iFOS
Intuitive Food Ordering System
Team 5

Phase 3

Yu Duan, James Kwok, Daniel Harris, Michael Morgoun, Shivin Gaba, Tasman Berry

Submission Date: Monday, 14 October 2019

Contents

Executive Summary

With a rise in the number of people consuming street food, there has been a big investment in the food truck industry. Food truck business usually operates in a fast paced environment and for a business to be successful it should be able to perform efficiently and without any errors. Therefore, there needs to be an application to cater for the people associated with this industry and its increasing demand. The Intuitive Food Ordering System (iFOS) is a system designed and targeted at a food truck business. The system is designed to be used on a laptop with mouse input, or on a touchscreen device with touch input. The key features of the system is the ability to track and store customer orders, ingredients, stock level, recipes, profits, and expenses.

Four stakeholders recognised are the food truck owner, development team, workers and the customer, these were based on their priorities determined by the team through how they would interact and their concerns with the system. The food truck owner was identified to be the most important stakeholder, as success of the system will improve the efficiency and manageability of the business, which will increase profits and simplify training for employees. Furthermore, the success of the system reflects the skills and professionalism of the development team, hence making the team a high impact stakeholder. The most prioritised quality requirements that were highlighted as key drivers were reliability, usability, efficiency, maintainability and security. These requirements were used to identify the most important features to the stakeholders of the system.

Some of the identified use cases for the system are creating an order, modifying stock, exporting data and modifying items. Each use case furthermore has a functional requirement associated with it, which was determined through the steps and flows from the textual use case description in section 2.2.2. All functional requirements were ordered based on priority through how how much on an impact it would make on the system. Furthermore quality requirements were discussed and prioritised, to ensure the product is easy and intuitive to use. These quality requirements were also prioritised through a key drivers table. Where Reliability, Efficiency and Usability were deemed most important when designing the program.

A UML package diagram is included along with a class diagram to show how the application interacts with each other. Notably there is a hierarchy, where classes can only access others inside of it's package, and only classes below in the hierarchy. This kept the structure of the system consistent and moderated the cohesion of the program.
Notable features which stand out over similar systems are a touched based design, visual representation of sales data, the ability to refund and create past orders and a loyalty system. Furthermore, testing procedures were implemented and carried out throughout the project following a test driven development model. Manual tests were documented to ensure each functional use case is met for the system. Furthermore each quality requirement is tested as it was specified in the Table 2.3.

The most important risks that were considered were setting unclear objectives and focusing on wrong design aspects. Both risks are caused by poor communication, thus keeping teammates informed and ensuring everybody agrees on the requirements is paramount.

The development of the software was planned using a Program Evaluation Review Technique chart where there is a visual representation of the time required. Extra time was allocated to account for assessment times. Multiple features were considered but were not planned to be implemented due to time constraints. These planned features can also be seen in section 6.2.

Chapter 1

System and Business Context

1.1 System Context

iFOS is designed for a food truck businesses to handle point of sales operations and business management. These functionalities include processing customer orders, stock tracking, managing menu items, and computing profits. The system will remove the need for recording data on paper or excel sheets, as all relevant data will be stored in one location on the computer along with the system.

The Figure 1.1 displays stakeholders interacting with the system, these are the kitchen staff, workers, and the food truck owner. The process of the system is as follows, the worker will place customer orders using the GUI, which will feedback to the worker relevant information to the order, such as cost. The system will automatically log these orders as sales data, and stock level will decrease appropriately. To achieve this process, the system will require menu items, their recipes and stock, which will be inputted by the owner and the worker.

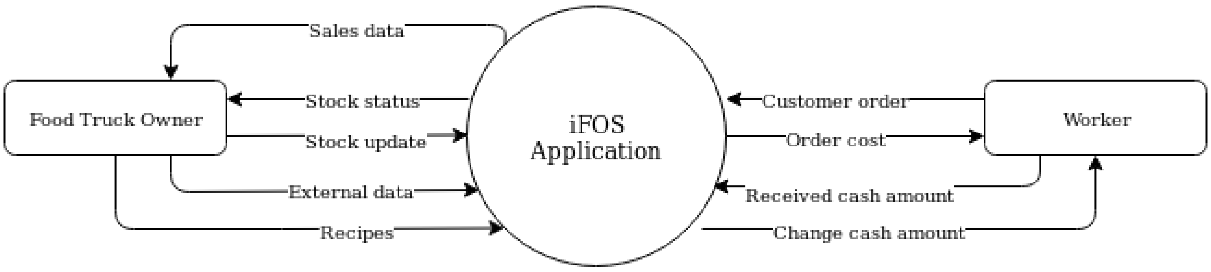


Figure 1.1: System context diagram displaying entities interacting with the system.

1.2 Relevant Business Information

Food truck businesses are increasing as many entrepreneurs opt for a food truck rather than purchasing a restaurant because food trucks are much more affordable. The effects are present at the University of Canterbury, where food trucks gradually increased in front of North Art lecture theatres. This gives the opportunity to develop a system designed for a food truck business. Users of other point of sales systems have noted the harsh and rugged layouts of other systems. Furthermore some users noted that systems were unable to track and manage the money in the till. These users were also asked what would make a good sales system, the main priorities for them were the system to be fast and reliable, whilst the system should be easy navigate and manage during busy times.

1.2.1 Services

The following services were designed for the application. These services were decided in conjunction with identifying stakeholders and their needs in Section 2.1.

Tracking Customer Orders

By tracking customer orders, the profits and expenses of the order can be viewed for analysis. The system also makes refunding orders possible and easy to track. Orders will display information of items in chronological order. This includes the item ingredients, the item recipe, the cost of the order, and the change to give back to the customer. Furthermore, the application can calculate the denominations required a refund.

Track Ingredients, Stock level and Recipes

The system will store important information such as ingredients, stock level, and recipes. Storing the ingredients and the stock level enables the system to notify the user when an ingredient is running low. Storing the recipe of each item and allowing the owner to change, add, and delete recipes. Furthermore the system decreases the stock level when an order is made.

Track Profits and Expenses

Profit and expenses are recorded for the owner or a hired accountant to manage the finance of the business. The system allows the owner to select and view data from any period of time.

Unique Selling Points

The system's unique points are the ability to manage dietary requirements for menu items, which should also consider additions to a menu item and if that item still meets the dietary requirements. The system will be easy to be use, as both the interface will be easy to navigate, with large buttons. And the system will be simple to learn and utilise. Sales data is also easy to view and manage, a user can select a range of dates to view the profits over and see the trend on a graph. The system also has a modifiable loyalty system, where members can spend earned points on discounts when completing a purchase.

Chapter 2

Stakeholders and Requirements

2.1 Stakeholders

2.1.1 Personas

Personas were created to envision the workers that will interact with the system and the employees that will be affected by the system. The following personas display their interest and concern with the system. The personas were used to generate stakeholders, requirements, use cases, and to clarify on the requirements specified for the system.

The developers were considered as stakeholders (S2) and were placed at a high priority. This was due to the result of the product, and the effects it could have on the reputation of the developer, as well as future career prospects. Furthermore, if this was a project in a professional work environment, it could have an impact on the business's reputation and their finances. The developers are also a high priority as the experience that they have affects the quality of the application, therefore they would have the largest impact on the quality of the product.

Customers

- Karen the 35-year-old 9-5 worker, only wants to buy dinner at a nearby food truck. The order must be quickly and easily processed as she is easily stressed and has a lack of patience.
- Katie is a university student and is vegan. She wants to know which meals are vegan and requires the option to change an item to suit her needs.
- Logan, a father of two primary school kids would like options for smaller meals for his children.
- A group of intermediate/high school students want lunch deals and quick service.

Workers

- Ben is a poor university student, working part-time at the food truck as a cashier and kitchen staff and wants an easy time at his job. He wants the application to be intuitive and easy to learn.
- Vladimir a non-native English speaker is working at the food truck and may have difficulties with understanding the application. Therefore the application needs to be and easy to learn. Other features he wants are changing the system to his language, and images and icons.
- Fred aged 49, the owner of the food truck for multiple years has decided he wanted something to help grow the business. He needs an application to make running his business easier. He is interested in using the application to upload recipes, but he has no idea how it works.
- Samantha aged 26, dropped out of high school and worked full time in retail. She decided to create a new business, however as a newbie in the industry, she would like to learn how the food truck industry operates. She needs the system to be intuitive, and has all the necessary features to operate a food truck business with ease.

2.1.2 Table With Stakeholders

The system was designed concerning the stakeholders in Table 2.1. Although not all stakeholders would interact with the system, they had significance in deciding the requirements. Stakeholders that have the most interaction with the product (such as the food truck owner, development team and worker) have higher priorities. Even though customers do not interact directly with the system, they have high priority as it effects the entire ordering process. Another stakeholder that was included in previous phases was the kitchen staff, but it was decided that the kitchen staff was essentially the same as the worker, thus it was decided to be removed as a stakeholder and considered a part of worker. The main concern of the stakeholders is efficiency, if the product is less efficient than what was used previously, then the system is useless.

Table 2.1: The stakeholder table prioritises stakeholder and displays their concerns with the system.

ID	Stakeholder	Description	Concerns	Priority (1: Low, 10: High)
S1	Food truck owner	The owner of the food truck that desires the system.	Wants to improve the efficiency of the business. Interested in faster ordering times, increased sales and providing easy training to employed workers.	10
S2	Development team	The software engineering students that are working to create the system.	The system reflects their skills and professionalism, which may influence future career prospects.	8
S3	Workers	Employees hired by the food truck owner that takes orders from customers.	Needs to be able to use the system effectively with minimal training.	8
S4	Customers	The customers that want to order food from the food truck.	An inefficient system would take longer for orders to be processed, which would decrease customer satisfaction. Other quality of life features may increase or decrease customer satisfaction.	6
S5	University of Canterbury	The university that the development team studies at.	They provide the resources required to create the system, and they expect the system to reflect the effectiveness of their teaching environment. This may positively or negatively affect their reputation depending on the quality of the system.	4
S6	Course staff	The SENG202 tutors and lecturer supervising and guiding the project.	A poor product would also reflect poorly on them, and negatively affect their reputation.	4
S7	Accountants	Accountants hired by the food truck owner to manage expenses, profits and tax of the business.	The effectiveness and accuracy of viewing financial data on the system. There would be no mistakes in the financial data.	3
S8	Local council/Government bodies	Health inspectors, Inland Revenue.	The system must be accurate and calculate the correct finances regarding profits and expenses of the business for tax purposes, renewing licences etc.	2
S9	Other business owners	Other business owners that owns a food truck business.	The success of the system decreases the ordering time. This may raise interest and may convince them to inquire about the product for their own business. The success of the system may also reduce their sales.	1
S10	Suppliers	Other companies that provide supplies of ingredients to the food truck business.	They would sell more supplies if the food truck business they are providing to becomes more successful.	1

2.2 Use Cases

2.2.1 Use Case Diagram

Previously defined stakeholders (Table 2.1) and the system context diagram (Figure 1.1) were used to identify actors. The use cases were identified in reference to "...the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal." Cockburn, 1999. Figure 2.1 displays the use cases and actors that belong to the system.

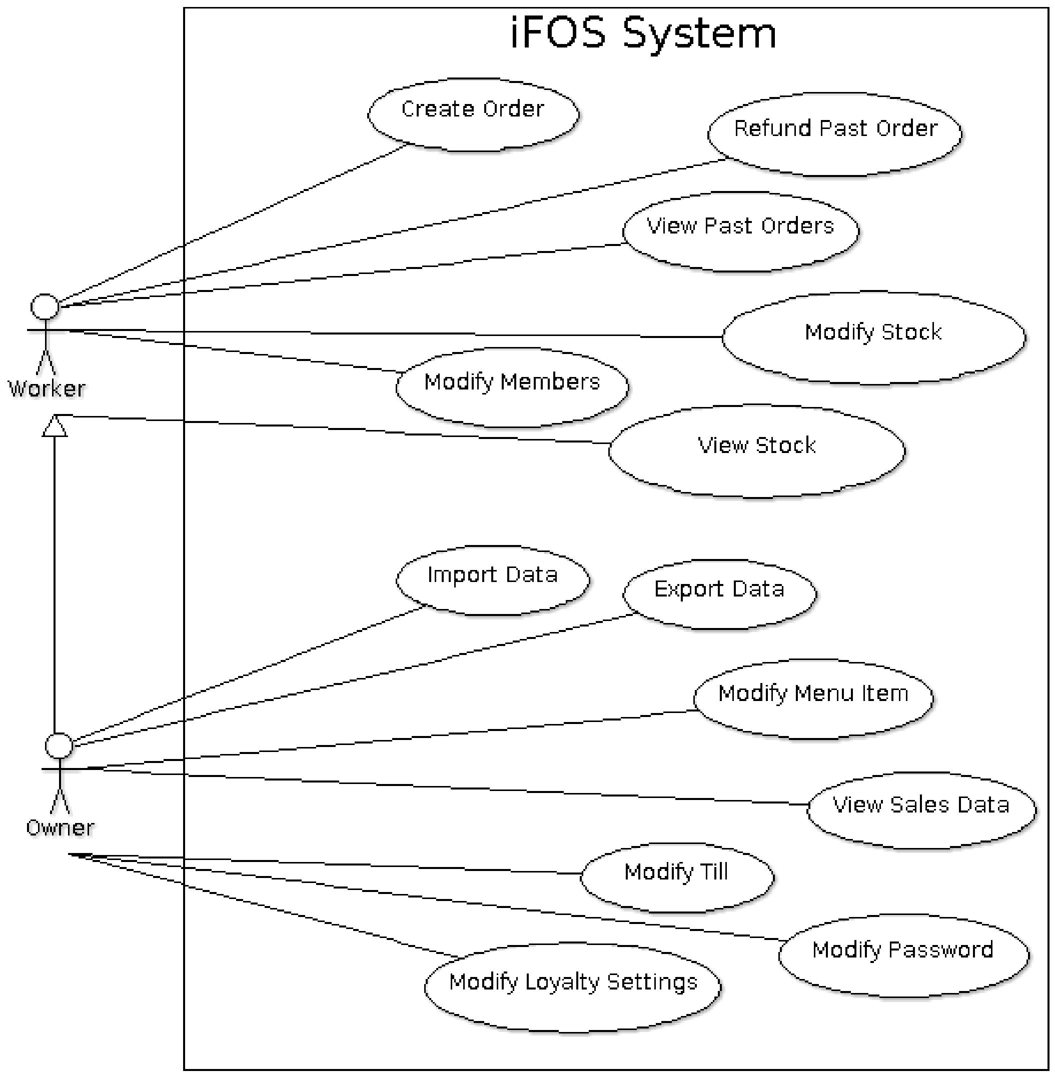


Figure 2.1: The use case diagram displaying use cases with their relation with the actors.

From previous iterations of the use case diagram, the number of use cases have been reduced. Additional details such as financial information and other requirements have also been omitted. This was to make the diagram more abstract, whilst also to limit the scope of the project.

The use cases were used to identify functional requirements for the project, as well as an aid for developing the UML class diagram in Section 5.2. The textual use cases are explored in more detail in Section 2.2.2.

2.2.2 Textual Description of Use Cases

Highlighted below are the basic and alternative functions of the system when it reaches a use case. Each use case is described in a form found in "The ≪include≫ and ≪extend≫ Relationships in Use Case Models" Rich Johnston, n.d.

(UC1) Use case: "Create Order"
Actor/User: Worker
Pre-conditions: No order currently exists.
Post-conditions: Order added to order history
Basic Flow:

1. User navigates to order menu
2. User adds menu items to order
3. User inputs payment
4. User confirms payment and order is added to order history
5. System displays change
6. System creates a new order

Alternative flow:
2a. The user changes the ingredients of the items that are in the order

1. User selects menu item existing in order
2. User modifies the ingredients in the item
3. Menu item cost changes resulting the order cost to change
4. Continue with basic flow step 3

3a. The customer wants to use loyalty discount points to order

1. User selects customer
2. User selects how many loyalty discount points
3. The system applies discount and reduces total cost of order
4. Continue with basic flow step 4

Exceptional flow:
2a. There is not enough stock for the menu item to add to the order

1. The system prompts the user that there is no stock and the menu item is not added to order
2. Continue with basic flow step 2

3a. The customer decides to cancel the order and leave

1. The user cancels the order
2. The system clears the items in the order
3. The order is cancelled so use case ends

(UC2) Use Case: "Refund Past Order"
Actor/User: Worker
Pre-conditions: Past order exists in the order history
Post-conditions: Past order is shown as refunded and appropriate denominations are removed from the system.
Basic Flow:

1. User selects the past order
2. User refunds the past order
3. System displays the money to refund

4. The system reduces the money in the till

Exceptional flow:
2a. There is not enough money in the till

1. System notifies the user that there is not enough money in the till to refund the order
2. System cancels the refund of the order
3. Use case ends

(UC3) Use Case: "View Past Orders"
Actor/User: Worker
Pre-conditions: Order history contains past orders
Post-conditions: No change to order history
Basic Flow:

1. User access the order history data
2. The system displays to the user a list of past orders, with the ID, data, time and menu items ordered with individual costs and the total cost, and whether this order was refunded.

Alternative Flow:
2a. There are no past orders in the system

1. The system displays no past order data
2. Use case ends

(UC4) Use Case: "Modify Stock"
Actor/User: Worker
Pre-conditions: None
Post-conditions: Changes to the stock data is updated and saved
Basic Flow:

1. User access the ingredients in the stock data
2. User can add an ingredient to the stock, this includes the name, category, cost, quantity, and dietary information of the ingredient
3. The user confirms and the system adds the ingredient to the stock

Alternative Flow:
2a. The user selects to modify an existing ingredient in the stock

1. User selects the ingredient to modify
2. User changes the values of the ingredient, such as the name, category, cost, quantity and dietary information of the ingredient
3. The user confirms and the system modifies the ingredient

2b. The user selects to delete an existing ingredient in the stock

1. User selects the ingredient to delete
2. User deletes the ingredient
3. The system removes the ingredient

Exceptional Flow:
2a. The new value of the ingredient fields are invalid or blank

1. System displays a warning about the invalid or blank field or prevents user from entering invalid values
2. Creation or modification of ingredient is stopped
3. Use case ends

(UC5) Use Case: “View Stock”
Actor/User: Worker
Pre-conditions: Ingredients exist in the stock
Post-conditions: No changes to the stock
Basic Flow:

1. User access the stock data
2. The system displays the ingredients, this includes the name, category, cost, quantity and dietary information of the ingredient

Alternative Flow:
2a. There are no ingredients recorded in the stock

1. The system displays no ingredients

(UC6) Use Case: “Import Data”
Actor/User: Owner
Pre-conditions: There are XML files for the stock, menu, and finance
Post-conditions: Data is loaded to the system and the system updates
Basic Flow:

1. User selects the XML files for stock, menu, and finance
2. User imports the three selected XML files
3. The system updates the existing data in the system with the imported data

Exceptional Flow:
2a. The selected data does no abide to the XML schema’s

1. The system notifies the user that the file is incorrect
2. The system does not import the data
3. Use case ends

(UC7) Use Case: “Export Data”
Actor/User: Owner
Pre-conditions: There are data in the system
Post-conditions: Files containing the data are exported to the selected directory
Basic Flow:

1. User selects the directory to export the data
2. The system exports files with data to the directory

Alternative Flow: 2a. There are no data in the system

1. The system exports files with no data to the directory
2. Use case ends

(UC8) Use Case: “Modify Menu Item”
Actor/User: Owner
Pre-conditions: None
Post-conditions: Changes to the menu data updated and saved
Basic Flow:

1. User access the menu data
2. User can add a menu item to the menu, this includes the name, image, recipe instructions, the menu type, the markup cost, and the ingredients in the menu item’s recipe
3. The system computes the ingredient cost, total cost of the menu item and the dietary information of the menu item based on the ingredients in the recipe
4. The user confirms and the system adds the menu item to the menu

Alternative Flow:
2a. User selects to modify an already existing menu item in the menu

1. User selects the menu item to modify
2. User can modify the following of the menu item, name, recipe instructions, the menu type, the markup cost, and the ingredients in the recipe
3. The system modifies the menu item

2b. User selects to delete an already existing menu item in the menu

1. User selects the menu item to delete
2. User deletes menu item
3. The system removes the menu item

(UC9) Use Case: "View Sales Data"
Actor/User: Owner
Pre-conditions: None
Post-conditions: No change to the sales data
Basic Flow:

1. User selects the start and end dates of the sales they want to view
2. The system displays the total cost of orders, average daily cost, total profits and average daily profits within the selected dates
3. The system displays a graph showing the total profits within the start and end dates

(UC10) Use Case: "Modify Till"
Actor/User: Owner
Pre-conditions: None
Post-conditions: The amount of each money denomination has changed in the system
Basic Flow:

1. The user changes the amount of the money denominations desired
2. The system updates with the new amounts

Alternative Flow: 1a. The user resets the till

1. The system sets all the amount of money denominations to zero
2. Use case ends

(UC11) Use Case: "Modify Members"
Actor/User: Worker
Pre-conditions: None
Post-conditions: The loyalty member's data is changed
Basic Flow:

1. User inputs the customer's data, including name and phone number.
2. The system adds the customer's data to the system.

