Team Six Deliverable Three

Andrew Clifford (19798632) Anzac Morel (66261880) Connor Macdonald (41647584)
George Stephenson (97277741) Hamesh Ravji (43832772) Rchi Lugtu (89933448)
Taran Jennison (67420293)

October 14, 2019

Contents

1 Executive Summary

FoodByte is on a mission to develop a food truck application that provides its users with an easy to use tool to assist in day to day operational tasks as well as inventory management and past order tracking. The most important quality requirements for FoodByte were found by surveying users of similar systems. The most important quality requirements were availability, reliability and useability. Key stakeholders holding a high priotity were the employees using the system and the truck manager as these are the key users of the system.

A major risk determined at the beginning of the project was that the product would not agree with the stakeholders expectations, the solution to this was constant communication. The largest risk that kept reappearing was a bug in the software causing the system to function incorrectly. The solution to this was to implement thorough testing, but this was not completed. Acceptance testing and manual testing has been completed, as well as basic JUnit testing with 70% where there are various getters and setters reducing this percentage.UL

2 Business and System Context

2.1 System Context

This system, FoodByte, is to be used in either a small café or food truck, by the owners and employees to assist in managing the flow of orders and various day to day operations. Employees will be using the system the majority of the time it is in use so it is important the system will enhance their work and not interfere with it. Managers are able to interact with the system to manage stock, menus, menu items as well as view or refund past order transactions. The system can also communicate to the cooks/baristas to provide them with information on each order including a hierarchical tree displaying items within the order and items if any that correspond to their structure.

2.2 Relevant Business Information

FoodByte is worth developing because there is a gap in the software market for an application which is able to digitally manage day to day operations such as creating orders, managing stock as well as have a count of what is currently in the register all in one small package, and is designed for small businesses. FoodByte is able to provide the owners of the business with a tool to enhance and ease their order systems whilst providing a wide range of backend support for stock and item management.

The application will also allow for owners to import existing data files, such as xml files, containing data on products they offer into the system where the system will update all relevant sections in the application. Having the ability to import data allows the business load different sets of menus and items. The owner is also able to export the menu if they wish which will save the contents to an external file. Having the ability to export data allows the owner to keep track of how their business was doing at a particular time.

FoodByte is unique. One of the key selling points comes when adding new food items to the inventory, the high-level inheritance structure allow the user to have multiple variants of the same item. This allows for on the fly switching between variants of items, such as switching a burger to use gluten-free buns. This would greatly benefit some business as each item is independent of the next where customisations are concerned, for example; A customer may order a Cheese-burger but want to add more cheese, they can with the click of a few buttons.

The target market for the application is small food truck and cafe owners. Suitable customers could be a food truck stall at the Sunday market or a café at the University of Canterbury.

3 Stakeholders

3.1 Stakeholder Personas

This section aims to outline all the stakeholders the project has, and their respective concerns and what the team can do to relieve them. This helps the team see the point of view of the potential users and adjust course to best suit them and keep the project on track. Below there is a table which outlines the types of stakeholders, their needs, possible impact, priority, and action required.
Types of stakeholders:

- SP: Sponsors. People who commissioned project to be completed, including the intended people using the application.
- IN: Internal stakeholders. Employees/managers creating the product.
- EX: External stakeholders. People not directly involved but affected by outcome.

ID: SP0
Priority: High
Impact: High
Persona: Front of house employee
Front of house employees will be working directly with customers and the order management part of the application.
Stakeholders concerns:
Ease of use and speed when selecting options or possibly doing slightly customised orders.
Solution:
Create a clear GUI without unnecessary information.

ID: SP1
Priority: High
Impact: Medium
Persona: Truck manager
Manages the employees in the truck. Will likely update menus and stock levels in the system. Possibly ordering more stock when necessary.
Stakeholders concerns:
Ease of use when updating items in the application. Clarity of information shown so that they can make informed decisions based on stock levels, sales, etc. They might also want the ability to export/import recipes/stock for use in multiple food trucks.
Solution:
Create an easy to use GUI for displaying information and adjusting stock levels/recipes/prices.

ID: SP2
Priority: Medium
Impact: Medium
Persona: Truck Owners
Truck owners who are not also truck managers will not necessarily use the software but will rely on their employees to use it.
Stakeholders concerns:
Truck owners will need the software to work reliably for their employees to ensure their business has little downtime.
Solution:
High coverage of automated tests to ensure high quality and stability.

ID: SP3
Priority: Medium
Impact: High
Persona: Stock managers
Stock managers will only be interacting with the management side of the system to view and change stock levels.
Stakeholders concerns:
Ease of use with the management system. Access to financials, and stock. If the system fails the stock management will likely become inaccessible.
Solution:
Create an easy to use GUI for displaying information and adjusting stock levels/recipes/price. High coverage of automated tests to ensure high quality and stability.

ID: SP4
Priority: Medium
Impact: High
Persona: Chef
The chef will be relying on the system to give them orders. Likely won't interact with the GUI directly.
Stakeholders concerns:
Ability to see orders easily whether by notes or screen. If by screen, then an ability to dismiss or mark as complete is necessary.
Solution:
Create an output after each order is finalized that can be sent to the chef and displayed or printed in a given form.

ID: EX0
Priority: Low
Impact: Medium
Persona: Event Managers
Stakeholders concerns:
Event managers want the food truck to ensure everyone gets their order so reliability is their priority.
Solution:
Ensure orders are as easy to manage and consistent.

ID: EX1
Priority: Low
Impact: Low
Persona: Council
Stakeholders concerns:
The council wants the food truck to abide by food standards.
Solution:
Add a feature which allows tracking of food expiry dates.

ID: EX2
Priority: Low
Impact: Low
Persona: Stock suppliers
Stakeholders concerns:
Reliable stock order requests.

ID: EX3
Priority: Low
Impact: Medium
Persona: Accountant
Stakeholders concerns:
Access to organised financials.
Solution:
Make the management side easy to read and simple to export.

ID: IN0
Priority: Medium
Impact: High
Persona: The team
The team working on the project
Stakeholders concerns:
Meeting required standards and creating a product that fills the needs of the clients.
Solution:
Frequent interaction with stakeholders and good project management.

3.2 Foodtruck/Cafe Survey

Reboot Cafe

1. What are the 3 most important features of a POS/FOH program?
   - It just needs to do its job in a reliable manner.

2. What do you like/dislike about your current system?
   - No likes or dislikes, "it works" and that's all that really matters.

3. If you could have a dream system, what features would you want?
   - The system is easy to use (and simple to learn).
   - Eftpos connectivity is almost vital.

4. If you could have a dream system, what features would you not want?
   - No detrimental features ("It just needs to work").

5. How do you run inventory management?
   - System keeps track of what is sold so they know what needs to be restored.

Notes: Looking at their current system they not only had separate tabs for drinks, food and misc, but they also subdivided each page by colouring each type a different colour. (ie. chilled drinks where blue and coffees where brown)

Reboot Cafe

1. What are the 3 most important features of a POS/FOH program?
   - It needs to be fast enough to not keep customers waiting.
   - It needs to be reliable and able to run for a full day with no issues.

2. What do you like/dislike about your current system?
   - An end of day tally receipt can be printed to provide a breakdown of the days sales.

3. If you could have a dream system, what features would you want?
   - Having the ability to modify prices and other such variables on the job would be a very convenient feature.
   - Being able to export each days sales to a USB drive so they can import to a spreadsheet would be amazing.

4. If you could have a dream system, what features would you not want?
   - Not having a daily breakdown would be detrimental.

5. How do you run inventory management?
   - Inventory is managed by eye in terms of how much is left at the end of the day and manually deciding if they need more.

6. What are the things you find difficult to manage in your business?
   - Managing inventory is a constant struggle

Notes: They mentioned having a really good idea of what sells best at different events/locations (ie. they sell effectively no drinks while at university but a lot of dipping sauces, conversely to events where they sell a lot of drinks.) This suggest towards it being potentially easy to tag days by location/event and build profiles on what sells where. It is also worth noting that their current system is a very analog system so many of the features they would want are features that would be expected from a software system.

4 Quality Requirements

Quality requirements describe the expectation of a customer for the system, and how it performs. The quality requirements were determined by going through the processes of the system and discussing if there were quality related expectations at given points. For example when an order is being placed the user/employee should not have to deal with the software crashing. In the two tables below the quality requirements are split into requirements for management and operational sides of the system and has ordered by priority.

The table below shows the ranking of importance of the quality requirements for this sytem application. These quality requirements are what the team considered as the essential for this application. These are availability, reliability, speed, usability, maintanbility, portability, scalability. For each stakeholder, there is a set weight value representing their importance level in the system. Each of them is assigned a metric value representing level of significance for each of the quality requirements for the system application. The result above clearly shows the level of significance for each of the quality requirements of the system.

The table shows that Reliability have the highest overall significance level according to the stakeholders selected. This shows that having a software that's reliable, and non-erroneous plays a big role in the development of this application. Having a non-reliable and error-prone system heavily impacts our application, as customers would not want something that is not properly working.

Operational / Management				
ID	**Description**	**Acceptance Tests**	**Stakeholder**	**Priority**
		Availability		
QR1	Software service needs to be available to the employees at all times when food truck is open / operational. No fatal errors.	System can run for 12 hours without an issue (12 hours because food trucks / small food businesses normally operates for around 10 hours or so). **- Test passed**	IN0, SP0, SP1	High
		Reliability (Robust)		
QR2	Software system should not crash or in general not be erroneous for the duration that the software service is being used.	If customization were made on a food by a customer (e.g no pickles in the burger), once the order has been processed, it should always notify the food staff on that customization. **- Test passed**	IN0, SP0, SP1, SP5, EX0,	High
		Usability / ease of use		
QR3	GUI should be visually clear, and simple to use by end users. The application must be user-friendly	User should not be presented with more than 7 interactions at a time. User should not require any technical knowledge to use the app. **- Test passed**	SP0, SP1, SP4, EX0	High
QR4	User interface should be easy to remember, and user should know how to use it on subsequent visits.	When the owner adds stock items, it should take $<$ 5000ms to do it. User should not be presented with more than 7 interactions at a time. **- Test passed**	SP0, SP1, SP4, EX0	High
		Speed		
QR5	Software functionalities should not take a considerable amount of time to process.	Use case operations should process $<$ 500ms, depending on the type of action the user performs, unless prompted otherwise. E.g When a customer orders food, the order processing time should take $<$ 2000ms. **- Test passed**	SP0, EX0	Medium
QR6	Some functionalities of the software should not take forever to process. However, it doesn't necessarily need to be super fast as well.	When the user checks the sales for the day, the information generated should take $<$ 1000ms to be presented on the screen. **- Test passed**	SP0, EX0	Medium
		Maintainability		
QR7	Low coupling high cohesion. Code should easily be understood, repaired, or enhanced by other developers for future feature additions.	All methods (excluding some GUI methods) should have Java Doc descriptions **- Test passed**	IN0	Low
		Portability		
QR9	This is the ability for our software to be accessed, deployed, and managed regardless of what platform it runs on.	System can be run on different platforms such as lab machines, or home desktop etc. As well as being able to import and export data. **- Test passed**	IN0, SP3	Medium
		Scalability		
QR10	The ability for our software to adapt to sudden changes in customer requirements, Ability for our software to react to changes made in the future due to requirement changes etc.	System can be run on different platforms such as lab machines, or home desktop etc. As well as being able to import and export data. Modules should have low interdependence with each other. Having a low dependence for software modules makes it easier for developers to add more functionalities etc, to the current version of the system **- Test passed**	IN0, SP3	Medium

c

Stakeholder	Weight	Availability	Reliability	Speed	Usability	Maintainable	Portable	Scalable
SP0	0.2	5	3.5	4	3.5	2	1	1
SP1	0.2	5	5	3.5	5	0.75	0.25	0.5
SP3	0.12	1	1	0.5	0.5	4	3	2
SP4	0.12	3	4	1	3.25	0.25	0.25	0.25
SP5	0.09	2	3.25	2	1	0.25	0.25	0.25
EX0	0.12	2.5	2	4.5	2	0.25	0.5	0.25
IN0	0.15	2	3	2	1.5	3	1.5	2
Total Weight	1	20.5	21.75	17.5	16.75	10.5	6.75	6.25

5 Use Cases

This section is a list of all possible use cases for the application. The team will use this to build on to create functional requirements which together help create a design for the application. The table below shows a description of a use case, the actors involved in it, the cause for the use case, the action needed as a result and the expected outcome. Note the following table is not ordered.

