NOTTINGHAM TRENT UNIVERSITY

College of Science and Technology
School of Science and Technology

SOFT30121: Advanced Analysis and Design

Systems Analysis Design and Implementation

By

SOFT30121 Group 11

Oliver Wortley - N1124638
Regan Wills - N0971350
Mouna Nadine Mokkedem - T0278551
Omar Abdin - T0166337
Wenjia Geng - T0268816
Kai Hei Wong - T0267278
Peter Gama - T0162792

Software Requirements Specification (SRS)

Table of Figures

1. Introduction

Future Fridges Ltd. have approached the team to develop the software for FFsmart, a new type of smart fridge targeted at the commercial restaurant sector to allow restaurants to manage their supplies of fresh ingredients more efficiently. The first FFsmart has already been produced, and they would like an external group to develop a working stand-alone system, before considering how to integrate it with their existing hardware.

The FFsmart is an advanced and powerful smart fridge with many capabilities beyond a regular fridge, and these features must be considered when the software solution is being designed. It features both a front and rear door and is intended to be placed against an external wall, allowing both restaurant staff and delivery drivers to interact with it at once.

The solution will keep track of ingredients inserted into and removed from the fridge in real time as well as tracking various datapoints about the ingredient such as the name, quantity, expiry dates and supplier. It will also allow users to register as either a delivery driver, chef, or head chef, all with different permissions and levels of access to the fridge, as well as allowing the software to track who has inserted or removed items and when. This will provide a range of insights to the head chef such as how productive each individual chef is and could help protect the restaurant in the case of missing goods. The login system will also feature an administration panel allowing the head chef to configure user roles at any time.

It will alert the head chef 3 days before the expiration of any ingredients or when any ingredients are running low, and each Monday will re-order any items which are predicted to or already have run out from the corresponding supplier. The head chef will be able to manually adjust or create purchase orders when necessary and a history of all purchase orders will be retained for future viewing. A checking function will be created which, using the history of purchase orders, will determine whether any unexpected items have entered the fridge or if any items are missing from any deliveries. These features will benefit owners of the FFsmart by reducing the time required to document and keep track of inventory, as well as increasing the accuracy and ease of access to stock information.

The software will also generate health and safety reports daily and on demand, which will require it to track the current inventory of the fridge as well as store data about any ingredient that has ever expired while inside the fridge. This will remove much of the overhead time and costs associated with manually bookkeeping this information and will allow chefs to spend more time in the kitchen, where they provide the most value to the restaurant.

The proposed software will be an efficient inventory management system, covering all facets of input, output, and storage of refrigerated ingredients for a commercial kitchen, allowing different users to fulfil their jobs by assigning them suitable privileges – all done through an easy-to-use app interface.

1.1. Purpose of Document

The purpose of this Software Requirements Specification (SRS) document is to outline the schedule and requirements for developing a stand-alone system for the FFsmart smart fridge. To define the requirements of the project comprehensively, this document will elaborate on the system's features both from the user's perspective and the developer's perspective. A detailed definition of the user characteristics for the FFsmart will allow precise identification

of the behaviours, features, and attributes the system needs to have to meet all user requirements. The process of collating assumptions that may influence these factors will effectively contribute to clarifying the project's scope and the system's constraints during development.

In summary, the SRS document describes in detail the scope, user profiles, and functional and non-functional requirements of the FFsmart fridge system and indicates the functionality to be completed and objectives to be achieved, as well as discuss any design constraints. The document will provide a clear development direction for the team, which will increase productivity, reduce development risk, and save development time.

1.2. User Characteristics

Three essential users will regularly be interacting with the FFsmart smart fridge: chefs, the head chef, and the delivery driver.

Head Chef

The head chef has complete control over the day-to-day operation of the kitchen. They are expected to be highly skilled cooks and great at delegating tasks to other chefs and making the best use of available ingredients by managing the menu or creating specials. However, the head chef also has the responsibility of the bookkeeping and administration of the restaurant such as re-ordering stock, keeping track of expiry dates, arranging deliveries, filing purchase orders, investigating unexpected or missing items from the food stores and writing health and safety reports. Any time that the head chef spends on laborious manual bookkeeping and admin is time that would otherwise have been spent cooking and delegating tasks around the kitchen, which is their primary role and how they bring the most value to the owners of the restaurant. Therefore, this user will desire speed and ease of use over anything else. Anything that can help to streamline the administrative parts of their job will be highly valued, such as the ability to generate health and safety reports automatically. The UI for the head chef must be carefully planned as they will need to access many different tasks and functions, while still maintaining an easy to use and quickly manoeuvrable interface. Alerts and notifications will be crucial for this user as they will bear the responsibility of any errors, such as food expiring, that occur in their kitchen.

Chefs

The chefs are likely to be the most frequent users of the software. The chef’s expertise is to ensure the food quality meets the restaurant’s standard, handling the meats, fish, and vegetables, cleaning prep stations and other areas, and handling the storage of ingredients. Most of the work done in the kitchen is divided between the chefs, in the clean and safe preparation of food to be served. Storing and removing ingredients from the fridge is an essential part of their job, so ensuring that this regular task is fast and easy to perform is important for the success of the application. It is unavoidable that this task will be slower using software than without, so it is important that we can utilise the administrative and management capabilities of the fridge to save them time in other areas. These users will only have access to key features so a well-designed UI and fast run time should allow them to avoid being too hindered by the app-based approach. It is also important that is easy to accurately add and remove items to and from the fridge.

Delivery Drivers

The delivery drivers either work for a specific supplier or work freelance, ensuring that the items are transferred from the supplier's warehouse directly to the fridge in a safe and timely fashion. The drivers would typically have to wait for their delivery to be checked and quantified by a chef before they can leave for their next delivery, therefore the speed and ease of use of the application is essential to them to gain benefit from using the smart fridge over a regular fridge. However, if the fridge's checking function works as intended, it may also help remove the burden of responsibility over missing or incorrect goods, making the integrity of the system key for this user. It is also important to note that the drivers should only be able to input goods using the external door; this user should not be able to remove items from the fridge, and therefore will only require a simple user interface, making it easier to use.

1.3. Assumptions

Category	Assumption	Description
User	Users have a unique registered account they can log in to.	All users must be able to have access to the system once the correct credentials have been entered. In this scenario, that will be either the driver, the chef, or the head chef. All these users have distinct roles and behaviours.
User	All users can physically interact with the fridge.	The smart fridge is designed and intended to be for commercial restaurants and should be well placed on-site. It has two doors to support this functionality with one on the back for suppliers and one on the front for the chefs and head chef.
User	Chefs and delivery drivers use the fridge regularly.	The chefs and suppliers will be interacting with the system frequently.
User	Users know how to, and can, physically operate the fridge.	Users of the fridge have at least a basic understanding of general applications and have both the base level of knowledge and physical ability to interact with their device and the fridge, i.e., opening its door.
User	All users have access to a device that the app can be installed and ran on.	Users will need to have a portable device such as a mobile phone that the application can be installed on. The restaurant may decide to give their employees devices for this purpose or allow them to use their personal devices.
Hardware	The fridge has internet access.	The fridge will be relying on having internet access to perform tasks such as sending purchase orders.
Hardware	Hardware can handle concurrent requests.	Courtesy of having both a front and rear door, the fridge is assumed to be able to both receive and output ingredients at the same time.
User	Users have valid and current email accounts.	An active email account will be used for account confirmation when signing up and requesting a password reset.
Physical location	The external door of the fridge is well-placed and	It is important that the external door should be easy for delivery drivers to interact with.

	easily accessible	
User	Users can identify items they wish to remove from the fridge	Users need to be able to identify the items through the user interface by searching or using item categories.
Hardware	The fridge has access to data storage	Some features of the smart fridge will require data to be saved externally from the software such as records of purchase orders and health and safety reports.
Security	Users not using the app are unable to access the fridge.	Users without the application that may be in the restaurant environment such as food servers, owners, customers, and visitors cannot access the fridge.
Roles	The established roles cover the needs of everyone that should be interacting with the fridge.	Everyone that should interact with the fridge is assumed to fall under either the chef, head chef or chef categories. If a particular restaurant requires other employees to interact with the fridge, the existing roles should be used to cover these needs.
User	Users have constant access to their personal devices whilst on site.	It is crucial that users can access their devices to be able to use the system. Some kitchens may limit access to personal devices and therefore should use devices with the explicit purpose of running this application.
Hardware	The fridge's storage is only limited by its physical capacity.	The software will assume that the fridge is large enough to fit all ingredients arriving from deliveries and will not concern itself with registering the size and weight of items.
Hardware	Hardware features always work as intended.	Features such as the checking function will rely heavily on the hardware provided by the FFsmart team. If they do not work correctly then it could have negative implications for the software.

1.4. Scope and Constraints

Scope

<u>In scope</u>

The software solution must be an application which can be deployed on varying mobile devices. It will have a GUI which is user-friendly, coherent, and responsive to different screen dimensions.

There will be three types of users: Chef, Head Chef, Delivery Driver. Each of these has a different access level to the system. Anyone can create an account, but all accounts created must be approved by the Head Chef. To perform an action, a user must first be logged in as a valid user.

When the user first starts the app, they are presented with a start screen which shows two options: Log in and Create account. If they already have an account, they will log in using their credentials. Otherwise, they must create an account using their first and last names,

their email address, a password, and select a user type. After the Head Chef has approved their account creation, they can log in.

Once the user has logged in, they will see a home screen with buttons related to the actions they are permitted to perform, such as inserting or removing items from the fridge.

Delivery Drivers can insert items into the fridge via the external door. Chefs and the Head Chef can both insert and remove items from the fridge via the internal door. The Head Chef can add or remove permissions from individual users, irrespective of their user type, as well as approving creation of new accounts.

After a Delivery Driver has replenished items in the fridge, a checking function will ensure that the items put into the fridge match what was ordered by using the hardware features installed within the FFsmart smart fridge.

The system will push alerts to the device of the Head Chef 3 days before items are due to expire. After alerting the head chef, the system will automatically create a purchase order of out-of-stock items, as well as items predicted to run out of stock within the next week, each Monday, which can then be approved or rejected by the Head Chef. The app will also store information necessary for the automatic generation of daily and on demand health and safety reports such as the current inventory and history of expired items within the fridge.

The inventory of the fridge, along with insertions/removals, purchase orders, suppliers, expiry history, and local system users, will be stored in a cloud NoSQL database, which must be accessible from any device which has the app running on it.

The solution must be highly reliable and performant due to the critical role of FFsmart in the successful operation of the restaurants it is deployed in. The app must run reliably, keep data secure, feature an easy-to-use UI, and contain predictive models that perform to a high level of accuracy to ensure that items do not run out of stock in the fridge.

We observed that the most valuable features of FFsmart, and therefore the features most worth focusing on are automated inventory tracking, automatic reordering of items based on predictive models, and automated health and safety report generation and sending.

<u>Out of scope</u>

One feature not explicitly called for in the specification but could be extremely useful is a fourth role: the Administrator. This would allow some of the administrative responsibilities of the Head Chef such as approving new sign ups and managing roles to be shifted to the system admin, which could be the owner, a senior employee, or anyone else that the restaurant desires. This would have the effect of lessening the strain on the Head Chef, allowing them to better see to their other duties.

A feature that is out of scope is additions to the prediction model such as visualized reporting and alerts on the most and least frequently used/ordered items could help with streamlining the restaurant menu, cutting down on unnecessary/unused ingredients, and thus, reducing waste, as well as increasing profitability for the restaurant.

The system could feature a food classification system to aid with inventory storage and management, such as identifying food as fruit, vegetables, or fish. This could make it easier for foods such as raw meat to be stored safely.

A way for the delivery drivers to message the head chef to warn them of deliveries being late or missing items could be implemented. Similarly, the head chef could have a way to give feedback to drivers or send reviews to suppliers.

Safety features such as automatically logging out inactive users could be implemented but may be a hinderance in a busy workplace with regular usage of the fridge.

If the hardware were available for it, the system could alert the head chef when either door of the fridge has remained open for more than a few minutes to avoid damage to the food or the fridge.

Potentially a feature that could be implemented is that a notification or email is sent to the device of the Head Chef every Monday (when reordering takes place), confirming the items and quantities that have been reordered by the system. This should also contain a receipt for the restaurant's accounts and bookkeeping. In addition, we would recommend adding the ability for the Head Chef or Administrator to download the log files from the system, again for Audit purposes.

Implementation of out-of-scope features will only commence once the core functionality of FFsmart is complete, tested, and approved by the client.

Constraints

We are constrained by various factors in this project. See table below for a full list of constraints:

Category	Constraint	Flexibility	Internal/ External	Present/F uture
Timeline	An MVP system must be completed by 10th Feb 2023	Inflexible	External	Present
Data storage	Data storage must comply with GDPR	Inflexible	External	Present
Reports	System must generate and send reports to health and safety officer every week	Inflexible	External	Present
Operating Environment	App must run on multiple operating systems and devices	Inflexible	External	Present
	Multiple users will need to access the system simultaneously from different devices	Inflexible	Internal	Present
Data storage	Store insertion and removal tracking data in database and log file for Audit purposes	Inflexible	Internal	Present

Client	Gather and store food order data and export this regularly to restaurant for accounting purposes	Inflexible	External	Present
	Data storage must comply with GDPR	Inflexible	External	Present
	Limited meetings with client to discuss requirements and progress on the product	Inflexible	Internal	Present
	Non-technical clients and users must be accounted for	Inflexible	Internal	Present
Financial	No funding for project, no ability to purchase software/hardware/other	Inflexible	External	Present
Design	Adhere to general principles of user interface design	Flexible	Internal	Present

1.5. Glossary of Terms

Keyword	Description
Administrator	A potential fourth role to add to the system. It has the highest level of permissions and administrative abilities, but no physical access to the fridge.
Alert	A notification sent to the Head Chef's device when an important event has occurred.
API	(Application Programming Interface). A software interface that allows two or more computer systems to communicate with each other.
Checking Function	A function performed by the system to check if items specified for insertion match those on the purchase order. This will utilise FFsmart's advanced hardware.
Client	The owners of the FFsmart smart fridge. *Within sections 2, 3, 7 and 9: an instance of the application on a user device.*
COCOMO	(Constructive Cost Model). A regression model relating total lines of code to estimated cost derived from previous software projects.
Credentials	Information used for authenticating users, in this case email and password.
Database	A structured set of data held in a system.
Relational Database	A database that recognises relations between stored items.
Device	A portable device with internet capabilities, presumably a smartphone or tablet
FPA	(Functional Point Analysis). The process of judging software based on the number of functions that it needs to contain.
Framework	A pre-packaged set of components or solutions made available to use and customise
GDPR	(General Data Protection Regulation). A set of regulations in EU law that concerns privacy and data protection.
Hardware	The tangible components of a computer system.

Health and Safety Report	A document containing details of the current inventory of the fridge, as well as a record of all items that have ever expired in the fridge.
IDE	(Integrated Development Environment) An application used for developing software, usually with a powerful suite of development tools
Inventory	All the items currently contained within the fridge.
ISO 9126 / ISO 25010	Software quality standards defined by the International Standards Organisation (ISO)
Items / Ingredients	Food products that belong in a fridge such as fresh fruit and vegetables or raw meat.
MoSCoW	(Must / Should / Could / Won't). A widely used technique used to prioritise software requirements.
MVC	(Model View Controller). A common software architectural pattern that separates a system into three main logical components.
MVP	(Minimum Viable Product). An early version of the system with enough features for it to be used by customers.
Operating System	Software acting as an interface between hardware components and the user. Common examples include Windows, MacOS, and Android.
Platform-Independent	Software that can run on a variety of different hardware platforms or software architectures.
Project	Referring to the FFsmart software project.
Purchase Order	A document detailing the contents of an order, such as quantities and names of items ordered and from whom they were ordered.
QA	(Quality Assurance). Ensuring that system features are of a high quality.
Role	A predefined set of permissions and attributes on the system. All users must be assigned one.
Software	The programs used by a computer system.
Solution	The system: our solution to the problem as defined by the client
Supplier	A business used by the restaurant to source ingredients.
UAT	User acceptance testing
UI	(User Interface). The user-facing part of the application.
GUI	(Graphical User Interface). The same as above, excludes command line interfaces and other text-based interfaces
UML	(Unified Modelling Language). A general-purpose visual language for specifying the design of a system.
Uptime	The time for which a system is operational expressed as a percentage of the total time the system was expected to be operational.
User	Someone who uses the system. Either a Chef, Head Chef or Delivery Driver.
WCAG 2.1	(Web Content Accessibility Guidelines). A wide range of recommendations for making web and mobile content more accessible.

1.6. Overview

The remainder of this Software Requirements Specification document contains the results of the planning and analysis stages of the project. Detailed documentation of functional and non-functional requirements of the software, obtained from the project scenario as well as an

interview with the client, and classified using the widely adopted MoSCoW prioritisation system has been written. As well as this, diagrams and descriptions of the user, software, and hardware interfaces that will make up the solution are provided, as well as a complete list of use and misuse cases following standard UML notation. A full project plan completes with a Gantt Chart, Functional-Point Analysis and COCOMO Cost Estimation has also been designed.

2. Functional Requirements

The functional requirements for our solution are detailed below, ranked in order of most important to least important, and classified with MoSCoW prioritisation system.

ID	Entity	Requirement	Priority
FR1	System	The system must track all items inserted and removed from the fridge.	MUST
FR2	System	The system must keep track of the name, quantity, expiry date and supplier of all items in the fridge.	MUST
FR3	System	The system must have 3 basic roles: Delivery Driver, Chef, Head Chef.	MUST
FR4	System	Users must be able to register on the system with an email, first name, last name, user role and password.	MUST
FR5	System	Users must be able to log in to the system with their email and password.	MUST
FR6	System	The system must have an administration panel where user roles can be configured, only accessible to the Head Chef / Administrator role.	MUST
FR7	System	The system must allow Delivery Drivers to insert items into the fridge.	MUST
FR8	System	The system must allow Delivery Drivers to record what items they put into the fridge.	MUST
FR9	System	A checking function must determine whether the right items have been inserted.	MUST
FR10	System	If the checking function determines an incorrect or missing item in a delivery, the system must send an alert to the Head Chef.	MUST
FR11	System	The system must allow the Chefs and the Head Chef to insert and remove items to and from the fridge.	MUST
FR12	System	The system must generate a purchase order every Monday containing items that have run out of stock and items that are due to run out of stock within the next 3 days.	MUST
FR13	System	The system must alert the Head Chef when a purchase order is generated.	MUST
FR14	System	The system must allow the Head Chef to manually re-order items.	MUST
FR15	System	The system must retain all submitted purchase orders.	MUST
FR16	System	The system must alert the Head Chef if items are due to run out within 3 days.	MUST
FR17	System	The system must add items due to run out within 3 days to the purchase order when reordering.	MUST
FR18	System	The system must store contact details of suppliers as well as a list of items they supply.	MUST
FR19	System	The system must not allow food servers access to the fridge.	MUST
FR20	Reports	The system must generate health and safety reports daily.	MUST
FR21	Reports	The system must be able to generate health and safety reports on demand.	MUST
FR22	Reports	The health and safety reports must contain the current inventory of the fridge as well as a list of all items that have ever expired in the fridge	MUST
FR23	System	The system must alert the head chef when a component in the system encounters a critical error.	MUST
FR24	System	For each insertion and removal from the fridge, the system should record the user, timestamp, item, quantity, and expiry date.	MUST
FR25	System	An Administrator role should be able to be added that does not have access to the fridge but can assign roles and access levels to users.	SHOULD

FR26	System	Users should be able to reset their password.	SHOULD
FR27	System	The system should allow users to log out of their account.	SHOULD
FR28	System	The system should allow users to delete their account.	SHOULD
FR29	System	The system should allow for the Head Chef / Administrator to delete accounts.	SHOULD
FR30	System	The system should require the Head Chef / Administrator to approve every new user account and role associated with the account.	SHOULD
FR31	System	The Head Chef should approve all purchase orders generated by the system before they are sent to the supplier.	SHOULD
FR32	System	The system should alert the Head Chef's device when a Delivery driver enters an invalid purchase order number 3 times in a row	SHOULD
FR33	Alerts	The system should alert the head chef when there are 3 or more invalid log in attempts on the same account within a 10m period	SHOULD
FR34	System	The system should allow Chefs and the Head Chef to view the current inventory through the user interface.	SHOULD
FR35	Reports	The health and safety reports should be in an email-friendly document format.	SHOULD
FR36	System	The system should differentiate between the front and rear doors of the fridge.	SHOULD
FR37	System	If multiple items are due to run out of stock on the same day a single combined alert could be sent to the Head Chef.	COULD
FR38	System	The system could have a food classification system such as identifying fruit, vegetables and raw meat.	COULD
FR39	System	The system could allow delivery drivers to message the Head Chef with queries about orders or updates about deliveries.	COULD
FR40	System	The system could allow the Head Chef to write reviews of suppliers and delivery drivers.	COULD
FR41	System	The system could allow the Administrator or Head Chef to view and download a report of all insertions and removals from the fridge.	COULD
FR42	System	The system could permit multiple users to have the Head Chef role.	COULD
FR43	System	The system could show visualisations of stock level of items over time.	COULD
FR44	System	The system could report on the most and least used items in the fridge.	COULD
FR45	System	The system could alert Chefs when door of the fridge has been left open for 5 minutes.	COULD
FR46	System	The system could log users out automatically after 2m of inactivity	COULD

3. Non-Functional Requirements

The non-functional requirements below are categorised according to the software properties outlined in ISO 9126.

3.1. Usability Requirements

Usability requirements define, in basic terms, how easy the system will be to use. Usability requirements are non-functional requirements as they don't specify functions of the system, but instead how that functionality is to be received by the user. The objective is to create a system that is simple to use and accessible for users of all experience levels.

- Learnability
  - The fridge system must be familiar and easy to pick up for any user with any type of experience using systems.
- Task efficiency
  - The system should be efficient for all users due to the demanding nature of restaurant work.
- Memorability
  - The system must be easy and quick to remember for repeated use.
  - UI should aid all users irrespective of skill level.
- Understandability
  - The UI must be easy to comprehend.
  - The user must understand the purpose of the system and its functionality.
  - The system should be compatible with multiple user devices.
  - The system must be easy to use the app to ordering food from the fridge.
- Accessibility
  - The UI should be accessible for colourblind and poorly sighted users.
  - The UI must be responsive to support a variety of screen sizes and user devices.
  - The UI should be clear with no visual noise to avoid confusion.

3.2. Reliability Requirements

This part of the report will focus on the reliability requirements. These requirements are linked to the product's robustness, fault tolerance and recoverability. Thus there is a need for the software to work according to system-specified requirements. See the requirements table below for full details.

3.3. Performance Requirements

The system needs to be designed and built for high performance when deployed in a restaurant setting. Providing a sufficiently short response time will allow users to obtain information punctually and complete tasks through the system efficiently. The system must have sufficient data storage to store all user accounts and the fridge inventory. The system also demands rigorous scrutiny and administration of permissions for access control, protecting data from unauthorized access and modification, which ensures data privacy and security.

ID	Entity	Requirement	Priority
Usability			
NFR1	UI	The system UI must be easy to use	MUST

NFR2	UI	The UI should be accessible for colourblind users and users with poor vision	MUST
NFR3	System	The system must be easy to learn for users of all experience levels	MUST
NFR4	UI	The UI must be easy to remember for later use	MUST
NFR5	System	The system must be easy to comprehend	MUST
NFR6	System	The system must be compatible with multiple separate user devices	MUST
NFR7	UI	The UI must be responsive to support a variety of screen sizes and user devices	MUST
NFR8	UI	The system UI must align with the web content accessibility guidelines WCAG 2.1.	MUST
NFR9	UI	The UI should be clear with no visual noise to avoid confusion	SHOULD
NFR10	Inventory	The inventory of the fridge should be sorted by name alphabetically and then by expiry date ascending	SHOULD
NFR11	System	The system should remain functional in locations where the internet connection is slow	SHOULD
NFR12	UI	The UI should be helpful for all users irrespective of skill level	SHOULD
NFR13	UI	The UI should be engaging and attractive	SHOULD
NFR14	UI	The UI should emulate a mobile app	SHOULD
Reliability			
NFR15	System	The system must have uptime of >95% over a 1w period	MUST
NFR16	System	The system must be recoverable in the event of a component or complete failure	MUST
NFR17	System	The system must remain functional under high usage levels	MUST
NFR18	App	The app must remain functional in the event of fridge software or hardware failure	MUST
NFR19	System	The system data must be maintained after restarting the system.	MUST
NFR21	System	The system must be able to restore and keep up with up-to-date changes or improvements when turned on, regardless of previous state.	MUST
NFR22	System	The system must be secure in terms of transactions between the restaurant and the supplier. Both parties must feel confident that their financial details are safe from the third party. Appropriate frameworks will be implemented.	MUST
NFR23	System	The system UI should be aligned with the web content accessibility guidelines WCAG 2.1.	SHOULD
NFR24	System	Due to consistence use of the system, it should be without delay or freezing	SHOULD
NFR25	System	Insertion of items should be queue based, one at a time.	SHOULD
NFR26	System	The system should remain functional in the event of any single hardware or software component failing	SHOULD
NFR27	System	The system should work simultaneously across different platforms.	SHOULD
NFR28	System	The system should create backups of data so that it can be restored easily to a working state	SHOULD
Performance			
NFR29	App	All pages and content within the app must load within 1s	MUST
NFR30	System	The system must respond to client requests in less than 1s	MUST

NFR31	Database	The database must be able to store at least 100,000 inventory items and 1000 users	MUST
NFR32	System	The system must store all data and files effectively	MUST
NFR33	System	Log in processing time should be less than 50ms	SHOULD
NFR34	System	Log out processing time should be less than 30ms	SHOULD
NFR35	Alerts	The app alerts should be received on user's device within 1m	SHOULD
NFR36	Alerts	The alert containing list of items to the head chef that are close to running out in the fridge should be sent to chef within 1m	SHOULD
NFR37	System	The order for the items should be sent to the supplier within 2m	SHOULD
NFR38	Report	The health and safety report should be sent via email within 2m	SHOULD
NFR39	System	The system should update the inventory of the fridge within 1m	SHOULD
NFR40	System	The system should be efficient to use for all users	SHOULD
NFR41	System	The system processing speed should remain stable whilst using the fridge	SHOULD
NFR42	System	The system should be able to handle more than 1000 client requests per minute	SHOULD
NFR43	Database	Querying the database should take less than 100ms	SHOULD
NFR44	System	The system should not use more than 4GB of RAM at any one time	SHOULD
NFR45	System	The system should be able to store annual amount of health and safety reports	SHOULD
Security			
NFR46	Database	Users' personal information must be encrypted in storage	MUST
NFR47	System	The system must operate securely by asking users to sign in before use	MUST
NFR48	UI	Passwords must not be revealed to users at any point	MUST
NFR49	System	Users must be logged in before being able to perform actions on the system	MUST
NFR50	Database	The database security layer must comply with GDPR regulations	MUST
NFR51	System	The system should have enhanced security by asking the administrator or head chef to approve each new account on the system	SHOULD
Portability			
NFR52	System	The system must be adaptable to new restaurant locations and user devices	MUST
Maintainability			
NFR53	System	The source code must be written according to ISO 25010 standards	MUST
NFR54	System	The architecture must be designed according to ISO 25010 standards	MUST

4. Interfaces

UML Class Diagram

Below is a UML class diagram for our system, modelling the objects (classes) and connections between them. It represents a static view of the system with a focus on the classes and their respective attributes and operations. For each class, the attributes and functions are assigned a visibility of either public or private, in order to give more detail and aid with implementation of the system. The relationships between the classes are illustrated clearly on the diagram – for example, 1 or more *Items* are associated with 1 or more *Suppliers*, as suppliers can supply many items, and items can be supplied by several suppliers.

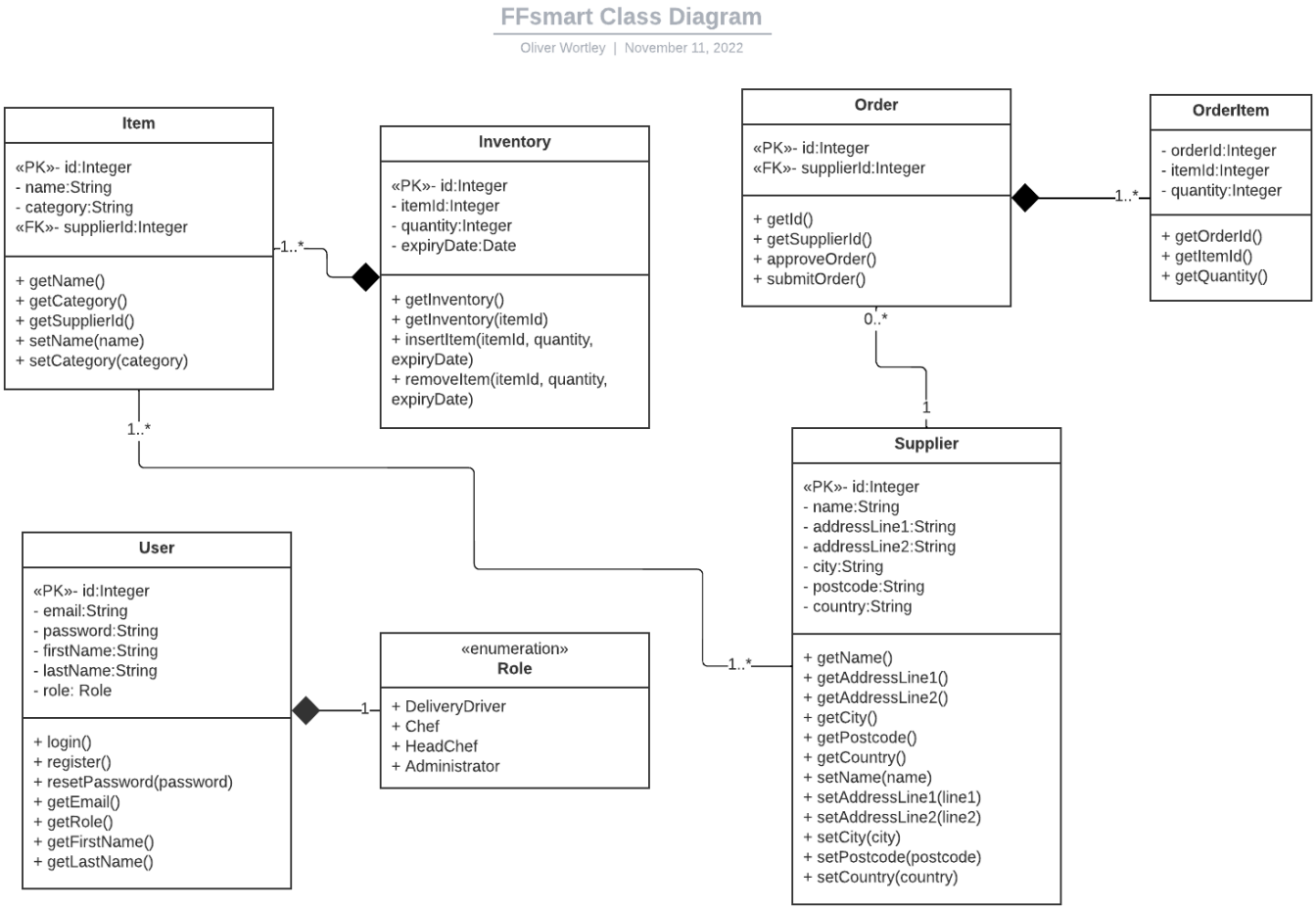
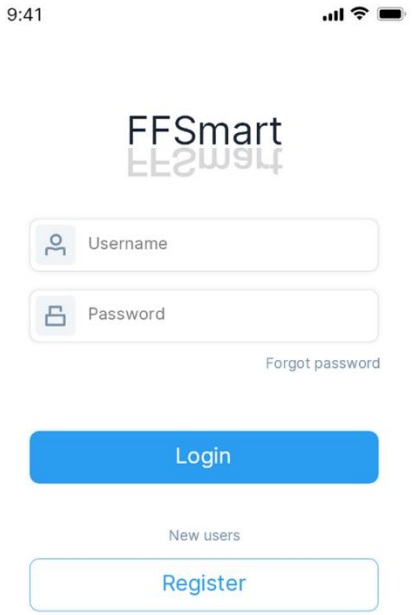
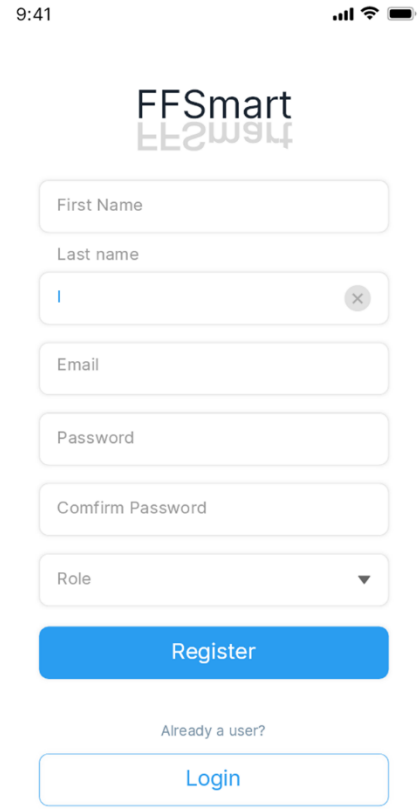
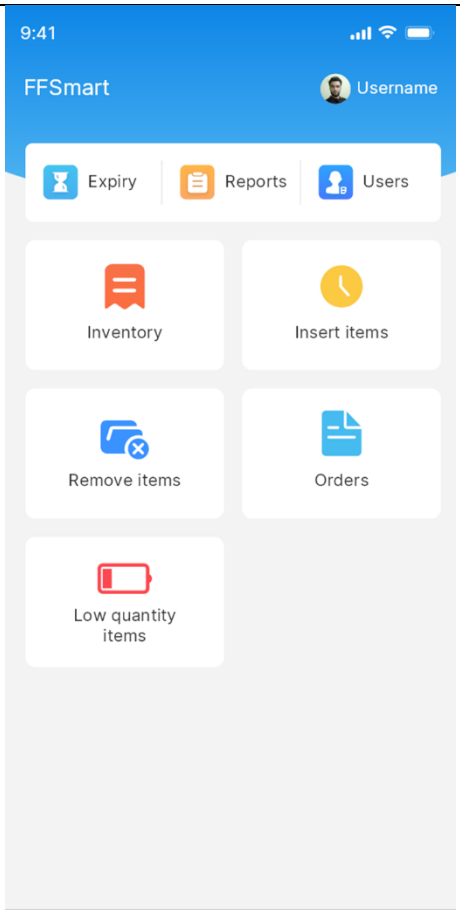
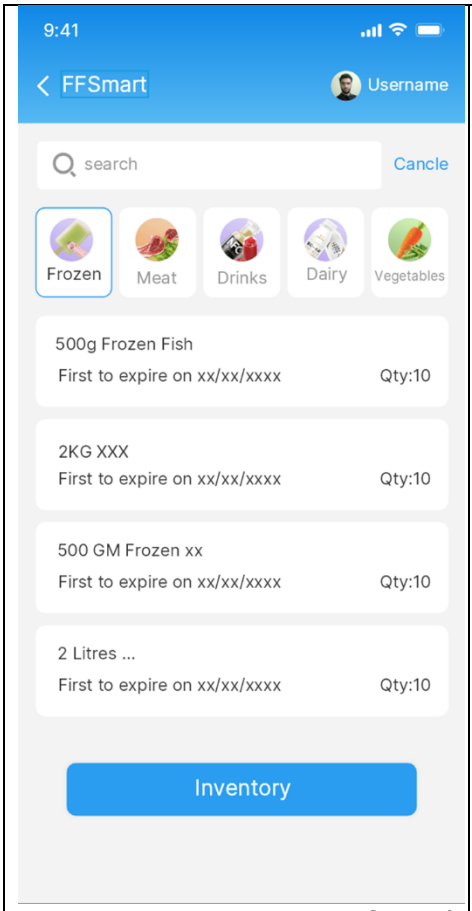
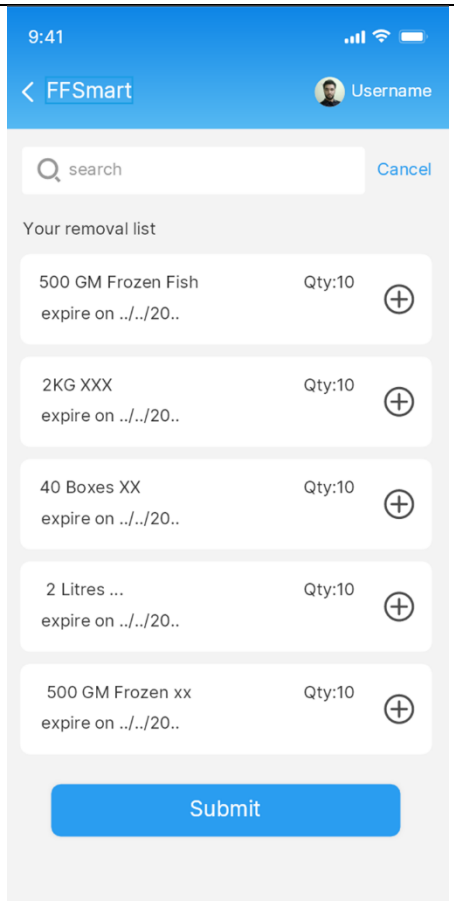
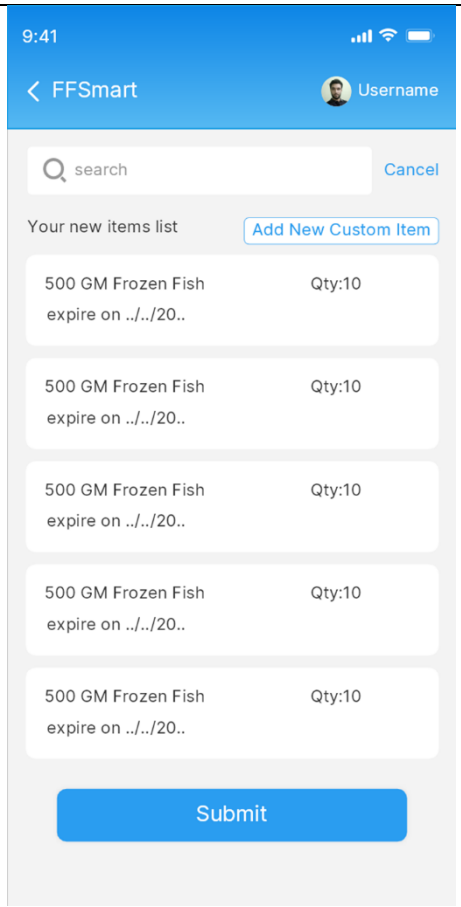
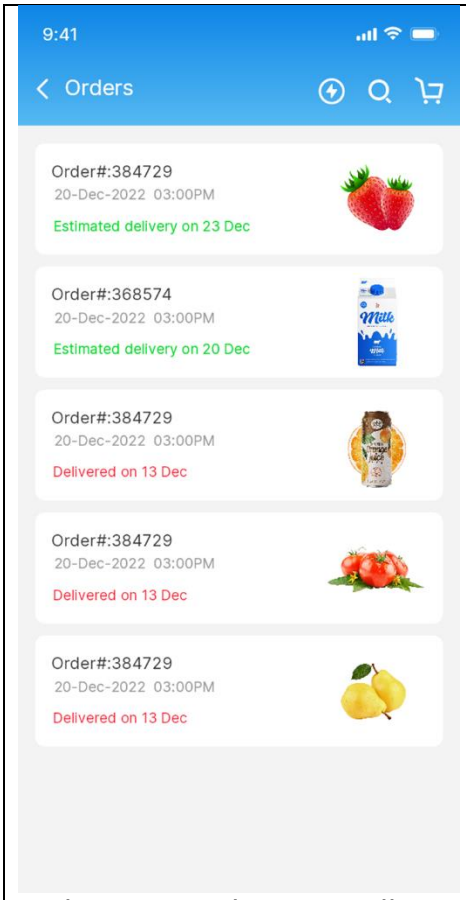


Figure 1: UML Class Diagram

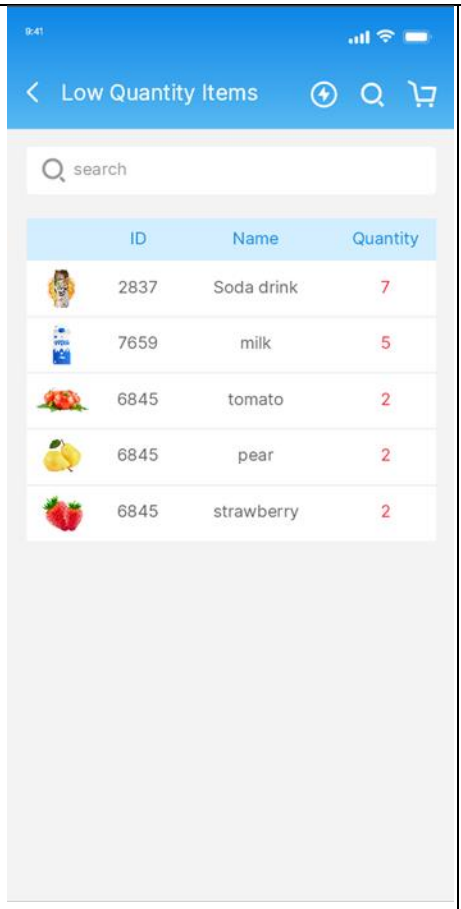
4.1. User Interfaces

		
Login page as referred to in UC6. This page will allow the user to gain access to the system by using their credentials, as long as they are already registered on the system.	Register page as referred to in UC7, this page will allow new users to register their accounts and gain validation and access to the system. Beside the normal input fields, a drop list is included for the role field to limit choices and make it easier to check and validate.	Main page. This page has all functions the user would be able to use as well as smaller buttons for different functions like user management, viewing nearly expired items and viewing/sending health and safety reports. We targeted an easily accessible UI with low visual noise to make it accessible for all types of users. (This prototype shows all functions - a similar design with disabled or hidden functions will be shown to different users based on their role and permissions.)

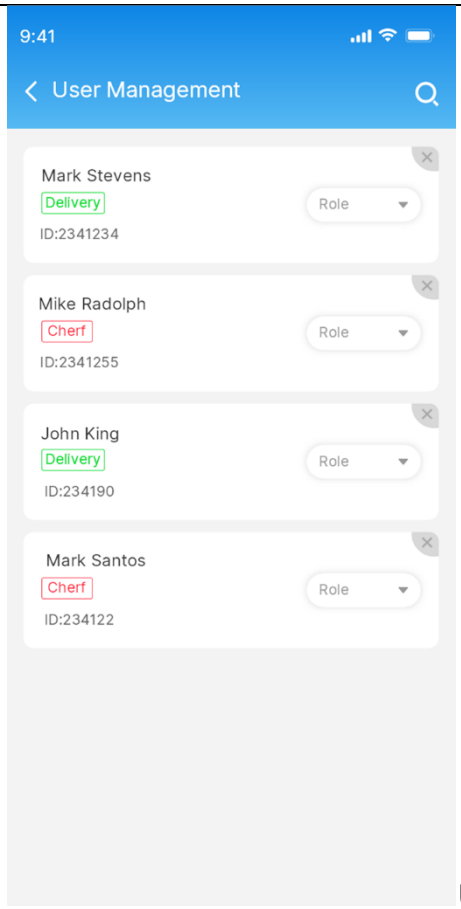
		
Inventory page, as referred to in UC5, inventory is categorized, and this will make it easier to find and sort items. A search bar is also included to be able to search for any item in the inventory. Also, you can send items to the removal page to make your removed items list.	Remove items page, as referred to in UC2, it shows the list of items sent and checked in the inventory that have been marked for removal. Once you press submit the items will be removed from the existing inventory. This should allow validation and checking of items removed with flexibility of cancelling removal of items by simply pressing the minus icon.	Insert items page as referred to in UC1, it allows the user to enter items already found on the system or even the user can enter customized items that's not yet registered. The Submit button allows the items to be validated and checked by the checking function.



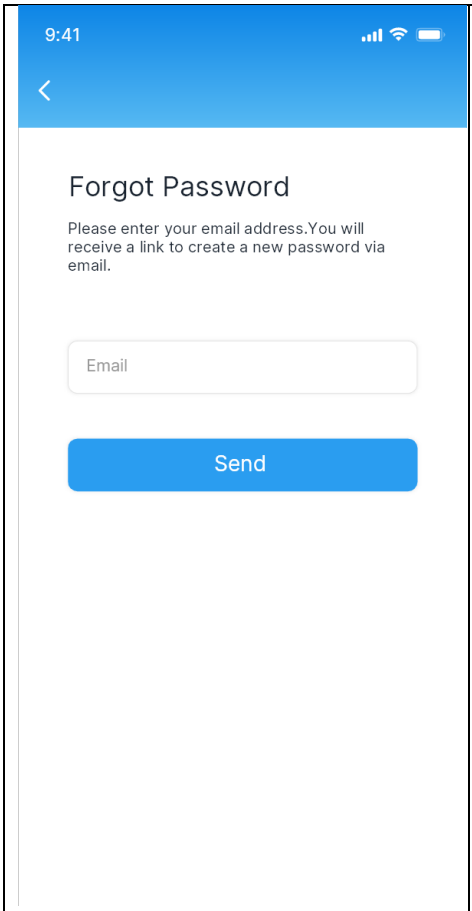
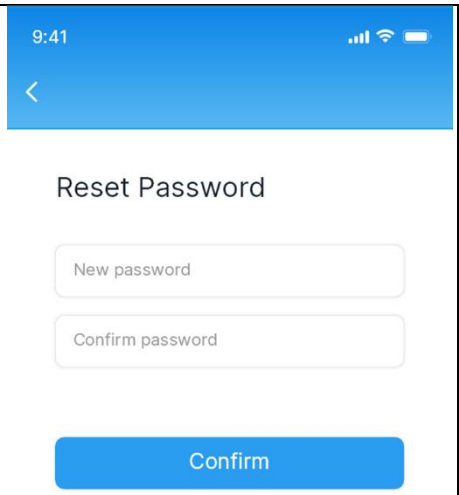
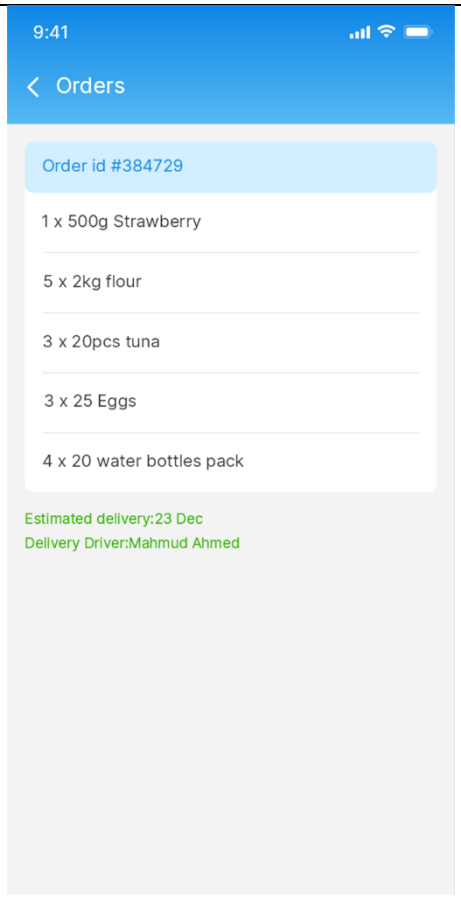
Orders page: this page allows you to edit and approve generated orders, view order history, and create new orders.

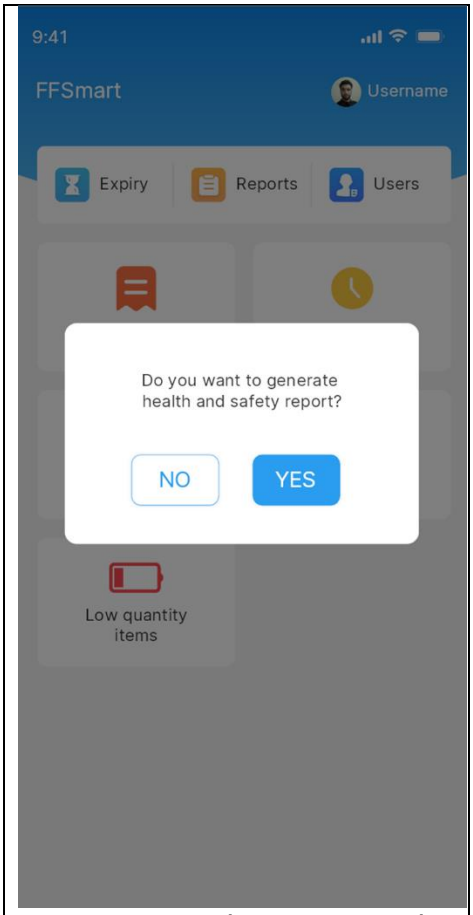


Low quantity items page: as mentioned in UC10, this page shows the low quantity items in the fridge and will allow users to reorder these items through the shopping cart icon on top. The same goes with nearly expired items which gives a notification to throw these items before expiry.