(UC12) Use Case: "Modify Password"
Actor/User: Owner
Pre-conditions: None
Post-conditions: The password is set to the new password given
Basic Flow:

1. User enters old password, and new password
2. User confirms and the system updates the password

(UC13) Use Case: "Modify Loyalty Settings"
Actor/User: Owner
Pre-conditions: None
Post-conditions: The loyalty member system is modified
Basic Flow:

1. User changes values, such as initial purchase points, point to dollar ratio and the value of each point
2. User saves values
3. The system updates the loyalty member system with these new values

2.3 Requirements

The requirements were made with the SMART mnemonics. SMART objectives increase the understanding of the requirement and to know when they were achieved. The requirements being specific and measurable made them easier to implement, while being realistic prevents wasting time.

2.3.1 Functional Requirements

The initial functional requirements were based off of the feature pack descriptions. The functional requirements are displayed in Table 2.2. These are linked to the stakeholders in section 2.1 and use cases in section 2.2.1. The table below explains the capability needed by a stakeholder to solve a problem. It also reflects the mandatory requirements that must be met or possessed by a solution. The solutions to meet the specified requirements were written in the form of use cases. The priority are from 1 to 10 which corresponds to low and high accordingly. This impacts which requirements are kept if not all requirements can be implemented before the final deliverable. The priorities of the functional requirements were derived from the feature packets, features in the first feature packet would most likely be given more priority than one is the following feature packets.

Table 2.2: The table describes the functional requirements in consideration of stakeholders. Priority is from 1 - low to 10 - high.

ID	Description	Stakeholder	Priority	Use case(s)
FR1	The user should be able to view data on ingredients stored in the stock, this includes name, unit, stock level, and the cost of ingredients. All data should be viewed in a table of ingredients with no data being incorrect.	Workers (S3), Owner (S1)	8	View Stock (UC5)
FR2	The total cost of the order displayed to the user should be equal to the sum of the cost of all items in the order.	Workers (S3)	9	Create Order (UC1)
FR3	Can select a menu item to add to the order. The total cost of the order should be recalculated and displayed when menu items are added. Menu items can be removed and the order can be cancelled, which removes all menu items.	Workers (S3), Customers (S4)	9	Create Order (UC1)
FR4	When order payment is confirmed, ingredient stocks should be reduced by the amount of ingredients used in the order and sale details including the order id, order cost should be recorded in the system.	Workers (S3)	9	Create Order (UC1)
FR5	When order payment is confirmed, order details including the order id, date of transaction, time of transaction, order total cost are recorded in the order history.	Workers (S3)	9	Create Order (UC1)
FR6	Orders should be refundable to the customer, reducing only the amount of cash recorded in the system's till.	Workers (S3), Customers (S4)	5	Refund Past Order (UC2)
FR7	The user can manually update the stock, this includes adding new ingredients, removing ingredients, and modifying the existing ingredients. Values of the ingredients that can be modified by the user include the name, category, cost, quantity, and the dietary information of the ingredient.	Owner (S1), Workers (S3)	8	Modify Stock (UC4)
FR8	The system will check that there are enough ingredients available in the stock to add an menu item to the order. If there is enough, the item is added to the order. If there isn't enough, the system will notify the user that there isn't enough ingredients in the stock to add the menu item to the order.	Workers (S3), Customers (S4)	7	Create Order (UC1)

FR9	Keep track of notes and coins in the float when transactions are made and change is given. This includes adding and removing coins and notes in the float automatically when a order transaction is made. The coins and notes added to the float is determined by the input of the user, while the removal of coins and notes is computed and displayed to the user.	Owner (S1), Workers (S3), Accountant (S7)	4	Create Order (UC1)
FR10	Able to import external XML data files which store stock, finance, history and menu items. The data in the system is cleared when new data is imported.	Owner (S1)	9	Import Data (UC6)
FR11	Able to export system data including stock, finance, history and menu items to xml files.	Owner (S1)	9	Export Data (UC7)
FR12	Can view items on the menu and the ingredients, ingredients' quantity in the item.	Workers (S3), Owner (S1)	9	Create Order (UC1)
FR13	Able to modify, add, and delete menu items from the system. This includes selecting the ingredients for the recipe, the markup cost of the menu item, and the name of the menu item.	Owner (S1)	10	Modify Menu Item (UC8)
FR14	Able to modify the menu item temporarily changing the ingredients and quantities when the item is in the order.	Workers (S3), Customers (S4)	5	Create Order (UC1)
FR15	Items should be displayed in proper orders (sorted) depending upon how a worker would like the application to display the ordering page. i.e. A worker may arrange the items according to the price but the owner may prefer the items to be in alphabetical order.	Workers (S3), Owner (S1)	3	Create Order (UC1)
FR16	Sales data regarding past orders are displayed to the user. This includes the total cost of orders, average daily cost, total profits and average daily profits within a date range.	Owner (S1)	3	View Sales Data (UC9)
FR17	Able to view past order transactions with the ID, date, time, and menu items ordered with individual costs and the total cost, and if the order is refunded or not.	Owner (S1), Accountant (S7)	3	View Past Orders (UC3)
FR18	Use the ingredient costs and recipes to compute the cost of each menu item. The price is based on this cost, plus an appropriate mark-up.	Owner (S1)	2	Modify Menu Item (UC8)

FR19	The owner should be able to generate visual representation of historical data such as stock and sales, and filtered by a particular hour, day, week, or month. They should be able to use this to get an estimate of peak times and average quantity of ingredients used. This information should be displayed in a relevant form of graph, like a bar graph.	Owner (S1)	2	View Sales Data (UC9)
FR20	The owner should be able to manually update the quantity of each money denomination of the till.	Owner (S1)	1	Modify Till (UC10)
FR21	A loyalty system created so that returning customers get discounts for points they ear from spending money. The user should be able to add new customers to the loyalty system and record their name, phone number, and the amount of loyalty points they have. Customers can ask to use these loyalty points to reduce the cost of their order. Loyalty points are earned through ordering.	Workers (S3), Customers (S4), Owner (S1)	1	Add Customer (UC11), Modify Loyalty Settings (UC13)
FR22	Authorised users should be able to change the password used to access the Admin functionalities, such as menu management, financial data, import/exporting data. This password prevents unauthorised users from access these data.	Owner (S1)	1	Modify Password (UC12)

2.3.2 Quality Requirements

Table 2.3 contains the quality requirements of the system. The requirements were designed focusing not on the functionality but rather the actual experience with the application. These were added in the table according to their order of importance and prioritised accordingly. Unit and time constraints were also used to define the requirements more specifically. The five most important requirements were also used to measure the key drivers later in the report. The table also displays that the five most important requirements are all important to the highest priority stakeholder the food truck owner. The priority are from 1 to 10 which corresponds to low and high accordingly.

Table 2.3: The table describes the quality requirements prioritised with consideration to stakeholders. Priority is from 1 - low to 10 - high.

ID	Description	Stakeholder	Priority
QR1	**Reliability**. The system will give feedback on actions the user take, so they are not left with ambiguity of their actions. The application is consistent that it does not freeze and crash over a long period (10 hours) time of use. Data being inputted to the system and displayed the user must be accurate and unambiguous.	Owner (S1), Worker (S3)	10
QR2	**Usability**. Designed with the needs of the user in mind. The owner or the worker should be able to find the item button that they are looking for in less than 0.5 seconds after they are used to the application.	Owner (S1), Worker (S3)	6
QR3	**Efficiency**. Every action performed by the worker/owner on the application should take less than 0.2 seconds. Able to process orders during busy times. If a customer wants to order a bulk of 20 items, the application should still take the same amount of time as it takes for a single item to process.	Owner (S1), Worker (S3), Customer (S4)	8
QR4	**Maintainability**. Updating the software should not be time-consuming. The software update should not take longer than 1 hour and with the option for the owner to schedule the update for that time that suits him without affecting any operation of his business.	Developer (S2), Owner (S1)	3
QR5	**Security**. Workers cannot access specific parts of the application including sales data and recipe modification.	Owner (S1)	4
QR6	**Extensibility**. There should be grounds to improve the application and to add features. If the application is successful and the owner wants to start taking to add transactions in the near future, it should be possible. The application should be built so adding new features or modifying existing ones could be performed easily by any developer.	Developer (S2), Owner (S1)	2
QR7	**Compatibility**. The application should be compatible with the operating application and hardware the owner would prefer to use. Developer has to make sure that application is able to perform efficiently on the operating system that client would like to use.	Owner (S1)	2

2.3.3 Key Drivers

Table 2.4 reflects the importance of each quality requirement for each stakeholder. The total weighted sum of each requirement in the key drivers table determines the priorities of the quality requirements shown in Table 2.3. The three most important quality requirements are QR1 Reliability, QR3 Efficiency, and QR2 Reliability, which will become the key drivers for the system.

Table 2.4: The key drivers table gives each stakeholder a weight and the importance of each quality requirement to them. The weighted sum equals to 100. The table shows that the three most important quality requirements are QR1, QR3 , and QR2.

Stakeholders	**Weight**	**QR1 Reliability**	**QR2 Usability**	**QR3 Efficiency**	**QR4 Maintainability**	**QR5 Security**	**QR6 Extensibility**	**QR7 Compatibility**
Food truck owner	0.2	25	25	15	5	20	5	5
Development team	0.16	20	10	20	30	10	5	5
Workers	0.16	30	50	30	0	0	0	0
Customers	0.14	50	0	50	0	0	0	0
University of Canterbury	0.08	14	14	14	15	14	15	14
Course Staff	0.08	14	14	14	15	14	15	14
Accountants	0.06	30	0	10	10	40	5	5
Local council/Government bodies	0.05	70	0	0	0	30	0	0
Other business owners	0.04	25	25	15	5	20	5	5
Suppliers	0.03	70	0	30	0	0	0	0
Total	**1**	**30.64**	**17.84**	**22.34**	**9.00**	**12.54**	**4.70**	**4.54**

Chapter 3

Acceptance Tests

The table below (Table 3.1) contains the acceptance tests ordered by criticality. These were implemented in Cucumber and automated to be used throughout development to verify the core classes and functions were working. The acceptance tests were created based on the use cases in Figure 2.1, and then ordered based on the relationship between it and the functional requirements. The responsibility for testing all acceptance tests is the development team, as the development team are responsible for designing and creating the system. The criticality were from 1 to 10 which corresponds to low and high accordingly.

Table 3.1: The table describes the main acceptance tests created with acceptance criteria. Priority is from 1 - low to 10 - high.

ID	Scenario	Given	When	Then	Priority	Use Case
AT1	Order's cost checked after a Burger added to it	New Order is created And A Burger costs $5.00	Burger is added to order	Orders total cost is $5.00	10	Create order (UC1)
AT2	A Burger is added to empty order	New Order is created	Burger is added to order	Order contains a burger	10	Create order (UC1)
AT3	An order containing burger has chips added to it	New Order is created And A Burger costs $5.00 And Burger is in order And Chips cost $5.00	Chips are added to order	Orders total cost is $10.00 And Order contains a burger And order contains chips	10	Create order (UC1)
AT4	Order uses ingredients	Stock has 10 of Buns	6 Buns are used	Stock now has 4 of Buns	10	Update stock (UC12) View stock (UC5)
AT5	Order is payed for with notes and till is updated	Till starts with 3 $10.00 notes	An order is payed for with 2 $10.00 note	Till has 5 $10.00 notes	10	Create order (UC1)
AT6	View an Order	New Order is created And A Burger costs $5.00 And Burger is in order	Order is viewed	Order contains a burger And Orders total cost is $5.00	10	Create order (UC1)
AT7	Order is payed for	Order Costs $7.50 And till starts with change	Payment of $10.00 is confirmed	$2.50 is displayed to be returned	10	Create order (UC1)

Continued on next page

Table 3.1 – continued from previous page

ID	Scenario	Given	When	Then	Priority	Use Case
AT8	A Burger is added to an order without the necessary ingredients	Stock has 0 of Buns And New Order is created And A Burger contains buns And A Burger costs $5.00	Burger is added to order	Receive error And Order Does not contain burger	9	Create order (UC1)
AT9	New order is created		New Order is created	Order is empty And Orders total cost is $0.00	9	Create order (UC1)
AT10	Buns are added to burger Recipe	Stock has 10 of Buns And New Order is created	burger Recipe is modified to contain 1 buns And Burger is added to order	Burger in the order contains 1 buns	8	Modify Menu Item (UC8)
AT11	Order is cancelled	New Order is created And Burger is in order	Order is cancelled	Order Does not contain burger	8	Create order (UC1)
AT12	Create new recipe	There is a menu	A new recipe is created And A new item is created for that recipe	The new recipe is in the menu	8	Modify Menu Item (UC8)
AT13	A Burger is added to order that contains chips	New Order is created And A Burger costs $5.00 And Burger is in order And Chips cost $5.00	Chips are added to order	Orders total cost is $10.00 And Order contains a burger And order contains chips	7	Create order (UC1)

Continued on next page

Table 3.1 – continued from previous page

ID	Scenario	Given	When	Then	Priority	Use Case
AT14	Customer uses points to get a discount	Order Costs \$7.50 And Customer has 10 purchase points	Customer uses 2 purchase points on an order which costs \$7.50	Order now Costs \$6.50	7	Create order (UC1)
AT15	Worker confirms an order for a customer which is registered.	Order Costs \$155.00 And Customer has 0 purchase points	Order gets payed \$155.00	Customer now has 15 purchase points	7	Create order (UC1)
AT16	A Burger is removed from an order	New Order is created And Burger is in order	Burger is removed from order	Order Does not contain burger	6	Create order (UC1)
AT17	A burger is removed from order which contained burger and chips	New Order is created And A Burger costs \$5.00 And Burger is in order And Chips cost \$5.00 And Chips is in order	Burger is removed from order	Orders total cost is \$5.00 And order contains chips	6	Create order (UC1)
AT18	Burger and chips are removed from order which contained burger and chips	New Order is created And A Burger costs \$5.00 And Burger is in order And Chips cost \$5.00 And Chips is in order	Burger is removed from order And Chips is removed from order	Orders total cost is \$0.00	6	Create order (UC1))

Continued on next page

Table 3.1 – continued from previous page

ID	Scenario	Given	When	Then	Priority	Use Case
AT19	Refund order	Order Costs $10.00 And Order has already been payed for	Orders is refunded	$10.00 is displayed to be returned	6	Refund past order (UC2)
AT20	View sales data	Order Costs $10.00 And Order has already been payed for	Sales data is viewed	Order is in sales data	5	View Sales Data (UC9)

Chapter 4

GUI Prototypes

4.1 First Prototype

A GUI prototype was made early in the projects development to visualise the functionality of the application to be built. The prototypes can be seen through Figure 4.1 to Figure 4.7. This allowed members to visualise ideas others were trying to communicate and decide on which features were important to implement first and which features should be implemented later as quality-of-life features. All of the features of the first prototype were based on the use case diagram (Figure 2.1).

Emphasis was placed on not letting the prototype define how the class diagram would be implemented, and furthermore how the project would look like in the future. Its sole purpose was the help communicate ideas and is subject to change as the code is written.

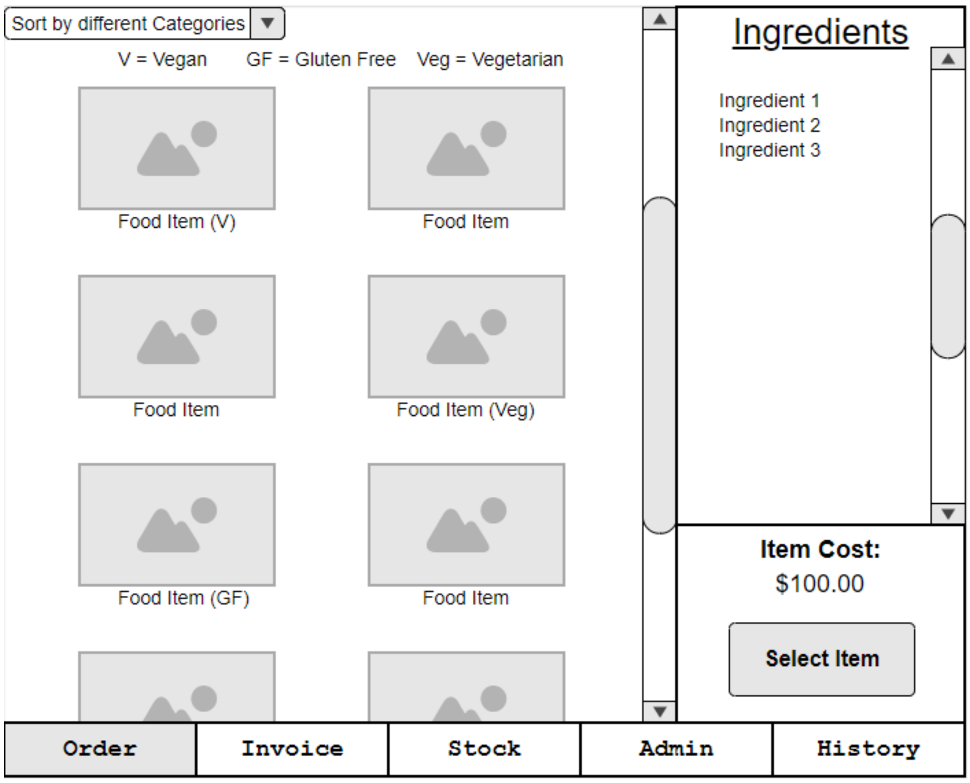


Figure 4.1: The initial order screen containing a menu where items are to be selected and once selected, they display the ingredients and the price of the item.

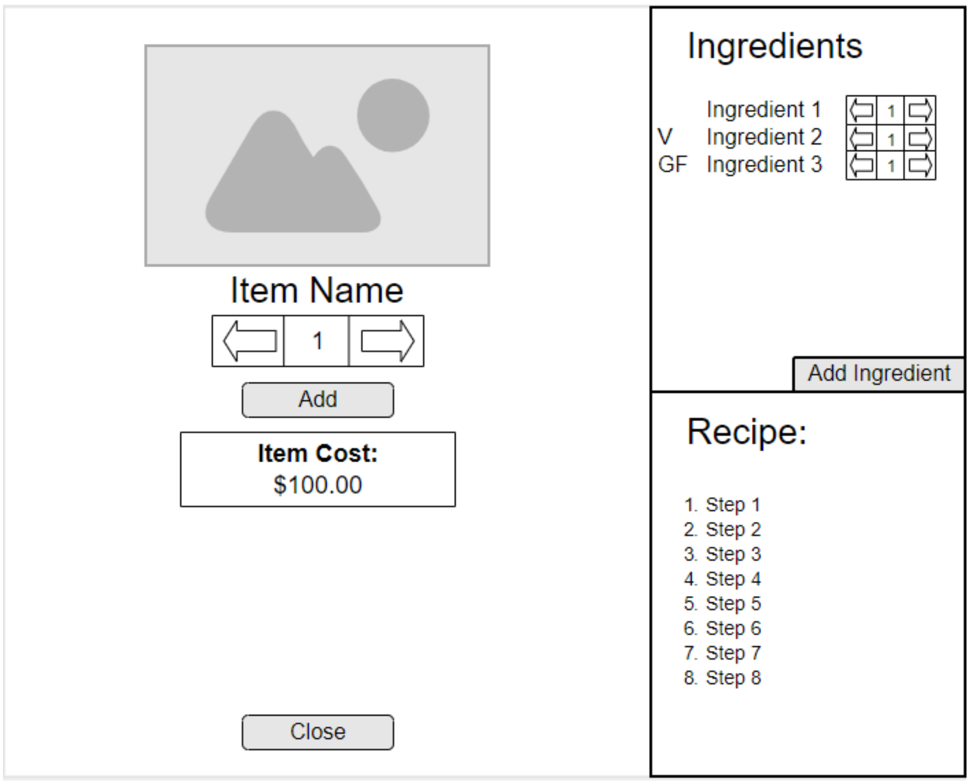


Figure 4.2: The item screen shown when an item is selected from the order screen. Details such as the ingredients and the recipe of the item is displayed and the ability to change the quantity of items to add.