Use cases

ID	Description	Actors	Pre-conditions	Main effect	Post-conditions	Category/ package
UC1	Cash register.	Employees. Manager. Owner.	Customer makes and pays for an order.	Send prompt to open cash register.	Cash is stored in the cash register, correct change is given and the register is closed.	Front of house/ Cheeseburger.
UC2	Add item to order.	Employees. Manager. Owner.	Customer orders an item/items.	Add requested item to the current order.	Item(s) added to the current order and total price updated.	Front of house/ Cheeseburger.
UC3	Remove item from order.	Employees. Manager. Owner.	Customer no longer desires a specific item/items in the current order.	Remove undesirable item from the current order.	Item(s) are removed for the current order and total price updated.	Front of house/ Cheeseburger.
UC4	Cancel order.	Employees. Manager. Owner.	Customer no longer wants to order from our truck.	Remove all items from the current order.	All items removed from the current order and total price updated.	Front of house/ Cheeseburger.
UC5	Refund order.	Employees. Manager. Owner.	A customer returns an item that was incorrect and or not up to standard.	Refund the total cost of the returned item.	Complaint is noted and cost of the item returned to the customer.	Front of house/ Cheeseburger.
UC6	Special order.	Employees. Manager. Owner.	Customer requests a menu item be modified.	Ingredients used in creation of item changed for this case only.	Chefs are informed of special order/ ingredient change.	Front of house/ Cheeseburger.
UC7	Create a menu.	Chefs. Owner.	New menu has been created and needs to be added to the system.	Add new menu to the system.	Menu is ready for use.	Management/ Boiled egg.
UC8	Add a menu item.	Chefs. Owner.	Menu needs to be edited to accommodate for new item(s).	Add an item to an existing menu.	Menu is updated with new item(s).	Management/ Boiled egg.
UC9	Remove a menu item.	Chefs. Owner.	Menu needs to edited to remove item(s).	Remove an item from an existing menu.	Menu is updated without removed item(s).	Management/ Boiled egg.

Use cases

ID	Description	Actors	Pre-conditions	Main effect	Post-conditions	Category/ package
UC10	Edit a menu item.	Chefs. Owner.	An aspect of an existing menu item has changed.	Change details of an item on an existing menu.	Menu is updated with new information about the item.	Management/ Boiled egg.
UC11	Add recipes.	Chefs. Owner.	New recipe for a given menu item is created.	Add a new recipe to the system.	Recipe is ready for use.	Management/ Onion soup.
UC12	Remove recipes.	Chefs. Owner.	A specific recipe is no longer required.	Remove existing recipe from system.	Recipe is no longer available for use.	Management/ Onion soup.
UC13	List recipes.	Chefs. Owner.	Recipes needed for creation of menu items.	List the recipe for a given menu item.	Recipe is open and readable so the item can be created.	Management/ Onion soup.
UC14	Add ingredients (stock).	Chefs. Owner.	Order of stock has arrived.	Add new items to stock.	The new level of stock is displayed in the system.	Management/ Onion soup.
UC15	Update stock.	Chefs. Owner.	Stock has been used to create menu items.	Update the stock to account for item usage.	The new level of stock is displayed in the system.	Management/ Onion soup.
UC16	List available stock.	Chefs. Owner.	Chefs need to check how much of each item they can create.	View the current level of stock.	The level of stock and quantity of each menu item that can be created is displayed.	Management/ Onion soup.
UC17	Check sales.	Owner.	Business hours have ended and no more sales will be made.	Check the number of sales made on a given day.	The number of sales on the given day is displayed.	Management/ Gumbo.
UC18	Generate sales report.	Owner.	Owner needs a sales record to show potential investors.	Generate formal report detailing sales,costs and profits.	A report detailing sales and costs, profit margins etc is generated with visual aids.	Management/ Ginger crunch.
UC19	Adjust prices.	Chefs. Owner.	The price of a given item is too high or low.	Adjust the price of a given menu item.	Menus are updated to reflect adjustment.	Management/ Onion soup.
UC20	Save Menus.	Chef. Owner.	Menus changes are needed to be saved.	Menus are stored in the system.	Menus that include changes are stored.	Management/ Onion soup.
UC21	Load Menus.	Employees. Manager.	Menus need to be displayed for customers to view.	Display menus.	Menus are displayed and readable.	Management/ Onion soup.

Use cases

ID	Description	Actors	Pre-conditions	Main effect	Post-conditions	Category/ package
UC22	View Historical Sales.	Owner.	Owner wants to see the sales for a	Display sales made on the given	The sales from the desired day are displayed.	Management/ Ginger crunch.
UC23	Place Order.	Employees. Manager.	Customer has finished ordering.	Commit order to the system.	Stock levels adjusted and preparation of items begins.	Front of house/ Cheeseburger.
UC24	Add Ingredients.	Chefs. Owner.	New ingredients used to create an item.	Add new ingredients to the system.	New ingredients added to the system and ready for use.	Management/ Onion soup.
UC25	Remove ingredients.	Chefs. Owner.	Ingredients no longer used to create an item.	Remove ingredients from the system.	Existing ingredients are no longer in the system.	Management/ Onion soup.
UC26	Add item when stock is (critically) low.	Employees. Manager.	Customer orders an item which has critical low stock level.	Item is not able to be added to order.	Item is only added if the stock is available.	Front of house/ Cheeseburger.
UC27	Check for number of servings.	Employees. Manager.	Employee is unsure of how many servings of an item can be created with given stock level.	Display an approximate number of servings left based on stock level.	Information displayed on the screen about the number of servings that can be made.	Front of house/ onion soup.
UC28	Add item to production queue	Employees. Manager.	An order has been placed by the customer and needs to be prepared.	Order placed at the end of the production queue.	Order is ready to be served.	Front of house/ Cheeseburger.
UC29	Remove item from production queue.	Employees. Manager.	An order has been prepared.	Order is removed from the top of the production queue.	Order is given to the customer and preparation of the next order can begin.	Front of house/ Cheeseburger
UC30	Print customer order receipt.	Employees. Manager.	An order has been processed.	Order summary receipt is printed.	Receipt is given to the customer.	Front of house/ Cheeseburger.
UC31	Edit recipe.	Chefs. Owner.	Change in recipe occured.	Change recipe in system.	New recipe is stored in system.	Management/ Onion soup.
UC32	Delete a menu	Owner. Manager	Existing menu in the system must be removed.	Remove the existing menu in the system.	Menu is removed, without errors.	Management/ Boiled egg.

6 Functional Requirements

6.1 Original Functional Requirements

Functional requirements are implementation level descriptions of functionality needed for use cases. They are important for the team as they are well defined goals for what it should be able to do. Below is a list of functional requirements of the application and a user story to express how the system reacts to the user. They are listed in no particular order.

FR1 - Add Cash To Register (UC1)
User can input number of different cash denominations and correct change is given. This action will occur every time a payment is received and needs to occur instantly, as cash will be added frequently and customers are not willing to wait a long time for their change. The system will always compute the correct change and will always complete this action immediately.

FR2 Calculate change - (UC1)
When cash is received the system will calculate the changed required ie Amount received - Amount required. This action will occur every time a payment is received and needs to occur instantly, as cash will be added frequently and customers are not willing to wait a long time for their change. The system will always compute the correct change and will always complete this action immediately.

FR3 Calculate Order Cost - (UC23)
System calculates and displays the sum of all items in the current order. This action will occur every time there is a change in the current order eg. Chips are added to the current order. This action will need to occur instantly so that the customer can be informed of how much they are required to pay for their order. The system will always compute the current order total and will always complete this action immediately.

FR4 Add Item to customer order - (UC2)
User selects menu item to add to the current order, the system then updates the current order to include the new item and updates total price. This action will occur every time an item is added to the current order eg. Chips are added to the current order. This action will need to occur instantly so that the user can keep track of the current order and as the customer will be unwilling to spend a long time placing their order. The system will always display the correct items in the current order and will always complete this action immediately.

FR5 Remove Item from customer order - (UC3)
User selects a menu item that is in the current order which is to be removed, the system then updates the current order to exclude the selected item and update total price. This action will occur every time an item is removed from the current order eg. Chips are removed from the current order. This action will need to occur instantly so that the user can keep track of the current order and as the customer will be unwilling to spend a long time placing their order. The system will always display the correct items in the current order and will always complete this action immediately.

FR6 Place order - (UC23)
User receives correct payment for the current order and moves order list into production queue. This action will occur every time a payment is received. This action will need to occur instantly so that the preparation of the items in the current order can begin. The system will only move order list into production queue once correct payment is received and will always complete this action immediately. To be implemented into the system before 24/09/19. High, to be done before second deliverable.

FR7 - Cancel Order (UC4)
User selects cancel order, if the current order is not empty then items in current order are removed. This action will occur every time the cancel order button is pressed and will need to occur instantly so that the next order can begin. The system will only clear the current order if there are items to be cleared and will always complete this action immediately.

FR8 - Refund Order (UC5)
User selects a previous order from list and selects refund. The user then optionally types in a description and can select which order items to refund cost for or reorder. The system then either displays change for user to give to the customer or resends order to production queue. This action will occur every time someone returns an order and will need to occur instantly so that change can be given to the customer or the preparation of the items can begin. The system will always give correct change, only perform the task selected and will always complete this action immediately.

FR9 - Special Order Change Ingredients (UC6)
User selects menu item and selects to add or subtract ingredient quantities and then confirms. The system then adds menu item to order list with selected changes. The system also modifies price of item based on changes. (This should be configurable). This action will occur every time a customer has a special request and will need to occur instantly so that the preparation of the items can begin. The system will always display the correct item adjustments and will always complete this action immediately.