User management page as mentioned in UC8 and UC9. This page will allow the administrator to change user roles, approve new accounts and remove existing users.

		
The forgot password page allows the user to reset their password by sending them a link that directs to the reset password screen	The reset password page allows any user to reset his password by entering a new password and confirming it	The order tracking details page shows more details about the order, date of delivery and name of delivery driver



As mentioned in UC4, the user should be able to trigger generation of a health and safety report which is then sent to the health and safety officer

WCAG 2.1

A key component of developing a user-friendly and inclusive digital experience is adhering to the Web Content Accessibility Guidelines (WCAG). WCAG specifies the requirements for how content should be made accessible to users who have disabilities, including optical, auditory, physical, linguistic, cognitive, and neurological impairments. Accessibility is considered throughout the entire design process. The use of plain, simple language, the inclusion of closed captioning for audio and video content, the use of appropriate headings and structures, the provision of alternative text descriptions for images, the use of adequate contrast between text and background, and the use of clear headings are all examples of this.

Tools like colour contrast analysers and accessibility checkers were utilised to make sure the app complies with WCAG 2.1. A solid WCAG-compatible design will make the app more inclusive and making the app simpler to use and navigate for everyone, including those with disabilities, will help to improve the overall user experience.

UI/UX Guidelines

It is vital to adhere to different UI guidelines which ensures enlarged access and improved accessibility with people facing different obstacles in terms of accessibility, during the design phase, designers have ensured these guidelines are followed and implemented. One of the guidelines were the Google UI/UX guidelines for accessibility, the guidelines gave us a thorough and structured approach to creating user-friendly interfaces, which enabled us to produce a visually beautiful and simple user interface. Our team devoted great attention to the regulations' key features, including typography, colour, and spacing.

4.2. Hardware Interfaces

Since our project is a software simulation, we will not be using any physical hardware for the implementation but will instead use cloud services as an imitation of the hardware. Our client-side application will be a web-app, hosted through a web hosting provider such as Heroku, and our server simulation will be a Azure VM or Amazon EC2 instance, for example. The server will provide the HTTP interface and API, as well as database and file storage. Each client device will make requests to the server over HTTP protocol, and receive responses from the server, which stores the inventory of the fridge as well as users, orders, and health and safety reports. Client devices must have an internet connection and browser for the simulation to work as described. Using a web app ensures that the app will be flexible, reliable and highly interoperable.

The diagram below shows a proposal for the future hardware implementation of the FFsmart system. It proposes having a "dumb" fridge that does not connect to Wi-Fi, but instead has Bluetooth connection capabilities, as well as a server hosted and maintained on-premises with internet connection capability. This server has a large amount of storage on a hard disk drive, which is written to by the programs running on the server. The HTTP server provides access to clients over the web. Clients send requests to the server. The server then processes these requests and sends a response back to the client. Once the client is authenticated (logged in and with appropriate permissions), they can pair their device with the fridge and send a Bluetooth signal to open the fridge door.

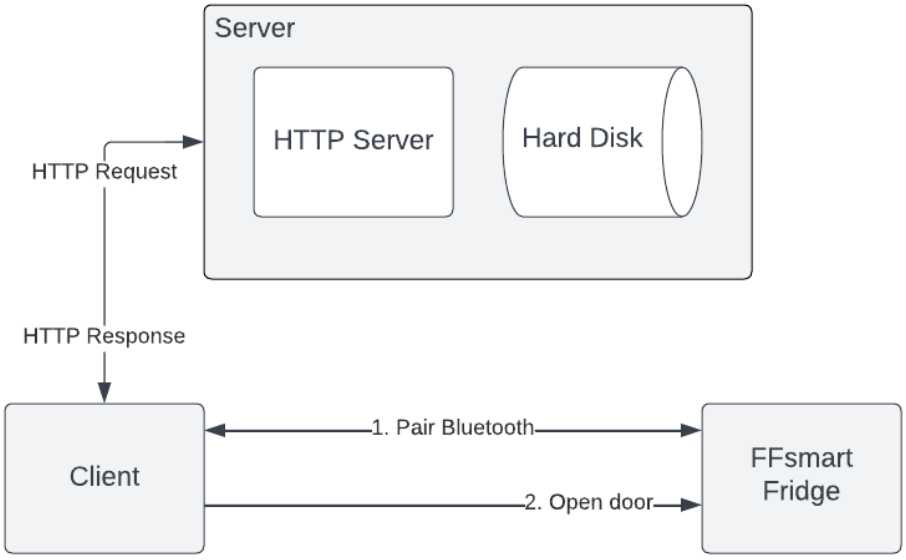


Figure 2: Proposed Hardware Interface Diagram

4.3. Software Interfaces

The software architecture pattern we will use is Model-View-Controller (MVC). This is where the components of the software are divided into 3 layers that interact with each other to form the overall system. Designing the software in this way ensures loo se coupling, high cohesion and strong separation of concerns. It also promotes high fan-in as it promotes reuse of the layers and modules involved. The model contains the data logic and interface(s) to the database, the view is the user-facing side of the system, and the controller contains the business logic. The steps of the MVC logical flow are as follows:

1) User interacts with view.
2) View alerts controller of event
3) Controller updates model
4) Model alert's view (through controller) that it has changed.
5) View grabs data from model (again through controller) and updates itself.

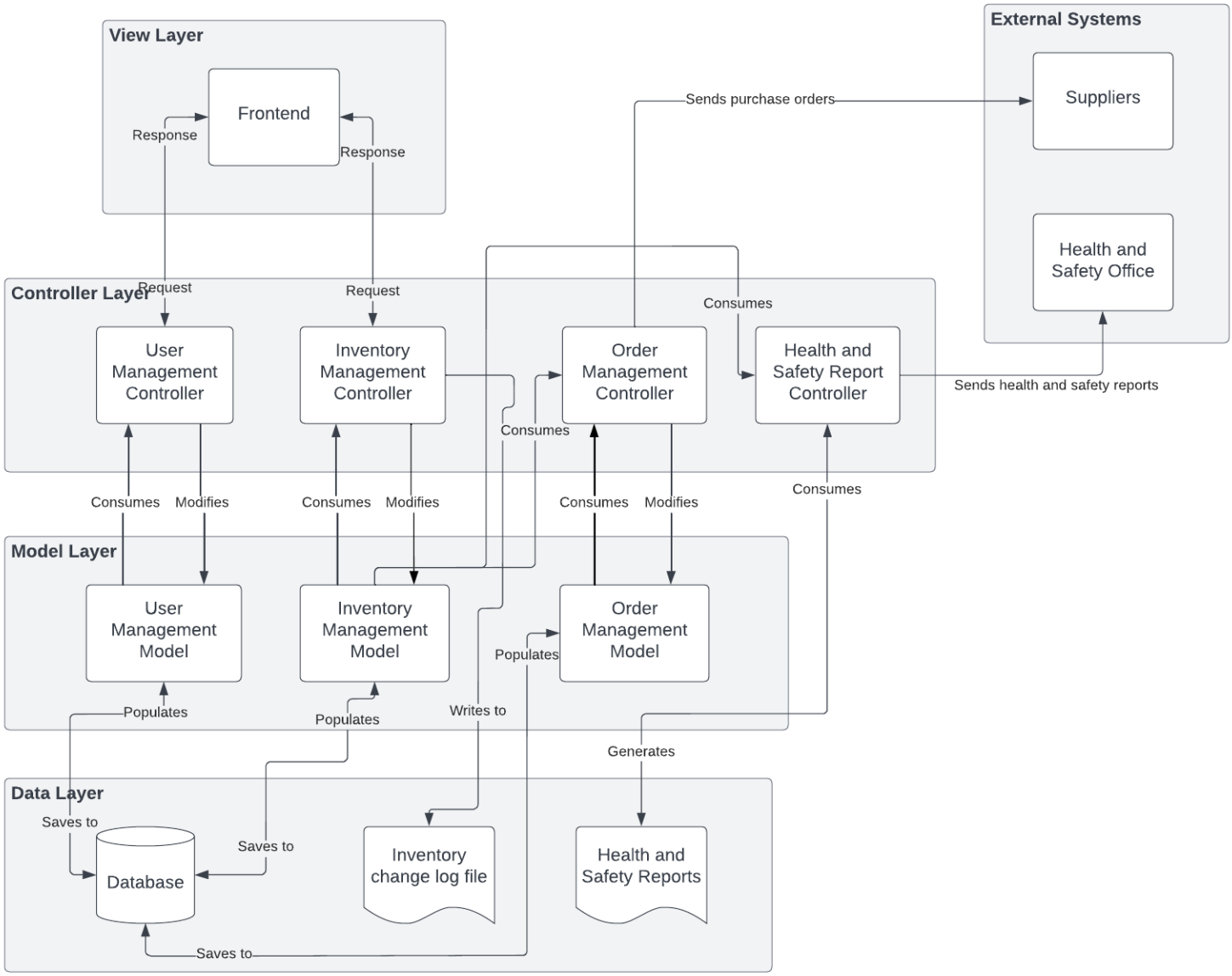


Figure 3: Software Interfaces Diagram

Model Layer
The model layer contains the data logic and interfaces to the database. It also contains the database entities, represented as classes.

View Layer
The view consists of the user interface of the system, represented in our case by a web application running on an internet browser.

Controller Layer
This layer contains all the business logic and functions that help the view get and post data to/from the database.

External Systems
This includes all libraries, functions and classes that are outside the source files of our system.

Data Layer
The data layer of the system, containing the database, health and safety reports and inventory change log file.

Database
The database for the FFsmart system, storing the inventory of the fridge as well as items, users, roles, suppliers, and purchase orders.

5. Use Case Modelling

The use case models show how various actors will interact with the app to achieve an objective and how the system helps them to achieve each objective. Part of the solution features demands both internal and external influences. The diagrams below illustrate the procedures which are essential to initiate these cases.

Actors

- Actors represent the various types of users who interact with the program.
- Head chef - The head chef is the manager in the kitchen, who has advanced permissions for inventory, order, and user management, and can receive alerts related to orders and inventory.
- Chef - The chef represents regular users that can insert and remove items in the fridge, as well as view inventory changes.
- Delivery driver - A user that can insert new items ordered by the system through the back door of the fridge.
- Administrator – The head chef also represents an administrator in the kitchen, who has advanced permissions to manage the other user accounts.
- Supplier – The supplier is not a user in the FFsmart system, but each supplier's details are stored by the system.
- Health and safety officer – The health and safety officer receives health and safety reports generated by the system. They do not have an account on the system.

Systems

- Inventory Management System - This is the most important system in the fridge and is responsible for managing the food items in the fridge, with features such as inserting and removing items, viewing inventory, and removing expired items.
- Order Management System – used to generate, store, and send purchase orders to suppliers.
- Health and Safety Report System – used to produce and send health and safety reports to the health and safety officer. This system will also store the previous health and safety reports in the system for audit purposes and make them available for download.
- User Management System - This system handles authorization of users and account management functions. It is also used by the Head Chef for changing user roles and deleting accounts.

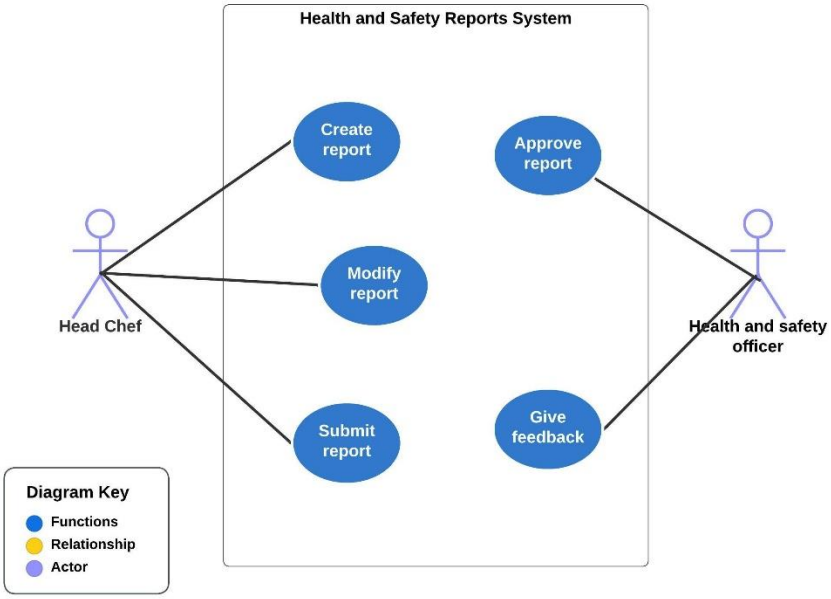


Figure 4: Health and Safety Reports Use Case Diagram

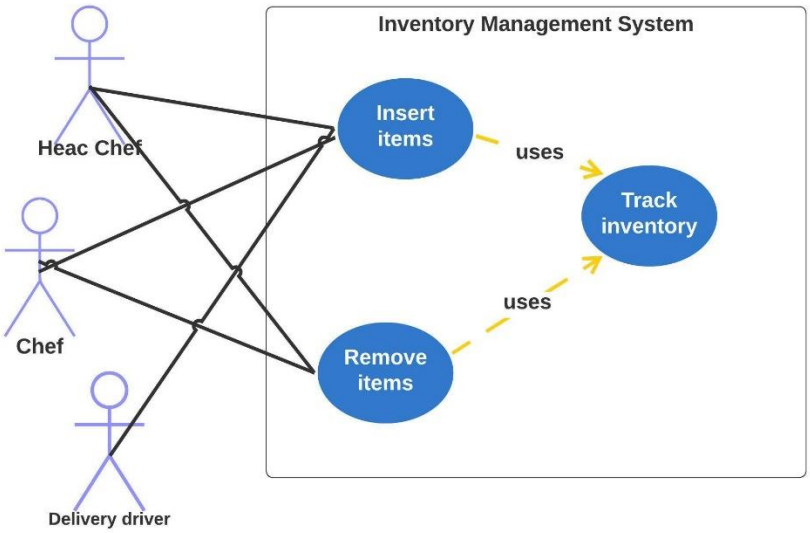


Figure 5: Inventory Management Use Case Diagram

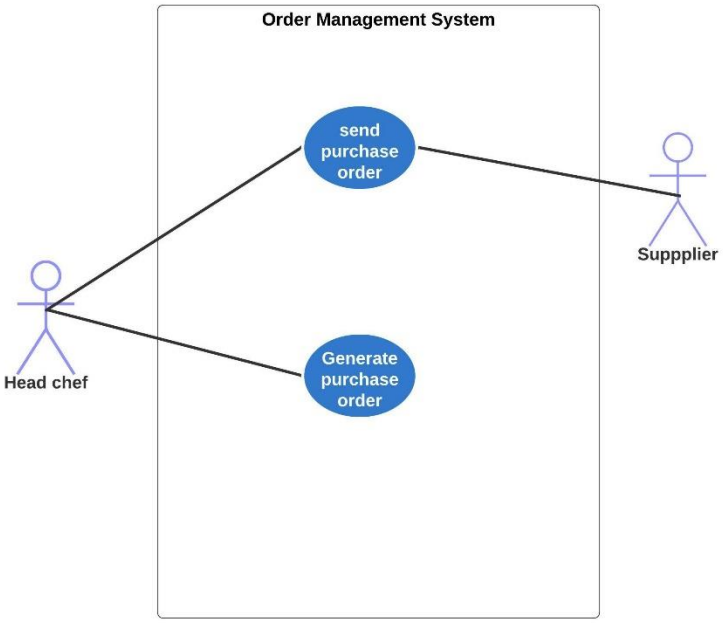


Figure 6: Order Management Use Case Diagram

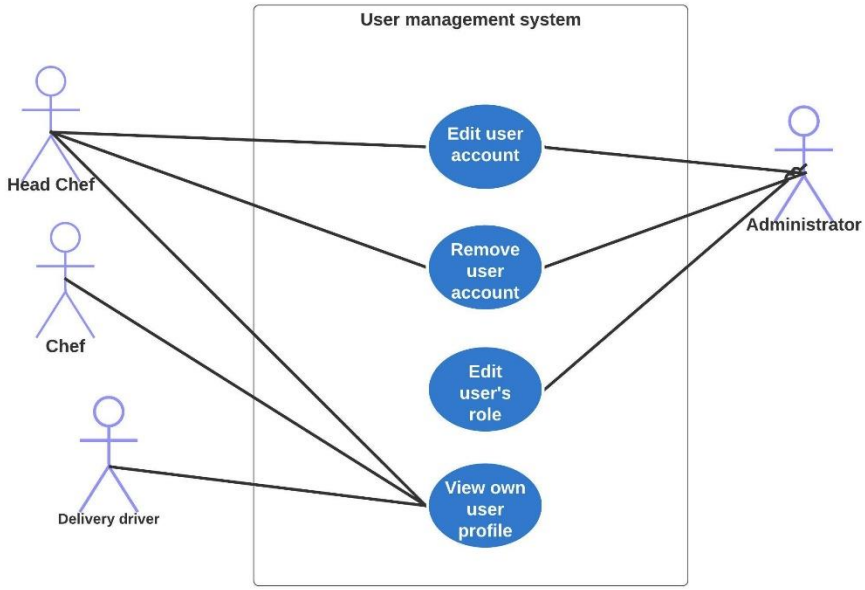


Figure 7: User Management Use Case Diagram

5.1. Use Cases

ID: UC1
Name: Insert items
Actor: Chef / Head Chef
Description: Chefs and Head Chefs can insert items into the fridge. The insertion is then tracked by the system
Trigger: A user wishes to insert an item into the fridge

Requirements: FR2, FR5, FR7, FR8, FR9, FR10
Preconditions: • Item(s) must be ready to be inserted into fridge. • User must be logged in
Normal Course: 1) User logs into application on their device 2) User selects "Insert items" option. 3) User enters items and quantities they are inserting into the fridge. 4) User places items into fridge 5) Inventory is updated, and inventory change logged
Postconditions: • Inventory updated to include inserted items, quantities, and expiry dates. • Insertion log updated

ID: UC2
Name: Remove items
Actor: Chef / Head Chef
Description: Chefs and Head Chefs need to remove items from the fridge in order to produce and cook meals
Trigger: A chef or head chef wishes to remove items from the fridge
Requirements: FR5, FR11
Preconditions: • User must be logged in and must have correct permissions
Normal Course: 1) User logs into application on their device 2) User selects "Remove items" option. 3) User enters items and quantities they wish to remove. 4) User removes items from fridge. 5) User confirms items to be removed from fridge. 6) Inventory is updated and removal logged.
Alternate Course: 1) User logs into application on their device 2) User selects "Remove items" option. 3) User enters items and quantities they wish to remove. 4) Item is out of stock or expired in the fridge. 5) User enters a different set of items. 6) User removes items from fridge. 7) User confirms the removal of items. 8) Inventory is updated and removal logged.
Postconditions: • Inventory updated and removal logged in inventory changes table.

ID: UC3
Name: Review generated order
Actor: Head Chef
Description: After the system has generated an order, the Head Chef should be able to approve or reject the order
Trigger: System generates an order

Requirements: FR32
Preconditions: • Head chef must be logged in to the application on their device. • System must have generated the order. • Must be items to be reordered
Normal Course: 1) Head chef receives alert that order has been generated. 2) Head chef selects Orders option. 3) Head chef looks through order and approves order. 4) Supplier purchase orders generated and sent, stored in system
Postconditions: • Order sent to supplier and status changed to approved.

ID: UC4
Name: Send health and safety report
Actor: Head Chef
Description: Health and safety reports must be sent to the health and safety officer on request
Trigger: Health and safety officer requests a health and safety report of the fridge
Requirements: FR20, FR21
Preconditions: • Restaurant must have received request for a health and safety report. • User must be logged in to application and must have Head Chef / Administrator user permissions
Normal Course: 1) User opens and logs in to application. 2) User selects the Health and Safety Reports icon on the home page. 3) Pre-generated health and safety report sent to the health and safety officer. 4) Health and safety report details can be checked through information button
Alternate Course: 1) User opens and logs in to application. 2) User selects Health and Safety reports option on home page. 3) Report has not been generated for the day due to an error, so a new one is generated. 4) The new health and safety report is sent to the health and safety officer.
Postconditions: • Health and safety report for that day generated and sent to health and safety officer when demanded

ID: UC5
Name: View inventory of fridge on user interface
Actor: Chef / Head Chef
Description: Chefs, Head Chefs should be able to view the current inventory of the fridge through the UI
Trigger: User wishes to view current inventory of the fridge
Requirements: FR2, FR35
Preconditions: • User must be logged in to application and must have Chef / Head Chef access level • Fridge must contain some items
Normal Course:

1) User logs in to application. 2) User selects "Inventory" option on home page. 3) Current inventory is shown on UI (items, quantity, expiry date) 4) Inventory can be filtered with different parameters. 5) User scrolls through and observes items in inventory
Postconditions: • User obtains and views the current inventory of the fridge

ID: UC6
Name: Login
Actor: Delivery Driver / Chef / Head Chef
Description: Allow users to log in to the system
Trigger: A user wants to log in to the application
Requirements: FR5
Preconditions: • User has valid registered account. • User is logged out
Normal Course: 1) User opens FFsmart app. 2) User enters their username and password and confirms. 3) System validates that username is valid and password matches stored value. 4) User logged in and directed to home screen
Alternate Course: 1) User opens app. 2) User enters their username and password and confirms. 3) System finds that username or password is invalid. 4) System prompts user to re-enter their username and password. 5) User re-enters values and presses confirm. 6) System validates that username is valid and password matches stored value. 7) User logged in and directed to home screen
Postconditions: • User is logged in to their account on the app

ID: UC7
Name: Register
Actor: Delivery Driver / Chef / Head Chef
Description: Allow users to register for accounts on the application
Trigger: A new user requires an account on the system
Requirements: FR4
Preconditions: • System must be live and functional. • User does not already have an account
Normal Course: 1) User downloads and opens application. 2) User enters an email, password, first name, last name, and role (Driver, Chef or Head Chef) 3) User clicks "Register". 4) System confirms values are valid (email unique, password meets minimum strength

requirements)
5) User informed their account has been registered and user directed to home screen.

Alternate Course:
1) User downloads and opens application.
2) User enters a username, password, and role (Driver, Chef or Head Chef)
3) User clicks "Register"
4) System finds one or more values is invalid.
5) User prompted to enter different values.
6) User enters new values and clicks "Register" again.
7) System confirms values are valid (username not in use, password meets minimum strength requirements)
8) User informed their account has been registered and user directed to home screen

Postconditions:
- User has a registered account on the system.

ID: UC8
Name: Change user role
Actor: Head Chef
Description: Allows Head Chef to view and change user roles of existing accounts on the system
Trigger: A current user has changed role (e.g., from Chef to Head Chef)
Requirements: FR6, FR26
Preconditions: • User must have an account
Normal Course: 1) User logs into system 2) User selects the person icon on the UI which refers to the "User Management" option. 3) UI shows all current user accounts on the system. 4) User selects an account. 5) User is presented with options: "Change role", "Remove user", "Cancel". 6) User selects "Change role". 7) User chooses a new role and clicks confirm. 8) Account role is updated in the system
Postconditions: • User role changed by administrator

ID: UC9
Name: Remove user
Actor: Head Chef
Description: Allows Head Chef to remove users on the system
Trigger: A current user has left the restaurant and/or no longer requires an account
Requirements: FR6, FR30
Preconditions: • User must have an account
Normal Course: 1) User logs into system 2) User selects "User Management" option. 3) UI shows all current user accounts on the system. 4) User selects an account.

5) User is presented with options: “Change role” and “Remove user”. 6) User selects “Remove user”. 7) User is removed from system
Postconditions: • User removed from system

ID: UC10
Name: View low quantity items
Actor: Head Chef / Administrator
Description: Allows Head Chef or Administrator to observe low quantity items on the system
Trigger: User notified that some items’ quantities are low.
Requirements: FR14, FR16
Preconditions: • At least one item in the fridge is low in quantity
Normal Course: 1) User receives low inventory alert to their device. 2) User selects “Inventory” option on home page. 3) UI shows all inventory in the fridge. 4) User filters by quantity less than 5
Postconditions: • User has viewed items with quantity less than 5

ID: UC11
Name: Remove expired items
Actor: Chef / Head Chef
Description: Allows chefs to remove expired items from the fridge
Trigger: A user wishes to remove expired items from the fridge
Requirements: FR11, FR38
Preconditions: • An item in the fridge has expired
Normal Course: 1) User logs into system 2) User selects “Inventory” option on home page. 3) UI shows all current inventory on the system. 4) User is presented with a button at bottom of screen: “Remove expired items”. 5) User selects “Remove expired items”. 6) User opens door and removes items from fridge
Postconditions: • Expired item(s) removed from fridge

5.2. Misuse Cases

ID: MC1
Name: Register using fake details
Actor: Hacker
Description: A user enters fake details and registers in an attempt to gain access to the system
Trigger: A user wants to fraudulently create an account

Preconditions:
- The system is live and functioning.
- User has access to the app interface

Normal Course:
1) User opens app and selects register option.
2) User enters fake details.
3) User clicks "Register"
4) System ensures email is not already taken and password meets minimum requirements.
5) User directed to home screen of app

Alternate Course:
1) User opens app and selects register option.
2) User enters fake details.
3) User selects "Register".
4) System determines email is already taken.
5) User prompted to enter a new email.
6) User clicks "Register" again
7) System approves details.
8) User directed to home screen

Postconditions:
- User creates a fake account on the system

ID: MC2

Name: Steal items

Actor: Delivery Driver

Description: A delivery driver tries to steal items from the fridge without a valid purchase order number

Trigger: A delivery driver wishes to take items from the fridge

Preconditions:
- Outer door of fridge not in use
- Delivery driver has an account on the system
- Delivery driver does not have a valid purchase order number

Normal Course:
1) User logs into system
2) User selects "My Orders".
3) User selects an order.
4) User clicks on "Deliver" button.
5) Fridge door opens and user removes items from fridge instead of depositing new items

Postconditions:
- User takes items from the fridge

ID: MC3

Name: Remove incorrect items

Actor: Chef / Head Chef

Description: A user removes items from the fridge that do not match the items and/or quantities selected to remove

Trigger: A user wishes to remove items from the fridge

Preconditions: • User has an account on the system with Chef or Head Chef role
Normal Course: 1) User logs into system 2) User selects "Remove items" option. 3) User selects items and quantities they wish to remove and confirms. 4) User removes incorrect quantity of item(s)
Alternate Course: 1) User logs into system 2) User selects "Remove items" option. 3) User selects items and quantities they wish to remove and confirms. 4) User removes incorrect items altogether
Postconditions: • Untracked items removed from fridge. • Fridge inventory no longer accurate • Invalid data in inventory changes table

ID: MC4
Name: Insert incorrect items
Actor: Chef / Head Chef / Delivery Driver
Description: A user inserts items into the fridge that do not match the items and/or quantities selected to be inserted
Trigger: A user wishes to insert items into the fridge
Preconditions: • User has an account on the system with permissions to insert items into the fridge
Normal Course: 1) User logs into system 2) User selects "Insert items" option. 3) User selects items and quantities they wish to insert and confirms. 4) User inserts incorrect quantity of item(s)
Alternate Course: 1) User logs into system 2) User selects "Insert items" option. 3) User selects items and quantities they wish to insert and confirms. 4) User inserts incorrect items altogether
Postconditions: • Untracked items inserted into fridge. • Fridge inventory no longer accurate • Invalid record in inventory changes table

ID: MC5
Name: Attempt to log into existing user's account
Actor: Hacker
Description: An unknown user tries to gain access to an existing user's account
Trigger: An unknown user wishes to gain access to the system through another user's existing account
Preconditions: • User does not have an account on the system.

• User has access to a device
Normal Course: 1) User opens application on device. 2) User selects “Log in”. 3) User enters email of an existing user. 4) User enters a password and clicks “Log in”. 5) System determines invalid credentials and prompts user to re-enter email and password. 6) [Steps 4 and 5 repeated twice] 7) On third incorrect attempt, system alerts Head Chef of invalid login attempts
Postconditions: • Head Chef alerted of invalid login attempts. • User does not gain access to system

6. Project Plan

6.1. Roles

Member	Role	Role Description
Kai Hei Wong	Project Manager	The Project Manager is responsible for the day-to-day management of the project and must be competent in scheduling, resource allocation, risk mitigation, cost estimation and scope. They must have an in-depth understanding of the client's needs and the scope of the project, as well as the risks involved in the project. Responsibilities include writing tickets, documenting meetings and requirements, assigning work, doing project planning, and motivating the team. Marco is an experienced project manager, as he worked in the same role in his year 2 Software Design and Implementation module. He knows how to summarize points made in meetings and make these notes available instantly through Teams, for example. He will utilize his experience and organizational abilities to fulfil the team's objectives and maximize the team's productivity by assigning team members diligently.
Wenjia Geng	Software Engineer / Quality Assurer	The Software Engineer is responsible for the implementation and delivery of the software system. They will design and program the software interfaces and underlying functionality of the system, as well as integrate the frontend and backend of the system, ensuring it meets the standards of ISO 25010. They must have good technical knowledge and understanding, and must be able to write clean, good quality code in the chosen language for the project. Finally, they must also document their code and provide regular updates to the rest of the team. The Quality Assurer is responsible for ensuring that all work produced by members of the team is completed to a standard that meets requirements of the chosen guidelines and relevant regulations, such as GDPR, and proposing measures to improve the work if those standards are not met. They also handle software testing (including requirements/standards testing and finding bugs), reporting on deliverables, and documentation. Wenjia Geng is a Programming enthusiast with experience in software development and is currently interested in affective computing and web 3.0 technologies. He has been involved in developing online education platforms for primary schools. He also has internship experience in web application development, which is helpful for project programming productivity and quality assurance. In addition, Geng Wenjia has a high level of enthusiasm and responsibility for the work he is involved in.
Peter Gama	Systems Architect	The Systems Architect is responsible for the architectural design and eventual deployment of the system. They define system architecture based on the requirements and use cases by researching and selecting suitable architectural patterns and technologies. They must be familiar with the newest relevant technologies and trends and take these into account when deciding on products and patterns to use in the system. Further to this, they document their research and architectural designs, and communicate them to the wider team.

		Peter is the systems architect, having significant experience in systems design through previous projects at university and in the workplace. He has broad knowledge of current trends in architectural patterns and hardware and is skilled at creating UML diagrams and other visualizations of computing systems.
Oliver Wortley	Software Engineer / Quality Assurer	See above for Software Engineer / Quality Assurer role descriptions. Oliver is an experienced engineer, having developed software at a startup and in financial services. He likes to keep up to date with the latest technology trends, specifically in web and mobile app development, and this will be helpful in choosing a framework to build the system on, as well as modules to use. Oliver has extensive knowledge of coding conventions, systems design, and documentation, and he possesses strong teamwork and communication skills.
Omar Abdin	UI Designer / Project Manager	The UI Designer is responsible for designing the user-facing interfaces of the system. They focus on the look and feel of the system and ensure that it meets accessibility and usability standards. The UI designer must be creative with an eye for detail and should be familiar with design tools such as Figma and Adobe XD. Administrative duties as handling team meetings and ensuring the meetings are efficient with proper results with less time wasting and increased functionality of team members. See above for Project Manager role description. Omar has experience as a developer as he worked in web development at a middle eastern medical systems organization. His experience in working within professional teams will help to uphold the standard of work and organization within the team. He will use his experience in developing systems whilst reviewing each stage of development.
Mouna Nadine Mokkedem	Systems Architect / UI Designer	See above for Systems Architect / UI Designer role descriptions. Mouna has experience working with systems and UI design from her projects in second year. She is creative and familiar with Material Design and tools such as Figma and LucidChart. She will create the user interface of the app through working and collaborating with product managers and engineers, as well as adhering to the functional and non-functional requirements.
Regan Wills	Test Engineer / Quality Assurer	The Test Engineer is responsible for writing tests, debugging, and refactoring test code. They must have a strong understanding of the codebase as well as the scope and requirements of the project to ensure that functional and non-functional requirements and tested for and met. They should be able to write both unit and integration tests to achieve high test coverage. They should have knowledge of modern testing frameworks and have good technical skills.

		See above for Quality Assurance Regan is an excellent tester, having worked together on a similar project last year with Wenjia. He will write, comment on, and rewrite much of the report and requirements to achieve the greatest documentation of the project's needs. Over the course of the project, he will scrutinize each section of the report and codebase and check whether they meet the specifications.

6.2. Methodology

The team decided to use the V-model (Waterfall) methodology to achieve the project's development. The following will elaborate on the process of comparing development methodologies, through further analysis of their variants, to the final decision.

When planning a project, choosing the project's development methodology is critical. The Software Development Life Cycle (SDLC) needs to be defined through scientific tailoring after selecting a development methodology applied to the project's characteristics and lifecycle. Next, the project development methodology should be matched to the SDLC model to ensure that the project's quality, cost, and delivery are effectively controlled. Therefore, the team has analysed the Waterfall and Agile methodologies to identify the one more suitable for this project.

Waterfall

The Waterfall model is a more traditional development model that follows the strict sequence of a pre-planned SDLC. It is more focused on the accuracy of the early software requirements analysis, the standardisation of the work process, and team collaboration. The outcomes of each stage need to be fed into the next stage as a baseline. The decision to start the next stage depends on whether the previous stage's deliverables meet expectations. Moreover, a continuous development process needs to be executed with rigorous planning. It finishes with a one-time delivery of all tasks.

The Waterfall model involves two variants, Parallel development, and V-Model development, which will be analysed separately.

Parallel Model

The Parallel model divides the development phase into five stages: planning, analysis, design, implementation, and testing. The development of Parallel is not entirely sequential. The project is divided into parallel subprojects, including design and implementation, after the analysis phase, where each subproject's design and implementation phases are sequential, and the outcome of the design phase will be the input to the implementation phase. When a problem occurs in one design phase, it will lead to rework and a return to the design phase, which may cause delays in the implementation phase. After all sub-projects have been completed, they need to be integrated into a single system. This process of integration can be challenging and involves cooperation between the various sub-projects.

V-Model

Compared to Parallel, V-Model is more focused on testing. It breaks down the development phases into analysis, design, implementation, unit testing, integration testing, system testing and acceptance testing. The different testing phases correspond to the coding, design, and analysis phases, respectively, to verify the outcome of each phase from analysis to coding by testing in various phases to ensure the project quality [4].

The V-Model also has its limitations. For one, the requirements satisfaction is not verified from unit testing to system testing, and requirements satisfactions is only verified in the final acceptance testing phase. Therefore, developers and testers must work simultaneously to conduct the testing during the design and development process to identify deviations in understanding of requirements at an early stage.

The V-Model development process places code implementation as the last step, which allows changes to be rapidly implemented even if customer requirements have changed. This feature reduces the cost and time of backtracking compared to the Parallel mode, which improves development efficiency and reduces development costs.

Agile

The agile development methodology focuses on the evolution of user requirements and uses an iterative, step-by-step approach to development [5]. It breaks the project into manageable sprints and produces valuable features at the end of each sprint, iterating and improving the corresponding features in each sprint through continuous communication with the customer. It emphasizes customer participation, people-oriented, light document, and iteration. All of these are reflected in the Agile Manifesto.

The most common frameworks in Agile models are Scrum and Kanban, which will be analysed in the following sections.

Scrum

Scrum is a lightweight, agile framework that creates value by breaking down large projects into sprints. The Scrum project team has three core roles: Product Owner, Scrum Master, and Scrum Team. The Scrum Master will create an environment where:

- Product Owner: analyses, organises, and generates a Product Backlog of the work required to meet user requirements.
- Scrum Master: Selects work for a sprint period (usually 1-4 weeks) and transforms the selected work into a valuable increment.
- Scrum Team and Stakeholders: Reviews the outcomes and adjusts for the next sprint.

In addition to this, scrum activities include sprint planning/refinement, daily stand-up meetings, sprint review meetings and sprint summaries.

Kanban

Kanban is a relatively easy-to-use project management tool. The core concept divides tasks into three states: pending, in progress and completed. It helps teams to visualise the status.

Each project team member moves the task to the relevant state based on its completion. Thus, Kanban is particularly suitable for simple projects or multi-task projects.

Methodology of Choice

Waterfall & Agile Comparison

Category	Waterfall	Agile
Requirement	Fixed and clear	Unfixed and unclear
Technique	Simple	Complex
Activity	Executed once during the entire project	Repeat until the end of the project
Delivery	One-time delivery	Multiple partial deliveries
Benefits	- Well-defined development phases: Provides review activities for projects by phases. - Roles and tasks are clear: Developers can focus more on development, increasing development efficiency.	- Clear sprint targets - Lower cost and delivery risk - Produce delivery value quickly - Continuous improve deliverables and adapt to changes in requirement
Drawbacks	- High cost of rework - Long development cycles - Focus on document management - Not adaptable to changes in requirements	- Difficult to plan - Projects are easily delayed - Difficult to maintain deliverables consistency - High team competence required -Time may not be guaranteed
Applicable Project	- Little or no change in requirements. - Developers familiar technique area - Low-risk projects - Stable system operating environment - Low need for customer participation	- Unclear requirements, - Innovative project - Market share products - Rapid prototyping tools

Project Characteristics

Category	Project characteristics	Applicability to the Project	
		Waterfall	Agile
Requirement	The requirements of this project are clear and well-defined	Suitable	Somewhat Suitable
Technique	Team members are familiar with the techniques applied with this project	Suitable	Somewhat Suitable
Activity	One-time delivery.	Suitable	Somewhat Suitable

System operating environment	System operating environment is stable.	Suitable	Somewhat Suitable
Customer participation	The stakeholders are only participating in the requirements phase	Suitable	Unsuited

Based on the above comparison, the methodology planned to be used for the project is the Waterfall. The following compares the Parallel and V-Model of the Waterfall methodology.

Parallel & V-Model Comparison

Category	Parallel Model		V-Model	
Characteristics	Suitable	The design and implementation phases are carried out as sub-projects in parallel. Integration of the outcomes of all sub-projects.	Suitable	The development phase and the testing phase are sequential/parallel.
Cost and complexity	Suitable	The cost and complexity of the subprojects are relatively low as they are carried out in parallel and independently of each other.	Somewhat Suitable	The cost and complexity are relatively high due to the need for design and testing to be executed in parallel.
Software quality	Unsuited	As integration is done after all sub-projects have been completed, the total number of defects found during integration will be higher.	Somewhat Suitable	As testing is done in parallel with development, the number of defects that occur is lower.
Testing	Somewhat Suitable	Testing takes place after the development is complete. Testers may not be involved in requirements analysis.	Suitable	Each development phase is tested at its own level. Testers are involved in the requirements analysis.

Based on the above table comparing Parallel and V-Model on four aspects: characteristics, cost and complexity, development quality and testing, we decided to use V-Model. Using V-

Model will help the developers to learn about user requirements and features and eventually generate test cases by reviewing the design documents.

However, the V-Model has some limitations. It divides the development process into different stages with fixed boundaries, making it difficult for developers to cross these boundaries to gather the information needed for testing, which tests should be executed earlier, and which tests should be delayed. This issue needs to be considered during project development.

6.3. Software Tools

Managing a software project can be a strenuous and time-consuming task, so it is important to have effective tools to assist with all parts of the project management process, such as sharing documents, tracking progress on issues, and designing the project roadmap.

Tool	Version	Used for	Justification
Jira	9.4	Project Management, Task Tracking	Jira is optimized for both Agile and Waterfall software projects and is the most popular project management tool. It provides a clear interface for story, task and bug tracking, and allows for custom statuses, as well as creating roadmaps. Each ticket (or piece of work) can be given an assignee, reporter, description, time estimate, epic link, labels and many more options.
MS Teams	1.5.0	Communications, File Sharing/Management	Microsoft Teams has emerged as a highly effective and powerful tool for both individual and group communication. It allows for defining groups and channels where messages can be posted to all members, and members can be tagged in all media posted. For file sharing and management, MS Teams is also a great choice due to being able to add files and directories to channels, as well as open and collaborate on Word documents and Excel spreadsheets in-app.
Git	2.35.1	Source Control	Git is the leading tool in version control and is mainly used through a CLI, though it is also built into VS Code. It features key commands such as cloning repositories, staging and committing changes, checking out new branches, and merging branches. It is also flexible, allowing for many different workflows, as well as being lightweight and easy to use.
GitHub	3.7.0	Cloud Code Hosting and Collaboration	Using a cloud code hosting platform keeps our codebase secure and removes the risk of accidental code loss. GitHub offers a simple visual interface for change tracking

			and version control and allows for assignment of different roles and access levels within a project (e.g., collaborator, reader, owner). Further to this, GitHub offers external integrations, including one for Jira, that can be used to increase cohesion between tools and platforms used in the project.
VS Code	1.73.1	IDE	VS Code will be used for developing the system simulation software. It provides a simple and clean user interface and has a built-in integration with Git. VS Code is able to support development in almost any language and allows users to define language-and-framework-specific preferences. On top of this, it has a huge library of extensions that can be installed to aid with development, such as IntelliCode for AI-assisted code prompts.
Firebase	9.14.0 (JavaScript SDK)	Backend-as-a-service	For the software simulation's server, the team has decided to use Firebase, due to its high reliability and performance, and set of security features such as its authentication service. Additionally, it includes a real-time database and cloud messaging service, along with easy access to Google's Machine Learning APIs.
Figma	116.5.17 (Windows release)	User Interface Design	Figma is an easy-to-use and flexible tool for user interface design. It is highly collaborative and allows for real-time reviewing and editing of design files. Figma works on all major platforms and makes it easy to share files between members of a team. Also, it has lots of integrations that can be used to help in designing the system interfaces.
Lucidchart	Unknown	UML and System Diagrams	Lucidchart is a highly effective and reliable tool for drawing UML class, system and process diagrams. It allows for straightforward visual collaboration and resource sharing and has a generous free plan that offers a wide range of functionality. It is easy to pick up for users of all experience levels and can integrate with other applications such as Jira. Lastly, the tool makes it simple to export diagrams to PDF and multiple image formats.

6.4. Risk Assessment

Severity	Probability	Impact
0 - 5 (Low – High)	0 - 5 (Low – High)	0 - 5 (Low – High)

Category	Risk	Severity	Probability	Impact	Mitigation
Design Changes	If the requirements and UI design change during follow-up development, it will affect the development progress and the program's quality.	3	3	5	Manage requirement changes for the project rationally by analysing the severity, priority, and impact of requirements. Then, choose the best way to address the changes to minimize the impact on the project via analysis.
Delayed Progress	Assignments in other modules may take time away from the group to develop this project and may delay development progress.	3	4	3	Follow up on the progress of each member ‘s task at weekly meetings and make timely adjustments to the project plan to ensure that all tasks are progressing smoothly.
Code Quality	The lack of consistency in coding standards results in each developer's outputs not being integrated and causing unnecessary re-work.	4	3	3	Create the code standard file for the project, share it with all developers, and keep updating it during development.
Inaccurate Estimation	The project plan will be disrupted due to inaccurate effort estimation.	3	2	5	Two weeks after the start of the development phase, we will re-estimate the project using actual data on development size and time and adjust the plan if deviations occur.
Lack of Experience	Lack of experience in the development of relevant products may lead to misunderstandings	3	3	3	Collect problems and concerns from team members about requirements during development and review, and confirm them in interviews with the stakeholders

	of the customer's requirements, resulting in incomplete function developed.				
Team Conflict	A conflict between team members may occur, which can seriously impact the project's progress.	4	1	4	During the project planning process communicating with the team members, dividing up the tasks appropriately and having the agreement of all members. Avoid conflicts among members due to ambiguous task allocation.
Inadequate Test Cases	Inadequate test cases in the testing phase can lead to incomplete testing and not finding all defects in the program, affecting the program's quality and performance.	4	3	4	Ask testers to participate in the analysis and review of project requirements. Check the test cases and test scenarios designed by the testers.

6.5. Task Estimation

Task List

Task ID	Task Name	Priority	Use Case ID	FR ID(s)
T1	Create Register page	MUST	UC7	FR3
T2	Create User management page	MUST	-	FR23
T3	Create Login page	MUST	UC6	FR4
T4	Create Forgot password page	MUST	-	FR19
T5	Create User management page	MUST	UC9	FR21
T6	Create User management page	MUST	UC8	FR2, FR5, FR22, FR24, FR39
T7	Create Main page	-	-	-
T8	Create Orders page	MUST	UC3	FR9, FR14, FR25, FR27
T9	Create Record expired items Function	SHOULD	UC11	FR28, FR29
T10	Create Review auto-reorder list	SHOULD	UC3	FR30

T11	Create Insert items page	MUST	UC1	FR6, FR7, FR17, FR33
T12	Create Remove items page	MUST	UC2	FR8
T13	Create Inventory page,	COULD	UC5	FR38
T14	Create Low quantity items page	COULD	UC10	FR40
T15	Create Send health and safety report Function	MUST	UC4	FR10, FR11, FR12

Gantt Chart

Estimation:

1. According to the estimation result (FPA and COCOMO cost estimation, Appendix B), the development Effort is about 7.099MM (Man Month).

   Estimation Effort = 7.099MM (Man Month)

2. The project period is from 12/10/2022 to 10/02/2023, nearly four months, with one month for First Submission and one month for Final Report, and two months for development and testing.

   Development & Testing Time = 4.0-2.0 = 2.0Months

   The predicted development time accounts for 45%, the testing time accounts for 45%, and the remaining 10% is used to write the user manual and check all achievements.

3. According to the above predicted effort and project period, the estimated number of people required for the project is: 3.5Persons.

   Need Persons = 7.099 / 2.0 = 3.5 Persons

   The size of the project team (7 persons) far exceeds the persons required to complete the project development work. However, considering some factors, such as the team members having other projects, the time assigned to every stage will be 7/3.5=2.0 times of predicted time.

The time allocation of each stage is shown in the following table:

No.	Task	Predict %	Effort (MM)	Months	Days (about)
1	First Submission	-	-	1.5	28
2	Development	45%	3.2	0.9	15
3	Testing	45%	3.2	0.9	15
4	Write user manual	5%	0.4	0.1	2
6	Review and complete all final deliverables of the project	5%	0.4	0.1	3

7	Final Report	-	-	1.5	25

The project Gantt chart can be found in Appendix(c)

Milestones with Key Dates
The milestones and key dates are shown in the following table:

Milestone ID	Milestone Name	Milestone Description	Milestone date
MS1	Planning Milestone	Complete the overall project planning, requirements analysis, UI design and review.	18/11/2022
MS2	Development milestone	Complete coding and code review of all functions.	09/12/2022
MS3	Testing milestone	Complete unit test and integration test of all functions.	30/12/2022
MS4	Delivery milestone	Complete the review of all development achievements to the extent that they can be delivered.	06/01/2023
MS5	Final milestone	Complete the final project report.	10/02/2023

Critical Path
As shown in the figure below, the project's critical path is marked by the red arrow. The earliest start time of the project is 12/10/2022, the earliest end time is 10/02/2023, and the minimum time is 28+15+15+3+25=86 days.

The Critical Path Diagram can be found in Appendix(D).

6.6. Procedures

Tasks not delivered on time

- If a task is not delivered on time, the assignee will be given 2-3 extra days to complete the assignment, depending on the timeframe. They will also be reminded by the team lead to complete the task once or twice during this timeframe. After this point, if the work is still only partially completed, the task will be handed over to another person who is able to pick up the work. If no one else can pick up the work within normal working hours, someone will need to pick it up anyway and work overtime to get it done.