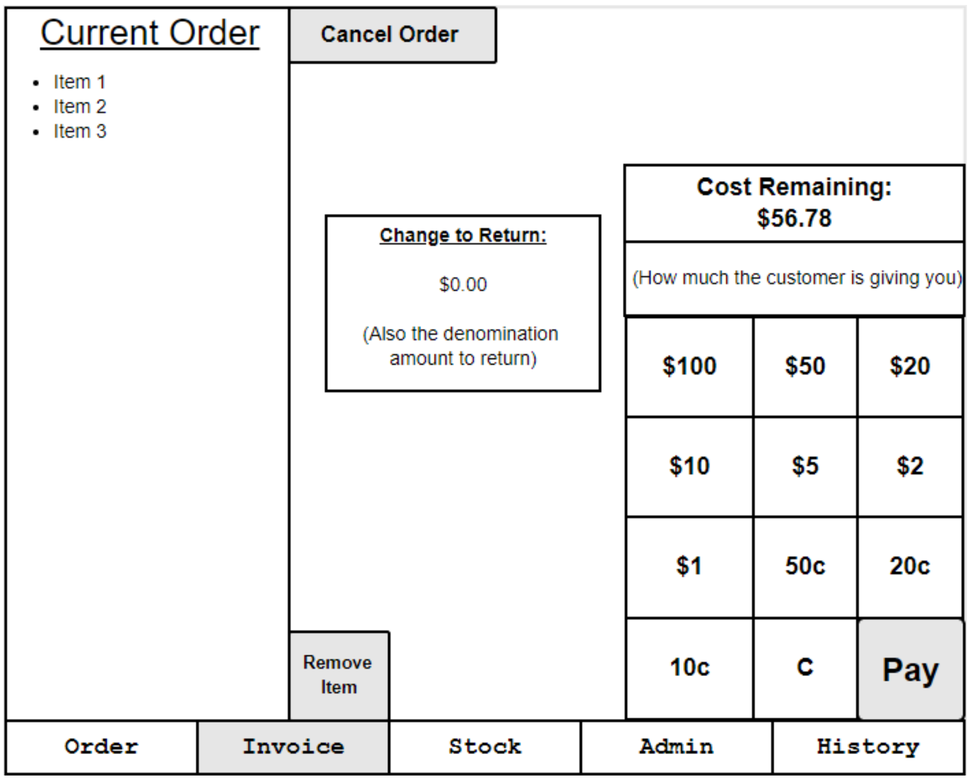
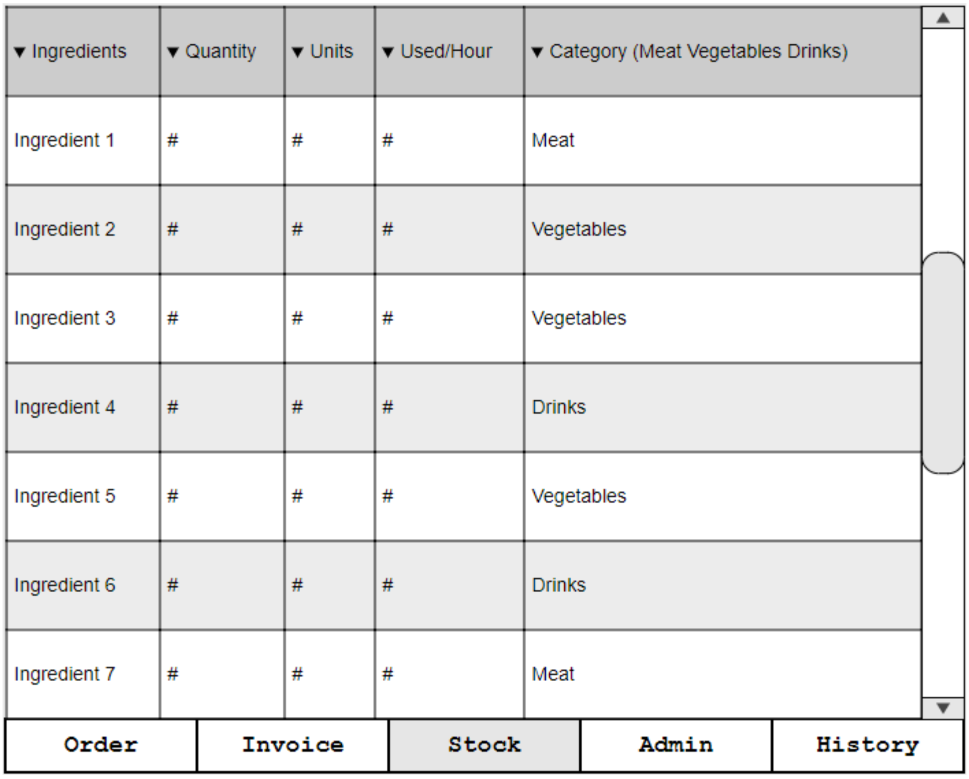
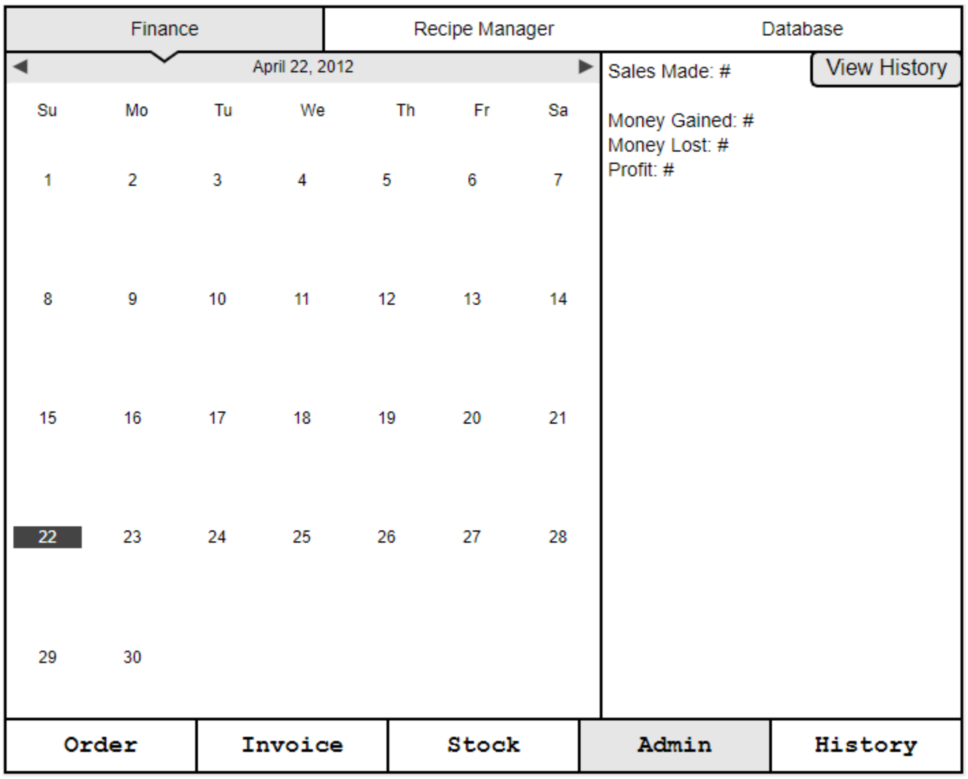


Figure 4.3: The invoice screen shown once an order is ready to be paid for. Allows the worker to see the cost remaining and add the denominations of what the customer is paying.



▾ Ingredients	▾ Quantity	▾ Units	▾ Used/Hour	▾ Category (Meat Vegetables Drinks)
Ingredient 1	#	#	#	Meat
Ingredient 2	#	#	#	Vegetables
Ingredient 3	#	#	#	Vegetables
Ingredient 4	#	#	#	Drinks
Ingredient 5	#	#	#	Vegetables
Ingredient 6	#	#	#	Drinks
Ingredient 7	#	#	#	Meat

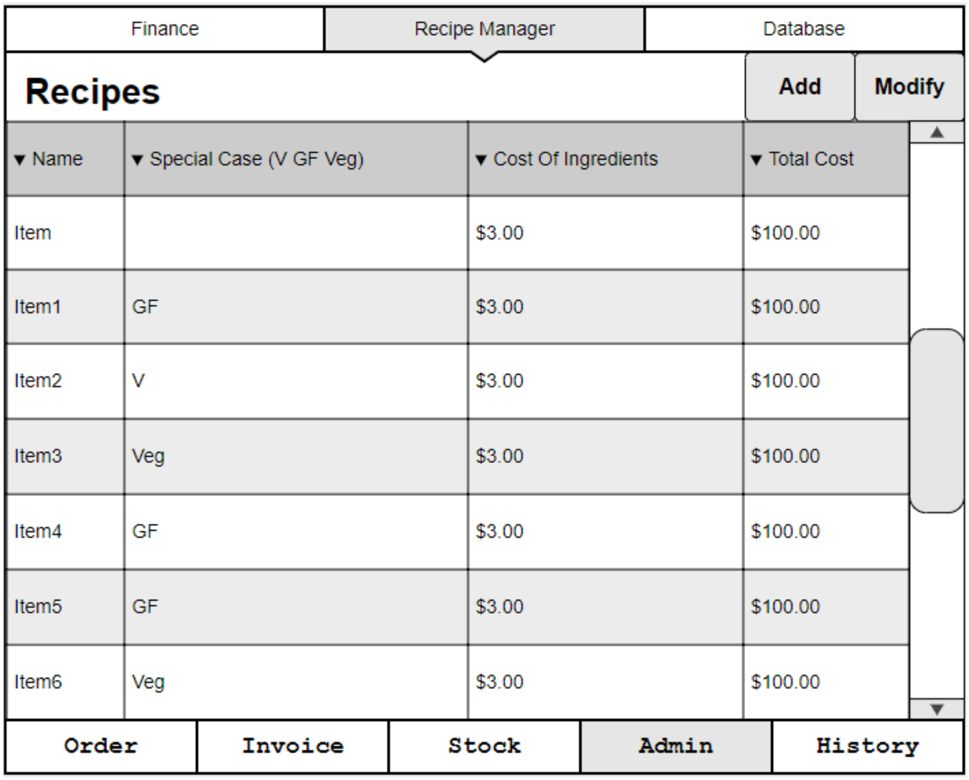
Figure 4.4: The stock screen showing every ingredient and their respective quantities, units, how much was used in the hour and which category it falls under.



Sales Made: #

Money Gained: #
Money Lost: #
Profit: #

Figure 4.5: The admin screen with has three parts, first of which being a finance screen that shows profits for each selected day in the calendar.



▼ Name	▼ Special Case (V GF Veg)	▼ Cost Of Ingredients	▼ Total Cost
Item		$3.00	$100.00
Item1	GF	$3.00	$100.00
Item2	V	$3.00	$100.00
Item3	Veg	$3.00	$100.00
Item4	GF	$3.00	$100.00
Item5	GF	$3.00	$100.00
Item6	Veg	$3.00	$100.00

Figure 4.6: The recipe manager which is a part of the admin screen. Displays all recipes currently in the system and their cost. The screen allows modification of current recipes and addition of new recipes.

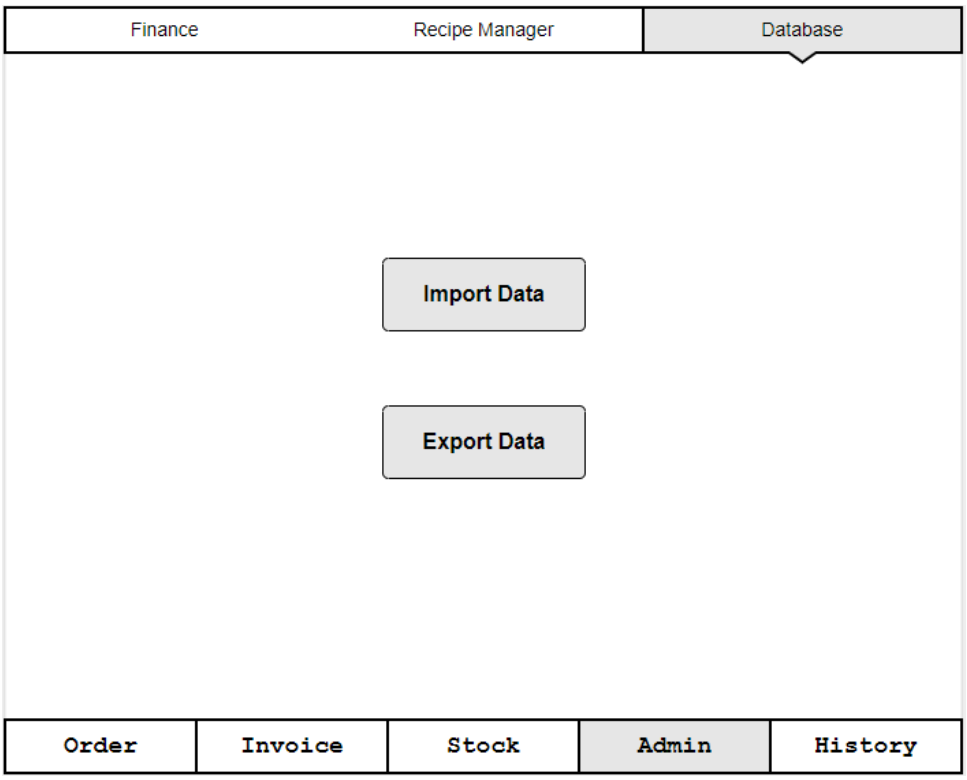


Figure 4.7: The database screen is the last part of the admin screen and allows data to be imported through the computer as XML files and exported data into a XML file.

Chapter 5

UML Diagrams

5.1 Deployment Model

The model in Figure 5.1 displays how the system will interact with the device and save the files on a device (PC1). Initially, it was planned to include a backup feature which would save the state of the database.

The model has been updated from the previous version in which the artefact iFOS.jar initially used to be on the outside of the device. In the new updated version the artefact is nested within the device which only accepts certain .xml files. For the application to be fully functional, the user should import the required files, or start from a default state.

However, a decision was made to remove the feature from the project as it was more of a quality requirement than a functional one. Implementing an external database would set the scope of the second deliverable too wide. Therefore it may be implemented as an option dependant on time constraints.

Having an external database would allow the main database to be saved externally, therefore lost data can be recovered easily. This could be done over the internet if available, or through external hard drives. Currently, the deployment model includes the software and the system it is running on, but the model will hopefully evolve as the project develops.

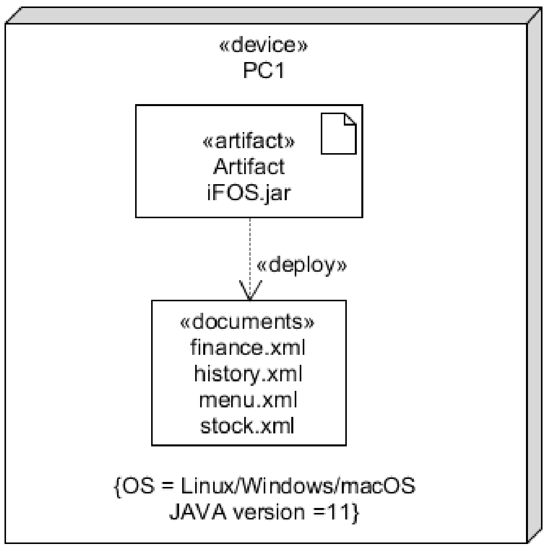


Figure 5.1: Deployment model displaying the interaction between iFOS.jar and PC1.

5.2 Detailed UML Class Diagram

The creation of the UML class diagram was based on the use cases found in Figure 2.1. The class diagram is in a hierarchical layout and can be found in Figure 5.2 below. The following lists the decisions and reasoning in the design of the UML class diagram.

- A Database class was suggested to deal with the raw XML files and its processing but we split the Database class into two classes as it was at risk of turning into a "god class" that would hold too many functions. The new classes were AppEnvironment and Database.
- Classes were created for each main group of data, such as Finance which handles calculation of different profits and storing orders, MenuManager which holds all MenuItems and their respective recipe, Stock which carries every Ingredient in the truck data relating to it and OrderManager which holds functions for the creation of an order and adding items to that order. Furthermore, a Till class was used to manage the cash present in the till. The Order class used by the OrderManager also holds information about the items in the order.
- This group of "logic" classes contain the majority of the logic of the application. These are restricted from accessing the AppEnvironment and GUI classes defined at a higher level. This class hierarchy is explained in more detail in the UML Package Diagram.
- Initially, we planned for all the logic classes except Order to be created in a composition by AppEnvironment. However, this became problematic, as for each order we needed to keep track of the total ingredients available, while still allowing the stock contents to be viewed normally. For this reason, we allowed Order to use a clone of the Stock to track the ingredients that would remain after the order was complete. However, the remaining logic classes were created as we initially planned.
- The remaining classes, including MenuItem, Recipe, Ingredient and Transaction, were stored in the "information" package. These classes were created so they may only access each other, since they are the lowest level set of classes in our design.
- MenuItem and Recipe share a one-to-one relationship, as each Recipe is uniquely assigned to a MenuItem. Future improvements on this product could include allowing items to have multiple recipes.
- The MenuManager holds all MenuItems, even those not currently in the menu.
- Ingredients are associated with Stock, and they have a relationship of one to many. The Recipe class also has an association with Ingredient because the Recipe contains a dictionary mapping Ingredients in the Recipe to their respective quantities. Each Ingredient may not be part of a Recipe and each Recipe may not have any Ingredients.
- In the previous phases we had a History class to store past orders. We initially thought this would give our system more modularity, as the user would be able to import and export the history data separately. However, in that design, all orders needed to be linked to the Transactions in the Finance class, which increased both the complexity and the risk of errors while importing finance and history data separately. To solve this, we decided to store the orders with the finance information.
- Persistence was achieved by using JAXB, which is an XML parser that converts instances of classes to and from XML files. This is explained in more detail later in the document.