FR10 - Special Order add ingredients (UC6)
User selects menu item and selects to add new ingredients. The user then selects an item from ingredient list, The user can then continue with FR8. The system then adds ingredients to custom order. This action will occur every time a customer has a special request and will need to occur instantly so that the preparation of the items can begin. The system will always display the correct item adjustments and will always complete this action immediately.

FR11 - Add ingredients to system (UC24)
User selects to add an ingredient. The user can then name, describe and tick properties of the ingredient. The user then clicks to add to the list of ingredients. The system then saves the ingredient type information into the complete ingredient list for the user. This action will occur every time new ingredients arrive. The system should complete this task within one second so that more ingredients can be added or other tasks can be completed. The system will always update with the correct information and will always complete within one second.

FR12 - Create a menu (UC7)
User selects to create a new menu, the user can name, describe the menu. The user then clicks confirm. The system then save the menu type information into the complete menu list for the user. This action will occur every time the create menu button is pressed. The system should complete the task within one second so that the new menu can be used. The system will always save the new menu with all the correct information and will always complete within one second.

FR13 - View Menu Items (UC21)
The User selects a menu from list. The system then presents the Menu details and a list of menu items from the menu's item list. This action will occur every time a menu item needs to be viewed and should be completed instantly so that the actions the user wishes to perform can be carried out. The system will always display the correct menu items and complete this action instantly.

FR14 - Add Menu Item (UC8)
The user selects a menu from list. The user then selects to add a new menu item. The user then selects an item from a list of recipes. The user can then select a markup/price for the menu item and name and select confirm. The system then adds menu item to menu's item list with details. This action will occur every time a new item needs to be added to an existing menu. The system should complete the task within one second so that the updated menu can be used. The system will always save the updated menu with all the correct information and will always complete within one second.

FR15 - Edit Menu Item (UC10)
The user selects a menu from the list and then selects a menu item from the list. The user can then modify the menu items details and confirm. The system then updates the details of the menu item in the item list. This action will occur every time an existing menu item needs to be adjusted. The system should complete the task within one second so that the updated menu can be used. The system will always save the updated menu item with all the correct information and will always complete within one second.

FR16 - Add Recipe (UC11)
The user selects to add a recipe and is brought to a recipe dialog. The user can then add ingredients and item details. The system then adds recipe to users complete recipe list. This action will occur every time a new recipe for an existing menu item is created. The system should complete the task within one second so that the new recipe can be used. The system will always save the new recipe with all the correct information and will always complete within one second.

FR17 - List Recipes (UC13)
The user selects to list profile recipes. The system then presents the complete recipe list. This action occurs every time a recipe is required. This should be completed instantly so that the desired recipe can be used. The system will always display an accurate list of all recipes in the system and complete instantly.

FR18 - Edit Recipe (UC31)
The user selects lists recipes and then selects a recipe to edit. The system presents the recipe dialog. The user can then modify the recipe details. The system then saves the changes to the complete recipe list. This action will occur every time a recipe needs to be adjusted. The system should complete this task within one second so that the updated recipe can be used. The system will always update the recipe with all the correct information and complete within one second.

FR19 - Add Stock (UC14)
The user selects list ingredients then selects an ingredient and selects add stock. The user can then specify the quantity and the expiry date and confirm. The system then adds stock for the ingredient to the total stock list for the profile. This action occurs every time new stock arrives.The system should complete this task within one second so that an accurate representation of the current level of stock is maintained within the system. The system will always update the inventory with all the correct information and complete within one second.

FR20 - List Stock (UC16)
The user selects list ingredients then selects an ingredient and selects view stock. The system then presents the stock list for the ingredient to the user. This action occurs every time the user needs information about the current level of stock. This action should complete instantly so that the information gained from this action can be used. The system will always display the correct stock items and complete instantly.

FR21 - Update Stock (UC15)
The user selects list ingredients then selects an ingredient and selects view stock. From there the user can change the quantity or completely remove stock. The system then updates values in the complete stock list. This action occurs every time any change to the stock occurs. This action should complete within one second so that an accurate representation of the current level of stock is maintained within the system. The system will always update with all the correct information and complete within one second.

FR22 - Check Sales (UC17, UC22)
The user selects view sales. The system then presents a list of all orders and refunds with profits and losses for the current day. The user can select different days to view. This action occurs every time the user needs to get information about the sales on any given day. This should be completed instantly so that the information gained can then be used. The system will always display the correct sales information and complete instantly.

FR23 - Generate sales report (UC18)
The user selects view sales and then selects analytics. The system then presents a graph of profits per hour over the current day. The user can select a longer period or different metrics to display, such as sales, gross figures or most popular items. This action occurs when the user needs a formal representation of company sales. The system should complete the action within one minute so that the report can then be printed and used. The system will always display accurate sales information and complete within one minute.

FR24 - (UC19)
User completes FR14 adjusting menu item price. This action will occur every time the price of a menu item needs to be adjusted. The system should complete this action instantly so that the menu with the updated prices can be used. The system will always display the updated menu with the correct information and complete instantly.

FR25 - Save Menu (UC20)
User selects export a menu from menu list and then selects save to external file. The user then selects a name and location and confirms. The system then serializes menu data into an xml file as well as any other dependent information about menu items or ingredients. This action occurs every time a menu needs to be displayed physically ie not digitally or used in a different system. The system should complete this action instantly so that the menu can be printed and or used elsewhere. The system will always save an accurate copy of the menu and will always complete instantly.

FR26 - Load Menu (UC21)
User selects import a menu and then selects an xml file location. The system then verifies the xml file is of valid format and then loads relevant information into data tables. The system ignores duplicate information. This action will occur every time that a new menu needs to be uploaded to the system. The system should complete this action within three seconds so that the new menu can be used. The system will always load an accurate copy of the menu and will always complete within three seconds.

FR27 - Edit Variant (UC6)
User selects item in customer order and can change type to similar variant. System then replaces order item with variant item. This action occurs every time the customer would like to change the type of an item eg regular bun to vegan bun. The system should complete this action instantly so that the user can keep track of the current order and as the customer will be unwilling to wait a long time to complete their order. The system will always add the correct items into the current order and will always complete instantly.

6.2 Functional Requirements Post-Mortem

7 Acceptance Testing

This section aims to outline all the acceptance tests that were implemented into the system. This helps the team see if the system meets requirement specifications. Below are all acceptance tests written in the Gherkin test format.

ID: UC1
Use case: Add money to the cash register
Given:
$200 in the cash register
When:
$50 is added
Then:
There is now $250 in the cash register

ID: UC2
Use case: Remove money from the cash register
Given:
$200 in the cash register
When:
$50 is removed
Then:
There is now $150 in the cash register

ID: UC3
Use case: Too much money is removed from the cash register
Given:
$30 in the cash register
When:
$50 is removed
Then:
Error is thrown

ID: UC4
Use case: Add an item to the current order
Given:
One burger in the current order
When:
Chips are added to the current order
Then:
The current order now contains one burger and one chips

ID: UC5
Use case: Remove an item from the current order
Given:
One burger and one chips in the current order
When:
Chips are removed from the current order
Then:
The current order now contains one burger

ID: UC6
Use case: Cancel the current order
Given:
The current order contains one burger and one chips
When:
The order is cancelled
Then:
The current order is now empty

ID: UC7
Use case: Refund the total cost of an order
Given:
The cost of the order that is to be refunded was $15 and there is $50 in the cash register

When:
$15 is removed from the cash register
Then:
$35 in the cash register

ID: UC8
Use case: Customise a menu item
Given:
The current order contains one burger
When:
Tomatoes are removed
Then:
Tomatoes are removed and the chefs order and receipt reflect this

ID: UC9
Use case: Add a new menu to the system
Given:
There are zero menus in the system
When:
New menu is added
Then:
There is one menu in the system

ID: UC10
Use case: Add an item to an existing menu
Given:
A menu contains zero items
When:
Burger is added to the menu
Then:
Menu contains one item, Burger

ID: UC11
Use case: Remove an item from an existing menu
Given:
A menu contains one item, Burger
When:
Burger is removed from the menu
Then:
Menu contains zero items

ID: UC12
Use case: Edit an item
Given:
Burger doesn't contain tomatoes
When:
Tomatoes added to burger
Then:
Burger contains tomatoes

ID: UC13
Use case: Add a recipe to a menu item
Given:
Burger doesn't have a recipe
When:
Recipe is added to burger
Then:
Burger now has an associated recipe

ID: UC14
Use case: View the recipe for a menu item
Given:
Recipe for a burger needs to be viewed

When:
Burger selected AND View recipe selected
Then:
Recipe for burger displayed

ID: UC15
Use case: Add stock items
Given:
5 buns are currently in stock
When:
10 buns are added to stock
Then:
15 buns are currently in stock

ID: UC16
Use case: Update stock when stock is used
Given:
2 chips are currently in stock
When:
Chips are ordered
Then:
1 chips are currently in stock

ID: UC17
Use case: Item with no stock is ordered
Given:
0 chips are currently in stock
When:
Chips are ordered
Then:
Error is thrown

ID: UC18
Use case: View available stock
Given:
Levels of stock need to be viewed
When:
Stock screen is opened
Then:
Available stock is displayed

ID: UC19
Use case: View daily sales
Given:
Daily sales need to be viewed
When:
Past orders is selected
Then:
Past orders are displayed

ID: UC20
Use case: Generate sales report
Given:
Sales report is needed
When:
Generate sales report is selected
Then:
Sales report containing financial information is generated

ID: UC21
Use case: Change price of item
Given:
Price of burger is $10

When:
Price changed to $12
Then:
Price of burger is now $12

ID: UC22
Use case: Save menus to external file
Given:
Menus need to be exported
When:
Save menus to external file is selected
Then:
Menus.csv saved in local directory

ID: UC23
Use case: Load menus from an external file
Given:
Menus need to be imported
When:
Load new menu is selected
Then:
File containing menus is loaded in the correct format

ID: UC24
Use case: View sales from previous days
Given:
Previous days sales need to be viewed
When:
Past orders is selected AND date is changed to 12/10/2019
Then:
Orders made on 12/10/2019 are displayed

ID: UC25
Use case: Confirming an order
Given:
Current order contains one burger and one chips, total cost is $15
When:
Order is selected and stock is available to create order items
Then:
Chef's order and receipt are printed and $15 added to register

8 Deployment Model

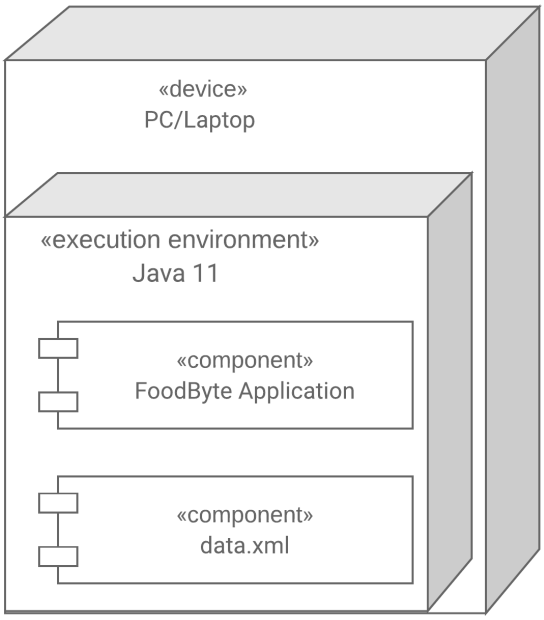

Figure 9: UML Deployment Diagram of the system

The application runs on a single machine, thus making a deployment model non-essential. Data, such as attributes related to stock instances, items, menu items, menus, and orders will be stored in an XML file which will is stored locally on the owner's machine with the added option to import other XML files formed from the same application. The 4-tier system uses a presentation, system, data, and physical layer where each layer interacts only with the layer beneath. This 4-tier system allows an easier transition from using XML files to store data, to using a database management system.

