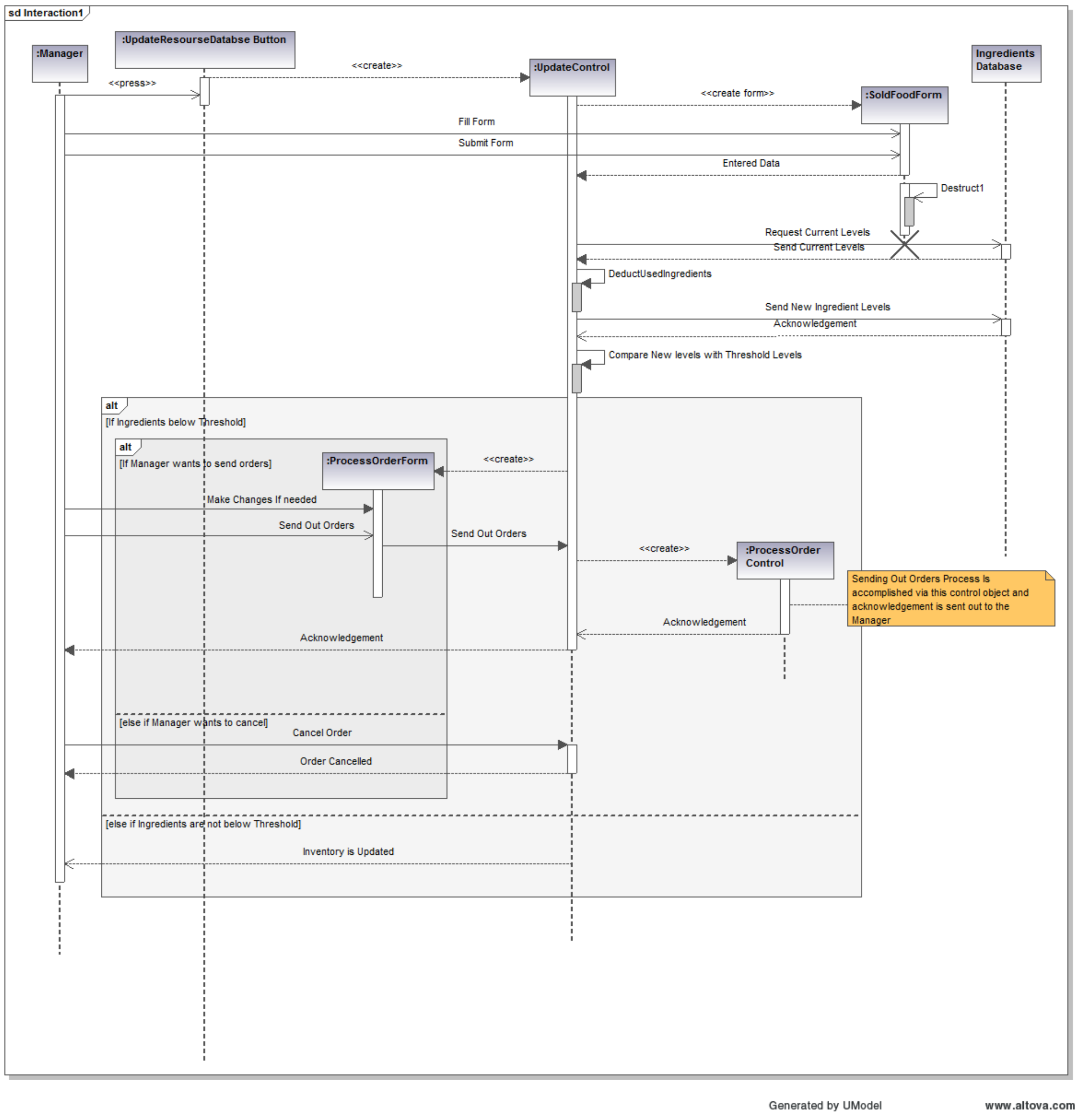


Figure 7: Update Resource Database Sequence Diagram

3.6.2 Add Recipe Sequence Diagram

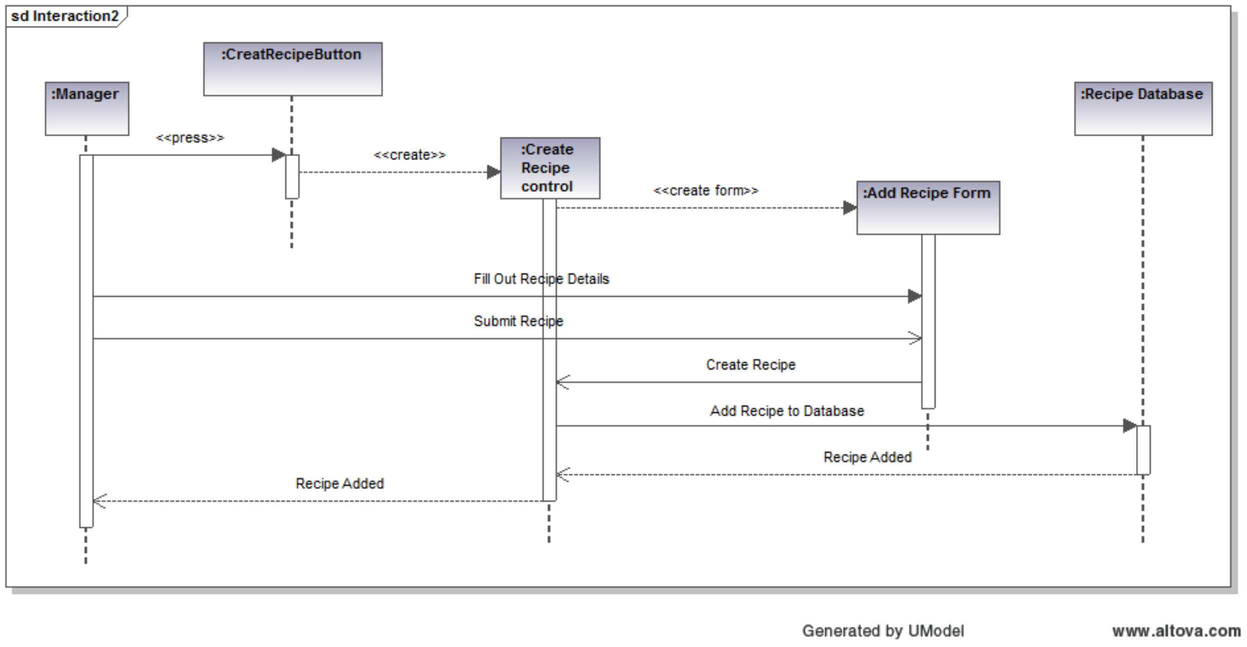


Figure 8: Add Recipe Sequence Diagram

3.6.3 Remove Recipe Sequence Diagram

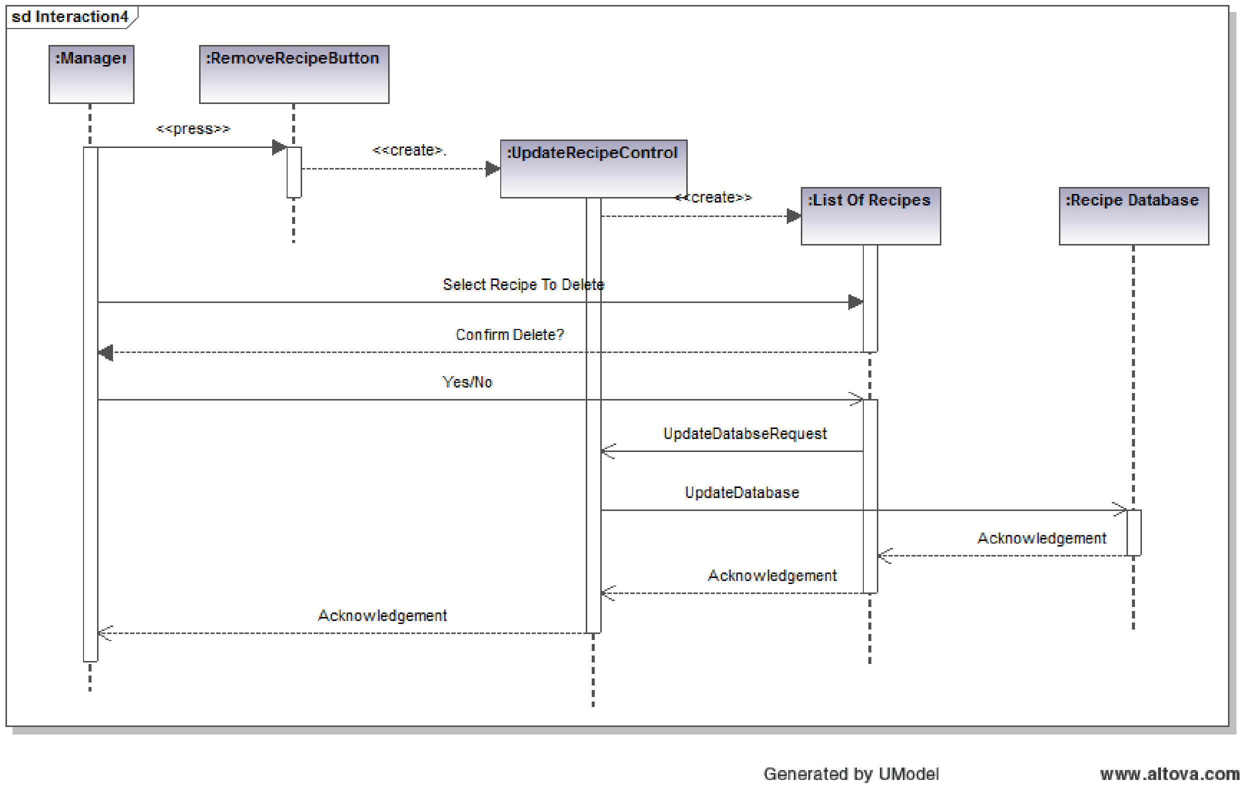


Figure 9: Remove Recipe Sequence Diagram

3.6.4 Update Recipe Sequence Diagram

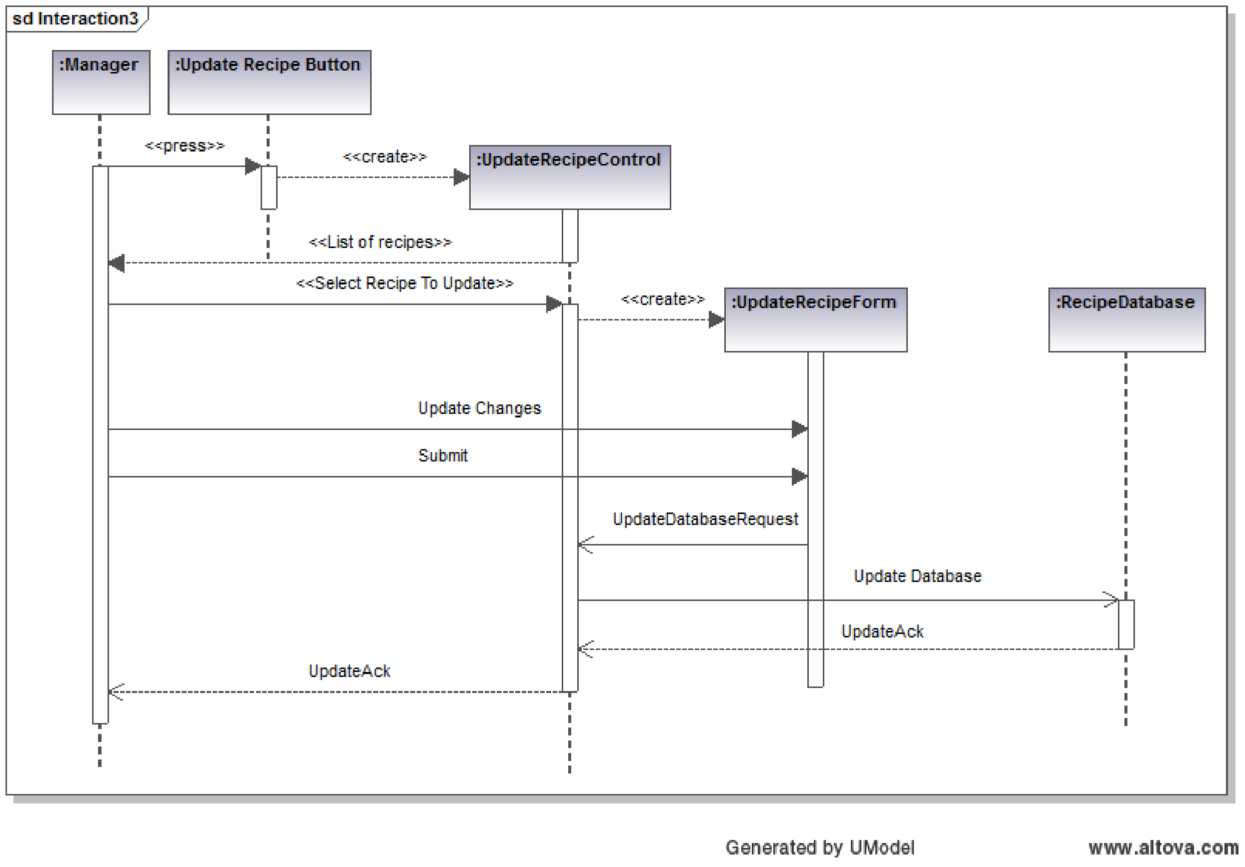


Figure 10: Update Recipe Sequence Diagram

3.6.5 Add Vendor Sequence Diagram

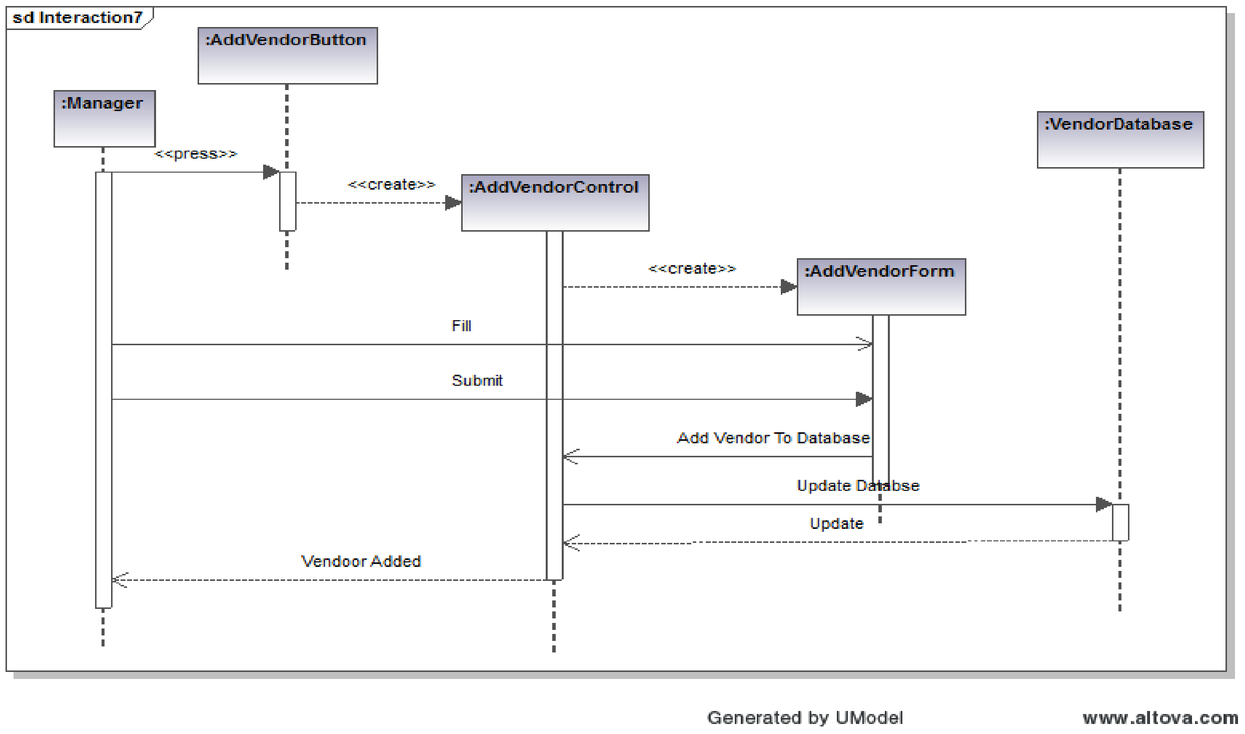


Figure 11: Add Vendor Sequence Diagram

3.6.6 Remove Vendor Sequence Diagram

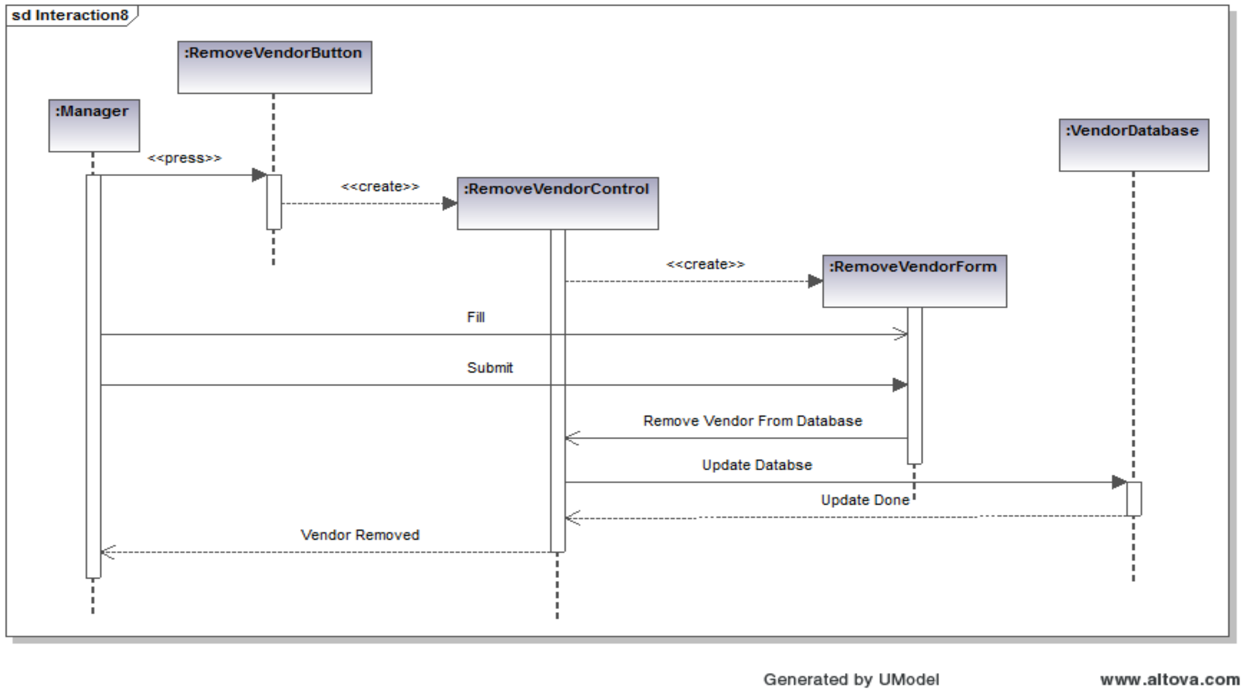


Figure 12: Remove Vendor Sequence Diagram

3.6.7 Update Inventory Sequence Diagram

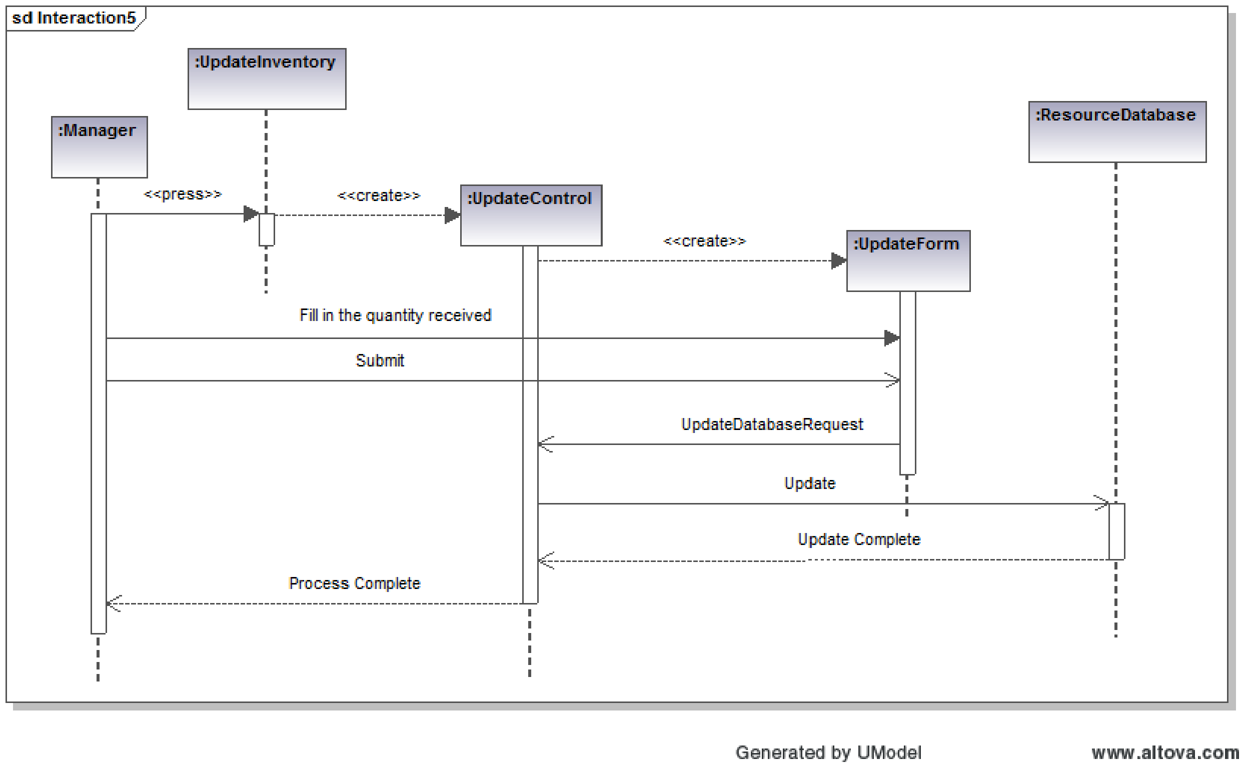


Figure 13: Update Inventory Sequence Diagram

3.6.8 Correct Inventory Sequence Diagram

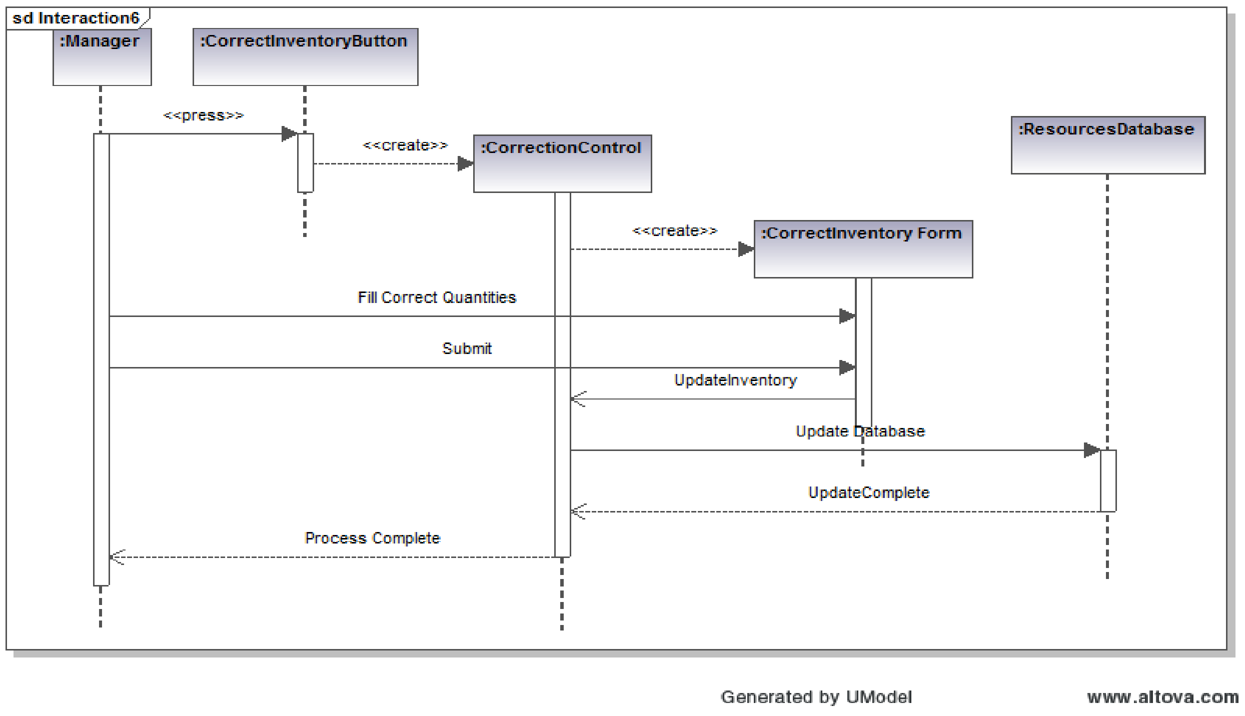


Figure 14: Correct Inventory Sequence Diagram

3.6.9 Add Occasion Sequence Diagram

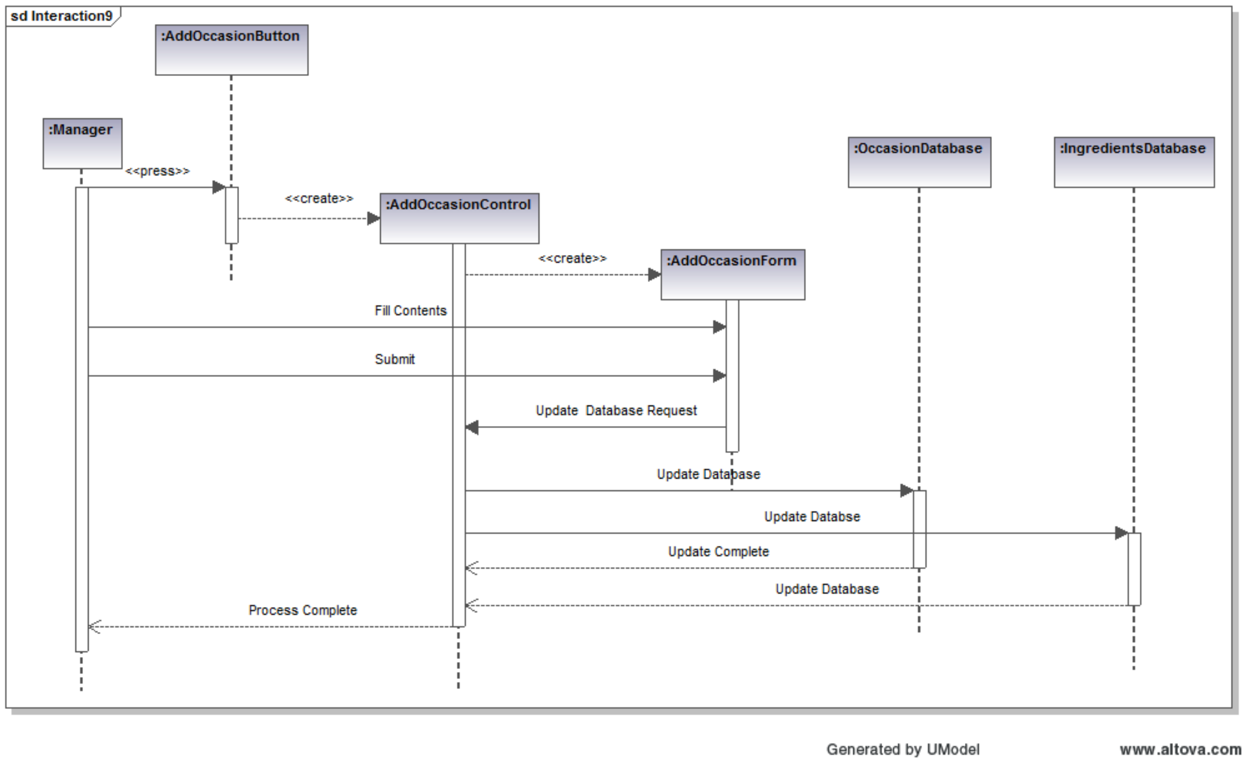


Figure 15: Add Occasion Sequence Diagram

4 DETAILED SYSTEM DESIGN

4.1 DESIGN GOALS

- **Low Response Time**: The main functionality of the system involves updating and reading the data from the database for different entities such as ingredients, recipes vendor etc. Thus the time required to retrieve/ update/ add data to the database should be minimum and preferably should be in the range of 2-5 seconds or lesser.

- **High Robustness:** The system should constantly check the user input at all instances that could generate errors in the program. For instance:

    - The system should be able to check input values for the amount of ingredients required for the recipe and should make sure the user enters a numeric value in the input box and the system shows an error and asks the user to re-input if in a perfectly validated field an improper data type is inputted.
    - The System should have validated input data fields and must put a constraint on the inputted names of recipe, ingredient, vendor, and occasion etc. to ensure no duplicate entries are added in the database. This ensures the robustness of the maintained database.
    - The system should verify all the inputs by the user by using a confirmation dialog box before processing and making changes to the data.

- **High Reliability:** The reliability of the system depends upon its ability to replicate the specified behavior. The safekeeping of the data is essential so as a result a backup of the levels is generated and stored in the warehouse. There are numerous factors on which reliability can be defined as for example, the specifications mention that the updating of database or the notification of a successful update must be carried out within 2-5 seconds of initiation and the system must adhere to these specifications to be called a reliable system. Similarly, the system should be able to achieve performance in lieu with the specifications mentioned.

- **Low fault tolerance**: The system works on sensitive data and therefore any fault in the functioning of the system will hinder accurate updating or reading of data. This could lead to invalid entries in the database. Thus, the system should have low fault tolerance. This is in tandem with the design goal of high robustness as the validation checks to ensure correct inputs from the user implies that the fault tolerance of the system is low.