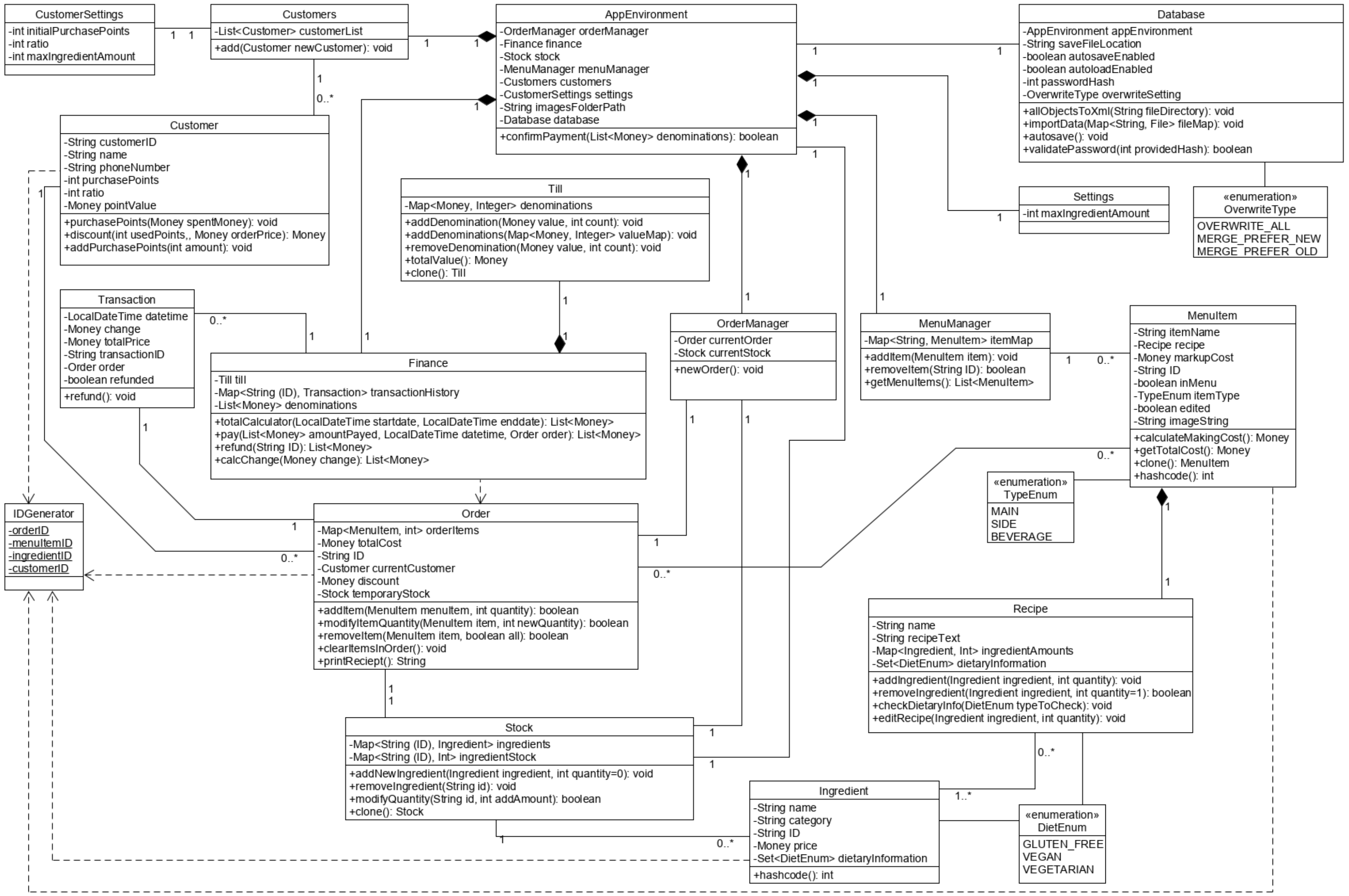

CustomerSettings
-int initialPurchasePoints
-int ratio
-int maxIngredientAmount
Customers
-List<Customer> customerList
+add(Customer newCustomer): void
AppEnvironment
-OrderManager orderManager
-Finance finance
-Stock stock
-MenuManager menuManager
-Customers customers
-CustomerSettings settings
-String imagesFolderPath
-Database database
+confirmPayment(List<Money> denominations): boolean
Database
-AppEnvironment appEnvironment
-String saveFileLocation
-boolean autosaveEnabled
-boolean autoloadEnabled
-int passwordHash
-OverwriteType overwriteSetting
+allObjectsToXml(String fileDirectory): void
+importData(Map<String, File> fileMap): void
+autosave(): void
+validatePassword(int providedHash): boolean
Settings
-int maxIngredientAmount
«enumeration»
OverwriteType
OVERWRITE_ALL
MERGE_PREFER_NEW
MERGE_PREFER_OLD
Customer
-String customerID
-String name
-String phoneNumber
-int purchasePoints
-int ratio
-Money pointValue
+purchasePoints(Money spentMoney): void
+discount(int usedPoints,, Money orderPrice): Money
+addPurchasePoints(int amount): void
Till
-Map<Money, Integer> denominations
+addDenomination(Money value, int count): void
+addDenominations(Map<Money, Integer> valueMap): void
+removeDenomination(Money value, int count): void
+totalValue(): Money
+clone(): Till
Transaction
-LocalDateTime datetime
-Money change
-Money totalPrice
-String transactionID
-Order order
-boolean refunded
+refund(): void
Finance
-Till till
-Map<String (ID), Transaction> transactionHistory
-List<Money> denominations
+totalCalculator(LocalDateTime startdate, LocalDateTime enddate): List<Money>
+pay(List<Money> amountPayed, LocalDateTime datetime, Order order): List<Money>
+refund(String ID): List<Money>
+calcChange(Money change): List<Money>
OrderManager
-Order currentOrder
-Stock currentStock
+newOrder(): void
MenuManager
-Map<String, MenuItem> itemMap
+addItem(MenuItem item): void
+removeItem(String ID): boolean
+getMenuItems(): List<MenuItem>
MenuItem
-String itemName
-Recipe recipe
-Money markupCost
-String ID
-boolean inMenu
-TypeEnum itemType
-boolean edited
-String imageString
+calculateMakingCost(): Money
+getTotalCost(): Money
+clone(): MenuItem
+hashcode(): int
«enumeration»
TypeEnum
MAIN
SIDE
BEVERAGE
IDGenerator
-orderID
-menuItemID
-ingredientID
-customerID
Order
-Map<MenuItem, int> orderItems
-Money totalCost
-String ID
-Customer currentCustomer
-Money discount
-Stock temporaryStock
+addItem(MenuItem menuItem, int quantity): boolean
+modifyItemQuantity(MenuItem item, int newQuantity): boolean
+removeItem(MenuItem item, boolean all): boolean
+clearItemsInOrder(): void
+printReciept(): String
Recipe
-String name
-String recipeText
-Map<Ingredient, Int> ingredientAmounts
-Set<DietEnum> dietaryInformation
+addIngredient(Ingredient ingredient, int quantity): void
+removeIngredient(Ingredient ingredient, int quantity=1): boolean
+checkDietaryInfo(DietEnum typeToCheck): void
+editRecipe(Ingredient ingredient, int quantity): void
Stock
-Map<String (ID), Ingredient> ingredients
-Map<String (ID), Int> ingredientStock
+addNewIngredient(Ingredient ingredient, int quantity=0): void
+removeIngredient(String id): void
+modifyQuantity(String id, int addAmount): boolean
+clone(): Stock
Ingredient
-String name
-String category
-String ID
-Money price
-Set<DietEnum> dietaryInformation
+hashcode(): int
«enumeration»
DietEnum
GLUTEN_FREE
VEGAN
VEGETARIAN
1
0..*
1..*

Figure 5.2: UML class diagram.

5.3 UML Package Diagram

5.3.1 Deliverable Two

The UML package diagram from deliverable two, as seen in 5.3, was created with the class diagram in 5.2. The structure of the system was made such that each class commonly used within the application fell within four sections, including the GUI package, the AppEnvironment (a class with its own section), the logic package and the information package. This tiered system allows any class in a package to access classes within its package, and any classes in lower tiered packages. This allowed us to keep the structure of the program consistent, and to prevent us from increasing cohesion beyond an acceptable level.

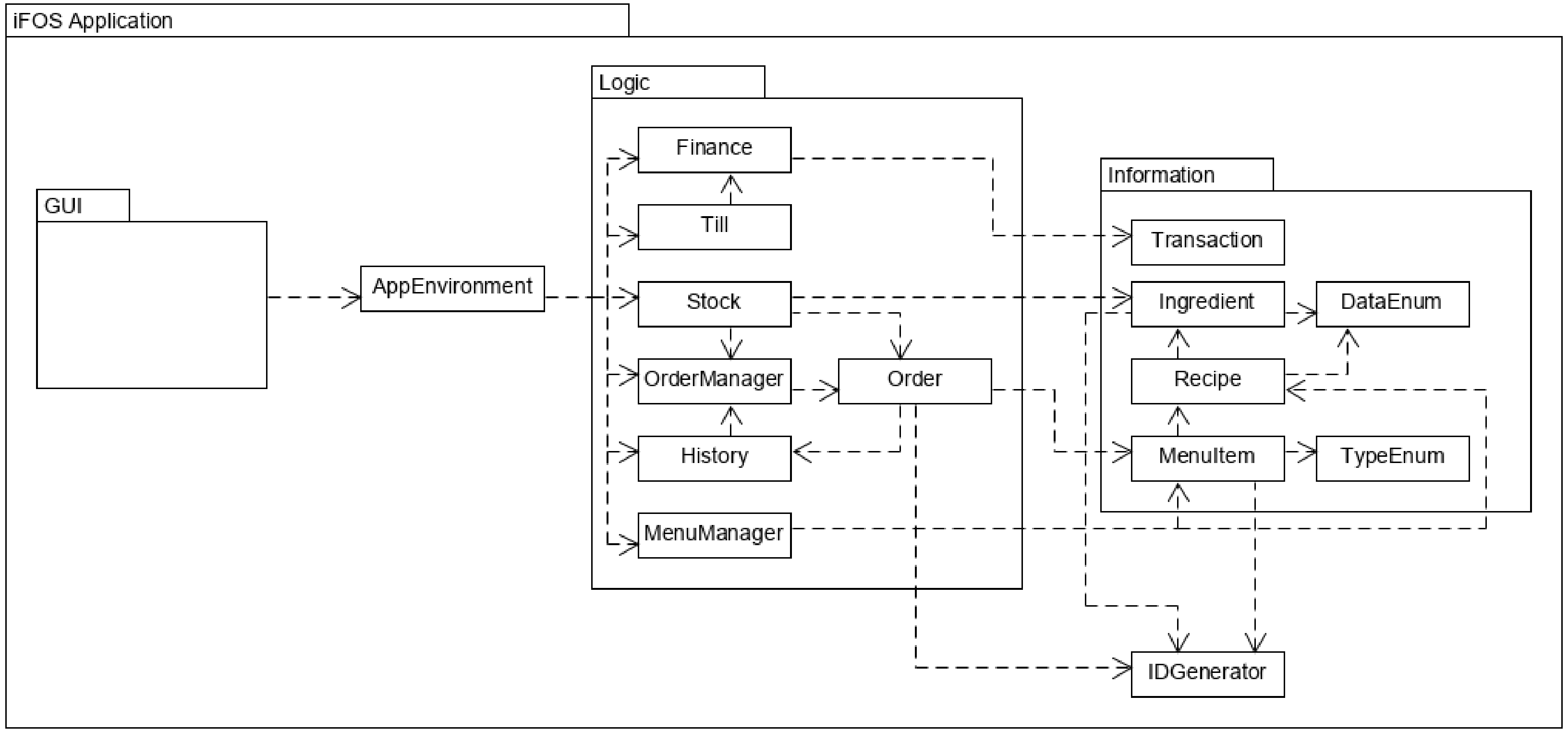


Figure 5.3: The old UML package diagram from deliverable two.

5.3.2 Deliverable Three

Like the diagram from Figure 5.3, the database is omitted as it creates all other objects using the .xml files. Notable changes since the previous iteration of the diagram is the diagram is simplified, whilst the hierarchical design was maintained as much as possible. Almost all of the controllers shown in the below figures extend from a GeneralController parent class, which allows for efficient screen changing and access to the AppEnvironment, which is passed between each GeneralController.

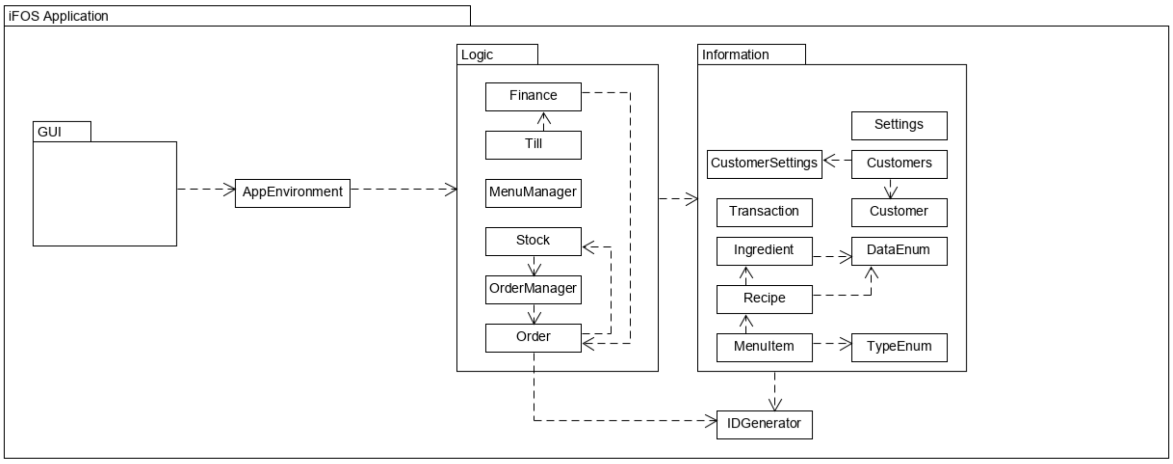


Figure 5.4: The new UML package diagram from deliverable three.

Admin Controller

The Admin controller has multiple functions due to the the use of tab panes splitting features. It contains finance information, till management, menu customisation, and data importing/exporting settings. Furthermore a password controller is used to handle a users input from a number pad to access the admin screen. AddRecipeController, which is used to create menu item recipes, starts the AddExtraIngredient-Controller. The AddExtraIngredientController is reused in multiple instances, namely when modifying an existing item in the order, when creating a new item, and when adding a past order to the history screen.

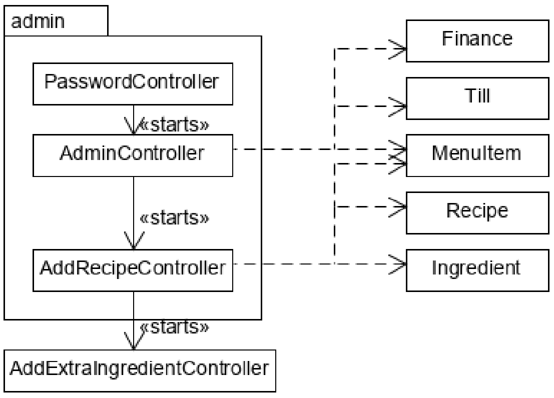


Figure 5.5: Admin package inside of the GUI package and it's relation to other classes.

History Controller

The History package handles the display of the history screen, which shows data from Transaction and Order objects. Notably, the AddPastOrderController extends off of OrderController allowing reuse of all of the components from the Order screen, whilst overriding and adding new functionalities such as selecting a time of when the order was completed.

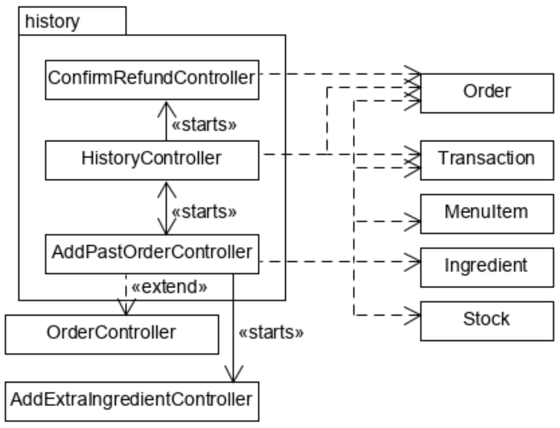


Figure 5.6: History package inside of the GUI package and it's relation to other classes.

Invoice Controller
The InvoiceController handles inputs from the Invoice screen and passes information to complete a payment. Furthermore, Controllers used for the loyalty system are called through this controller to handle new and existing members and discounts.

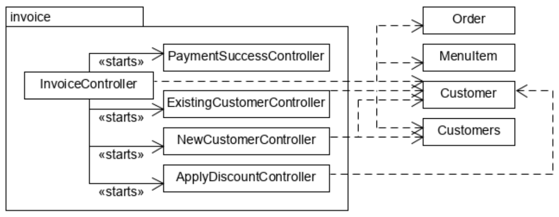


Figure 5.7: Invoice package inside of the GUI package and it's relation to other classes.

Order Controller
The OrderController is the most used throughout the application, it handles all of the food-realted classes and notably passes MenuItem and Ingredient objects. Orders are created here, and items can be added or removed. Menu items can also be filtered by specific tags.

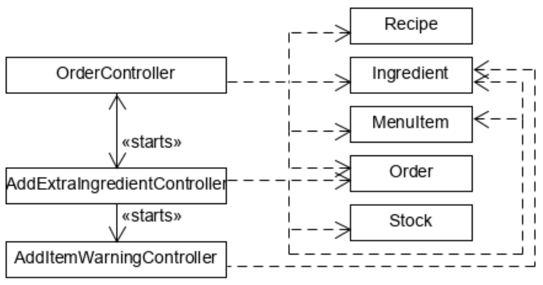


Figure 5.8: Order related controllers inside of the GUI package and it's relation to other classes.

Stock Controller

The StockController is not linked to many classes but still handles stock and passes updated Stock levels when modifying ingredients. A design decision about removing items from an already existing MenuItem is explored in Section 6.2.3.

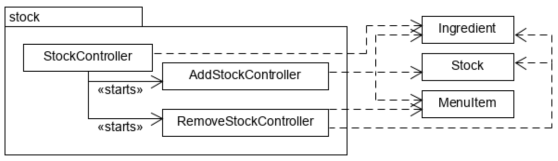


Figure 5.9: Stock package inside of the GUI package and it's relation to other classes.

Chapter 6

Description of Current Product Version

6.1 Deliverable Two

6.1.1 Requirements and Features Implemented

All functional requirements stated in 2.2 were implemented, except for:

- Displaying sales data (FR16)
- Warnings when stock is low
- Visual representation of data (FR19)

This was due to time constraints and the low priority of these functional requirements. The features which are to be implemented that were stated in previous sections can see seen below in Section 6.1.3. The features which are planned to be implemented can be found in Section 9.2.

6.1.2 Notable Features Implemented

JAXB
JAXB was used as a parser to convert .xml files to objects used in the application. This includes four files, storing the stock, menu, finance information, and order history information. It was chosen over the other parsers DOM and SAX for it's ability to convert objects into .xml and .xml to objects easily when the user wants to import or export the system's data.

Validation for .xml files was required to ensure that the data imported is correct. Each of the .xml files are validated against their own schema, for example stock.xml is validated against stock.xsd. These validations ensures that the .xml files import are correct and follow the correct format to be imported.

Touch-Screen Oriented
The invoice screen was made to be navigated via touch. Therefore large buttons and text were included to aid in the users experience when using the system. Furthermore no keyboard is required when entering denominations into the invoice screen as the cash is predefinied as buttons.

Recipe Creation and Modification
Recipes can be created and modified by the user. This includes the ability to change the ingredients required, changing the name of a menu item and changing the markup cost.

Adding Extra Ingredients
The ability to add extra ingredients is used by multiple controllers and classes, thus the controller was made to be modular. This controller is called by AddRecipeController, AddPastOrderController and OrderController. each requiring their own separate inputs. But all requiring a stock and an item to be modified.

6.1.3 Requirements and Features not Implemented

More Touch-Screen Friendly Design
Some other screens require more design efforts in making the interface more user-friendly. For example spinners would be extremely hard to press in a touch-screen environment. This will be implemented in the third deliverable.

Business Expenses and Profit
Past transactions and customer orders can be viewed as stated in FR15. The total cost of orders from a specific date along with the average total money gained per day. Although business expenses and profits are not currently being calculated by the system.

Limiting User Inputs
Like the Stock screen, user inputs should be limited to certain characters, for example a user cannot type a negative number as an input. For example, should apply when a user is entering recipes.

Security QR5
A screen prompting the user to enter verification should be implemented to secure data and to prevent other users from tampering with the data contained in the system. Furthermore the system should encrypt the .xml files so they cannot be accessed without using the application.

