**ADVANCED ANALYSIS AND DESIGN**

**FFSmart Fridge Project**

**Group: First Solutions**



**Author(s):** Jared Awiti (N0992216), Akshay Chudasama (N0999738), Dammy Ademola (N1000282), and Max Maltby (N1002723)

**Contents:**

1. Introduction................................................................................4

1.1. Purpose.........................................................................4

1.2. User Characteristics......................................................4

1.3. Assumptions..................................................................5

1.4. Scope and Constraints ……………………………….....5

1.5. Glossary of terms...........................................................6

1.6. Overview........................................................................6

1.6.1. Tech Stack Overview…………………………..6

1. Functional Requirements............................................................8

2.1. Requirement elicitation…...............................................8

2.1.1. Document analysis..........................................8

2.1.2. Interviews........................................................8

2.2. FFSmart Fridge Functional Requirements.....................9

1. Non- Functional Requirements...................................................12

3.1. Usability Requirements..................................................12

3.2. Reliability Requirements................................................12

3.3. Performance requirements............................................13

3.4. Security requirements....................................................13

1. Interface......................................................................................14

4.1. User Interface…..........................................................14

4.2. Hardware Interface…..................................................20

4.3. Software Interface..........................................................20

1. Use case Modeling......................................................................22

5.1. Use case diagram…....................................................22

5.2. Misuse case diagram…...............................................23

1. Project Plan.................................................................................25

6.1. Roles...............................................................................25

6.2. Software methodology…..............................................25

6.3. Software tool....................................................................26

6.4. Risk Assessment..............................................................27

6.4.1. Technical risk assessment.................................27

6.4.2. Organisational risk assessment…......................27

6.4.3. Economic risk assessment................................28

6.5. Task estimation ...............................................................29

6.6. Procedures.......................................................................30

1. Design documentation...................................................................33

7.1. Architecture and design patterns…................................33

7.2. Adopted error handling…...............................................35

7.3. Coding standards and conventions…............................35

7.4. Physical deployment.........................................................36

7.4.1. Deployment diagram..........................................36

7.5. Structure diagram..............................................................37

7.5.1. Component diagram...........................................37

7.5.2. Class diagram ....................................................39

7.6. UML diagrams...................................................................40

7.6.1. Sequence diagram.............................................40

7.6.2. Activity diagram..................................................41

7.7. ERD diagram.....................................................................42

1. User help documentation...............................................................43
2. Test Plan........................................................................................54

9.1. Unit tests...........................................................................54

9.2. User Acceptance tests......................................................57

9.3. Test summary...................................................................58

1. Project execution and management...............................................66

10.1. Team organisation...........................................................66

10.2. Methodology used…........................................................68

10.3. Software Tools used........................................................69

10.4. Conclusion......................................................................73

1. References......................................................................................75
2. Appendix A: Contributions..............................................................76

**1.** **Introduction:**

Our skilled team of developers has partnered with the innovative company Future Fridges to create software for their revolutionary new smart refrigerator, the FFsmart. As an ambitious software startup, we relish the opportunity to showcase our talents by designing an advanced solution for Future Fridges' cutting-edge product.

The FFsmart aims to revolutionize kitchen workflow in commercial restaurant settings. This internet-connected refrigerator features a rear door for delivery access, eliminating disruptions during busy service hours. Our software will enable the FFsmart to automatically track ingredients in real-time, send expiry and shortage alerts, and even autonomously reorder supplies.

By leveraging our expertise in requirements elicitation, architecture design, and Agile development processes, we will build a robust system to maximize the potential of the FFsmart. Our developers are dedicated to delivering a user-friendly and reliable software experience. This project allows us to demonstrate our proficiency in developing enterprise-level software that solves real-world problems.

**1.1. Purpose:**

This documentation's main objective is to give Future Fridges an extensive foundation for assessing if the software, created by our software development team, perfectly satisfies their unique demands and expectations. Before the crucial stage of integrating the software with the actual FFsmart hardware.

**1.2. User Characteristics:**

The User Characteristics analysis provides valuable insights into the individuals responsible for the regular management and interaction with the FFsmart fridge within the restaurant environment. This analysis delineates their specific roles, professional experiences, technical proficiencies, and assorted general attributes, offering a comprehensive understanding of the key personnel engaged in the restaurant's operational processes.

**Delivery Personnel**

Responsible for restocking the fridge with fresh ingredients through the rear door

* Have experience using tablet devices to track inventory
* Moderate technical expertise - able to navigate apps and use barcode scanners
* Need simple and efficient interfaces to perform their tasks quickly

**Head Chef**

Manages kitchen operations and oversees ingredient levels

* Extensive culinary expertise but limited technical background
* Relies on software alerts and notifications to monitor fridge status
* Values usability and intuitiveness when using apps and devices

**Sous Chefs**

Work under the head chef to prep ingredients and cook menu items

* Similar expertise and tech proficiency as head chef
* Will occasionally need to check ingredients or be notified of shortages
* Apps should be straightforward with minimal training needed

**Restaurant Owner**

Responsible for overseeing entire restaurant operations

* Often has a business or management background more than technical
* Relies heavily on head chef and managers to handle daily kitchen operations
* Needs oversight of orders and alerts without being involved in day-to-day details

**1.3. Assumptions:**

Numerous assumptions have been made by the software development team responsible for the FFsmart fridge system. It is imperative to recognise that these assumptions carry the potential to exert a profound impact on the system's functionality and, consequently, pose an elevated risk concerning the quality of ingredients within the restaurant's inventory. For instance:

**Network Connectivity:** Assuming a stable and reliable network connection is available for real-time tracking and communication with devices.

**Supplier Cooperation:** Assuming that suppliers are willing and able to collaborate with the system for automated reordering, and that they have the necessary infrastructure for receiving and processing orders.

**Data Accuracy:** Assuming that the data input by users (e.g., item quantities, expiration dates) is accurate and up to date.

**User Access Management:** Assuming that appropriate authentication and access controls can be implemented to restrict system access only to authorized users.

**Hardware Maintenance:** Assuming that any maintenance, repairs or part replacements needed for the physical refrigerator hardware will be handled separately from the software system.

**1.4. Scope and Constraints:**

It is essential that we understand the constraints and limitations that define the design space in order to develop an effective software solution for the FFsmart fridge. These limitations stem from various sources, including legacy systems, compliance requirements, and technical capabilities. We can create an architecture that is optimised for FFsmart's particular needs and environment by first analysing the constraints.

Several factors will constrain the software design options:

* Integration with Legacy Systems: The software needs to allow restaurants to run it in parallel with any existing local inventory systems during a transitional phase.
* Audit Functions: The ability to generate reports showing audit logs, usage history, and reordering records will be required for compliance.
* Access and Security: Role-based access controls are needed to limit system access only to delivery staff and authorized kitchen users. User actions must be logged.
* Reliability Criticality: As a mission-critical kitchen system, maximizing uptime and system robustness is highly important.
* Hardware Limitations: The fridge hardware provides constrained storage, memory, and processing capabilities that software must operate within.
* Regulatory Compliance: The system must comply with all relevant restaurant industry regulations for inventory tracking and reporting.

**1.5. Glossary of terms:**

**-**

**1.6. Overview:**

This document offers a thorough technical overview of the FFsmart solution implementation strategy for our software startup. Utilising our proficiency in resilient software architecture and flexible development methodologies, we will produce a cutting-edge refrigerator inventory management system that satisfies Future Fridges' requirements.