Dealing with team members that do not attend meetings

- In the event that a team member is not attending meetings or delivering pieces of work towards the project, the team lead will reach out to them regarding their participation. There could be various reasons why a team member is not taking part in the project: for example, they may have been assigned work that is too difficult or too easy for them, or they may simply feel like they do not need to attend and are doing enough outside of meetings. In the first case, the team member should be given more easy or

difficult tasks. In the latter case, no further action needs to be taken as long as their contribution is observed to be satisfactory. Alternatively, they may be facing some external circumstances that are affecting their ability to spend time working on the project. In this situation, the member will need to be given softer deadlines and a smaller number of deliverables to complete. Another possible situation is they don't feel that their role is suited to their skills and/or are not satisfied with their position in the team. This situation should be treated on a case-by-case basis, as more detail is needed on the needs of the specific team member.

Setting deadlines

- Deadlines will be set according to the estimated effort required to complete the tasks involved. It is important to set realistic deadlines, so we will use COCOMO cost estimates to judge the tasks that can be completed within the final submission deadline, as well as judge deadlines for completion of individual tasks. To confirm each deadline, the cost estimation and chosen deadline must be approved by another team member.

Reporting problems

- All problems within the team should be reported to the team lead, who will deal with them on a case-by-case basis, ensuring that any conflict is resolved and that all team members are happy and satisfied with the work and overall status of the project.

Reaching consensus

- On all key decisions around technology, architecture, design, features etc., we will use a voting mechanism, in which the option with the most votes is chosen. Only the team members who are directly affected by the decision will take part in the voting (e.g., only the users involved in designing the system will take part in a decision around a feature on the UI).

- In the event of a draw between 2 options, either the project manager or the team member with the most knowledge and experience in the area will make the final judgement.

7. Design Documentation

7.1. Architecture and Design Patterns

7.1.1. Model-View-Controller

The development team chose Model-View-Controller (MVC) as the architectural pattern for the client. As a result, the application is set up so that its components are separated, making it simpler to add to and maintain the software over time. The Model contains the data logic and interface(s) to the database, the View is the user-facing side of the system, and the Controller contains the business logic.

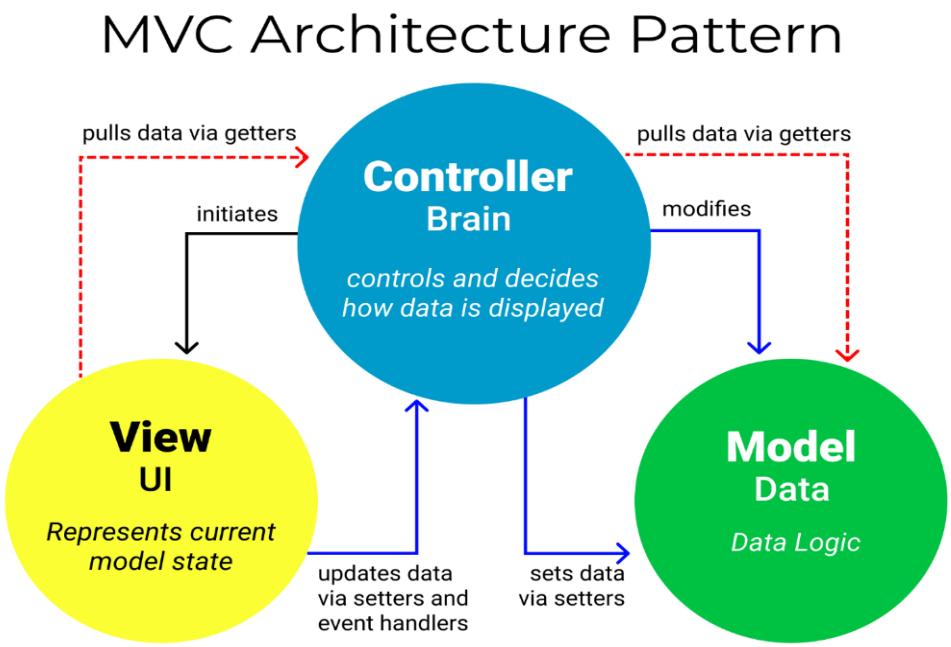


Figure 8: MVC pattern diagram (2023) Freecodecamp.org. Available at: https://www.freecodecamp.org/news/content/images/size/w1600/2021/04/MVC3.png (Accessed: 10 February 2023).

The 'View' of the system is responsible for rendering the user interface and allows users to log in, view and update items in the fridge, view the inventory, view health and safety reports, and view purchase orders. The view also displays alerts and notifications, such as when an incorrect item has been inserted or when items are running low represented in our case by a web application running on an internet browser.

The view is updated via the controller by receiving a notification from the view when a user engages with the application. Handling events such as button clicks, form submissions, and other user actions may be involved. This notification includes details on the type of event that occurred as well as any relevant data related to the event. The view is an independent component which enables updates or changes to be made to it without having an impact on the rest of the application. It is simpler to maintain and improve the application over time due to this separation of concerns and loose coupling.

The ‘Controller’ acts as the intermediary between the view and the model and gets input from the View, analyses it, and then instructs the Model to take the necessary actions. Additionally, it updates the View with the Model's newly updated data. The Controller oversees coordinating interactions between the Model and View and holds the application's business logic. The Controller is responsible for updating the Model when a user performs an action, such as inserting or removing an item from the fridge and then sending the updated information to the View to display to the user. Checking new users’ password strength and filtering inventory items is also performed by the client-side controller.

The ‘Model’ represents the data layer of an application and acts as a mediator between the view (the user interface) and the controller (the part of the system that processes user input). The model is responsible for storing and manipulating the data that is used in the system, such as inventory, users, suppliers, and inventory changes.

Advantages

Firstly, the separation between the model, the view, and the controller in the application into three separate components, each with a specific responsibility, makes it easier to understand and maintain the code. So, changes on any of the three components might not cause changes to the other elements of the system. This loose coupling also makes it easier to test the different components of the application in isolation, which leads to more reliable and maintainable code.

MVC can help to improve user experience as the View component can be easily updated to provide a better user interface without affecting the underlying data or control logic. This makes it possible to improve the usability of the application without having to make changes to the Model or Controller. In addition, the MVC components are often reusable, allowing developers to reuse code across multiple pages and applications. This also allows us to reduce development time and improve code quality.

Disadvantages

MVC can add complexity to a project, as it involves dividing the application into three separate components, each with its own responsibility. This can make it more difficult to understand the overall structure of the application and how the components interact with each other. For example, testing the Model component might involve verifying that it correctly retrieves and updates data, while testing the View component might involve verifying that it displays the data correctly. Testing the Controller component might involve verifying that it correctly handles user interactions and updates the Model and View components accordingly. This can result in a higher likelihood of bugs being missed, as it is easy to overlook interactions between components that are not immediately apparent.

Another disadvantage of the MVC architecture is that the View component’s performance can sometimes suffer, for instance if it receives many updates in a short period of time. This can result in a less seamless user experience and a slower or unresponsive application.

An Alternative Architecture Pattern

Another common architectural pattern that my group could consider is the Model-View-

Presenter (MVP) pattern. This pattern separates the application into three distinct components - the Model, the View, and the Presenter. The Model contains the data and logic for the application, the View is responsible for rendering the UI, and the Presenter acts as the bridge between the Model and the View. We determined that the MVC pattern best met our needs and goals, as it provided a clear separation of responsibilities between components and allowed us to effectively manage and maintain our codebase. While the MVP pattern may have its own benefits, it was not the best fit for our project and the specific challenges we were facing.

7.1.2. Spring WebFlux

Spring WebFlux is the alternative to Spring MVC and introduces support for reactive programming to the Spring framework. The Reactive manifesto [ref] states that application requirements have changed significantly in recent years with the advent of cloud computing and big data, and users expecting "millisecond response times and 100% uptime". So-called reactive systems are proposed as an adaptation to these new requirements: more flexible, scalable, and loosely coupled, which in turn makes them more tolerant of failure and responsive. Reactive programming is focused on reducing blocking operations and having as many asynchronous operations as possible so as to increase the availability and responsiveness of the application. The properties of reactive systems are:

- Responsive (responds in a timely manner)
- Resilient (stays responsive in face of failure)
- Elastic (stays responsive under varying workload)
- Message driven (asynchronous message passing to ensure loose coupling and isolation)

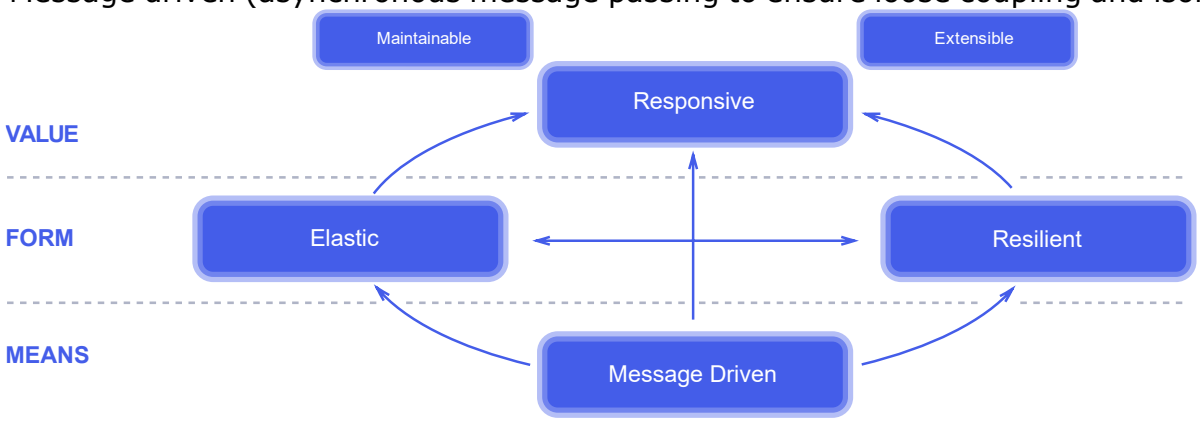


Figure 9: Reactive traits (The Reactive Manifesto (2023). Available at: https://www.reactivemanifesto.org/ (Accessed: 10 February 2023)).

WebFlux is used to create fully non-blocking and asynchronous applications (reactive systems), and is built on Project Reactor, the details of which are out of the scope of this document. Reactor builds on Java reactive streams and provides two specialised types which extend the Java Publisher class: Mono and Flux. A Mono is a publisher of 0 or 1 elements, while a Flux is a publisher of any number N of elements. They can be thought of as being equivalent to a single instance of a class, or a list of instances. These two types are used throughout WebFlux applications: service and repository functions return them, and handling of Mono and Flux responses is done automatically by Spring. When a request to the API is made, a response is only returned when the Mono or Flux is fulfilled.

It was judged that the highly responsive and available properties of WebFlux applications were

more appropriate for our use case than the blocking and less responsive properties of the traditional Spring MVC. In a restaurant setting, chefs will expect quick response times when they are trying to remove items from the fridge, or view inventory, for example. If part of the system fails, they will expect the system to remain available for use, so that restaurant service can continue effectively. Also, there will be periods of less activity and periods of higher activity: restaurants are often busy at lunchtime and in the evening, but less busy in the afternoon. Having a system that can adapt and remain responsive under highly varying conditions is crucial. These properties are exactly those provided by reactive systems, and by Spring WebFlux.

7.2. Design Patterns

Design patterns are reusable solutions to common problems that occur in software design. They provide a way to write maintainable, flexible, and scalable code. In the context of the Model-View-Controller (MVC) architectural pattern, we used the controller pattern and the observer pattern.

7.2.1. Observer Pattern

The observer pattern in the MVC can be used to allow the View component of the MVC architecture to receive notifications when data changes in the Model component. The Model component acts as the subject, and the View component acts as the observer. The subject keeps a list of interested observers who would like to be notified when its state changes. Every time the subject's condition changes, it informs its observers so they can keep informed on the most recent information. For example, in the project, we have a system where the data storage component is the subject that is being observed, and the reporting component is the observer. Whenever the data changes, the reporting component will receive a notification, allowing it to generate and send the reports to the health and safety officer, or to notify a component responsible for exporting data to the restaurant whenever data changes and the Observer pattern also would allow multiple users to access the system simultaneously from different devices, as each user would be able to receive updates independently of the others.

Advantages

The observer enables things to be freely connected with one another. This means that modifications to one thing can cause other items to act without those objects first needing to be aware of one another. By allowing objects to interact with one another without being closely connected, this pattern has the advantage of encouraging flexibility and maintainability. This makes it simpler to add or delete things, as well as to update existing objects, without having an impact on the system as a whole.

Disadvantages

The Observer pattern has a potential for performance overhead and memory leaks if not implemented correctly. It can lead to constant monitoring of subject objects by observer objects, resulting in persistent use of system resources. This can also increase coupling between observer and subject, making them less flexible to change or reuse independently.

7.2.2. Controller Pattern

The Controller pattern can be used in this context to simplify the design of complicated user interfaces, for example, the Controller receives user input from the View (for example, the

user's login credentials or a request to insert an item into the fridge), processes the data, and communicate with the Model to carry out the appropriate actions. And responsible for handling the business logic, such as determining whether the item being inserted into the fridge is correct or not and sending alerts to the Head Chef if necessary.

Advantages

The Controller pattern allows for a separation of concerns, making it easier to maintain the code and make changes to the model or view without affecting the other. Additionally, it provides a way to manage user inputs and coordinate the flow of data between the model and the view.

Disadvantages

The Controller pattern can become complex and difficult to maintain if there are multiple controllers handling different aspects of the application. It can also lead to tight coupling between the controller and the view, making it hard to change or reuse either one independently.

7.2.3. Strategy Pattern

The strategy pattern is another design pattern used in MVC and is a key example of polymorphism in software. It is applied in this project's context for handling user events in the view layer with an added controller class, instead of handling the event entirely within the view. In this way, the logic of the system is separated from the view layer (separation of concerns) and thus can be changed more easily. This helps to make the code more understandable and maintainable and aligns with SOLID principles. Further to this, multiple controllers can be added if different functionality is required for different cases. For instance, consider a system that handles the contents of a refrigerator. For delivery drivers and chefs, it makes sense to insert and remove products from the refrigerator in different ways. The Chef has the ability to insert and remove items, but the Delivery Driver can only remove items. In such a case, it would be beneficial to have multiple controllers that each implement a role-independent high-level controller interface.

Advantages

The strategy pattern increases the system's scalability and testability by allowing alternative strategies to be tested independently of the rest of the system. This makes it possible to maintain the system's accuracy and stability as it expands and changes over time.

Also, the pattern allows for the separation of business logic from the view layer, resulting in independent components that may be easily modified or expanded without affecting one another or other parts of the system.

Disadvantages

The splitting of the view into a view and controller layer can make the system more complex because it demands maintaining a larger variety of classes and linkage between them. Because of this, it could be more challenging for developers to understand the system, particularly if the code is otherwise not well-organized or documented, and the code can become more difficult to debug due to the logic being in separate classes.

Utilizing a greater number of components might also increase system overhead because it uses up slightly more memory at runtime. This may affect performance, especially in systems with restricted resources, but this does not apply in our use case. Finally, if the controller methods are not properly maintained, this could result in bugs entering the software.

7.3. Error Handling

7.3.1. Server

On the server, custom exception handling was implemented, overriding the default Spring error configuration. This enables sending custom error responses with a message, status code and timestamp, and also allows for attaching custom HTTP error response codes. The screen grab below shows the override of Spring's default global error attributes and the use of a specialised function for extracting a HTTP status from any error thrown by the system.

```
public class GlobalErrorAttributes extends DefaultErrorAttributes {

    no usages   orw22
    @Override
    public Map<String, Object> getErrorAttributes(ServerRequest request, ErrorAttributeOptions options) {
        Throwable error = getError(request);

        final String timestamp = LocalDateTime.now().format(DateTimeFormatter.ISO_DATE_TIME);
        String message = error instanceof ResponseStatusException ? error.getMessage().split( regex: "\"")[1] : error.getMessage();
        return Map.of(
                ErrorAttributesKey.STATUS.getKey(), determineHttpStatus(error).value(),
                ErrorAttributesKey.MESSAGE.getKey(), message,
                ErrorAttributesKey.TIME.getKey(), timestamp);
    }
```

Figure 10: Custom error attributes configuration

```
public Mono<String> getUserNameFromId(String userId) {
    return userRepository.findById(userId)
            .switchIfEmpty(Mono.error(new ResponseStatusException(HttpStatus.NOT_FOUND, USER_NOT_FOUND_MSG)))
            .map(usr -> usr.getFirstName() + " " + usr.getLastName());
}
```

Figure 11: Mono switchIfEmpty example

Another example of error handling is shown above: The Mono class's *switchIfEmpty* method is utilised to handle void responses from the database (meaning that a user with the requested ID does not exist). In this case, a 404 not found response code and the message "User not found" is returned, which would then be displayed on the UI as a toast.

Table 1: API Error Response Codes

HTTP Response Code	Description	Example
400	Bad Request – user has provided incorrect data	Log in request with email in number format instead of string
401	Unauthorized	User does not provide a token in their request to the API
403	Forbidden – user is authorized but does not have correct permissions (wrong user role).	Delivery driver tries to remove items from the fridge
404	Not found	Head chef attempts to view deleted

		user's profile
405	Method not allowed	Mobile app tries to call an endpoint that does not exist
422	Unprocessable entity	Malformed JSON in request to server
500	Internal server error	Connection to cloud database failed
503	Service unavailable	Server has crashed due to overload of requests

7.3.2. Frontend

Error handling has also been implemented on the client side for all methods in the two classes that are encapsulated for reuse. These classes are *MyJsonUtil* and *MyHttpUtil*, and the following code describes how these two classes implement error handling.

MyJsonUtil

MyJsonUtil is used to convert between JSON and java objects. Exceptions may occur when calling the *fromJson()*/*toJson()* methods of the google GSON library, e.g., when the data passed in is not in JSON format or a Java object, which may cause a *JsonSyntaxException* to be thrown. Therefore, Java try-catch is used in each method of this class for error handling.

As shown in the code below(Figure 12), the try-catch catches the Exception thrown during the Gson conversion using the *fromJson()* method and prints it out in the catch section using the *e.printStackTrace()* method.

```
public static String toJson(Object object) {
    String result = null;
    try {
        Gson gson = new Gson();
        result = gson.toJson(object);
    }catch (Exception e){
        e.printStackTrace();
    }
    return result;
}
```

Figure 12:GSON error handling

This exception handling prevents the client from crashing because of processing an HTTP JSON response with abnormal data. In addition, the system can store the error messages in the catch in the system log, which can help the system maintenance locate and solve the problem that caused the error quickly.

MyHttpUtil

The *MyHttpUtil* class implements the HTTP protocol communication between the client and the server. The methods in the class use try-catch and try-with-resources methods to catch errors that occur during server communication.

As shown in the code below (Figure 13), try-catch is used to catch *IOExceptions* (input-output

exceptions) that occur when sending requests using *HttpURLConnection*, and try-with-resources is used to catch *IOExceptions* that occur when getting the data from the buffer when using *BufferReader* and *InputStreamReader*.

As both *BufferReader* and *InputStreamReader* extend the Reader's *AutoCloseble* interface, there is no need to write code to close the resources occupied by *BufferReader* and *InputStreamReader* in finally after each response has been read. As shown in the code extract below, only the connection resources used in the try-catch need to be closed. The use of try-with-resources in error handling significantly simplifies the complexity of the code. Also, the resources to be closed are clearly indicated in the parentheses after the "try", which improves the readability of the code. Lastly, the try-with-resources automatically closes resources mechanism avoids the risk of resource leaks caused by forgetting close resources, thereby further optimizing the system's security.

```
try {
    url = new URL(apiUrl);
    connection = (HttpURLConnection) url.openConnection();
    stringBuilder = new StringBuilder();
    connection.setRequestMethod(method);
    String token = SPUtils.getInstance().getString(Objects.requireNonNull(LoginSession.getUid()));
    connection.setRequestProperty(AUTHORIZATION, BEARER + token);
    connection.connect();
    responseCode = connection.getResponseCode();
    MyLogUtil.log(ACTION_GET_RESPONSE_CODE + responseCode);
} catch (IOException e) {
    e.printStackTrace();
}

if (connection != null) {
    try (BufferedReader reader = new BufferedReader(new InputStreamReader(responseCode <= 300 ? connection.getInputStream()
        //Retrieve the result of the http request and pass it into the StringBuilder
        String line = reader.readLine();
        while (line != null) {
            stringBuilder.append(line).append("\r\n");
            line = reader.readLine();
        }
        String json = stringBuilder.toString();
        if (responseCode <= 300) {
            MyLogUtil.log(ACTION_GET_SUCCESS_RESPONSE + json);
            new Handler(Looper.getMainLooper()).post(() -> callback.onSuccess(json));
        } else {
            MyLogUtil.log(ACTION_GET_ERROR_RESPONSE + json);
            new Handler(Looper.getMainLooper()).post(() -> callback.onFailure(json));
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            connection.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Figure 13: HTTP request error handling

7.4. Coding Standards and Conventions

For any piece of commercial software, it is crucial that the code is written to a high standard. It was established early on in the project that our solution must conform to the standards set in ISO 25010. The ISO 25010 quality model consists of eight main attributes, each with its

own set of sub-attributes:

- **Functional suitability**
  - Completeness – covers all specified tasks and user objectives.
  - Correctness – provides correct results with required degree of precision.
  - Appropriateness – functions facilitate the accomplishment of user objectives.
- **Performance efficiency**
  - Time behaviour – response and processing times meet requirements.
  - Resource utilization – amounts and types of resources used meet requirements.
  - Capacity – maximum limits meet requirements.
- **Compatibility**
  - Co-existence - can perform efficiently while sharing environment with other products.
  - Interoperability – system can exchange information effectively with other systems.
- **Usability**
  - Learnability – can be learned with satisfaction and freedom from risk.
  - Recognizability – users can recognise that system is appropriate to their needs.
  - Operability – easy to operate and control.
  - User error protection – protects users against making errors.
  - Aesthetics – pleasing and satisfying user interaction.
  - Accessibility – can be used by people with a wide range of characteristics such as impaired vision.
- **Reliability**
  - Maturity – reliable under normal operating conditions.
  - Availability – operational and accessible when required for use.
  - Fault tolerance – operates as intended despite presence of hardware and/or software faults.
  - Recoverability – can recover data affected by failure or interruption and re-establish desired state of system.
- **Security**
  - Confidentiality – ensures that data are accessible only to authorised users.
  - Integrity – prevents unauthorized access.
  - Non-repudiation – actions or events can be proven to have taken place.
  - Accountability – actions of an entity can be traced uniquely to that entity.
  - Authenticity – identity of a user or resource can be proved to be the one claimed.
- **Maintainability**
  - Modularity – composed of discrete components so that a change to one has minimal effect on others.
  - Reusability – components can be reused in other systems.
  - Analysability – easily diagnosable.
  - Modifiability – can be modified effectively without introducing faults.
  - Testability – test criteria can be established, and tests can be performed efficiently.
- **Portability**
  - Adaptability – can be adapted for different use cases.
  - Installability – can be successfully installed in a specified environment.
  - Replaceability – can replace another specified software product for the same purpose.

Applying Code Quality Standards to FFsmart

ISO 9126 and ISO/IEC 25010 are both standards that provide guidelines for software quality evaluation. ISO/IEC25010 also provides a methodology for assessing workflows and procedures. The standards check for quality in a variety of areas: performance functionality, reliability, compatibility, usability, security, and maintainability.

For functionality, the team started with a clear definition of the requirements and user expectations. Implementing the desired features requires a thorough comprehension of the demands of the users and the functions they need to carry out using the software. Clear and concise standards and guidelines have been embedded in the software's design and development to ensure high software quality and to help achieve these objectives.

Regarding performance, the backend of the application has been tested under high stress as well as under normal conditions. It showed good results with reasonable response times, although with scaling better times could be achieved. In terms of portability, it must be ensured that the software can run on different hardware and software platforms. This was one reason we chose to work with Java, as it will allow a high variety of users to be able to use the application. For maintainability, this involves making sure that the system can be easily modified, updated, and repaired, and that it is designed with interfaces and a modular structure to allow for easier future modification.

Regular testing, monitoring, and assessments are essential to ensuring that the software product satisfies these criteria by spotting and resolving any flaws before release. Software has strong usability features because it has considered a variety of elements, including navigation, user interface design, and overall user experience, the user interface is simple to use and well-organized, with labelling and directions that are easy to understand. Navigation is simple and straightforward, and behaviour is consistent and predictable. Other factors that affect usability are error handling, assistance and support, and accessibility. Error management should be brief and straightforward, and it should give users the knowledge and direction they need to fix any problems. For the software to be relevant and continue to satisfy the demands of its users throughout time, it must also be regularly updated and maintained.

SOLID principles

The SOLID principles are a popular coding standard in object-oriented design and are used extensively in the software industry. Following these principles results in higher code quality and greater maintainability. Throughout the development phase of the project, care was taken to ensure each part of the code followed the following principles:

- Single Responsibility Principle – each class should do one thing and should only have a single reason to change.
- Open-Closed Principle – classes should be open for extension and closed to modification (able to add new functionality without touching existing code)
- Liskov Substitution Principle – subclasses should be substitutable for their base/parent classes (child classes extend parent behaviour and never narrow it down)
- Interface Segregation Principle – a client should not be forced to implement functionality that it doesn’t need (separate interfaces for different clients)
- Dependency Inversion Principle – classes should depend on interfaces or abstract classes instead of concrete classes and functions.

SonarLint

The IntelliJ SonarLint plugin, which is an agent of SonarQube and works with both Java and Android, was used for assessing and improving code quality. SonarLint finds bugs, vulnerabilities and "code smells", and can provide quick fixes and guidance for remedying issues. It grades issues on a severity scale from minor to critical. The tool was used to aid the development process and it was ensured that all issues raised by SonarLint were fixed before submission of the code. An example usage on a server file (ResponseHandler.java) is shown below.

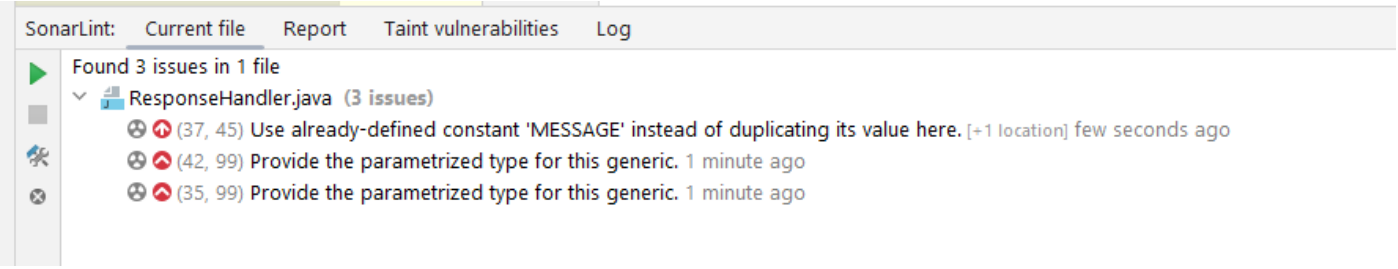


Figure 14: SonarLint IntelliJ Linting Example 1

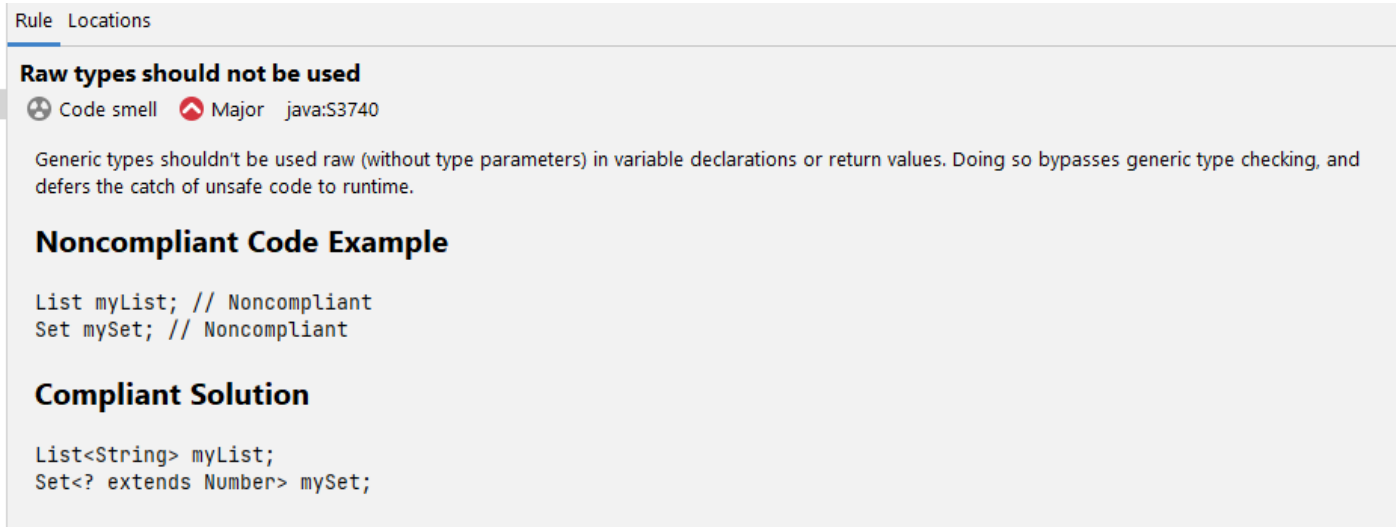


Figure 15: SonarLint IntelliJ Linting Example 2

All code for the Android client was evaluated using the SonarLint plugin in android studio. Changes were made based on the suggestions made. For example, in Figure 16 & 17, the click function for the user information icon in the UserAccountActivity class was suggested by SonarLint to use Lambda expressions instead of anonymous inner classes. The use of Lambda expressions(Figure 18)effectively reduces the complexity of the code while improving its readability.

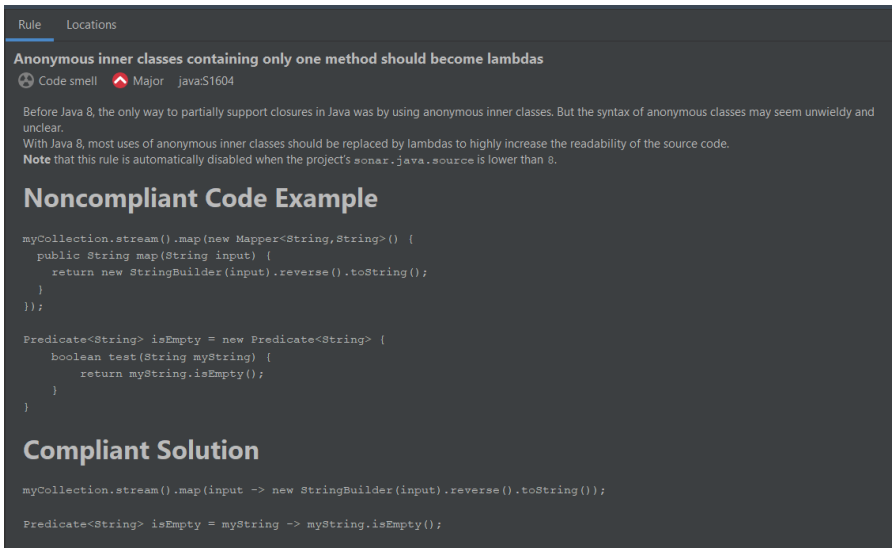


Figure 16: Android client Sonarlint issue

```
helper.getView(R.id.iv_infoUser).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent();
        intent.setClass(context, UserManageActivity.class);
        intent.putExtra( name: "Manage_userId", user.getId());
        startActivity(intent);
    }
});
```

Figure 17: Android client Sonarlint issue code(Before changed)

```
helper.getView(R.id.iv_infoUser).setOnClickListener(v -> {
    Intent intent = new Intent();
    intent.setClass(context, UserManageActivity.class);
    intent.putExtra( name: "Manage_userId", user.getId());
    startActivity(intent);
});
```

Figure 18: Android client Sonarlint issue code(After changed)

7.5. External Modules

Group ID	Artifact ID	Version	Usage
Server			
io.jsonwebtoken	Jjwt-api	0.11.2	JWT authorization
com.jayway.jsonpath	json-path	2.4.0	Deserializing JSON
com.github.librepdf	openpdf	1.3.30	PDF generation
Client			
com.google.code.gson	gson	2.10	Deserializing JSON
com.blankj:utilcodex	SPUtils	1.31.1	Shared Preferences Access
com.github.barteksc:android-pdf-viewer	pdfviewer	3.2.0-beta.1	PDF reader
com.github.bumptech.glide	glide	4.11.0	User Avatar

8. User Help Documentation

The system was designed with the fundamentals of portability in mind to create adaptable, reliable, maintainable code. Part of this philosophy is ensuring that the programs involved in the system can successfully be installed and ran in different environments, and on machines with different hardware specifications. This is important not only to the end users of FFsmart, but also to the developers and testers needing to collaborate to write and test the code. When writing and testing the installation of the system, errors and bugs arose on different machines and networks that have now been fixed, ensuring enhanced compatibility and fewer issues to clients. Running the system on multiple hardware setups also led to running into common installation problems and allowed the User Help documentation to be improved to incorporate solutions.

8.1. Installation Guide

8.1.2. Git

Git is a widely used version control tool that allows developers to collaborate on source code during software development. It was used throughout the development of the project by the coders and tester to support their workflow and allow work done individually or together to be integrated cleanly. The system is currently available to be downloaded from GitHub, the most widely used Git repository.
GitHub link: `https://olympuss.ntu.ac.uk/T0268816/Group_11_SOFT30121_AAD`

To clone the code repository, Git should be installed, to ensure that the user is able to easily push and pull changes. Go to the Git website and download the latest version for your operating system.
Git link: `https://git-scm.com/`

Once downloaded, run the executable, and follow the installation instructions provided. Once installed, launch the command prompt (or terminal for non-Windows users) and navigate to the directory where you want to clone the project, for example '`cd Documents/FFsmart`'. After changing your working directory, enter:

```
git clone https://olympuss.ntu.ac.uk/T0268816/Group_11_SOFT30121_AAD.git
```

The repository has now been cloned to your computer and you can begin to work with code. To keep your repository up to date with the latest changes, use '`cd`' as before to navigate to your repository (such as `cd Documents/FFsmart/Group_11_SOFT30121_AAD`' and run the following command:

```
git pull origin master
```

This will download the latest version and update your local copy.

If Git or an alternative cannot be installed, then a copy of the current version can be obtained by clicking the code button and selecting download as ZIP. Once downloaded, the ZIP file should be moved to the desired directory (such as `Documents/FFsmart`) and extracted. The files would be the same as if the project was cloned using Git, however it will be unable to give or provide updates to the codebase.

If you encounter any technical difficulties, refer to Git/GitHub's documentation.

8.1.3. Java

The team elected to use IntelliJ due to already being comfortable using it and its relation to Android Studio. However, any IDE capable of running Java projects such as Eclipse or NetBeans can also be used. You will likely already have Java installed as it is required for the running of many applications. If not, it is available for download from Oracle's website. Upon downloading, run the executable and follow the provided instructions.
Java link: `https://www.java.com/en/download/`

Run your IDE of choice and select Open Project, navigate to the directory you cloned the repository to and select FFsmart to import the project. Once this has been done, build the project. After building is complete navigate to the main server class. The full file path is:

```
Group_11_SOFT30121_AAD\ffsmart\src\main\java\com\aad\ffsmart
```

Once this file has been located within your IDE, open and run it. The server will now start running, which you should be able to see in your output window.

If you encounter any technical difficulties, refer to your chosen IDE's documentation.

8.1.4. Android Studio

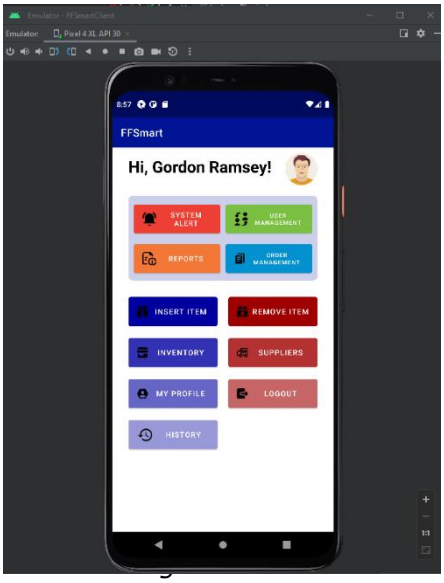
Because the development team wanted to make the software as applicable to real-world scenarios as possible, they elected to create the client for mobile phones, thus needing Google's Android Studio. Android Studio is the official IDE for the Android operating system and is built on IntelliJ's existing software, making them perfect to partner together. First, the latest version of Android Studio must be downloaded from their website (currently Electric Eel).
Android Studio Link: `https://developer.android.com/studio`

After it has finished downloading, run the executable and follow the on-screen instructions, including setting up the Android SDK (Software Development Kit). No plugins are necessary to run the project.

Open the project by navigating to the location of the repository in Android Studio and selecting the client folder. The icon for the folder may be a green Android face. After opening the project, build it, and navigate to the main application. The full file path:

```
Group_11_SOFT30121_AAD\client\app\src\main\java\com\ffsmartclient
```

After locating it, build it, and attempt to run it. At this stage you may be prompted to set up an Android emulator.



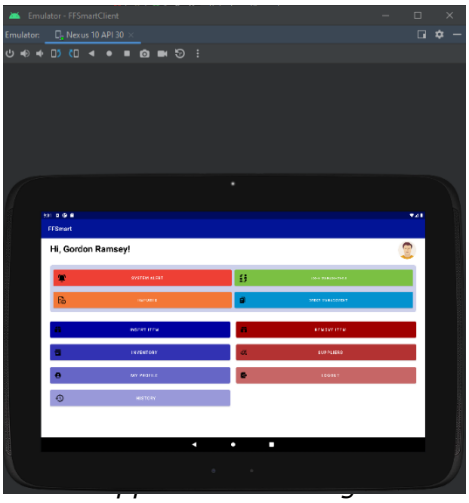
8.1.5. Android Emulator

Next, an Android Emulator will be required to simulate the application's usage in real-world scenarios. Some users will be able to select the device manager, create devices, and follow the on-screen prompts to create an emulation of any sized phone, tablet, or television. However, if you have issues setting up an Android Emulator through Android Studio you may have to download alternative emulator software such as BlueStacks. Some common issues and solutions can be found in the relevant section.

BlueStacks provides a variety of different options for emulating Android devices, their products are primarily intended for gaming but can also be used for this purpose. From their website download one of the versions of their emulator application, such as BlueStacks 5. Run the executable and follow the on-screen instructions to set it up.
BlueStacks link: `https://www.bluestacks.com/`

Once BlueStacks has been installed, open the settings, select advanced, and enable Android Debug Bridge (ADB). This will allow it to be paired with Android Studio and used as the client.

If you encounter any technical difficulties, refer to the Common Issues section or the BlueStacks documentation.



Nexus 10 tablet.

8.1.6. Directly on Android

If neither of these solutions are viable, Android Studio has an experimental option whereby a smartphone connected via Wi-Fi or USB cable is able to be tested on directly. Either of these options will require the smartphone to have Android Developer Bridge enabled in the developer settings. Like all experimental and beta features, there may be unforeseen bugs and errors associated with this option. Where possible it is preferred to run a local emulator.

8.1.7. Common Issues

Hardware Virtualization

Some users with AMD CPUs may find themselves unable to run the Android Emulator in Android Studio, and it is recommended that all users follow this step regardless of their processor drivers for enhanced performance.

- Activate Hardware Virtualization in their BIOS.
- Turn off Hyper-V and Hypervisor Platform.

Hardware Virtualization is a technology that allows a single physical computer to run many

operating systems simultaneously as if each were running on a separate. This technology is integrated into the hardware and firmware of the computer and offers the operating systems an abstracted layer of virtualized hardware resources, including the CPU, memory, storage, and network. Hardware Virtualization tends to be disabled by default as it can cause performance or compatibility issues with older or lower-end hardware, and as it isn't required by Hyper-V, it is usually only enabled on a case-by-case basis when the user of a machine requires it.

Hyper-V is Windows' native hypervisor technology, which allows for the creation and management of virtual machines. With Hardware Virtualization enabled, it is recommended to disable Hyper-V, as they compete for the same resources as other virtualization techniques and can cause compatibility issues when both are running. Some Windows features may rely on Hyper-V; therefore care should be taken before enacting this step by researching potential issues and backing up your data and configuration.

Users may still be able to run Android Emulators without following these steps, but it may hinder the performance of the application and negatively impact the user's experience.

Version

Any recent Java Development Kit (JDK) and recent Android Software Development Kit (SDK) should enable the application to work perfectly, however in testing JDK versions 17.0.5 and versions 19.0.2 were used, as well as SDK versions 11, 12 and 13. It is always recommended to use the latest stable build for your computer, ensuring to select the right version for your hardware architecture. If the latest stable build is not working for any reason, please try JDK version 19.0.2 and SDK version 11.

System Requirements

Android Studios list their system requirements as a 2nd generation Intel Core CPU or AMD CPU with support for Windows Hypervisor Framework, as well as a 64-bit Operating System and 8GB of RAM. Some older or weaker devices may struggle to meet these requirements and may therefore have difficulty using the app in its current form. Unfortunately, any users whose machines do not meet these requirements may have unforeseen difficulties with running the application such as impacted performance.

Firewall

Some firewall setups may block the required software and dependencies from downloading necessary components and updates. In this scenario, it is important to add exceptions to the firewall settings to install the software without issues.

FUTURE FRIDGE LTD.

A New Type of Smart Fridge to Storing Food

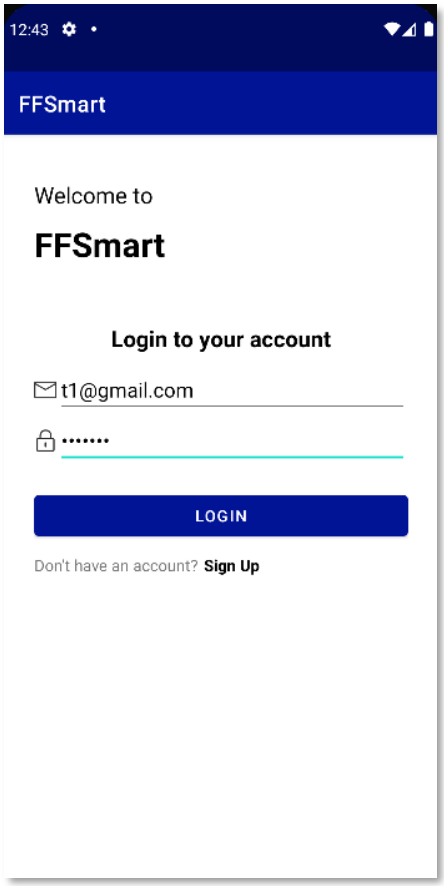
FFsmart User Guide

Table of Contents

Chapter

1

8.2. Generic Guide

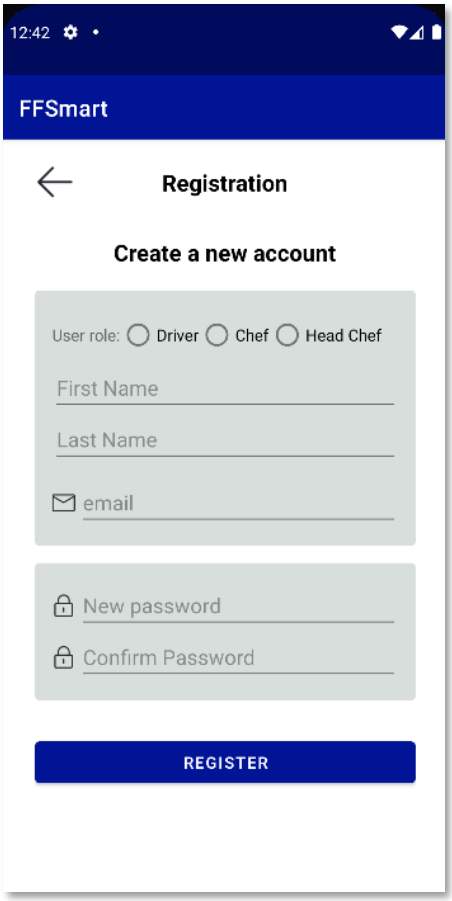


8.2.1. Login Page

- Enter the correct user account email and password on the login page and click on the "Login" button to access the home page.
- Click on "Sign Up" below the login button to go to the Registration page to create a new user account.

Note:
Email must be a valid email format, e.g., user@email.com

Password should be a combination of 6 to 10 digits and letters.



8.2.2. Registration Page

- After completing the first name, last name, e-mail, and password and choosing the user's role, click on the "Register" button to create a new user account. Once registered will access the home page.

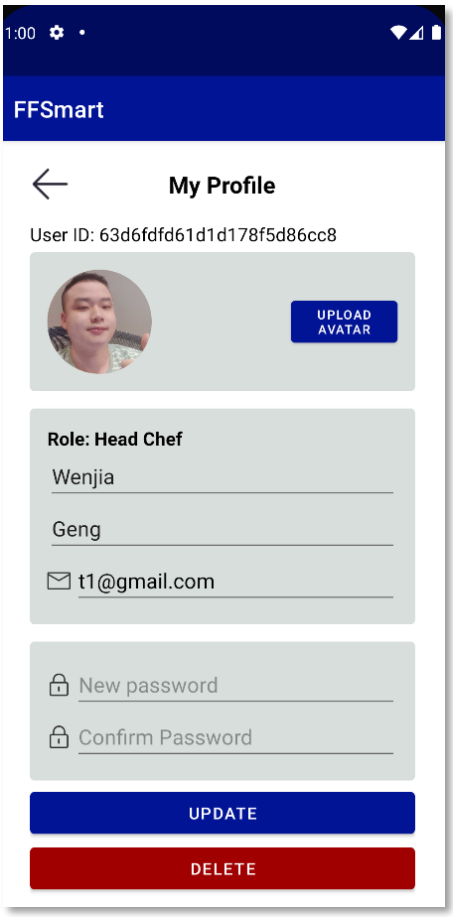
Note:
Missing information will fail to register.

The password needs to keep the same for both entries.

User Role Types:
1. Driver
2. Chef
3. Head Chef

8.2.3. Logout Page

- Click on the avatar in the top right-hand corner to access the My Profile page.
- Click on the "Logout" button in the bottom right-hand corner to log out from the current account and return to the login page.



8.2.4. My Profile Page

On the My Profile Page, the account's first name, last name, email, and password can be changed, where the password needs to be entered twice to be confirmed.

- Click the "Upload Avatar" button to select an avatar image from the system album.
- Once the account information has been modified, click the " Update" button to update the account.
- Click the "Delete" button to delete the current account and return to the Login Page.

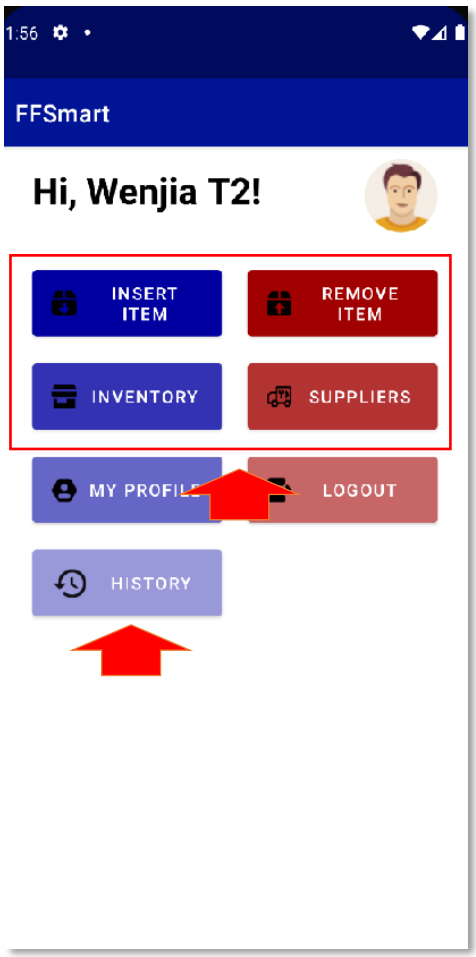
Note:
The User ID is displayed above the avatar section.

The user's role cannot be changed once the account has been created.

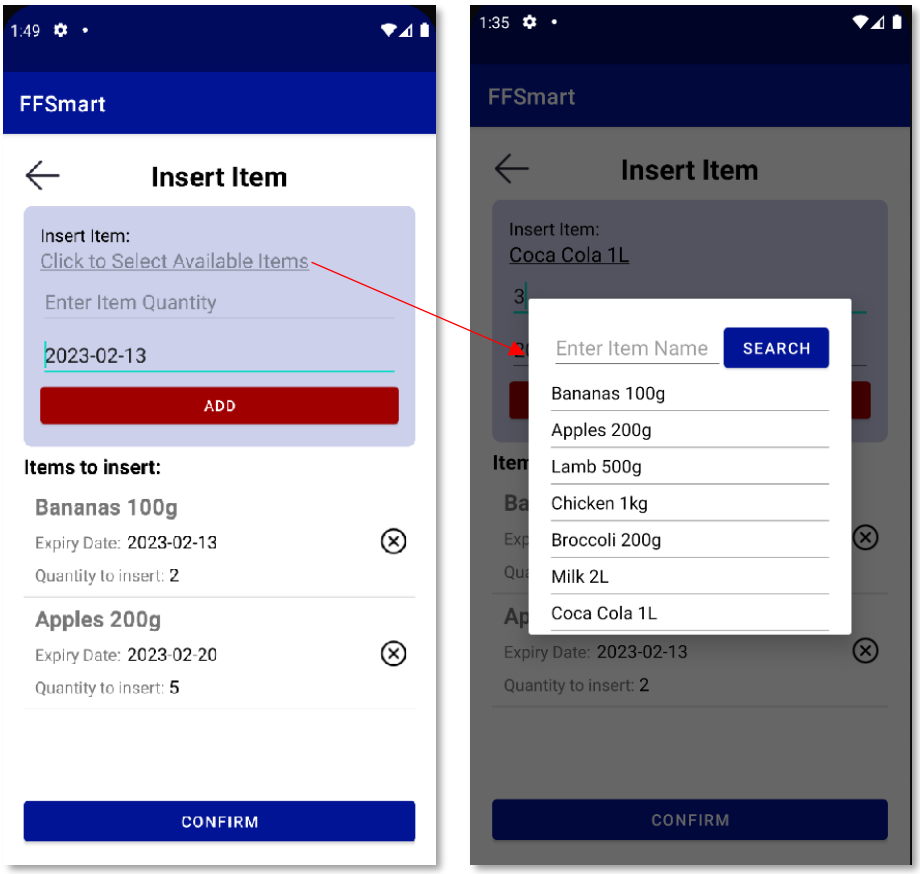
Chapter 2

8.3. Chef Role Guide

8.3.1. Chef Home Page



- Click on the "Insert item" button to go to the Insert Item Page to add new items to the fridge.
- Click on the "Remove item" button to go to the Remove Item Page to take inventory items out of the fridge.
- Click on the "Inventory" button to go to the Inventory Page to check the items available in the fridge inventory.
- Click on the "Suppliers" button to go to the Supplier Page to see the suppliers for the restaurant item.
- Click on the "History" button to go to the Inventory History Page to check the history of all inventory item changes.



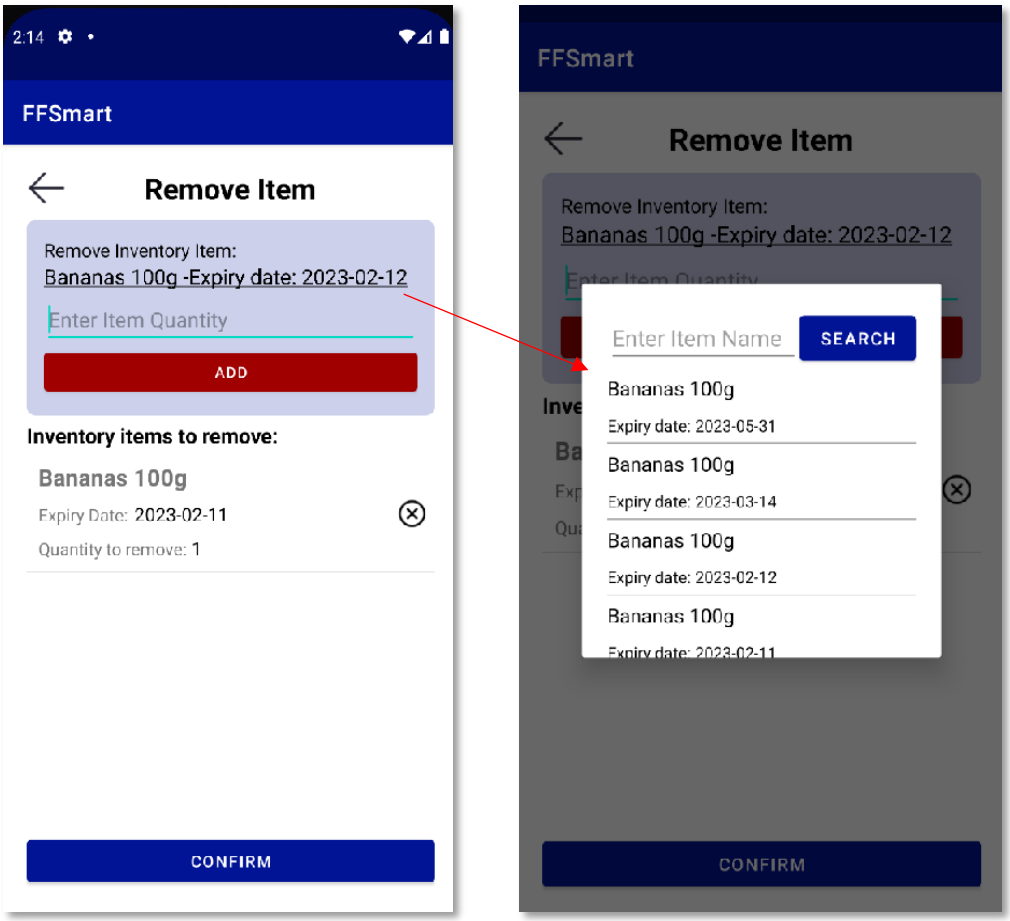
8.3.2. Insert Item Page

Multiple items can be inserted into the fridge simultaneously on the Insert Item Page.

- Click on the button under "Insert item" to check the currently available items in the pop-up window. In the window, the relevant item can be searched by entering the item name and selected by clicking on the item in the list. After entering the item quantity and expiry date, click the "Add" button to add the item to the list of inserted items.
- Click the "Delete" icon on a list item to remove the item from the list of inserted items.
- Click on the "Confirm" button to insert the items currently in the inserted items list into the fridge's inventory database once the items have been added to the fridge.
- Clicking on the "Back" button returns to the Home Page.

Note:

The list of un-inserted items will be cached, and the unconfirmed inserted items can be modified when re-entering the Insert Item Page.



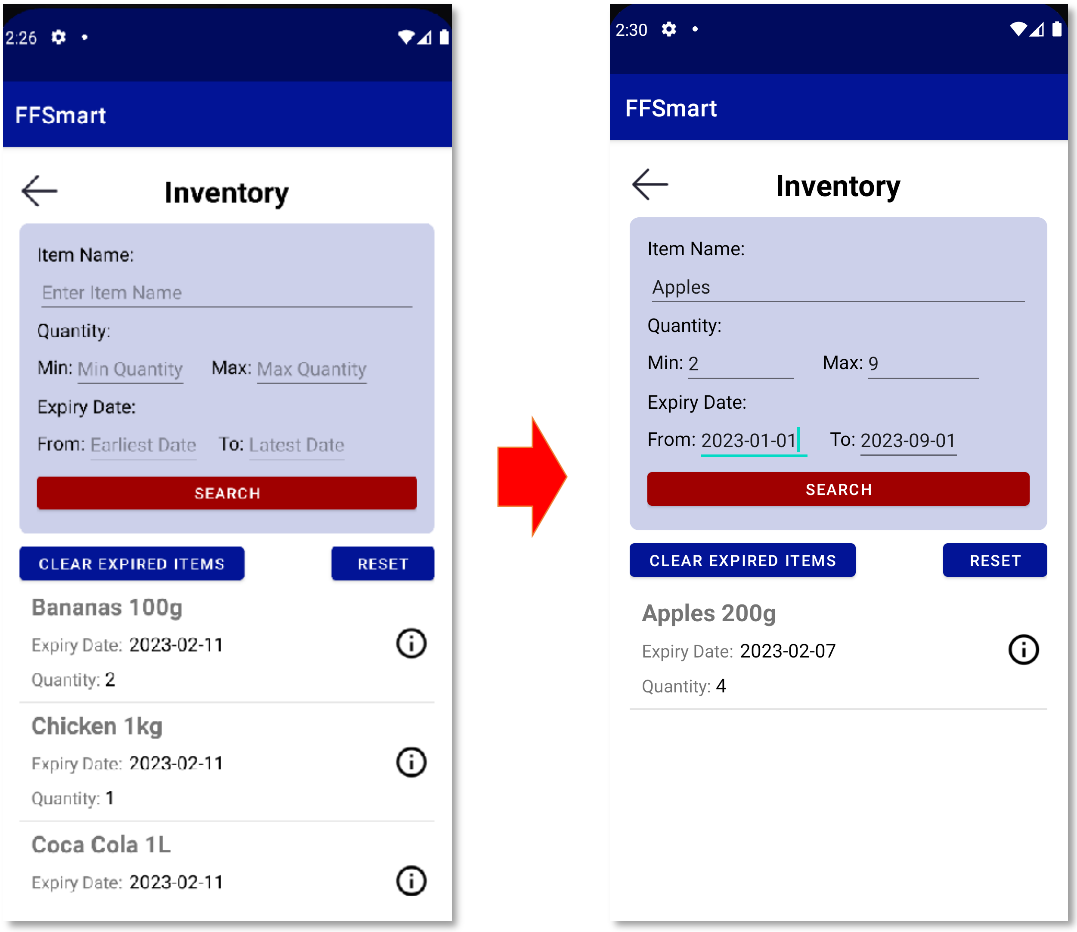
8.3.3. Remove Item Page

Multiple items can be removed from the fridge simultaneously on the Remove Item Page.

- Click on the button under "Remove inventory item" to check all inventory items stored in the fridge from the pop-up window. In the window, the relevant inventory item can be searched by entering the item name and selected by clicking on the item in the list. After entering the item quantity, click the "Add" button to add the inventory item to the list of removed items.
- Click the "Delete" icon on a list item to remove the item from the list of removed items.
- Click on the "Confirm" button to remove the items in the removed items list from the fridge's inventory once the item has been taken out of the fridge.
- Click on the "Back" button to return to the Home Page.

Note:

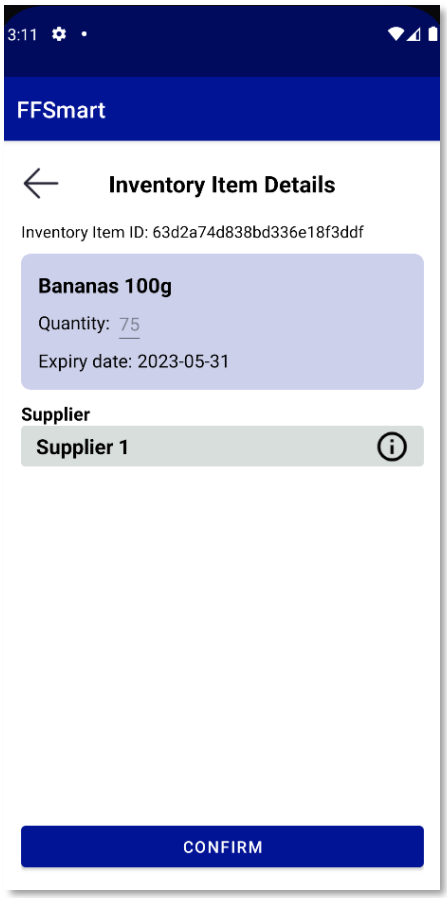
The list of un-removed items will be cached, and the unconfirmed removed items can be modified when re-entering the Remove Item Page.



8.3.4. Inventory Page

In the Inventory Page, all the inventory items in the fridge can be checked, and the items can be searched for by entering the item's name, quantity, and expiry date.

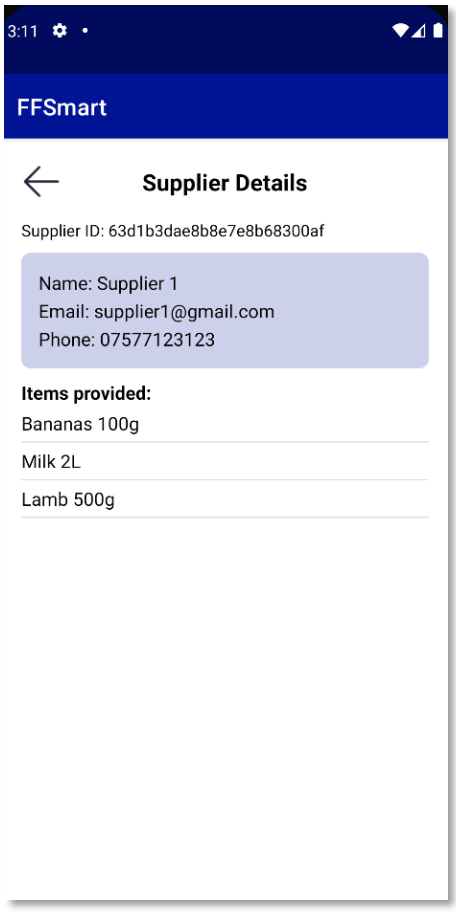
- Enter the item's name, the maximum/minimum quantity, and the expiry date range to set the conditions for item searching. Click on the "Search" button to search for items. The items that match the requirements will be displayed in the list at the bottom of the page. Click on the "Information" button for the item in the list to go to the Inventory Item Page to see the item's details.
- Click on the "Clear expiry items" button to remove all the expired items from the fridge inventory, after taking the expired items out of the fridge.
- Click on the "Reset" button to reset the search conditions, and the list will display all the items in the inventory by default, sorted by order of insertion.
- Click on the "Back" button to return to the Home Page.



8.3.5. Inventory Item Details Page

The inventory item ID, item name, item quantity, item expiry date, and item supplier information can be checked on the Inventory Item Details Page.

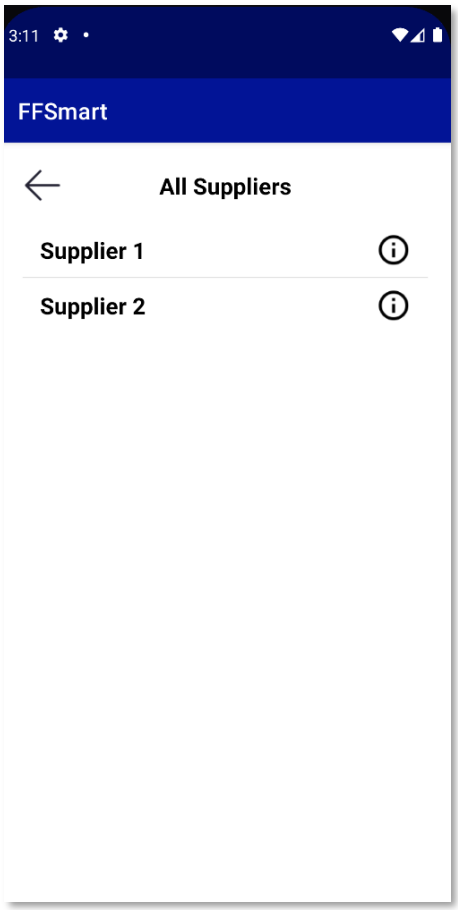
- Enter the number to be updated in the quantity input field and click on the "Confirm" button to complete the update of the item's stock quantity.
- Click on the "Information" button at the right of the supplier's name to go to the Supplier Details Page to check the supplier's details.
- Clicking on the "Back" button returns to the Inventory Page.



8.3.6. Supplier Details Page

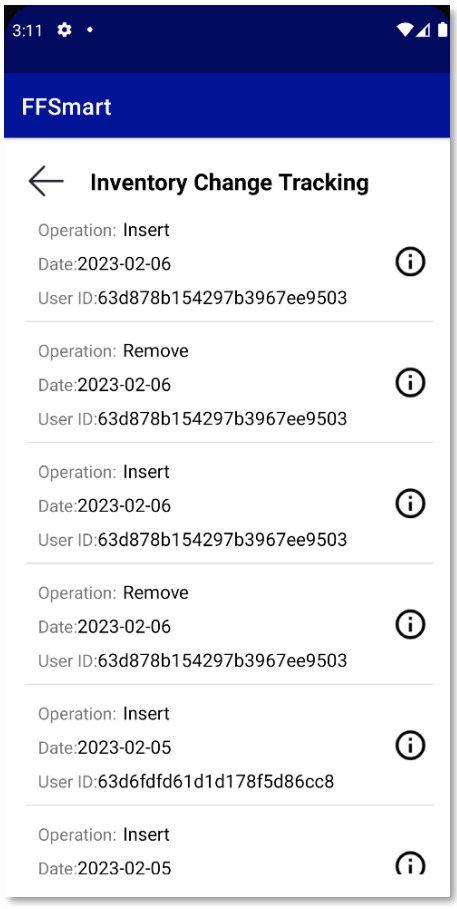
The supplier ID, supplier's name, email address, contact number and items supplied can be checked in the Supplier Details Page.

- Clicking on the "Back" button returns to the Inventory Item Details Page.



8.3.7. Suppliers Page

- The Supplier Page provides all the suppliers who are supplying items to the restaurant, and the supplier details page can be accessed by clicking on the supplier's "Information" button.
- Clicking on the "Back" button returns to the Home Page.

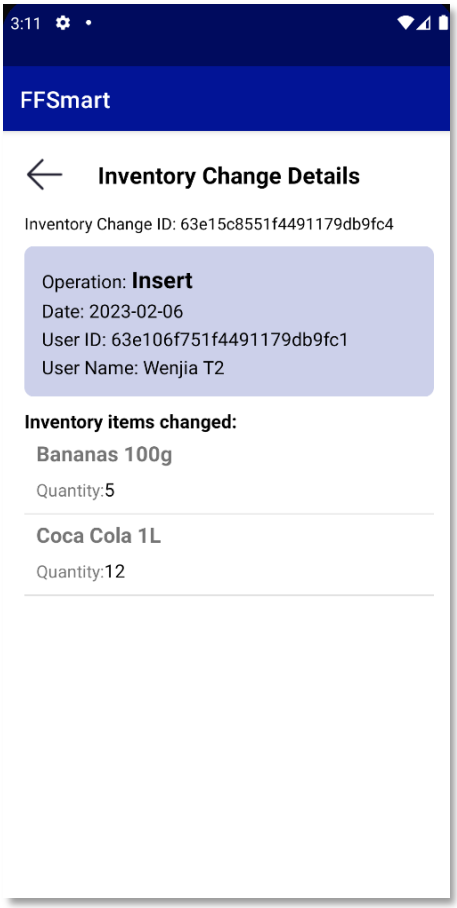


8.3.8. Inventory Change History Page

The record of inventory changes can be checked on the Inventory Change History Page, where each record contains the operation, operation date, and operator user ID.

- Click on the "Information" button for the record to go to the Supplier Details Page to see the supplier's details.
- Clicking on the "Back" button returns to the Home Page.

Note:
Operation types：**1. Insert 2. Remove**



8.3.9. Inventory Change Details Page

The inventory change ID, the operation type, the operation date, the operator ID, and the name can be checked on the Inventory Change Details Page.

The name and quantity of the items inserted/removed can be checked in the inventory items changed list at the bottom of the page.

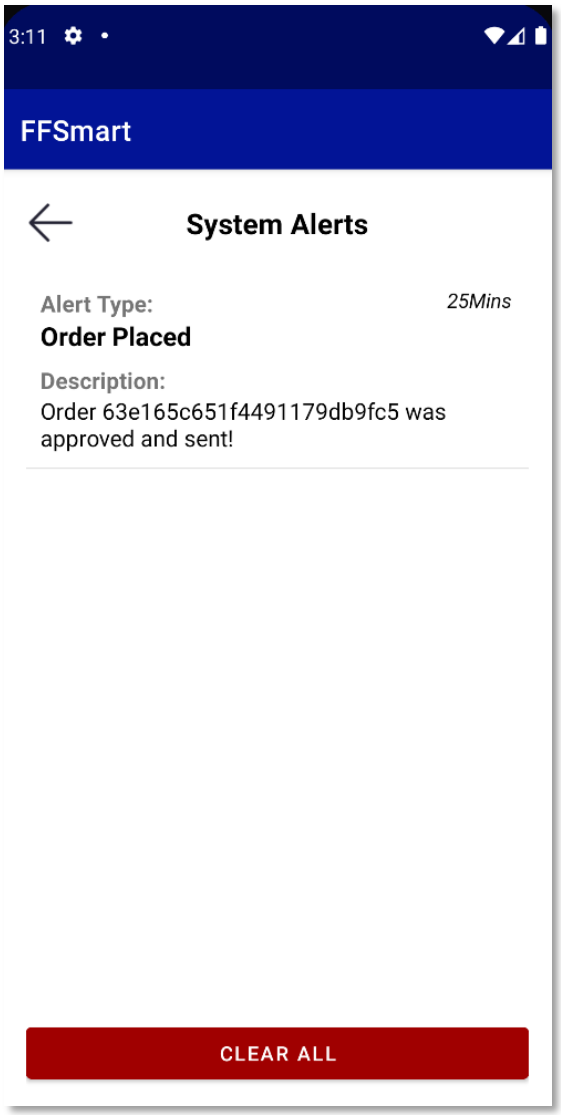
- Click on the "Back" button to return to the Inventory Change History Page.

Chapter 3

8.4. Head Chef Role Guide

8.4.1. Head Chef Home Page

- Click on the "System Alert" button to go to the System Alerts Page to check the System alerts received.
- Click on the "User Management" button to go to the User Accounts Page to check all user accounts on the system.
- Click on the "Reports" button to go to the Health & Safety Reports Page to check the reports the system has sent to the safety officer.
- Click on the "Order Management" button to go to the All-Orders Page to check orders in the system.



8.4.2. System Alerts Page

On the System Alerts Page, the alert type, the description of the alert event, and the time of receiving the alert can be checked.

- Click on the "Clear all" button at the bottom of the page to clear all the alerts received.
- Click on the "Back" button to return to the Home Page.

Note:

Types of Alerts:
Expired Items: There are expired items in the inventory.

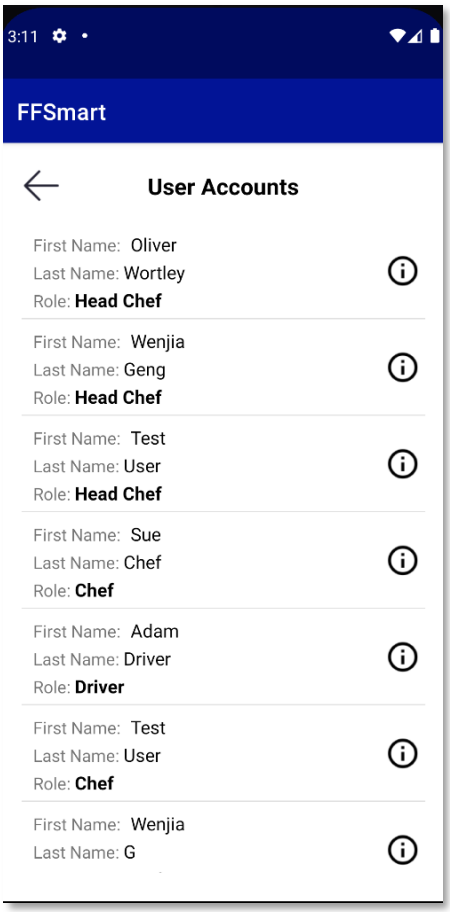
Low Quantity Items: Items with low quantity in the inventory

Order Ready: An order has been generated and is waiting to be approved by the Head Chef.

Order Placed: An order has been approved by the head chef and sent to the supplier.

Order Delivered: An order has been delivered by the driver.

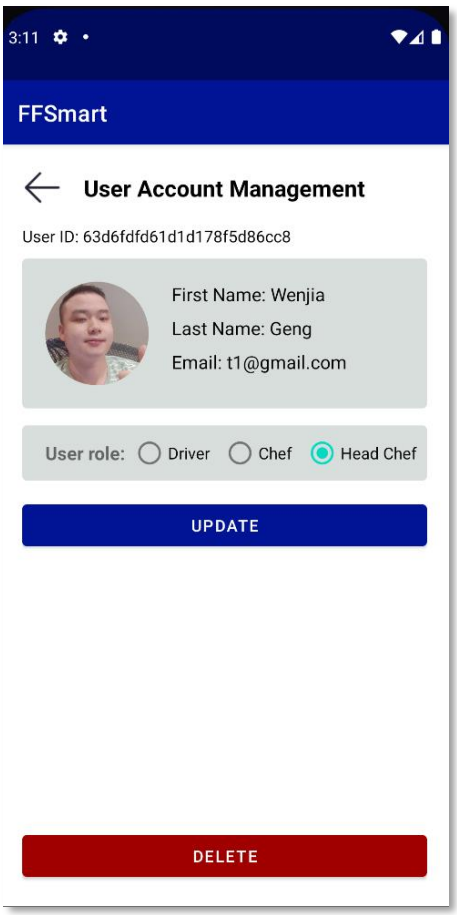
Order Check Successful: The items in the order have been checked and put in the fridge.



8.4.3. User Accounts Page

All user accounts can be checked on the User Accounts Page, which includes each account's first name, last name, and user role.

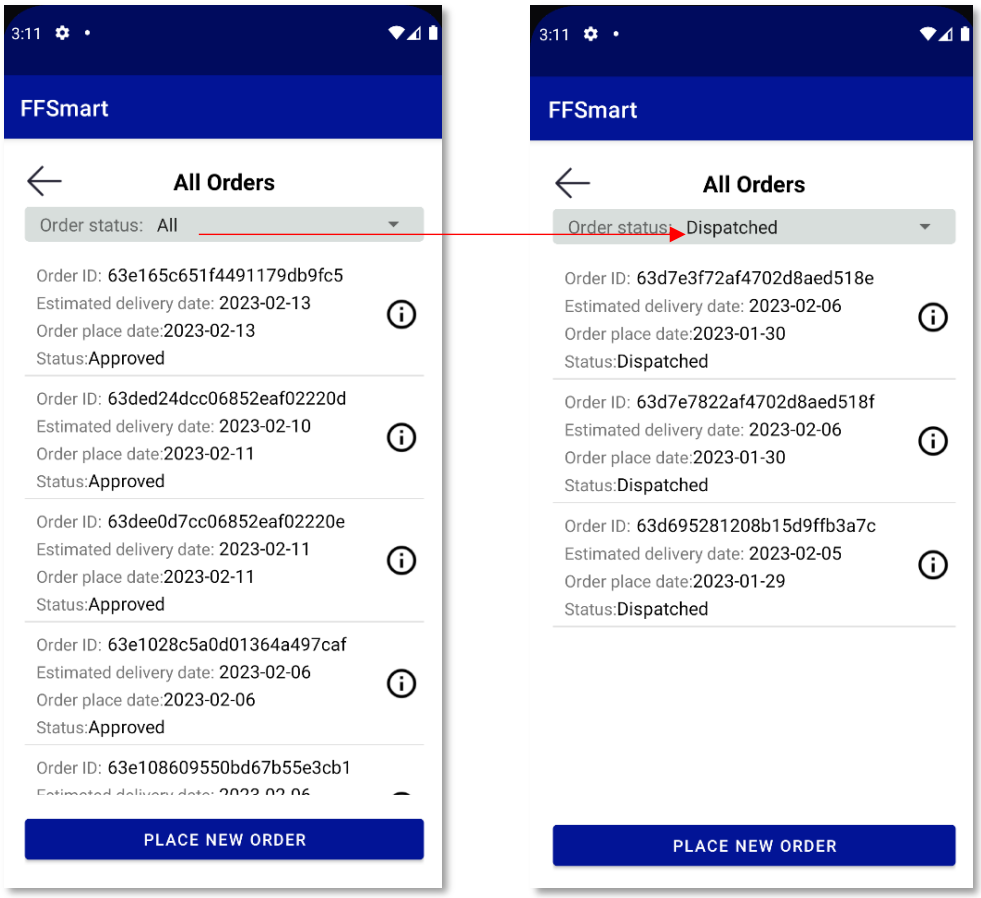
- Click on the information button for each account to go to the User Account Management Page and check the user account details.
- Click on the "Back" button to return to the Home Page.



8.4.4. User Account Management Page

The user ID, first name, last name, email address and user role can be accessed on the User Account Management Page.

- Select the user role and click on the "Update" button to change the user role.
- Click on the "Delete" button at the bottom of the page to delete the current user account and return to the User Accounts Page.
- Click on the "Back" button to return to the User Accounts Page.



8.4.5. All Orders Page

On the All Orders Page, all orders in the system can be viewed, showing the order ID, estimated delivery date, order placed date, and order status for each order.

- Click on the "Information" button for the order to go to the Order Details Page to check the order details.
- Click on the "Place new order" button at the bottom of the page to go to the Place Order Page to place a new order.
- Click on the "Back" button to return to the Home Page.

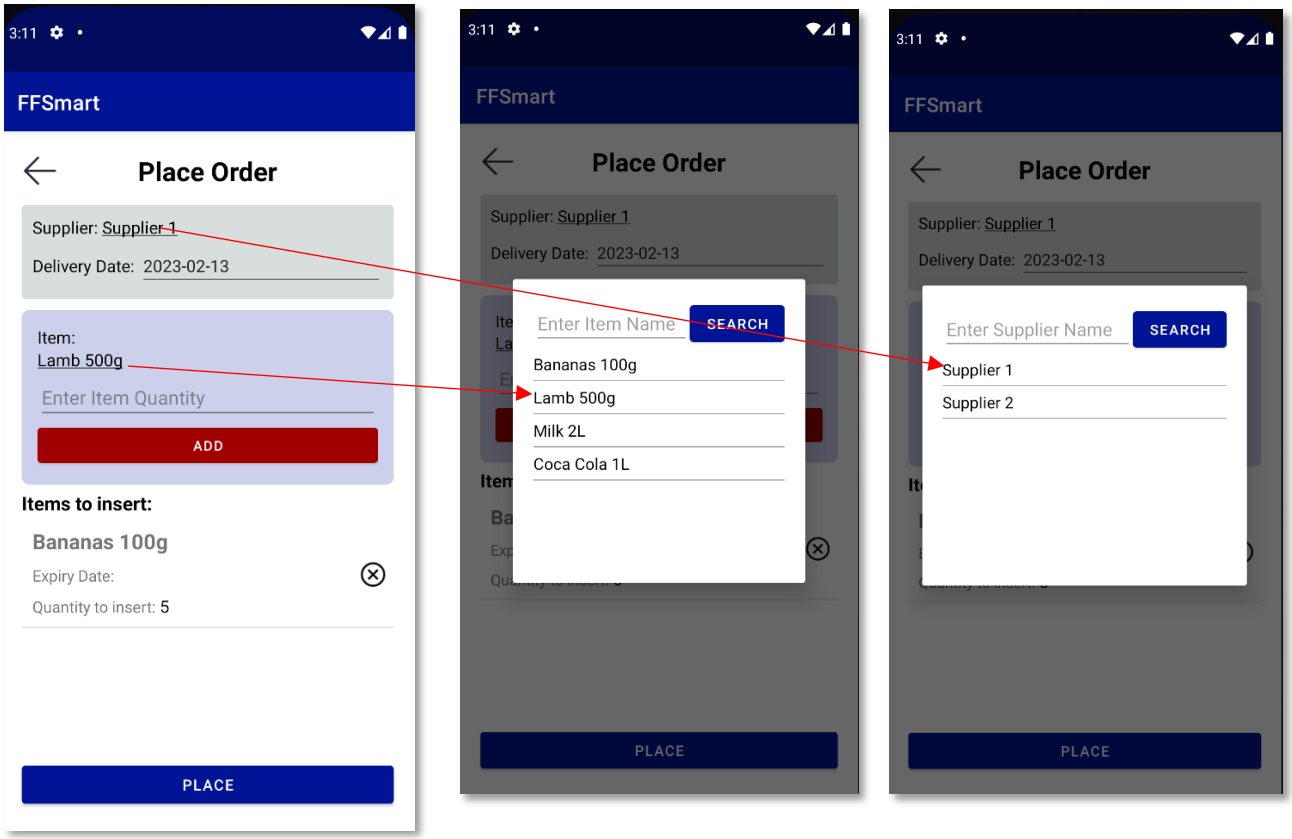
Note:
Order statuses:

1. **Waiting for approval**: Waiting to be approved by the Head Chef

2. **Approved**: Approved by the Head Chef and sent to the supplier

3. **Dispatched**: Has been accepted by the Driver and is being shipped to the restaurant

4. **Delivered**: Has been delivered to the restaurant, and the items in the order have been put into the fridge.



8.4.6. Place Order Page

New orders can be placed with suppliers on the Place Order Page.

1. Click on the button to the right of the "Supplier" to see the currently available suppliers in a pop-up window, enter the supplier's name to search for the relevant supplier and click on the supplier in the list to select one.

2. Enter the date in the delivery date field below to assign a delivery date for the order.

3. After selecting the supplier, click on the button below the "Item" to check the list of items supplied by the selected supplier in a pop-up window, enter the item name to search for the relevant item and click on the item in the list to select one.

4. Once the quantity has been entered, click the "Add" button to add the item to the order items list.

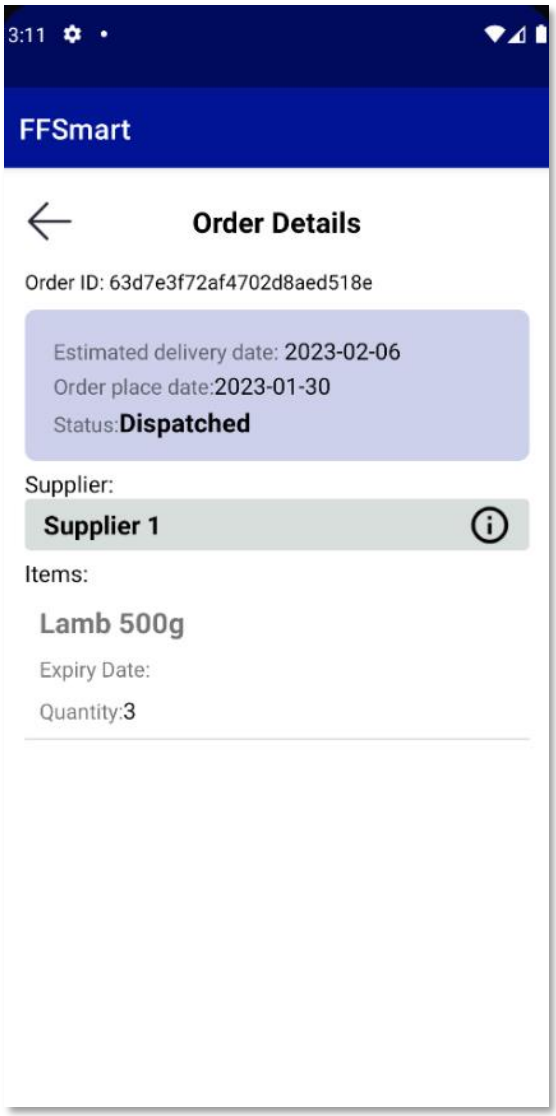
5. Click on the "Delete" button for the item in the list will remove the item from the order items list.

6. Lastly, click the "Place" button to send the order to the supplier once the items required for the order have been added.

- Click on the "Back" button to return to the Home Page.

Note:
After changing the selected supplier, the list of previously selected order items will be cleared.

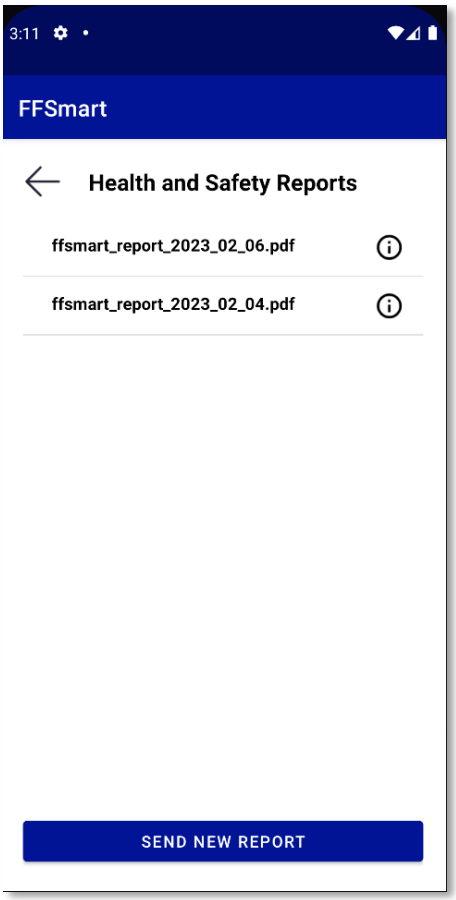
If there is an unplaced order, the order will be cached and can be edited when re-entering the Place Order Page.



8.4.7. Order Details Page

The Order Details can be accessed on the Order Details Page, including order ID, estimated delivery time, order place date, order status, order supplier, and items in the order.

- Click on the "Information" button for the supplier to go to the Supplier Details Page to check the supplier's details.
- Click on the "Back" button to return to the All-Orders Page.

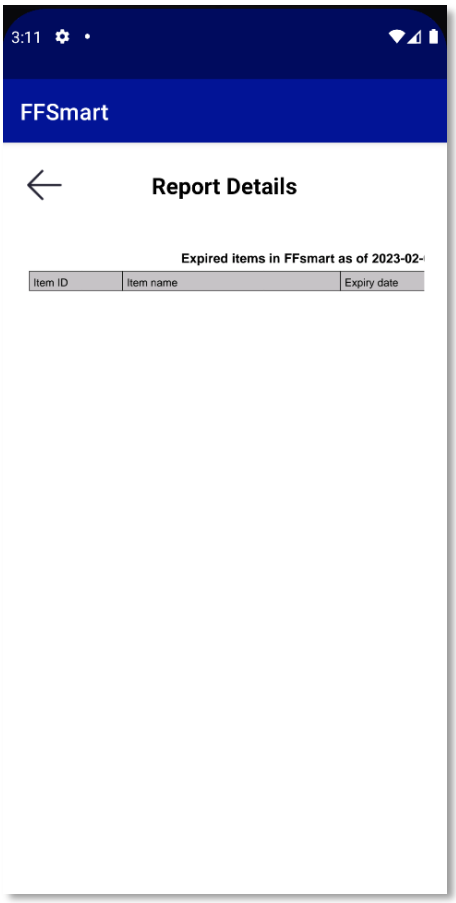


8.4.8. Health & Safety Reports Page

All reports generated by the system and sent to the safety officer can be checked on the Health & Safety Reports Page.

The report name will indicate the report's date of generation.

- Click on the "Send new report" button to automatically generate the report and send it to the safety officer.
- Click on the information button for the report to go to the Report Details Page to view the report.
- Click on the "Back" button to return to the Home Page.



8.4.9. Report Details Page

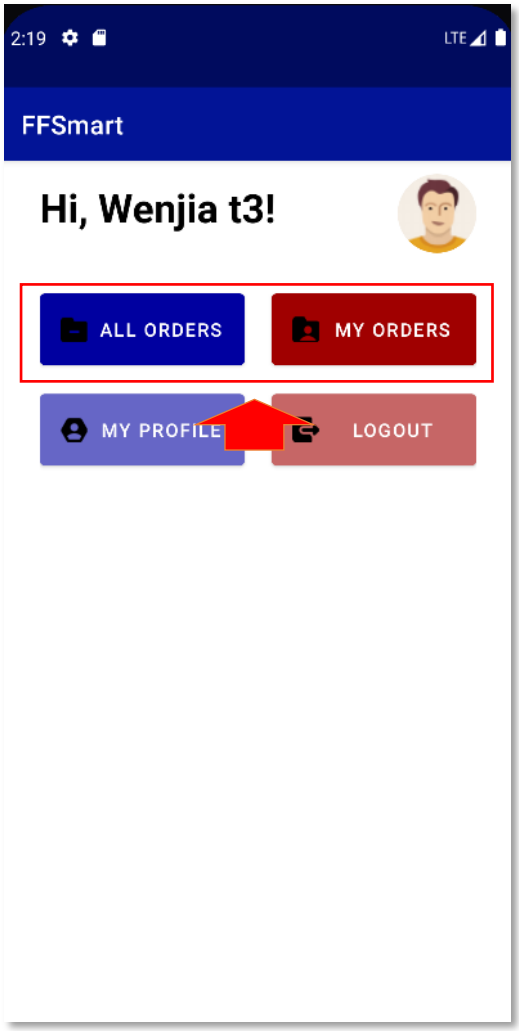
Each report can be viewed in PDF format on the Report Details Page, which provides a table of items due to expire in the inventory.

Standard gestures on mobile devices can pan and zoom the PDF.

- Click on the "Back" button to return to the Health & Safety Reports Page.

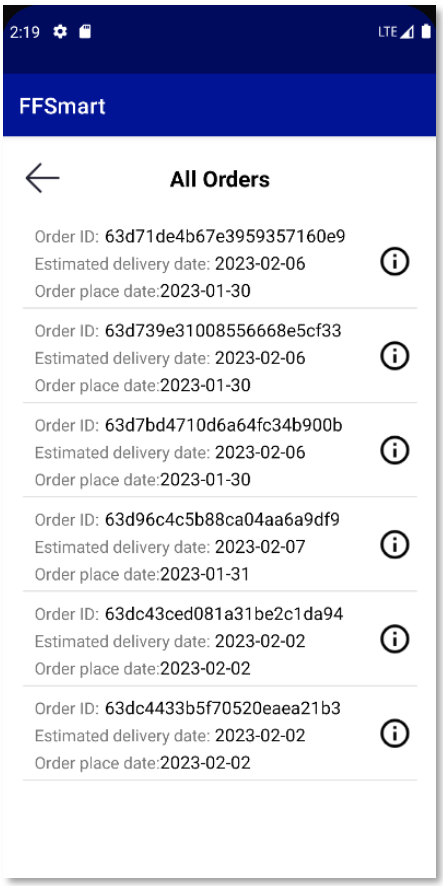
Chapter 4

8.5. Driver Role Guide



8.5.1. Driver Home Page

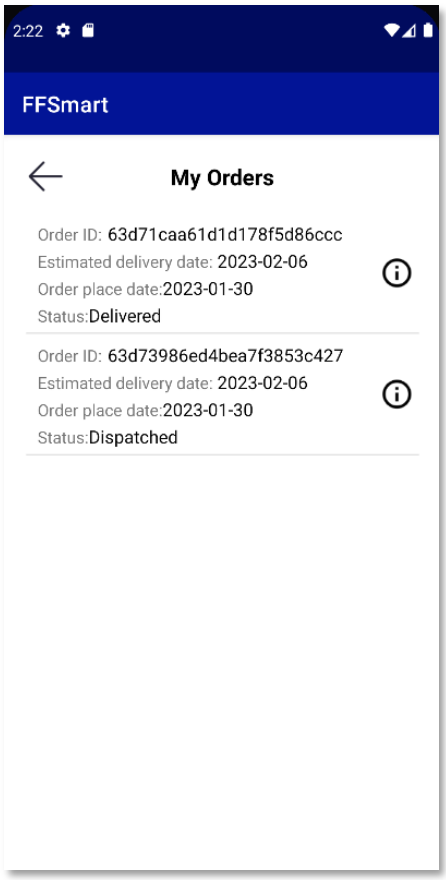
- Click on the "All orders" button to go to the Driver Orders Page to see all orders waiting to be taken for delivery.
- Click on the "My orders" button to go to the My Orders Page to check the orders currently being delivered.



8.5.2. Driver Orders Page

All orders waiting to be dispatched can be viewed on the Driver Orders page. Each order contains the order ID, estimated delivery date and the order placed date.

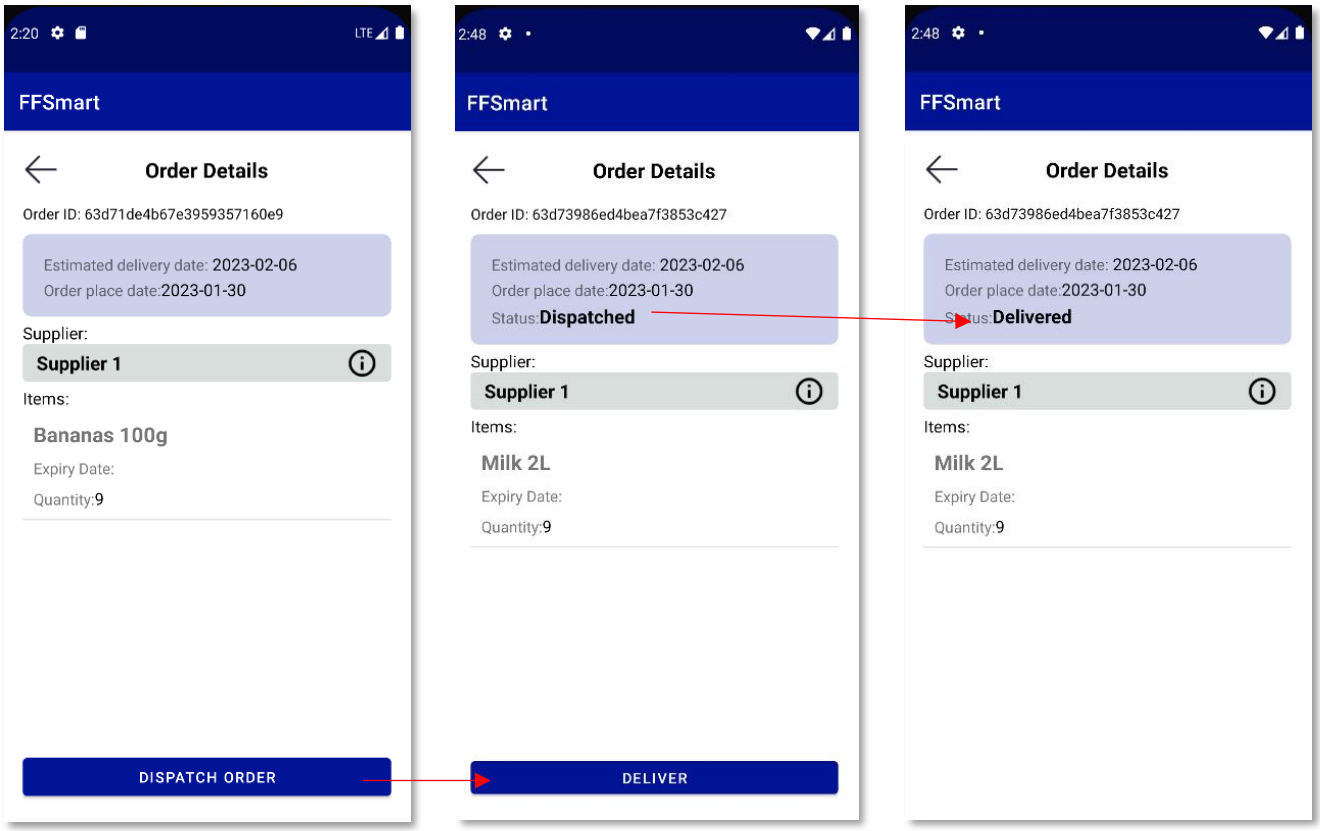
- Click on the "Information" button for the order to go to the Driver Order Details Page to check the order details.
- Click on the "Back" button to return to the Home Page.



8.5.3. My Orders Page

All orders currently being delivered and orders that have been delivered (by the current user) can be found on the My Orders Page. Each order contains the order ID, estimated delivery date, order place date, and order status.

- Click on the "Information" button for the order to go to the Driver Order Details Page to check the order details.
- Click on the "Back" button to return to the Home Page.



8.5.4. Driver Order Details Page

The order details can be checked on the Driver Order Details Page, including order ID, estimated delivery time, order place date, supplier, and items in the order.

- Click on the "Information" button for the supplier to go to the supplier details page to view the supplier's details.
- Click on the "Back" button returns to the All-Orders Page.

1. Click the "Dispatch order" button to become the assigned driver for that order.
2. After dispatch, the "Dispatch order" button will change to the "Deliver" button. The order status will also be displayed as "Dispatched".
3. After putting the order's item into the fridge, click on the "Deliver" button to change the order's status to "Delivered" and add the order’s items to the fridge invent

8.6 User Help Feedback

The User Help documentation has been a collaborative effort between the developers and tester and has therefore been the result of several members experiences with installing and using the system and has seen many changes over its lifespan. It also has been tested on multiple hardware setups, which enabled us to discover some common issues that may be experienced by users, such as difficulties with running Android Emulators using AMD brand CPUs. The step-by-step installation guide was written by the tester upon first installing the software and doing fresh installations of any pre-requisites such as Git and Java, simulating the experience of setting up a new device or the experience of a non-technically proficient user. Once install had been completed it was then updated to include common issues and pitfalls found by the tester, both developers, and online communities.

The guides to using the application once installed was written by the development team, and then amended and improved at the feedback of the tester. Both sections of the guide are understandable and helpful and have been tested first-hand by team members. The user guide was utilised extensively by the tester when writing system and acceptance tests to ensure that the application functions as intended. The extensive documentation as well as non-technical language and easy-to-follow steps included throughout ensure usefulness to clients with mixed technical abilities and experience.

9. Testing

Test plan

Goals

- To ensure the application meets functional and non-functional requirements fully.
- To check that the application has no defects and functions correctly in all use cases.

Test types to be performed:

- Unit testing
- Integration testing
- System testing
- Performance testing
- User Acceptance Testing (UAT)

Scope

- Whole application (all MUST/SHOULD requirements and use cases)

Phases

- Phase 1: Unit testing (1 week / 20 hours)
- Phase 2: Integration testing (1 week / 20 hours)
- Phase 3: System and performance testing (2 weeks / 40 hours)
- Phase 4: UAT (1 week / 40 hours)

Processes and procedures

- Unit and integration tests
    - Write tests with JUnit/Mockito based on requirements and implemented features.
    - Handle any exceptions.
    - Add run configuration(s).
    - Execute unit and integration tests separately.
- System tests
    - Write test cases based on Functional Requirements and additional features that have been implemented by developers.
    - Differentiate between tests that can be automated versus ones that have to be carried out manually.
    - Install and set up Katalon Studio to function with Android Studio's emulator or BlueStacks.
    - Write and test automated test cases.
    - Compile full test suite.
    - Analyse results using the test suite report and log viewer.
    - Carry out manual test cases.
    - Discuss implementation of Functional Requirements including non-implemented and additional features.
- Performance tests
    - Install JMeter.
    - Deploy API on Azure Spring Apps.
    - Add vertical and horizontal scaling configurations.

    - Create test plan and thread group in JMeter.
    - Add HTTP request and header manager, configure protocol host, port, and URL path.
    - Write response code assertions and add results graph listener to view response times and throughput.
- UAT
    - Create user stories based on the requirements, use cases and needs of expected users.
    - Conduct user stories in an as close to a real environment as possible.
    - Analyse results of each story and consider whether all stakeholder needs have been met.
    - Conclude and evaluate the testing phase of the project.

Tools and frameworks

- JUnit
- Mockito
- PowerMock
- Espresso
- Spring Boot Test
- Katalon Studio
- IntelliJ Test Runner / Android Studio Test Runner

Tester profiles

- Oliver
    - Unit and integration tests (API)
    - Performance testing
- Wenjia
    - Unit and integration tests (Android client)
    - Android UI tests
- Regan
    - System tests
    - UAT

The team will spend a total of approximately 120 hours on testing the application. The testing will involve three members of the team, each working on separate types of testing on the API, Android client and the system as a whole. On both the client-side and API, unit and integration tests have been mapped to functional requirements and the relevant user acceptance tests.

9.1. Unit tests

Unit testing is the first stage of the testing process, and involves testing individual units of code, such as a component or API endpoint, to determine whether they are fit for purpose. In our case, a substantial proportion of unit testing was performed during the development phase in order to assist development, attain higher code quality and make sure that each part of the software was executing its function effectively. The majority of tests are named according to the given-when-then (GWT) convention, with some exceptions in cases where GWT was not deemed necessary to fully describe the test.

9.1.1. Server

JUnit and Mockito have been used for server-side unit tests. Mockito is a mocking framework for Java and allows you to mock interfaces and add dummy functionality, whilst avoiding external dependencies. Here, it is used in conjunction with JUnit for testing interface methods. All Controller, Service and Repository classes (where custom methods have been developed) have been unit tested. Test packages are arranged in the same way as the main packages with test classes mapped to the main classes. Where a class is named "{module} Tests", this is an integration test class. It should be noted that although the repository tests are technically integration tests (as they integrated with an external database), I have included them as unit tests here as they test a single interface within the codebase. Additionally, some test packages do not have a Repository test class. This is either because a) they do not require database integration, or b) no custom repository methods have been added to the repository interface. The full list of server test classes is shown below:

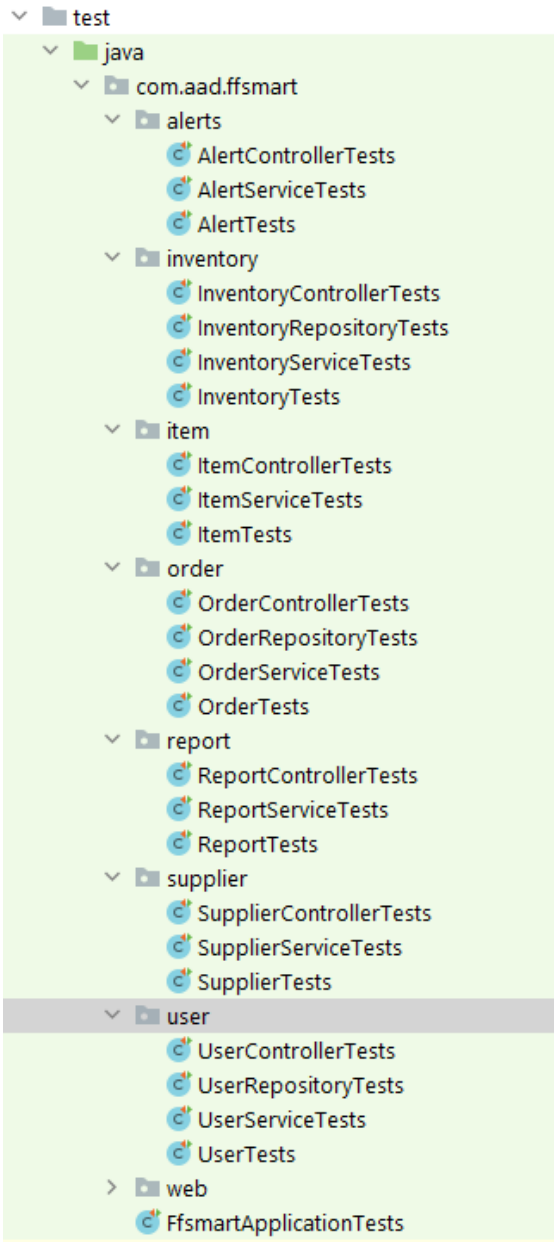


Figure 19: All server test packages and classes

Some examples of unit test classes will now be given.