6.2 Deliverable Three

6.2.1 Requirements and Features Implemented

Loyalty Program
A loyalty system was implemented and is accessed through the invoice screen. Members can be created, searched or deleted from a list. Members gain a certain amount of points through spending money, the amount they gain and start with can be set in the settings in the admin screen. Members can use their points which are worth money, to get a discount off of their current order.

Settings
A settings screen was added to the admin screen to be able to change the stored password, along with the ratio of money to points and initial points when a new member is added to the above loyalty program.

Refunding of Orders (FR6)
Due to a critical bug, this was taken out of deliverable two, but has now been fixed. An order can be successfully removed, and the till will calculate and remove the change required to refund. Furthermore if there is not enough money in the till then the refund operation is cancelled.

Displaying sales data (FR16) & Calculation of profits (FR17)
The sales data can be displayed to the user in the admin screen under finance, the data can be shown over a range of dates which the user can select. The information which is displayed includes:

- The total cost of creating orders
- The average total cost
- Total profits
- Average profits

Furthermore, a table showing total profits is included, this is discussed in the section below.

Visual Representation of Data (FR19)
A JavaFX chart was used to display profits from each day over a range of dates. This graph is refreshed when a user selects and confirms new dates to view the data over.

Touch-Screen Friendly Design
With a lot more emphasis being placed on the design and layout of the graphical interface, many inputs were changed to be easier to press. For example spinners when changing quantities of ingredients were changed to have the arrows on opposite sides, to prevent accidentally selecting the wrong quantity. Text was also made larger and bolder. Some screens went through noticeable layout changes, for example the invoice screen's number pad was shifted to the right, so the users arm would not block the display. Furthermore the denomination information was made much larger and easier to read.

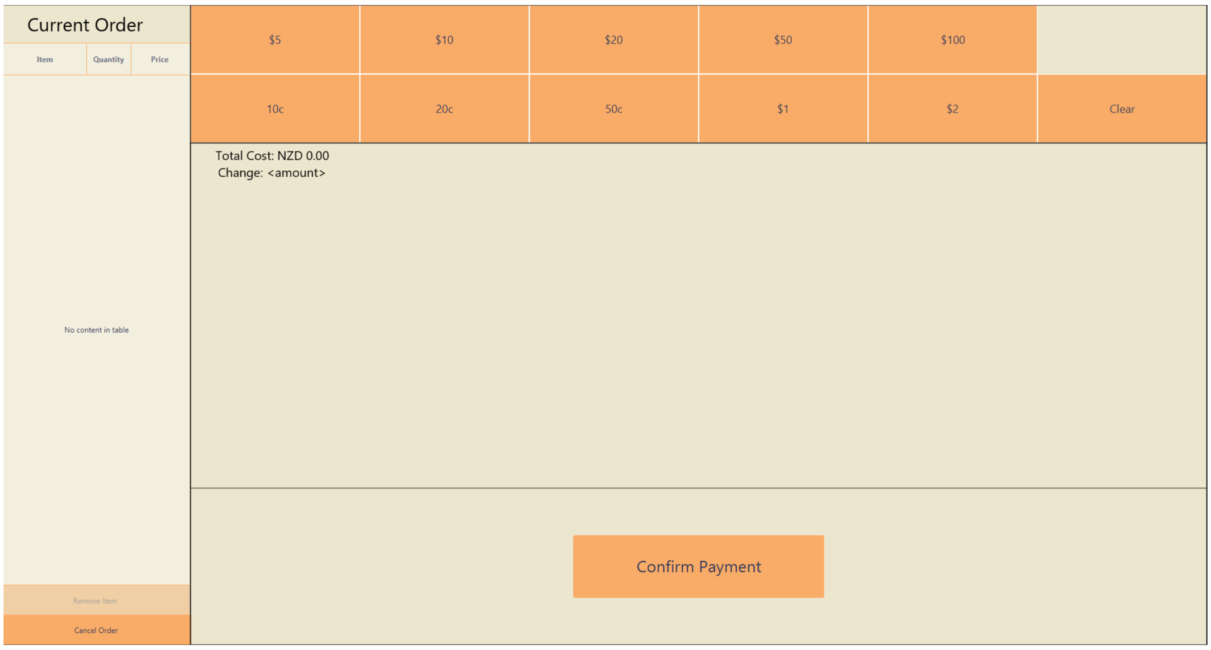


Figure 6.1: Screenshot of the old invoice screen from deliverable two.

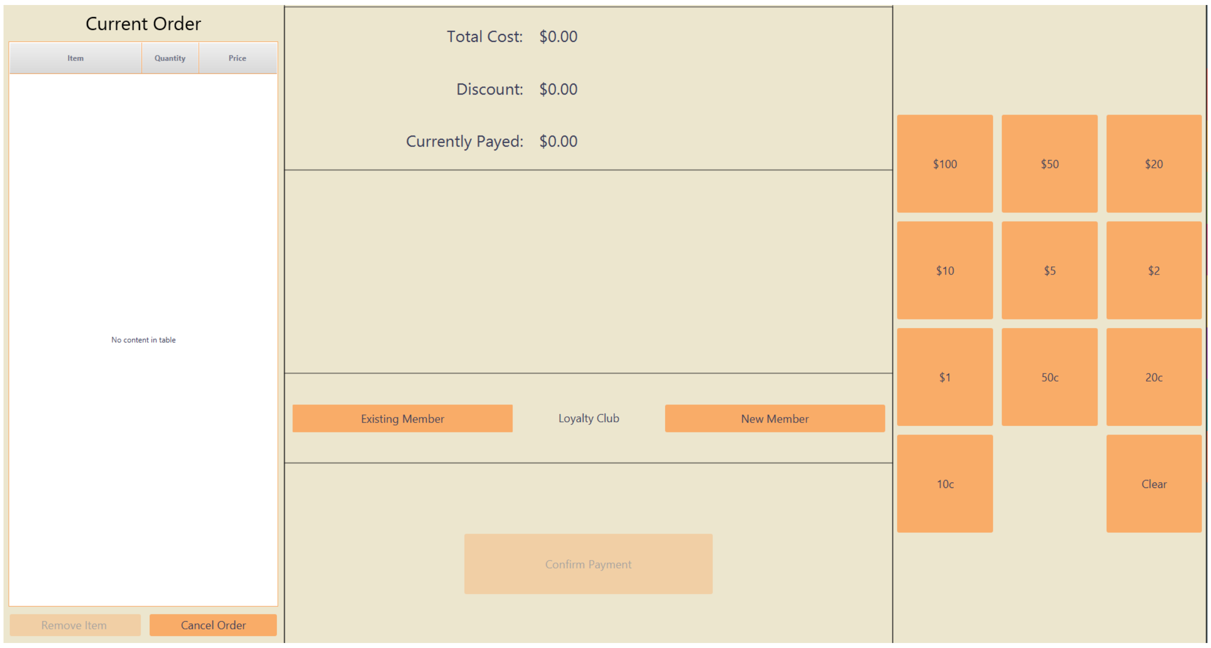


Figure 6.2: Screenshot of the new invoice screen from deliverable three.

The Order screen also went through an overhaul, the table of ingredients was moved to the left hand side to make information visible on one side so the user does not have to move their head as often. Furthermore adding items was changed from a selection and modify based implementation, to a system where items are added immediately when their corresponding buttons are pressed, there are a few notable upsides to this implementation:

- There are a lot less buttons which need to be pressed to add an item to the order
- Less buttons can be displayed on the screen which makes the layout easier to navigate and makes the design look cleaner.

Although there were a few negatives such as:

- Adding multiple items has an unnoticeable, but larger delay. (Tested in 7.1.2)

- Items need to be modified individually, if the user were to have an order with multiple items which have the same changes, it would take much longer to enter the order.

Furthermore the filters for items was moved to the top of the screen to be easily seen.

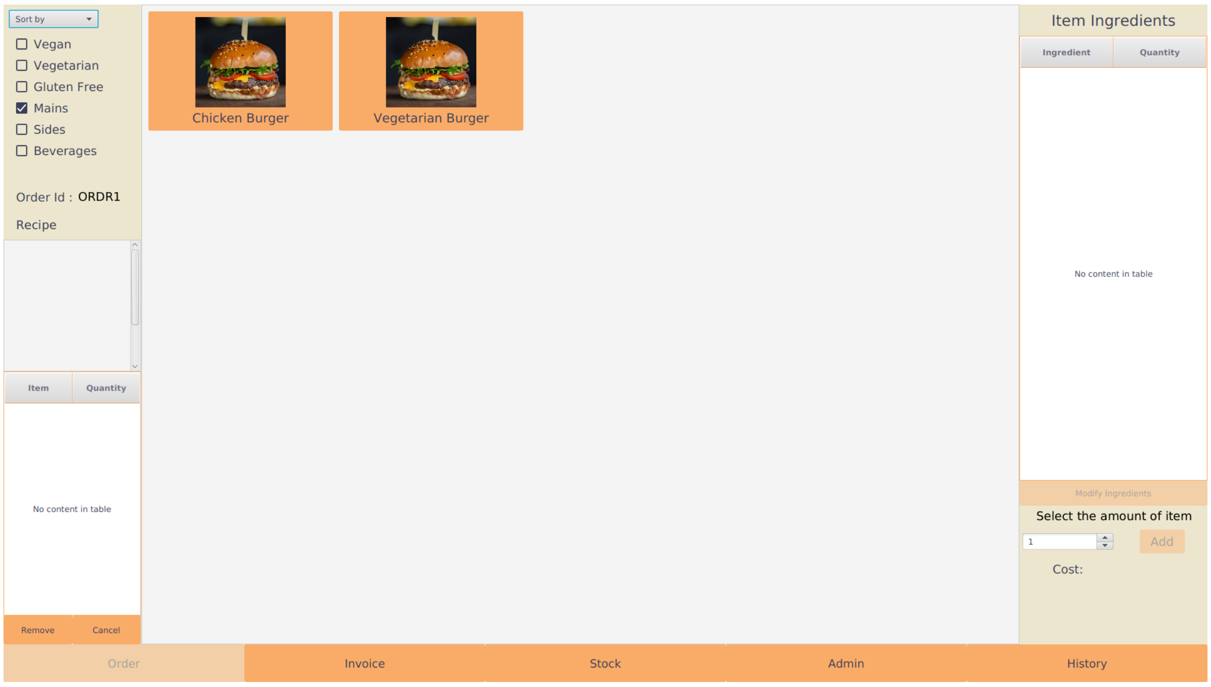


Figure 6.3: Screenshot of the old order screen from deliverable two.

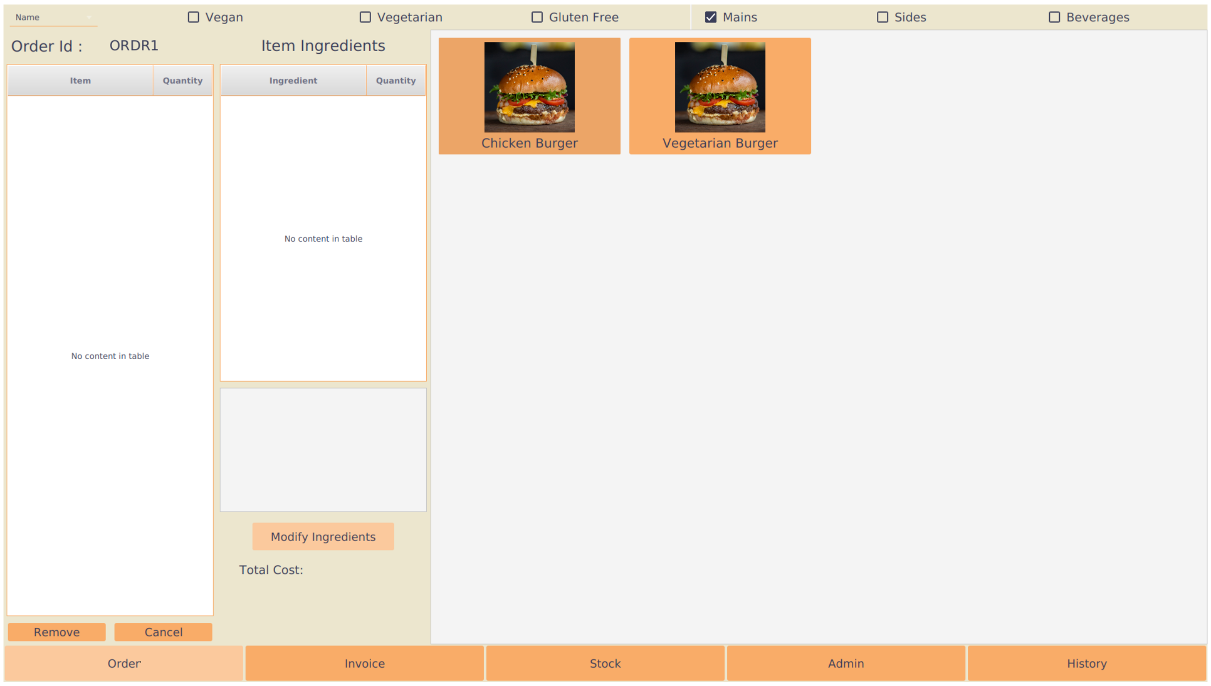


Figure 6.4: Screenshot of the new order screen from deliverable three.

Limited User Inputs

Many more listeners were added to inputs to prevent users from inputting erroneous data, these can be seen when adding quantities and IDs when searching. This made testing easier and more streamlined as users now cannot enter erroneous data. Furthermore, many buttons were made to be disabled, for example if an order is in progress, the user is limited to certain actions, such as being unable to modify stock. Also, buttons such as modifying ingredients are disabled, if there is no item selected.

Security: Admin Screen Verification (QR5)
A number pad was added to create verification for accessing the admin screen. This prevents unauthorised users from tampering and viewing sensitive data.

Dietary Requirement Warnings
A warning prompt now asks the user for confirmation, if the item they are adding breaks any existing dietary requirements of the item they are modifying, for example, adding meat to a vegetarian item.

Removing Item Warnings
A prompt asks the user for confirmation if they are sure they want to remove an item from the stock. There was discussion regarding if removing the ingredient should cause a ”cascading” delete throughout the items (which is the current implementation), if items containing the ingredient should be deleted or if the user has to delete the items containing the ingredient first, before deleting the ingredient. The latter was suggested but rejected as that would create a lot of work for the user, and the user would also have to have access the admin screen to be able to delete the items.

6.2.2 Requirements and Features not Implemented

Examples of requirements which were considered during implementation of the third deliverable, but were discarded due to time constraints are as follows:

- Sorting items by Price (FR15)
- Removing a single count of an item from an order
- Viewing the price of an item on the button on the order screen
- .xml encryption
- Variable cooking times in a recipe when ingredients get added or removed
- Tracking and storing data about suppliers
- Having names of edited items change to show ingredients which have been added and removed

Adding suppliers to the system was considered but were rejected as the design of the system was based on being able to manage and track orders. With changing requirements and stakeholders, suppliers were less considered as stock is managed by the owner. Furthermore the ability to track suppliers was given a very low priority during the design of the system, thus was never considered as a requirement. Although given time, features related to this and the other features would be considered and prioritised to be added to the application in the future.

6.2.3 Other Notes

Creating an Item With No Ingredients
The ability to create items with no ingredients was added as the team believed that being able to make a dummy item for exchanging cash may be required if a customer only wanted to swap denominations with the food stall.

Chapter 7

Testing Procedures

A combination of JUnit and Gherkin was used to test the code. These tests were written before code was implemented as stated in PP2 & PP5 in chapter 10 to follow a Test Driven Development process. This allowed the team to have an expectation of how the code should behave, and made implementation of code much simpler. All modules inside of the information and logic packages were considered for testing.

Bugs

Majority of bugs encountered were due to original tests being too lenient. Furthermore, not enough thought was placed into how other modules will interact with each other. Thus a lot of errors occurred when data was being passed between objects. This was further shown when data in the graphical user interface was not updating and displaying data properly.

7.1 Manual Testing

Manual tests was used used to test the graphical user interface (GUI). Deliberate erroneous user data was input when creating recipe names, inputting recipe cost and importing the wrong named .xml files. Furthermore an emphasis was placed on limiting user inputs, all type-able inputs except for entering dates have listeners which prevent users from entering erroneous data, for example id, price and quantity entries are limited to positive integers. Other examples of erroneous data included negative prices and quantities, alphanumeric inputs for data which is expected to be digits, no input at all, as well as edge cases such as zero, and extremely large numbers, which are now prevented by listeners.

7.1.1 Functional Testing

Functional tests were done to tests whether a use case or functionality is working through using the GUI application, this reduce the likelihood that bugs will appear in the system when in use.

Functional tests were tested with the same data loaded from pre-made .xml files included with the application which are automatically pre-loaded to the system. To make sure that the data is pre-loaded into the system please launch the application and click "Import/Export Data" at the top and in the "Autosave and Autoload" box, make sure "Autoload" checkbox is checked and set the autosave/load location to where the provided .xml files are stored and also click "Select Images Folder" and select the images folder. Then relaunch the application.

(FT1)
System Under Test: Orders
Test: Add menu item to order
Description:

1. Launch application
2. Navigate to "Order" screen
3. Click Vegetarian Burger
4. Click Chicken Burger

Result: The total cost for the two items in the order is displayed as ”NZD 12.40” and the menu items are in the order table
Error Checking:

1. Not enough stock to add the menu item stops the item being added to order table

Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC1 ”Create Order”, FR3

(FT2)
System Under Test: Payment
Test: Paying for an item
Description:

1. Follow FT1 description
2. Navigate to ”Invoice” screen
3. Pay more than the total cost of the order with $20
4. Confirm payment without being a member

Result: The required change for the order is output in a separate screen, with 1x $5.00, 1x $2.00 and 1x $0.50 and 1x $0.10 as change.
Error Checking:

1. Not enough money in the till is available to be returned.

Pass/Fail: Pass
Tester: James
Use case/Requirement: UC1 ”Create Order”, FR9

(FT3)
System Under Test: Orders
Test: Edit a menu item that is in the order
Description:

1. Follow FT1 description
2. Click the ”Vegetarian Burger” in order table
3. Click Modify Ingredients
4. Increase the selected value of ”Chicken Patty” to 1
5. Click confirm
6. The system displays a warning regarding the change of the item’s dietary information
7. Click confirm
8. Click confirm

Result: The total cost of the order is updated to ”NZD 13.00” and the vegetarian burger is now named edited vegetarian burger, a Chicken Patty ingredient is now in the item ingredients table
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC1 ”Create Order”, FR14

(FT4)
System Under Test: Orders
Test: Cancel Order
Description:

1. Follow FT1 description
2. Click ”Cancel Order”

Result: There are no menu items in the order table, and there are no item ingredients displayed and no recipe text displayed
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC1 ”Create Order”, FR3

(FT5)
System Under Test: Order History
Test: Refunding a past order
Description:

1. Launch Application
2. Click ”History” at the bottom
3. Locate the order with ID ORDR0
4. Click the ”Refund” that is in the same row
5. System displays refund amounts
6. Click ”Yes”
7. Click ”Continue”

Result: The refund button for the order is now disabled
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC2 ”Refund Past Order”, FR6

(FT6)
System Under Test: Stock
Test: Adding a new ingredient to the stock with valid values
Description:

1. Launch Application
2. Click ”Stock” at the bottom
3. Click ”Add” at top left corner
4. Enter into Ingredient Name: Apple (200grams)
5. Enter into Category: fruit
6. Enter into Cost (NZD): 0.50
7. Enter into Quantity: 30
8. Tick ”Vegan”
9. Tick ”Vegetarian”
10. Tick ”Gluten Free”
11. Click ”Create”

Result: The ingredient ”Apple (200grams)” is created and displayed with the correct values in the stock table
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC4 ”Modify Stock”, FR7

(FT7)
System Under Test: Stock
Test: Adding a new ingredient to the stock with invalid values
Description:

1. Launch Application
2. Click ”Stock” at the bottom

3. Click "Add" at top left corner
4. Enter into Cost (NZD): qwerty
5. Enter into Quantity: qwerty

Result: The system doesn't allow the invalid values to be inputted Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC4 "Modify Stock", FR7