The next sections will go over our suggested methodology for creating, testing, and implementing dependable software to make the most out of the FFsmart platform. Focus areas will consist of:

* Compiling detailed requirements based on interactions with stakeholders.
* Creating user-friendly UI customised for every user type.
* Building a software architecture that is scalable and modular with best practises.
* Implementing food-safe systems that meet regulatory standards.
* Incorporating agile principles into the software development life cycle.
* Delivering comprehensive documentation, help guides, and training materials.
* Providing ongoing maintenance and support.

**1.6.1. Tech Stack Overview:**

To ensure the FFsmart platform's success, our team methodically crafted a comprehensive tech stack, employing a combination of languages and cloud services to create a strong and scalable system that fulfils the highest performance and reliability criteria. In our pursuit, we carefully chose a balanced and dynamic backend architecture, using Java, JavaFX, and AWS as the foundational pieces of our technological framework.

Java is the solid foundation of our backend infrastructure, known for its versatility and broad use in enterprise-grade applications. Java's mature development tools and vast ecosystem enable us to design sophisticated systems that are long-term compatible and maintainable.

In conjunction with Java, JavaFX allows us to design a visually appealing and intuitive user experience. Its adaptable design features enable us to create compelling interfaces that seamlessly link with our backend logic, improving the overall usability of the FFsmart platform.

Furthermore, our use of Amazon Web Services (AWS) demonstrates our dedication to scalability, dependability, and cybersecurity. Using AWS's extensive suite of cloud services, which includes simple queue messaging, storage, and database solutions, we can easily grow our infrastructure to meet changing user demands while maintaining optimal performance and uptime.

In conclusion, our strategic choice of Java, JavaFX, and AWS as the technological foundations of the FFsmart platform indicates our commitment to provide a cutting-edge solution that not only meets, but exceeds, customer expectations.

**2. Functional Requirements:**

**2.1. Requirement elicitation:**

To gather strict requirements, we are going to be using the following techniques: Document analysis and Interview

**2.1.1. Document analysis:**

Document analysis is an elicitation technique where we gather existing documents about Furture Fridge’s FFSmart Fridge and go through them to create requirements. This is useful for many reasons. Firstly, helps to identify key stakeholders within the project, for the smart fridge this includes chefs, delivery drivers, servers, and owners. Secondly, helps our team understand the existing process in place, for example, ordering products on Mondays. Finally, it helps our team find unclear requirements, we can then create questions for the interview from this (next section).

**2.1.2. Interviews:**

Interviews are where we ask specifically selected questions to a Future Fridges member. By using interviews, we are able to achieve stricter requirements and remove unclear requirements from our list.

Our team have come up with the following questions for our interview.

* What are the specific user interface design preferences for each user role?
* Are there any design guidelines, colour schemes, or branding considerations?
* What method should the logins be distributed especially with delivery drivers?
* What should the contents of the health and safety report be?
* Is there any OS-specific compatibility due to the hardware they are using?
* In requirement 4 how will we know the quantity will run out in 3 days? Is there a certain quantity where the system should push a notification that an item will run out? E.g. might use a lot of 1 item on one day but not another.
* What level of security does the system need?
* What device will the drivers use to access the application?
* Are there going to be multiple suppliers based on an item?
* Will a delivery driver need a 2-factor authentication process to access the application?
* What pain points does your client’s current inventory management process have?
* To you what features are absolutely critical?
* Will there need to be an admin which will have more functionality than the head chef?
* Would you rather an recipe based functionality or reporting ingredients report and usage analytics functionality? Explain?
* Are there any constraints with fridge capacity and size?
* Does the user manually set the quanity they want?

**2.2. FFSmart Functional Requirements:**

The FFsmart software must meet the following functional requirements:

High Importance

* Automated and real-time tracking of food inserted into and removed from the fridge. The software must track all food in the fridge and update the fridge stock level accordingly. This information must be accessible by all authorised users.
  + User(chefs,delivery drivers etc) should be able to add and remove items from fridge.
  + The added or removed item will automatically be logged on the application and stock level will be updated.
  + Stock level will be visible to all authorised users.
* A login system to track who inserts/removes items from the fridge. The software must have a login system to track which user inserted or removed a food item from the fridge. This information can be used to track expiration dates.
  + The application will have a login system.
  + If a user’s credentials are correct, allow access, if not deny access.
  + Once logged in the user will be able to access the fridge and insert and remove items.
  + The application will assign the user’s username to each fridge transaction.
* Keep track of the name, quantity and expiration date of all items in the fridge. Push notifications/alerts to the device of the head chef 3 days before the expiration date of an item. This will allow the chef to reduce waste by using the item before expiration.
  + Application will keep track of item details(name, quantity, and the expiration date).
  + 3 days before an item expires a notification will be sent to the head chef alerting him/her.
* Alert the head chef if an item is about to run out of quantity 3 days before the item runs out. The software must push alerts to the head chef before an item runs out. This will allow the chef to reorder the item before it runs out.
  + Application will keep track of item details(name, quantity, and the expiration date).
  + 3 days before an item runs out a notification will be sent to the head chef alerting him/her.
  + The application will then give the head chef the choice to reorder the item.
* Give the head chef priority access to configure/change what users have access to the fridge and whether they are allowed to insert/remove items. The head chef must also be able to change each user's permissions. These features must not be visible to regular chefs.
  + The head chef will have special permissions in the application.
  + The head chef will ahve the ability to change other user’s permissions from within the application.
  + These special permissions will not be accessible to any other user.
* The application must enable the delivery person to record what items they replenish in the fridge. The software must provide an app that allows the delivery person to update the stock in the fridge. This should be updated with the fridge.
  + Deliver drivers will have access to the application.
  + The application will give delivery drivers the ability to record what items they ahve added to the fridge.
  + The application will automatically update the stock information.
* Provide a list of items to the head chef that are close to running out in the fridge and on each Monday auto reorder the items. The order for the items needs to be sent to the correct supplier. This will ensure all items will be stocked for the following days.
  + On a specific day e.g. Sunday, the application will push an alert of a list of Items that are close to running out to the head chef.
  + On Monday the application will automatically reorder any item that is close to running out.
  + The application will send the correct item to the correct supplier.
* Create an easy-to-use app interface for each user on their respective devices. The software interface must be easy to use for all. The app interface should be clear, concise, and easy to navigate.
  + The application will be easy to use, with a simple interface.
  + The application will be clear and the colour scheme will distinguish text from the background.
* The app should allow delivery drivers to log in. This login should allow them to open the door of the FFsmart. This will enable the delivery person to replenish the fridge without having to interact with others in the restaurant.
  + The application will allow drivers to login with correct credentaials.
  + If a driver does has incorrect credentials the application will deny them entry.
  + Once logged in the application will allow them access to the FFSmart fridge.
* The software must provide a checking function that allows the head chef to verify that the delivery person has inserted the correct items into the fridge. This will ensure the correct stock is in the fridge at all times.
  + The application will have a function that allows the head chef to check if the correct items are inserted into the fridge.
  + If the incorrect items are inserted the application will send an alert to the delivery driver.
* The software must prevent food servers from accessing the fridge. This is important to ensure food safety and to prevent food theft.
  + The application will not be accessible to servers.

Middle Importance

* The software must generate reports that the head chef can submit to the health and safety officer. These reports should include information on food safety, such as temperature logs and expiry dates etc.
  + The application will generate a report about food safety, temperature logs and expiry dates.
  + This will be sent to the health and safety officer by the application routinely.