```
@WebFluxTest(AlertController.class)
@Import(WebFluxTestSecurityConfig.class)
class AlertControllerTests {
    1 usage
    @Autowired
    WebTestClient webTestClient;

    2 usages
    @MockBean
    private AlertService alertService;

    no usages
    @MockBean
    private GlobalErrorAttributes globalErrorAttributes;

    no usages  orw22
    @DisplayName("Get alerts, expect status Ok")
    @WithMockUser(roles = "HEAD_CHEF")
    @Test
    void givenAlerts_whenGetAlerts_thenStatusOk() {
        Alert alert = new Alert(AlertCode.ORDER_READY, title: "Order ready", message: "Order is ready for approval", new Date());
        when(alertService.getAlerts()).thenReturn(Flux.just(alert));

        webTestClient.get() RequestHeadersUriSpec<capture of ?>
                .uri( uri: "/alerts") capture of ?
                .exchange() ResponseSpec
                .expectStatus().isOk()
                .expectBody() BodyContentSpec
                .jsonPath( expression: "$.data[0].alertCode").isEqualTo(AlertCode.ORDER_READY.value)
                .consumeWith(System.out::println);
        verify(alertService, times( wantedNumberOfInvocations: 1)).getAlerts();
    }

}
```

Figure 20: Alerts controller unit test class

Above is the alerts controller test class with get alerts unit test. Spring's @WebFluxTest annotation is used, which creates the minimum application context required to launch a web server and auto-configures the web test client, which is injected into the class with the *@Autowired* annotation. *@WebFluxTest* is designed specifically for testing reactive Spring web controller classes. The *@MockBean* annotation is used to mock the alert service interface. Notice that only interfaces are mocked in order to align with the Dependency Inversion Principle.

The unit test itself is named in given-when-then format; given some alerts, when the user calls the get alerts endpoint, the status should be Ok (200 response code). The test is annotated with *@WithMockUser*, which allows for controlling the role behind the request and thus testing of security features and access control. The *HEAD_CHEF* role is used, as this is the only role that has permission to view alerts. Mockito's *when* method is used to add dummy behaviour to the alert service mock. Then, web test client gets from the relevant endpoint and tests the JSON response body to ensure that the data returned matches that specified for the mock. Finally, Mockito's *verify* method is used to ensure that the relevant alert service method was called.

```
@ExtendWith(MockitoExtension.class)
class ItemServiceTests {
    1 usage
    @InjectMocks
    private ItemServiceImpl itemService;

    no usages  orw22
    @DisplayName("Get all items, expect flux of all items returned")
    @Test
    void givenItems_whenGetAllItems_thenItemsReturned() {
        Flux<Item> itemFlux = itemService.getAllItems( name: null,  supplierId: null).take( n: 1);

        StepVerifier
                .create(itemFlux)
                .consumeNextWith(item -> {
                    assertNotNull(item.getId());
                    assertEquals( expected: "0", item.getId());
                    assertEquals( expected: "Supplier 1", item.getSupplierName());
                })
                .verifyComplete();
    }

}
```

Figure 21: Item service unit test class

This is an example of a service test class. Again, Mockito is used, and mocks are injected into the item service class. *@ExtendWith* is used to add JUnit/Mockito test lifecycle handling and allow for injecting mocks. The test is given a display name which will appear in the test console when the test is run. Within the get all items unit test itself, Spring's *StepVerifier* is used to access and make assertions on the contents of the Flux returned by the service class. If the item ID and supplier name are not as expected, the test will fail. The *verifyComplete* function expects a completion signal, meaning that there are no more items to process in the Flux object. This will always succeed, as we have taken only the first item from the flux (line 22).

Unit Test Results

Table 2: Unit test results

Test name	**Class**	**Role**	**Output**	**Passed (Y/N)**
givenAlerts_whenGetAlerts_thenStatusOk	AlertControllerTests	Head Chef	200 OK	Y
givenAlertsAndDeliveryDriverRole_whenGetAlerts_thenStatusForbidden	AlertControllerTests	Delivery Driver	403 Forbidden	Y
givenAlerts_whenGetAlerts_thenAlertsReturned	AlertServiceTests	n/a	Alert(alertCode=ORDER_READY, title=Order ready, message=Order is ready for approval, timestamp=Sat Feb 04 16:08:22 GMT 2023)	Y
whenAddInventory_thenStatusCreated	InventoryControllerTests	Delivery Driver	201 Created	Y
givenInventory_WhenRemoveInvento	InventoryControll	Chef	200 OK	Y

ry_thenStatusOk	erTests			
givenInventory_WhenGetAllInventory_thenStatusOk	InventoryControllerTests	Chef	200 OK	Y
givenInventory_WhenGetInventoryById_thenStatusOk	InventoryControllerTests	Chef	200 OK	Y
givenInventory_WhenUpdateInventoryById_thenStatusOk	InventoryControllerTests	Chef	200 OK	Y
givenInventoryChanges_WhenGetInventoryChangeHistory_thenStatusOk	InventoryControllerTests	Head Chef	200 OK	Y
givenExpiredItems_WhenGetExpiredItems_thenStatusOk	InventoryControllerTests	Head Chef	200 OK	Y
givenExpiredItems_WhenRemoveExpiredItems_thenStatusNoContent	InventoryControllerTests	Head Chef	204 No Content	Y
givenInventory_whenFindAll_thenInventoryReturned	InventoryRepositoryTests	n/a	n/a	Y
givenInventory_whenFindByItemIdExpiryDate_thenInventoryReturned	InventoryRepositoryTests	n/a	n/a	Y
givenExpiredInventory_whenFindExpired_thenExpiredItemsReturned	InventoryRepositoryTests	n/a	n/a	Y
givenExpiredInventory_whenDeleteExpired_thenNoExpiredItemsReturned	InventoryRepositoryTests	n/a	n/a	Y
givenInventory_whenAddInventory_thenInventoryChangeReturned	InventoryServiceTests	n/a	InventoryChange(id=123, userId=789, items=[InventoryItem(id=null, itemId=0, itemName=Bananas 100g, supplierId=63d1b3dae8b8e7e8b68300af, supplierName=Supplier 1, quantity=10, expiryDate=Sat Feb 04 17:01:13 GMT 2023)], operation=INSERT, date=Sat Feb 04 17:01:13 GMT 2023)	Y
givenInventory_whenRemoveInventory_thenInventoryChangeReturned	InventoryServiceTests	n/a	InventoryChange(id=123, userId=789, items=[InventoryItem(id=null, itemId=0, itemName=Bananas 100g, supplierId=63d1b3dae8b8e	Y

			7e8b68300af, supplierName =Supplier 1, quantity=0, expiryDate=S at Feb 04 17:04:05 GMT 2023)], operation=RE MOVE, date=Sat Feb 04 17:04:05 GMT 2023)	
givenInventory_whenGetAllInventory _thenSuccess	InventoryService Tests	n/a	n/a	Y
givenInventory_whenGetInventoryByI d_thenSuccess	InventoryService Tests	n/a	n/a	Y
givenInventory_whenUpdateInventory ById_thenSuccess	InventoryService Tests	n/a	n/a	Y
givenInventoryChange_whenGetInven toryChangeHistory_thenSuccess	InventoryService Tests	n/a	n/a	Y
givenInventoryChange_whenGetInven toryChangeById_thenSuccess	InventoryService Tests	n/a	n/a	Y
givenExpiredItems_whenGetExpiredIt ems_thenSuccessful	InventoryService Tests	n/a	n/a	Y
givenExpiredItems_whenRemoveExpir edItems_thenSuccessful	InventoryService Tests	n/a	n/a	Y
givenInventory_whenAggregateInvent ory_thenSuccessful	InventoryService Tests	n/a	SupplierItems (supplierId=1 23, supplierName =ABC, items=[])	Y
givenItems_whenGetAllItems_thenSta tusOk	ItemControllerTe sts	Any authorized user	200 OK	Y
givenItems_whenGetAllItems_thenIte msReturned	ItemServiceTests	n/a	n/a	Y
givenHeadChefRole_whenCreateOrder _thenStatusCreated	OrderControllerT ests	Head Chef	201 Created	Y
givenOrderStatusDelivered_whenGetA llOrders_thenStatusOk	OrderControllerT ests	Head Chef	200 OK	Y
givenOrdersAndRoleDeliveryDriver_w henGetMyOrders_thenStatusOk	OrderControllerT ests	Delivery Driver	200 OK	Y
givenOrdersAndRoleChef_whenGetMy Orders_thenStatusForbidden	OrderControllerT ests	Chef	403 Forbidden	Y
givenOrders_whenGetOrderById_then StatusOk	OrderControllerT ests	Head Chef	200 OK	Y
givenReadyOrder_whenApproveOrder _thenStatusOk	OrderControllerT ests	Head Chef	200 OK	Y
givenReadyOrder_WhenRejectOrder_t henStatusNoContent	OrderControllerT ests	Head Chef	204 No Content	Y
givenOrders_whenFindApprovedOrder s_thenApprovedOrdersReturned	OrderRepositoryT ests	n/a	n/a	Y

givenOrders_whenFindReadyOrders_thenReadyOrdersReturned	OrderRepositoryTests	n/a	n/a	Y
givenOrders_whenFindByDriverId_thenDriverOrdersReturned	OrderRepositoryTests	n/a	n/a	Y
givenUser_whenCreateOrder_thenOrderCreated	OrderServiceTests	n/a	n/a	Y
givenNoStatus_whenGetAllOrders_thenOrdersReturned	OrderServiceTests	n/a	n/a	Y
givenStatusApproved_whenGetAllOrders_thenApprovedOrdersReturned	OrderServiceTests	n/a	n/a	Y
givenOrders_whenGetMyOrders_thenOrdersReturned	OrderServiceTests	n/a	n/a	Y
givenOrders_whenGetReadyOrders_thenOrdersReturned	OrderServiceTests	n/a	n/a	Y
givenOrders_whenGetApprovedOrders_thenOrdersReturned	OrderServiceTests	n/a	n/a	Y
givenOrders_whenGetOrderById_thenOrderReturned	OrderServiceTests	n/a	n/a	Y
givenInvalidOrderId_whenGetOrderById_thenError	OrderServiceTests	n/a	Not found exception	Y
givenOrder_whenApproveOrder_thenOrderStatusApproved	OrderServiceTests	n/a	n/a	Y
givenOrder_whenRejectOrder_thenOrderDeleted	OrderServiceTests	n/a	n/a	Y
givenOrder_whenDispatchOrder_thenOrderStatusInTransit	OrderServiceTests	n/a	n/a	Y
givenOrder_whenDeliverOrder_thenOrderStatusDelivered	OrderServiceTests	n/a	n/a	Y
givenReports_whenGetAllReports_thenStatusOk	ReportControllerTests	Head Chef	200 OK	Y
givenReportsAndRoleDeliveryDriver_whenGetAllReports_thenStatusForbidden	ReportControllerTests	Delivery Driver	403 Forbidden	Y
givenRoleHeadChef_whenGenerateReport_thenStatusCreated	ReportControllerTests	Head Chef	201 Created	Y
givenRoleHeadChef_whenDownloadReport_thenStatusOkAndContentTypePDF	ReportControllerTests	Head Chef	200 OK and Content-Type PDF	Y
givenReports_whenGetAllReports_thenReportsReturned	ReportServiceTests	n/a	n/a	Y
givenUser_whenGenerateReport_thenReportGenerated	ReportServiceTests	n/a	n/a	Y
givenReports_whenDownloadReport_thenSuccess	ReportServiceTests	n/a	n/a	Y
givenSuppliers_whenGetAllSuppliers_thenStatusOk	SupplierControllerTests	Chef	200 OK	Y
givenSuppliers_whenGetSupplierById_thenStatusOk	SupplierControllerTests	Chef	200 OK	Y
givenNoSupplierName_whenGetAllSuppliers_thenSuccess	SupplierServiceTests	n/a	n/a	Y
givenSupplierName_whenGetAllSuppliers_thenSuccess	SupplierServiceTests	n/a	n/a	Y
givenSupplier_whenGetSupplierById_thenSuccess	SupplierServiceTests	n/a	n/a	Y

givenInvalidSupplierId_whenGetSupplierById_thenError	SupplierServiceTests	n/a	Not found exception	Y
givenUser_whenLogin_thenStatusOk	UserControllerTests	Any role	200 OK and token	Y
givenUser_whenRegister_thenStatusCreated	UserControllerTests	Any role	201 Created and token	Y
givenUsers_whenGetAllUsers_thenStatusOk	UserControllerTests	Head Chef	200 OK	Y
givenUserId_whenGetUserById_thenStatusOk	UserControllerTests	Head Chef	200 OK	Y
givenUserId_whenGetUserNameFromId_thenStatusOk	UserControllerTests	Chef	200 OK	Y
givenUserAndId_whenUpdateUserById_thenStatusOk	UserControllerTests	Head Chef	200 OK	Y
givenUserId_whenDeleteUserById_thenStatusNoContent	UserControllerTests	Head Chef	200 OK	Y
givenToken_whenGetUserByToken_thenStatusOk	UserControllerTests	Any role	200 OK	Y
givenToken_whenUpdateUserByToken_thenStatusOk	UserControllerTests	Any role	200 OK	Y
givenToken_whenDeleteUserByToken_thenStatusNoContent	UserControllerTests	Any role	200 OK	Y
givenEmail_whenFindUserByEmail_thenUserReturned	UserRepositoryTests	n/a	n/a	Y
givenUserExistsAndCorrectPassword_whenLogin_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenUserExistsAndIncorrectPassword_whenLogin_thenError	UserServiceTests	n/a	n/a	Y
givenNewUser_whenRegister_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenUsers_whenGetAllUsers_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenUserId_whenGetUserById_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenInvalidUserId_whenGetUserById_thenError	UserServiceTests	n/a	n/a	Y
givenUserId_whenGetUserNameFromId_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenUserId_whenUpdateUserById_thenSuccessful	UserServiceTests	n/a	n/a	Y
givenUserId_whenDeleteUserById_thenSuccessful	UserServiceTests	n/a	n/a	Y

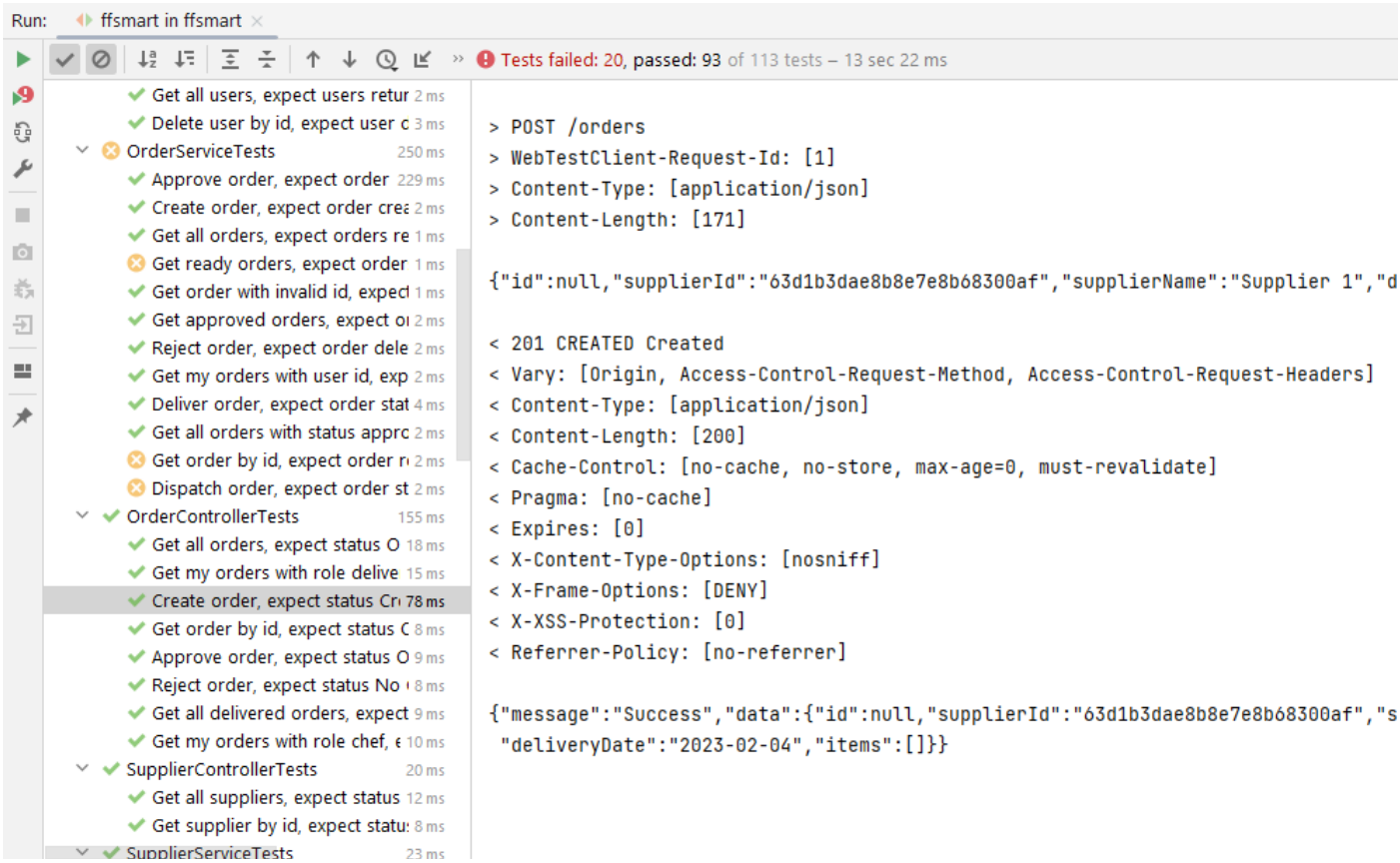


Figure 22: Complete unit and integration test run in IDE.

The IDE used (IntelliJ) has an automated testing feature that can run all the tests in a package one after another. Above we can see an example of a full unit and integration test run performed during the development phase of the application with 90 passes and 20 failures. The output from one of the successful order controller unit tests is shown.

9.1.2. Client

The figure below shows that the Android client-side unit testing was performed on the *LoginSession*, *MyJsonUtil* and *RegisterActivity* classes. The frameworks used were JUnit, Mockito and PowerMock. As the methods in the *LoginSession* and *MyJsonUtil* classes are static, using the more comprehensive PowerMock makes up for the lack of support for static method mocking in Mockito. Furthermore, many of the methods in the client application use Android components that Mockito and PowerMock cannot mock. This issue seriously impacted the code coverage of the test, so UI testing was also performed to test components with the Espresso testing framework.

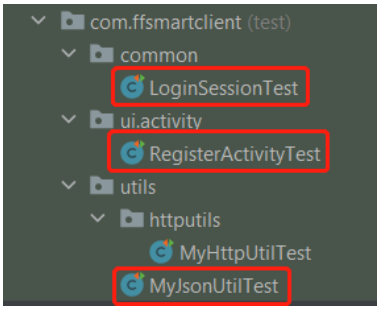


Figure 23: Client-side unit tests shown in their respective packages

Unit testing has been automated using built-in Android studio components. For Java classes like *LoginSession* and *MyJsonUtil* which were continuously updated during the development, the efficiency of regression testing can be effectively improved, which further optimizes the productivity of the development.

LoginSession and *MyJsonUtil* are two classes that are encapsulated and reused many times in the client application. Therefore, unit testing the methods in these classes can improve test code coverage by testing a greater proportion of the working code with the same number of test cases. Secondly, the *RegisterActivity* class unit tests are based on methods for validating user input.

Automated unit testing of these frequently reused methods is vital, as even minor changes made within the method can significantly impact the client application's overall functionality. In summary, all the methods in the three classes passed the unit tests, and the following will illustrate how the unit tests were performed in each of the three classes.

LoginSession

As shown in the figure below, the *LoginSession* class was tested using the JUnit and PowerMock frameworks and called the static methods in the *LoginSession* class using *mockStatic(LoginSession.class)*. In addition, the four methods *getUid()*, *getUser()*, *setUser()* and *logout()* were unit tested in the *LoginSession* class. These four methods are responsible for accessing the information in Android's shared preference to get the user ID, get the user object, set a new login user object, and clear the current login user object.

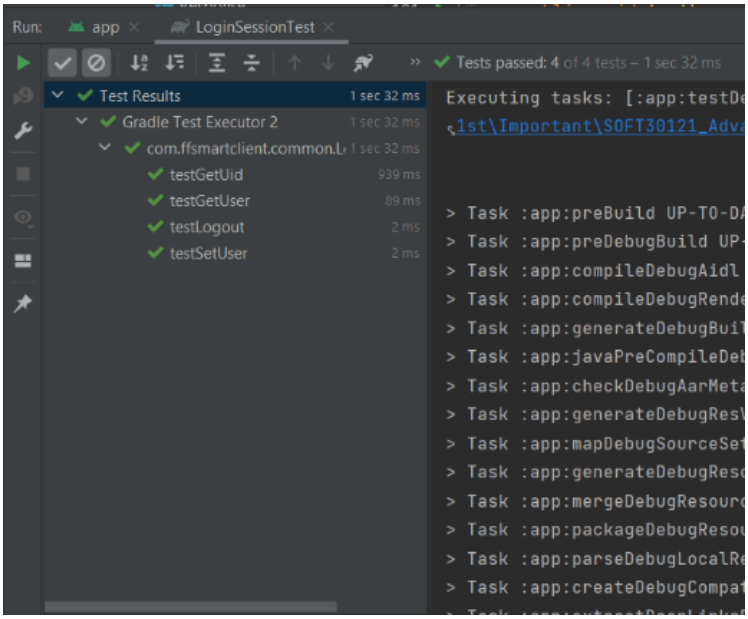


Figure 24: LoginSession test results

As shown in the code below, the *@RunWith* and *@PrepareForTest* annotations are used to specify the *PowerMockRunner* as the test runner and prepare the *LoginSession* and *SPUtils* classes for testing, respectively. Next, the @Before implements invoking all static methods of the *LoginSession* class with *mockStatic(LoginSession.class)* and initialising the mocked object with *initMocks(this)*. Finally, use the *@Test* annotation has been added to recognise the test*GetUid()* method as a JUnit test.

First, the test used a new user object to define the expected return value for the *getUser()* method, which was used in the *getUid()* method to simulate an actual operation. Then, the real *getUid()* method was called using *thenCallRealMethod()*, and finally it was verified that the *getUid()* return value was equal to the expected value via *assertEquals*.

```java
@RunWith(PowerMockRunner.class)
@PrepareForTest({LoginSession.class, SPUtils.class})
public class LoginSessionTest {
    @Mock
    private MyJsonUtil myJsonUtil;
    @Before
    public void setUp() {
        mockStatic(LoginSession.class);
        initMocks( testClass: this);
        System.out.println("New Test Begin =>");
    }
    @Test
    public void testGetUid() {
        String expectedUserId = "63e018d851f4491179db9fbc";
        User user = new User();
        user.setId("63e018d851f4491179db9fbc");
        user.setEmail("wj@gmail.com");
        user.setPassword("$2a$12$JS9nd2OoHljvpqxlUDI8M.Ze/ccZQBVbjzLh8GDNQb8qSGuzCPcp2");
        user.setFirstName("Wenjia");
        user.setLastName("G");
        user.setRole(2);
        user.setAvatar(null);
        when(LoginSession.getUser()).thenReturn(user);
        when(LoginSession.getUid()).thenCallRealMethod();
        String aUid = LoginSession.getUid();
        assertEquals(expectedUserId, aUid);
    }
```

Figure 25: PowerMock login session unit test

JSON Utilities

As shown in the figure below, the *MyJsonUtil* class was tested using the JUnit and PowerMock frameworks, and the static methods in the *MyJsonUtil* class were called using *mockStatic(MyJsonUtil.class)*. The *toJson()* and *fromJson()* methods, which are for serializing a java object to a JSON format string or deserializing a JSON format string to a java object respectively, were unit tested in *MyJsonUtilTest*.

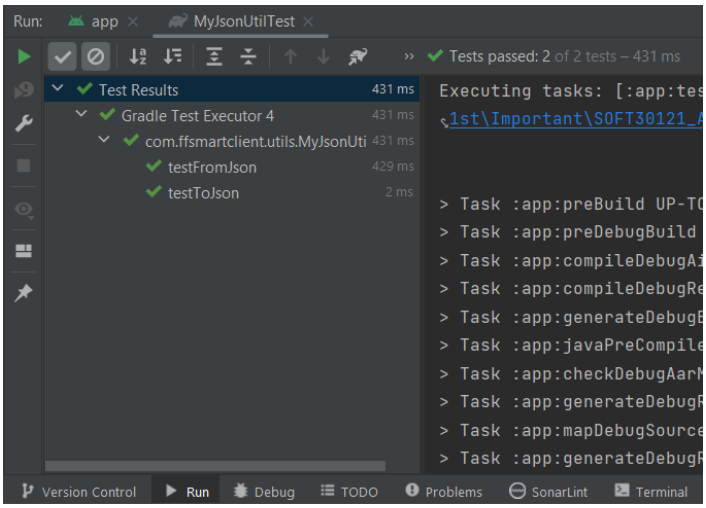


Figure 26: JSON utils unit test results

As shown in the code below, the *toJson()* method in *MyJsonUtil* is tested in the test method *testToJson()* by creating a *LoginCredential* object with email and password defining the expected result of the serialisation of the *LoginCredential* object. Next, the actual *toJson()*

method is called using the *thenCallRealMethod()* function to get the serialised JSON result. Finally, the *assertNotNull* method asserts that the result is not null before *assertEquals* is invoked to check if the result is equal to the expected value.

```
@RunWith(PowerMockRunner.class)
@PrepareForTest({MyJsonUtil.class})
public class MyJsonUtilTest {
    @Before
    public void setUp() {
        mockStatic(MyJsonUtil.class);
        initMocks( testClass: this);
        System.out.println("New Test Begin =>");
    }
    @Test
    public void testToJson() {
        LoginCredential loginObj = new LoginCredential();
        loginObj.setEmail("t1@gmail.com");
        loginObj.setPassword("123123b");
        String expectedJson = "{\"email\":\"t1@gmail.com\",\"password\":\"123123b\"}";
        when(MyJsonUtil.toJson(loginObj)).thenCallRealMethod();
        String result = MyJsonUtil.toJson(loginObj);
        assertNotNull(result);
        assertEquals(expectedJson, result);
    }
```

Figure 27: Unit test for toJson method

User Input Validation

As shown in the figure below, the *RegisterActivity* class was tested using JUnit and Mockito frameworks. In *RegisterActivityTest*, four methods were unit tested: *checkUserName()*, *checkEmailFormat()*, *checkPasswordFormat()* and *checkPasswordMatched()*, which are responsible for checking if the username is empty, checking the email format, checking the password format, and checking if the passwords entered twice matched. As these methods are not static and do not use Android elements, the more straightforward Mockito framework could be utilised for testing.

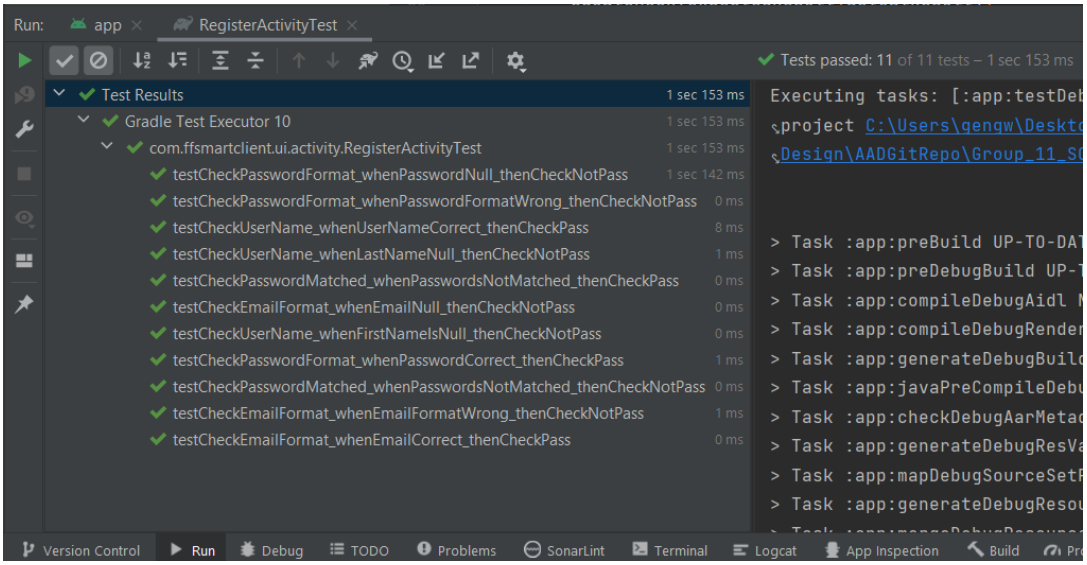


Figure 28: Results of an Android unit test run

As shown in the code below, the mock object for the test is initialised using *initMocks(this)* before the test starts. During the test, *when(registerActivity.checkUserName(firstName,lastName)).thenCallRealMethod()* is used to configure the mock object so that the actual method is called directly when *checkUserName()* is invoked. Finally, *assertEquals* is invoked to check that the actual output of *checkUserName* matches the expected result.

```
public class RegisterActivityTest {
    @Mock
    private RegisterActivity registerActivity;
    @Before
    public void setUp() {
        initMocks( testClass: this);
        System.out.println("New Test Begin =>");
    }
    private void equalCheck(boolean expectedResult, boolean actualResult){
        assertEquals(expectedResult, actualResult);
    }
    @Test
    public void testCheckUserName_whenUserNameCorrect_thenCheckPass() {
        String firstName = "Wenjia";
        String lastName = "Geng";
        boolean expectedResult = true;
        when(registerActivity.checkUserName(firstName,lastName)).thenCallRealMethod();
        boolean actualResult = registerActivity.checkUserName(firstName,lastName);
        equalCheck(expectedResult,actualResult);
    }
```

Figure 29: Register activity unit test

Future Improvement

In different android activities, all methods for user input non-empty checking and format checking can be encapsulated and tested across the application to further improve the efficiency of automated unit testing and reduce the frequency of regression testing.

Client Unit Test Results

Figure 30: Client unit test results table

Test Name	Class	Description	Expected Output	Passed (Y/N)
testGetUid	LoginSessionTest	GIVEN user object, WHEN get user id, THEN login user id gets	Login user id	Y
testGetUser	LoginSessionTest	GIVEN user object, WHEN get user, THEN login user object in JSON gets	Login user object in JSON	Y
testSetUser	LoginSessionTest	GIVEN user object, WHEN set user, THEN login user object sets	n/a	Y
testLogout	LoginSessionTest	GIVEN user object, WHEN logout user, THEN login user object delete	n/a	Y
testToJson	MyJsonUtilTest	GIVEN LoginCredential object, WHEN convert LoginCredential to JSON, THEN get LoginCredential object in JSON	LoginCredential JSON	Y
testFromJson	MyJsonUtilTest	GIVEN LoginCredential JSON, WHEN convert LoginCredential to object, THEN get LoginCredential object	LoginCredential object	Y
testCheck Password Matched	RegisterActivityTest	GIVEN passwords matched, WHEN passwords match check, THEN return true	True	Y
testCheck Password NotMatched	RegisterActivityTest	GIVEN passwords not matched, WHEN passwords match check, THEN return false	False	Y
testCheck PasswordFromat Null	RegisterActivityTest	GIVEN null password, WHEN password format check, THEN return false	False	Y
testCheck PasswordFromat Wrong	RegisterActivityTest	GIVEN wrong format password, WHEN password format check, THEN return false	False	Y
testCheck PasswordFromat Correct	RegisterActivityTest	GIVEN correct format password, WHEN password format check, THEN return true	True	Y
testCheck UserName FirstNameNull	RegisterActivityTest	GIVEN null first name, WHEN username check, THEN return false	False	Y
testCheck	RegisterActivityTest	GIVEN null last name,	False	Y

UserName LastNameNull		WHEN username check, THEN return false		
testCheck UserName Correct	RegisterActivityTest	GIVEN correct username, WHEN username check, THEN return true	True	Y
testCheck EmailFromat Null	RegisterActivityTest	GIVEN null email, WHEN email format check, THEN return false	False	Y
testCheck EmailFromat Wrong	RegisterActivityTest	GIVEN wrong format email, WHEN email format check, THEN return false	False	Y
testCheck EmailFromat Correct	RegisterActivityTest	GIVEN correct format email , WHEN email format check, THEN return true	True	Y

9.2. Integration Tests

Integration tests aim to test multiple layers of an application at once to ensure that they are functioning according to specifications and without defects. While unit tests focus on individual components, integration tests consider how these components combine and work together. There is no mocking involved in integration tests; in all cases, the "real" class instances and Spring beans are used. All the integration tests have been separated from the unit tests due to their greater complexity and resource utilisation.

9.2.1. Server

On the server side, Spring provides great support for writing integration tests with the *@SpringBootTest* annotation. When a class annotated with this is run, an application context is bootstrapped and started with a server on a randomly chosen port. The *@AutoConfigureWebTestClient* was also added so that the *WebTestClient* could be used to test the API layer and indirectly trigger service and repository functions.

An example of an integration test (from *SupplierTests.java*) is shown below. In this test, every layer of the API is tested: First, a HTTP request is sent by the web test client, which is processed by the controller layer. Then, the controller layer calls the service class, which in turn gets the supplier information from the repository layer, and thus the external database. To test that the resultant response body produced is correct, a *SupplierRepository* bean is used to directly call the repository method, and the results are compared with the ID and email fields with Spring's built-in *StepVerifier*.

```
50      @DisplayName("Get supplier by id integration test")
51      @WithMockUser(roles = "CHEF")
52      @Test
53      void givenSuppliers_whenGetSupplierById_thenSupplierReturned() {
54          String json = webTestClient.get() RequestHeadersUriSpec<capture of ?>
55                  .uri( uri: "/suppliers/" + SUPPLIER_ID) capture of ?
56                  .exchange() ResponseSpec
57                  .expectStatus().isOk()
58                  .expectBody(String.class) BodySpec<String, capture of ?>
59                  .consumeWith(System.out::println) capture of ?
60                  .returnResult() EntityExchangeResult<String>
61                  .getResponseBody();
62          Supplier res = JsonPath.parse(json).read( s: "$.data", Supplier.class);
63
64          Mono<Supplier> supplierMono = supplierRepository.findById(SUPPLIER_ID);
65          StepVerifier
66                  .create(supplierMono)
67                  .consumeNextWith(supplier -> {
68                      assertEquals(supplier.getEmail(), res.getEmail());
69                      assertEquals(supplier.getId(), res.getId());
70                  })
71                  .verifyComplete();
72      }
73  }
```

Figure 31: Get supplier by ID integration test

As before, the *@WithMockUser* annotation is used to simulate the user role making the request and test the permitted roles on the endpoint. The external JsonPath module is used to find the data path in the response body and parse this into our Supplier data model. Additionally, we assert that the HTTP response code is 200 Ok, and the response is logged to the test runner console:

```
> GET /suppliers/63d1b3dae8b8e7e8b68300af
> WebTestClient-Request-Id: [1]

No content

< 200 OK OK
< Vary: [Origin, Access-Control-Request-Method, Access-Control-Request-Headers]
< Content-Type: [application/json]
< Content-Length: [349]
< Cache-Control: [no-cache, no-store, max-age=0, must-revalidate]
< Pragma: [no-cache]
< Expires: [0]
< X-Content-Type-Options: [nosniff]
< X-Frame-Options: [DENY]
< X-XSS-Protection: [0]
< Referrer-Policy: [no-referrer]

{"data":{"id":"63d1b3dae8b8e7e8b68300af","name":"Supplier 1","items":[{"id":"0","name":"Bananas 100g","supplierId":null,"supplierName":null},{"id":"6","name":"Milk 2L",
```

Figure 32: Example of integration test output

Automating the integration tests was again done easily through the IDE, with the ability to add a custom run configuration with pattern matching which ignores all unit tests and runs only the integration test classes. The image below shows the results of a full integration test run, with class names and individual test names displayed on the left.

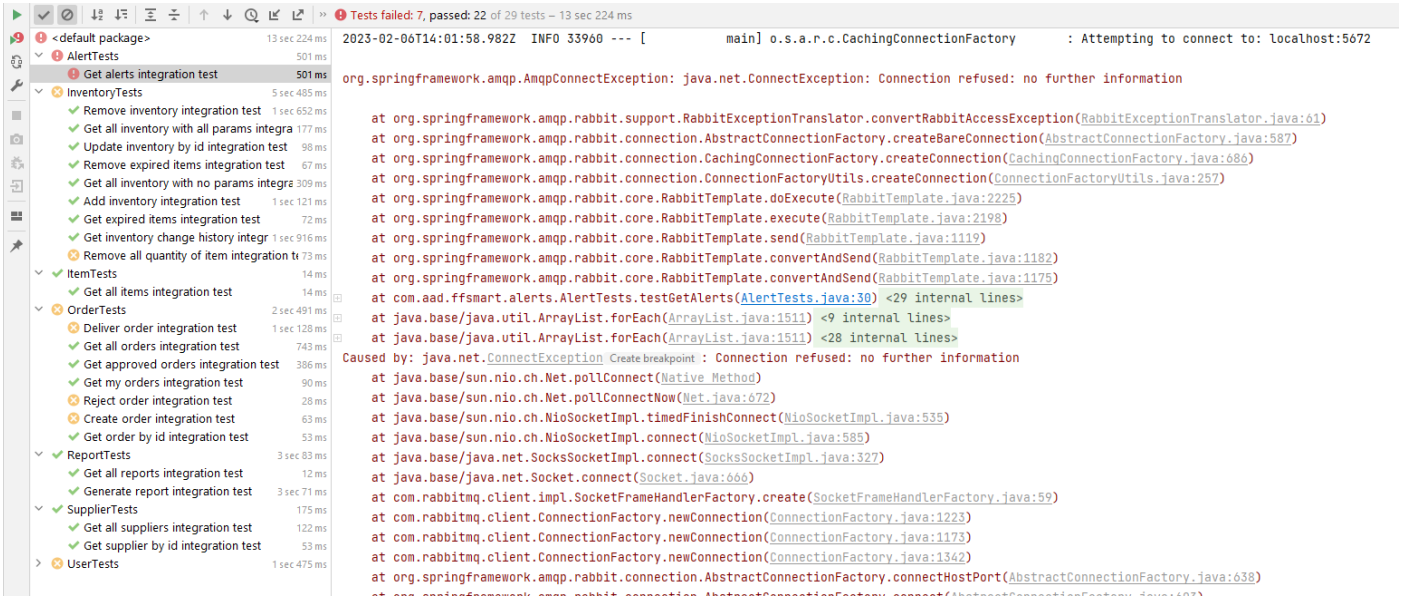


Figure 33: Failed integration test example

In this case, for the get alerts integration test, there was a connection refused exception. This was due to the RabbitMQ server not running at the time the test was run, and the error was easily fixed by simply starting the RabbitMQ server locally. From the same test run, the results for the get reports integration test are given. This test was successful with a 200 response code, and the full controller response has been printed.

Figure 34: Server integration test run

Server Integration Test Results

All integration tests are passing as of 08/02/2023. When run locally, it is possible that several of the integration tests will fail due to future database changes resulting in test IDs being invalidated. Additionally, it should be noted that alert integration tests and several other tests that involve generating alerts will fail if RabbitMQ is not set up and running locally.

Table 3: Server integration test results

Test name	Class	Role	Passed

			(Y/N)
givenAlerts_whenTestGetAlerts_thenStatusOkAndAlertsReturned	AlertTests	Head Chef	Y
givenContext_whenAddInventory_thenInventoryAddedToDatabase	InventoryTests	Head Chef	Y
givenInventory_whenRemoveInventory_thenInventoryRemovedFromDatabase	InventoryTests	Head Chef	Y
givenItemWithQuantity_whenRemoveAllQuantity_thenInventoryItemDeleted	InventoryTests	Head Chef	Y
givenNoParams_whenGetAllInventory_thenInventoryReturned	InventoryTests	Chef	Y
givenAllParams_whenGetAllInventory_thenInventoryReturned	InventoryTests	Chef	Y
givenInventory_whenUpdateInventoryById_thenInventoryUpdated	InventoryTests	Chef	Y
givenInventoryChanges_whenGetInventoryChangeHistory_thenInventoryChangesReturned	InventoryTests	Chef	Y
givenExpiredItems_whenGetExpiredItems_thenItemsReturned	InventoryTests	Head Chef	Y
givenExpiredItems_whenRemoveExpiredItems_thenItemsDeleted	InventoryTests	Head Chef	Y
givenItems_whenGetItems_thenStatusOkAndItemsReturned	ItemTests	Any authorized user	Y
givenUserAndContext_whenCreateOrder_thenOrderAddedToDatabase	OrderTests	Head Chef	Y
givenOrders_whenGetAllOrders_thenStatusOkAndOrdersReturned	OrderTests	Head Chef	Y
givenOrders_whenGetMyOrders_thenStatusOkAndOrdersReturned	OrderTests	Delivery Driver	Y
givenOrders_whenGetReadyOrders_thenStatusOkAndReadyOrdersReturned	OrderTests	Delivery Driver	Y
givenOrders_whenGetOrderById_thenOrderReturned	OrderTests	Head Chef	Y
givenOrders_whenRejectOrder_thenOrderDeleted	OrderTests	Head Chef	Y
givenOrders_whenDeliverOrder_thenStatusOkAndItemsAddedToInventory	OrderTests	Delivery Driver	Y
givenReports_whenGetAllReports_thenResponseContainsFilename	ReportTests	Head Chef	Y
givenContext_whenGenerateReport_thenReportGenerated	ReportTests	Head Chef	Y
givenSuppliers_whenGetAllSuppliers_thenStatusOkAndSuppliersReturned	SupplierTests	Head Chef	Y
givenSuppliers_whenGetSupplierById_thenStatusOkAndSupplierReturned	SupplierTests	Chef	Y
givenUsers_whenLogin_thenValidTokenReturned	UserTests	Any user	Y
givenUsers_whenRegister_thenValidTokenReturned	UserTests	Any user	Y
givenUsers_whenGetAllUsers_thenUsersReturned	UserTests	Head Chef	Y
givenUsers_whenGetUserById_thenUserReturned	UserTests	Head Chef	Y
givenUsers_whenGetUserNameFromId_thenUse	UserTests	Head Chef	Y

rNameReturned			
givenUsers_whenUpdateUserById_thenUserUpdated	UserTests	Head Chef	Y
givenUsers_whenDeleteUserById_thenUserDeleted	UserTests	Head Chef	Y

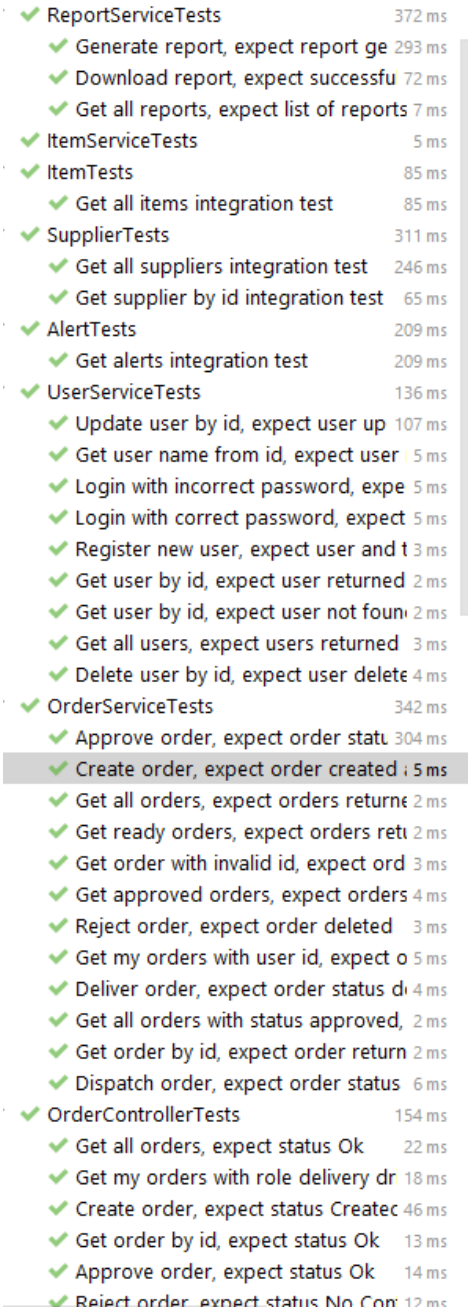


Figure 35: Results of full test run on server

9.2.2. API Communication Testing (Client)

The Android client has used JUnit to perform automated integration tests on the client's HTTP request methods based on the *MyHttpUtil* class. The integration tests checked that the methods

in *MyHttpUtil* can successfully send requests to the server and receive the response from the server.

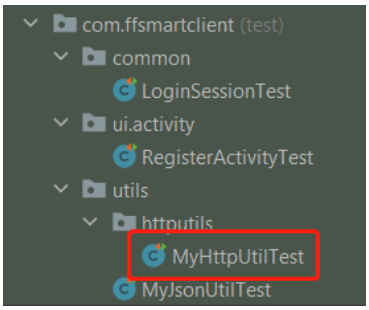


Figure 36: MyHttpUtilTest class shown in test packages

The three methods: *getWithToken()*, *post()*, and *postWithToken()* in *MyHttpUtil* were tested in *MyHttpUtilTest* and were used to:

1. Send a GET/DELETE method HTTP request to the server with a JWT (JSON Web Token).
2. Send a POST/PUT method HTTP request to the server.
3. Send a POST/PUT method HTTP request to the server with the JWT.

Three methods will call back data to the relevant page after receiving the server response.

In addition, the server's response returns either an input stream or an error stream, depending on the response status code. Hence, the integration tests create different test methods for both the input and error streams in each HTTP response.

As shown in the Figure below, a total of six test cases for the input stream and error stream from three HTTP request methods all pass the integration test, and the following will describe how the integration tests are implemented:

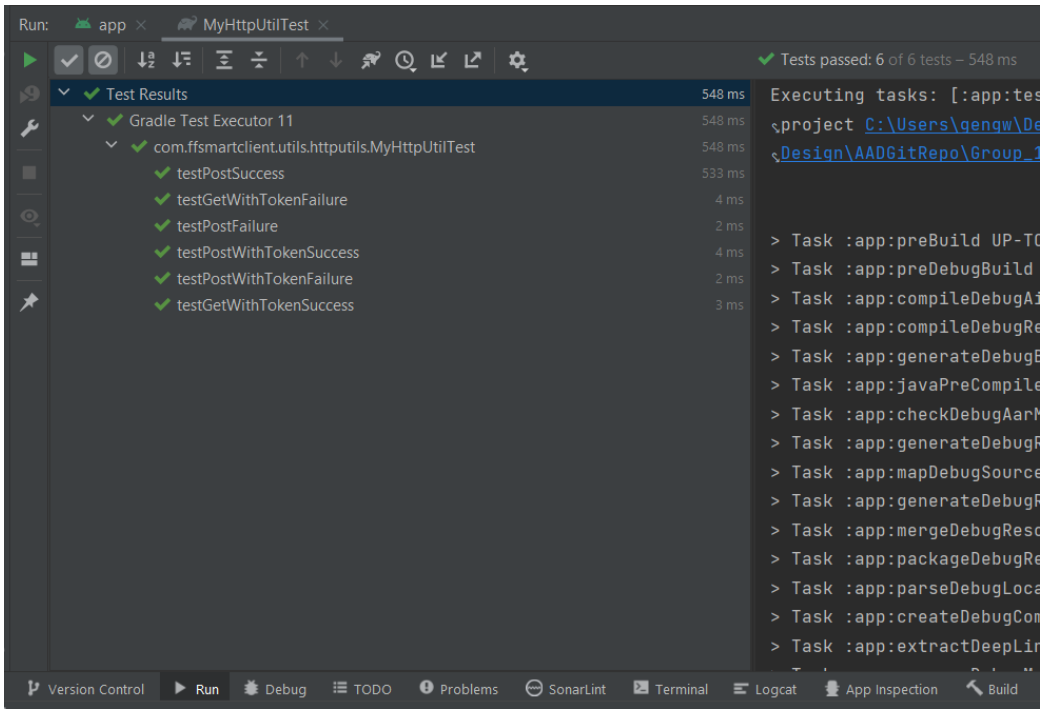


Figure 37: Client-API integration test run results

As shown in the code below, the functionality of the *post()* method in *MyHttpUtil* has been tested in the *testPostSuccess()* test function by creating a *LoginCredential* object with email and password, defining the expected response from the server. Next, the *post()* method is invoked to send a PUT method HTTP request with the *LoginCredential* object to the server's login API. Finally, the returned data is checked against the expected response in *MyHttpCallbackUtil's onSuccess* method by calling back a *MyHttpCallbackUtil* instance, which will test if the HTTP request processing is successful.