*****Note: For detailed Design Report, refer the Design Report submitted earlier. [10]***

- **Security:** The system must provide a login functionality to the *Manager* as the manager is the authenticated controller of the system and any other user is not permitted to use the system functionality and make changes in the database. Thus proper user authentication should be necessary before system launch.

- **High Extensibility:** The design of the system should be such that any future improvement can be added with no or minimum improvements. It is in one's best interest to always give space for future enhancements. For instance, right now there is no class that will help the user to manipulate prediction of values and the current system only predicts the ingredient usage for a certain date, but a feature can easily be added to incorporate prediction of recipe usage, order prediction etc.

- **Low Adaptability:** The system is designed to work on the domain of inventory control and management in the restaurant and catering industry. The functioning of this project is limited only to these particular businesses which have similar functioning and thus it would then be subject to structural re-modification in order to to apply it in some other application domain.

- **High Readability**: The system code should be properly commented so as to explain the functionality of the code fragments. The code comment should explain the function or task the code fragment performs and the result and the return value of the corresponding function or task should also be mentioned.
- **High Traceability**: The coding scheme of the system should be such that it could be traced back to its requirements specifications. This will enable high traceability of the code of the system.

4.2 Subsystem Decomposition

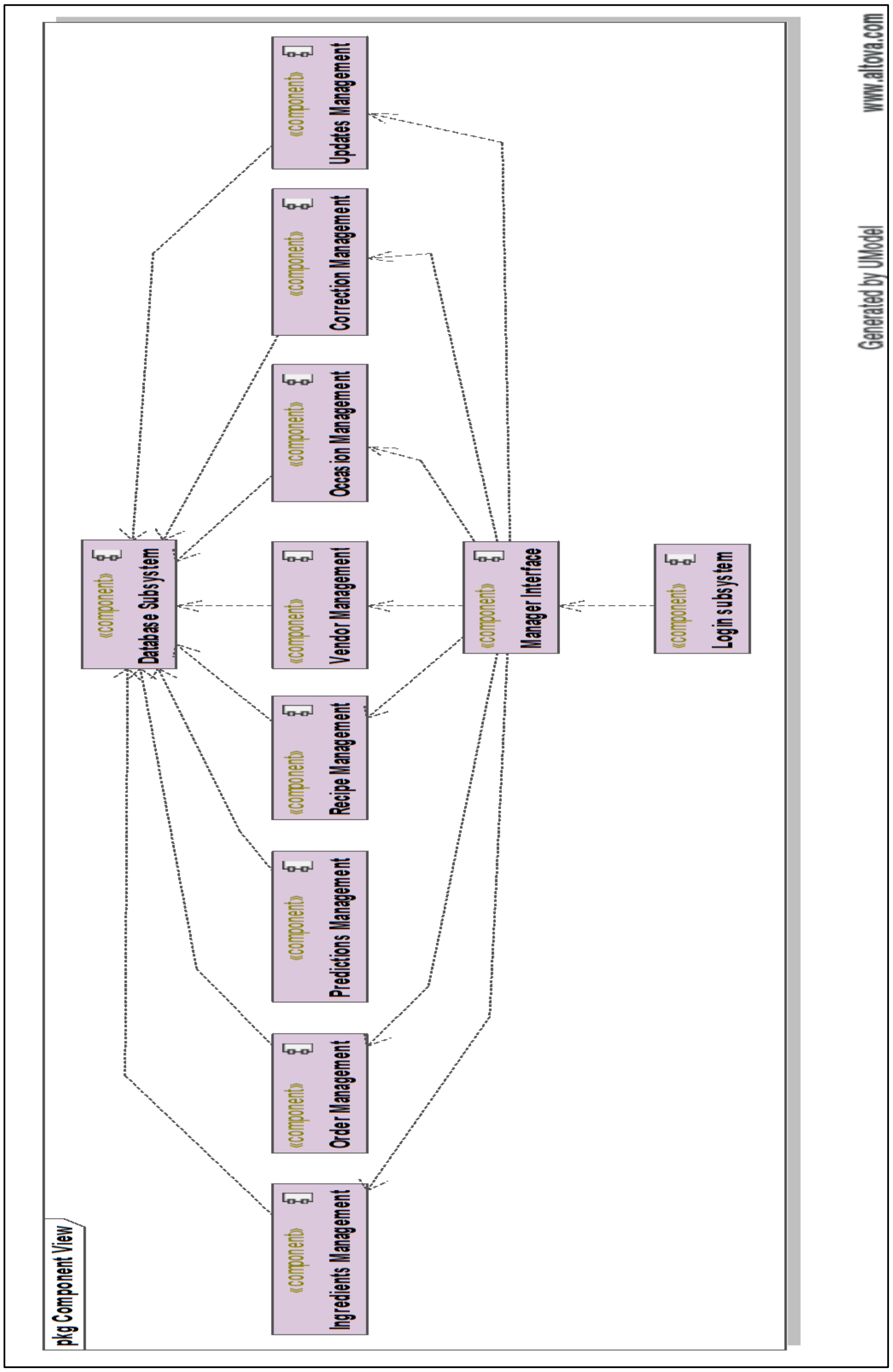

pkg Component View
«component» Ingredients Management
«component» Order Management
«component» Predictions Management
«component» Recipe Management
«component» Vendor Management
«component» Occasion Management
«component» Correction Management
«component» Updates Management
«component» Database Subsystem
«component» Manager Interface
«component» Login subsystem
Generated by UModel
www.altova.com

Figure 16: Subsystem Decomposition

Subsystem Description

ManagerInterface	This subsystem defines an interface between the user and the system. The user through this interface can access and execute different functions on the various subsystems.
IngredientsManagement	This subsystem provides services to manage the ingredients inventory of the system. This subsystem provides services such as providing list of available ingredients, providing details of individual ingredients in terms of the current inventory levels, threshold levels etc. This subsystem requires the services of Database subsystem to retrieve required details.
RecipeManagement	This subsystem provides services to manage the Recipes in the inventory. This subsystem provides services such as adding/updating/removing a recipe to/from the inventory. This subsystem requires the services of Database subsystem to retrieve the list of ingredients that make up the recipe.
VendorManagement	This subsystem provides services to manage the vendor that deliver Ingredients. This provides services such as providing details of the vendor, providing list of ingredients that the vendor supplies. This subsystem communicates with the Database subsystem to retrieve the list of vendors and the ingredient list to match them with the vendor names.
PredictionManagement	This subsystem provides prediction of usage per ingredient to the user. This subsystem communicates with the Database subsystem to access past data of used resources from the database and then apply prediction algorithm on the retrieved data to give estimates of usage to the user.
OrderManagement	This subsystem provides services to generate orders for vendors for the ingredients that are below threshold levels. This subsystem also provides like editing a generated order and cancelling an generated order.
OccasionManagement	This subsystem provides services to the user to add an occasion date to the system so that the system prepares itself for an upcoming event on which day the sales will be more than usual day sales. This subsystem requires the services of Database subsystem to update/remove occasion days from the database.

CorrectionManagement	This subsystem provides services to the user to correct the levels of inventory and avoid inventory slips. The user here uses this service to match the actual inventory levels with the inventory levels in the system.
UpdatesManagement	This subsystem provides services to the user to perform updates on the database.
DatabaseSubsystem	This subsystem connects to the database and provides requested database to the other subsystems that request data from it.

Table 13: Subsystem Description

4.3 Hardware Software Mapping

The system runs on a standalone system without the need of any external server connection or internet connection. Thus the hardware requirement of the product is a personal computer which meets the requirements mentioned in the specifications. The product is programmed in Java programming language and uses MySQL for database service. Thus the client computer requires the installation of JDK 1.6 and MySQL server on its machine. The mapping between the hardware and the software can be interpreted by the following diagram.

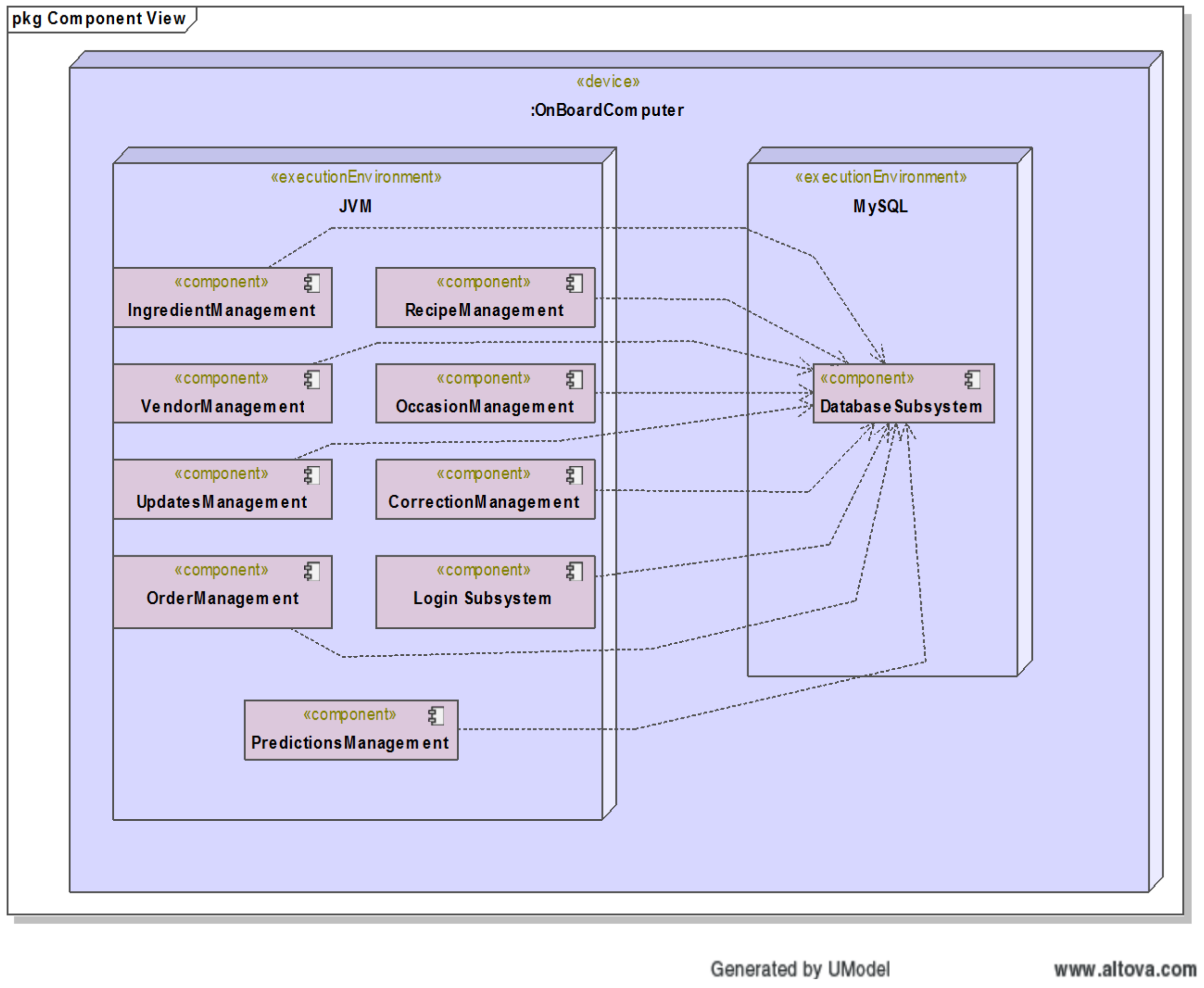


Figure 17: Deployment Diagram

4.4 PERSISTENT DATA MANAGEMENT

4.4.1 Persistent Objects

The main data entity that is persistent in the system is objects of class Ingredients. The objects of class Ingredients are used by various classes to function. For example, the class Recipe uses the objects of Ingredients class to define the contents of recipe with the name of ingredients that are accessed by the objects of the class of Ingredients.

The object of the class Recipe have also to be classified as persistent objects even though the class derives some of its properties from the Ingredients class. The reason for this is the classes AddRecipe, RemoveRecipe, UpdateRecipe and Occasion are derived from the Recipe class and require the object of the recipe class for their functionality.

Similarly, the objects of class Vendor have to be persistent as it also forms a building block of the whole database system. The Vendor class objects give the list of vendor along with the ingredients they provide. Thus the Vendor class uses the objects of the Ingredients class for its functionality.

In short, the objects of classes Ingredients, Recipe and Vendor are connected and depend highly upon each other for their functionality. Additionally, other classes defined in the class diagram of the system depend upon the data provided by the above mentioned classes for their functionality.

4.4.2 Storage Strategy

The database is created and maintained in the open source environment MySQL. The subsystems connect to the database via JDBC API. Using an open source database management system enables us to reduce cost of development of the system plus it allows us to maintain the database on the same machine on which the other subsystems are functioning. The only thing required is a JDBC connector driver to setup and maintain connection to the database. The type of database used is Relational database, as using a flat file database would prove tedious if used on such highly connected data entities that we use in our system.

4.5 Access Control and Security

4.5.1 Access Matrix

Class	**Functions accessible to *Manager***

Recipe	addRecipe()
	removeRecipe()
	updateRecipe()

Ingredients	getIngredientsList()
	addIngredient()
	updateIngredient()

Vendor	getVendorDetails()
	getIngredientListFromVendor()

Prediction	getPredictedUsage()

Updates	updateAfterSales()
	updateAfterReceiving()

Occasion	addOccasion()

Order	sendOrder()
	editOrder()
	cancelOrder()

Table 14: Access Matrix

4.6 Global Software Control

The type of global control flow used here is 'Event driven'. The system works on the services requested by the *Manager.* As a result the system waits for some activity on the part of the *Manager* to initiate any process. Until then the system waits on the main user interface that provides the manager access to various functions on the system.

To ensure a robust design of the system, we define some strategic goals for the system as mentioned below:

- ✓ *Boundary objects do not define any of the fields in the System,* instead they are only associated with creation of control objects upon request of access to specific functions by the *Manager.*
- ✓ *Control Objects must not be shared among functions,* instead if one function on the interface is activated the other functions must not be accessible by the *Manager* until the current function complete its operation.
- ✓ *Entity objects must not allow direct access to its fields to any other class or object,* instead it should use getter and setter method to get and set the values of its attributes for better encapsulation.
- ✓ *Database connection must not be open all the time,* instead the connection to the database should be made only when any functions requests data or wishes to update data.
- ✓ *Entity data validation should be conducted at point where data is written into the database,* this will ensure that no invalid data is entered into the database avoiding any serious inventory slips in the future.
- ✓ *The system will require an authenticated login and password for accessing the main interface and hence all the functions,* this will prevent any unauthorized login into the system and hence make the system secure. The master login and password will also ensure that the user is enabled to connect to the database subsystem with any explicit login into the database management system.

4.7 Boundary Conditions

StartSystem: The *Manager* initiates the system using this function. At the startup, the *Manager* is asked for authentication and if the authentication is successful, the main interface becomes visible to the manager. There are no database connections established at in this phase hence meeting our global control flow specification mentioned earlier. The *Manager* can now access and perform various services accessible from the main interface. Any function accessed opens a database connection to the desired database and closes the connection on termination of the function.