Low Importance

The client also asked for other features which may be useful for the fridge. The functional requirements include:

* A recipe management system that allows chefs to create and store recipes, and track the ingredients required for each recipe. This would help chefs to plan their meals and ensure that they have all.
  + The application should save recipes in a database.
  + Chefs should be able to navigate through recipies and see the contents.
  + The recipies should highlight the stock level of the ingrediants and show if the recipie can be made at the current moment.

**3. Non-Functional Requirements:**

The FFsmart software must meet the following non-functional requirements:

**3.1. Usability requirements:**

High Importance

* The system should be easy to learn and use, even for users with no prior experience with smart fridges. This will mean creating a simple user interface, clearly labelled buttons etc.This will allow easy use for all parties using the system.
  + The application will have a simple interface.
  + The application will be clear and easy to learn.
* The system should be responsive to user input and provide feedback on user actions. There should be success messages when items are added and buttons should correlate with their functions.
  + The application will be responsive to every user input.
  + The application will push successful alerts for key functions.
  + The application will push fail alerts to unsuccessful user activity.
  + Buttons will be clear and correlate with their functions.
* The system should be able to handle errors in user input gracefully. If there is an error the system should display an appropriate error message and advise a solution.
  + The application will handle errors without the whole application crashing.
  + The application will display errors if there is an error in the system.
* The user interfaces (both for the app and fridge controls) should be intuitive and user-friendly.
  + The user interfaces will be clear.
  + The user interfaces will be user-friendly.
* The system should provide clear and helpful error messages to guide users in case of issues. When their incorrect user input or the system has an error an appropriate error message should occur.
  + Error messaging will be clear, appropriate, and helpful.
  + An Error message will suggest useful ways to resolve the issue.

**3.2. Reliability requirements:**

High Importance

* The system should be available 99.9% of the time. This will ensure the chefs will be able to access the fridge at all times.
  + The application will be active and running for the majority of the time.

Medium Importance

* The system should be able to process 100 transactions per second. This will increase productivity within the resturant.
  + The application will be able to process 100 transactions per second as a base line.

**3.3. Performance requirements:**

High Importance

* The system should be responsive, with minimal latency when users interact with the app and fridge door controls.
  + The application will be responsive to user action.
  + Using the application to open the fridge will be fast and hastle-free.
* The system should support concurrent access by multiple users, including delivery people, chefs, and the head chef, without significant performance degradation.
  + The application will support multiple users at the same time.
  + The speed of the application will not be affected with concurrent users.
* The real-time tracking of food items should be accurate, with updates reflected immediately. This will ensure reliable information is relayed to the chefs and the head chef.
  + When item transactions occur the application will automatically update the item quantities.
  + The update will be pushed to all users immediately and automatically.

Medium Importance

* The system should be able to process up to 1000 food items in its database. This will ensure numerous amounts of different food will be able to be stored.
  + The application database will be able to store up to 100 different food items.
  + The database will not contain duplicate items.

**3.4. Security requirements:**

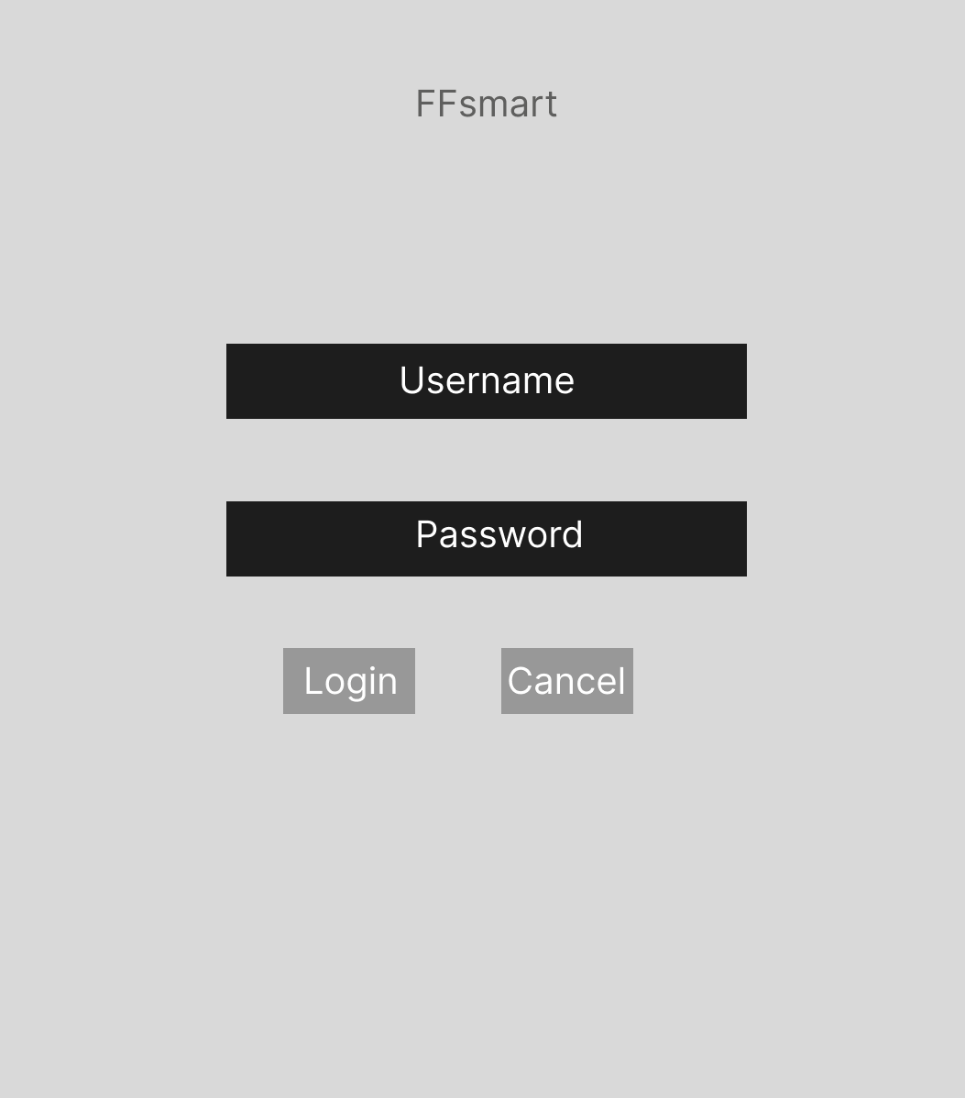
High Importance

* User authentication and authorization mechanisms should be secure, preventing unauthorized access to the system. No user without a login should be able to access the system.
  + The user login system will be secure.
  + No user without a login will be able to access the application.
* Password should be encrypted, this ensures that if the database is breached accounts will not be able to be accessed.
  + Password will be encrypted using a hashing algorithm.