```
@Test
public void testPostSuccess() {
    LoginCredential login = new LoginCredential();
    login.setEmail("wj@gmail.com");
    login.setPassword("123123a");
    String expectedSuccessResponseJson = "{\"data\":{\"token\":\"eyJhbGciOiJIUzUxMiJ9.eyJyb2xlIjoiIlJPT" +
            "EVfSEVBRF9DSEVGIlOsInN1YiI6IjYzZTAxOGQ4NTFmNDQ5MTE3OWRiOWZiYyIsImlhdCI6MTY3NTYzMDgwO" +
            "CwiZXhwIjoxNjc1NjM0NDA4fQ.ymEfMX5fSHY3ysviBcZaXCTyXBfpsPqJrtN2JHPBnExKmSpBpjc8Ep-p8MgkG" +
            "iF5ImpcaF8A1cm4eJLTbRP32g\",\"user\":{\"id\":\"63e018d851f4491179db9fbc\",\"email\":\" +
            "\"wj@gmail.com\",\"password\":\"$2a$12$JS9nd2OoHljvpqxlUDI8M.Ze/ccZQBVbjzLh8GDNQb8qSGuzCPcp2\"," +
            "\"firstName\":\"Wenjia\",\"lastName\":\"G\",\"role\":2,\"avatar\":null}},\"message\":\"Success\"}";

    MyHttpUtil.post( apiUrl: MyUrlConfig.user + "/login", login, method: "POST", new MyHttpCallbackUtil() {
        @Override
        public void onSuccess(String data) {
            result = data;
            assertNotNull(result);
            assertEquals(expectedSuccessResponseJson, result);
        }
        @Override
        public void onFailure(String data) {
        }
    });
}
```

Figure 38: Client-API integration test for login endpoint

Client-API Integration Test Results

Figure 39: Client-API integration test results table

Test Name	Class	Description	Expected Output	Passed (Y/N)
testPostSuccess	MuHttpUtilTest	GIVEN correct login credential, WHEN send post request with credential, THEN login success	Login Success JSON response from input stream	Y
testPostFailure	MuHttpUtilTest	GIVEN post request with wrong login credential, WHEN send post request with credential, THEN login failed	User not found Failure JSON response from error stream	Y
testPost WithTokenSuccess	MuHttpUtilTest	GIVEN none-null item object, WHEN send post request with JWT and item, THEN item insert success	Item insert Success JSON response from input stream	Y
testPost WithTokenFailure	MuHttpUtilTest	GIVEN null item object, WHEN send post request with JWT and item, THEN item insert failed	Failed to read HTTP message Failure JSON response from	Y

			error stream	
testGet WithTokenSuccess	MuHttpUtilTest	GIVEN existed supplier id, WHEN send get request with JWT and supplier id, THEN get supplier success	Get supplier Success JSON response from input stream	Y
testGet WithTokenFailure	MuHttpUtilTest	GIVEN non-existent supplier id, WHEN send get request with JWT and supplier id, THEN get supplier failed	Supplier not found Failure JSON response from error stream	Y

9.2.3. Android UI Tests

As shown in the figure below, the client-side Android automated UI tests were implemented using JUnit and Espresso. The use of the Espresso library enables the invocation of Android UI components during testing, which is not offered by Mockito or PowerMock.

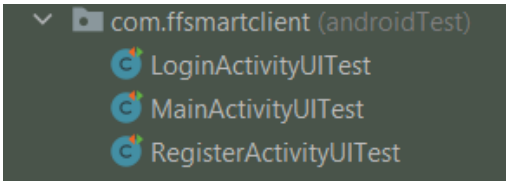


Figure 40: Android UI test filenames

Android UI testing was split into:

1. The functionality of the email/password input components and the login button on the login page.
2. The functionality of the account information input components and the register button on the registration page.
3. The functionality of all buttons on the home page.

The results of the tests were determined by checking whether the application was successfully navigated to the correct activity after the button was clicked.

Automated UI testing eliminates the need for frequent manual regression testing on updated UI component functionality caused by code changes during development. It reduces the probability of manual testing errors and improves overall development velocity. As shown in the figure below, the tests for the main UI elements in all three activities passed. The next section will elaborate on the implementation details of the UI test.

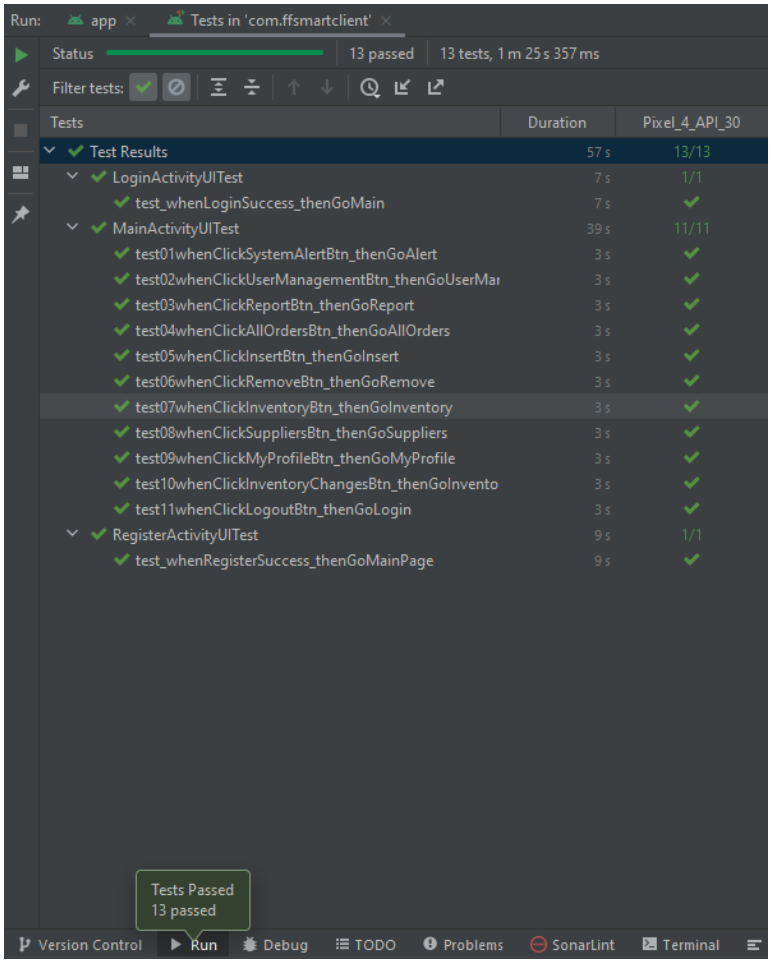


Figure 41: UI test run results in IDE

In the *LoginActivity* test method, the *ActivityTestRule* class is used to create the test rule, followed by a call to the *onView()* method to implement entering the input boxes on the login page with the login email and password and click on the login button. Lastly, the *intended()* method is used to check if the application has navigated to the home page. This test case checks the UI functionalities of clicking the login button to go to the home page after entering correct login credentials.