(FT8)
System Under Test: Stock
Test: Adding a new ingredient to the stock with no values
Description:

1. Launch Application
2. Click "Stock" at the bottom
3. Click "Add" at top left corner
4. Click "Create"

Result: The system doesn't allow the ingredient to be created Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC4 "Modify Stock", FR7

(FT9)
System Under Test: Stock
Test: Adding a new ingredient to the stock with no values
Description:

1. Launch Application
2. Click "Stock" at the bottom
3. Click "Add" at top left corner
4. Click "Create"

Result: The system doesn't allow the ingredient to be created Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC4 "Modify Stock", FR7

(FT10)
System Under Test: Import/Export Data
Test: Importing all the xml files
Description:

1. Launch Application
2. Click "Import/Export Data" at the top
3. Untick "Autoload" in bottom left box
4. Close Application
5. Launch Application
6. Click "Menu Item Manager" at the top
7. Check that table displays "No content in table"
8. Click "Import/Export Data" at the top
9. Click "Select stock.xml"
10. Locate and select "stock.xml" and click "Open"
11. Click "Select menu.xml"
12. Locate and select "menu.xml" and click "Open"
13. Click "Select finance.xml"

14. Locate and select ”finance.xml” and click ”Open”
15. Click ”Select customers.xml”
16. Locate and select ”customers.xml” and click ”Open”
17. Click ”Select settings.xml”
18. Locate and select ”settings.xml” and click ”Open”
19. Click ”Import Data”
20. Click ”Menu Item Manager” at the top

Results: There are menu items displayed in the table Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC6 ”Import Data”, FR10

(FT11)
System Under Test: Import/Export Data
Test: Exporting all data into xml files
Description:

1. Launch Application
2. Click ”Import/Export Data” at the top
3. Click ”Export Data”
4. Choose a directory for the files to store in

Result: The files stock.xml, menu.xml, fiance.xml and customers.xml are in the chosen directory Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC7 ”Export Data”, FR11

(FT12)
System Under Test: Stock
Test: Adding a new menu item to the menu with valid values
Description:

1. Launch Application
2. Click ”Menu Item Manager” at the top
3. Click ”Add” at top
4. Enter into Menu Item Name: Potato Fritters
5. Select Menu Type: Side
6. Enter into Markup cost (NZD): 2.50
7. Click ”Modify Recipe Content”
8. Locate Potato (200grams) and change selected to 3
9. Click ”Save”

Result: The menu item ”Potato Fritters” is created and displayed with the correct values in the menu item recipes table with selling price ”NZD 2.80”
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC8 ”Modify Menu Item”, FR13, FR18

(FT13)
System Under Test: Finance sales data
Test: Viewing finance sales data within a time period
Description:

1. Launch Application
2. Click ”Finance” at the top

3. Select start data as 24/09/2019
4. Select end data as 25/09/2019
5. Click ”Get finance information”

Result: The graph updates and shows a line going down and the following values are displayed, Total cost of orders: NZD 11.50, Average daily cost: NZD 5.75, Total profits: NZD 8.70, and Average daily profits: NZD 4.35. Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC9 ”View Sales Data”, FR16

(FT14)
System Under Test:Till
Test: Changing the amount in the Till
Description:

1. Launch Application
2. Click ”Till Manager” at the top
3. Change the quantities in the Till
4. Click ”Stock” at the bottom
5. Click ”Admin” at the bottom
6. Enter password by clicking 1 1 1 1
7. Click ”Till Manager” at the top

Result: The changes in quantity for the money denomination was updated and saved
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC10 ”Modify Till”, FR20

(FT15)
System Under Test: Loyalty Customer
Test: Adding a new customer to the loyalty customer data
Description:

1. Launch Application
2. Click ”Invoice” at the bottom
3. Click ”New Member”
4. Enter Name: Bob Ross
5. Enter Phone Number: 0212345678
6. Click ”Confirm”
7. Click ”Clear Selected Member”
8. Click ”Existing Member”

Result: The customer ”Bob Ross” with phone number ”0212345678” is now in the existing member table
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC11 ”Add Customer”, FR21

(FT16)
System Under Test: Loyalty Customer
Test: Searching for an existing member of the Loyalty System Description:

1. Launch Application
2. Click ”Invoice” at the bottom
3. Click ”Existing Member”
4. Enter into ”Search Name”: Bob Ross

5. Enter into ”Search Phone Number”: 0212345678

Result: The only showing member is a customer with the name ”Bob Ross” and a phone number of ”0212345678”.
Pass/Fail: Pass
Tester: Michael
Use case/Requirement: UC11 ”Add Customer”, FR21

(FT17)
System Under Test: Loyalty Customer
Test: Removing an existing customer from the loyalty customer data
Description:

1. Launch Application
2. Click ”Invoice” at the bottom
3. Click ”Existing Member”
4. Enter Name: Bob Ross
5. Enter Phone Number: 0212345678
6. Right click on the only showing member.
7. Click ”Delete Member”

Result: The customer ”Bob Ross” with phone number ”0212345678” has been removed from the table and thus from the system.
Pass/Fail: Pass
Tester: Michael
Use case/Requirement: UC11 ”Modify Members”, FR21

(FT18)
System Under Test: Password Settings
Test: Change the password
Description:

1. Launch Application
2. Click ”Settings” at the top
3. Enter Old Password: 1111
4. Enter New Password: 1234
5. Enter Confirm Password: 1234
6. Click ”Confirm & Save”
7. Click ”Stock” at the bottom
8. Click ”Admin” at the bottom
9. Enter the password by clicking 1 1 1 1
10. Enter the password by clicking 1 2 3 4

Result: The password successfully changed and can access ”Admin” screen with 1234 and can’t access screen with 1111
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC12 ”Modify Password”, FR22

(FT19)
System Under Test: Payment
Test: Paying for an item and applying a loyalty member discount
Description:

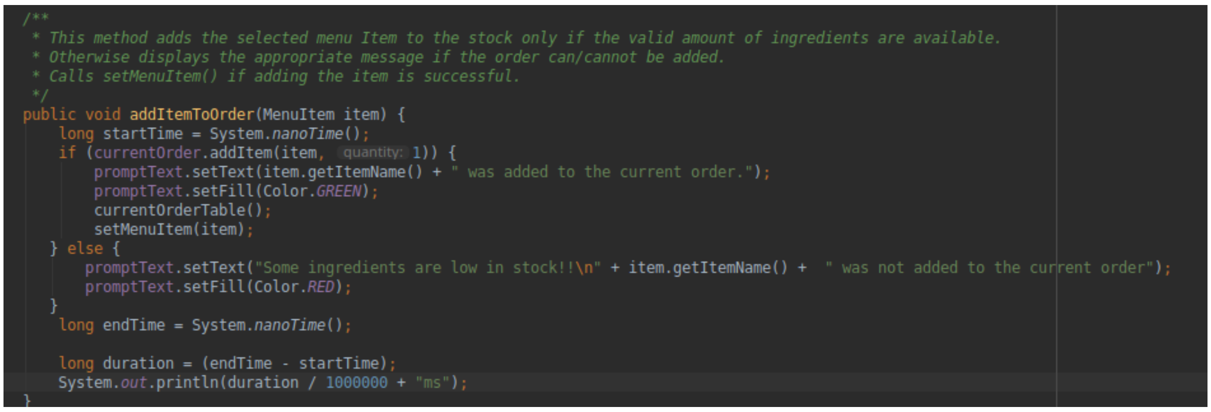
1. Follow FT1 description

2. Navigate to ”Invoice” screen
3. Click ”Existing Member”
4. Select the row with ID ”CUST0”
5. Click ”Select”
6. Click ”Apply Discount”
7. Click ”All”
8. Click ”Apply”
9. Click ”$10”
10. Click ”Comfirm Payment”

Result: The payment completes and the discount is applied, displays that the change is $0.10: 1, $2.00: 1, $0.50: 1
Pass/Fail: Pass
Tester: Yu
Use case/Requirement: UC1 ”Create Order”, FR21

7.1.2 Quality Testing

Quality testing was done to ensure the product meets the quality requirements stated in Section 2.3.2. These tests were done as manual tests when the graphical user interface was completed, and throughout production of the application to ensure our requirements were on task, and furthermore once the application was compiled into a .jar file. Quality requirements that had time requirements such as QR2 and QR3 used system timers to test the speeds at which functions would complete at. This can be seen below in Figure 7.1.



```
/**
 * This method adds the selected menu Item to the stock only if the valid amount of ingredients are available.
 * Otherwise displays the appropriate message if the order can/cannot be added.
 * Calls setMenuItem() if adding the item is successful.
 */
public void addItemToOrder(MenuItem item) {
    long startTime = System.nanoTime();
    if (currentOrder.addItem(item,  quantity: 1)) {
        promptText.setText(item.getItemName() + " was added to the current order.");
        promptText.setFill(Color.GREEN);
        currentOrderTable();
        setMenuItem(item);
   } else {
       promptText.setText("Some ingredients are low in stock!!\n" + item.getItemName() +  " was not added to the current order");
       promptText.setFill(Color.RED);
   }
    long endTime = System.nanoTime();

    long duration = (endTime - startTime);
    System.out.println(duration / 1000000 + "ms");
}
```

Figure 7.1: Example of testing the completion time of functions.

Reliability QR1

Testing was difficult due to the team’s commitments as students to be able to test the system for 10 hours concurrently. Instead, the program was loaded with 50 of each item, with multiple edited recipes to be as large as possible. There was no noticeable lag when swapping between multiple screens, an example of the order used can be seen in E.1.

Usability QR2

Usability was tested by adding multiple more items to the order screen, then testing the time taken to find the item. This was easy due to the team being experienced with the system, as well as the items were able to be ordered alphabetically.

Efficiency QR3
When adding a single burger to the order, it took a range of 18ms to 58ms on the lab computers, and on a fresh compilation of the system, the first item took 25ms to be added, whilst consecutive items took 3ms on average. This lead a total 90ms to add 20 burgers, this does not take into account the user pressing the button, only the processing after the button is pressed. This test was also recreated on a home machine and took 150ms of processing time to add the burgers. It is noted that adding a new item to the order was slightly more delayed as an entry needed to be created in the allocated HashMap. Furthermore the original implementation of the order screen allowed the user to add multiple items at once, but was not tested it's time was not tested using lab computers, this implementation took 17ms to complete on the same home computer, so arguably the original implementation was faster, although reasoning for changing how the order screen functions is explored in Chapter 6.2.

Maintainability QR4
Updating the software is easy through git, although knowledge would be required for turning the said files into a .jar file. Furthermore this would require one of the developers sending the .jar file to the owner, thus it would be difficult to schedule updating the software. As a university project this would be hard to test due to the current limitations of the team's knowledge.

Security QR5
The addition of a keypad on the admin screen prevents others from tampering with data such as creating, modifying and removing menu items, changing the password, as well as viewing financial information. .xml encryption was considered, but was not able to be implemented due to time constraints, given more time the team would have considered adding this. This is further explored in Chapter 6.2.

Extensibility QR6
This requirement was difficult to test as the system has not been implemented in a professional environment, although ensuring that the program is modular and well documented allows for the developers to easily find and fix functions. Furthermore the layout of the system allows for more screens to be added across the navigation buttons along the bottom, so new features can be easily added to the system.

Compatibility QR7
The app was tested on multiple systems, specifically every developers laptop and multiple lab machines to ensure the application runs on multiple systems. Furthermore the application was made to be resizable to work with screens of any ratio/resolution, despite it working best in full screen.

7.2 Coverage

80% coverage was aimed for when creating tests. Furthermore, all use case blue sky, alternative and exceptional flows were considered in the tests. The graphical user interface was not covered in automatic tests as it was considered time consuming and better suited for manual testing. The updated coverage for the program for Deliverable Three can be seen in Figure 7.2 through to Figure 7.5. Compared to the coverage from Deliverable Two, the overall coverage has gone down due to the database not being automatically tested as thoroughly. It was easier to see the .xml files created and to manually test them rather than set them up automatically. Although new classes such as Customer were tested thoroughly to ensure that edge cases were considered.

7.3 Acceptance Testing

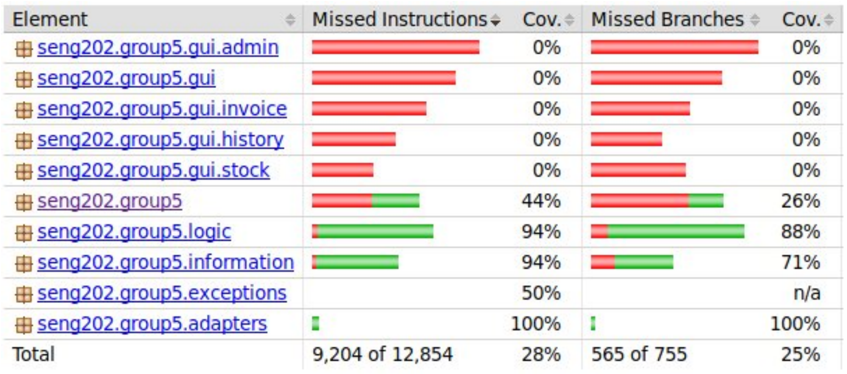
Cucumber tests specified in Table 3.1, were used throughout development of the project to ensure that the whole system runs as expected. As the team had implemented the tests early into deliverable two, a much clearer implementation of how each process should start and how each process should behave when running. Furthermore by defining the tests first and making them fail, it made verifying how the methods behaved together easier as the team was able to expect how the system should behave from a higher level.

7.3.1 JUnit

JUnit was used to ensure every module was performing as expected. This was to ensure that the consistently fits the tests we've made. By creating tests first, it covered each use case or functional requirement which allowed specific methods to be created. Most unit tests were made to be edge cases, although more edge cases could have been created in retrospect.

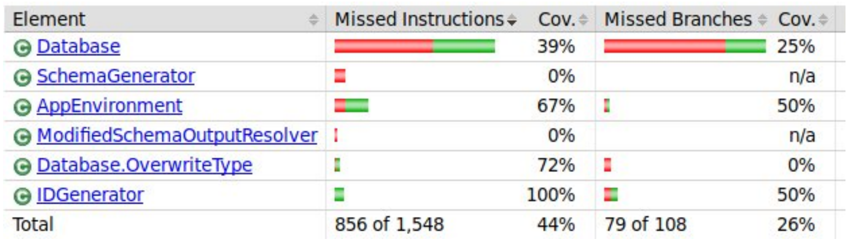
7.3.2 Continuous Integration

The acceptance and JUnit tests were specified to run when a build is pushed onto GitLab. Initially the team had problems with the pipeline failing as implemented tests were not ignored as mix of JUnit Vintage and Jupiter was being used and a combination of @Ignore and @Disabled was used. Once disabling all of the failing tests the team has had no further issues with the pipeline failing. Consideration was also placed on adding checkstyle to continuous integration but was not added due to time constraints and as the team was using a common code style exported from intelliJ.



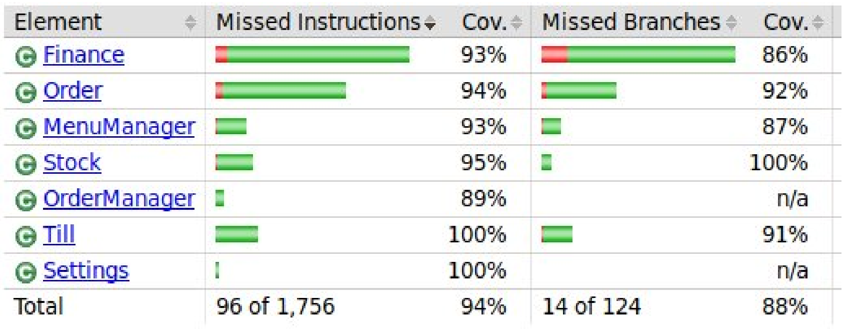
Element	Missed Instructions	Cov.	Missed Branches	Cov.
seng202.group5.gui.admin		0%		0%
seng202.group5.gui		0%		0%
seng202.group5.gui.invoice		0%		0%
seng202.group5.gui.history		0%		0%
seng202.group5.gui.stock		0%		0%
seng202.group5		44%		26%
seng202.group5.logic		94%		88%
seng202.group5.information		94%		71%
seng202.group5.exceptions		50%		n/a
seng202.group5.adapters		100%		100%
Total	9,204 of 12,854	28%	565 of 755	25%

Figure 7.2: Total coverage over all of the classes as of 14th October 2019.



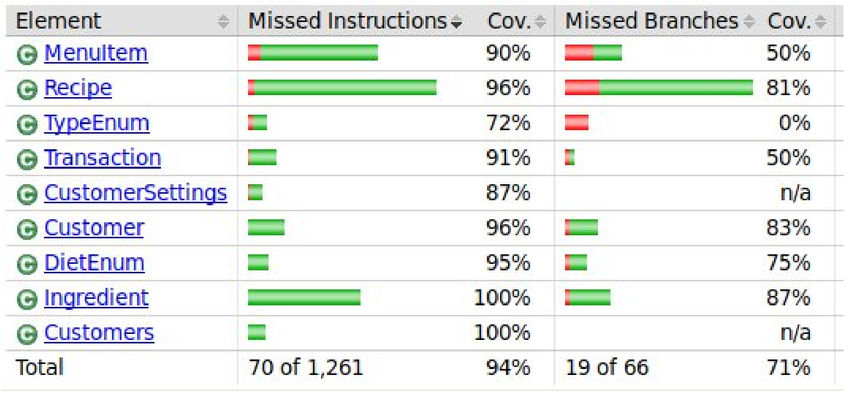
Element	Missed Instructions	Cov.	Missed Branches	Cov.
Database		39%		25%
SchemaGenerator		0%		n/a
AppEnvironment		67%		50%
ModifiedSchemaOutputResolver		0%		n/a
Database.OverwriteType		72%		0%
IDGenerator		100%		50%
Total	856 of 1,548	44%	79 of 108	26%

Figure 7.3: Coverage of all classes to do with XML marshalling and unmarshalling.



Element	Missed Instructions	Cov.	Missed Branches	Cov.
Finance		93%		86%
Order		94%		92%
MenuManager		93%		87%
Stock		95%		100%
OrderManager		89%		n/a
Till		100%		91%
Settings		100%		n/a
Total	96 of 1,756	94%	14 of 124	88%

Figure 7.4: Coverage of all logic classes.



Element	Missed Instructions	Cov.	Missed Branches	Cov.
MenuItem		90%		50%
Recipe		96%		81%
TypeEnum		72%		0%
Transaction		91%		50%
CustomerSettings		87%		n/a
Customer		96%		83%
DietEnum		95%		75%
Ingredient		100%		87%
Customers		100%		n/a
Total	70 of 1,261	94%	19 of 66	71%

Figure 7.5: Coverage of all information classes.