Originally, the plan was to implement a database into the FoodByte system, though being pressed for time meant that this goal was not achieved. With more time, this 4-tier application allows for simple implementation of a database which would be ideal for larger sets of data though at the moment the FoodByte application just stores data in an XML file called 'data.xml'. The process of saving data checks for corruption by storing data in a 'temp.xml' file then renames this file to 'data.xml'.

The system prints customers receipts and chef order slips via the command line interface when orders are finalised. This is because the development team does not yet have the knowledge or hardware to implement physically using the system with a printer to print receipts on command.

9 Data Modelling

FoodByte stores data within XML files, though it was planned to move to storing data within a database to handle larger data sets. The separate data layer of the system allows developers to switch from XML storage access to database storage access a lot simpler. Data required to be stored was constantly changing as we imlemented more features into the project.

Data Modelling is about modelling the data that would need to be stored for the FoodByte application to function to the user's expectation. This included the ability to store ingredients. Ingredients required to create the menu items are created as Stock Instance objects. To start, the application required the below attributes to go with each Stock Instance object.

Ingredients		
Attribute	Type	Notes
Ingredient ID	String	Unique identifier.
Name	String	The name of the Ingredient or item purchased from supplier.
Cost Price	Float	Dollar values stored.
Amount Purchased	Int / Float	User selects unit; KG, G, L, Quantity.
Expiry Date	Date	
Is Gluten-Free	Boolean	
Date Purchased	Date	Refer to date purchase was made or date payment went through.
Payment Type	String	E.g. Cash, Credit, or Eftpos.

Data related to each Item that the food truck might sell would beed to have it's own class. These would be attributes belonging to a class Item, breaking down into a subclass FoodItem to handle dependants and variants, such as Buns with or without Semame Seeds.

Menu Item		
Attribute	Type	Notes
Item ID	String	Unique identifier.
Name	String	Name of item.
Ingredients: Quantity	Float	Composite Attribute Ingredients, quantity one of the attributes it can be broken down into.
Is Gluten-Free	Boolean	
Description	String	Optional for food item.
Recipe	String	Instructions to make item.
Selling Price	Float	

The way FoodByte stored data drastically changed throughout the implementation of the system. Items were created as Item objects. If stock were to be added, a Stock Instance object would be created and would include a reference to the Item of which stock was being added. If that Item were to be added to an order, an OrderItem object would be created which would hold a reference to the Item as well as it's associated attributes. This implementation helped with the addition of implementing custom orders, such as if a user wanted to purchase a gluten-free variant of a burger the employee could easily edit the item to include the gluten-free variant of the bun instead. This also helped with the implementation of the tree-structure to show items in the order. As represented in the customer receipts and chef order slips.

Initially, the idea was to keep track of customer transactions via Statements. A sum of the amounts received from each transaction could be easily derived to show the income each day. Attributes listed below would represent customer transactions.

Statement		
Attribute	Type	Notes
Statement Number	Int	Unique identifier. Reset each day.
Amount Received	Float	Dollar value received stored here.
Payment Type	String	E.g. Cash, Eftpos, or Credit.
Items Purchased	ArrayList¡Item¿	List storing references to each item added to the order.

Later it was decided that this information would still be stored but within each order. This enabled the application to show past order transactions which included the date and the total cost of the order, with the ability to perform a refund or cancel a refund if the button was accidentally clicked. With more time, this could also be modified simply to show the tree structure of the order.

10 Technical Design

From the analysis of quality requirements it was revealed that Reliability(QR), Maintainability(QR) and Flexibility(QR) were all deemed to be of high or medium priority. These factors are influenced by the software having low-coupling and systems. Investing time on technical design and architecture at the start of the project allows for reduced cost of refactoring in the long term; This will be mentioned more under Design Post-Mortem. For this reason it was deemed an important focus on architecture throughout the project. The classes in this section represent the core of the application and aside from Managers, minor services and GUI classes is complete.

10.1 Data Model

Featured below is the UML Class diagrams of model objects to be implemented in Java. They were made from analysis of Data Modelling and modified to meet the needs of Functional Requirements. This model does not represent the full system but just the core classes in the model; As these are the most complicated parts of our system and therefore needing to be diagramed. All attributes can be assumed to have getters or setters unless stated otherwise and the functions mention are examples of additional functionality and may not be reflective of the final system completely. Each of these core classes inherits from UUID Entity that gives the object a unique identifier that is used internally within the system that is discussed further in Section 10.2.

10.1.1 Menu

The menu class is used to act as a collection of menuitems for the business. In practice this could be a breakfast menu, lunch menu, drinks menu etc. Any time of items which the user would like to combine together into one screen.

A menu consists of a name, description and a list of menuitems. Menuitems are unqiue to the menu they belong to and consist of a name, description, colour, price and an Item reference of the food item it sells. The colour is used for gui display. The price is an optional overide to allow the user to have different prices on the same item for different menus. An example of this could be upselling for a festival.

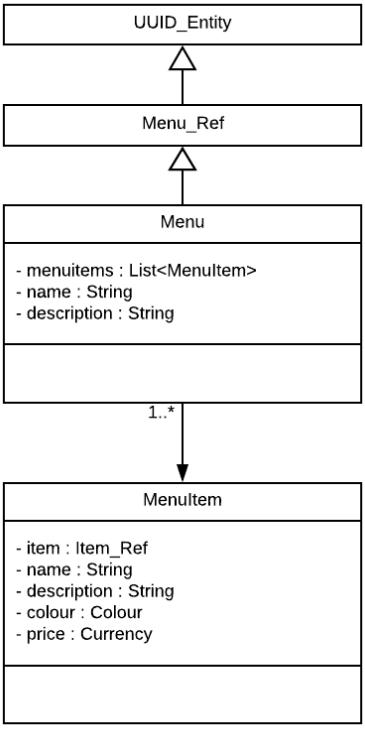

Figure 1: UML Menu Class

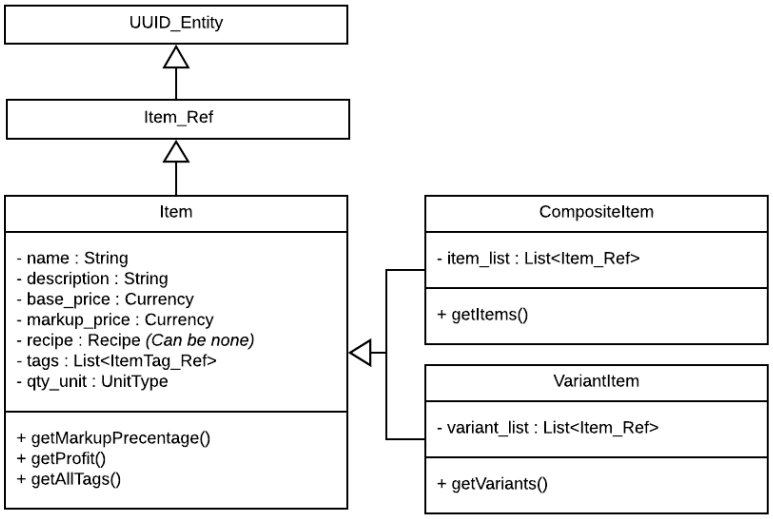

Figure 2: UML Item Class and Subclasses

10.1.2 Item

Item class and its subclasses are used to model the process of ingredients inside a kitchen moving from storage into intermediary products and then finally finished products. It models this by using the Composite class pattern, where CompositeItem is the Composite class and Item is the leaf class. Using this pattern the system can represent heirarcys of ingredients into final products. For instance ground beef into patties and patties into burgers. This gives the user great flexibility and depth in representing production chains in the software and gives the software many metrics to provide functionality or statistics. The design deviates from the original gang of four's pattern by using an additional Composite class VariantItem. The purpose of variant item is to group similar items that can be interchanged in recipes. For instance gluten free bread and regular bread would be grouped in the variant item Bread. This allows for the system to dynamically adjust recipes by knowing which ingredients can be substituted to create different properties in the final product. Such as selecting that the final product be made from only gluten free ingredients. These features are only possible by having a seperate item model from menu items.

The design was picked for robustness as the model makes no assumptions directly about selling food and can be fitted for any goods the user may want to sell. Since ingredients and products are the assumed to be the same in our system the user has complete control to ignore this feature if they wish to not engage with it and only list final products. They sacrifice features like order customisation, variants and percision stock control, but this is still supported to engage with the widest audience of stakeholders.

An item consists of a base price and markup price. This is the price the user can buy the item for and the price the user sells the item for. The base price is used to calculate profit margins for the user and the markup price is used as the default sell price of menu item. This field is useful even for ingredients that will never be on a menu as it is used to calculate add on cost for order customisation discussed further in Section 10.1.5.

The qty unit on item is an enum to designate the quantity unit for stock. This could be kilogram, litre or count.

10.1.3 ItemTag

ItemTag class can be created and applied by the user to designate collections of items under a similar tag, such as gluten free or vegan. The class has two attributes name and dominant flag. The dominant flag is used in descision making in the software to how the tag interacts in composite items. If a tag is dominant then the parent item inheirts the tag if it has one child that owns it. For instance 'Contains peanuts' would be considered a dominant tag because anything made using that tag would also contain peanuts. If a tag is non-dominant then the parent tag only inherits the tag if all its children have the tag. For instance 'gluten-free' is only true if all ingredients are also gluten free.

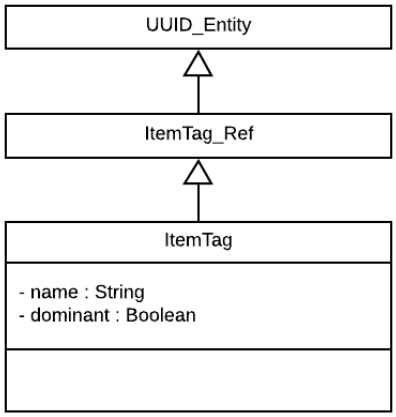

Figure 3: UML ItemTag Class

10.1.4 StockInstance

The StockInstance class represents physical stock of an item that the user has. Using this information the system calculates what products can be made from the ingredients avaliable and warns when running low or close to expiry. The model uses a seperate stock instance class rather than a quantity attribute for item to better emulate the real world. From stakeholder surveys(Section **??**) managing inventory is an important feature and the pattern in which stakeholders bought ingredients was in discrete bulk lots with individual expiry dates. The stock instance class aims to models these bulk lots and give more detail to the system of ingredients freshness and quantities.

