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 13, 2019

Contents

1 Executive Summary

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

Operational / Management		
ID	**Description**	**Acceptance Tests**
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).
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.
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.
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.
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.
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.
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 JavaDoc descriptions
Flexibility		
QR8	This is the ability on how easy our software can adapt to changes made in the future due to requirement changes etc.	When creating a newer and updated version for the software. The 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
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.
Scalability		
QR10	The ability for our software to adapt to sudden changes in customer requirements.	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.

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.

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 / Cheese-burger.
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 / Cheese-burger.
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 / Cheese-burger.
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 / Cheese-burger.
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 / Cheese-burger.
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 / Cheese-burger.
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.
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.

Use cases

ID	Description	Actors	Pre-conditions	Main effect	Post-conditions	Category/ package
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.
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.

Use cases

ID	Description	Actors	Pre-conditions	Main effect	Post-conditions	Category/ package
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

UC1	Add money to the cash register	$200 in the cash register
UC2	Remove money from the cash register	$200 in the cash register
UC3	Too much money is removed from the cash register	$30 in the cash register
UC4	Add an item to the current order	One burger in the current order
UC5	Remove an item from the current order	One burger and one chips in the current order
UC6	Cancel the current order	The current order contains one burger and one chips
UC7	Refund the total cost of an order	The cost of the order that is to be refunded was $15 and there
UC8	Customise a menu item	The current order contains one burger
UC9	Add a new menu to the system	There are zero menus in the system
UC10	Add an item to an existing menu	A menu contains zero items
UC11	Remove an item from an existing menu	A menu contains one item, Burger
UC12	Edit an item	Burger doesn't contain tomatoes
UC13	Add a recipe to a menu item	Burger doesn't have a recipe
UC14	View the recipe for a menu item	Recipe for a burger needs to be viewed
UC15	Add stock items	5 buns are currently in stock
UC16	Update stock when stock is used	2 chips are currently in stock
UC17	Item with no stock is ordered	0 chips are currently in stock
UC18	View available stock	Levels of stock need to be viewed
UC19	View daily sales	Daily sales need to be viewed
UC20	Generate sales report	Sales report is needed
UC21	Change price of item	Price of burger is $10
UC22	Save menus to external file	Menus need to be exported
UC23	Load menus from an external file	Menus need to be imported
UC24	View sales from previous days	Previous days sales need to be viewed
UC25	Confirming an order	Current order contains one burger and one chips, total cost is $

7 Acceptance Testing

8 Deployment Model

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.

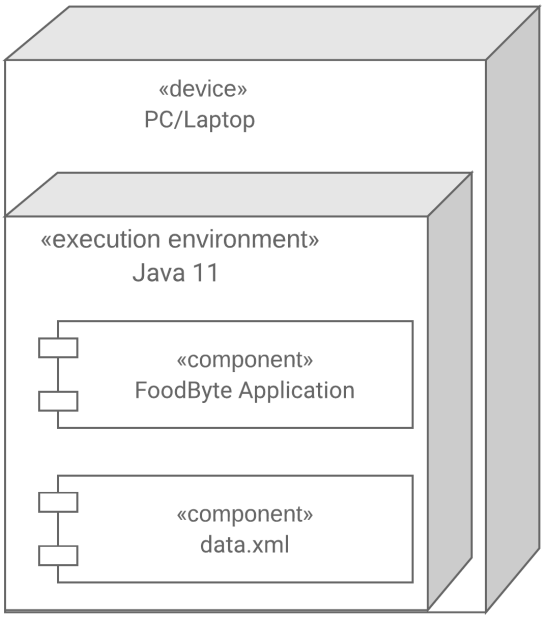

Figure 9: UML Deployment Diagram of the system

9 Data Modelling

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 robust 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.

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. 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

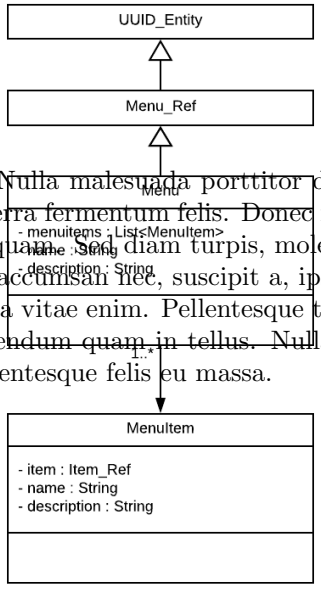

Figure 1: This is a figure caption.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

10.1.2 Item

10.1.3 ItemTag

10.1.4 StockInstance

10.1.5 Order

10.2 Architecture

10.3 Design Post-Mortem

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.