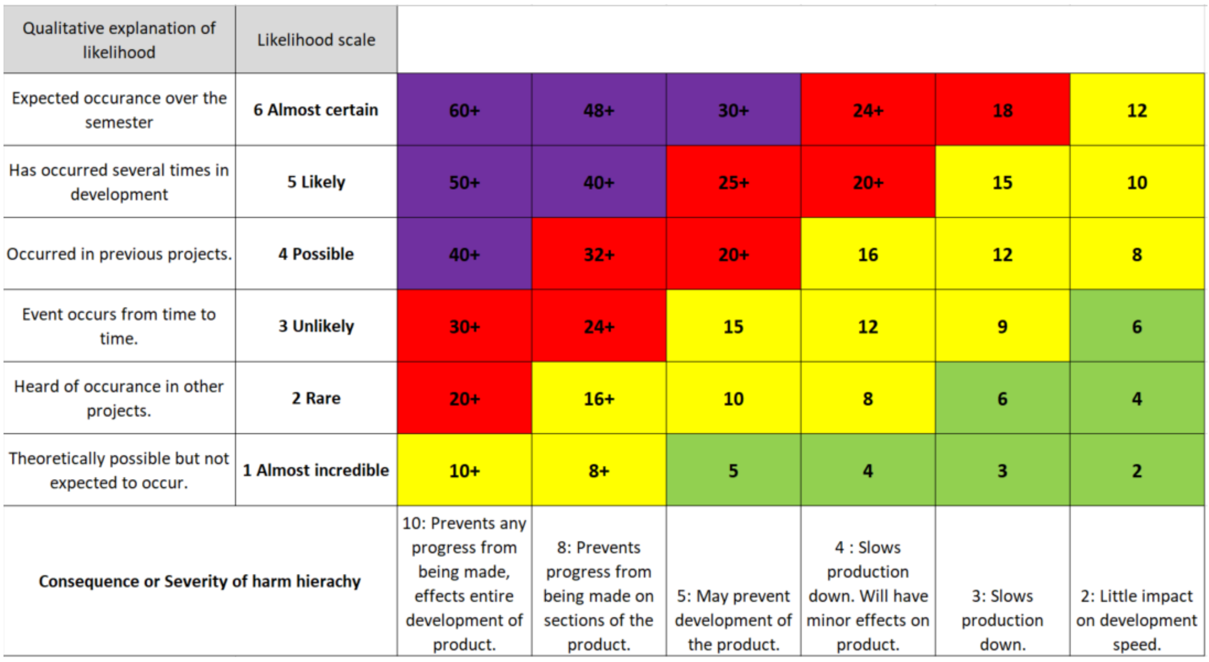
Chapter 8

Risk Assessment

A risk assessment table was used to categorise threats that may hinder the development of the product. These risks were brainstormed under categories such as weaknesses and threats, using a Strength Weakness Opportunity Threat (SWOT) analysis. By identifying these weaknesses, the team can plan to prevent and mitigate these risks.

8.1 Risk Matrix

The risk matrix (Figure 8.1) was used to quantify risks, by their likelihood and severity. The matrix was used in Table 8.1 to define a scale for the severity and likelihood for each risk, and furthermore to consider the associated level of threat.



Qualitative explanation of likelihood	Likelihood scale						
Expected occurance over the semester	**6 Almost certain**	**60+**	**48+**	**30+**	**24+**	**18**	**12**
Has occurred several times in development	**5 Likely**	**50+**	**40+**	**25+**	**20+**	**15**	**10**
Occurred in previous projects.	**4 Possible**	**40+**	**32+**	**20+**	**16**	**12**	**8**
Event occurs from time to time.	**3 Unlikely**	**30+**	**24+**	**15**	**12**	**9**	**6**
Heard of occurance in other projects.	**2 Rare**	**20+**	**16+**	**10**	**8**	**6**	**4**
Theoretically possible but not expected to occur.	**1 Almost incredible**	**10+**	**8+**	**5**	**4**	**3**	**2**
Consequence or Severity of harm hierachy		10: Prevents any progress from being made, effects entire development of product.	8: Prevents progress from being made on sections of the product.	5: May prevent development of the product.	4 : Slows production down. Will have minor effects on product.	3: Slows production down.	2: Little impact on development speed.

Figure 8.1: Risk matrix used to associate the criticality of a risk depending on the likelihood and consequence of the risk.

8.2 Risk Assessment Table

In Table 8.1, the majority of the responsibilities were associated with the development team, as most weaknesses and threats would be due to the team itself. It is noted that the majority of these risks are caused by poor communication and teamwork, thus there was a focus on keeping the whole team informed, by weekly milestones set as a team.

Table 8.1: Risk assessment table describes the responsibility, consequence and prevention of each risks.

ID	Description	Impact	Likelihood	Exposure	Responsibility	Consequences	Prevention
1	**Team-related**						
RA1.1	Conflict between teammates	5	6	30	Developer team	Lower motivation and morale for whole team. Less effort placed by some members of team.	Ensure conflicts are resolved between both parties. Which can be done by having a mediator and letting both sides explain their ideas.
RA1.2	Poor productivity (Procrastination)	7	9	63	Each individual member	Members portion of the project won't meet the set deadline.	Set consistent and frequent milestones to ensure everyone is up to date. Ensure everybody is motivated, by consistently encouraging teammates and having times at meetings talking about accomplishments will also act as a morale boost.
RA1.3	Member leaves project	10	2	20	Developer team	Less resources to work on project, harder to meet deadlines.	Unable to prevent. Can be mitigated by efficient planning and effectively splitting work evenly throughout remaining team.
RA1.4	Low motivation	5	5	25	Each individual member	Members portion of the project won't meet the set deadline.	Ensure members are encouraging each other, ensure member is taking adequate breaks.
RA1.5	Overconfidence	7	5	35	Developer team	Requirements may be set too wide, team plans to add more than they are able. Team does not meet deadline.	Ensure project plan is realistic and measurable. Consistently referencing the project plan and ensuring milestones are met.

Continued on next page

Table 8.1 – Continued from previous page.

ID	Description	Impact	Likelihood	Exposure	Responsibility	Consequences	Prevention
RA1.6	Poor communication	5	7	35	Developer team	Implementation for a member may be different compared to others expectations.	Ensure all members are aware of each members progress. Set meetings to ensure everyone understands what is being done.
2	**Project Planning**						
RA2.1	Unclear objectives set for project	10	8	80	Developer team	Each member has different expectations for how the project is.	Consistently set milestones to check each members progress and ideas.
RA2.2	Focusing on wrong aspects	10	7	70	Developer team	Project does not meet expected requirements, does not perform as expected by the client.	Ensure the team frequently meets with the client to ensure their product meets the client's requirements.
RA2.3	Poor scheduling	10	6	60	Developer team	Team does not meet deadlines.	Ensure all members are up to date with current set milestones.
3	**Implementation**						
RA3.1	Client changes requirements	10	7	70	Developer team	Team needs to change implemented code and must make changes to design to fit clients requirements.	Unable to prevent. Can mitigate by ensuring project is up to date at all times by ensuring documentation and testing is done at every stage.
RA3.2	Poor/Inconsistent styling	5	6	30	Developer team	Inconsistent code styling, harder to read code, harder to understand what each member has done.	Ensure all members agree on a set style before any programming begins. Checkstyle can also be used to ensure code is meeting the same guidelines and is consistent.

Continued on next page

Table 8.1 – Continued from previous page.

ID	Description	Impact	Likelihood	Exposure	Responsibility	Consequences	Prevention
RA3.3	Inexperience with resources	8	9	72	Developer team	More time spent on learning how to use the resource than working on the project.	Team should allow adequate time to learn new resources, or use technology which is familiar.
RA3.4	Lack of testing	8	5	40	Developer team	Code has bugs and may perform poorly.	Constantly test code as it is being made, create test cases before coding.
RA3.5	Requirement is too complex to implement	9	3	27	Developer team	Team unable to implement a certain requirement.	Ensure team is aware of each others strengths and capabilities. Discuss personal strengths and weaknesses, employ peer programming to improve on members weaknesses.
RA3.6	Focusing on quality requirements over functional	5	6	30	Developer team	Final product may not meet clients needs, product may be unfinished.	Ensure team are focused on the correct requirements, set relevant milestones to ensure team are working on necessary requirements.
RA3.7	Poor version control	4	4	16	Developer team	Team may be unable to roll back changes if requirements change.	Use of a version control software to manage previous versions and snapshots of code.
RA3.8	Poor code documentation	7	3	21	Developer team	Code will be difficult to maintain and change in the future, would be harder for other members to modify code.	Write documentation before implementing code, ensure documentation follows the same style rules.
4	**External risks**						

Continued on next page

Table 8.1 – Continued from previous page.

ID	Description	Impact	Likelihood	Exposure	Responsibility	Consequences	Prevention
RA4.1	Other assignments or tests due	7	10	70	Developer team	Project does not meet set milestones and deadlines.	Time management; each member should ensure they balance their time efficiently. Set consistent milestones to ensure team allows time to work on the project.
RA4.2	Learn outage	3	3	9	Developer team	Unable to access resources at time.	Ensure documents are available at all times, ensure members are not falling behind.
RA4.3	Sickness	4	3	12	Developer team, tutors	Member may not be able to contribute to the product.	Unable to prevent. Can mitigate by splitting work equally across all remaining team members, can also let tutors know.

Chapter 9

Project Plan

To see if creating the project's requirements were achievable, a Program Evaluation Review Technique (PERT) chart was used to highlight milestones and estimate times for tasks. This was used to identify dependencies for tasks, as well as to schedule and plan tasks for implementation. Furthermore, time buffers were placed in order to consider for risks specified in the Table 8.1.

9.1 Deliverable Two

A main consideration was test periods (RA4.1), hence space was left prior to September 13 and 16, as seen from the GUI implementation (PP9) to finalising the document (PP14) in Figure 9.1. This leads to the model taking 37 days, with 5 days of leeway. The chart was based from August 14 to the due date of the second deliverable, September 24.

The amount of time considered for a task was based on the size and difficulty of implementing it. The sum of relevant tasks is then split into six, accounting for each member. A day or two may be added as a buffer. Considering the number of classes in Figure 5.2, it was estimated that each person would implement two classes, although the number of people working on a task is not proportional to the speed at which it is completed.

A PERT chart was preferred over a GANTT chart as it had the ability to show dependencies between tasks and to order the tasks effectively. A PERT chart usually shows multiple tasks in parallel, but the team decided to work on tasks in set blocks, leading to a chart with less branches.

Considering the scope of the third deliverable, features that are planned to be implemented are financial displays and more administrator commands. Furthermore, additional thought would like to be placed towards the external database mentioned in Chapter 5.1. These functionalities were not considered due to time and resource constraints.

Updated Project Plan

The original pert chart from deliverable one (D.1) went through multiple iterations. Many changes were made such as generating acceptance tests (PP2) were moved to the start of the implementation. Furthermore the graphical user interface testing and module testing were made to be done in parallel, with two groups working on testing at the same time. The updated PERT Chart can be seen below in Figure 9.1.

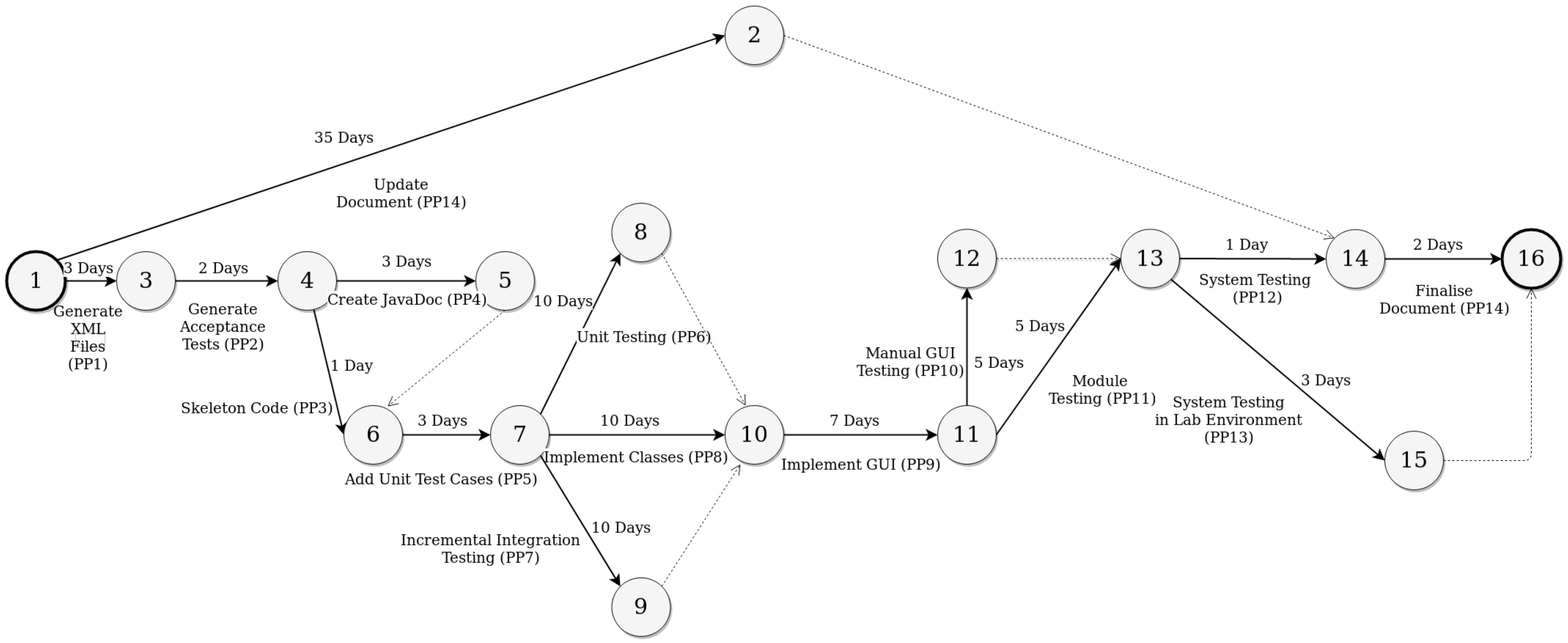


Figure 9.1: Updated PERT Chart for deliverable two, used to schedule and estimate times taken for tasks.

9.2 Deliverable Three

Deciding and allocating tasks for deliverable three was easier than the previous deliverable as the team was more aware of each others strengths and weaknesses. Furthermore better expectations of time constraints has allowed for better time management and allocation of these tasks. Similar to deliverable two, the project plan is expected to evolve over the course of the third deliverable.

Tasks stated in 6.1.3 are also considered in the plan. Limiting user inputs and more touch-screen friendly design will be apart of PP3:1 as the user interface gets further developed. For deliverable three, features to be implemented will be split into sets of three which can be implemented and tested by a group of two. Each new feature will have to have it's corresponding skeleton code, testing and documentation done prior to implementation, in the same order as Figure 9.1.

The tighter time constraints of deliverable three meant not including FR19 and FR16, which will be implemented if time allows. The current project plan allocates 19 days maximum out of the 20 days before the third deliverable for the full implementation of the features, leaving one day of slack time. Furthermore some simple granular additions (PP3:5, PP3:8) have been included as smaller tasks, to account for upcoming test periods for other subjects such as COSC264 and COSC265.

Styling the Graphical User Interface

Designing and styling the user interface is a large task for the group to handle, thus it was left for the third deliverable. Currently the group has decided on spending at most 45 minutes discussing and deciding on a consistent style the the application should have. The rest of the allocated meeting time is spent researching CSS or JFoenix as a means to stylise the application. The implementation of the stylised graphical interface will be split by tasks, currently there are 10 screens that need to be stylised, with an estimated three screens and an estimated two dialog boxes as more features are implemented. The tasks will be equally split to allow for all members to experience stylising the interface to avoid working in silos, this will estimate to two screens per person. This time is also allocated to change window titles of each screen and to add an icon.

Granularity, Priorities and Selection of Features

When deciding what features the system were to have, the team initially prioritised features using the functional requirements in Table 2.2. Furthermore as the project developed, more features were considered and added to the requirements table. Newer features over deliverable two and three were considered based on how much of an effect they make on the system, and if they were unique and would make the system stand out over others. A lot of considered features were not able to be implemented (see section 6.2) due to time constraints. Features were also sorted by granularity in the Trello board, the size of a task was discussed during weekly meetings to ensure that everyone was getting equal work. Most features were split in a way that they could be implemented either alone or by peer programming. Furthermore by meeting twice a week, the team could go over the tasks they completed, and set new ones over the development of the project.

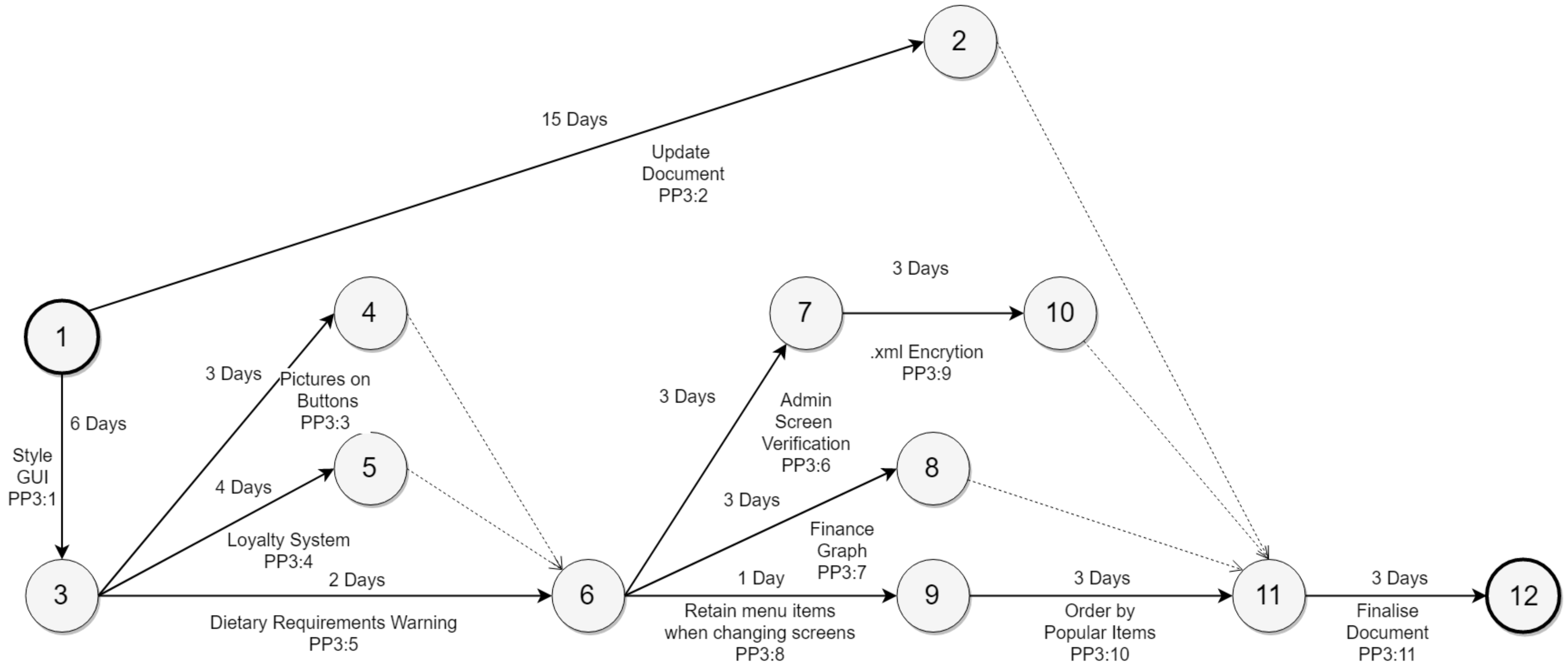


Figure 9.2: PERT Chart for deliverable three, used to schedule and estimate times taken for tasks.

Chapter 10

Lessons Learned

10.1 James

Time Management

Good time management was prioritised as deadlines and assessments for other courses made meeting deadlines for this course difficult and pressuring on the team. This was well accounted for in the project plan and a lot of "crunch-time" was avoided. Furthermore the team met frequently which was initially an issue during the design phase of the first deliverable, as a lot of time was spent discussing minor details of how the product should behave, rather than discussing more important issues. Although as the project developed, the team and I learnt to allocate our meeting times better, with around 25 minutes allocated to recap what each member has done since the last deliverable. Which lead to smoother implementation as each team member knew more details of what was happening in the project, and a lot more time was spent on collaborative work.

Strengths, Weaknesses and Peer Programming

Working in a team has been interesting as each member has their own strengths and weaknesses regarding design and coding. Over the course of the project my team members have taught me many new techniques and skills and vice versa, which will aid in future projects that we work on in the future. Peer programming taught me multiple ways of handling certain challenges, as others have differing thought processes on how a problem can be solved, which made solving problems both easier and more efficient as more solutions were being made.

Modularity and Integration

Making sure my code was modular and worked cohesively with others code was interesting as there were a lot of people working on the same project. Therefore ensuring that my code is well documented, readable and easy to follow was a priority for me. Furthermore by following all of the design processes done in Deliverable one made implementation a lot easier as we all had expectations of how certain modules should interact with each other.