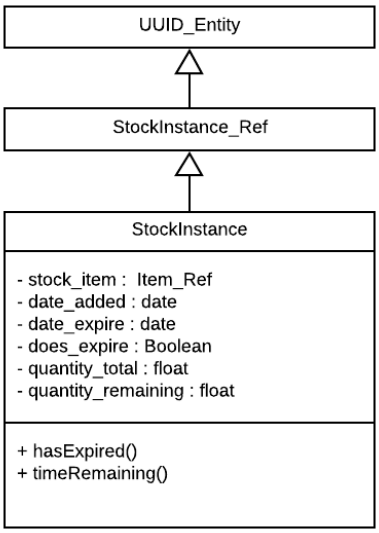

Figure 4: UML StockInstance Class

10.1.5 Order

The Order class models an order made to the system for a single or multiple menu items. The order class was designed with the ability to customise the ingredients in the order by adding, replacing or removing. To accomplish this the Order has to be expressed in the same form as the items and so a heirarcy of OrderItems is used. This is created by deep copying Composite Items into OrderItems. For variant items the variantItem is added to the order and the first variant is added as its child. This is done to perserve the information of the variant item so it can be swaped for other variants. See Appendix **??** for an example of this conversion.

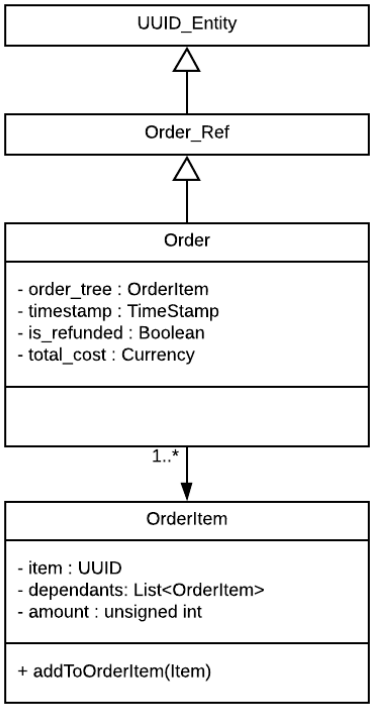

Figure 5: UML Order Class

10.2 Architecture

10.2.1 Model-View-Controller

The system was decided to use the Model View Controller design pattern for its architecture for the advantages of its low coupling of system and gui logic. The core of the data model is described with detail in Section 10.1. To assist in the design of maintainable code the Controller was split into three sublayers of classes. Firstly Managers are identified systems of related code in the program that have shared state. They act as a linker foremost managing the access and use of shared state. The program has very litte state other than data model and so the only manager is OrderManager; That faciliates the order cart and the constuction and customisation of Orders. Second is Logical Services that provide some single responsibility service to the higher levels of the program. An example of this in the software is DataQuery which provides a service of data filtering data model lists. Lastly is GUI Controllers, which are in charge of data sanatation for lower layers and building of dynamic gui elements. Having these sublayers allows for finer control of single responsibility and low coupling. In turn making code more maintainable and easier to test. Below is a visual diagram of these layers.

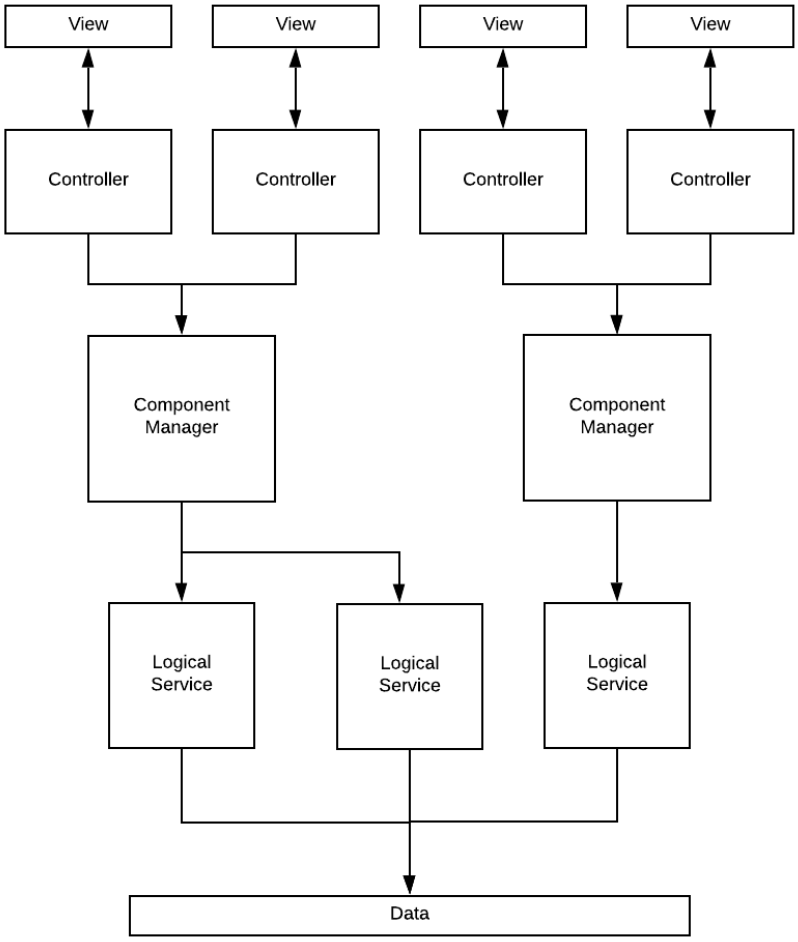

Figure 6: Software Layer Communication Diagram

10.2.2 Data Persistence

Data persistence was important to the project to give the user a Consistent(QR) view of the dataset in the application at all times. To ensure that data model changes were universal in the software the references to data model objects had to be the same through out the program. This was accomplished using a centralised storage system called StorageAccess.

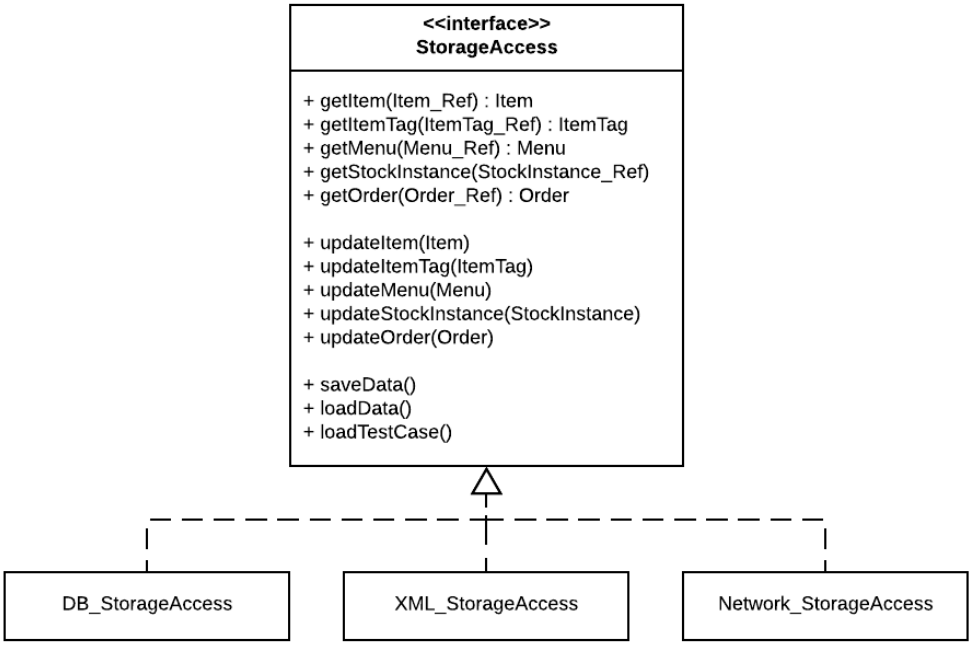

Figure 7: UML StorageAccess Class Diagram

StorageAccess is an interface whos single responsibility is to convert data model objects from physical storage to the class objects. This interface is implemented by classes that provide a type of physical storage, for instance XML StorageAccess which saves and loads xml files into memory. This single interface allows for Flexibility(QR) in storage medium and means the program is open to upgradeability by changing the StorageAccess implementation. Storage access is a singleton since there is only need for one dataset loaded at a time in the system. The reason a single point of access for the dataset of the application is to increase maintainability in the code as it provides a single location to modify how data is stored or how the system reacts to updates. For example StorageAccess can implement the observable pattern and alert gui systems when it needs reload elements because of data updates. This sort of feature is much harder to maintain when you have multiple managers who keep track of this data, as you require multiple locations for the same logic. This creates stronger integration testing as you can better guarantee what data is loaded in the system via test data sets.

StorageAccess has get and update functions for all objects which need to be persistent between runs of the application. These persistent objects are refered to in Section 10.1.

To ensure that every persistent object has a primary key, A required attribute for many storage mediums, all persistent objects inherit from the UUID Entity class which randomly assigns a unique 128 bit number to that object. It also provides methods for comparing equality of two objects. UUID Entity is all that is needed to reference a persistent object, however StorageAccess and most classes use an object reference instead. Object References inherit from UUID Entity and have no additional attributes. They are used to provide an alias to UUID Entity and add context to code for what type of object the reference represents.

10.3 Design Post-Mortem

10.3.1 Investing in Architecture

From reflection the hypothesis that investing in architecture would save developer time and create more maintainable(QR) and flexible(QR) was correct. Most systems are well isolated and only communicate with services that it uses(reduced coupling). There is no god classes in the application and a large majority of classes have strict single responsibility. This has allowed the code to be very easily extendable. An example of this is the feature of auto-saving. By having StorageAccess solely responsible for persistence of data adding auto saving was trivial by calling save data method on any modification to the persistent state, effectively introducing auto save on any committed changes. This is a single example however the time savings are more revealed in the absence of infrastructure road blocks. Not once in the project was there a feature which was incompatible with the design of the system requiring work around solutions, everything had a place. This is a hard claim to proof so we can only refer to the commit logs as proof of this.

10.3.2 Design Reflections

XML was chosen as the default to meet client requirements for import and export of XML data and while useful is not suited for being the primary StorageAccess implementation used. XML is designed to be human readable and editable however this comes at the cost of processing speed and storage. This trade off is not worth it in the current implementation as it is not expected that most users are manually editing the file but modifying data through the use of the application. It therefore makes sense to refactor this system into a more binary format to speed up the program and reduce data size. If given more time this would be a futher step.

Autosaving as it stands works well, due to the issues mentioned above however autosaving will cause stutters as the application grows due to the requirements of JAXB(XML Library) to save the whole file on change. There are several mitagating factors that could be implemented to accomadate for this while still using XMl; These are mentioned in XML Storage Class header in the code base.