ExitSystem: The *Manager* can stop the system using the exit function on the main interface. This action terminates the system. It is assumed that there are no active database connections at this point of time as the *Manager* is at the main interface windows of the system and the design goals define no database connectivity at this instance. Database connections are opened and closed at the initiation and termination of the certain

Defining Exceptions

The scenarios of failure of the system can be stated as follows:

- The database connection cannot be established.
- Incorrect data is entered into the database in spite of the validations being carried out.
- The system crashes in the middle of an update process being carried out.

To deal with the above mentioned exceptions, we define two use-cases that will be used to overcome or at least reduce the effects of the exception.

CheckDataIntegrity	This use case can be invoked in the event of the one of the last two exceptions occurs. This use-case will run through the whole database and check for non-related data entries, incorrect data entries and incomplete data-entries. It will then delete these irrelevant data entries from the database and provide the *Manager* with an error free database to work with.
ResetDataConnectivity	This use case can be invoked in the event the first exception occurs. This use case with restart/re-establish all the database connectivity for the application, invoke the CheckDataIntegrity use case mentioned above and then provide the user with fresh error free database connectivity.

Table 15: Exception cases

4.8 Subsystem services

The subsystem services diagram can be shown below

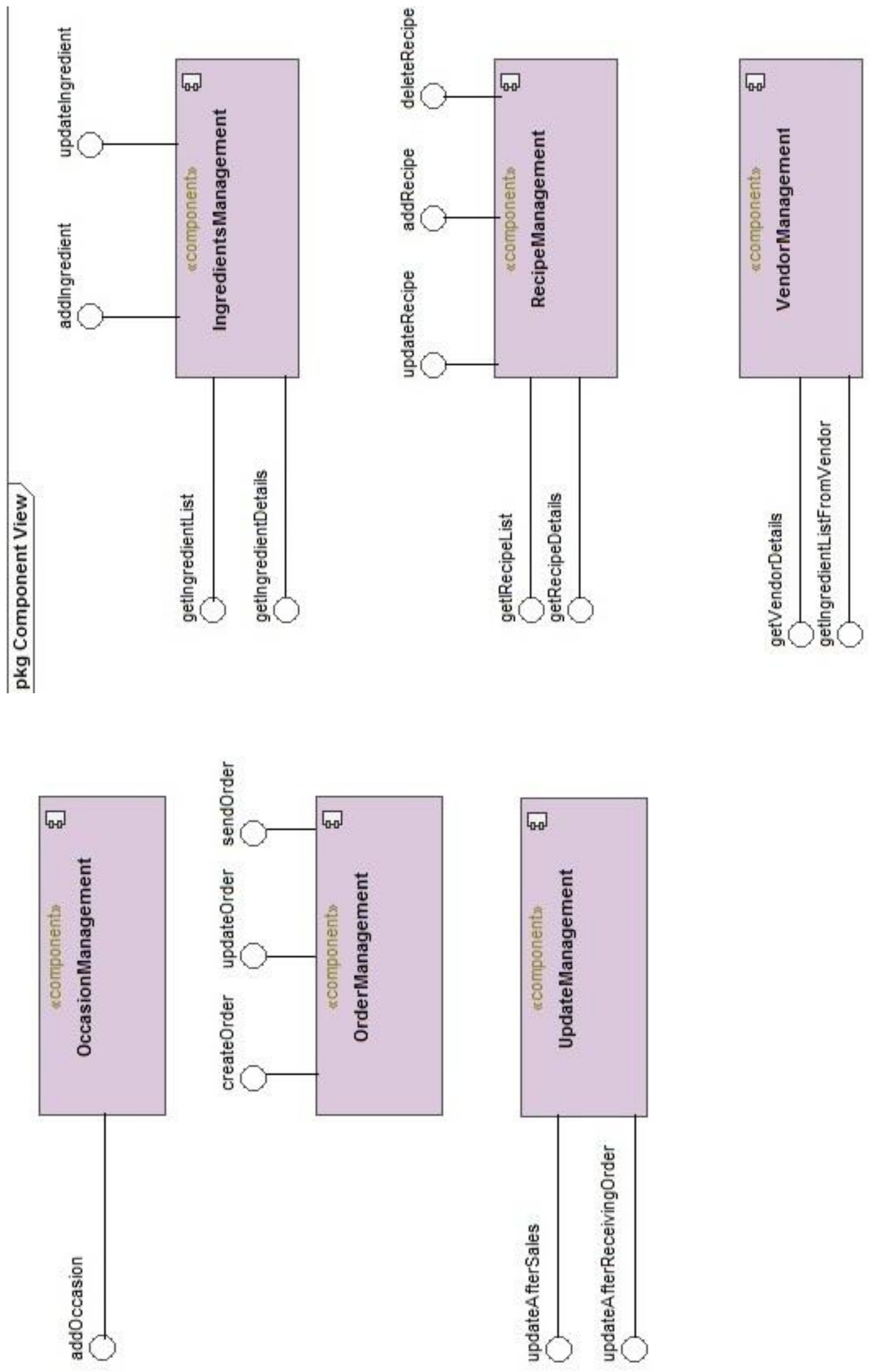


Figure 18: Services Diagram

4.9 Object Design Tradeoffs

- **Space vs. Speed:** The product is based on data inventory management and thus requires a lot of space for the storage of data. Thus to make the system read and write data at a faster speed we need more memory space.

- **Build vs. Purchase:** This product mostly uses open source products as development tools, so the scope of purchase is minimized of off the shelf products required for this product. Also this product uses open source libraries and source codes for some parts thus avoiding the Build vs. Buy dilemma.

- **Delivery time vs. Functionality**: As this project is running on a tight schedule, it will be difficult to produce a system with all the functionalities that are mentioned in the specifications by the date of delivery but a prototype system with the important functionality must be ready by the requested date. Functionalities such as prediction of data can be delivered at a later time.

4.10 Interface Documentation Guidelines

Below are mentioned guidelines for the interface documentation:

- The names of the class and class attributes should start with an uppercase letter and if the class name consists of more than one word then *CamelCasing* should be used.

Example: Public class Ingredient{}

- The class methods begin with a lowercase and if the method name consists of more than one word, *camelCasing* should be used.

Example: public boolean addIngredient()

- The Package names should start with and uppercase letter and use CamelCasing when package names consist of multiple names. Also the package names must end with the word ‘Package’

  Example: package *IngredientsPackage*

- Constants are represented by all uppercase letters and constant names with multiple words should be separated with underscores (‘_’).

Example: TOTAL_NUMBER_OF_RECIPES

- The class diagrams represent the access specifiers using symbols that can be summarized in the image shown below

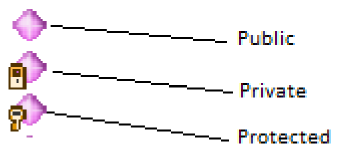


Figure 19: Attributes Naming Convention

- The method and attribute names should be such that they are self-explanatory of their context of use.

Example: If the method adds a new recipe to the database the name of the method should be *addRecipe*.

If an attribute holds the value of the name of the ingredient, the name of the attribute should be *IngredientName.*

- Comments must be used extensively to make the code easily understandable. All the classes must have preface comments, all the methods must have the functionality commented in the code and the attributes must have their usage comments too.

4.11 Packages

The figure below gives an overview of the package composition of the system.

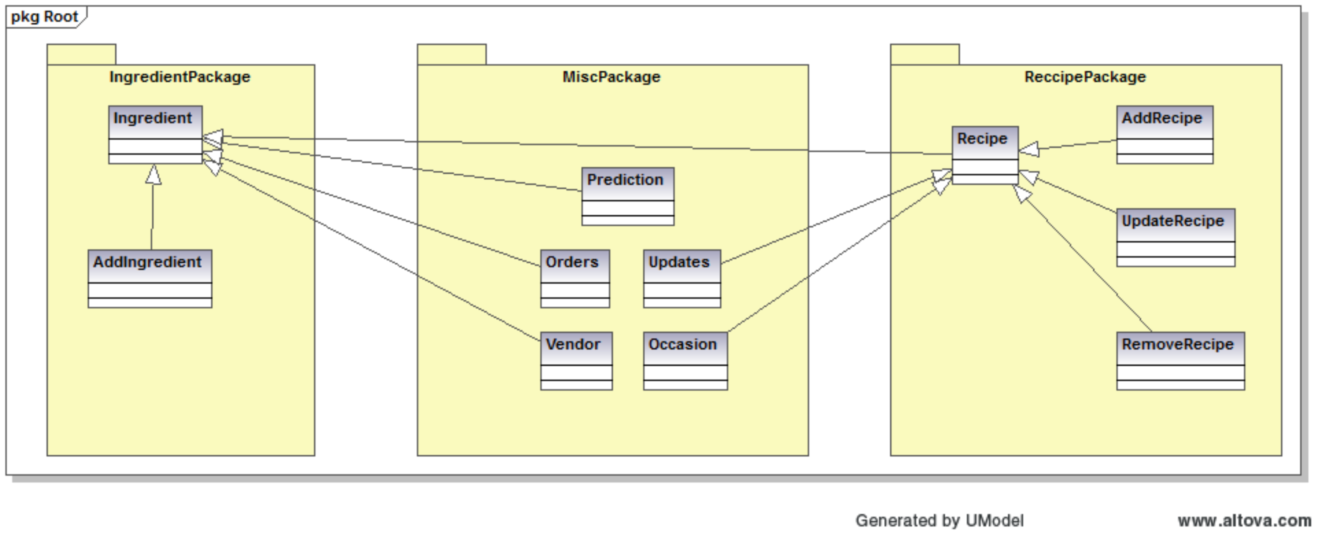


Figure 20: Packages Diagram

Package Description

4.11.1 IngredientPackage:

- ✓ This package constitutes of two classes namely the Ingredient and Add Ingredient which forms the basis of the system.
- ✓ This package provides functionalities to the user for adding new ingredients in the database. It also provides the current list of ingredients in the database and the corresponding details.
- ✓ These functions act as a basic functionality for the classes that inherit the classes in the IngredientPackage.
- ✓ For instance, the Recipe class requires the list of ingredients to assign ingredients to the recipe that is being added, the list of ingredient is required for creating orders for the corresponding recipe. Thus in short this package has classes that provide vital functionality to the classes in the other two packages.

4.11.2 MiscPackage:

- ✓ This package constitutes of five classes which perform tasks quite different to each other and are linked to the other two packages by the connections differing in functionality.
- ✓ The classes namely are Prediction, Orders, Updates, Vendor and Occasion. The Prediction and Occasion classes can be considered as a special feature wherein ingredients are constantly being used and if any occasion is near, then the prediction is done in accordance with the existing levels and past history of usage.
- ✓ The Vendor and Orders are intertwined amongst themselves as in the usage. If low inventory levels are sensed then the orders of required ingredients are passed by to the manager and an order form is generated which is given to the vendor for the replenishing the inventory. Once the stock levels are

more than the threshold level then the update can be performed and the new levels are taken into consideration.

4.11.3 RecipePackage:

- ✓ This Package Constitutes the Recipe, AddRecipe and RemoveRecipe which forms its classes. The AddRecipe classes is usually used so as to include a new row in the recipe table and which in turn is linked to the ingredients as when the following is utilized it indirectly uses up the ingredients involved.
- ✓ So along with the recipe, all the links to the ingredient are mandatory. Removing the recipe from the recipe list may not affect the ingredients as the one to many relation for the recipe and ingredients is still preserved.
- ✓ This package acts as an interface for the user, usage in order to make changes into the inventory with perspective to usage. The recipe details usually include the recipe name, recipe ID and the associated Ingredient ID as well. These classes mentioned above are inter linked so as to form a cohesive output.

4.12 CLASS INTERFACES

The figure below shows the class diagram of the whole system.

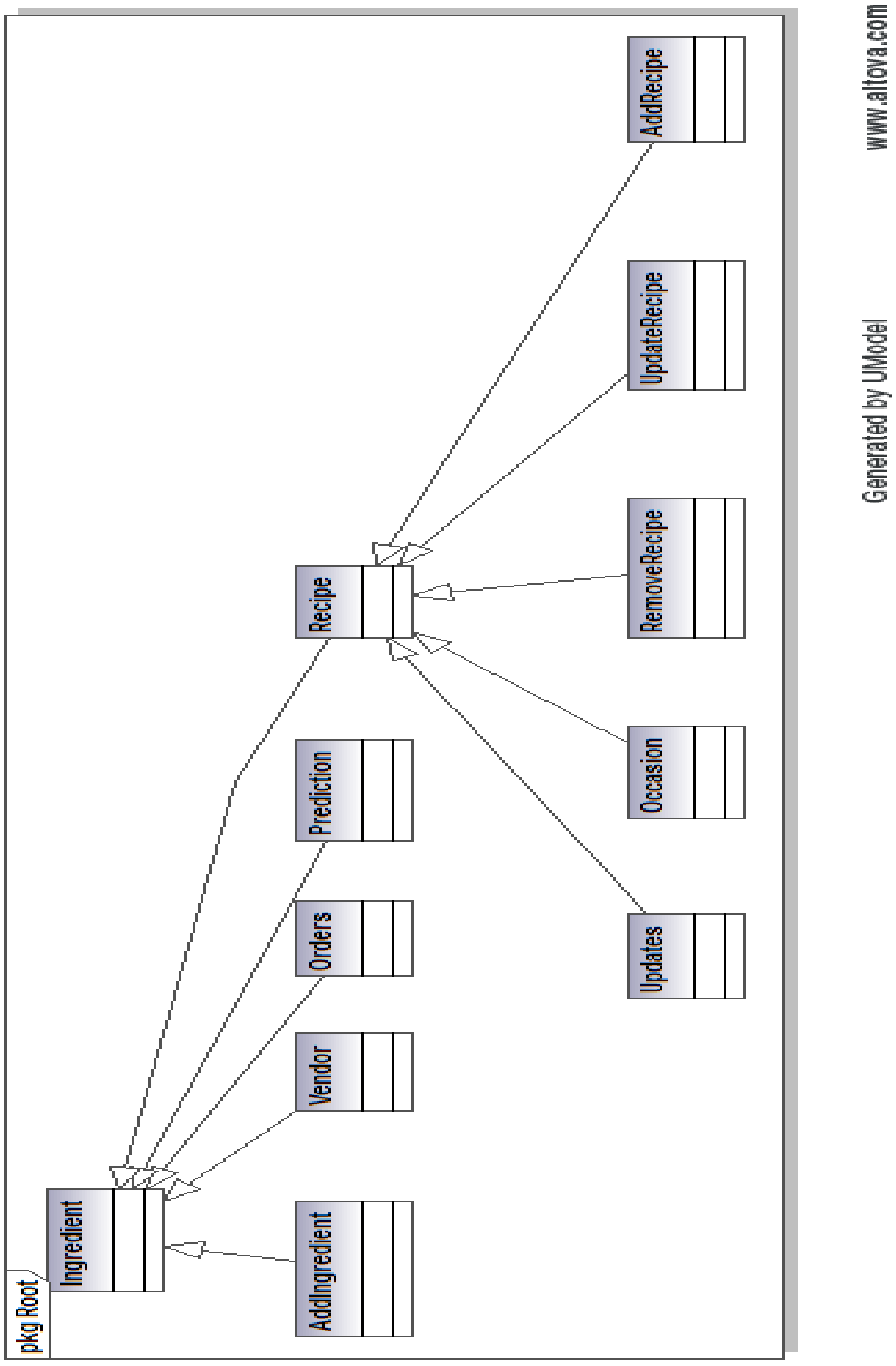


Figure 21: Overall Class Diagram

The figure above gives an overview of the class structure of the system. The details of the attributes and functions along with access specifiers, return types and parameters are listed in the diagrams below.