**4 Interface :**

**4.1 User interface:**

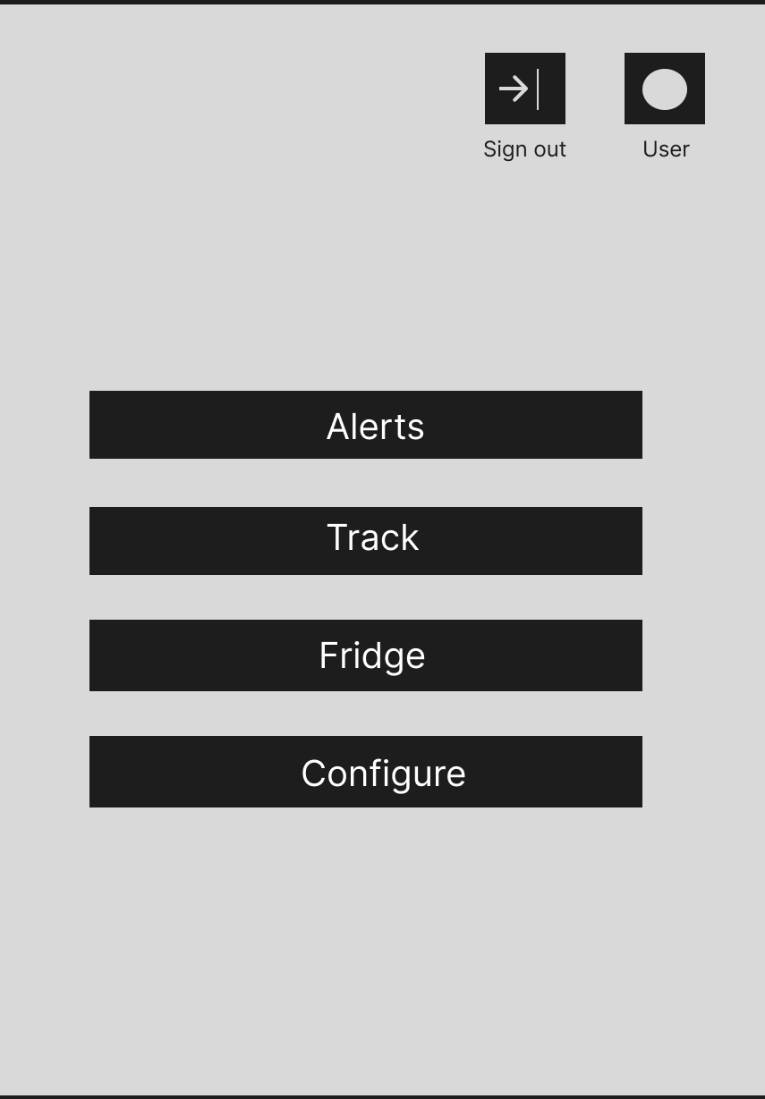
**Login page**



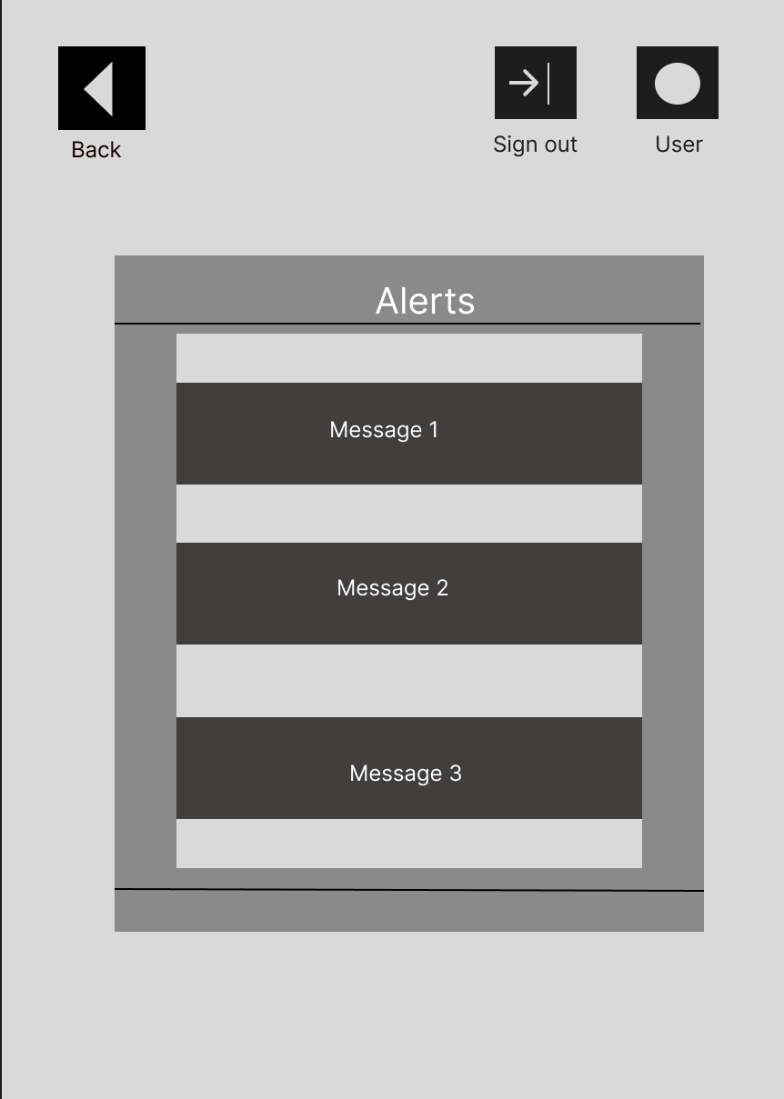
**Main page for normal users**



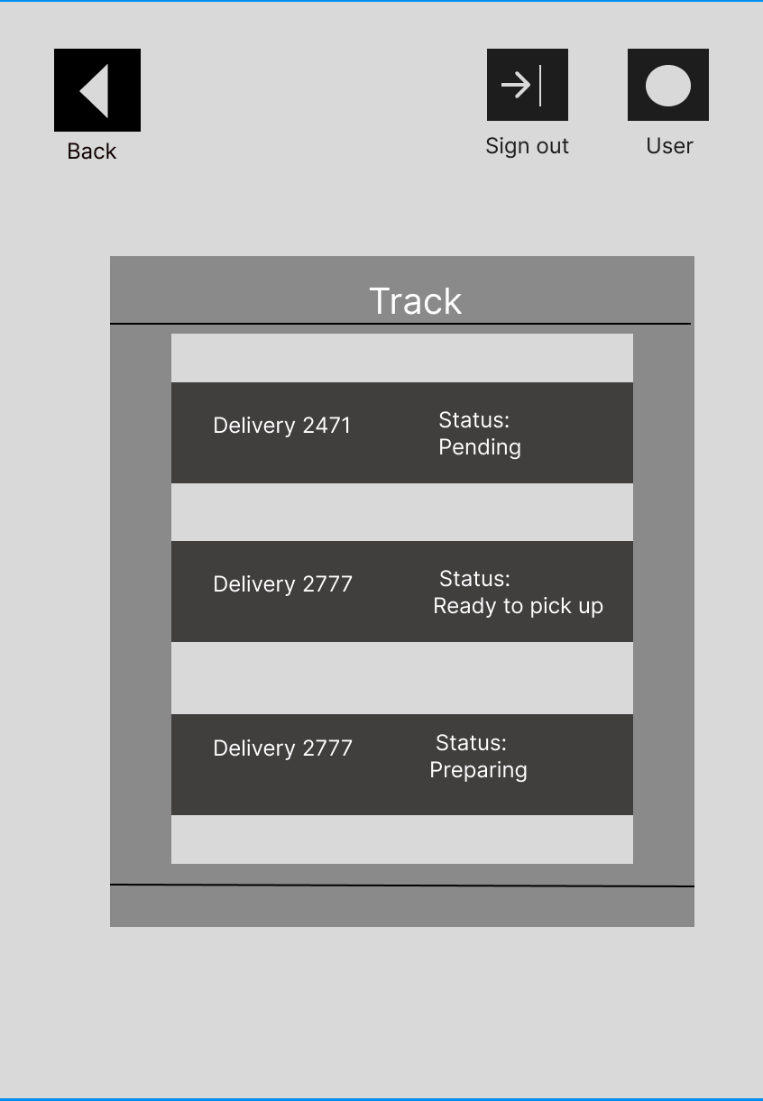
Main page for head chefs where the configure button allows for the head chefs to allow roles of the users to be changed