11 GUI Prototypes

To start the development of our GUI prototypes, we drafted some GUI wireframes for the main order/service screen and the main inventory screen, as well as a window for adding a new stock item and a window for adding a new menu item.

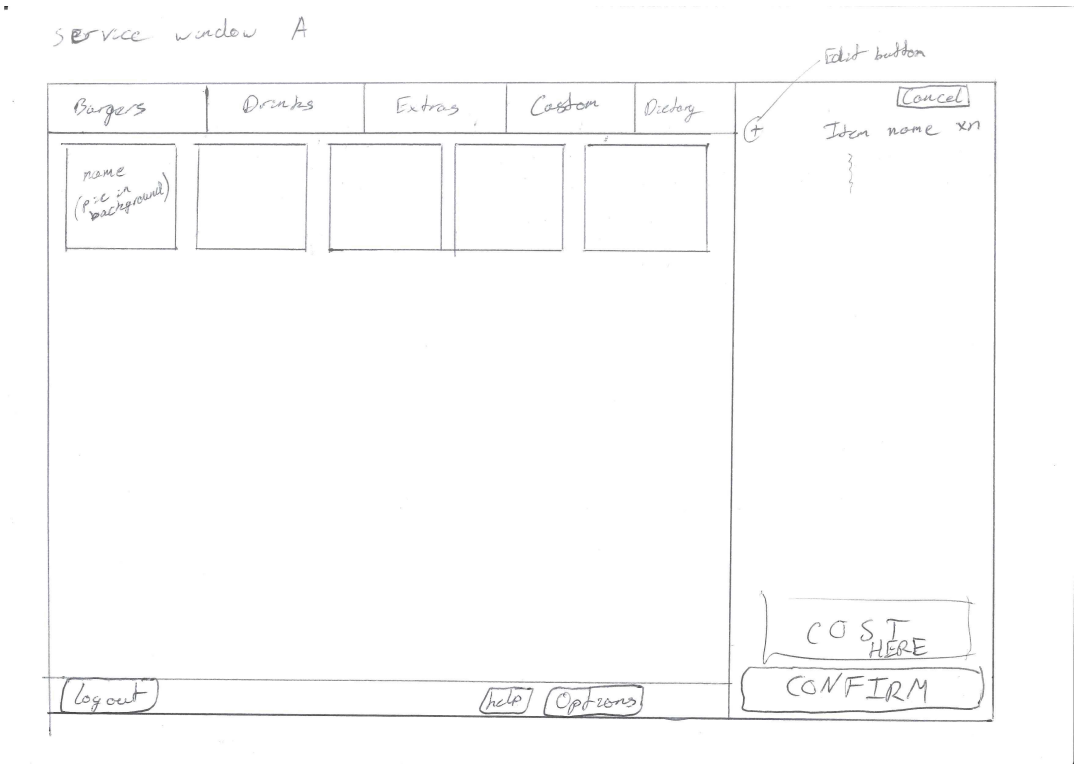

Figure 8: Initial wireframe design of the order screen

Figure 8 shows a first draft of the order/service window. The main idea behind this design was to have large buttons for each item and a tabbed menu selector along the top for selecting categories. It also includes the idea of having an edit button for each item in the order. The other initial warframe concepts can be found in appendix A.

After doing some initial wireframes it was decided that going out and surveying some end users to find out what is important in a POS system would allow us to ensure that further development worked towards bettering the design. To such ends we talked to the Reboot cafe and the doughnut food truck on campus. The full questions and answers can be found in the apendix under user serveys as well as pictures of what each vendor currently has for their point of sales system.

After the serveys there were a few key points that were identified towaords moving forward. The first point was that as long as the system does not get in the way of their work it is a positive addition. The second key point was that it needs to be simple to use and that over complicating things would be a potential pit fall. It was also worth noting that although the Reboot cafe had semi automatic stock managment that gave them a end of day tally for use in restocking, the doughnut truck did not and all of their stock management was by eye. Finally we noticed whilst talking to the staff at the Reboot cafe that their system allowed them to colour code buttons by having their cold drinks using blue buttons and their coffees using brown buttons.

After this feedback from some end users more in depth GUI prototypes were developed using the online tool moqups. During this time we also took inspiration from the Reboot cafe and decided to colour code each of the item buttons to help with usability.

In this GUI prototype (see figure 9) several of the features, quality requirements and their use cases have been taken into consideration. Each item has a large button that is colour coded based on what it is (ie. cheese based/featuring items are yellow). This helps with the useability of the GUI (QR3) as well as directly implementing the ability to add an item to the current order (FR4). There is a clear and large cancel button to allow for the canceling of orders (FR6). The running total cost of the order is clearly displayed above the order confirmation button (FR3). The order confirmation button opens a popup for the order confirmation where, if payment is via cash, the change can be calculated (FR2). Each item in the current order has an edit button to allow for items to be edited (FR8, FR9). It can also be noted that there are selectable tabs for different categories as well as options to search and/or filter items. The date and time is displayed in the bottom left as this is a convenient feature to have during service. The next/previous page buttons are to allow for an overflow of items as page switching can be more reliable than scrolling (QR2). Finally there is an “Options” button that opens a popup with access to the management side of the application as well as the ability to modify aspects of the service side (ie. prices). The options popup also has a button to bring up an order history to facilitate providing refunds (FR7). The prototype sub windows and popups related to the order screen can

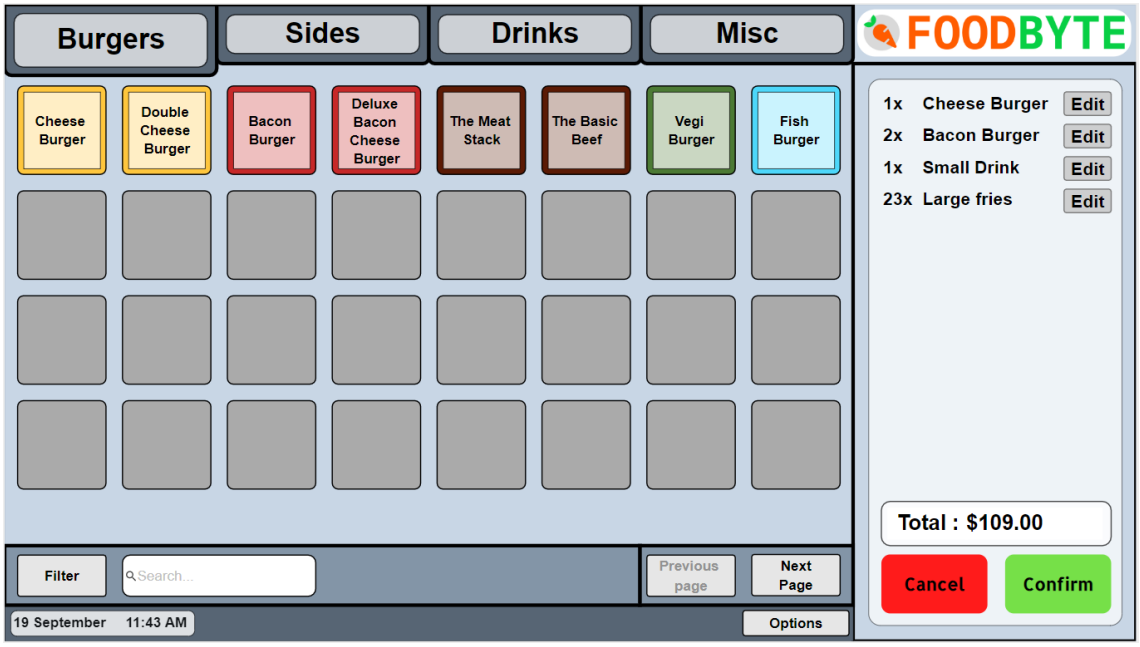

Figure 9: Refined mock up of the order screen

be found in the appendices.

Figure 10) shows a prototype design for the Inventory management system that features tools for stock management and product management. On the left there are options for filtering and searching items which will appear in the middle (FR16, FR19). Items can be added with the "Add New Item" button in the middle (FR20, FR18, FR10). Each item will have the current quantity displayed as well as the cost of the item and nearest expiry date, this will be alongside buttons to adjust stock level and edit item (FR20, FR14, FR24). On the right is the edit screen for editing a selected item, this could also be used for creating new items (FR13, FR10). At the top there are buttons for file which would allow for import/export (FR25, FR26). There is also a button to switch back to operational view and a button to switch to an analytics view (FR21, FR23).

12 GUI Devolopment and deployment

Due to the detailed planning and devolopment of the GUI elements during the first deliverable the creation of the main operations GUI elements were able to be completed rather swiftly. However there was an initial lag caused by the lack of familiarity with JavaFX and scene builder. After a few hours of familiarization, a FXML closely resembling the protoyped design was created for the main order screen.

Most of the differences between the first implimentation and the protoype (11 and 10 respectivly) can be attributed to the differences between the programs used to create them as well as constraints created when implimenting elements with relative alignments or elements that exist within other elements.

The first implimentation of the GUI was effectivly a simple skin of the prototype with buttons that printed their name but no further functionallity. This allowed for a devolopment of basic skills and expereince as well as a platform for further devolopment moving forward. The experience gained from this base was then taken forward and used in the creation of the other UI elements for the operations side whilst refining the main order screen and adding the functionality. The creation of these scenes and their functionallity was significantly easier after getting over the initial learning curve of JavaFX and SceneBuilder.

During this time there was a converstion surrounding the implimentation of the buttons for adding items to orders. Initially the buttons were designed and implimented to all be generated as blank, disabled buttons and then be enabled and filled when needed, however we decided to switch to an approach using buttons that were dynamically created and loaded when the main order screen was loaded in an effort to reduce cluttering of the main order window (QRXX?).

Through out the devolopment of the operations side of the GUI small changes and adjustments were made to the design in an effort to clean and improve elements as they were noticed. An example of this was removing the black outline sorounding each item button. This change gave the main order screen a cleaner appearance, which was part of the driving force for the operational GUI.

The devolopment of the management side GUI was unfortuantly neglected during the planning stages so its devolopment and implimentation became plain and focused on the function when compared to the operations UIs. This caused the initial management screen prototype (see 10) to be paritally overhauled to a more tabular based aproach that focused on presenting information and implimenting functional elements in an efficent and simplified manner.