4.12.1 Class Ingredient

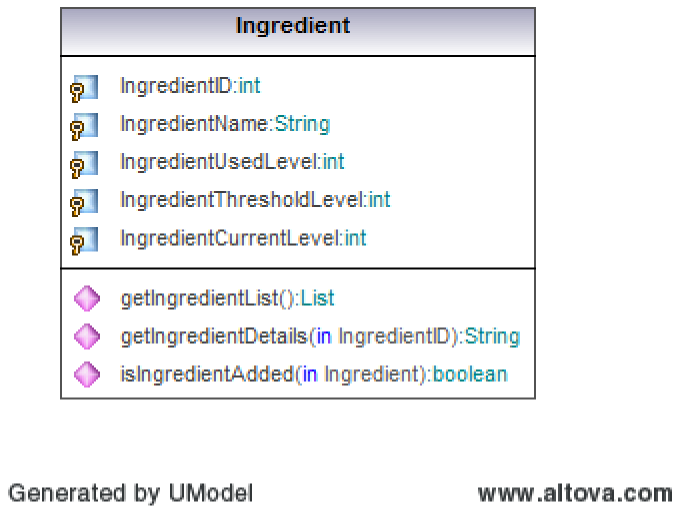

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
Generated by UModel
www.altova.com

Figure 22: Class Ingredient

4.12.2 Class AddIngredient

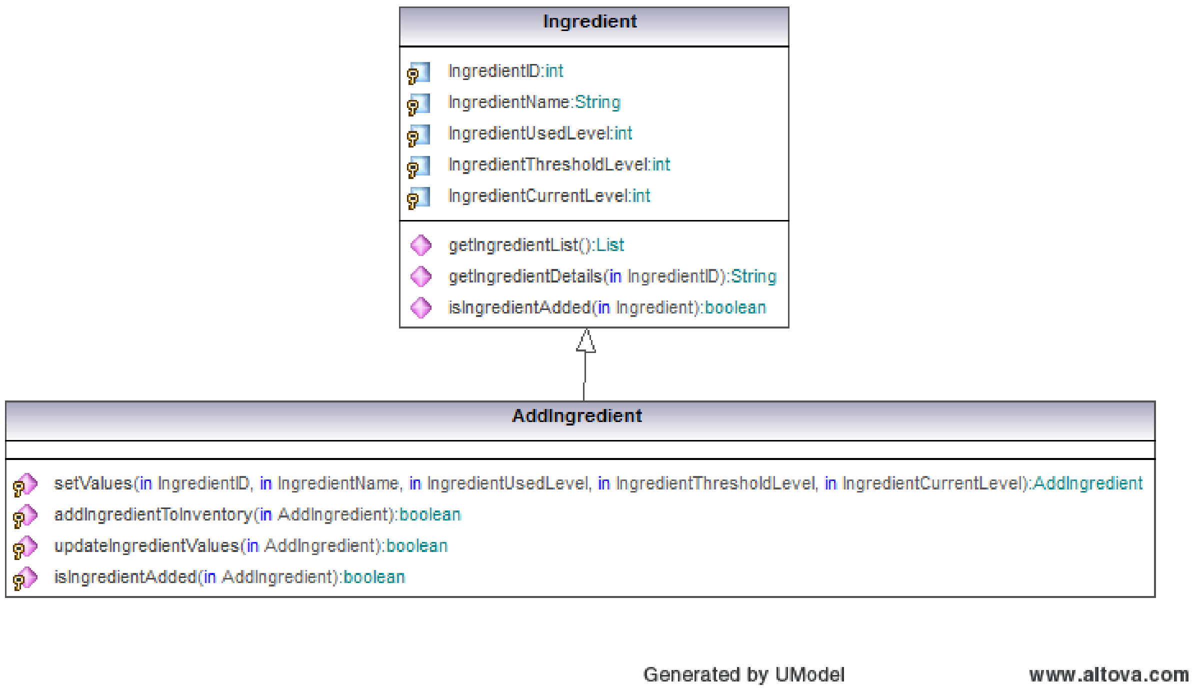

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
AddIngredient
setValues(in IngredientID, in IngredientName, in IngredientUsedLevel, in IngredientThresholdLevel, in IngredientCurrentLevel):AddIngredient
addIngredientToInventory(in AddIngredient):boolean
updateIngredientValues(in AddIngredient):boolean
isIngredientAdded(in AddIngredient):boolean
Generated by UModel
www.altova.com

Figure 23: Class AddIngredients

4.12.3 Class Recipe

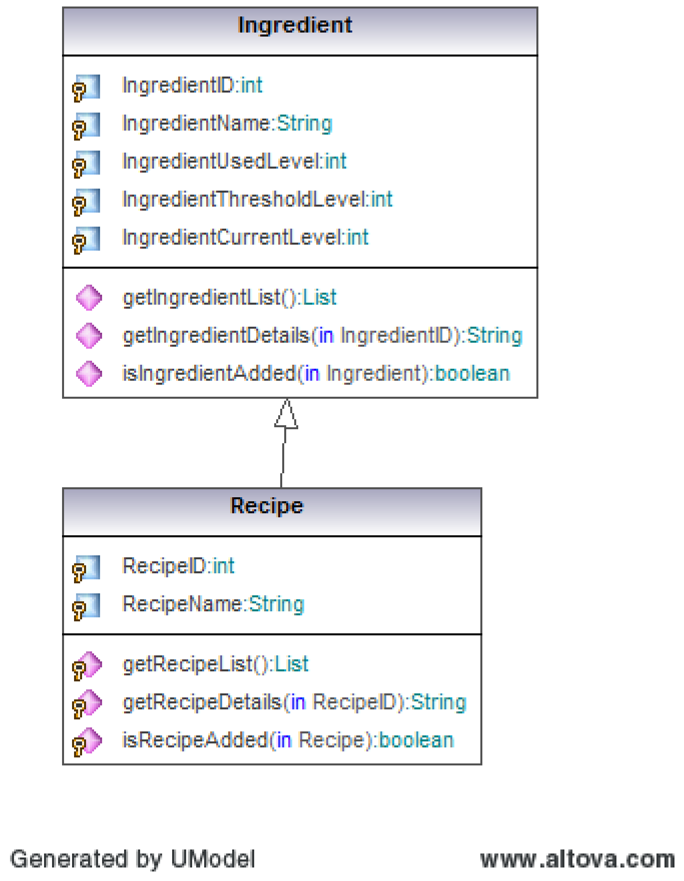

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
Generated by UModel
www.altova.com

Figure 24 : Class Recipe

4.12.4 Class Vendor

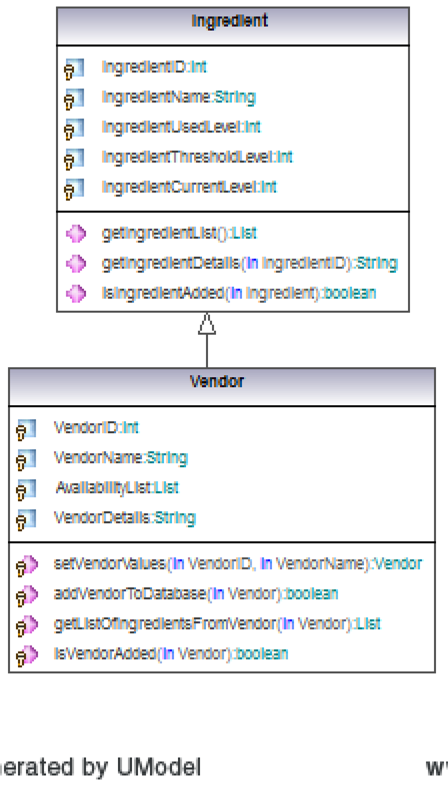

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
Vendor
VendorID:int
VendorName:String
AvailabilityList:List
VendorDetails:String
setVendorValues(in VendorID, in VendorName):Vendor
addVendorToDatabase(in Vendor):boolean
getListOfIngredientsFromVendor(in Vendor):List
isVendorAdded(in Vendor):boolean
Generated by UModel
www.altova.com

Figure 25: Class Vendor

4.12.5 Class Prediction

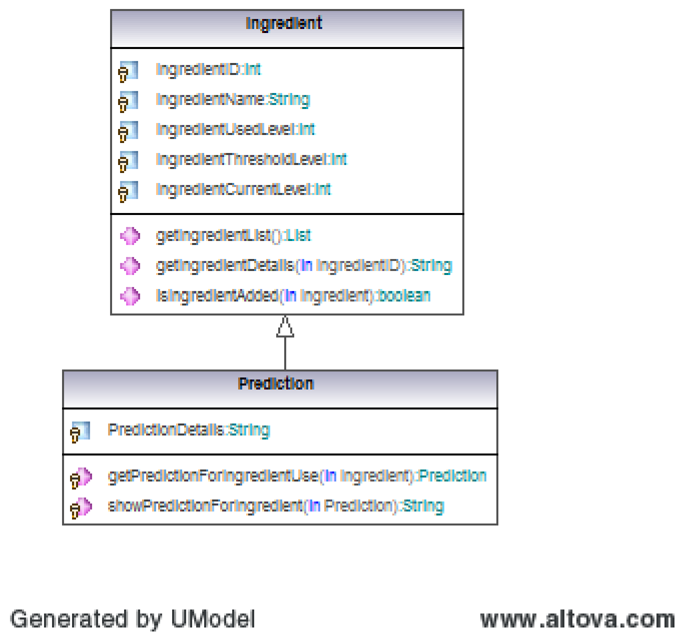

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
Prediction
PredictionDetails:String
getPredictionForIngredientUse(in Ingredient):Prediction
showPredictionForIngredient(in Prediction):String
Generated by UModel
www.altova.com

Figure 26: Class Prediction

4.12.6 Class AddRecipe

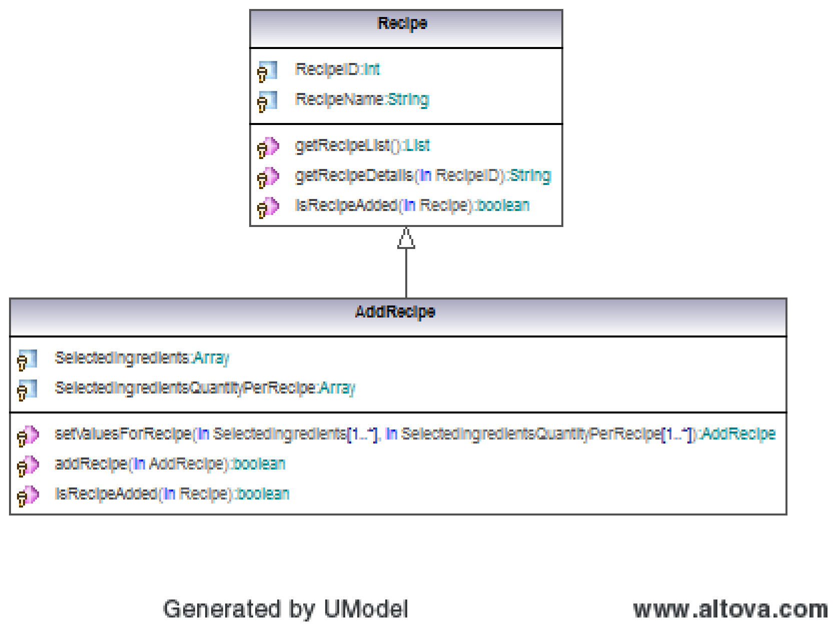

Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
AddRecipe
SelectedIngredients:Array
SelectedIngredientsQuantityPerRecipe:Array
setValuesForRecipe(in SelectedIngredients[1..*], in SelectedIngredientsQuantityPerRecipe[1..*]):AddRecipe
addRecipe(in AddRecipe):boolean
isRecipeAdded(in Recipe):boolean
Generated by UModel
www.altova.com

Figure 27: Class AddRecipe

4.12.7 Class RemoveRecipe

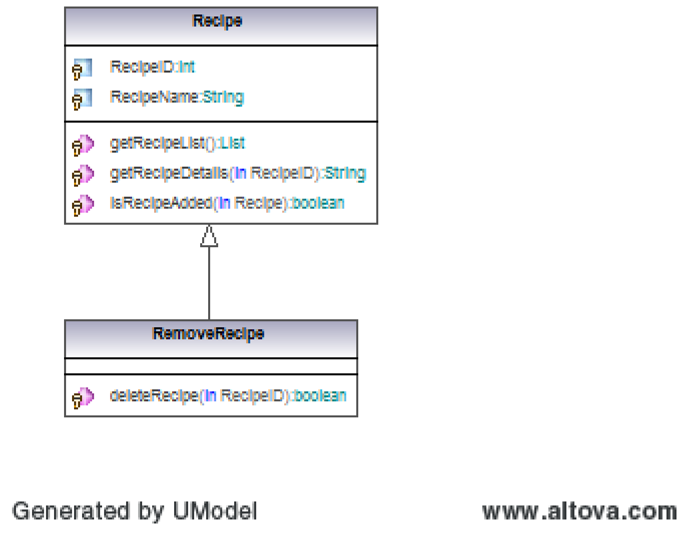

Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
RemoveRecipe
deleteRecipe(in RecipeID):boolean
Generated by UModel
www.altova.com

Figure 28: Class RemoveRecipe

4.12.8 Class UpdateRecipe

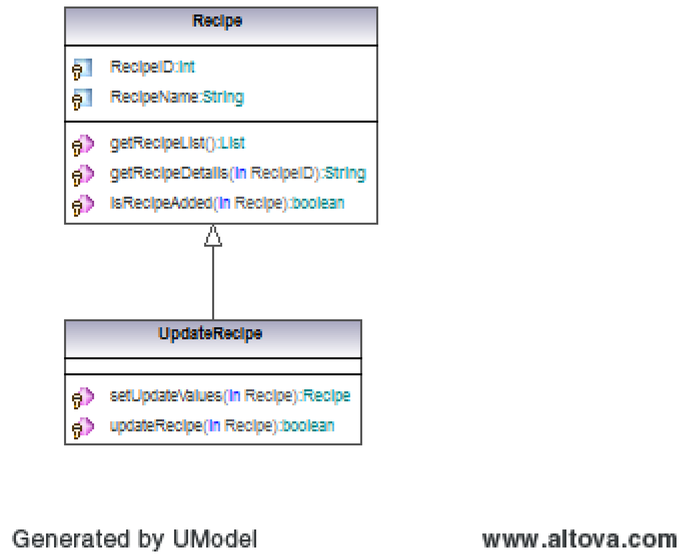

Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
UpdateRecipe
setUpdateValues(in Recipe):Recipe
updateRecipe(in Recipe):boolean
Generated by UModel
www.altova.com