```
@RunWith(AndroidJUnit4.class)
public class LoginActivityUITest {
    @Rule
    public ActivityTestRule<LoginActivity> rule=new ActivityTestRule<LoginActivity>(LoginActivity.class, initialTouchMode: true);
    @Test
    public void test_whenLoginSuccess_thenGoMain() throws InterruptedException {
        Intents.init();
        onView(withId(R.id.edit_email)).perform(typeText( stringToBeTyped: "t1@gmail.com"),closeSoftKeyboard());
        onView(withId(R.id.edit_password)).perform(typeText( stringToBeTyped: "123123b"),closeSoftKeyboard());
        onView(withId(R.id.btn_login)).perform(click());
        Thread.sleep( millis: 3000);
        intended(hasComponent(MainActivity.class.getName()));
        Intents.release();
    }
}
```

Figure 42: Android UI test for login activity

Android UI Test Results

Figure 43: Android UI test results table

Test Name	Class	Description	Expected	Passed (Y/N)
testLogin	LoginActivityUITest	GIVEN correct login credential, WHEN click login button, THEN go to home page	Go to MainActivity	Y
testSystemAlertBtn	MainActivityUITest	GIVEN on home page, WHEN click alerts button, THEN go to system alerts page	Go to AlertActivity	Y
testUser ManagementBtn	MainActivityUITest	GIVEN on home page, WHEN click user management button, THEN go to user management page	Go to UserAccount Activity	Y
testReportBtn	MainActivityUITest	GIVEN on home page, WHEN click report button, THEN go to safety & health report page	Go to ReportsActivity	Y
testAllOrdersBtn	MainActivityUITest	GIVEN on home page, WHEN click order management button, THEN go to all orders page	Go to OrdersActivity	Y
testInsertBtn	MainActivityUITest	GIVEN on home page, WHEN click insert item button, THEN go to insert item page	Go to InsertActivity	Y
testRemoveBtn	MainActivityUITest	GIVEN on home page, WHEN click remove item button, THEN go to remove item page	Go to RemoveActivity	Y
testInventoryBtn	MainActivityUITest	GIVEN on home page, WHEN click inventory button, THEN go to all inventory page	Go to Inventory Activity	Y
testSuppliersBtn	MainActivityUITest	GIVEN on home page, WHEN click suppliers button, THEN go to all supplier page	Go to SuppliersActivity	Y
testMyProfileBtn	MainActivityUITest	GIVEN on home page, WHEN click my profile button, THEN go to my profile page	Go to MyProfile Activity	Y
testInventory ChangesBtn	MainActivityUITest	GIVEN on home page, WHEN click history button, THEN go to all inventory item change history page	Go to InventoryChange Activity	Y
testLogoutBtn	MainActivityUITest	GIVEN on home page, WHEN click logout button, THEN go to login page	Go to LoginActivity	Y

testRegister	RegisterActivity UITest	GIVEN valid registration information, WHEN click register button, THEN go to home page	Go to MainActivity	Y

9.3 System Tests

9.3.1. Overview

Automated testing has become a particularly crucial component of software development, particularly during the later stages of testing such as system tests, where tests are being performed on the final fully integrated system. During system tests the system is tested as a whole and checked to see if it meets the project's specified functional requirements as informed by the stakeholder.

Katalon Studio is a tool made to support automated testing for a variety of different types of software, such as the Android application produced in this project. It boasts an impressive array of features that allow the tester to automate their manual testing procedures to ensure that tests written are precise, consistent, and repeatable. The ease with which test cases can be created and maintained is one of the major advantages of using an automated testing solution. Test cases can be easily maintained over time in case the application is updated and allows for multiple creation methods such as recording inputs, manual scripting, and UI based editing. The ability to combine groups of test cases into a larger test suite also allows for many tests to be run in series once initially set up, making future testing more efficient.

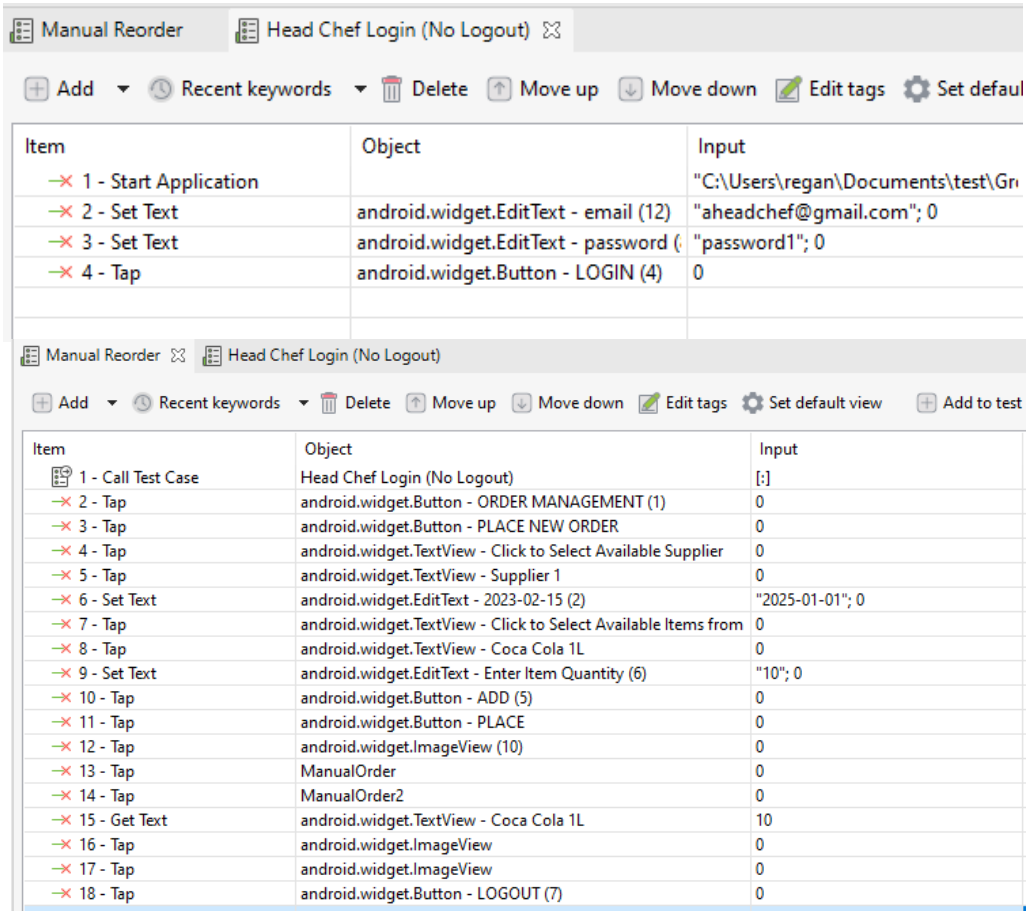
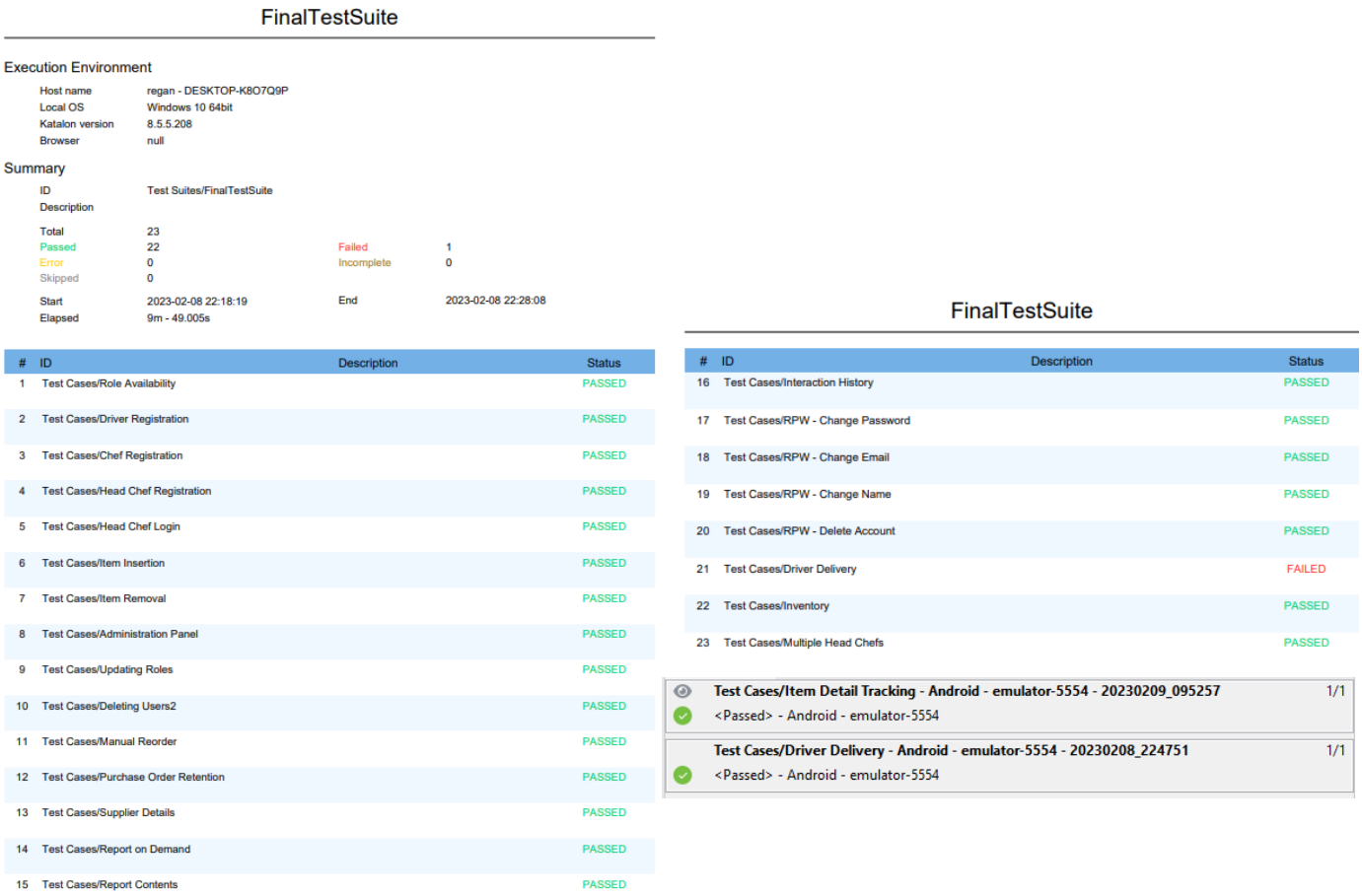


Figure 44: Katalon Studio automated testing

This example test case shows Test Case 11 for automatically testing whether the Head Chef can manually place orders through the application instead of having to place orders to suppliers through traditional methods. The test case begins by invoking a previously defined test case that was defined for the explicit purpose of being used by other test cases that logs in to the application as a sample Head Chef. The usage of reusable 'function' test cases such as these improves the ease of creation and maintenance of the test suite and improves the reliability of the suite by minimising the variance between tests.

Once the automated user has logged in successfully, it simulates tapping on the Order Management button, navigating to the relevant screen, and walks through placing a sample order. Care was taken when selecting the variables used; typing '2025-01-01' means that the expected date of arrival will be in the distant future, and so the order will be at the top of the Order Management screen when sorting by date, removing the need to scroll to find it and thereby minimising the risk of errors. After the order has been placed, the test case verifies that the order has been registered correctly, then simulates navigating back to the home screen and logging out and closing the application. This ensures that the next test case in the suite has a blank slate to work with and reduces co-dependency between test cases.



FinalTestSuite

Execution Environment

Host name	regan - DESKTOP-K8O7Q9P
Local OS	Windows 10 64bit
Katalon version	8.5.5.208
Browser	null

Summary

ID	Test Suites/FinalTestSuite		
Description			
Total	23		
Passed	22	Failed	1
Error	0	Incomplete	0
Skipped	0		
Start	2023-02-08 22:18:19	End	2023-02-08 22:28:08
Elapsed	9m - 49.005s		

#	ID	Description	Status
1	Test Cases/Role Availability		PASSED
2	Test Cases/Driver Registration		PASSED
3	Test Cases/Chef Registration		PASSED
4	Test Cases/Head Chef Registration		PASSED
5	Test Cases/Head Chef Login		PASSED
6	Test Cases/Item Insertion		PASSED
7	Test Cases/Item Removal		PASSED
8	Test Cases/Administration Panel		PASSED
9	Test Cases/Updating Roles		PASSED
10	Test Cases/Deleting Users2		PASSED
11	Test Cases/Manual Reorder		PASSED
12	Test Cases/Purchase Order Retention		PASSED
13	Test Cases/Supplier Details		PASSED
14	Test Cases/Report on Demand		PASSED
15	Test Cases/Report Contents		PASSED

FinalTestSuite

#	ID	Description	Status
16	Test Cases/Interaction History		PASSED
17	Test Cases/RPW - Change Password		PASSED
18	Test Cases/RPW - Change Email		PASSED
19	Test Cases/RPW - Change Name		PASSED
20	Test Cases/RPW - Delete Account		PASSED
21	Test Cases/Driver Delivery		FAILED
22	Test Cases/Inventory		PASSED
23	Test Cases/Multiple Head Chefs		PASSED

Test Cases/Item Detail Tracking - Android - emulator-5554 - 20230209_095257 1/1
<Passed> - Android - emulator-5554

Test Cases/Driver Delivery - Android - emulator-5554 - 20230208_224751 1/1
<Passed> - Android - emulator-5554

Figure 4545: Report produced by the Test Suite

Of the 23 cases in the test suite, 22 test cases passed on the first attempt. This was not unexpected as the system had already been extensively tested internally, and when running the test cases individually the system proved to be robust, reliable and compliant with the Function Requirements laid out by the stakeholder. After analysing the log viewer, the failure of test case 21 was due to the usage of incorrect login details and was adjusted to include

another 'function' test case for logging in as a sample driver. Once the test case had been modified to reflect this change, it ran flawlessly. Additionally, test case 24 had not been added to the test suite. However, after running it individually, this error was also amended.

Some features of the program that required additional manipulation such as the need for multiple emulators to be used concurrently or the editing of variables such as the date were determined to be more effectively tested manually. In cases where setting up automated tests would have been extremely complex or infeasible, it was decided that the cost of configuring the tests would outweigh any of the benefits received from automation. An example of some of these test cases are the test cases for FR12, where a purchase order is generated every Monday, requiring manipulation of the date.

In total 24 automated test cases and 12 manual test cases have been written and ran for a total of 36 test case. All 'Must' requirements (using the MoSCoW prioritisation system) have been tested, as well as all implemented 'Should' and 'Could' requirements, as well as additional features implemented over the course of development that should be system-tested prior to UAT.

When comparing to the initially established requirements, the test suite and manual system tests ensure that all the 'Must include' features have been implemented successfully.

9.3.2. Automated - Must Include

This section of the test suite is arguably the most important; it covers all 'Must include' requirements that do not rely on external factors such as the date and discusses the fundamental features of the program. For the application to be considered successful all these test cases must be passed and will ensure that the product is viable for release.

ID	Test Case 1 – Role Availability	FR ID(s)	Attempt	Result
ST1	Given: I am on the 'Registration' screen When: I look at the 'User Role' options Then: I should see 'Driver', 'Chef' and 'Head Chef' options	FR3	1: Pass	Pass

ID	Test Case 2 – Driver Registration	FR ID(s)	Attempt	Result
ST2	Given: I am on the 'Registration' screen When: I press 'Driver' And: First Name of 'Adam' is typed And: Last Name of 'Driver' is typed And: Email of 'adriver1@gmail.com' is typed And: Password of 'password1' is typed twice And: I press 'Register' Then: I should be registered and logged in.	FR4	1: Pass	Pass

ID	Test Case 3 – Chef Registration	FR ID(s)	Attempt	Result
ST3	Given: I am on the 'Registration' screen When: I press 'Chef' And: First Name of 'Sue is typed And: Last Name of 'Chef' is typed	FR4	1: Pass	Pass

	And: Email of 'achef1@gmail.com' is typed And: Password of 'password1' is typed twice And: I press 'Register' Then: I should be registered and logged in.			

ID	Test Case 4 – Head Chef Registration	FR ID(s)	Attempt	Result
ST4	Given: I am on the 'Registration' screen When: I press 'Head Chef' And: First Name of 'Gordon is typed And: Last Name of 'Ramsey' is typed And: Email of 'aheadchef1@gmail.com' is typed And: Password of 'password1' is typed twice And: I press 'Register' Then: I should be registered and logged in.	FR4	1: Pass	Pass

ID	Test Case 5 – Login	FR ID(s)	Attempt	Result
ST5	Given: I am on the 'Login' screen When: Email 'aheadchef1@gmail.com' is typed And: Password 'password1' is typed Then: I should be logged in.	FR5	1: Pass	Pass

ID	Test Case 6 – Item Insertion	FR ID(s)	Attempt	Result
ST6	Given: I am on the 'Insert' page When: When 'Insert Item' is pressed And: 'Bananas 100g' is selected And: Quantity of '5' is typed And: Expiry date of '2024-01-01' is typed And: I press 'Add' And: I press 'Confirm' Then: I should see the item on the 'Inventory' page	FR1 & FR11	1: Pass	Pass

ID	Test Case 7 – Item Removal	FR ID(s)	Attempt	Result
ST7	Given: I am a Chef or Head Chef When: I press the 'Remove Item' page And: There are 5 lots of 'Bananas 100g' expiring on '2024-01-01' And: I select that item And: Quantity of '3' is typed And: I press 'Add' And: I press 'Confirm' Then: On the 'Inventory' page I should see 2 lots of 'Bananas 100g' expiring on '2024-01-01'	FR1 & FR11	1: Pass	Pass

ID	Test Case 24 – Item Detail Tracking	FR ID(s)	Attempt	Result
ST24	Given: I am a Chef or Head Chef When: I press the 'Inventory' page And: There are 200g of Bananas expiring on 15/02/2023 And: I press the 'Information' icon alongside that item Then: The system should display the item ID, Quantity, Expiry date, and Supplier	FR2	1: Pass	Pass

ID	Test Case 8 – Administration Panel	FR ID(s)	Attempt	Result
ST8	Given: I am a Head Chef When: I press the 'User Management' Page Then: I should see a list of users.	FR6	1: Pass	Pass

ID	Test Case 9 – Updating Roles	FR ID(s)	Attempt	Result
ST9	Given: I am a Head Chef When: I press the 'User Management' Page And: I press the 'Information' icon alongside 'Test User' And: I press 'Driver' And: I return to the 'User Management' Page Then: I should see that 'Test User' has been updated to be a 'Driver'.	FR6	1: Pass	Pass

ID	Test Case 10 – Deleting Users	FR ID(s)	Attempt	Result
ST10	Given: I am a Head Chef When: I press the 'User Management' Page And: I press the 'Information' icon alongside 'Test User' And: I press 'Delete' And: I return to the 'User Management' Page Then: I should see that 'Test User' has been removed.	FR6 & FR29	1: Pass	Pass

ID	Test Case 11 – Head Chef Manual Re-Order	FR ID(s)	Attempt	Result
ST11	Given: I am a Head Chef When: I press the 'Order Management' page And: I press 'Place New Order' And: 'Supplier 1' is selected And: Delivery date of '2025-01-01' is typed And: 'Coca Cola 1L' is selected And: Quantity of '10' is typed And: I press 'Add' And: I press 'Place' Then: The system must display the placed order on the 'Order Management' page with the correct details.	FR14	1: Pass	Pass

ID	Test Case 12 – Purchase Order Retention	FR ID(s)	Attempt	Result
ST12	Given: I am a Head Chef When: I press the 'Order Management' page And: Select 'Delivered' in the dropdown menu Then: I should be able to see past orders.	FR15	1: Pass	Pass

ID	Test Case 13 – Supplier Details	FR ID(s)	Attempt	Result
ST13	Given: I am a Chef or Head Chef When: I press the 'Suppliers' page And: I press the 'Information' icon alongside 'Supplier 1' And: I return to the 'Suppliers' page And: I press the 'Information' icon alongside 'Supplier 2' Then: I should see a different list of items being sold by each supplier.	FR18	1: Pass	Pass

ID	Test Case 14 – H&S Report on Demand	FR ID(s)	Attempt	Result
ST14	Given: I am a Head Chef When: I press the 'Reports' page And: I press the 'Send New Report' button Then: I should see a new report generated.	FR20 & FR21	1: Pass	Pass

ID	Test Case 15 – H&S Report Contents	FR ID(s)	Attempt	Result
ST15	Given: I am a Head Chef When: A report has been generated Then: The report should contain a list of any items to ever expire inside the fridge.	FR22	1: Pass	Pass

ID	Test Case 16 – Interaction History	FR ID(s)	Attempt	Result
ST16	Given: I am a Chef or Head Chef When: I press the 'Reports' page Then: I should be able to see all previous interactions made with the fridge, and the user associated with the action.	FR24	1: Pass	Pass

ID	Test Case 21 – Driver Delivery	FR ID(s)	Attempt	Result
ST21	Given: I am a Driver When: I press the 'All Orders' Page And: I press the 'Information' icon alongside the order And: I press the 'Dispatch Order' button And: I press the 'Deliver' button Then: The order should be delivered to the fridge.	FR7 & FR8	1: Fail 2: Pass	Pass

9.3.3. Automated - Should/Could Include

This section of the test suite covers functions of the system that were assessed to be possible inclusions but are not fundamental to the success of the project. Therefore not all 'Should' or 'Could' include features were implemented. Instead 'Additional Features' were implemented which the team determined would be better suited to the needs of the client. All test cases passed on the first run of the test suite.

ID	Test Case 17 – Change Password	FR ID(s)	Attempt	Result
ST17	Given: I am logged in as 'Regan Pierre White' When: I am on the 'My Profile' page And: Password of 'password2' is typed twice And: I press update And: I log out And: I log in using 'rpw@gmail.com' and 'password2' Then: I should be taken to the home screen.	FR26 & FR27	1: Pass	Pass

ID	Test Case 20 – Account Self-Deletion	FR ID(s)	Attempt	Result
ST20	Given: I am logged in as 'Regan Wills' When: I am on the 'My Profile' page And: I press 'Delete' And: Email of 'rpw@outlook.com' and password of 'password2' are typed Then: I should be unable to login, as my account has been deleted.	FR28	1: Pass	Pass

ID	Test Case 22 – Inventory	FR ID(s)	Attempt	Result
ST22	Given: I am a Chef or Head Chef When: I press the 'Inventory' page Then: I should be able to see a list of items currently stored in the fridge.	FR34	1: Pass	Pass

ID	Test Case 23 – Multiple Head Chefs	FR ID(s)	Attempt	Result
ST23	Given: I am on the 'Registration' screen When: Head Chef 'Gordon Ramsey' already exists within the system	FR42	1: Pass	Pass
	And: I sign up as a new 'Head Chef'			
	Then: I should be allowed to login and use both profiles as normal.			

9.3.4. Automated - Additional Features

ID	Test Case 18 – Change Email	FR ID(s)	Attempt	Result
ST18	Given: I am logged in as 'Regan Pierre White' When: I am on the 'My Profile' page And: Email of 'rpw@outlook.com' is typed And: I press update		1: Pass	Pass

	And: I log out And: I log in using 'rpw@outlook.com' and 'password2' Then: I should be taken to the home screen.			

ID	Test Case 19 – Change Name	FR ID(s)	Attempt	Result
ST19	Given: I am logged in as 'Regan Pierre White' When: I am on the 'My Profile' page And: Name of 'Regan Pierre Wills' is typed And: I press update And: I log out And: I log in using 'rpw@outlook.com' and 'password2' And: I am taken to the home screen. Then: The name at the top of the screen should read 'Regan Wills'.		1: Pass	Pass

9.3.5. Manual - Must Include

As previously discussed, it was unfeasible to perform all tests using Katalon Studio. Tests that require the manipulation of variables, multiple emulators or otherwise are unable to be assessed by an automated test suite, such as test case 30 which requires the Health and Safety Report to be produced an email-friendly format.

ID	Test Case 25 – Checking Function		FR ID(s)	Attempt	Result
ST25	Description:	In its current implementation, the checking function works by randomly causing orders to fail when the Driver delivers them.	FR9	1: Pass	Pass
	Manual?	The reason this was tested manually is by default the chance of a checking function failing is 10%, testing manually allows for this to be set to 100%.			

ID	Test Case 26 – Checking Function Alert		FR ID(s)	Attempt	Result
ST26	Description:	If the checking function determines that the delivery has failed, it sends an alert to the Head Chefs	FR10	1: Pass	Pass
	Manual?	This was tested manually alongside ST25, as it is dependent on that test case determining the results of the checking function.			

ID	Test Case 27 – Purchase Order every Monday		FR ID(s)	Attempt	Result
ST27	Description:	As well as the Head Chef being able to manually generate purchase orders, every Monday an order should be generated containing items that have run out of stock, or are due to run out of stock within the next 3 days.	FR12 & FR17	1: Pass	Pass

	Manual?	This functionality will only be activated once per week, and so variables must be modified to reproduce the scenario on demand.			

ID	Test Case 28 – Purchase Order Alert		FR ID(s)	Attempt	Result
ST28	Description:	When the weekly purchase order is generated, an alert should be sent to the Head Chef's device.	FR13	1: Pass	Pass
	Manual?	This test case is directly tied to ST27 and should be tested at the same time.			

ID	Test Case 29 – Due to Run Out Alert		FR ID(s)	Attempt	Result
ST29	Description:	When the system detects that an item is due to run out within 3 days, an alert should be sent directly to the Head Chef. This is a separate functionality, as depending on the scenario the Head Chef may wish to place an order separately to the weekly automatic reorder, for example: if the system detects that the restaurant will run out of Chicken within 3 days on a Tuesday.	FR16	1: Pass	Pass
	Manual?	This test case requires similar setup to ST27 & ST28 and therefore would be sensible to test subsequently.			

9.3.6. Manual - Should/Could include

ID	Test Case 30 – H&S Report Format		FR ID(s)	Attempt	Result
ST30	Description:	The Health and Safety reports should be in a PDF format so they can easily be stored and emailed to the relevant Health and Safety body.	FR35	1: Pass	Pass
	Manual?	The report files are saved to the user's device and therefore cannot be detected by Katalon Studio.			

9.3.7. Manual - Additional Features

Whilst not strictly related to the initially established Functional Requirements, over the course of development many new features were implemented to extend the functionality of the program and improve the experience for the end users, that also require testing. Unlike the features set out by the Functional Requirements, some of these features are less suited to automated testing as their usage is harder to measure, for example the introduction of profile pictures and reports being accessible in-app, each of which require a more qualitative approach. Exceptions to this such as the ability to change the name and email address associated with an account have been included in the Katalon Studio test suite.

ID	Test Case 31 – Android Client		Attempt	Result
ST31	Description:	By using the Android APK file or enabling Android Studio's experimental features the client can be used directly on an Android device, simulating the experience had by end-users in the restaurant. Where possible tests should be carried out in this mode to ensure the environment is as similar as possible to its target usage.	1: Pass	Pass
	Manual?	Katalon Studio requires an emulator to be used and cannot be ran on physical mobile devices.		

ID	Test Case 32 – Order Caching		Attempt	Result
ST32	Description:	Caching has been implemented when placing orders. If an order has been created but not placed, when backing out and returning to the order creation screen, the unplaced order can continue to be edited or placed.	1: Pass	Pass
	Manual?	Katalon Studio's had trouble with some implementations of caching and was therefore tested manually.		

ID	Test Case 33 – Insertion Caching		Attempt	Result
ST33	Description:	Caching has been implemented when inserting items. If a list of items has been created but not confirmed, when backing out and returning to the 'Insert Item' screen, the list of items can continue to be edited or inserted.	1: Pass	Pass
	Manual?	Katalon Studio's had trouble with some implementations of caching and was therefore tested manually.		

ID	Test Case 34 – Removal Caching		Attempt	Result
ST34	Description:	Caching has been implemented when removing items. If a list of items has been created but not confirmed, when backing out and returning to the 'Remove Item' screen, the list of items can continue to be edited or removed.	1: Pass	Pass
	Manual?	Katalon Studio had trouble with some implementations of caching and was therefore tested manually.		

ID	Test Case 35 – Viewable Report		Attempt	Result
ST35	Description:	The Health and Safety reports should be viewable in-app. They should also be able to be scrolled or 'pinched' to zoom.	1: Pass	Pass
	Manual?	Katalon Studio has a variety of ways to interact with the emulated Android device such as short and long taps, as well as swiping and scrolling until it reads certain texts, however it is unable to 'pinch' and therefore cannot zoom.		
ID	**Test Case 36 – Password Masking**		**Attempt**	**Result**
ST36	Description:	When a user is entering the password, rather than displaying the password as typed, instead an equivalent number of 'dots' or 'pips' should be displayed. For example 'password1' should become '●●●●●●●●●'.	1: Pass	Pass

	Manual?	Katalon Studio may not recognise the dots and instead read the text, even though visually the password is masked correctly		

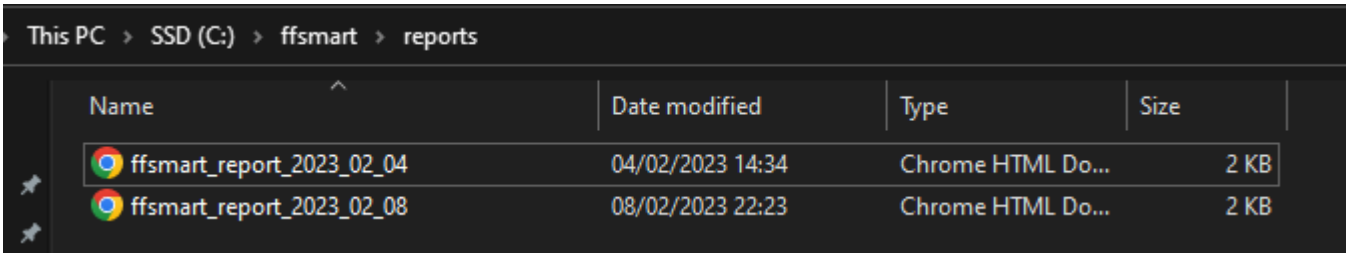


Figure 4646: An example of reports saved as PDF files (using Chrome as the default PDF viewer).

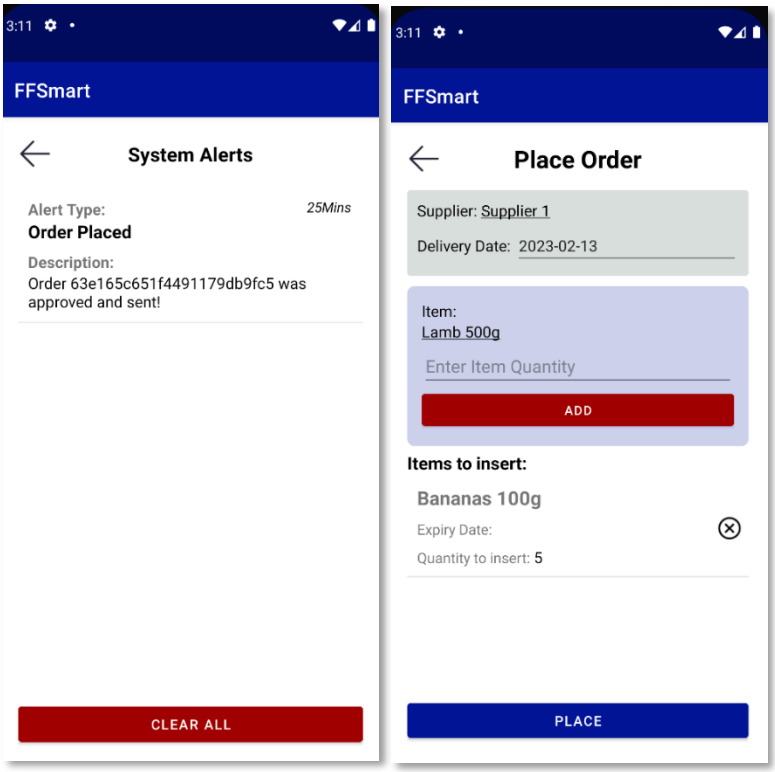


Figure 4747: An alert shown on another Head Chef's account after placing an order.

9.3.8. Implementation

A total of 36 test cases have been produced and run, with all but one passing on the first attempt and all test cases ultimately proving successful. Every 'Must Include' Functional Requirement has an associated test case proving that it has been met successfully. Additionally, every 'non-essential' feature implemented by the development team has been tested to ensure it functions as intended.

A full video of the 23 test cases featured in the automated test suite is available at: https://youtu.be/a7J6VNhuOqs (accessed 09/02/2023).

As the project developed it became clear that some of the optional features set out in the Functional Requirements were either unnecessary, low-priority, were not aligned with the direction of the project, or were too far out-of-scope to implement. When coding a project that had to be finished within a confined timeframe, it was important to ensure that effort was being spent wisely and that all features implemented would directly improve the experience of the end-users.

One feature that the team elected not to implement was the Administrator role, which would not have access to the physical fridge but would instead be able to manage users and orders. This role may have been useful for some restaurant setups where the book-keeping or inventory management was done by an owner or manager. Ultimately, the development team integrated these permissions as part of the Head Chef role and implemented a 'control panel' styled section of the home page containing administrative functions visible only to the Head Chef. This delineation ensures that the UI remains simple to learn and use, and if an Administrator role is required by the restaurant they can simply register as a Head Chef and utilise the 'control panel' feature.

Some features, such as recognising the difference between the front and rear doors of the fridge, have been implemented implicitly. For example, Chefs inserting into the front door of the fridge have options to sort the items, as well as see all their details such as precise quantities and expiry dates of any item. Drivers accessing the rear door to insert deliveries are unable to see the inventory and are instead given a 'My Orders' page with a simple button to confirm a delivery has been made. This implementation improves the experience of both users and maximises utility as it allows the Chefs to see all the details they need whilst enhancing the simplicity and speed of the Driver's UI, allowing them to make more deliveries.

Furthermore, several features suggested in the Functional Requirements could have been beneficial but would have changed the function of the application too significantly to be worth implementing. An example of this was the suggestion of implementing communication channels between the Head Chefs and Drivers. While a feature such as the ability for a Head Chef to leave reviews, or for the Drivers to communicate details about their deliveries may be useful, it was not suggested by the stakeholder and would have significantly affected the capabilities and UI of the application, and so was decided against given the remaining timeframe for the development phase of the project. Instead, the choice was made to prioritise meeting the stakeholder's core requirements and to keep the UI as uncluttered as possible.

Other features suggested may have subjective utility and would require further dialogue with the stakeholder. An example of this was automatically logging out users after two minutes of inactivity. While this feature may have been beneficial to include, some restaurants or Future Fridges themselves may object to it, as it could make the application more difficult to use, particularly when first adopting the technology.

The decision to omit these kinds of features allowed the development team to devote additional time to alternative additional features that would have a larger impact on improving the application for end users. These features range in size and complexity, from large-scale improvements such as the Android client to small quality-of-life improvements such as implementing password hashing that otherwise may not have been completed.

9.4. Acceptance Tests

Acceptance Testing is the final stage of the development process of this project and will demonstrate that the software produced meets the end-users' needs and expectations and is therefore ready to be deployed. User Acceptance Testing (UAT) is intended to simulate the experience of real users as closely as possible, and ideally would take place in a live environment with real external users such as in alpha or beta testing. Unfortunately, in this case this form of testing was unfeasible, and instead the tester will use the established use cases and their knowledge of the system's capabilities to produce 'user journeys' or 'user stories' that ensure all requirements and expectations of the software are met. The focus of these user journeys is not to uncover bugs or errors but rather to ensure that the software has been interpreted and implemented as intended from the end user's perspective.

User Story 1	
User:	As a busy chef I must be able to login and scan the inventory quickly so I can check what food is available when gathering ingredients for orders.
Acceptance Criteria:	I should be able to login using my unique credentials. The login process should take no more than a couple of seconds, and preferably under a second. The Inventory page should be directly accessible from the main menu. The Inventory page should be sensibly sorted and organised in an easy-to-read manner so I can quickly identify what ingredients are available. When I return to the app, I should still be on the Inventory page.
Use Case Tested?	UC6, UC5
Result:	- Most criteria achieved successfully. - Items on the Inventory page are sorted newest to oldest, it may be preferable to sort oldest to newest or by ingredient, although additional options may clutter the UI. - Current page is not cached, however if the device is powered off or the app is minimised the user remains logged in and on their current page.

User Story 2	
User:	As a trainee chef I must be able to remove and insert items precisely so I can return ingredients if I pick up the wrong ones.
Acceptance Criteria:	I should be able to access the removal and insertion pages directly from the main menu. Both the Removal and Insertion pages should display all known details about the item, including the name, quantity, and expiration date. The application should require me to confirm removal or insertions before executing the change within the system. The UI should be as simple as possible to avoid overwhelming me. The UI should assign different colours/symbols to different elements of the UI to make it more learnable. The system should provide clear feedback that an item has been successfully removed or inserted.
Use Case Tested?	UC1, UC2
Result:	- Most criteria achieved successfully. - If the fridge is especially full it may be difficult to find items from the remove items menu. This could be solved by displaying only the item types rather than individual items, however it would also add an extra step to the removal process as expiry date would need to be set separately.

	- The search bar helps minimise this problem; could an autocomplete or search after every letter entered be implemented? - Item quantity isn't displayed in application when removing items, however the user will be able to see the quantity in real life, and the application doesn't allow more than is currently in the fridge to be removed.

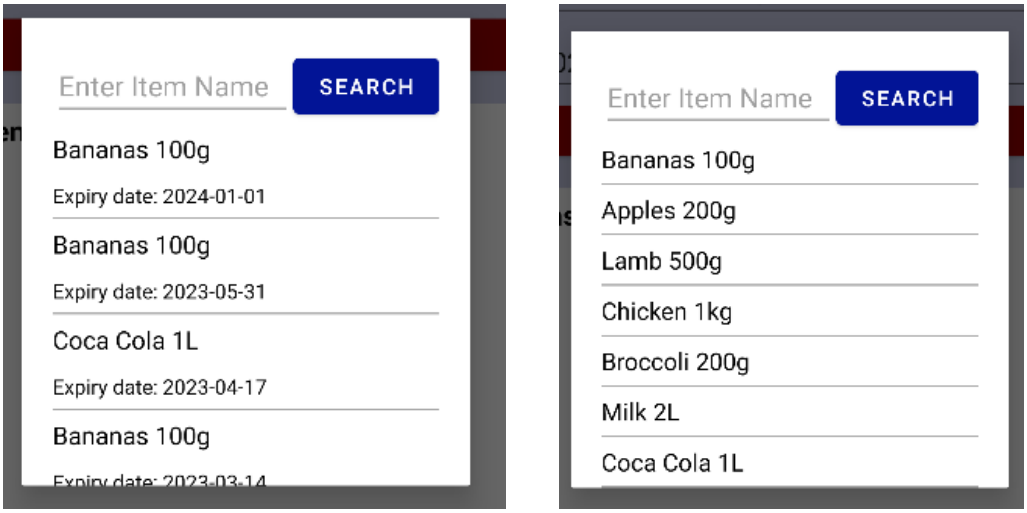


Figure 4848: The item removal pop-up (left) as opposed to the insertion pop-up (right)

User Story 3	
User:	As an elderly chef with poor vision and colour-blindness I must have a UI catering my needs so I can complete my tasks effectively.
Acceptance Criteria:	UI designed with accessibility in mind for colour-blindness and poor vision: - Font size should be easily readable. - Colour contrast between texts and backgrounds should be high enough to be distinguishable. - Care should be taken to avoid colours difficult to read for common colour-blindness issues (red/green and blue/yellow). - The system should provide audio feedback when buttons have been pressed. - Layout should be simple and straight forward. Inventory should be accessible to assist those with mobility issues. Buttons should be large enough for someone with shaky hands to use.

Figure 4949: The main menu screen on a landscape Nexus 10 tablet

Use Case Tested?	-
Result:	- Generally Successful, consideration has clearly been considered. - Most accessibility issues are helped by the UIs responsiveness to different screen sizes as well as adapting between portrait and landscape mode; if a mobile phone proves difficult to use, an Android tablet in portrait or landscape mode would likely provide a screen large enough for this user's needs. - There are no instances of contrasting colours being laid on top of each other. - A font size option or UI scale option could still be included.

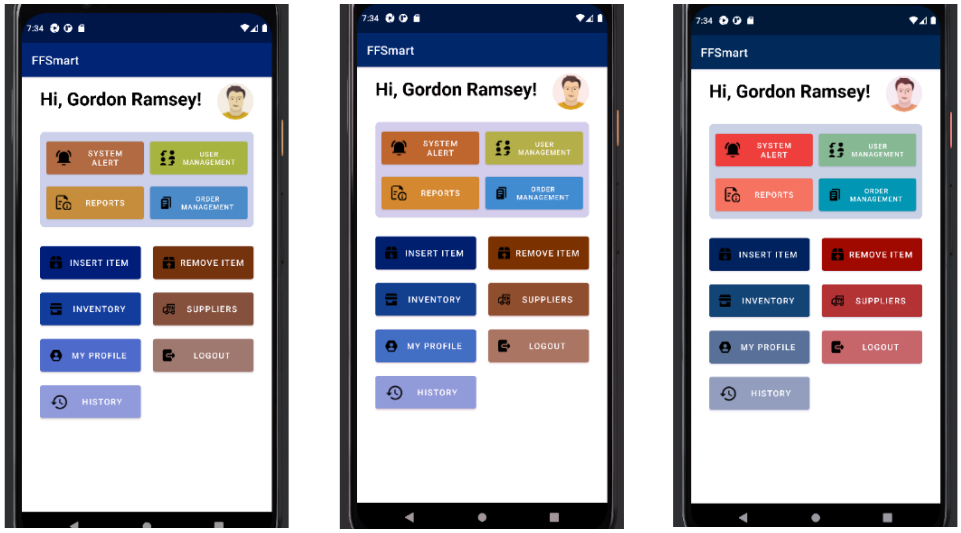


Figure 5050: Pictured from left-to-right are Protanomaly (Red-Weak), Deuteranomaly (Green-Weak) and Tritanomaly (Blue-Weak). (Coblis, 2021)

User Story 4	
User:	As a conscientious head chef I must be able to clear expired food and generate Health and Safety reports at any time so I can make sure that no staff or customers get sick.
Acceptance Criteria:	I should be able to clear all expired items with a single button after cleaning the fridge. An alert should be sent to my device when items are going to expire soon. I should be able to generate a Health and Safety report at any time Health and Safety reports should be saved for filing purposes and to show relevant Health authorities. Health and Safety reports should also be viewable within the app.
Use Case Tested?	UC4, UC11
Result:	- Generally Successful. - The ability to scroll and zoom in to Health and Safety reports is a great feature and shows the abilities of the application have been maximised for mobile devices. - Perhaps the ability to write notes or comments in the Health and Safety report could be useful to provide additional context or explanation?

User Story 5	
User:	As a pizza takeaway owner with a high turnover of staff I must be able to easily manage user details so I can change my cooks' roles to driver or head chef and delete users who have left.

Acceptance Criteria:	I should be able to view a full list of drivers, chefs and head chefs that have access to the fridge. I should have the ability to edit users' roles. 'Roles' should handle user permissions automatically, so I don't need to configure each user manually. I should be able to delete or deactivate staff that are no longer employed at the takeaway; all their details should be deleted to ensure compliance with relevant data protection laws. Sign-ups for new users should be quick and easy. The system should track interactions with the fridge so I can maintain accountability amongst staff.
Use Case Tested?	UC7, UC8, UC9
Result:	- Very Successful. - Users can currently reset their own passwords; however the ability could also be added for Head Chefs to reset the passwords of Drivers or Chefs if they are locked out of their accounts. Currently this issue can be avoided by the Head Chef deleting the account, and the Driver or Chef signing up again with the same details.

User Story 6	
User:	As a new kitchen manager, I must be able to view a history of previous orders and fridge usage, so I can get to grips with my new job.
Acceptance Criteria:	I should be able to access previous orders. I should be able to access interaction history with the fridge. I should be able to view a list of users. I should be able to view past Health and Safety reports. I should be able to view the details of Suppliers associated with the restaurant. Weekly purchase orders should be generated for me, to lighten my workload early on. Daily Health and Safety reports should be generated for me, to lighten my workload early on. I should be alerted when we are due to run out of an ingredient.
Use Case Tested?	UC10
Result:	- Very Successful. - Some additional features could be added such as visual representations of user interactions (such as who is using the fridge the most) and of item usage (which items are being removed and at what rate). - The due to run out feature could also be extended to visually show a longer-term prediction of ingredient usage based on past usage.

User Story 7	
User:	As an impatient driver I must be able to deposit deliveries without fuss so I can continue to my next destination.
Acceptance Criteria:	I should be able to make a delivery with the minimum number of steps possible. Non-essential data such as expiry dates of individual items don't need to be displayed. The UI should be as clear and concise as possible. The system should validate deliveries as quickly as possible. I should have access to a history of deliveries I have made. The system should be reliable and have minimal downtime or maintenance time. Sign-ups and logins should be as simple and fast as possible.

Use Case Tested?	UC1
Result:	- Generally Successful. - Difficult to test downtime and maintenance in real scenarios without beta testing. - Current page is not cached, however if the device is powered off or the app is minimised the user remains logged in and on their current page.

User Story 8	
User:	As a transgender driver I must be able to update my name and profile picture so I can reflect changes to my identity.
Acceptance Criteria:	I should be able to change my name within the app. I should be able to change my profile picture within the app. These changes should be saved and displayed consistently throughout the application, such as in the main menu and profile screens. These changes should be reflected on other user's devices. Changing details should not affect my ability to perform the functions I previously could.
Use Case Tested?	-
Result:	- Very Successful. - Gender is not required when using the system, so it is a good idea not to track it. - One feature that should be implemented is encryption to ensure that information saved to the database is kept confidential and secure.

The UAT phase of testing has been remarkably successful. Eight user journeys were created and tested to evaluate the system's performance and ensure that user's needs were met by the program in a situation as close to a live environment as possible. All eight users were able to perform their desired tasks effectively with all basic needs and requirements met, and in some cases, users benefitted from advanced functionality beyond their expectations. No bugs or errors were experienced during the running of these user stories, which is to be expected thanks to the extensive testing that has already been carried out.

While these results are encouraging, it is important to note that testing was carried out in a controlled environment by the test engineer using emulated user stories. This method still provides worthwhile results but may not accurately reflect the performance of the system and requirements of users in a real-life scenario. It would have been preferred to conduct an alpha or beta test with real potential users interacting with the application for the first time to gain a better understanding of the actual usage of the system.

Some of the improvements and considerations suggested in the 'Result' rows may not have been implemented due to the constraints of the current scope of the project rather than lack of forethought. Regardless, the insights they provide may be useful when considering any futures updates or improvements to the system. Overall, the system has been deemed fit for submission to Future Fridges and it is believed that the system will provide a positive user experience for its intended end-users.

9.5. Other Tests

9.5.1. API Performance Testing with JMeter

When commencing performance testing, in order to be able to execute more extensive and high-load tests, the backend was deployed to the cloud using Azure Spring Apps. This was done

to allow for vertical and horizontal scaling, and to give a better idea of how the application would run if deployed on an in-house server in a restaurant setting.

First, with an initial 1GB of memory and 1 vCPU, testing was performed on the GET /inventory endpoint, which would likely be the most used endpoint of the application if deployed. 100 users were simulated with a ramp-up period of 1 second and a loop count of 10. This represents a peak load that the API could experience once deployed. As shown in the results graph below, the API struggled to cope with the high number of users in such a short period of time, with an average response time of 5.8s, and the longest response time being around 7s. By any modern standard for responsiveness, this would be considered a failure, and would likely harm the restaurant's overall speed of service.

Figure 5151: Performance testing without vertical scaling

We then added some vertical scaling through the Azure Spring Apps service, so that a server with greater compute power could be simulated. The memory was doubled to 2GB, still with 1 vCPU due to Azure access restrictions. Using the same test configuration of 100 users and 1s ramp-up period, the results were slightly better:

Figure 5252: Performance testing with vertical scaling

This time, the average response time was 5.7s, with a similar throughput, but a lower peak response time of 6.3s, 10% lower than with 1GB of memory. Thus, it was concluded that vertical scaling could lead to large improvements in response time, but that most of these gains would be made when the number of CPUs and memory are increased significantly (for example, 8 CPUs and 16GB memory).

Horizontal scaling (increasing the number of servers and spreading the request load between them) is also an option for deployment in restaurants and is something that can be simulated effectively with Azure. The instance count was increased from 1 to 4, maintaining the 2GB of memory from the previous test, and a load balancer was integrated to optimise performance by spreading the load between the instances, simulating the setup of four servers on a local area network. Then, the same test was carried out once more.

Figure 5353: Performance testing with vertical and horizontal scaling

This time, the average response time decreased to 937ms with a maximum of just over 1s, representing a 5x improvement in performance. Although 1s is still considered relatively slow by modern standards, this is significantly better than 5s and meets the performance requirements of the system. Introducing more server instances (8-12) would likely have an even greater effect. However, vertical scaling would be more suited to the use case of deployment in restaurants due to its higher maintainability, relatively lower complexity, and lower overall costs. Achieving this same performance with vertical scaling would require adding several more CPUs as well as a sizeable amount of RAM.

9.5.2. Justification for Omission of Other Testing Types

Several alternative testing types outlined in the module content were not included in the project's software testing. These omissions are justified by the following:

- Regression testing: it was determined that, due to the project following Waterfall methodology, the majority of testing would be done after development has finished. The purpose of regression testing is to ensure that major code changes do not produce new defects or break the application. Thus, it was agreed that regression testing would not be necessary in this case due to the strict timeframe for completion and because all of the major code changes had already been made.
- Usability testing: it was thought that the usability of the application would be sufficiently tested during user acceptance testing and system testing. Also, usability testing is commonly performed with end users, which we did not have access to.
- A/B testing: this is a unique form of testing that is statistical in nature and involves collecting data from users on two variants of the application (A and B), in order to

determine which is more effective. This type of testing is commonly considered to be effective for measuring user engagement and satisfaction. However, A/B requires a substantial number of end-users, as well as specialised event collection software and an extended time period, for it to be carried out effectively. It was decided that this would not be an inefficient use of project time and resources.

9.6. Test Summary

In summary, the system has been shown to meet all core requirements through all unit, integration, and system tests having passed. Before each stage of testing began, it was made sure that all tests from the previous stage were executed successfully. Further to this, user acceptance testing performed with a wide variety of potential users has demonstrated that the system would be fit for use in restaurants and meets the criteria specified by the client in the scenario document and follow-up meeting. The overall test coverage is good (~80% on server tests), with no significant components or functionality left untested. The fact that all unit and integration tests passed on the API shows that it is highly stable, and performance testing has suggested that with some vertical or horizontal scaling, it would be able to cope with a high volume of simultaneous requests. As it is, both the API and Android UI are fully able to cope with expected request loads without producing slow response times or exceptions, and thus the system can be considered stable and effective.

Testing stages summary:

- Unit testing validated individual components of code on both client and server-side, ensuring they are fit for use and functioning as expected.
- Integration testing confirmed that these individual components had successfully been combined together and interacted as expected.
- System tests evaluated the functionality and stability of the application in its entirety, confirming that functional requirements had been met successfully and the application did not encounter any exceptions during normal use.
- User acceptance testing was performed to emulate the experience of real users in a live environment and ensure that a positive user experience had been implemented with all needs and expectations met.
- Finally, performance testing confirmed that the system would be able to meet the required rigours of usage in a live environment, and what measures could be taken to improve performance.

It can be confidently stated that the software simulation is ready for approval from Future Fridges. The testing phase has provided some valuable insights into the capabilities of the system and has highlighted several areas that could be improved in future versions. The success of this phase and general lack of bugs and other defects is testament to the quality of work produced by the development and testing teams.

10. Project Execution and Management

Team organisation

During the development process, the team met each week online via MS Teams to review the progress of the application and the report. In addition to this, the developers met for 30 minutes to an hour every day to share ideas and issues they were having. The project plan was followed closely, with the only deviation being the design and development phases taking longer than expected.

All tasks and bugs were tracked using Jira tickets, with each ticket going through analysis, review, and testing statuses before eventually being marked done once it had passed testing. Each member of the development and testing teams checked the Jira project each day. Project managers oversaw creating new tickets (Apart from bug tickets that were raised by developers) and assigning them to team members. The tickets would then be 'picked up' and work would start. Risks were managed and mitigated successfully according to the risk assessment shown in section 6.

Development commenced in early January, with the software being planned and broken down into small tasks to enable higher developer velocity and more accurate task tracking (see section 6.5). A prototype version of the application was produced and showcased to the rest of the team in mid-January. The decision was made to go with a client-server architecture, with a reactive server and MVC architecture being used on the client. Java was used on both client and server thanks to its substantial platform independence. An Android app was developed as the client, and Spring framework was used for the server to create a responsive and high-performing RESTful API. The UI design was used extensively when building the software, as well as the ISO 25010 coding standards, with SonarLint being used to ensure high code quality. All core requirements were implemented first, with the later addition of extra features, such as downloading and viewing health and safety reports in PDF format. Pair programming was used extensively, with the server and client being developed concurrently by the development team.

A positive and supportive team environment was cultured by the project managers and development team, where team members would feel comfortable asking for help and giving feedback to other members. The Group 11 Teams channel was used to organize meetings, discuss issues, and share important updates. Finally, it was ensured that there were clear processes in place for reporting bugs and evaluating the system during and post-development, and this proved to be critical in developing a quality final product.

Review of Software Methodology Used

The project was executed using the classic Waterfall methodology, with each major project phase taking place after the previous phase had reached completion to the required standard, and a strong focus on establishing a high quality and comprehensive requirements towards the start of the project.

Using Waterfall, the team was able to provide a concrete project plan with attainable and clear target dates for the completion of each stage. In addition, expected deliverables were confirmed for each stage, and this served as a form of progress tracking throughout the course of the work. Combined, these defined a clear project structure that the team could stick to, in the knowledge that if the plan was kept to and risks mitigated, the project had a high chance of being delivered in good time.

Waterfall also suited the style of client interactions that we experienced, with only one client meeting towards the start of the project. With Agile and similar methodologies, it is common practice that there are frequent, sometimes even daily, meetings with the client. Once we had established a clear set of requirements, the client did not require frequent updates or meetings, and there was little risk of changing requirements, which is ideal for Waterfall projects.

On the other hand, using Waterfall meant that we had less flexibility in the project schedule, and changes could not be made to already complete work in a previous phase. However, this did not result in any major issues, partly due to the extensive collaboration and frequent internal work reviews. It was also difficult for testing to result in any changes in requirements, as requirements had been decided months before testing began. As well as this, Waterfall has the issue that only certain members of the team are able to work on it at once. For instance, in the development phase, only the developers are able to actively contribute to the project. These said tasks were found for the rest of the team members at each stage, principally associated with writing the report.

Overcoming Risks

The first risk identified in the risk assessment was that of scope creep and changes to the design. Some minor changes to the design were required due to difficulties in implementation client-side and changes in the model on the backend to adapt to the choice of database service. However, as the changes were only minor, these did not have a large impact on the project schedule, or the total development work required.

Next, it was noted that assignments from other modules may affect progress on the project and delay development work. Although there was significant work set from other modules, it cannot be said that these assignments had a significant impact on the project. This was also ensured through regular check-ups with the team on progress and availability.

The lack of coding standards was also identified as a relatively severe risk to the system and the project as a whole. Early in the project it was determined that the team would adhere to the ISO-25010 guidelines and that a code-checking tool would be used to help enforce this, so this risk did not occur. Inaccurate effort estimation was also recognized as a risk, with a low possibility of happening. The cost/effort estimation proved to be highly accurate, with the man-hours mapping closely to the actual hours put in by the development and testing teams.

A lack of experience in developing similar products was also pointed out as a relatively severe risk. However, the initial scenario document from the client was reviewed in-depth by the team, and an extensive list of clarifying questions was posed to the client in a follow-up meeting. This in-depth review and clarification process served to prevent misunderstanding of any requirements. Additionally, the conflict between team members was identified as a severe but unlikely risk. As predicted, this risk did not occur during the completion of the project.

Lastly, the risk of inadequate test cases was recognized as a severe risk to the project delivery. In order to mitigate this, test cases were reviewed carefully by developers and the rest of the team, and it was checked that each test case was carried out successfully at various testing levels (unit to UAT).

Other Risks That Occurred

There were several delays in the project schedule due to team member absence, particularly in December, when two members, including a project manager, caught COVID-19, and were unable to contribute for 2 weeks. Fortunately, since we had two project managers, the other was able to take on his tasks and keep the project on track. When roles were assigned to the team at the start of the project, we considered the risk of illness or member loss, and thus multiple roles were given to each team member in order to maintain the project flow even if several members are unable to work. At the same time, it was understood that this could lead to conflict as multiple team members could attempt to pick up the same work. To solve this problem, progress on tasks was frequently monitored and task assignees tracked using Jira.

In terms of technical risks, Java version compatibility was a significant concern. To mitigate the risk of application defects associated with incompatibility, it was ensured that both the client and server used the same JDK (19) and therefore the same Java version. Later in the project, when the server was deployed to Azure Spring Apps for performance testing, the latest accepted JDK version was JDK 17. So as not to risk breaking the application, a copy was made of the repository locally, and it was compiled to a JAR file again, this time with JDK 17, before being deployed to the cloud. The effect of changing from JDK 19 to 17 did not result in any required code changes or other configuration updates.

Following the Original Project Plan

The initial plan was committed with only minor adjustments to ensure that the targets of each stage were reached. The initial plan gave a clear road map for the project, which made it easier for every member to stay on track and prevent any significant delays. However, as the project developed, a few modest adaptations to the plan were required as some parts of the project took longer than expected, such as the development and design phases. Moreover, various external factors affected the project's progress timeline, such as commitments to other modules and team member illness. These were the main reasons for the delay in the design and development phases. Nevertheless, the team did not allow these occurrences to disrupt them from achieving the required tasks and goals on schedule.

As mentioned in section 6, a variety of software was utilized at all stages of the project for everything from designing use case diagrams to building the application itself. Each piece of software used by the team during the project is detailed below. For the most part, the original plan and V-model methodology was adhered to, and this resulted in a well-organised and strongly structured project, which in turn led to an enhanced product and report.

The group showed strong commitment to throughout the course of the project with multi-perspective reviews and feedback from different members for every piece of work to ensure everyone is up to date with the latest updates. Starting with meetings around roles distribution to design meetings and report building and finalizing development plan and implementation.

Tools Used in Project Execution

Tracking Progress - Jira

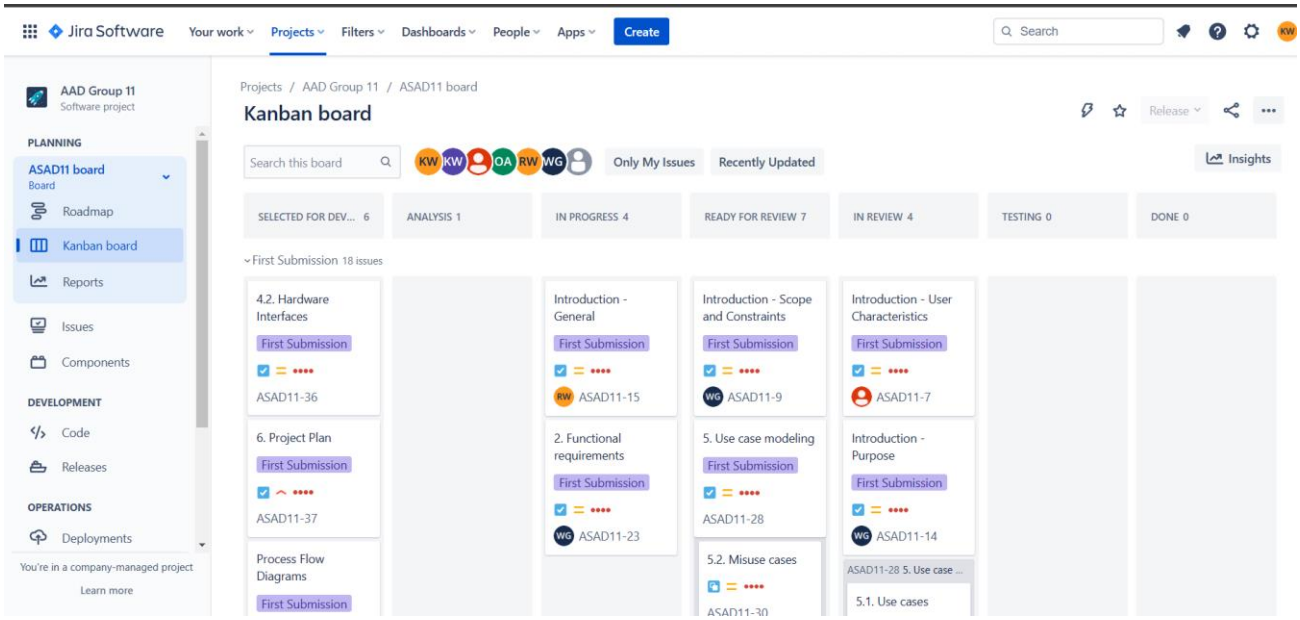


Figure 5454: Jira task tracking

Team Communications - MS Teams

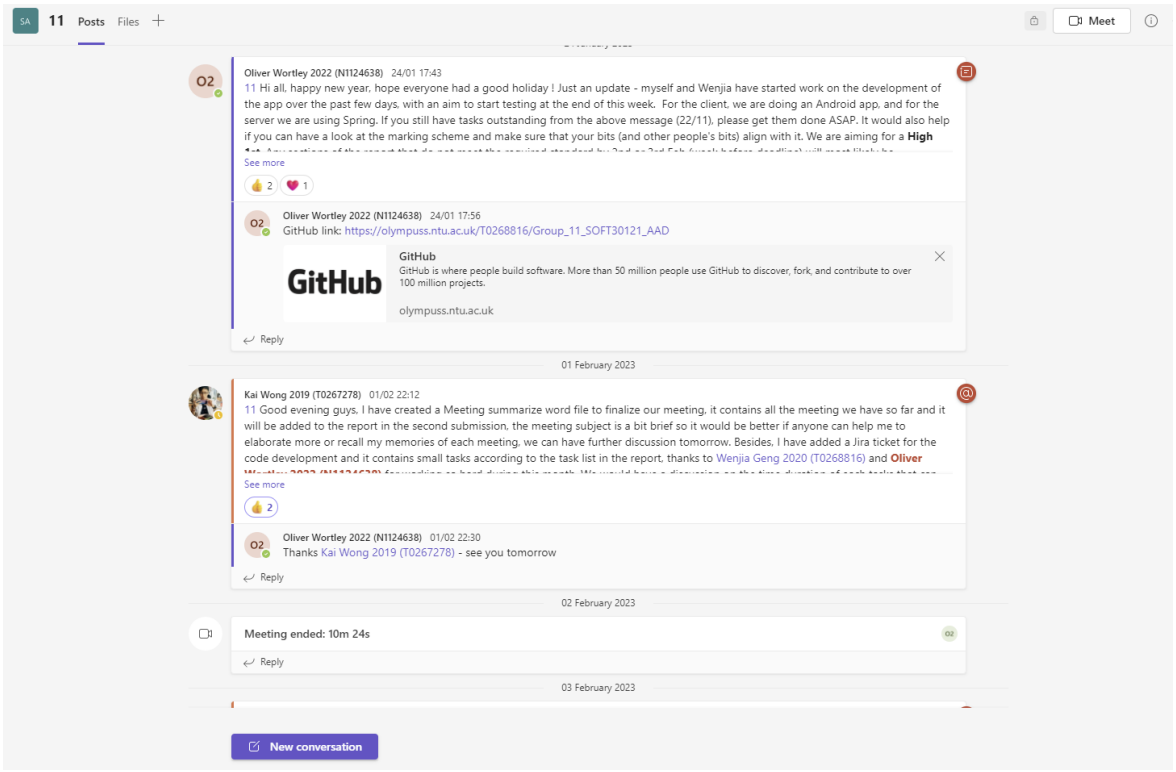


Figure 5555: Microsoft Teams - Group 11 channel

Design and Design Document Sharing - Figma/Lucidchart/MS Teams

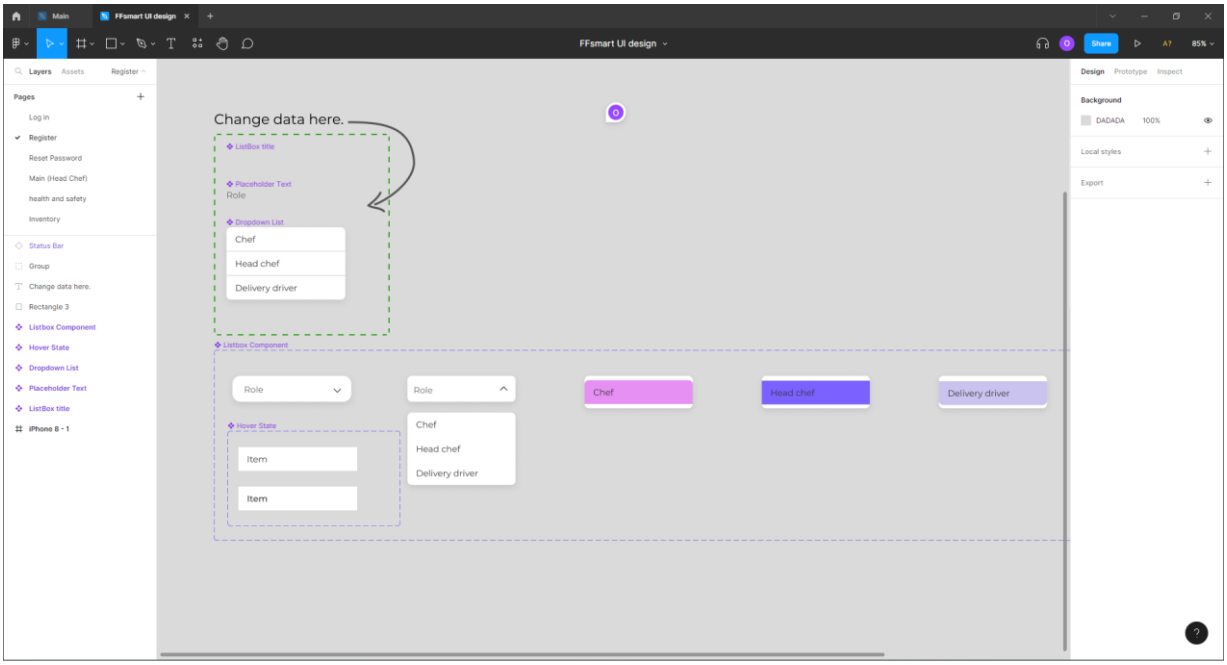


Figure 5656: Figma UI design tool

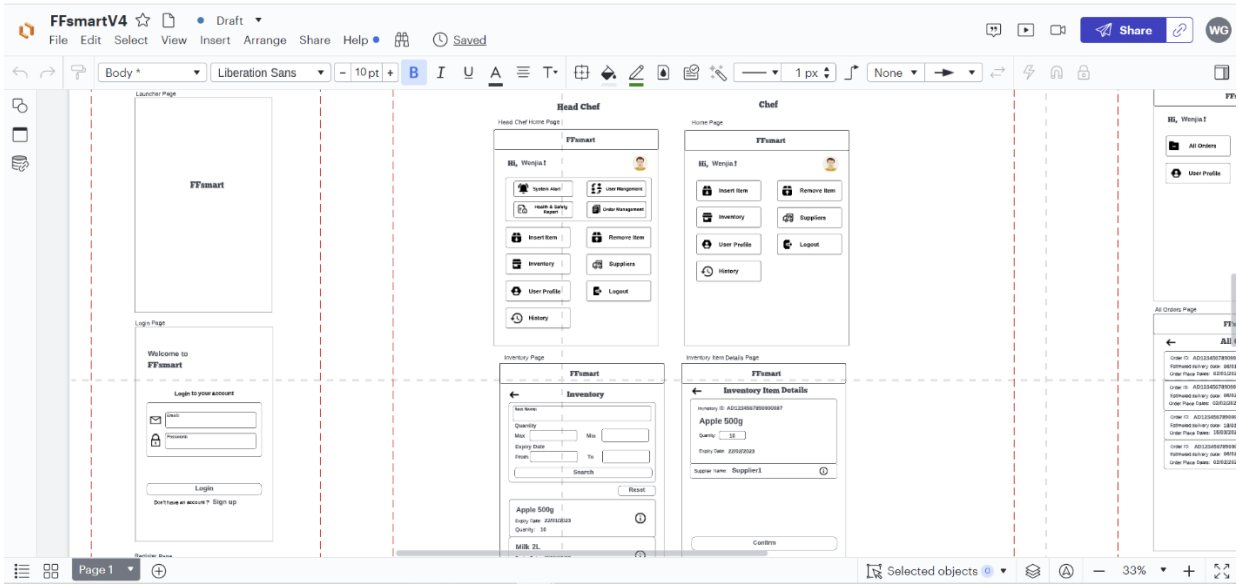


Figure 5757: Android Client design shared file on LucidChart

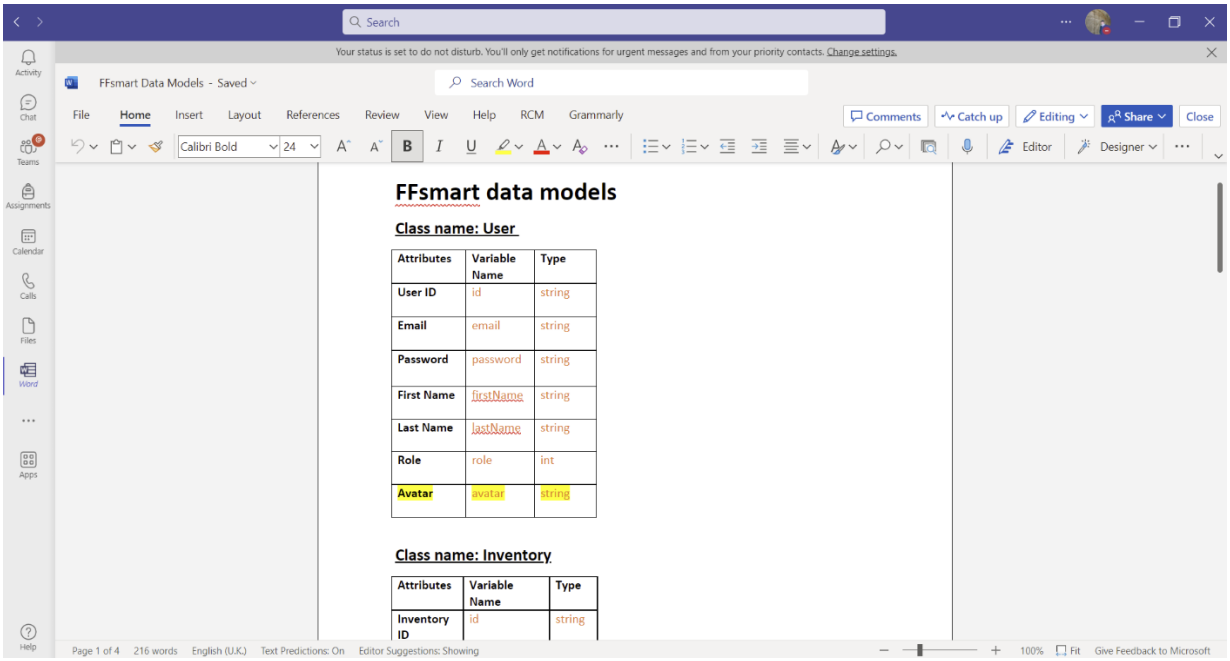


Figure 5858: Data Model shared file on MS Teams

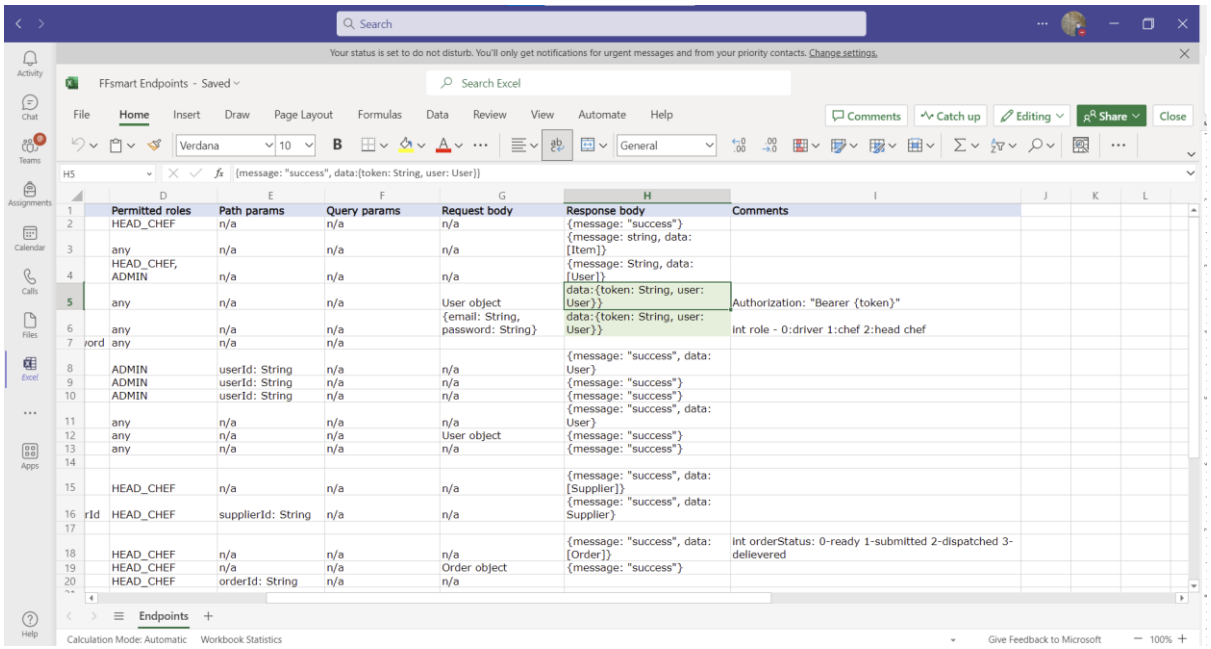


Figure 5959: Client-Server endpoints shared file on MS Teams

Source Control - Git/GitHub

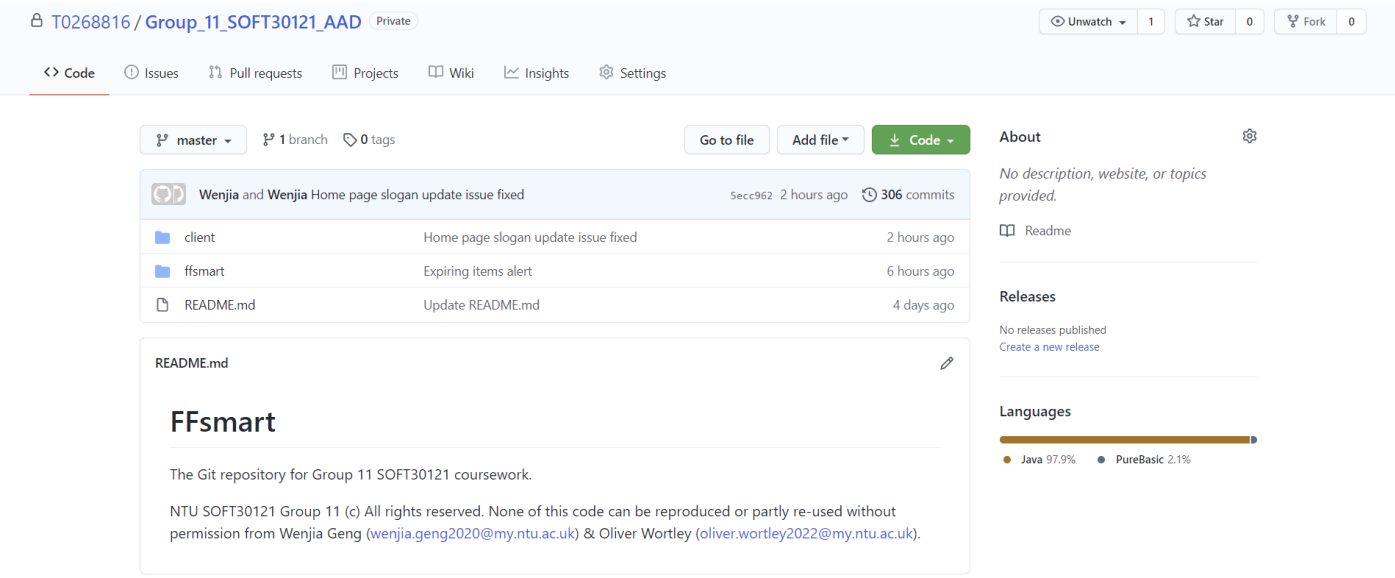


Figure 6060: GitHub repository for FFsmart

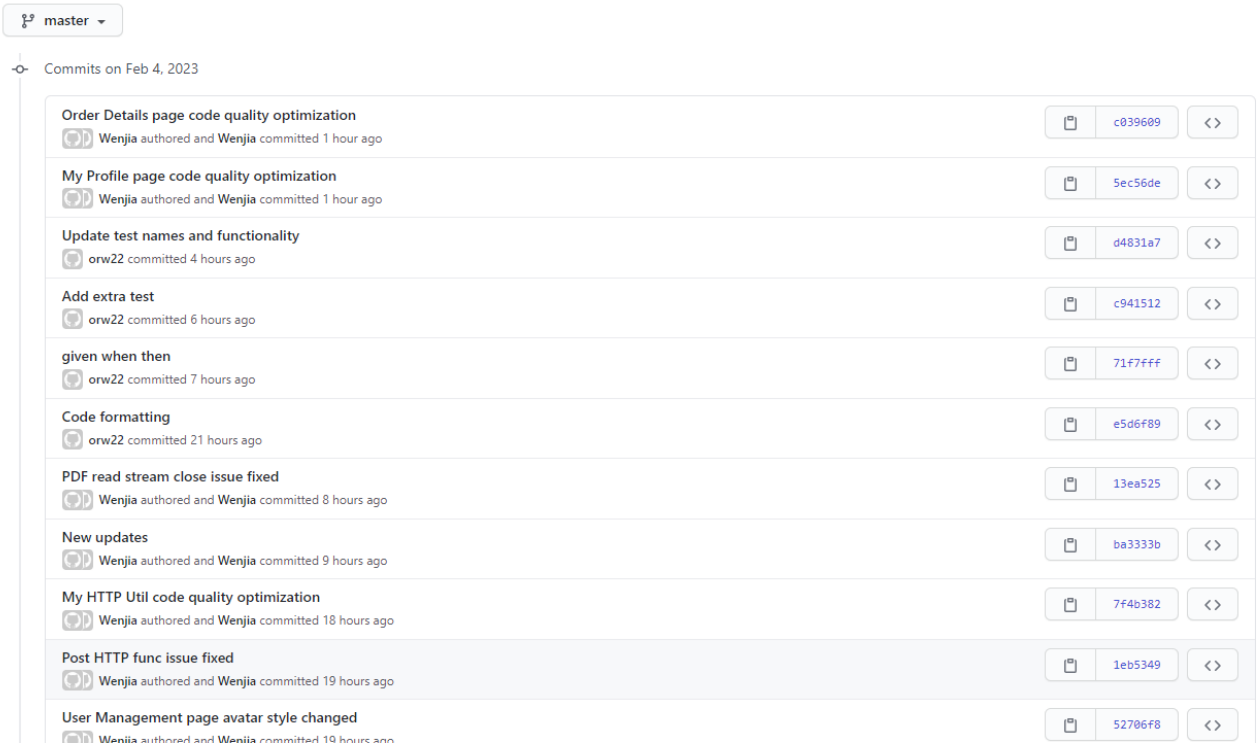


Figure 6161: GitHub commit history

IDE - IntelliJ/Android Studio

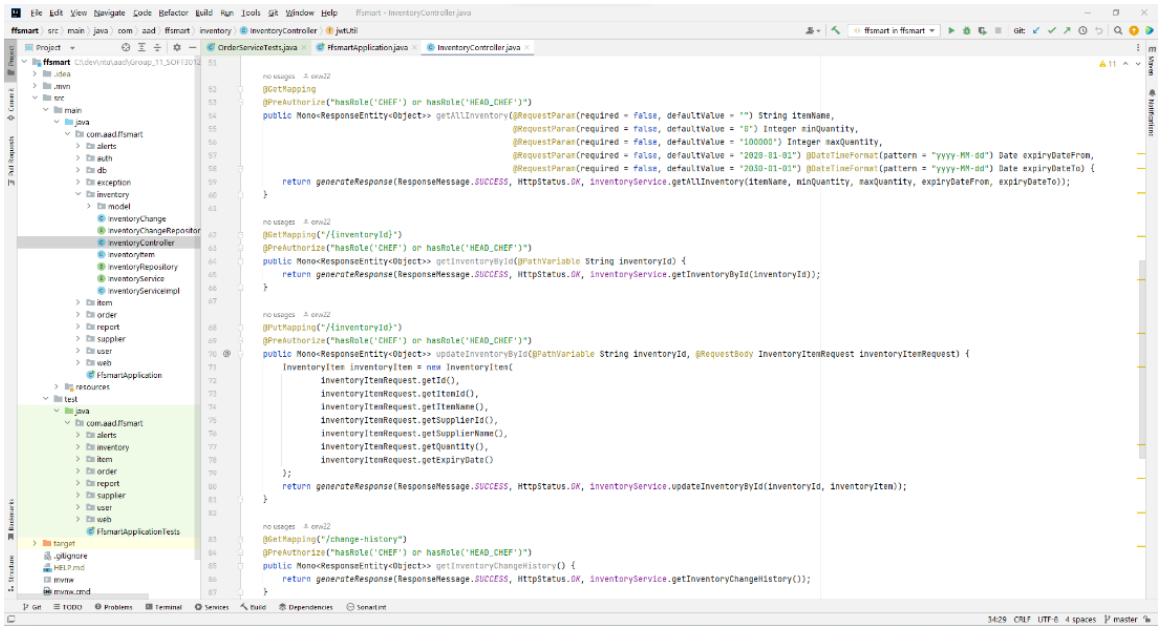


Figure 6262: IntelliJ IDEA

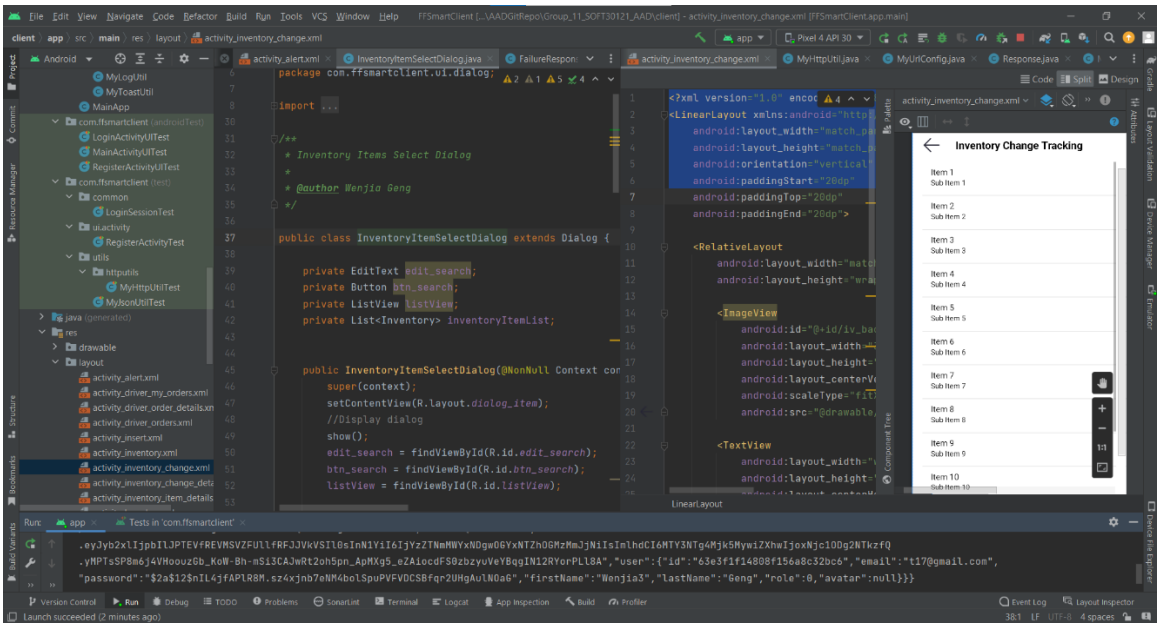


Figure 6363: Android Studio

Testing Frameworks - JUnit/Mockito/PowerMock/Espresso

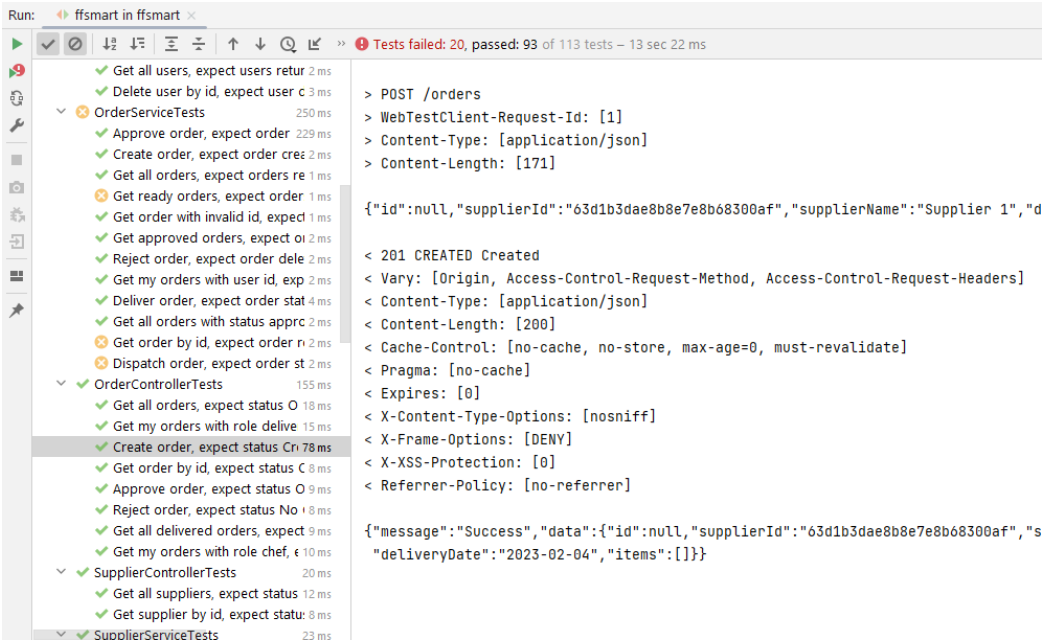


Figure 6464: IntelliJ test run results

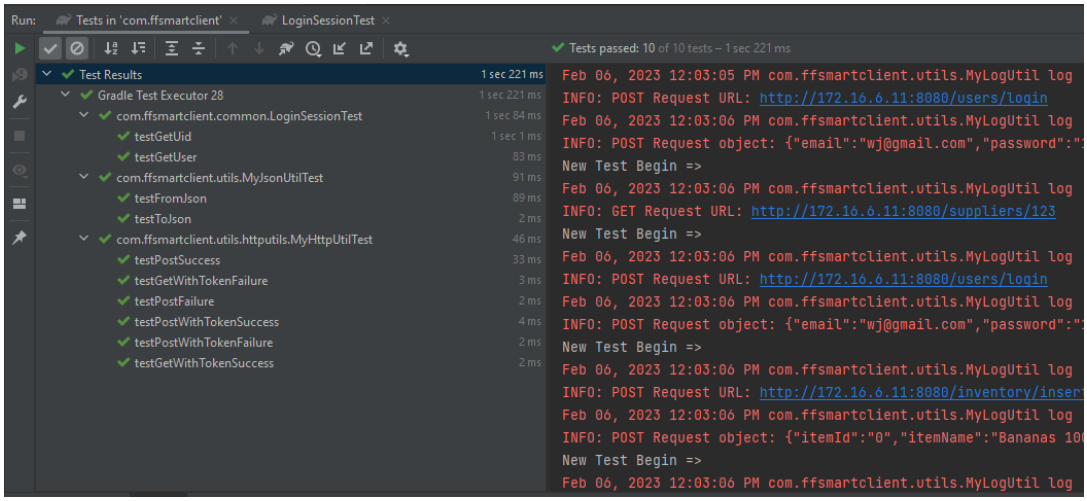


Figure 6565: Android Studio test run results