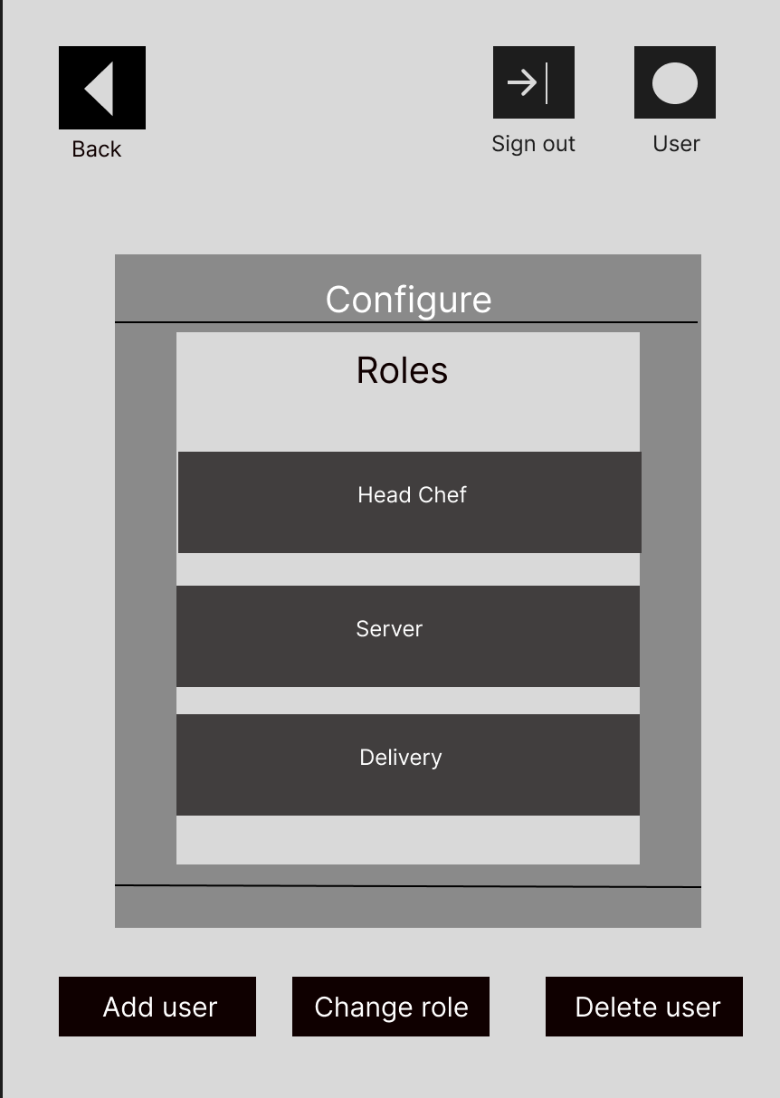
This what the alerts and message will look like as alerts and messages come through.