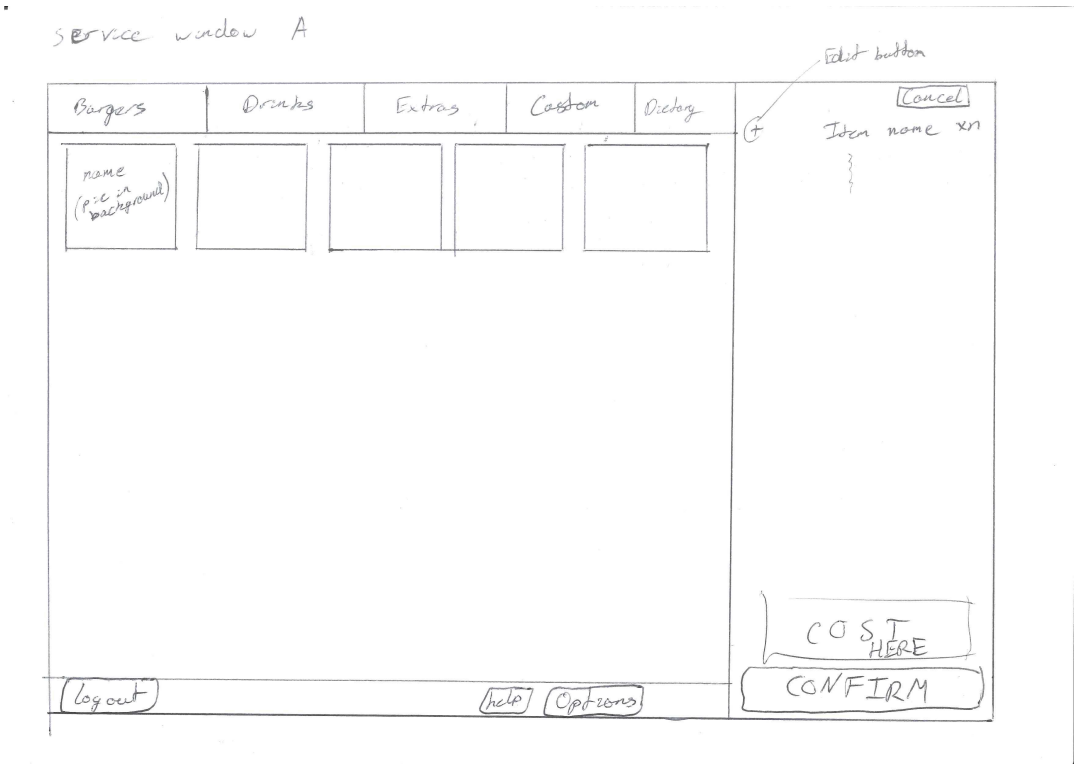

Figure 2: Initial wireframe design of the order screen

Figure 2 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 3) 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