It is also worth noting that significant efforts towards planning and implimenting the system architecture proved

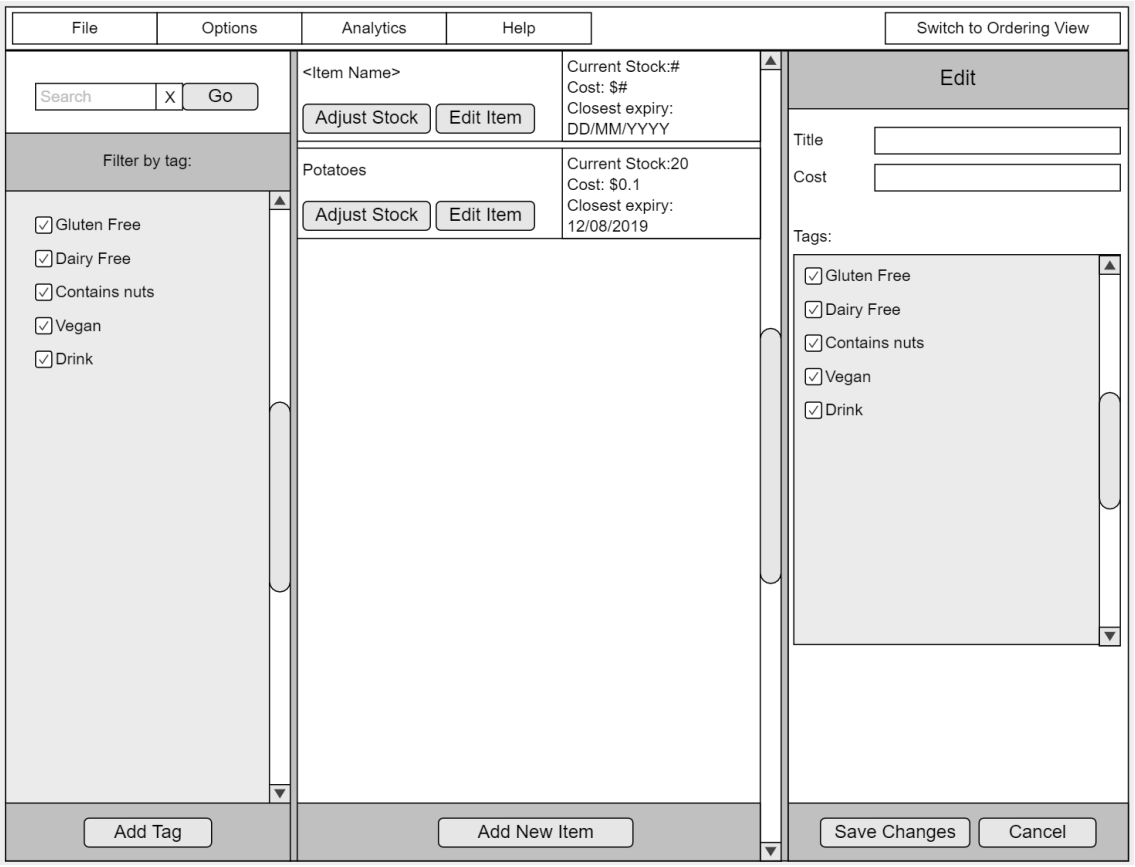

Figure 10: Refined mock up of the inventory management screen

extremely valuable during the implimentation of the functional elements and methods of the GUIs. This allowed for rapid implimentation of features even when under time pressures close to the devlierable deadlines.

Retrospectively the devolopment and implimentation of the GUIs should be considered with a split between the operations and the management sides. This both reflects how the system functions as well as how the GUIs were implimented.

The operations side was the major focus during both the planning and the implimentation stages. Because of this there was a very clear plan and style that was carried through strongly into the implimentation stages. This resulted in a final implimentation (12) that is very similar to the initial design with only a few minor aesthetics changes. The only major change betweeen the prototype and the final implimentation is the change from having pages with next/previous buttons to hold any overflow items for a given menu in the prototype, to a scrolling pane in the final implimentation. This change was made, inspite of our earlier choice to the counter, because it allowed for easier scaling of the interface when the window size was changed whilst not making any major comprimizes towards usability. All other operations GUI scenes can be found in the apendix.

The management side of the GUI was unfortunately neglected during the planning stage and this was also carried through to the implimentation stage. During implimentation the main focus was on creating function and although there was consideration and care towards usability there was little focus towards aesthetics. This has resulted in the manegement side not having the same visual standard as the operations side which is regrettable. The final manegment side UIs (example in 13) ened up implimenting a stripped down tabular focused design. Although the management side is visually lacking when compared to the operations side it was agreed by the devolopment team that, as the operations side will be the main focus of use during deployment, it was more important to have that to a high standard.

When retrospecting the entire aplications UIs as a whole it can be evidanced from the plans and the end result that heavier focus on the operations side has resulted in a application with a very clean and user friendly front end whilst and unfortunately lacklusterr, in comparison, manegment side. If this project were to be further devoloped there is clear room to bring the manegment side up to the same stylazed standard as the operations side and the style used in the operations side should be able to be ported into the manegment side without need for a complete redesign. It is also worth mentioning that there are areas in the options popup in the operations side that would allow for easy insertion of expanded features. On the managment side this could also be acomplished by either adding more tabs to the tab pane or by pulling the tab pane down and useing the row of space above that to impliment more expansions.

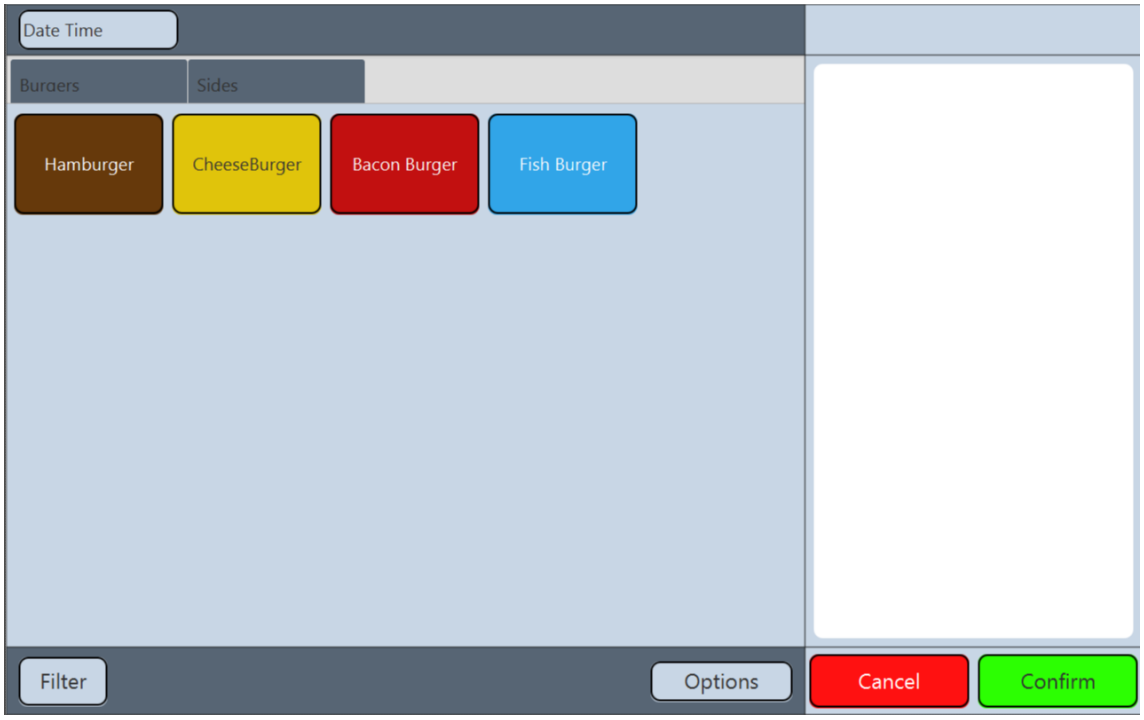

Figure 11: Initial implimentation of the prototyped design

13 Risks

12 Risks

The risk assessment module analyzes a number of different risks that both the team as well as the operator (user of the software) must be aware of during development and use of the software. Each risk is analyzed by multiplying its likelihood of occurring by the impact of the consequences on the group/user. This allows (low-likelihood, high impact) risks to be compared to (high-likelihood, low impact) risks. Most importantly, the last column of the table indicates how the risk can be avoided altogether, so this table should be referenced regularly.

12.1 Team Risks

ID	Description	Likelihood(%)	Impact (1 - 10)	Exposure L * I
R-01	Team Conflict	10%	1	0.1
R-02	Unfamiliar Development Tools	20%	2	0.4
R-03	Unfamiliar APIs/ libraries, various programming skill levels	90%	5	4.5
R-04	Miscommunication with lecturers	80%	5	4
R-05	Loss of data, problem with import of data	20%	6	1.2
R-06	Product does not agree with stakeholders expectations	90%	10	9
R-07	Code written by individual team members not readable by others	70%	4	2.8
R-08	GitLab, Google Drive, etc. become unavailable	2%	7	0.14
R-09	No Internet	5%	6	0.3
R-11	Bug in software e.g. bug says something gluten free when actually it isn't	90%	10	9

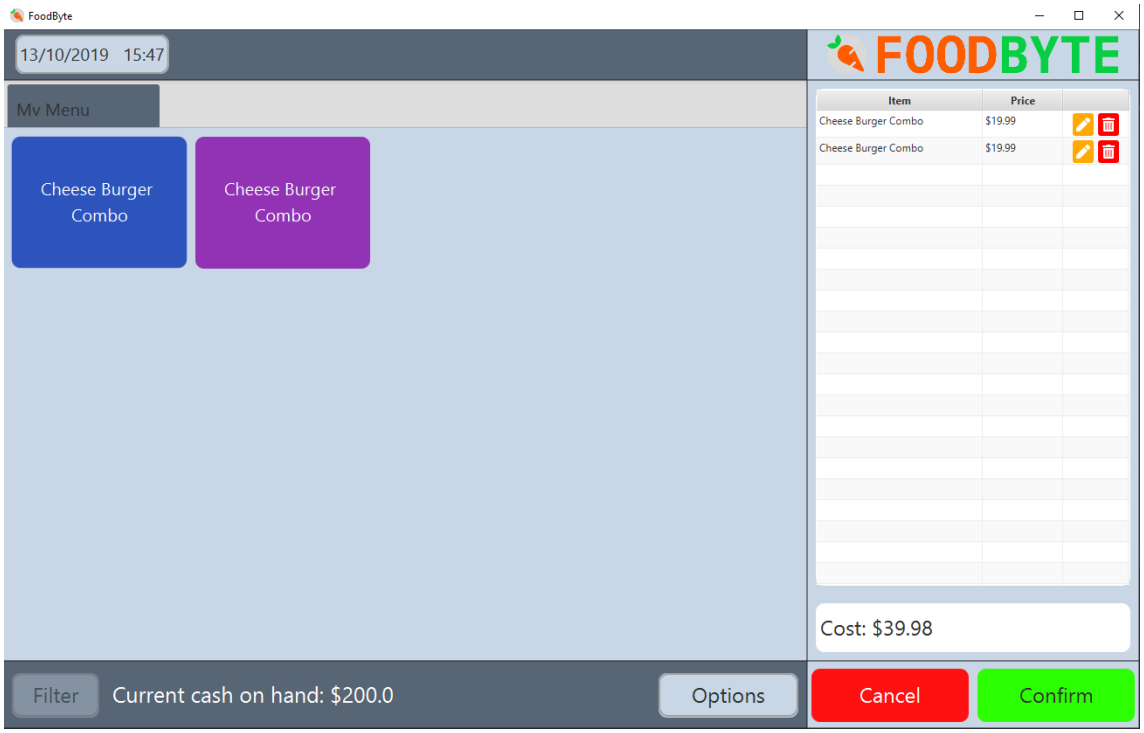

Figure 12: Final implimentation of the main order screen

ID	Description	Consequences	Justification of likelihood percentages	Prevention
R-01	Team Conflict	Reduced Productivity	Team has rules in place to avoid conflict.	Don't be mean, rude, etc
R-02	Unfamiliar Development Tools	Reduced Productivity as time spent learning how to use dev tools	Members of the team have all been learning how to use the same development tools.	Use Dev Tools that majority are familiar with / easy to learn
R-03	Unfamiliar APIs/ libraries, various programming skill levels	Some members limited to certain tasks, may mean some members have to do more work than others	This is the team's first project of this scale. Therefore there are many different API's and libraries that members of the team are not familiar with.	Discuss what libraries may be used
R-04	Miscommunication with lecturers	Doing tasks incorrectly and will have to redo or get a bad mark	The team has very little contact time with the lecture team. Therefore it is easy for a team member to hear and implement something different to what the lecturer was trying to say.	Make sure all team members attend lectures and labs
R-05	Loss of data, problem with import of data	Have to re-complete work / manually import	All members of the team are familiar with Git, which will be used for version control.	Make sure dev tools are compatible with each other
R-06	Product does not agree with stakeholders expectations	Software won't sell (fake world) / bad mark from lecturers (real world)	It is very unlikely that the system will be able to be built to the stakeholders exact specifications and therefore constant communication between the	Make sure we have a balance of focussing on what the lecturers want compared to our stakeholders