Figure 29: Class UpdateRecipe

4.12.9 Class Updates

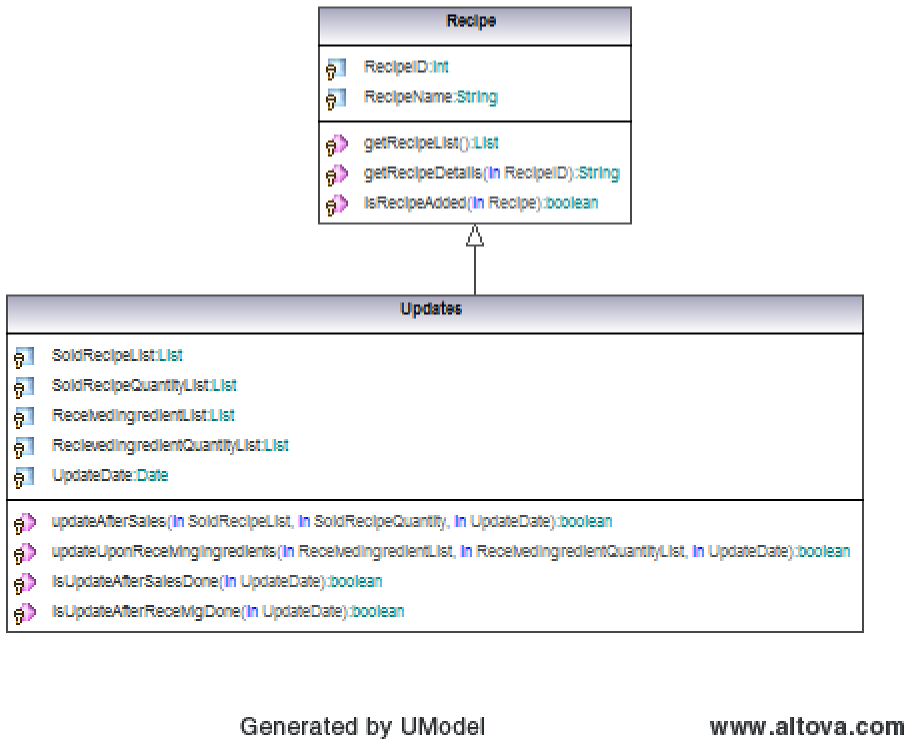

Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
Updates
SoldRecipeList:List
SoldRecipeQuantityList:List
RecievedIngredientList:List
RecievedIngredientQuantityList:List
UpdateDate:Date
updateAfterSales(in SoldRecipeList, in SoldRecipeQuantity, in UpdateDate):boolean
updateUponReceivingIngredients(in RecievedIngredientList, in RecievedIngredientQuantityList, in UpdateDate):boolean
isUpdateAfterSalesDone(in UpdateDate):boolean
isUpdateAfterReceivigDone(in UpdateDate):boolean
Generated by UModel
www.altova.com

Figure 30: Class Updates

4.12.10 Class Occasion

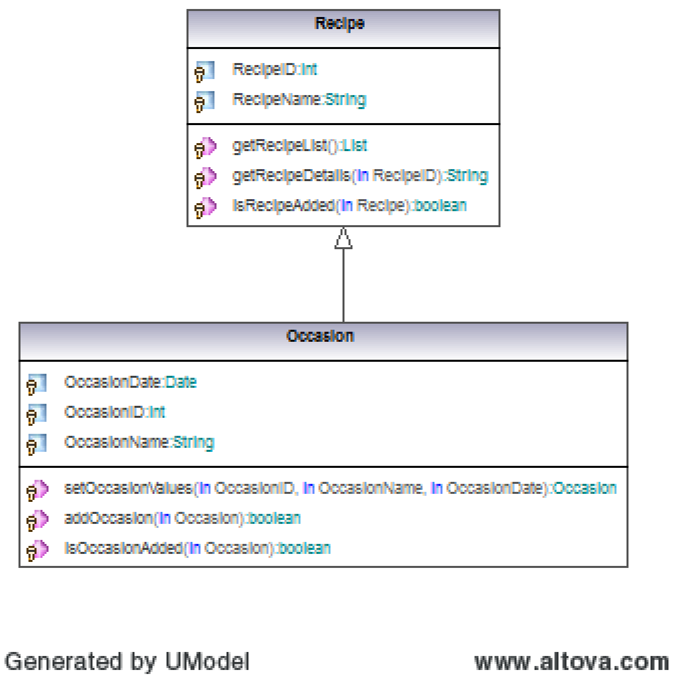

Recipe
RecipeID:int
RecipeName:String
getRecipeList():List
getRecipeDetails(in RecipeID):String
isRecipeAdded(in Recipe):boolean
Occasion
OccasionDate:Date
OccasionID:int
OccasionName:String
setOccasionValues(in OccasionID, in OccasionName, in OccasionDate):Occasion
addOccasion(in Occasion):boolean
isOccasionAdded(in Occasion):boolean
Generated by UModel
www.altova.com

Figure 31: Class Occasion

4.12.11 Class Orders

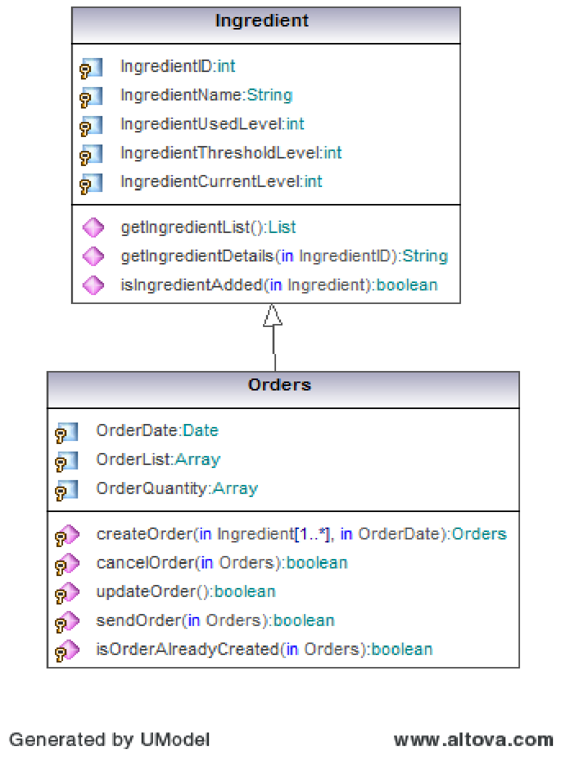

Ingredient
IngredientID:int
IngredientName:String
IngredientUsedLevel:int
IngredientThresholdLevel:int
IngredientCurrentLevel:int
getIngredientList():List
getIngredientDetails(in IngredientID):String
isIngredientAdded(in Ingredient):boolean
Orders
OrderDate:Date
OrderList:Array
OrderQuantity:Array
createOrder(in Ingredient[1..*], in OrderDate):Orders
cancelOrder(in Orders):boolean
updateOrder():boolean
sendOrder(in Orders):boolean
isOrderAlreadyCreated(in Orders):boolean
Generated by UModel
www.altova.com

Figure 32: Class Orders

5 TESTING

5.1 FEATURES TO BE TESTED/NOT TO BE TESTED

5.1.1 Features to be tested

Below table lists the features that will be tested during the current test or the subsequent planned tests

Features to be tested	**Test Description**
Login to the system	This tests the login interface of the system.
Adding a Recipe to database	This test is conducted to verify if a recipe is successfully added to the database. This will check if the recipe is added to its header table and also check if the recipe details are added to the recipe details table.
Adding an Ingredient to database	This tests checks if new ingredient is added correctly to the database with the specified details.
Adding a Vendor to the database	This test checks if the newly added vendor is correctly added to the database with the specified details.
Checking the threshold levels	This test is conducted to verify if the ingredients that are below the threshold levels are listed by the function when called by the user. The verification is done by referring to the database.
Updating the sales for the day	This test is conducted to test the sales update in the database. The test checks if the database is updated with the correct ingredient values based on the sales data input to the system.
Updating the order reception to database	This test is conducted to test the correct updating of the database after receiving the order from the vendor.
Create Orders	This test is conducted to check the order creation capability of the system. The list of ingredients that is generated for order must comply with the set conditions of threshold levels

Process Orders	This test is conducted to test if the created orders are processed correctly into a file.
Updating a recipe	This test checks for the correct updating of the selected recipe.
Deleting a recipe	This test checks for the deletion of the selected recipe by the user. Also it has to check it corresponding entries in the related tables are deleted correctly.
Manager Interface	This test will determine if the user is able to navigate through the interface and if the user can access the all the functions of the system navigating via the main manager interface.

Table 16: Features to be tested

5.1.2 Features not to be tested

Features not to be tested	**Reason for not testing**
Prediction	Prediction requires a lot of past usage data to function accurately. But, the system which is to be tested is a prototype and is being tested on a small set of data. Even if a large set of dummy database were to be created, there will not be any way in which the accuracy of prediction can be determined unless matched with a real-time data.
Correcting the inventory	This feature actually tweaks the prediction and the threshold whenever the correction is performed and as this deals with the proper functioning of prediction, the data is unverifiable and hence will not be tested.
Occasion Management	This feature again deals with the prediction of ingredient usage for different occasions set by the user and hence are not completely verifiable and thus skipped in this test.

Table 17: Features not to be tested

****Note: For detailed test cases and results, refer to Testing Document submitted earlier. [11].***

5.2 Pass/Fail Criteria

The main motive of testing is to find faults in the system/component so that they can be dealt with in the future. Thus it is really necessary to define the pass/fail criteria of the system to know which areas of the system require a developer's attention.

Here, in each of the test case, we define the Expected result as well as the Actual Result. Now after testing each test case with a variety of boundary inputs, we are in a state to compare the expected result to the actual result received.

If the actual result is in agreement with the expected test result, we term the test as *pass* and if the actual test result and the expected result vary, the test is termed as *fail*. Whenever a test fails, we know which input/sequence of inputs caused the test to fail and therefore can deal with it so that the same type of input does not cause errors in the system at a later stage.

5.3 Approach

The tests carried out follow for the most part of the testing the *unit testing strategy* and to be more specific, boundary testing. Here, we try to figure out various input values, scenarios in which the user can interact with the system/component, and test if the expected output matches the actual result of the system. This will help us know the various errors/exceptions that can occur in the system/component with the range of possible inputs.

For each input field/case we have tried and considered various extreme inputs that the user can accidentally or intentionally feed in which may cause the system to break or the database inventory to slip. The result of these tests will help us to know on which user input the system falters (does not function normally) and thus precautions can be taken to prevent the any changes being submitted to the database for the incorrect inputs.

Further, during the testing, we also test if the correct user inputs are correctly updates to the database, precisely checking if the data reached its target table at the correct place. This test will requires all the user inputs to be valid to be successfully tested.

As the system is still under development, the testing of the whole system is not possible. Thus, we choose to integrate the components that are developed and test their functionality when under this integration. In this part, we choose to check if the user is able to access the functions that are available and if the user is able to go through the interface without trouble.

5.4 Suspension and Resumption

5.4.1 Suspension

The item that are listed under the *Features not to be tested table* are the items suspended from the testing. Also some items that are not tested completely remain suspended from the testing. The items can be summarized as follows

- Recipe Management
    - Updating a recipe.
    - Deleting a recipe.
- Ingredient Management
    - Deleting an ingredient
    - Correcting the ingredient quantity in the inventory
- Vendor Management
    - Deleting a vendor
- Occasion Management
    - Adding/Removing an occasion
    - Predicting the requirements for the occasion day
- Prediction
    - Predicting requirements in general
    - Alerting the user of predicted low inventory level for a specific day

5.4.2 Resumption

As the system is currently under development, the above mentioned features are currently suspended from testing as they are partially developed or the development for them have not yet started.

Whenever the testing for the suspended features is resumed, it will be utterly necessary to redo the testing that are currently performed as only then it will ensure the integrity that any change that was made during development of components or any new component that was added during the development process. This will help us know that the development process has not introduced any new errors in the system and new components were added to the system without introducing any more complexity.

Many of the components in the program manipulate data in the database. Any components that have been changed may cause changes to how the database is read from and written to. That's why it is important to essentially retest any affected components (subsystems that also use the same table in the database). If we did not do this, we may experience catastrophic database errors.

5.5 TESTING MATERIALS (HARDWARE/SOFTWARE REQUIREMENTS)

5.5.1 Software requirements

The coding for the system is being done on Netbeans IDE for Java and the database management utility that is being used is MySQL. As the system uses no network connectivity, the system does not require any other specialized software for network connectivity. Any operating system that supports these two modules is a perfectly suitable OS for the testing purposes.

- ✓ Netbeans IDE for Java
- ✓ MySQL
- ✓ Windows XP, 7.

Also one driver is required for facilitating the connectivity between Netbeans and MySQL and that is MySQL Connector/J 5.1.6. This driver must be explicitly imported in the project directory of Netbeans.

5.5.2 Hardware Requirements

The hardware requirement of the testing is specifically a Laptop/Desktop with the following minimum hardware configuration

- ✓ **Processor:** 800MHz Intel Pentium III or equivalent
- ✓ **Memory:** 512 MB
- ✓ **Disk space:** 750 MB of free disk space.

5.6 TEST CASES

5.6.1 Test case 1: Testing the Add Recipe Interface and its functioning

- ✓ Case 1.1: Testing the Quantity input field.
- ✓ Case 1.2: Testing the Recipe Name field.
- ✓ Case 1.3: Testing the Ingredients in recipe list and Quantity of ingredient list.
- ✓ Case 1.4: Testing the available ingredients list.
- ✓ Case 1.5: Testing the all the above cases together and checking if the entries are updated to the tables in database.

5.6.1.1 Test case specifications for Test case 1: Testing the Add Recipe Interface and its functioning

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**	**Interface Dependencies**
Case 1.1	Quantity text field	1) Input negative numbers. 2) Input String 3) Input zero. 4) Input floating point numbers. 5) Leave the field blank 6) Enter special character in the field. 7) Input integer numbers greater than zero less than	1) Input specifications 1, 2, 3, 4, 5, 6 & 8 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 7 should not generate any error.	Select an ingredient from the ingredient list and enter a quantity in the quantity field and press add to recipe button.	N.A

		10000. 8) Input integer number greater than 9999.			
Case 1.2	Recipe Name Field	1) Input numerical value for the name 2) Leave the field blank 3) Enter an existing recipe name. 4) Enter special characters in the field 5) Enter a non-existing recipe name, (string)	1) Input specifications 1, 2, 3 and 4 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 5 must not generate an exception except for a really long string (more than 50 characters)	Enter a name for the recipe, add some ingredient to the list along with appropriate quantity and press the submit button.	N.A
Case 1.3	Ingredient in recipe list & corresponding Quantity list	1) List is left empty. 2) One ingredient is added twice to the list.	1) The input specification 1 & 2 result in an exception being thrown.	For the first input: Enter an appropriate Recipe Name and press *Add to database button*. For the second input: Selected one ingredient, enter an appropriate	N.A

				quantity and press *Add to Recipe* button twice.	
Case 1.4	Testing the available ingredients list.	N.A	The List of ingredients must show all the ingredients that are currently in the database	N.A	N.A
Case 1.5	Testing the components mentioned above together and adding a recipe to the database	1) All the required quantities are inserted into their respective fields.	1) If all the above tests are passed without an exception, the recipe is successfully added to the database	1) Enter a recipe name. 2) Select Ingredient from the ingredient list and enter quantity amount for the recipe and press the add to recipe button. 3) Repeat step 2 until all the desired ingredients are added to the list. 4) Press the Submit button	N.A

Table 18: Test case specifications for Test case 1: Testing the Add Recipe Interface and its functioning

5.6.1.2 Preliminary test results for test case 1

Test Case	**Completed / Not Completed**	**Result summary**
Case 1.1	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 1.2	Completed	The results for the mentioned input specifications have been passed except for the inputs × Recipe Name= 1223234 × Recipe Name = %$^&$
Case 1.3	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 1.4	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 1.5	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.