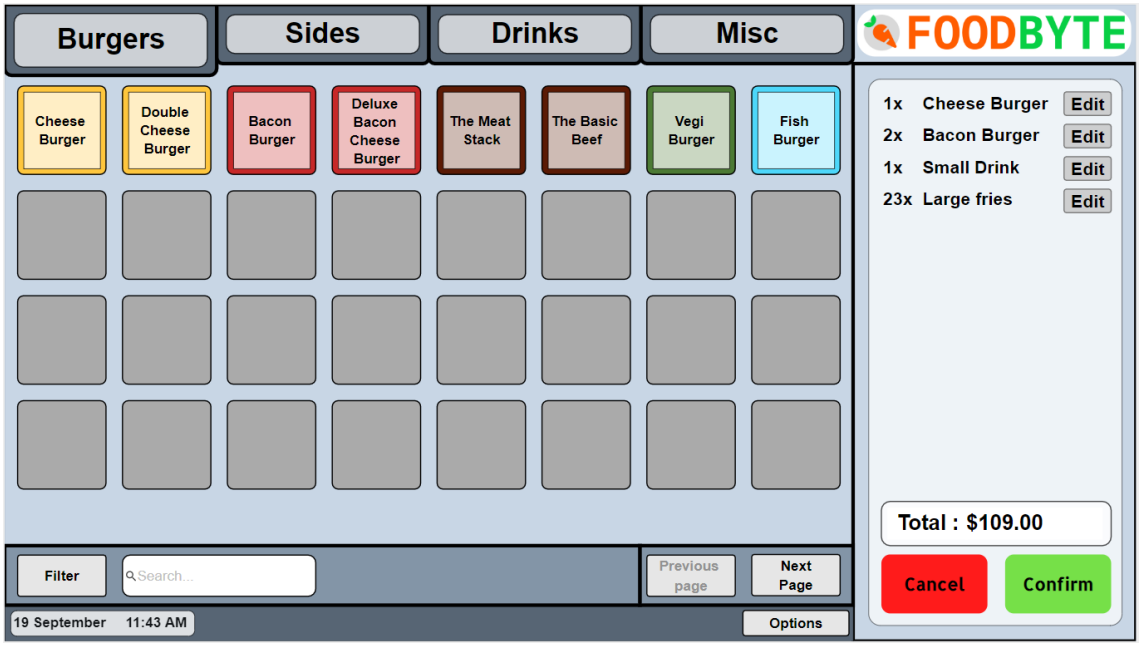

Figure 3: Refined mock up of the order screen

be found in the appendices.

Figure 4) 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 (5 and 4 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 were unfortuantly neglected during the planning stages so their devolopment was left more plain and focused on the function. This caused the initial management screen prototype (see 4) 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 extremely valuable during the implimentation of the functional elements and methods of the GUIs. This allowed for

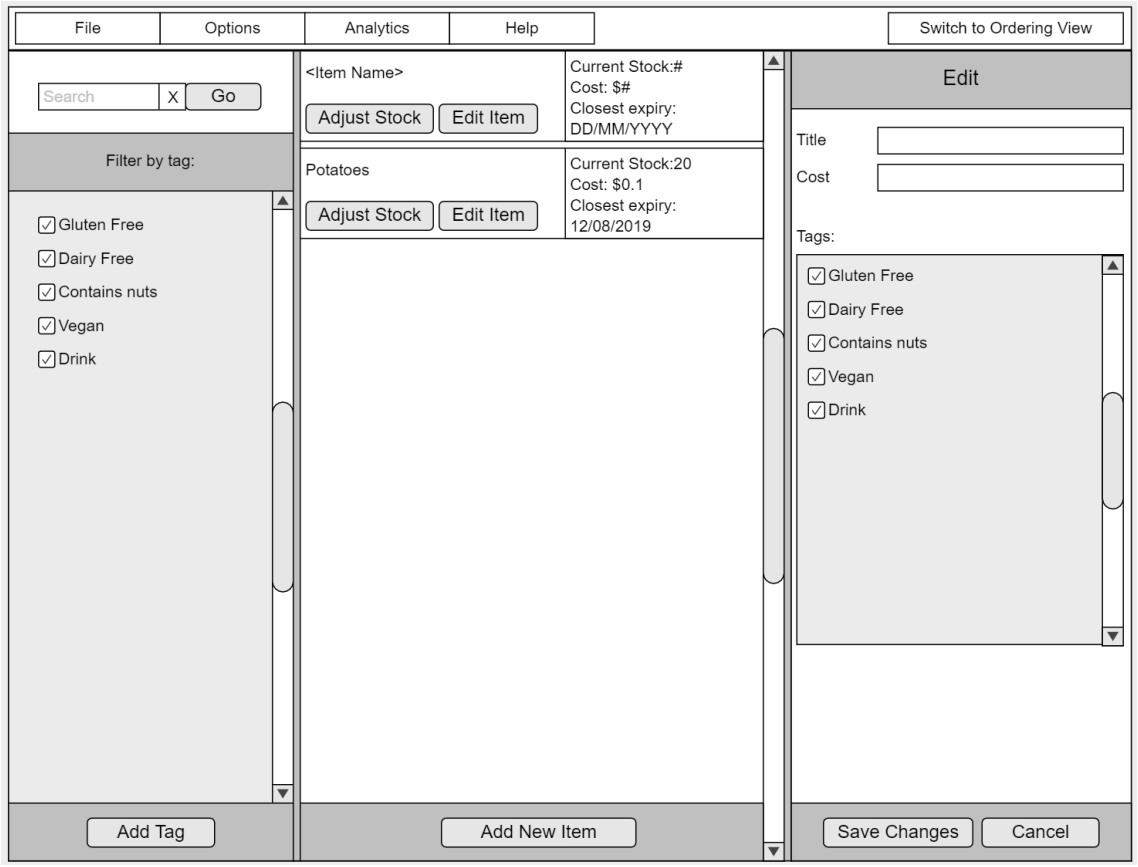

Figure 4: Refined mock up of the inventory management screen

rapid implimentation of features even when under time pressures close to the devlierable deadlines.