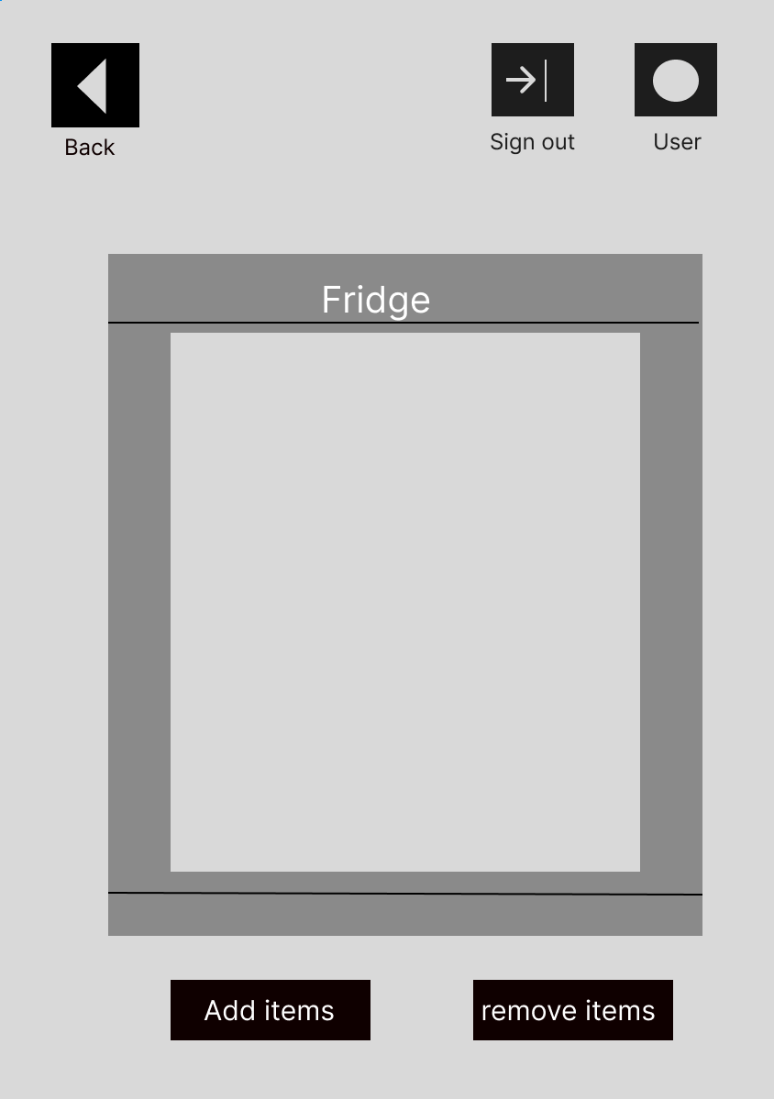


**This how deliveries will be tracked:**



**How users will be added , changing roles and deleted**





**4.2 Hardware interface:**

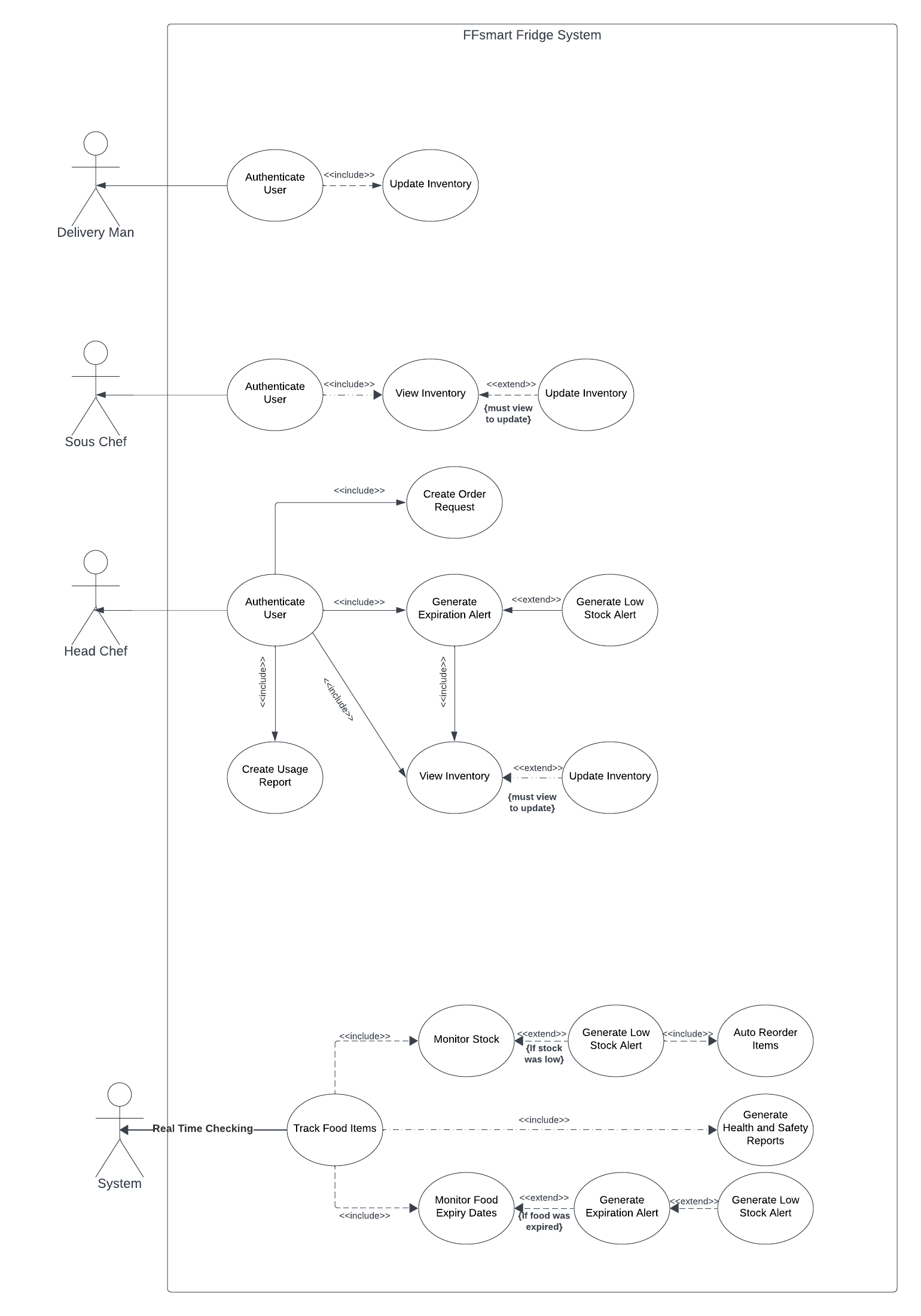
* User devices to run on various devices for specfiic user roles
* Cameras in order to idenfity what items have been put in the fridge
* Temprature sensors that can track if specifc food need to be at a specific temprature
* A physical or virtual server in order to store and mange the database
* Touch screen display
* Delivery person deivce
* RFID or barcode scanner to input items into the fridge

**4.3 Software interface:**

* A database has to made in order to store staff that take out food as well as storing what food is in the fridge
* User hierarchy so that only certain users have access to certain stock
* Need to create alerts for stock that is running out of stock 3 days before.
* Communication protocols e.g. how the apps will communicate with each other in order to place orders etc.
* Colours for the user interface need to be compatible for people with colour blindness.
* API for supplier integration
* Security protocols
* Notification Service
* Backend server to handle things like user authentication and inventory management