Table 19: Preliminary test results for test case 1

5.6.2 Test case 2: Logging in to the system

This case will test the login system. The test must be conducted to see if access is allowed only to the authenticated users (in this case it is only one user i.e. the Manager). On Successful login, the main interface must be visible to the user.

5.6.2.1 Test case specifications for Test case 2: Logging in to the system

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**	**Interface Dependencies**
Case 2.1	Login text field and password field	1) Login name is incorrect. 2) Login name is correct but password is incorrect. 3) Login name or password is blank or both are blank. 4) Login Name and password both are correct.	1) The input specifications 1, 2 & 3 must generate an exception and ask the user to input the credentials again 2) The input specification 4 must show the user *Main Interface*	Enter the login name and password and press the login button.	N.A

Table 20: Test case specifications for Test case 2: Logging in to the system

5.6.2.2 Preliminary test results for test case 2

Test Case 2	**Completed/Not Completed**	**Result Summary**
Case 2.1	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.

Table 21: Preliminary test results for test case 2

5.6.3 Test Case 3: Testing the Add Ingredient Interface of the system

This case will test the Add Ingredient processing. The test must be conducted to ensure whether when user (Manager) when text inputs the ingredient name and the quantity in the provided text fields of the Add ingredient form. We must make sure that the each of the fields are validated and proper input should certify and reflect the changes in the database and also when quantity check is done.

- ✓ Case 3.1: Test the Ingredient name field.
- ✓ Case 3.2: Test the Threshold value field.
- ✓ Case 3.3: Test the Current Quantity field.
- ✓ Case 3.4: Test the select vendor field.
- ✓ Case 3.5: Test the Current Ingredient list field.

5.6.3.1 Test case specifications for Test case 3: Testing the Add Ingredient Interface of the system

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**	**Interface Dependencies**
Case 3.1	Ingredient Name field	1) Input numerical value for the name. 2) Leave the field blank 3) Enter an existing Ingredient name. 4) Enter special characters in the field. 5) Enter a non-existing Ingredient	1) Input specifications 1, 2, 3 and 4 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 5 must not generate an exception in general except for a really long name (more than 25 characters)	Enter the Ingredient Name, appropriate threshold and current quantity values, select a vendor and Press *Submit* button.	N.A

		name, (string)			
Case 3.2	Threshold value field	1) Input negative numbers. 2) Input String 3) Input zero. 4) Input floating point numbers. 5) Leave the field blank 6) Enter special characters in the field. 7) Input integer numbers greater than zero less than 10000 8) Input integer numbers greater than 9999.	1) Input specifications 1, 2, 3, 4, 5, 6 & 8 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 7 should not generate any error.	Enter an appropriate Ingredient Name, threshold values, current quantity, and select a vendor and then press the submit button.	N.A
Case 3.3	Current Quantity field	1) Input negative numbers. 2) Input String 3) Input zero. 4) Input	1) Input specifications 1, 2, 3, 4, 5, 6 & 8 must generate exceptions asking the user to re-enter the text	Enter an appropriate Ingredient Name, threshold values, current quantity, and select a vendor and	N.A

		floating point numbers. 5) Leave the field blank 6) Enter special characters in the field. 7) Input integer numbers greater than zero less than 10000 8) Input integer numbers greater than 9999.	in the field. 2) Input specification 7 should not generate any error.	then press the submit button.	
Case 3.4	Select Vendor Drop down box	Load the form/Activate the Add Ingredient function	The combo box for select vendor should show all the available vendors from the database	The combo box for select vendor shows all the available vendors from the database	NA
Case 3.5	Current Ingredient list	Activate the Add ingredient function	The Current Ingredient List must show all the ingredients from the database	The Current Ingredient List must show all the ingredients from the database	N.A

Table 22: Test case specifications for Test case 3

5.6.3.2 Preliminary test results for test case 3

Test case	**Completed/Not Completed**	**Results Summary**
Case 3.1	Completed	The test results for this test case has been passed for most input specifications but fails for the inputs mentioned below × Ingredient Name= 123123 *(The test fails for numerical inputs)* × Ingredient Name = %&*&^ *(The test fails for special character inputs)*
Case 3.2	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 3.3	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 3.4	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 3.5	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.

Table 23: Preliminary test results for test case 3

5.6.4 Test Case 4: Testing the Add vendor Interface of the system

- ✓ Case 4.1 Test the Vendor name field.
- ✓ Case 4.2 Test the vendor type field.
- ✓ Case 4.3 Test the vendor details field.
- ✓ Case 4.4 Test the email address field.

5.6.4.1 Test case specification for test case 4: Testing the Add vendor Interface of the system

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**
Case 4.1	Vendor Name field	1) Input numerical value for the name. 2) Leave the field blank. 3) Enter an existing Vendor name. 4) Enter special characters in the field. 5) Enter a non-existing recipe name. (string)	1) Input specifications 1, 2, 3 and 4 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 5 must not generate an exception in general except for a really long name (more than 25 characters)	Input all the required fields and press the Submit button
Case 4.2	Vendor Type Field	1) Input numerical value for the name. 2) Leave the field blank. 3) Enter an existing Vendor type.	1) Input specifications 1, 2, and 4 must generate exceptions asking the user to re-enter the text in the field. 2) Input	Input all the required fields and press the Submit button

		4) Enter special characters in the field. 5) Enter a non-existing recipe name. (string)	specification 5 must not generate an exception in general except for a really long name (more than 25 characters) 3) Input specification 3 must not result in an exception.	
Case 4.3	Vendor Details Field	1) Leave the field blank 2) Input details in the field	1) Input specification 1 must result in an exception 2) Input specification must not result in an exception	Input all the required fields and press the Submit button
Case 4.4	Vendor Email Field	1) Leave the field Blank 2) Input email in an incorrect format 3) Input email in a correct format.	1) Input specifications 1 and 2 must generate exception. 2) Input condition 3 must not generate an exception	Input all the required fields and press the submit button.

Table 24: Test case specification for test case 4: Testing the Add vendor Interface of the system

5.6.4.2 Preliminary Test Results for test case 4

Test Case	**Completed/Not Completed**	**Result Summary**
Case 4.1	Completed	The test results for this test case has been passed for most input specifications but fails for the inputs mentioned below × Vendor Name= 123123 *(The test fails for numerical inputs)* × Vendor Name = %&*&^ *(The test fails for special character inputs)*
Case 4.2	Completed	The test results for this test case has been passed for most input specifications but fails for the inputs mentioned below × Vendor Type= 123123 *(The test fails for numerical inputs)* × Vendor Type = %&*&^ *(The test fails for special character inputs)*
Case 4.3	Completed	The test results for this test case has been passed for most input specifications but fails for the inputs mentioned below × Vendor Details = Patel Brothers, Devon Street, Chicago, IL *(Duplicate Details)* × Vendor Details = $^%$^% *(The test fails for special character inputs).*
Case 4.4	Completed	The test results for this test case has been passed for most input specifications but fails for the inputs mentioned below × Vendor Email = 123214 × Vendor Email = abc@xyz.com (existing email) × Vendor Email = $^%$^%

Table 25: Preliminary Test Results for test case 4

5.6.5 Test Case 5: Check Threshold Interface

- ✓ Test Case 5.1: Check if the Ingredients under the threshold values are shown in the Ingredients below threshold list.
- ✓ Test Case 5.2: Check if the Create order button asks the user to enter values for all the ingredients listed under the ingredients below threshold list.
- ✓ Test Case 5.3: Check if pressing the Process Order button creates a file with the order details in it.

5.6.5.1 Test case specification for test Case 5: Check Threshold Interface

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**
Case 5.1	Ingredients Below Threshold List	Press the Check Threshold Button	The Ingredients below threshold list must show all the ingredients below threshold level	Press the Check threshold button on the Check Threshold form
Case 5.2	Create Order Button	Press the Create Order Button	The user must be prompted to input order quantity for all the ingredients that are currently below threshold	Press the check threshold button and the press the check threshold button.
Case 5.3	Process Order Button	Press the process order button	A file with the order details must be created.	Press the check threshold button, then press the create order button and enter quantities for corresponding ingredients and then press the process order button.

Table 26: Test case specification for test Case 5

5.6.5.2 Preliminary Test Reports for test case 5

Test Case	**Completed/Not Completed**	**Result Summary**
Case 5.1	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 5.2	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.
Case 5.3	Completed	The results for all the input specification for this test is passed and no difference was detected between the actual and the expected results.

Table 27: Preliminary Test Reports for test case 5

5.6.6 Test Case 6: Testing the Update after sales interface

- ✓ Case 6.1 Test the Recipe list box.
- ✓ Case 6.2 Test the quantity text field.
- ✓ Case 6.3 Test the recipe sold list box quantity sold list box.
- ✓ Case 6.4: Test if the details are updated to the database when requested.

5.6.6.1 Test case specification for test Case 6: Testing the update after sales interface

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**
Case 6.1	Recipe List box	Load the Update After Sales interface	The Recipe list box must show all the current recipes on the database	N.A
Case 6.2	Quantity text field	1) Input negative numbers. 2) Input String 3) Input zero. 4) Input floating point numbers. 5) Leave the field blank 6) Enter special characters in the field. 7) Input integer numbers greater than zero less than 100 8) Input integer numbers greater than 99.	1) Input specifications 1, 2, 3, 4, 5, 6 & 8 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 7 should not generate any error	Select a recipe from the list and enter the quantity then press the add button.

Case 6.3	Recipe Sold List Box	1) Select a recipe and enter an appropriate quantity then press the add button	The Selected Recipe must Show in the Sold Recipe List Box along with the corresponding quantity in the quantity sold list box.	N.A
Case 6.4	Testing if the selected data is processed properly and updated to the database	1) Select recipe and enter a corresponding quantity press the add button and the when the recipe shows in the sold recipe list box, press the update button.	A dialog box “Success” must show and the database must be checked for appropriate updates.	N.A

Table 28: Test case specification for test Case 6: Testing the update after sales interface

5.6.6.2 Preliminary test results for test case 6

Test Case	**Completed/Not Completed**	**Result Summary**
Case 6.1	Not Completed	N.A
Case 6.2	Not Completed	N.A
Case 6.3	Not Completed	N.A

Table 29: Preliminary test results for test case 6

5.6.7 Test Case 7: Testing the Update After receiving interface

- ✓ Case 7.1: Check the Ingredient list box.
- ✓ Case 7.2: Check the Quantity text field.
- ✓ Case 7.3: Check the Ingredient Received and Quantity Received List boxes.
- ✓ Case 7.4: Check if the received Ingredient quantities are updated in the database.

5.6.7.1 Test case specification for Test case 7: Testing the update after receiving interface

Test case Identifier	**Test Items**	**Input Specifications**	**Output Specifications**	**Special Procedural Requirements**
Case 7.1	Ingredient list box	Load the Update After Receiving interface	The Ingredient list box must show all the current Ingredients on the database	N.A
Case 7.2	Quantity text field	1) Input negative numbers. 2) Input String 3) Input zero. 4) Input floating point numbers. 5) Leave the field blank 6) Enter special characters in the field. 7) Input integer numbers greater than zero less than 100 8) Input integer numbers greater than 99.	1) Input specifications 1, 2, 3, 4, 5, 6 & 8 must generate exceptions asking the user to re-enter the text in the field. 2) Input specification 7 should not generate any error	Select an ingredient from the list and enter the quantity then press the add button.

Case 7.3	Ingredient Received and Quantity Received List boxes.	1) Select an received ingredient, enter corresponding quantity and press the add button	1) The selected ingredient and quantity must show in the Ingredient received and Quantity Received list boxes	NA
Case 7.4	Testing if the selected data is processed properly and updated to the database	1) Select Ingredient and enter a corresponding quantity press the add button and the when the Ingredient shows in the sold recipe list box, press the update button.	A dialog box “Success” must show and the database must be checked for appropriate updates.	N.A

Table 30: Test case specification for Test case 7: Testing the update after receiving interface

5.6.7.2 *Preliminary Test Results for test case 7*

Test Case	**Completed/Not Completed**	**Result Summary**
Case 7.1	Not Completed	N.A
Case 7.2	Not Completed	N.A
Case 7.3	Not Completed	N.A
Case 7.4	Not Completed	N.A

Table 31: Preliminary Test Results for test case 7

5.7 Component Inspection

The following inspections look at the source code for the Check Threshold and Add Vendor interface. The logic code is located in the .java files, but there are more files that include the user interface design. Inspections were completed following Michael Fagan's inspection method and are documented below.

The inspection team was told to keep in mind that the NetBeans IDE automatically generated code associated with the creation of the object and GUI. This code was excluded from the inspection as tinkering with it may have caused errors in the GUI.

5.7.1 Inspection of Check Threshold

5.7.1.1 Overview

The 'Check Threshold' component is one of the features required for our project. The component allows the manager to see which items have fallen below their threshold. It also gives them the option to select items and amounts to be ordered. These orders are then generated automatically by the system. The text fields of the form are utilized for mapping and validating the inputs accordingly. The 'Check Threshold' form was designed, integrated and executed into the package by *'Simant Purohit'*.

5.7.1.2 Preparation

The review of form perfectly suggested the inputs and the outputs of the 'Check Threshold' component. The text fields had satisfied the criteria of validation, duplication and also inconsistency of the data. The reflections are also checked in the data for proper inputs of data. The form was reviewed by *'Bart Miczek'* and *'Akshay Thirkateh'*.

5.7.1.3 Inspection Meeting

The Inspection meeting was attended by *'Simant Purohit'*, *'Akshay Thirkateh'* and *'Bart Miczek'* and the key issues discussed are given as:

- ✓ String associated with the database connection (such as username, password, and connection string) should be global package variables. This allows the values set to these strings to be manipulated at a larger scale (shared by all components of the program). This way, if the password to access the database is changed, there is no need to go change it in each respective source code file. Rather, we change the global variable and it continues to be used throughout the program.
- ✓ Some "try" code segments weren't following by a "catch" segment in case an error does occur.
- ✓ The document generated by component is a simple text based file. This should be later expanded to a possible pdf document that can be emailed as a properly formatted order form.

5.7.1.4 Rework

Some of the changes were corrected satisfying the discussed criteria in the inspection meeting. The Rework was done by *'Simant Purohit'* and *'Bart Miczek'*.

5.7.1.5 Follow up
The follow up to the ‘Check Threshold’ form was attended by *‘Akshay Thirkateh’* and *‘Bart Miczek’* and no errors were found. Thus it resulted in a successful integration into the package.