```java
@WebFluxTest(AlertController.class)
@Import(WebFluxTestSecurityConfig.class)
class AlertControllerTests {
    1 usage
    @Autowired
    WebTestClient webTestClient;

    2 usages
    @MockBean
    private AlertService alertService;

    no usages
    @MockBean
    private GlobalErrorAttributes globalErrorAttributes;

    no usages  orw22
    @DisplayName("Get alerts, expect status Ok")
    @WithMockUser(roles = "HEAD_CHEF")
    @Test
    void givenAlerts_whenGetAlerts_thenStatusOk() {
        Alert alert = new Alert(AlertCode.ORDER_READY, title: "Order ready", message: "Order is ready for approval", new Date());
        when(alertService.getAlerts()).thenReturn(Flux.just(alert));

        webTestClient.get() RequestHeadersUriSpec<capture of ?>
                .uri( uri: "/alerts") capture of ?
                .exchange() ResponseSpec
                .expectStatus().isOk()
                .expectBody() BodyContentSpec
                .jsonPath( expression: "$.data[0].alertCode").isEqualTo(AlertCode.ORDER_READY.value)
                .consumeWith(System.out::println);
        verify(alertService, times( wantedNumberOfInvocations: 1)).getAlerts();
    }

}
```

Figure 6666: Spring JUnit/Mockito unit test example

```java
import org.junit.FixMethodOrder;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.MethodSorters;

@RunWith(AndroidJUnit4.class)
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class MainActivityUITest {

    @Rule
    public ActivityTestRule<MainActivity> rule=new ActivityTestRule<MainActivity>(MainActivity.class, initialT

    @Test
    public void test01whenClickSystemAlertBtn_thenGoAlert() throws InterruptedException {
        Intents.init();
        onView(withId(R.id.btn_alert)).perform(click());
        Thread.sleep( millis: 2000);
        intended(hasComponent(AlertActivity.class.getName()));
        onView(withId(R.id.iv_back)).perform(click());
        Intents.release();
    }

    @Test
    public void test02whenClickUserManagementBtn_thenGoUserManagement() throws InterruptedException {
        Intents.init();
        onView(withId(R.id.btn_userManagement)).perform(click());
        Thread.sleep( millis: 2000);
        intended(hasComponent(UserAccountActivity.class.getName()));
        onView(withId(R.id.iv_back)).perform(click());
        Intents.release();
    }
```

Figure 6767: AndroidJUnit & Espresso UI test examples

Drawing Diagrams - Lucidchart

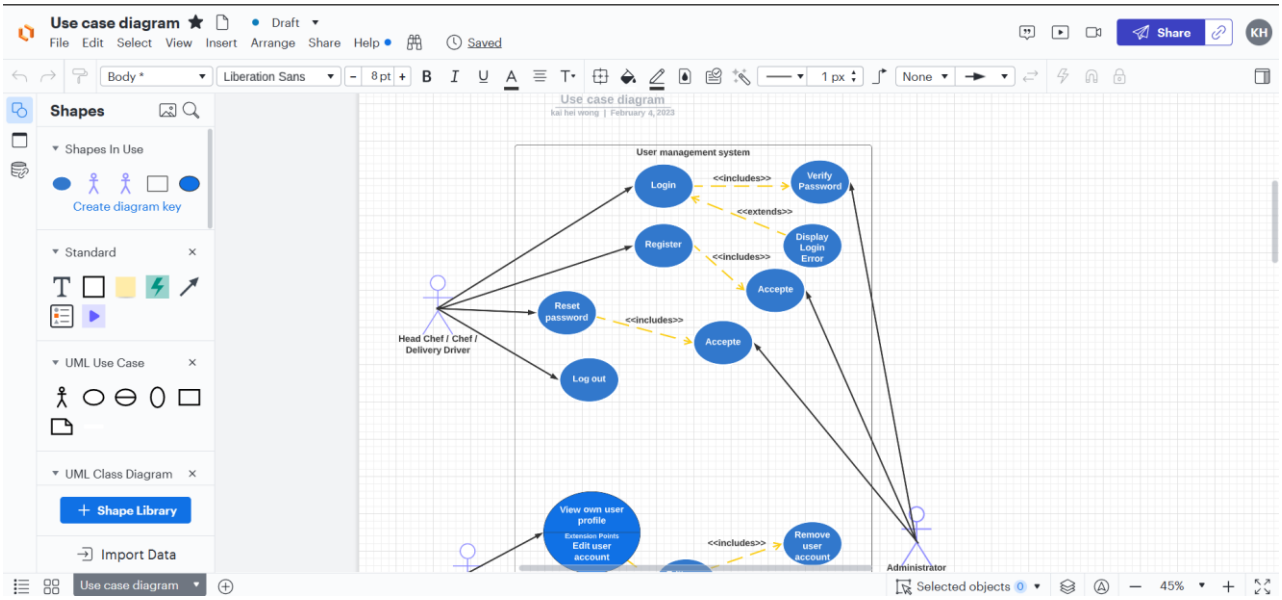


Figure 6868: Lucidchart use case diagram editing

11. References

[1] ujjwal, r. (2018) Software Engineering | Project size estimation techniques - GeeksforGeeks. Available at: https://www.geeksforgeeks.org/software-engineering-project-size-estimation-techniques/ (Accessed: 8 November 2022).

[2] Function Point Languages Table (2009). Available at: https://www.qsm.com/resources/function-point-languages-table (Accessed: 8 November 2022).

[3] kalopsia (2018) Software Engineering | COCOMO Model - GeeksforGeeks. Available at: https://www.geeksforgeeks.org/software-engineering-cocomo-model/ (Accessed: 8 November 2022).

[4] SDLC - V-Model - Tutorialspoint. (n.d.). Www.tutorialspoint.com. https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm#:~:text=The%20V%2Dmodel%20is%20an

[5]Non-functional requirement . https://www.altexsoft.com/blog/non-functional-requirements/ (Accessed: 10 November 2022)

[6] Schwaber, K., & Sutherland, J. (2017). Scrum Guide | Scrum Guides. Scrumguides.org. https://scrumguides.org/scrum-guide.html

[7] MVC figure. https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/ (Accessed: 5 February 2023)

[8]Minhas, S. (2021) 8 rules of Mobile Design, Medium. UX Collective. Available at: https://uxdesign.cc/8-rules-of-mobile-design-1b8d9936c241 (Accessed: February 2, 2023).

[9]Design guidelines, conversational, google developers (2021) Google. Available at: https://developers.google.com/assistant/interactivecanvas/design (Accessed: February 5, 2023).

[10] Coblis - color blindness simulator (2021) Colblindor. Available at: https://www.color-blindness.com/coblis-color-blindness-simulator/ (Accessed: February 8, 2023).

Appendix A: Contributions

Contributions Table

Name	Primary Role(s)	Areas worked on	Contribution rating (0-4)
Kai Hei Wong	Project Manager	**Report:** Introduction - User Characteristics, Non-Functional Requirements, Use Case Modelling, Project Plan, Project execution and management, Meeting records. **Other:** Meeting records, Jira tickets	4
Omar Abdin	Project Manager/ UI Designer	**Report:** Introduction - Overview, Non-Functional Requirements - Performance, Use Cases, User Interfaces, Design Documentation, Coding standards check, Design standards check, Project execution and management. **Other:** Interview Questions, team meetings.	4
Oliver Wortley	Software Engineer / Quality Assurer	**Report:** Section 1: Scope and Constraints Section 2 and 3: (all) Section 4: User Interfaces, Software Interfaces, Interview Questions, Section 5: Written use and misuse cases, Section 6: Risk assessment and task estimation, Tool Analysis Section 7: Spring WebFlux, Coding standards, API error handling, External modules Section 8: User guide review and feedback Section 9: API unit and integration testing, API performance testing with JMeter, Test summary Section 10: (all) **Code/implementation:** Server/API development (Spring RESTful API)	4

		Other: Full review of report and UI, Interview questions	
Mouna Nadine Mokkedem	Systems Architect / UI Designer	**Report:** Non-Functional Requirements– Usability, overview, User Interface, Use Cases, Design documentation	4
Regan Wills	Test Engineer / Quality Assurer	**Report:** Section 1: Purpose, User Characteristics, Assumptions, Overview Section 2: Functional Requirements Section 3: Non-Functional Requirements Section 8: User Help Guide (Installation) Section 9: System Tests, Acceptance Tests, Test Summary **And:** Review/Comments/Verbal Feedback/Editing Throughout **Other:** Interview Questions, Delivering Interview, Document Setup, Formatting, Dialogue with Dev Team	4
Wenjia Geng	Software Engineer / Quality Assurer	**Report:** Section 1: Purpose Section 2: Functional Requirements Section 4: User Interfaces Section 6: Methodology, Risk assessment, Gantt chart, Task List, Critical Path, Task Estimation Section 7: Frontend Error Handling, Code Standards (SonarLint), External Modules Section 8: User Guide Section 9: Unit Tests (Client), Integration Tests (Client), Android UI Tests **Code/implementation:** Android Client Development	4

		Other: Interview questions, GitHub repository management	
Peter Gama	Systems Architect	**Report:** Non-Functional Requirements, Assumptions, Use Cases.	2

Signed by:

Name	**Signed**
Kai Hei Wong	Kai Hei Wong
Oliver Wortley	Oliver R Wortley
Mouna Nadine Mokkedem	Mouna Nadine Mokkkedem
Omar Abdin	Omar Ashraf Abdin
Peter Gama	P.s Gama.
Wenjia Geng	Wenjia Geng
Regan Wills	Regan Wills

Appendix B: Functional-Point Analysis and COCOMO Cost Estimation

Functional-Point Analysis

The following is a Functional-Point Analysis of our proposed solution, utilising the calculations and guidance provided in "*Project Size Estimation Techniques"* (ujjwal, 2018).

The first step is to count the number of functions of each of the below types:

- Transactional:
  - External inputs
  - External outputs
  - External inquiries
- Data:
  - Internal files
  - External interface files

(Due to the changing nature of software projects, we will update the counts iteratively as the design and development work progresses. The below counts are initial estimates based on our current interface designs.)

Function type	**Count (Estimated)**
External inputs	5
External outputs	2
External inquiries	1
Internal files	1
External interface files	4

Next, we compute the Unadjusted Function Points (UFP), by categorising each of the above function types by their complexity (simple, average, complex), and then multiplying the count of each function type with its respective weighting factor. Complexities:

Function type	**Complexity**
External inputs	Simple
External outputs	Simple
External inquiries	Simple
Internal files	Average
External interface files	Simple

The weighting factors we are using are shown below, taken from the source mentioned above (ujjwal, 2018):

Function type	**Simple**	**Average**	**Complex**
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
Internal files	7	10	15
External interface files	5	7	10

So, using the above weightings, our function counts and complexity categorisation, we calculate UFP:

UFP = (3 * 5) + (4 * 2) + (3 * 1) + (7 * 1) + (5 * 4) = 53

After this, we find the Total Degree of Influence (TDI) of our system, using the 14 general system characteristics and rating the degree of influence of our system on a scale of 0 to 5 for each of them, and then adding them together:

General System Characteristic	**Degree of Influence**	**General System Characteristic**	**Degree of Influence**
Data communications	3	Online update	3
Distributed data processing	2	Complex processing	1
Performance	2	Reusability	4
Heavily used configuration	2	Installation ease	2
Transaction rate	1	Operational ease	4
On-line data entry	4	Multiple sites	0
End-user efficiency	4	Facilitate change	3

TDI = 35

From this, we can determine the Value Adjustment Factor (VAF), which is used to adjust the UFP value and obtain our function points (FP):

VAF = 0.65 + (0.01 * 35) = 1.00

FP = UFP * VAF = 53 * 1.00 = 53.00

Conversion from FP to LOC

Now that we have our function points, we can convert this to a LOC value for our COCOMO effort estimation using average LOC/FP ratios from similar past projects. We will use the average LOC/FP ratios provided by QSM (2009).

Since we are using Java/C++ for the development of the application, LOC/FP = 53. Therefore:

LOC = 53 * 53.00 = 2809

COCOMO Cost Estimation

For the below calculations, I will use the COCOMO (Constructive Cost Model) equations and information provided in "*Software Engineering | COCOMO Model*" (kalopsia, 2018).

Given our LOC value, we will now use the COCOMO Basic Model to estimate the total effort required to complete the project.

COCOMO defines 3 types of projects: Organic, Semi-detached, and Embedded. Because our problem has been solved before and we have a small team with nominal experience, we will define our project as **Organic**.

We will use the following equations:

Effort = a(KLOC)^b
Time = c(Effort)^d
Persons required = Effort/time

- *where KLOC is thousands of lines of code, Effort is in person months and Time is in months*

These are our values for a, b, c and d:

Project Type	a	b	c	d
Organic	2.4	1.05	2.5	0.38

Effort = 2.4 * $(2.809)^{1.05}$ = 7.099
Time = 2.5 * $(7.099)^{0.38}$ = 5.265
Persons required (for 5 months development time) = 7.099 / 5.265 = 1.348

Given our development time is more like 2 months, we will use 2 as our Time value and recalculate persons required:

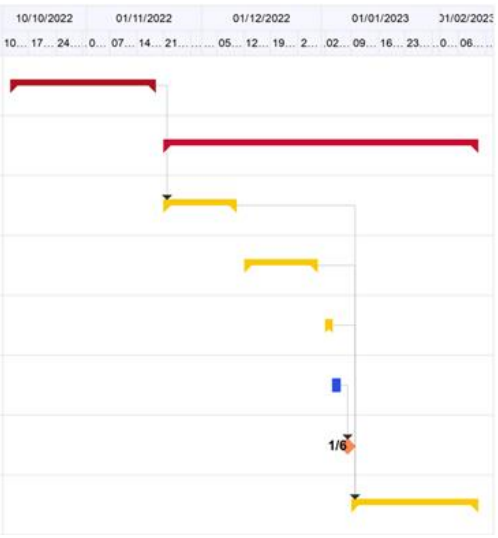
Persons required = 7.099 / 2 = 3.55 persons

The size of our team (7 people) far exceeds the persons required to complete the development work of the project, so this sets us in good stead for the project work.

Appendix C: Gantt Chart

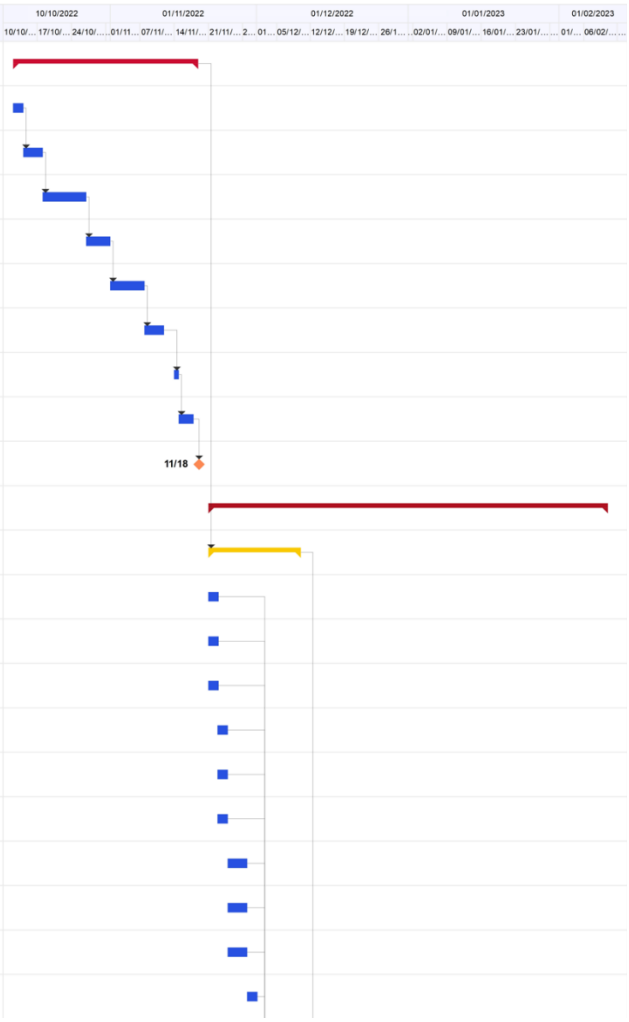
High-Level

ID	Task Name	Start Date	Finish Date	Duration	Complete
1	1. First Submission	12/10/2022 09:00	18/11/2022 22:00	28.0 d.	0.0%
11	2. Second Submission	21/11/2022 08:00	10/02/2023 22:00	60.0 d.	0.0%
12	2-1.Development	21/11/2022 08:00	09/12/2022 22:00	15.0 d.	0.0%
30	2-2. Testing	12/12/2022 08:00	30/12/2022 22:00	15.0 d.	0.0%
49	2-3. Create user manual	02/01/2023 08:00	03/01/2023 22:00	2.0 d.	0.0%
52	2-4.Review and complete all final deliverables of the project	04/01/2023 08:00	05/01/2023 22:00	2.0 d.	0.0%
53	milestone4 : Delivery milestone	06/01/2023 08:00	06/01/2023 22:00	1.0 d.	0.0%
54	2-5. Create Final Report	09/01/2023 08:00	10/02/2023 22:00	25.0 d.	0.0%

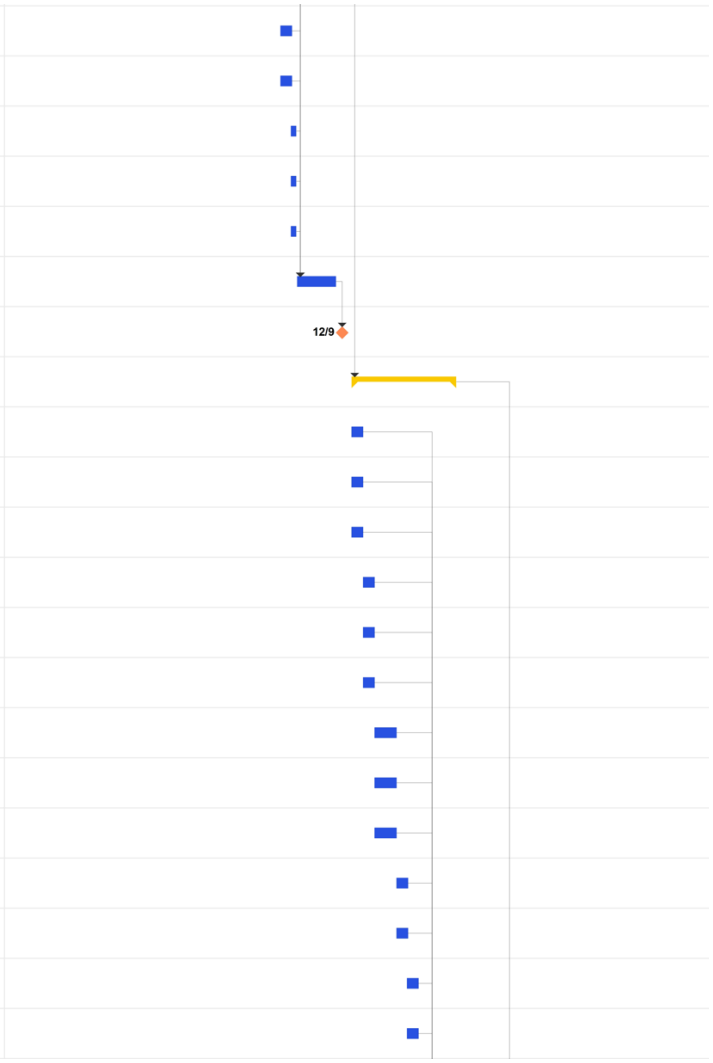


Detailed

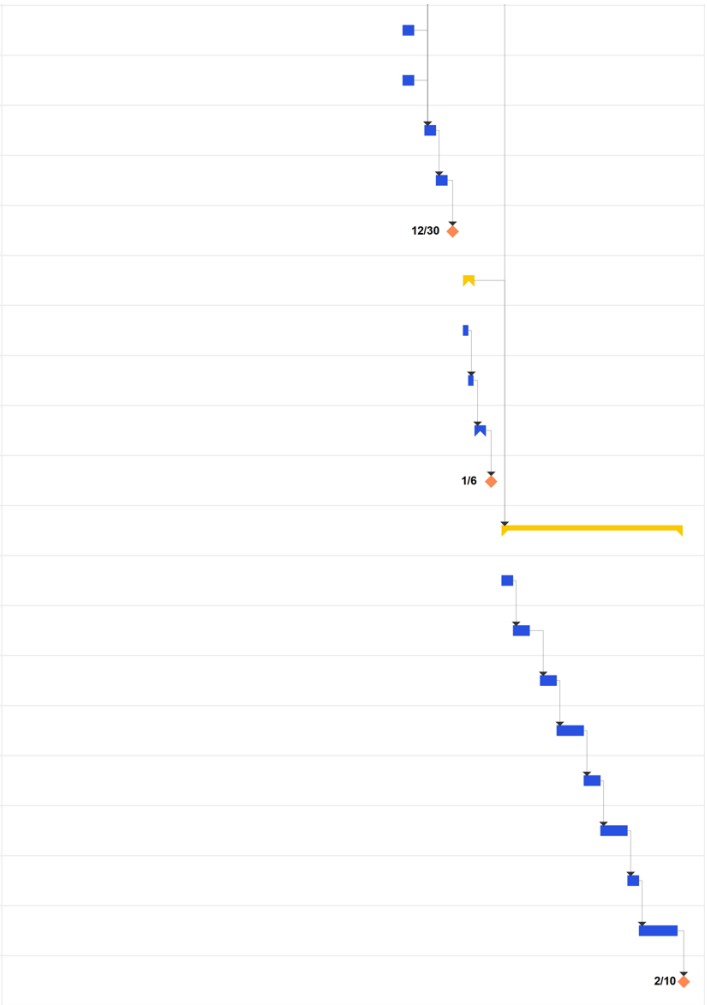
ID	Task Name	Start Date	Finish Date	Duration	Complete
1	1. First Submission	12/10/2022 09:00	18/11/2022 22:00	28.0 d.	0.0%
2	1-1Introduction	12/10/2022 09:00	13/10/2022 23:00	2.0 d.	0.0%
3	1-2Functional Require	14/10/2022 09:00	17/10/2022 23:00	2.0 d.	0.0%
4	1-3Non-Functional Requirements	18/10/2022 09:00	26/10/2022 23:00	7.0 d.	0.0%
5	1-4Interfaces	27/10/2022 09:00	31/10/2022 22:00	3.0 d.	0.0%
6	1-5Use Case Modelling	01/11/2022 08:00	07/11/2022 22:00	5.0 d.	0.0%
7	1-6Project Plan	08/11/2022 08:00	11/11/2022 22:00	4.0 d.	0.0%
8	1-7References	14/11/2022 08:00	14/11/2022 22:00	1.0 d.	0.0%
9	1-8Consolidation and arrangement	15/11/2022 08:00	17/11/2022 22:00	3.0 d.	0.0%
10	Milestone1:Planning Milestone	18/11/2022 08:00	18/11/2022 22:00	1.0 d.	0.0%
11	2. Second Submission	21/11/2022 08:00	10/02/2023 22:00	60.0 d.	0.0%
12	2-1.Development	21/11/2022 08:00	09/12/2022 22:00	15.0 d.	0.0%
13	T1:Create Register page	21/11/2022 08:00	22/11/2022 22:00	2.0 d.	0.0%
14	T2:Create User management page	21/11/2022 08:00	22/11/2022 22:00	2.0 d.	0.0%
15	T3:Create Login page	21/11/2022 08:00	22/11/2022 22:00	2.0 d.	0.0%
16	T4:Create Forgot password page	23/11/2022 08:00	24/11/2022 22:00	2.0 d.	0.0%
17	T5:Create User management page	23/11/2022 08:00	24/11/2022 22:00	2.0 d.	0.0%
18	T6:Create User management page	23/11/2022 08:00	24/11/2022 22:00	2.0 d.	0.0%
19	T7:Create Main page	25/11/2022 08:00	28/11/2022 22:00	2.0 d.	0.0%
20	T8:Create Orders page	25/11/2022 08:00	28/11/2022 22:00	2.0 d.	0.0%
21	T9:Create Record expired items Function	25/11/2022 08:00	28/11/2022 22:00	2.0 d.	0.0%
22	T10:Create Review auto-reorder list	29/11/2022 08:00	30/11/2022 22:00	2.0 d.	0.0%



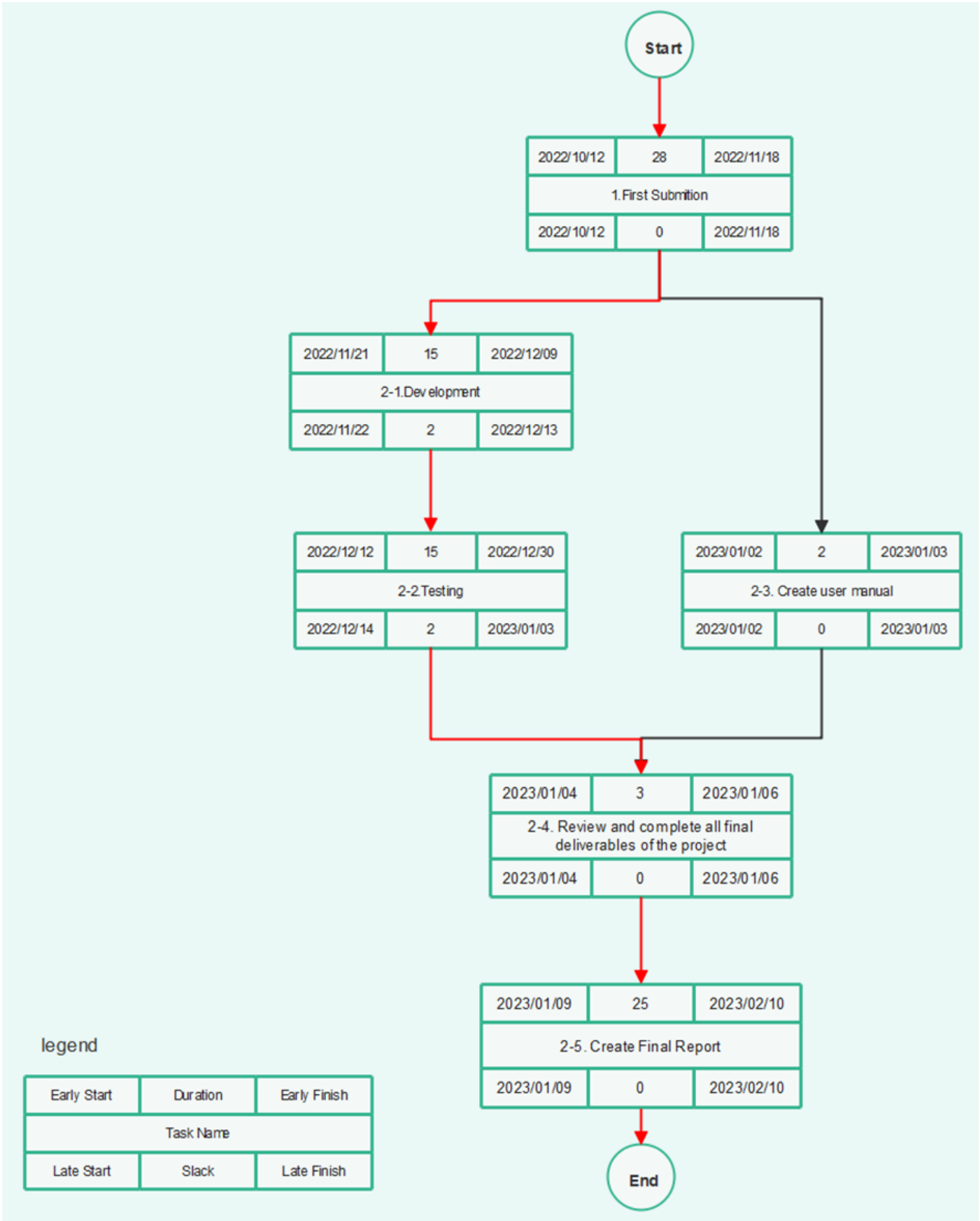
23	T11:Create Insert items page	29/11/2022 08:00	30/11/2022 22:00	2.0 d.	0.0%	
24	T12:Create Remove items page	29/11/2022 08:00	30/11/2022 22:00	2.0 d.	0.0%	
25	T13:Create Inventory page,	01/12/2022 08:00	01/12/2022 22:00	1.0 d.	0.0%	
26	T14:Create Low quantity items page	01/12/2022 08:00	01/12/2022 22:00	1.0 d.	0.0%	
27	T15:Create Send health and safety report Function	01/12/2022 08:00	01/12/2022 22:00	1.0 d.	0.0%	
28	Review coding	02/12/2022 08:00	08/12/2022 22:00	5.0 d.	0.0%	
29	Milestone2:Development milestone	09/12/2022 08:00	09/12/2022 22:00	1.0 d.	0.0%	
30	**2-2. Testing**	12/12/2022 08:00	30/12/2022 22:00	15.0 d.	0.0%	
31	T1:Test Register page	12/12/2022 08:00	13/12/2022 22:00	2.0 d.	0.0%	
32	T2:Test User management page	12/12/2022 08:00	13/12/2022 22:00	2.0 d.	0.0%	
33	T3:Test Login page	12/12/2022 08:00	13/12/2022 22:00	2.0 d.	0.0%	
34	T4:Test Forgot password page	14/12/2022 08:00	15/12/2022 22:00	2.0 d.	0.0%	
35	T5:Test User management page	14/12/2022 08:00	15/12/2022 22:00	2.0 d.	0.0%	
36	T6:Test User management page	14/12/2022 08:00	15/12/2022 22:00	2.0 d.	0.0%	
37	T7:Test Main page	16/12/2022 08:00	19/12/2022 22:00	2.0 d.	0.0%	
38	T8:Test Orders page	16/12/2022 08:00	19/12/2022 22:00	2.0 d.	0.0%	
39	T9:Test Record expired items Function	16/12/2022 08:00	19/12/2022 22:00	2.0 d.	0.0%	
40	T10:Test Review auto-reorder list	20/12/2022 08:00	21/12/2022 22:00	2.0 d.	0.0%	
41	T11:Test Insert items page	20/12/2022 08:00	21/12/2022 22:00	2.0 d.	0.0%	
42	T12:Test Remove items page	22/12/2022 08:00	23/12/2022 22:00	2.0 d.	0.0%	
43	T13:Test Inventory page,	22/12/2022 08:00	23/12/2022 22:00	2.0 d.	0.0%	



44	T14:Test Low quantity items page	22/12/2022 08:00	23/12/2022 22:00	2.0 d.	0.0%	
45	T15:Test Send health and safety report Function	22/12/2022 08:00	23/12/2022 22:00	2.0 d.	0.0%	
46	Intergration Test	26/12/2022 08:00	27/12/2022 22:00	2.0 d.	0.0%	
47	Review the Result of Test	28/12/2022 08:00	29/12/2022 22:00	2.0 d.	0.0%	
48	Milestone3:Testing milestone	30/12/2022 08:00	30/12/2022 22:00	1.0 d.	0.0%	
49	**2-3. Create user manual**	02/01/2023 08:00	03/01/2023 22:00	2.0 d.	0.0%	
50	Write user manual	02/01/2023 08:00	02/01/2023 22:00	1.0 d.	0.0%	
51	review user manual	03/01/2023 08:00	03/01/2023 22:00	1.0 d.	0.0%	
52	**2-4.Review and complete all final deliverables of the project**	04/01/2023 08:00	05/01/2023 22:00	2.0 d.	0.0%	
53	milestone4 : Delivery milestone	06/01/2023 08:00	06/01/2023 22:00	1.0 d.	0.0%	
54	**2-5. Create Final Report**	09/01/2023 08:00	10/02/2023 22:00	25.0 d.	0.0%	
55	1) Architecture	09/01/2023 08:00	10/01/2023 22:00	2.0 d.	0.0%	
56	2) Physical Deployment	11/01/2023 08:00	13/01/2023 22:00	3.0 d.	0.0%	
57	3) Behaviour Diagrams	16/01/2023 08:00	18/01/2023 22:00	3.0 d.	0.0%	
58	4) AStructure Diagrams	19/01/2023 08:00	23/01/2023 22:00	3.0 d.	0.0%	
59	5) Testing and Results	24/01/2023 08:00	26/01/2023 22:00	3.0 d.	0.0%	
60	6) Conclusion	27/01/2023 08:00	31/01/2023 22:00	3.0 d.	0.0%	
61	7) Consolidation and arrangement	01/02/2023 08:00	02/02/2023 22:00	2.0 d.	0.0%	
62	Review Final Report	03/02/2023 08:00	09/02/2023 22:00	5.0 d.	0.0%	
63	Milestone5:Final milestone	10/02/2023 08:00	10/02/2023 22:00	1.0 d.	0.0%	



Appendix D: Critical Path


Start
2022/10/12
28
2022/11/18
1.First Submition
2022/10/12
0
2022/11/18
2022/11/21
15
2022/12/09
2-1.Development
2022/11/22
2
2022/12/13
2022/12/12
15
2022/12/30
2-2.Testing
2022/12/14
2
2023/01/03
2023/01/02
2
2023/01/03
2-3. Create user manual
2023/01/02
0
2023/01/03
2023/01/04
3
2023/01/06
2-4. Review and complete all final deliverables of the project
2023/01/04
0
2023/01/06
2023/01/09
25
2023/02/10
2-5. Create Final Report
2023/01/09
0
2023/02/10
End
legend
Early Start
Duration
Early Finish
Task Name
Late Start
Slack
Late Finish

Appendix E: Meeting Records

Group 11 1st Meeting

Date: 10/04/2022 9am-11am

Location: ISTEC 029

Attendees: Kai Hei Wong, Wenjia Geng, Peter Gama, Oliver Wortley, Regan Wills

Duration: 120 minutes

- **Continued discussion of requirements gathering and documentation of questions to ask the stakeholder**
- **Determined every Thursday from 3 pm to 4 pm as the weekly meeting time**
- **Set up Jira software for group management tools and assign all members to the Jira team.**
- **Confirmed Jira as a group development tool and registered accounts for members in attendance**
- **Discussed the database and UI development to be used.**

Actions

Who	**What**	**Deadline**
All	**Add questions to the development file on Teams. Think about architecture and design.**	6/10/2022
Kai Hei Wong	Building meeting table and manage meeting	6/10/2022
Wenjia Geng	Setting up a GitHub repository and learning React library for the front-end development.	6/10/2022
Peter Gama	Learning the UML tools for architecture design.	6/10/2022
Oliver Wortley	Learning C++ for application development.	6/10/2022
Regan Wills	Familiar with testing techniques that may be used in future testing.	6/10/2022
Omar Abdin	Consider the roles that will be taken in the development project.	6/10/2022
Mouna Nadine	Consider the roles that will be taken in the development project.	6/10/2022

Other notes:

Omar and Mouna are absent,

Please sign up for the account in Jira software by using your T-number email, please contact Oliver if you have any issues with using Jira.

Group 11 2nd Meeting
Date: 10/06/2022 3pm-4pm
Location: ABK 210
Attendees: Kai Hei Wong, Wenjia Geng, Peter Gama, Oliver Wortley, Regan Wills, Omar Abdin, Mouna Nadine
Duration: 60 minutes

- **Discussion of functional and non-requirements by gathering questions to ask the stakeholder**

- **Discussion of the usage of Jira software**
- **Assign member roles.**
- **Assign the introduction of the group report to every member**

Actions

Who	**What**	**Deadline**
All	**Add questions to the development file on Teams.** **Collect and have further discussion on project ideas.** Every member will be involved to finish the introduction in the group report	13/10/2022
Kai Hei Wong	Building meeting table and manage meeting Use Jira software to create tickets for tasks Finish the **User characteristics** in Introduction	13/10/2022
Wenjia Geng	Building a list of each member's skills and techniques. Finish the **purpose** in Introduction Functional requirement Non-functional requirement	13/10/2022
Peter Gama	Finish the **assumptions** in Introduction	13/10/2022
Oliver Wortley	Finish the **scope and constraints** in Introduction	13/10/2022
Regan Wills	Finish the **purpose** in Introduction	13/10/2022
Omar Abdin	Finish the **overview** in Introduction User interface, Use case, misuse case	13/10/2022
Mouna Nadine	Finish the **overview** in Introduction User interface	13/10/2022

Other notes:
Awaiting confirmation of the meeting time with the stockholders
Question > requirement

Group 11 3nd Meeting

Date: 11/10/2022 9am-11am

Location: ISC 029

Attendees: Kai Hei Wong, Wenjia Geng, Peter Gama, Oliver Wortley, Regan Wills, Omar Abdin, Mouna Nadine

Duration: 120 minutes

- **Discussion of functional and non-requirements questions to ask the stakeholders.**

Actions

Who	**What**	**Deadline**
All	**Collect and have further discussion on project ideas.** Every member will be involved to finish the introduction in the group report	13/10/2022
Kai Hei Wong	Building meeting table and manage meeting Use Jira software to create tickets for tasks Finish the **User characteristics** in Introduction Use case, misuse case	13/10/2022
Wenjia Geng	Building a list of each member's skills and techniques. Finish the **purpose** in Introduction Functional requirement Non-functional requirement	13/10/2022
Peter Gama	Finish the **assumption** in Introduction	13/10/2022
Oliver Wortley	Finish the **scope and constraints** in Introduction	13/10/2022
Regan Wills	Finish the **purpose** in Introduction	13/10/2022
Omar Abdin	Finish the **overview** in Introduction User interface, Use case, misuse case	13/10/2022
Mouna Nadine	Finish the **overview** in Introduction User interface	13/10/2022

Other notes:

Awaiting confirmation of the meeting time with the stockholders

Question > requirement

Group 11 4th Meeting

Date: 13/10/2022 2pm-3pm

Location: ISC 029

Attendees: Kai Hei Wong, Wenjia Geng, Peter Gama, Oliver Wortley, Regan Wills, Omar Abdin, Mouna Nadine

Duration: 60 minutes

- **Discussion of functional and non-requirements questions to ask the stakeholders.**
- **Ordering questions in relation to their priority**

Actions

Who	**What**	**Deadline**
All	**Add and format questions for upcoming client interview** Every member will also be involved in the Functional and Non-functional requirements	16/10/2022
Kai Hei Wong	Start work on Non-functional requirements Manage meeting Create tickets for tasks in Jira Making notes during the interview with the stakeholders	16/10/2022
Wenjia Geng	Start work on Functional requirements	16/10/2022
Peter Gama	Start work on Non-functional requirements	16/10/2022
Oliver Wortley	Start work on Functional requirements Review report sections further	16/10/2022
Regan Wills	Finish the **review** of the report Reformat questions Put questions in priority order	17/10/2022
Omar Abdin	Add to questions for stakeholder Reformat questions where applicable **S**tart work on the Functional requirements	16/10/2022
Mouna Nadine	Start work on the Non-functional requirement	18/10/2022

Other notes:

Awaiting confirmation of the meeting time with the stockholders.

Group 11 5th Meeting

Date: 14/10/2022 4:00pm-4:30pm

Location: Teams online meeting

Attendees: Kai Hei Wong, Wenjia Geng, Oliver Wortley, Omar Abdin, Mouna Nadine

Duration: 30 minutes

- **Discussion of interview strategy**
- **Reminder of reformatting the questions and ordering by priority.**

Notes

2 strategies for the interview were discussed: 1) closed and specific questions, 2) presentation of early prototype

It was decided unanimously that we would stick to our approach of gathering appropriate, well-written and closed questions and asking them in order of importance. This was decided to be the best use of time and most efficient way of extracting information from the stakeholder.

Requirements were also briefly discussed. Assignments remain the same as the previous meeting, but priority has been given to the stakeholder questions, with requirements to be completed and finalized after the interview.

Actions

See previous meeting notes (13/10/2022)

All team members to think about client questions and add to the questions document (preferably with answer options as well) by Sunday (16/10/2022) so that they can be prioritized

Also, to work on NFRs/FRs as specified in previous meeting

Other notes:

Awaiting confirmation of the meeting time with the stockholders.

Group 11 6th Meeting

Date: 20/10/2022 1:00pm-2:00pm

Location: Teams online meeting

Attendees: Kai Hei Wong, Wenjia Geng, Oliver Wortley, Omar Abdin, Mouna Nadine, Peter Gama

Duration: 60 minutes

- **Review the interview questions**
- **Review the initial functional and non-functional requirements.**
- **Decide everyone to keep working on FRs/NFRs if you have a bit of time and aren't too busy with FYP**
- **Cancel meeting next week due to PPD deadline.**

Notes

First, we reviewed the interview question done by Regan, and let other members check if there is any mistake.

Next, we reviewed the initial work of functional and non-functional requirement done by Wenjia and Marco.

Who	What	Deadline
All	**Add and format questions for upcoming client interview** Every member will also involve in working on the Functional and Non-functional requirements	3/11/2022
Kai Hei Wong	Start work on Performance requirements in Non-functional requirement. Making notes and record the interview with the stakeholders Add subtasks to FR and NFR tickets on Jira and assign to respective team members	3/11/2022
Wenjia Geng	Keep working on Functional requirements	3/11/2022
Peter Gama	Keep working on Reliability requirements in Non-functional requirement.	3/11/2022
Oliver Wortley	Keep working on Functional requirements	3/11/2022
Regan Wills	Keep working on Functional requirements	3/11/2022
Omar Abdin	Keep working on the Functional requirement	3/11/2022
Mouna Nadine	Keep working on the Usability requirement in Non-functional requirement	3/11/2022

Other notes:

Awaiting confirmation of the meeting time with the stockholders.

- Cancelling meeting next week due to PPD deadline, any questions please message/call on WhatsApp or teams
- Marco to add subtasks to FR and NFR tickets on Jira and assign to respective team members
- Everyone to keep working on FRs/NFRs if you have a bit of time and aren't too busy with FYP
- Remember to add additional features to questions doc if you come up with any nice ideas

-

Group 11 7th Meeting

Date: 2/11/2022 2:00pm-3:00pm

Location: Teams online meeting

Attendees: Kai Hei Wong, Wenjia Geng, Oliver Wortley, Omar Abdin, Mouna Nadine, Peter Gama

Duration: 60 minutes

- **Assign everyone to start working on use case modelling and interface part in the report**
- **Finalize the functional requirement and non-functional requirement.**

Notes:

Who	**What**	**Deadline**
All	Every member will also be involved in working on the Interfaces and Use case modelling after finalizing the functional requirement and non-functional requirement.	10/11/2022
Kai Hei Wong	Start working on the use case modelling.	10/11/2022
Wenjia Geng	Keep working on functional requirements	10/11/2022
Peter Gama	Keep working on the reliability requirement in non-functional requirement.	10/11/2022
Oliver Wortley	Keep working on functional requirements	10/11/2022
Regan Wills	Start working on project plan	10/11/2022
Omar Abdin	Start working on the user interface	10/11/2022
Mouna Nadine	Start working on the user interface	10/11/2022