**5.** **Use Case Modelling:**

**5.1. Use case diagram:**



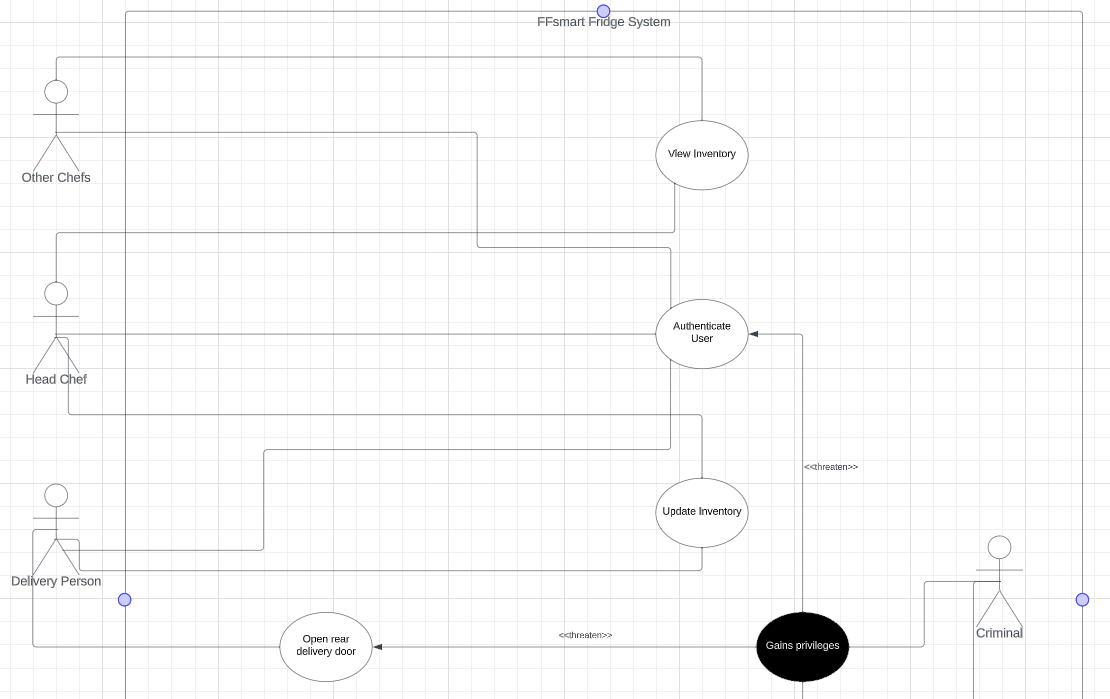
**Use Case Explaination:**

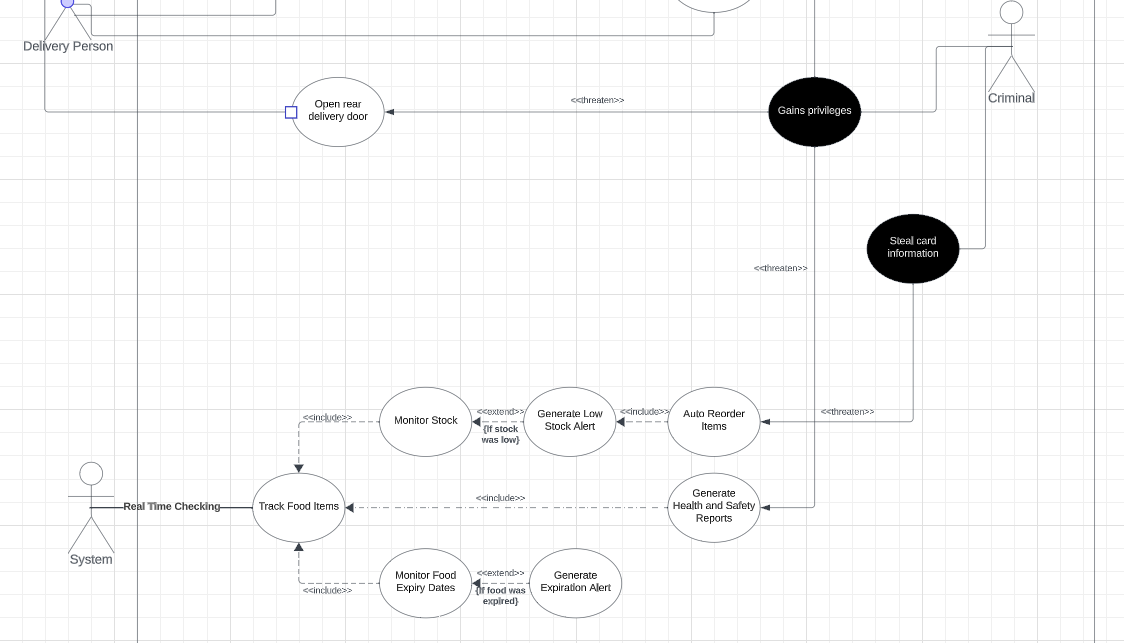
The use case diagram offers a high-level overview of the FFsmart fridge system's functions as well as the interactions between its key users. There are four primary actors: the System, which handles automatic alerts, restocking, and reports the Delivery Person, who stocks goods, the Head Chef, who manages ordering and inventory requirements, and the Chef, who assists.

The system's features are organised into seven primary use cases: authenticate user, view inventory, update inventory, create reorder request, generate expiration alert, generate low stock alert, and create usage report. The graphic depicts how each actor interacts with each use case; for example, the System generates alerts automatically, while the Delivery Person merely needs to authenticate and update inventory. Relationships between use cases are also demonstrated, such as View Inventory extending Update Inventory.

This use case diagram provides a visual description of the FFsmart fridge system's primary user goals and needed high-level system operations, allowing users to quickly comprehend expected system capabilities and roles. The diagram organises logically connected functions, depicts essential relationships, and defines the scope of the system being designed. Overall, it represents the fundamental functional needs and user interactions for the new FFsmart system in an implementation-independent, abstract manner.

**5.2. Misuse cases:**





**Misuse Case Explanation:**

This misuse case diagram provides a graphical visualisation of the FFsmart system and the possible vulnerabilities that may come with it. The five actors that are involved within this diagram are the System, which performs real time checking in order to track food items and monitor and reorder stock, the Delivery Person, who can update inventory and open the rear delivery door, the Head Chef, who can update and view inventory, and the other chefs who are only able to view inventory. These users are logged in through an authentication system. The final actor is a Criminal, which represents any possible attacker that may have a motive for accessing the system for malicious intent.

There are two primary potential misuse cases that can be seen in the diagram. The first one is the criminal gaining administrative privileges over the FFsmart system, which would open up numerous points of vulnerability for the company, For example, the criminal could potentially open the rear delivery door remotely, opening up the risk for burglary. There are also other potential attacks the criminal could utilise, such as generating false health and safety reports and gaining access to sensitive employee data. Another potential misuse case is a criminal gaining access to FFsmart’s financials, such as stealing bank details. This could potentially put the business in jeopardy and will make it impossible for new stock to be ordered and obtained.

**6.** **Project Plan:**

**6.1. Roles:**

All team members are required to work on all aspects of the project. However, each team member will have a primary role in which they will specialise in. We are going to organise our team as the following:

Project manager – Jared

Project managers plan and coordinate projects from idea to delivery. Jared has previous experience in project management proving he is most suitable for the role. He has the organised mindset, focus and determination needed for a project manager. This will ensure successful completion of a project to a high standard.

Lead Software developer – Dammy

A lead developer is a technical professional who guides a team of software developers through the implementation phase of the SDLC. Dammy is most suitable for this role due to his technical experience. He has experience working in large and small software development teams. He has the technical skills to ensure the implementation of the project goes smoothly.

Analyst - Max

An analyst’s role is to assess the company and client’s needs, review the information about the product and analyse it. Max has client-facing experience and therefore is most suitable for the role. He has a very analytical mindset and this will him even more in the role. With Max as the analyst, we will be able to understand the project and all information given.

Software Tester – Akshay

A software tester is an individual who tests for bugs, defects, or errors in the project. Akshay is the software tester due to his experience in testing code. In multiple previous projects, he has been in charge of testing the code and making sure the code performs well. He has a strong knowledge and understanding of testing methodologies and technologies. With Akshay as the tester will will be able to deliver the project bug-free.

**6.2. Software methodology:**

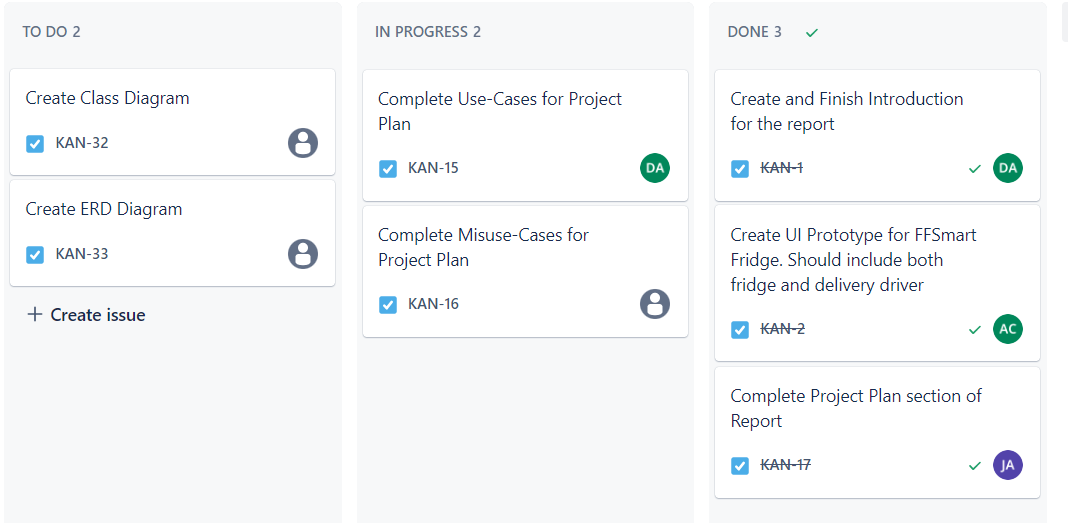
We will be using the Agile methodology for our project. Agile methodology involves breaking the project into smaller subparts and developing iteratively. Since developing complex software can no longer be completely defined at the start (Karrenbauer, J., Wiesche, M. and Krcmar, H., 2019). Developing in iterations allows the development team to adapt quickly to changing requirements. (Cohen, D., Lindvall, M. and Costa, P., 2004). Agile is the best methodology for our project due to its flexibility and faster delivery. With Agile our team will be able to create smaller pieces of functionality. This will allow us to create an application, although incomplete, faster. Furthermore, agile offers flexibility to make changes to the software even during the late stages of the SDLC. (Gaba, I., 2023). Due to uncertainty with some requirements and our team extending the application and adding requirements the flexibility of agile is suitable for our project. However, the drawback of agile is that it is resource intensive. The agile methodology requires close collaboration and communication. If this is not done then the project may get delayed because of miscommunication, changing requirements etc. We will combat this by closely communicating and collaborating as well as having weekly meetings.