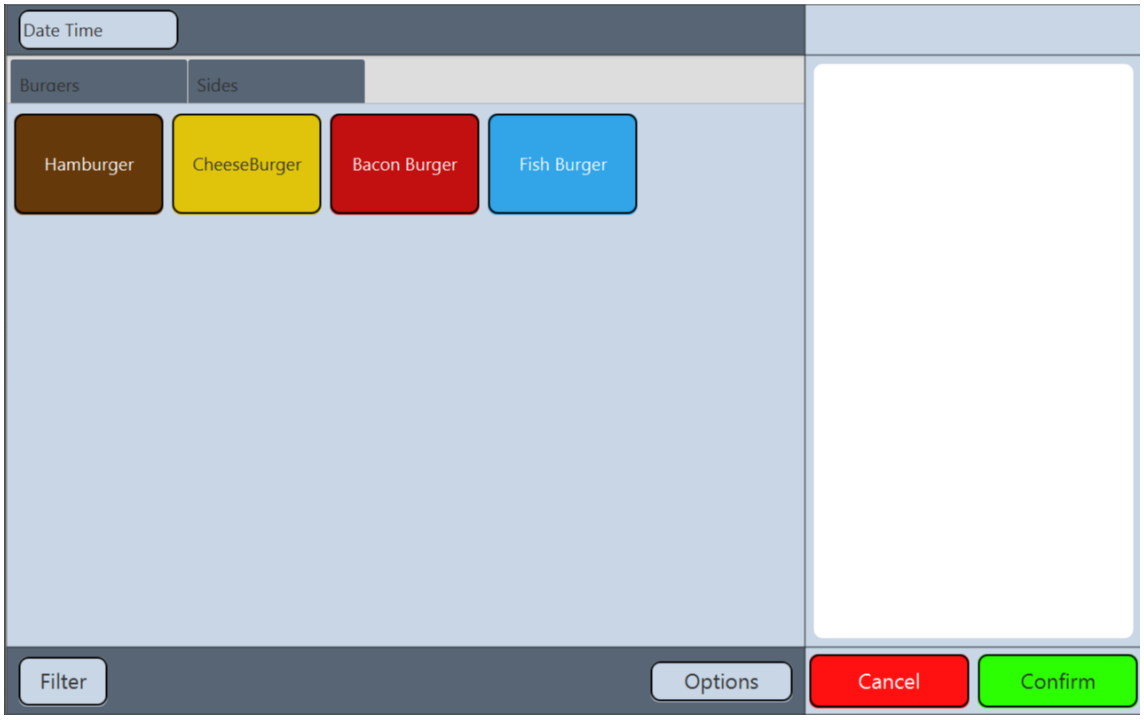

Figure 5: Initial implimentation of the prototyped defsign

Table 1: Summary of exposures to different risks that could occur during development

	Team Risks			
ID	Description	Likelihood %	Impact 1 - 10	Exposure
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-10	Bug in software e.g. bug says something gluten free when actually it isn't	90%	10	9

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

12.2 User Risks

12.3 Risks Discussion - talk about what risks we have encountered and how we prevented / mitigated their effects, and justify why some values changed

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

Table 2: Summary of consequences to different risks that could occur during development

ID	Consequences	Justification˜of
R-01	Reduced Productivity	Team has rules
R-02	Reduced Productivity as time spent learning how to use dev tools	Members of the
R-03	Some members limited to certain tasks, may mean some membes have to do more work than others	This is the tean
R-04	Doing tasks incorrectly and will have to redo or get a bad mark	The team has v
R-05	Have to re-complete work / manually import	All members of
R-06	Software won't sell (fake world) / bad mark from lecturers (real world)	It is very unlike
R-07	Reduced Productivity / Code has to be re-written	With new tools
R-08	Reduced Productivity, Can't commit new changes, may have to switch platforms	These services a
R-09	Can't commit changes to GitLab	Internet is prov
R-10	People could get sick	It is very unlike

Table 3: Summary of exposures to different risks that could occur during use of the software

User Risks				
ID	Description	Likelihood˜%	Impact˜1 - 10	Ex
R-11	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

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.

Table 4: Summary of consequences to different risks that could occur during use of the software

ID	Consequences	Justification˜of likelihood˜percentages
R-11	People could get sick	It is very likely that a user makes a mistake as mistakes happen frequentl
R-12	Angry customers lines get long, lose order	There is a low chance that the system will have a bug that will crash the
R-13	Angry customers	There is a low chance that the system will have a bug with such an integ

14 Testing Protocol

1	RowCol1	Row1Col2
2	Row2Col1	Row2Col2
3	Row3Col1	Row3Col2

Table 5: First Table

15 Project Timeline

16 Document Changelog

Appendices

A Class Diagram