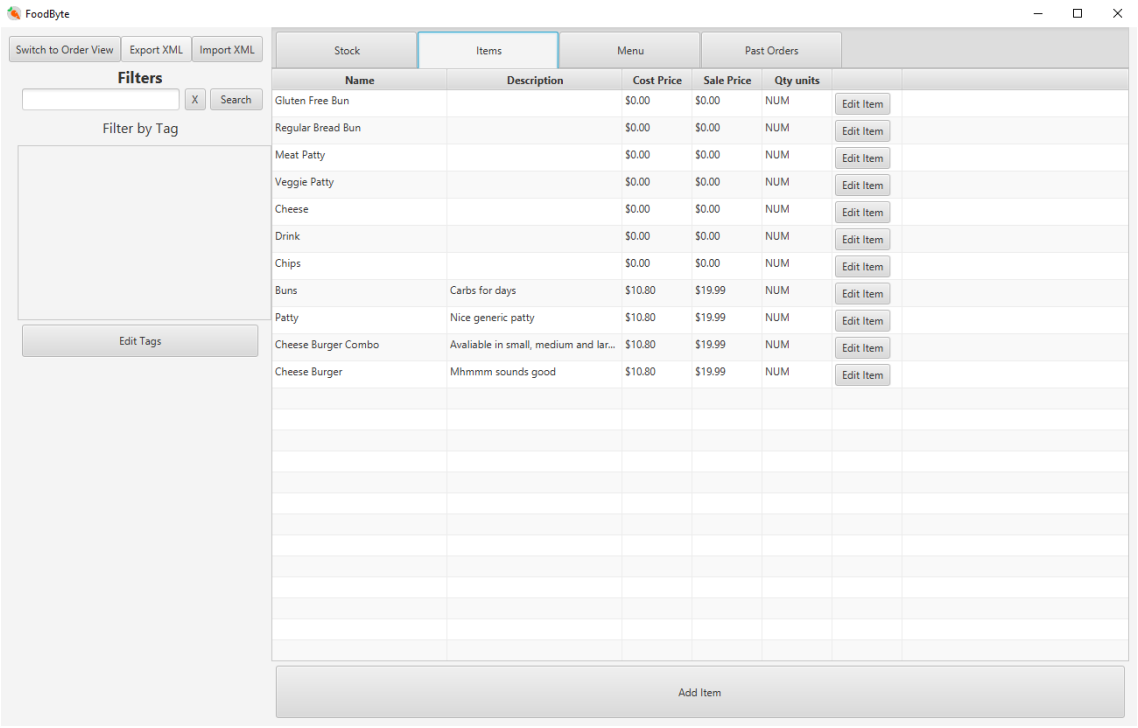

Figure 13: Final implimentation of the item management screen

12.2 User Risks

ID	Description	Likelihood(%)	Impact 1 - 10	Exposure L * I
R-10	Human error (misuse of software)	80%	9	7.2
R-12	Program freezes while processing customer's orders	20%	10	2
R-13	Screen showing the cooks what orders to make is inconsistent with actual order	20%	10	2

ID	Description	Consequences	Justification of likelihood percentages	Prevention
R-10	Human error (misuse of software)	People could get sick	It is very likely that a user makes a mistake as mistakes happen frequently no matter the task.	Popups (i.e. are you sure this is a gluten free item), allows the user to mend their mistake before it becomes an issue.
R-12	Program freezes while processing customer's orders	Angry customers lines get long, lose order	There is a low chance that the system will have a bug that will crash the system once it has been deployed.	Make sure program is stable, allow the user to perform a quick hard reset
R-13	Screen showing the cooks what orders to make is inconsistent with actual order	Angry customers	There is a low chance that the system will have a bug with such an integral part of the system once it has been deployed.	Effective integration testing

12.3 Risks Discussion

Based on the feedback from the 1st deliverable and the 2nd deliverable it was clear that the risks section was not as extensive as it should have been, and some of the values were not correct e.g. We had the likelihood of team members being unfamiliar with libraries at 30% when really it should have been at 90%. In hindsight, we should have had more than one person deciding on the values for the risk assessment module as it resulted in biased and less thought

through values. However time constraints near the end of the deliverable didnt allow for this. Time management was something that definitely held our grade back in the 1st and 2nd deliverable and this is a clear example of that.

For deliverable 2 we changed some of the likelihood values as you noticed and added a 'Justification of liklihood percentages' column too. Below are some examples of some of the risks we actually encountered and how they affected the development of the project.

R-03 - Unfamiliar libraries: This became clear to us very early on in deliverable 2 when using libraries such as JavaFX. Not many members of the group were familiar with it to begin with, even after having completed the JavaFX Lab. This hindered development in some areas where basic GUI functionality actually took a lot longer than expected to get up and running without any bugs.

R-02 - Unfamiliar development tools: Most members of the team had only used Eclipse from SENG201 for Java projects. Switching over from Eclipse to IntelliJ was hard for some members of the team as the Project and Module SDK settings were playing up, however once we got it working there were no more problems and we concluded that IntelliJ is a lot better than Eclipse. SceneBuilder was also very new for most people, Taran did a lot of the design work for our GUI, so he had to learn how to use it by himself but he got the hang of it pretty quickly and produced an appealing GUI. We used Google Drive to store all of our design documentation as everyone was familiar with it, however we eventually had to switch over to LaTeX which was a new development tool for all of us. As this switch happened towards the end of deliverable 3, in hindsight we should have used LaTeX from the beginning as it is better than Google Drive.

R-08 GitLab, Google Drive, etc. become unavailable: We had the likelihood of this risk at 2%, as it seemed so unlikely as the development tools we were using such as GitLab and Google Drive are run by large companies. We were proven wrong when Google Drive crashed on us. Our design doc was 60 pages and when we had seven people trying to edit it at once, it crashed. Hence we switched our design documentation over to LaTeX which was much more friendly and has much better formatting tools. Connor's laptop also crashed two days before deliverable 2 was due. This was unfortunate, however we mitigated its consequences by making sure we always met where there was a lab computer available for Connor to work on. Hamesh also had a spare laptop that he kindly lended to Connor when we had to meet where there were no lab machines.

14 Testing Protocol

1	RowCol1	Row1Col2
2	Row2Col1	Row2Col2
3	Row3Col1	Row3Col2

Table 1: First Table

15 Project Timeline

Deliverable 1:
The first week of the deliverable was spent getting to know members of the team, establishing a team policy and setting up tools that would be used throughout the project (design document, trello board etc). After this initial phase had been completed a meeting was held to create and distribute tasks. This was done by creating a trello card for each task that needed to be complete. Then each group member assigned them selves to a card or cards and they were to complete the associated task before the next meeting. This same method was used for the entire first deliverable and with continual updating of tasks each week meant that the team was able to complete all tasks required without a large push close to the deadline. One issue with this method is that with team meetings being used primarily for task distribution all tasks were completed by individuals. The problem with this is that most if not all of the tasks required collaboration and therefore the quality of our deliverable suffered as a result.

Deliverable 2:
No work was completed during the first week of the deliverable as other commitments were priority at this time. After this a meeting was had to discuss the second deliverable and again create and allocate tasks. This was heading into the mid semester breaks which meant that meeting up and communicating as a whole group was going to be more difficult as some people would not be in Christchurch during the break. To combat this those still in Christchurch had meetings as per usual while a summary was created and send to the other members of the group and they were advised to check trello and assign themselves tasks. No main functionality of the system was implemented over the break only base classes. Once break was over it was time to tackle some of the larger classes and main functionality of the system. After a week it was found that our current system for completing tasks was not going to work any more as the tasks were larger and a lot more communication was required. Therefore we adapted by adding more meetings where we would all come together to work on the project. These peer programming sessions proved to be successful as we again were able to complete all the tasks required without a large push close to the deadline. One thing that we learnt from this deliverable is that even though the project code was well underway the design document should not be left behind. During the second deliverable the design document was just an after though and this wasn't good enough as it should be an active document that accurately reflects our system and we would have to make up for this in during the third deliverable.

Deliverable 3:
No work was completed during the first week as other commitments were priority at this time. After thus a meeting was held to discuss what tasks to prioritize. This was important as we needed to make sure that it was clear what tasks were going to maximise our number of marks while being achievable in the time remaining. The tasks that were decided upon were made into trello cards and were to be completed within the week. During the third week the final trello cards completed and final tweaks and bug fixes on the project completed. Overhaul of the design document begun, moving from google docs to latex as well as updating each section from the design document. This was thought to be necessary as our design document was severely lacking in quality and google docs was not stable enough to allow all members of our team to work on it at the same time. A lot of effort was put in during the final week which was to be expected. During week two, although we were able to complete a lot of work towards the project it was not enough. We did not follow the deadline we set for ourselves strictly enough and this caused us to have to put in a lot more effort in the final week then we had initially planned.

Overall:
Overall the team worked well together and we were able to complete the project to a good standard. Looking back there is a lot to be improved on in terms of planning. One things that was discussed in the lab times is getting the most out of the tools we are using. It was not until the third deliverable that we were adding anything more than just a name to a trello card. Adding deadlines and check boxes in earlier deliverables would have been beneficial as it would help to get tasks done correctly the first time and to help solidify the deadlines that we set for ourselves. Also taking an entire week of was not a good idea especially considering that we did it multiple times. This time should have been used to at least plan ahead and complete smaller tasks. Finally the peer programming sessions that we implemented during the second deliverable should have been something that we did from the very start as they increased the quality of our work as well as keeping everyone on the same page.

16 Document Changelog

Appendices

A Class Diagram