5.7.2 Inspection of Add Vendor

5.7.2.1 Overview
The ‘Add vendor’ is one of the features required for our project. This option of adding a vendor links to the multiple tables in the database as every item is linked to the Ingredients and the changes being made to it. If a vendor is added then the ingredients which are provided by him are stored and when order form for the particular ingredients is being done then the appropriate vendor should be chosen. The text fields of the form are utilized for mapping and validating the inputs accordingly. The ‘Add vendor’ form was designed, integrated and executed into the package by *‘Simant Purohit’*.

5.7.2.2 Preparation
The review of form perfectly suggested the inputs and the outputs of the ‘Add vendor’. The text fields had satisfied the criteria of validation, duplication and also inconsistency of the data. The reflections are also checked in the data for proper inputs of data. The form was reviewed by *‘Akshay Thirkateh’ and ‘Bart Miczek’.*

5.7.2.3 Inspection Meeting
The Inspection meeting was attended by *‘Simant Purohit’, ‘Akshay Thirkateh’ & ‘Bart Miczek’*. The key issues discussed are given as:

· Prohibition of the use of special characters in the text field.

· Linking the table to the other tables.

· Linking the Generation of Order to the data accumulated by this form.

· Printing the Order form.

· Problem with the Catch method in some instances.

5.7.2.4 Rework
Some of the changes were corrected satisfying the discussed criteria in the inspection meeting. The Rework was done by *‘Simant Purohit’*.

5.7.2.5 Follow up
The follow up of the ‘Add vendor’ form was attended by *‘Akshay Thirkateh’ and ‘Bart Miczek’* and no errors were found. Thus it was successfully integrated into the package.

6 CONCLUSION:

The project "Inventory Control System for Calculation and Ordering of Available and Processed Resources" mainly as the name suggests deals with the calculation of the available and processed resources for an accurate inventory control and process management for a domain specific client who are related to the subject of food chains/outlets. This enables the inventory to be applied at every level in the hierarchy of the products and its complex combinations of recipes.

A system that accurately calculates the atomic ingredients used for making a recipe then automatically performs the back end operation pertaining to a database of many relational tables onto which the changes are being made with each and every operation performed on the front end and which also shows up if at the time of retrieval. The most important part of Inventory controlling is its ability to check for threshold levels and alert the manager to replenish the stock before it reaches a danger zone. So as when an ingredient level goes below the threshold level then it routes an alert to the manager. Then if needed accordingly an automated order form is produced so as to each specific vendor along with the quantities needed for replenishment.

As a part of the standard maintaining a drill of risk management is done in order to sustain during the days of special occasion or holidays when the demand reaches to rather more different scale as compared to other days. These occasions call on for special inclusions into the menu which reflects on the recipes and in turn reflects the ingredients being used up eventually. Thus was provided the liberty of adding special recipe to the menu for some special occasion and is regarded as a key feature.

To be able to simplify the user friendliness even more the concept of 'prediction' is added which enables the manager to see the past years prediction of the ingredients usage and then based on the informational analysis done on the data a prediction is then generated which would suit the requirements of the current year and then accordingly an appropriate order form is generated and then passed on to the vendor as the requirements for replenishing the stock.

7 Process Improvement:

The general software development process could have been more streamlined when it comes to the sharing of documents and code. Due to individual preferences, documents were shared through multiple interfaces including email, Google Drive, Microsoft SkyDrive, and USB sticks. Choosing a specific platform dedicated to the project would have been more ideal. There also exist online drives specific to programmers and code, such as Git-Hub and Google Code that we would utilize in future projects.

We also made the change from developing in Visual Basic to Java in a matter that worked for the group but would most likely be a rather large change in a larger group of developers. Such a change would have to be documented heavily as it influences all of the documents that had been generated to that point.

The process of generating the proposal, requirements, design, and testing documents feels like a very intuitive and streamlined process. What might have helped is programming the project prior to any documentation. This would then give us an idea of what we needed to achieve and what was necessary. A program that handles large amounts of data and requires the level of user interaction with a database was generally a new idea to us. A small test program would have helped put an idea of what we were trying to achieve at the very beginning of the process.

8 FUTURE DEVELOPMENT

There is a scope of lot of improvement given that the application of this project which is now limited to only food chains / outlets can be applied to other branches also given that it is subjected to appropriate changes. The prediction algorithm needs to be still more enhanced, but that is possible with years of data analysis and would then might also be changed to an artificially intelligent system. Also considering the large technological movement, access to the program through a web application would be ideal for remote access to the program and database. This would require a dedicated server to host the database and therefore has been considered as an optional enhancement.

To list, the future development may include the following

- ✓ Improvised Prediction.
- ✓ Use of AI to learn user behavior and responses.
- ✓ Expansion of software domain.
- ✓ Expansion to web domain.
- ✓ Allowing remote access.
- ✓ Allowing orders to be sent electronically.

9 BIBILIOGRAPHIC REFERENCES:

[1] Java Swing, 2nd Edition, Marc Loy, Robert Eckstein, Dave Wood, James Elliott, Brian Cole, O'Reilly Media-2002.

[2] Database programming with JDBC and Java, O'Reilly and George Reese, O'Reilly Media-2002.

[3] Bernd Bruegge & Allen H. Dutoit**.** Objec**t-Oriented Software Engineering** Using UML, Patterns, a**nd Java,** Third Edition, 2005.

[4] Object design tradeoffs for Project, http://www.scribd.com/doc/38302828/29/Object-design-trade-offs

[5] Technical Design Document. www.in.gov/fssa/files/QualCheck.pdf

[6] Union Design Pattern: Inheritance and Polymorphism. http://cnx.org/content/m11796/latest/

[7] CS 440 Fall 2012 Homepage. http://www.cs.uic.edu/~i440/

[8] Volere Requirement Resources. http://www.volere.co.uk/index.htm

[9] Requirements Analysis Document, submitted 19th October 2012. https://skydrive.live.com/redir?resid=2CEDE9F7F5F99604!195&authkey=!AJpVRmJ8w8giN_M

[10]Design Report, submitted 9th November 2012. https://skydrive.live.com/redir?resid=2CEDE9F7F5F99604!196&authkey=!AO5IghTCML6xAk8

[11]Testing Document, submitted 26th November 2012. https://skydrive.live.com/redir?resid=2CEDE9F7F5F99604!161&authkey=!AC8P0Lqe9amxSeM

[12]Project Proposal Document, submitted 21st September 2012. https://skydrive.live.com/redir?resid=2CEDE9F7F5F99604!175&authkey=!AB_dzPgl7Y2vDdw

10 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.

11 APPENDIX

11.1 TEST RESULTS FOR TEST CASE 1

Test case ID	**Input for the test**	**Expected result**	**Actual result**	**Pass/Fail**
Case 1.1	Quantity = -10	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 1.1	Quantity = xyz	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 1.1	Quantity = 0	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.1	Quantity = 12.4	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.1	Quantity =	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.1	Quantity = $#^	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.1	Quantity = 15	No error must be displayed and ingredient must be added to the list with the	No error is displayed and ingredient is added to the list with the	**Pass**

		corresponding quantity.	corresponding quantity.	
Case 1.1	Quantity = 10000	Error must be displayed.	An error is displayed and the user is asked to enter the quantity again.	**Pass**
Case 1.2	Recipe Name = 123214	Error message must be displayed.	Error message is not displayed	**Fail**
Case 1.2	Recipe Name =	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.2	Recipe Name = Kadai Paneer (existing recipe name)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 1.2	Recipe Name = $^%$^%	Error message must be displayed.	No error displayed	**Fail**
Case 1.2	Recipe Name = Cheese Pizza (non existing)	No error must be displayed that relates to this input	No error displayed that relates to this input	**Pass**
Case 1.2	Recipe Name= (a really long text with more than 50 characters)	An error must be displayed for too long name.	An error message is displayed.	**Pass**
Case 1.3	No data is added to the list.	Error must be displayed when Add to database button is pressed.	Error is displayed when Add to database button is pressed.	**Pass**
Case 1.3	User tries to add the same ingredient twice	Error must be displayed and the duplicate entry	Error is displayed and the duplicate	**Pass**

	to the Ingredient in recipe list.	must not show in the list	entry does not show in the list	
Case 1.4	No specific input, the user must just start the Add a recipe function	The Available Ingredients list must show all the Ingredients that are currently added to database.	The Available Ingredients list shows all the Ingredients that are currently added to database.	**Pass**
Case 1.5	Enter an appropriate Recipe Name, add ingredients to the list with appropriate quantities. Then press the Add to database button. (Enter all inputs such that none of the above exceptions occur)	A dialog box "Successfully Added" must be displayed and the new recipe must reflect in the database.	A dialog box "Successfully Added" is displayed and the new recipe reflects in the database.	**Pass**

11.2 Test Results for test case 2

Test case ID	**Input for the test**	**Expected result**	**Actual result**	**Pass/Fail**
Test case 2.1	Login= Incorrect Password= something	A window "Incorrect credentials" must be shown to the user and access must not be granted.	A window "Incorrect credentials" is shown to the user and access is not granted.	**Pass**
Test case 2.1	Login= Project440 (Correct name)	A window "Incorrect credentials" must be shown to the	A window "Incorrect credentials" is shown to the user	**Pass**

	Password= Incorrect	user and access must not be granted.	and access is not granted.	
Test case 2.1	Login= (blank) Password= (blank)	A window "Incorrect credentials" must be shown to the user and access must not be granted.	A window "Incorrect credentials" is shown to the user and access is not granted.	**Pass**

Test case 2.1	Login= Project440
	
	Password= inventory (correct password)
		Access must be granted to the user to the main interface.	Access is granted to the user to the main interface.	**Pass**

11.3 TEST RESULTS FOR TEST CASE 3

Test case ID	**Input for the test**	**Expected result**	**Actual result**	**Pass/Fail**
Case 3.1	Ingredient Name = 123214	Error message must be displayed.	Error message is not displayed	**Fail**
Case 3.1	Ingredient Name = (*Blank*)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.1	Ingredient Name = Kadai Paneer *(existing ingredient name)*	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**

Case 3.1	Ingredient Name = $^%$^%	Error message must be displayed.	No error displayed	**Fail**
Case 3.1	Ingredient Name = Cucumber *(non existing)*	No error must be displayed that relates to this input	No error displayed that relates to this input	**Pass**
Case 3.1	Ingredient Name= (a really long text, a paragraph)	An error must be displayed for too long name	An error message is displayed.	**Pass**
Case 3.2	Threshold Value = -10	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 3.2	Threshold Value = xyz	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 3.2	Threshold Value = 0	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.2	Threshold Value = 12.4	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.2	Threshold Value =	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.2	Threshold Value = $#^	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**

Case 3.2	Threshold Value = 15	No error must be displayed.	No error is displayed.	**Pass**
Case 3.2	Threshold Value = 10000	Error must be displayed.	An error is displayed and the user is asked to enter the Threshold Value again.	**Pass**
Case 3.3	Current Quantity = -10	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 3.3	Current Quantity = xyz	Error message should be displayed.	Error message is displayed and user is asked to re-enter the value	**Pass**
Case 3.3	Current Quantity = 0	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.3	Current Quantity = 12.4	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.3	Current Quantity =	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.3	Current Quantity = $#^	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 3.3	Current Quantity = 15	No error must be displayed.	No error is displayed.	**Pass**

Case 3.3	Current Quantity = 10000	Error must be displayed.	An error is displayed and the user is asked to enter the Current Quantity again.	**Pass**
Case 3.4	Load the Add Ingredient Form.	The Vendor drop down box must show all the current vendors	The vendor Drop down box show all the current vendors	**Pass**
Case 3.5	Load the Add Ingredient Form	The current ingredient list must show list of all the ingredients that are in the database	The Current ingredient list shows all the ingredients that are in the database.	**Pass**

11.4 TEST RESULTS FOR TEST CASE 4

Test case ID	**Input for the test**	**Expected result**	**Actual result**	**Pass/Fail**
Case 4.1	Vendor Name = 123214	Error message must be displayed.	Error message is not displayed	**Fail**
Case 4.1	Vendor Name = (Blank)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 4.1	Vendor Name = Patel Brothers (existing vendor name)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 4.1	Vendor Name = $^%$^%	Error message must be displayed.	No error displayed	**Fail**

Case 4.1	Vendor Name = Ghareeb Nawaz (non-existing)	No error must be displayed that relates to this input	No error displayed that relates to this input	**Pass**
Case 4.1	Vendor Name= (a really long text - more than 25 characters)	An error must be displayed for too long name	An error message is displayed.	**Pass**
Case 4.2	Vendor Type = 123214	Error message must be displayed.	Error message is not displayed	**Fail**
Case 4.2	Vendor Type = (Blank)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 4.2	Vendor Type = Grain (existing vendor Type)	Error message must not be displayed.	Error message is not displayed.	**Pass**
Case 4.2	Vendor Type = $^%$^%	Error message must be displayed.	No error displayed	**Fail**
Case 4.2	Vendor Type = Ghareeb Nawaz (non-existing)	No error must be displayed that relates to this input	No error displayed that relates to this input	**Pass**
Case 4.2	Vendor Name= (a really long text - more than 25 characters)	An error must be displayed for too long vendor type	An error message is displayed.	**Pass**
Case 4.3	Vendor Details = 123214	Error message must not be displayed.	Error message is not displayed	**Pass**
Case 4.3	Vendor Details = (Blank)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**

Case 4.3	Vendor Details = Patel Brothers, Devon Street, Chicago, IL (Duplicate Details)	Error message must be displayed.	Error message is not displayed.	**Fail**
Case 4.3	Vendor Details = $^%$^%	Error message must be displayed.	No error displayed	**Fail**
Case 4.3	Vendor Details = Ghareeb Nawaz, Halsted and Roosevelt, Chicago, IL (non-existing)	No error must be displayed that relates to this input	No error displayed that relates to this input	**Pass**
Case 4.3	Vendor Details= (a really long text - more than 25 characters)	An error must not be displayed for too long detail	An error message is not displayed.	**Pass**
Case 4.4	Vendor Email = 123214	Error message must be displayed.	Error message is not displayed	**Fail**
Case 4.4	Vendor Email = (Blank)	Error message must be displayed.	Error message is displayed and user is asked to re-enter the value.	**Pass**
Case 4.4	Vendor Email = abc@xyz.com (existing email)	Error message must be displayed.	Error message is not displayed.	**Fail**
Case 4.4	Vendor Email = $^%$^%	Error message must be displayed.	No error message is displayed.	**Fail**
Case 4.4	Vendor Email = pqr@xyz.com (non-existing)	No error must be displayed that relates to this input	No error displayed that relates to this input.	**Pass**

Case 4.4	Vendor Email= really long email address - longer than 25 characters.	An error must be displayed for too long email	An error message is displayed.	**Pass**

11.5 Test results for test case 5

Test case ID	**Input for the test**	**Expected result**	**Actual result**	**Pass/Fail**
Case 5.1	Press the check threshold button on the interface	The Ingredients Below threshold must show the Ingredients that are below threshold level	The Ingredients Below threshold shows the Ingredients that are below threshold level	**Pass**
Case 5.2	Press the Create Order Button	The user should be asked to enter ingredients order quantity for all the ingredients listed in the Ingredients below threshold list.	The user is asked to enter ingredients order quantity for all the ingredients listed in the Ingredients below threshold list.	**Pass**
Case 5.3	Press the Process Order Button	A file with all the order details must be created in the project folder	A file with the details of the order is created in the project folder	**Pass**