10.2 Yu

Teamwork is Difficult

Working in a team is difficult as I have realised everyone has different view on how to approach things. This makes teamwork harder because other people's view on approaching things do not align with mine and it causes me stress but I also realised that other people's approach may be better in some situation and it allowed me understand their approach and work better with them. Overall I have learned to work better with other people, and became more confident in working with other people.

Time Management and planning is important

Throughout the project I have learned that time management and planning is important. It was really hard to manage time for the project= when there are assignments and tests from my other courses. Throughout the project I have learned to manage my time more effectively. I also realised that planning is a huge part of succeeding in the project, as planning helps to get all team members to be on the same page, in this project I think we didn't do enough planning or didn't plan well, thus leaving us with lots of work on the last few days of a deliverable being due. In future projects, I will take planning more seriously and plan better.

Software tools

Throughout the project I have learned a variety of software tools that are beneficial in the project. I found maven beneficial as it helps automatically manage libraries for the project. Sometimes I also find these software tools troublesome, because they are difficult to learn, and there are errors that happen time to time that I have difficulty fixing.

10.3 Daniel

Working under a deadline

The daunting task set before us was a big hurdle to get over, and I know that we have come a long way. Before this project I haven't really been pushed to complete anything in short periods of time. Our team worked really well together and we managed to split the work fairly evenly. This helped me gain valuable experience completing specific tasks fairly quickly, at least compared to some other COSC courses.

Learning to use new tools

Prior to this, I generally kept coding to the code itself, and rarely used any supportive tools such as build automation and code coverage. This project helped me learn about many of these tools. In particular, I found the code coverage tool very helpful. I also appreciate the time we took as a team to learn to use LaTeX for our documents.

Feature creep

As we continued to design our product, we kept coming up with good ideas for features we could include in the project. Over time this list grew, and although we managed to keep it under control, I think the consequences could have been disastrous if we did not. At the end of each deliverable, we were always pushed for time, often not finishing our implementation until mere hours beforehand. This taught me to keep planning of the product to a reasonable level.

10.4 Michael

Commenting in Code

Good practices of coding includes commenting as the code is being made. Not everyone was doing this leaving a lot of the code not commented in both Java comments and JavaDoc. This made it hard as we would eventually work on that code at a later date or need it but not always understand how it works. This was also a time sink as before every deliverable there was an attempt to comment all that was left. Having to go through others code without comments has helped me realise that there is much to gain from more specific comments in my own code as this will help not only myself, but others too.

Taking the Initiative

A personal problem of mine was the lack of taking the initiative when it comes to doing something. Often times I would only do work when it was allocated to me by others, rather than taking a look at our Trello board which would always have work on it. As the semester progressed I started to see that a lot more good be done if I were to just start working on something immediately.

10.5 Shivin

Team Work

Working in a group has been a challenge especially when everyone has different strategy to approach to the same problem. The most important thing that I learnt was how to meet half ways to a solution with the other team members. One of the main things I have also learned was dealing with conflicts, writing more efficient code and have gained a lot more confidence when working in larger teams.

Time Management

From the beginning of the project every member in our team was highly motivated and wanted to deliver a product that they can be proud of. So managing time and work along with other courses on hand and meet the deadlines was a challenge. By the end of each deliverable, I was getting much more confident and capable of managing not just the time but also the stress.

Overall Experience

Overall I have enjoyed working in the team and am proud of the product that we have managed to produce in such a short span of time. I personally have up-skilled my knowledge in G.I.T and learned more clever and faster ways to write efficient code.

10.6 Tasman

Documentation and testing

When writing tests I sometimes used the incorrect version of functions when there were two functions with similar purposes or an overloaded function and I usually had to look at existing tests to figure out which function should be used. Through this I have learned to make sure to document and test alternative versions of functions to make sure they are working correctly and that deprecated functions should be marked to prevent use.

Time Management

I noticed that I was doing much less work than I should have been during weeks where other classes had tests or things due. I have learnt that I must pay more attention to upcoming deadlines and plan for them.

References

Books

Cockburn, A. (1999). *Writing effective use cases.* The Agile Software Development Series. Addison-Wesley Longman Publishing Co., Inc.

Booklets

Rich Johnston, C. M. (n.d.). *The ≪include≫ and ≪extend≫ relationships in use case models.*

Appendix A

Document Responsibility

The following lists responsibilities that each team member mainly contributed towards to during phase 1, although all members also provided consistent feedback and additions to ensure all sections are thorough and approved by the team.

Yu Duan Use case diagram, Functional and Quality Requirements, System and Business Context, and Executive Summary. Set up and format the LaTeX document.

James Kwok Use Case Diagram, Textual description of Use case diagram, System and Business Context, Risk Assessment, Project Plan, Testing Procedures, Description of Current Product Version, Recording group meeting notes, setting up the git repository, stylising and extending the documents write-ups.

Daniel Harris Stakeholders, Functional and Quality Requirements, Key Drivers and UML class diagram.

Michael Morgoun GUI prototypes, Acceptance Tests, UML class diagram, setting up the git repository, Testing Procedures, Description of Current Product Version, Textual description of Use Cases.

Shivin Gaba Stakeholders, functional and quality requirements, deployment model, key drivers, executive summary and the write ups for the tables.

Tasman Berry Use case diagram, acceptance testing, system and business context, and recording assessment dates.

Combined Personas, proofreading and editing.

Appendix B

Change Log

Table B.1: The table containing the changes made to the document and the section that contains the changes for deliverable 2.

Section Number	Section Title	Change	Responsibility
1	System and Business context	Added paragraphs regarding new sections.	James
1.2.1 and 1.2.2	System and Business Context	Added sentence on iFos' unique selling points, and elaborated on difficulties with using a POS system.	James
2.2	Use case diagram	Use cases "Select item", "View order", "Refund past order", "View total cost", "View change" were reduced into steps. Renamed a few use cases and added a use case for importing data.	James
2.2.2	Textual Description of use cases	All changed use cases in 2.2 were updated.	James
2.3	Requirements Tables	Moved QR6, QR7, QR8, QR11 to functional requirements table, named FR19, FR15, FR16, FR20 accordingly. QR8's description was changed to be more specific and targeted.	James
2.3.1	Functional Requirements	Updated all functional requirements in Table 2.2 to be more specific.	Yu
3	Acceptance Test Table	Updated acceptance tests to be more specific and added more.	Tasman
3	Acceptance Test Table	Hyperlinks and references updated to be in date with 2.2 changes.	James
5.1	Deployment Model	The artefact is nested within the device. A bit more information about deployment model in context to the application and xml files.	Shivin
5.2	Detailed UML Class Diagram	Updated the class diagram to more accurately represent the design of our program. The Worker and Database classes were changed to OrderManager and AppEnvironment respectively. Two new classes, Till and Transaction, were added for Finance Two enumerators were added to monitor attributes of MenuItems and Recipes	Daniel

5.3	UML Package Diagram	Created a new section explaining the structure of our program with a UML Package Diagram.	Daniel
6	Description of Current Product Version	Added paragraphs regarding JAXB and it's validation.	Yu
6	Description of Current Product Version	Added paragraphs and sections.	James
7	Testing Procedures	Added section regarding testing procedures.	James
7	Testing Procedures	Replaced image with a more descriptive one.	Daniel
8	Risk Assessment	Updated prevention measures in Table 9.1 to be more specific.	James
9	Project Plan	Updated Figure 9.1 and added section 9.2 regarding the third deliverable project plan.	James
D1	Appendix	Added Trello Example and Old PERT chart	James

Table B.2: The table containing the changes made to the document and the section that contains the changes for deliverable 3.

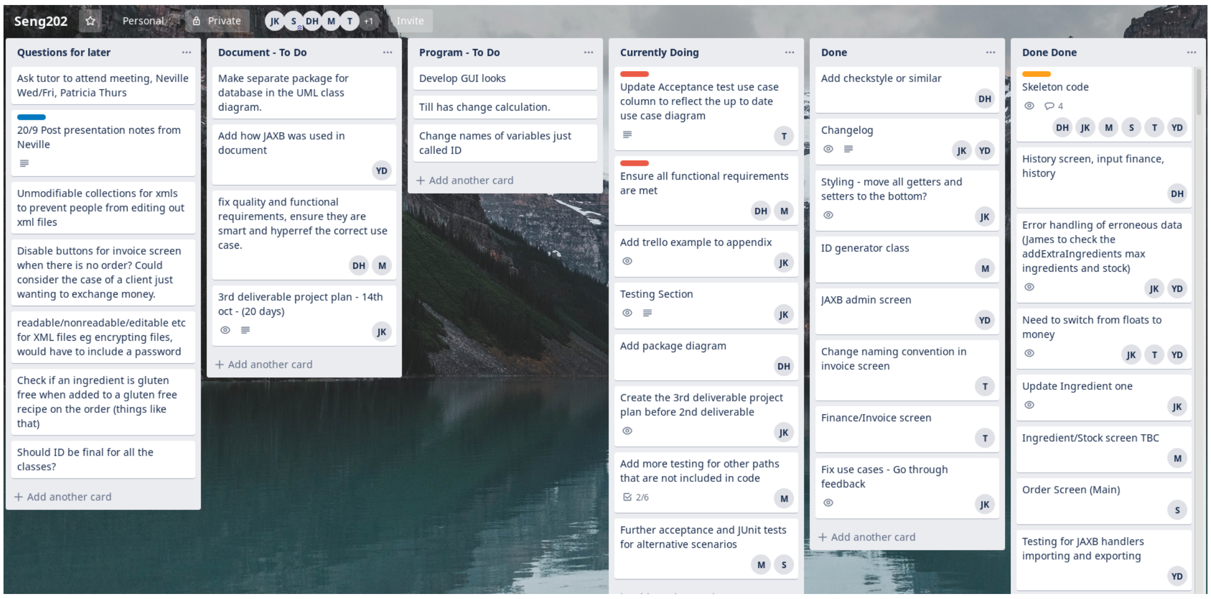
Date	**Section Number**	**Section Title**	**Change**	**Responsibility**
08/10/19	1	System and Business Context	Updated tenses. Removed repeated information and attempting to make the information flow.	Yu
09/10/19	7	Testing Procedures	Section regarding quality testing was added.	James
11/10/19	7.1.3	Quality Testing	Created Section and started description.	James
12/10/19	2.3	Quality Requirements	Updated the quality requirements to be more specific and remade the key drivers table to follow the format on the lecture slides. Then re-prioritised the quality requirements based on the key driver table results	Yu
12/10/19	2.2	Use Cases	Updated the use case diagram and the textual use cases to be more in depth	Yu
12/10/19	2.3	Functional Requirements	re-prioritised functional requirements and updated with refactored use case diagram	Yu
13/10/19	2.1	Stakeholders	Removed kitchen staff from stakeholder as the team decided that the thee was no distinct differences between the kitchen staff and a worker, as they have the same use cases and cover the same people.	Yu

13/10/19	1.1	System Context	Updated system context diagram Figure 1.1 by removing kitchen staff as it was decided that kitchen staff is not a relevant stakeholder, as they are covered by worker.	Yu
12/10/19	7	Testing	Changed description of manual testing due to the introduction of listeners. Also extended the first section regarding testing procedures.	James
13/10/19	7	Testing	Created Manual functional tests FT1, FT3-FT17.	Yu
13/10/19	7	Testing	Created Manual functional tests FT2.	James
13/10/19	7.1.2	Quality Testing	Finished description.	James
13/10/19	6.2	Description of Current Product	Description regarding the product in it's current state.	James
13/10/19	6.2	Description of Current Product	Description regarding features which were considered to be added to the product.	James
14/10/19	2.3	Functional Requirements	Added more functional requirements that were missing as new features were implemented for this deliverable, such as the password setting, loyalty system, and finance graph.	Yu
14/10/19	9.2	Granularity, Priorities and Selection of Features	Section added regarding how tasks were split over the deliverable phases.	James
14/10/10	7	Testing	Created more tests for viewing existing members of the loyalty system and removing members.	Michael
14/10/10	3	Acceptance Tests	Added the acceptance tests that were in Gherkin that had not yet been included.	Daniel
14/10/19	7	Testing	Created Manual functional tests FT18, FT19.	Yu
14/10/19	0	Executive Summary	Extended and updated the executive summary section to be more in depth and to explain newer parts of the design document and system.	James
14/10/19	1	Relevant Business Information	Added small section regarding other users opinions of other point of sales systems.	James
14/10/19	2.2	Use Cases	Added more use cases to the diagram, and wrote the textual description of the use case.	Yu
14/10/19	1	System Context	Extended section to reflect changes in the system and changed wording to be more assertive for unique selling points of the system.	James

14/10/19	5.3.2	Deliverable Three Package Diagram	Added figures and descriptions explaining the how controllers work with implemented classes	Daniel & James

Appendix C

Project Organisation


Seng202 · Personal · Private · JK S DH M T +1 · Invite

Questions for later
- Ask tutor to attend meeting, Neville Wed/Fri, Patricia Thurs
- 20/9 Post presentation notes from Neville
- Unmodifiable collections for xmls to prevent people from editing out xml files
- Disable buttons for invoice screen when there is no order? Could consider the case of a client just wanting to exchange money.
- readable/nonreadable/editable etc for XML files eg encrypting files, would have to include a password
- Check if an ingredient is gluten free when added to a gluten free recipe on the order (things like that)
- Should ID be final for all the classes?
- + Add another card

Document - To Do
- Make separate package for database in the UML class diagram.
- Add how JAXB was used in document — YD
- fix quality and functional requirements, ensure they are smart and hyperref the correct use case. — DH M
- 3rd deliverable project plan - 14th oct - (20 days) — JK
- + Add another card

Program - To Do
- Develop GUI looks
- Till has change calculation.
- Change names of variables just called ID
- + Add another card

Currently Doing
- Update Acceptance test use case column to reflect the up to date use case diagram — T
- Ensure all functional requirements are met — DH M
- Add trello example to appendix — JK
- Testing Section — JK
- Add package diagram — DH
- Create the 3rd deliverable project plan before 2nd deliverable — JK
- Add more testing for other paths that are not included in code — 2/6 — M
- Further acceptance and JUnit tests for alternative scenarios — M S

Done
- Add checkstyle or similar — DH
- Changelog — JK YD
- Styling - move all getters and setters to the bottom? — JK
- ID generator class — M
- JAXB admin screen — YD
- Change naming convention in invoice screen — T
- Finance/Invoice screen — T
- Fix use cases - Go through feedback — JK
- + Add another card

Done Done
- Skeleton code — 4 — DH JK M S T YD
- History screen, input finance, history — DH
- Error handling of erroneous data (James to check the addExtraIngredients max ingredients and stock) — JK YD
- Need to switch from floats to money — JK T YD
- Update Ingredient one — JK
- Ingredient/Stock screen TBC — M
- Order Screen (Main) — S
- Testing for JAXB handlers importing and exporting — YD

Figure C.1: Trello page as of 17 September.

Appendix D

Deliverable One PERT Chart

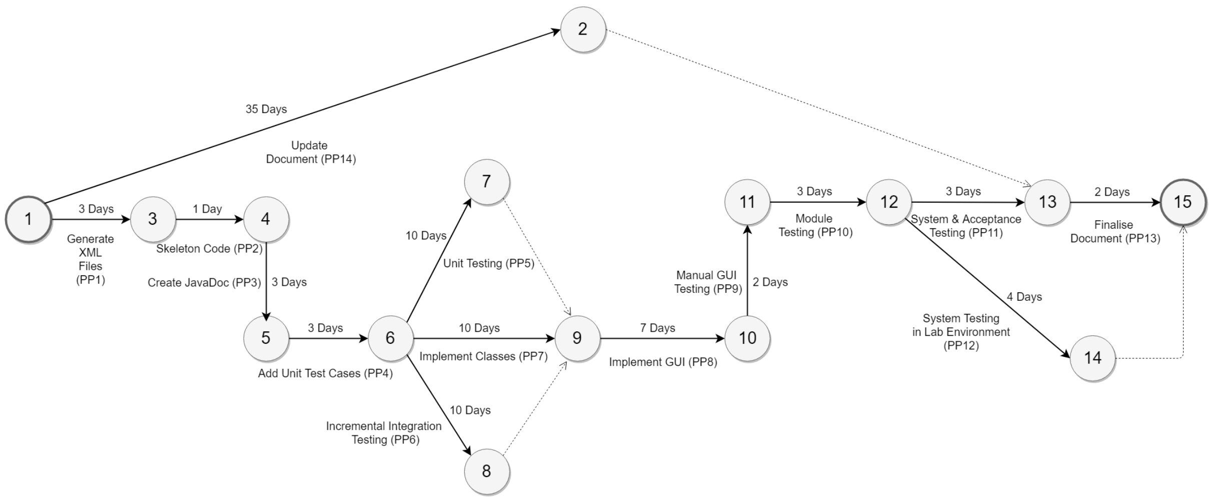
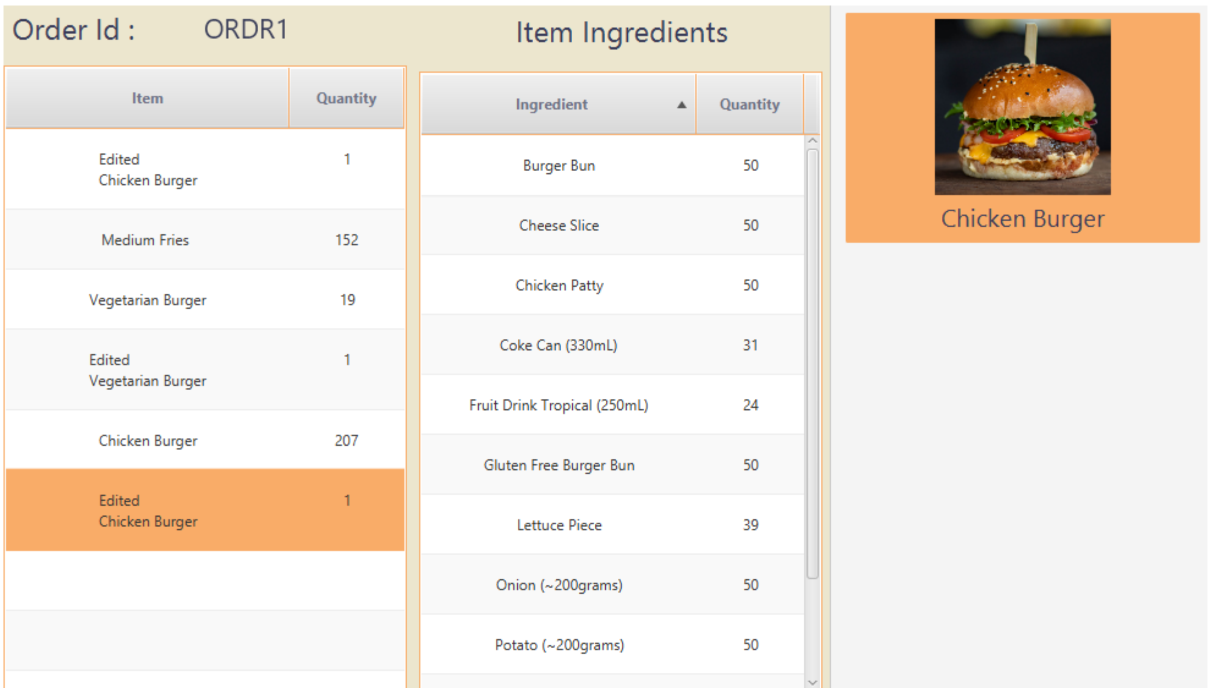


Figure D.1: Unrevised version of the PERT chart from the Project Plan section.

Appendix E

Testing Examples



Order Id : ORDR1

Item	Quantity
Edited Chicken Burger	1
Medium Fries	152
Vegetarian Burger	19
Edited Vegetarian Burger	1
Chicken Burger	207
Edited Chicken Burger	1

Item Ingredients

Ingredient	Quantity
Burger Bun	50
Cheese Slice	50
Chicken Patty	50
Coke Can (330mL)	31
Fruit Drink Tropical (250mL)	24
Gluten Free Burger Bun	50
Lettuce Piece	39
Onion (~200grams)	50
Potato (~200grams)	50

Chicken Burger

Figure E.1: An example of a large order used for testing QR1.