We also took the waterfall methodology into consideration. The waterfall method is a static model and approaches systems development linearly and sequentially, completing one activity before the other(Adenowo, A.A. and Adenowo, B.A., 2013). The waterfall method is beneficial as it allows for a clear structured approach to the problem. It is truly effective when the requirements of a project are clearly defined from the start. Furthermore, the waterfall method is very predictable in terms of timing. This is because all the requirements are stable and structured so time can be estimated easily. However, what lets the waterfall method down is its lack of flexibility. Due to not all our requirements being certain and maybe changing the waterfall method is not suitable for our team. This is why we chose the agile method over it.

**6.3. Software tools:**

Here are the following software tools we will be using to complete the project:

We will be using Jira to track our progress. Jira Software is a highly customizable project management solution that enables teams to build frameworks around whichever project management methodology they decide to use (Good L, 2023). It is especially useful for Agile or Scrum methodologies. This is beneficial to our team as we are using an agile approach. This will allow us to streamline our tasks instantly and efficiently in order to build our project. This is why we will use Jira to track our progress. A screenshot of how we are using Jira can be seen below.



We will be using Microsoft Teams to facilitate communication within the team. Microsoft Teams is a communication application for businesses. It is beneficial for our team as all users are familiar with the application. Furthermore, it allows for easy meeting scheduling, notifications for messages and mobile support. This allows for easy and quick communication with the whole team. This is why we decided to use Microsoft Teams for communication.

We will be using Microsoft Word for documentation. Microsoft Word is beneficial as all team members have used it before and it allows for easy collaboration. Microsoft Word easily integrates with Microsoft Teams. All changes autosave to the cloud so all work is automatically backed up and work will not be lost. Also, it allows for use to easily track changes that each team member has done. This will ensure only the best quality work is being done.

We will be using Git and GitHub for code source control. Git is version control software and GitHub is a web-based platform that uses Git. Git allows us to code collaboratively. It is the most popular source control platform and a platform that our team has experience in. It allows us to track code changes, track who made changes and work on different requirements using branches. This will help us, especially in our agile approach. This is why we will be using Git and GitHub.

We will be using IntelliJ for developing our software. IntelliJ is an integrated development environment written in Java that all our team have used before. It integrates with the languages we will use, Java and AWS. It offers a fast and efficient way to navigate code and has superior debugging functionality which will help us find errors quickly. Furthermore, it supports JavaFX which we will be using to create the frontend. This is why we will be using IntelliJ.

For testing JUnit. This is a common unit testing functionality. It allows developers to write tests and run them on the platform. It also has a built-in reporter that can print out the results of the tests. Finally, it allows for automated testing which will allow us to find bugs quicker.

We will be using Lucid Chart to draw diagrams. Lucid charts allow our team to easily build multiple different types of diagrams for our project. Furthermore, it allows us to easily share the diagram with other users. This makes Lucid Chart superior to other websites. This is why we chose to use Lucid charts.

**6.4. Risk assessment:**

Several risks come with the project. Here is an assessment of these risks:

**6.4.1. Technical Risk Assessment:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Description** | **Probability** | **Impact** | **Mitigation Plan** |
| Poorly defined requirements which can lead to poorly defined features. | If the requirements of the application are not properly defined it can lead to improper development of the system's features, this can cause instability and/or lack of functionality in the software resulting in a product that is not fit for its purpose. | Low | High | We plan to mitigate this by defining the requirements early using document analysis. Once this is done we will have an interview with the stakeholder. In this interview we will clarify all unclear requirements. |
| Poor code quality. | Code of low quality will lead to more bugs and more failures in the code. This results in a failing system for our clients. | Moderate | High | We will mitigate this by following good coding practices. As well as this we will add comments to the code to explain what we are doing. |
| Scope Creep. | Scope creep is the increase in requirements over time. This can lead to more time loss for other parts of the project e.g. testing. This can lead to an insufficient product for our clients. | High | High | We will mitigate this by prioritising our requirements and following only our requirements. Once the requirements are completed and tested. |

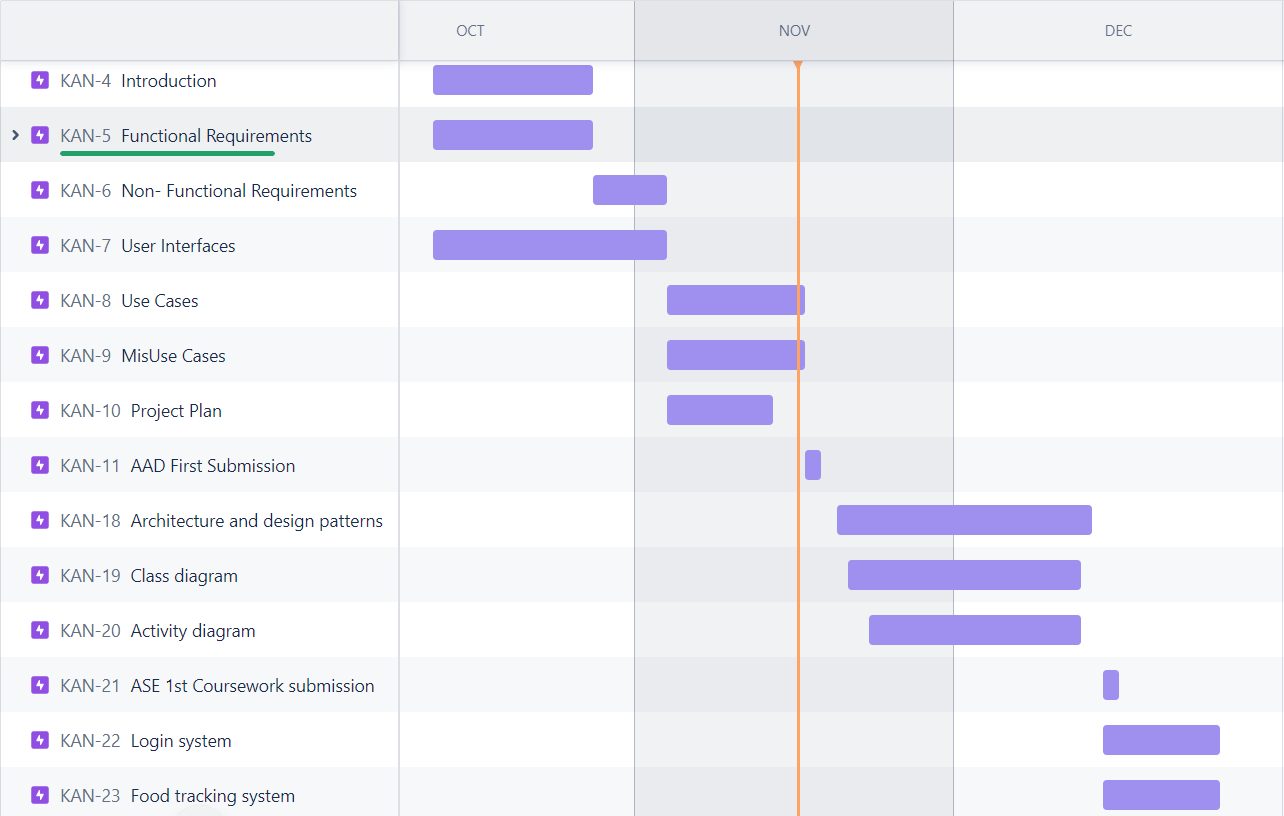
**6.4.2. Organisational Risk Assessment:**

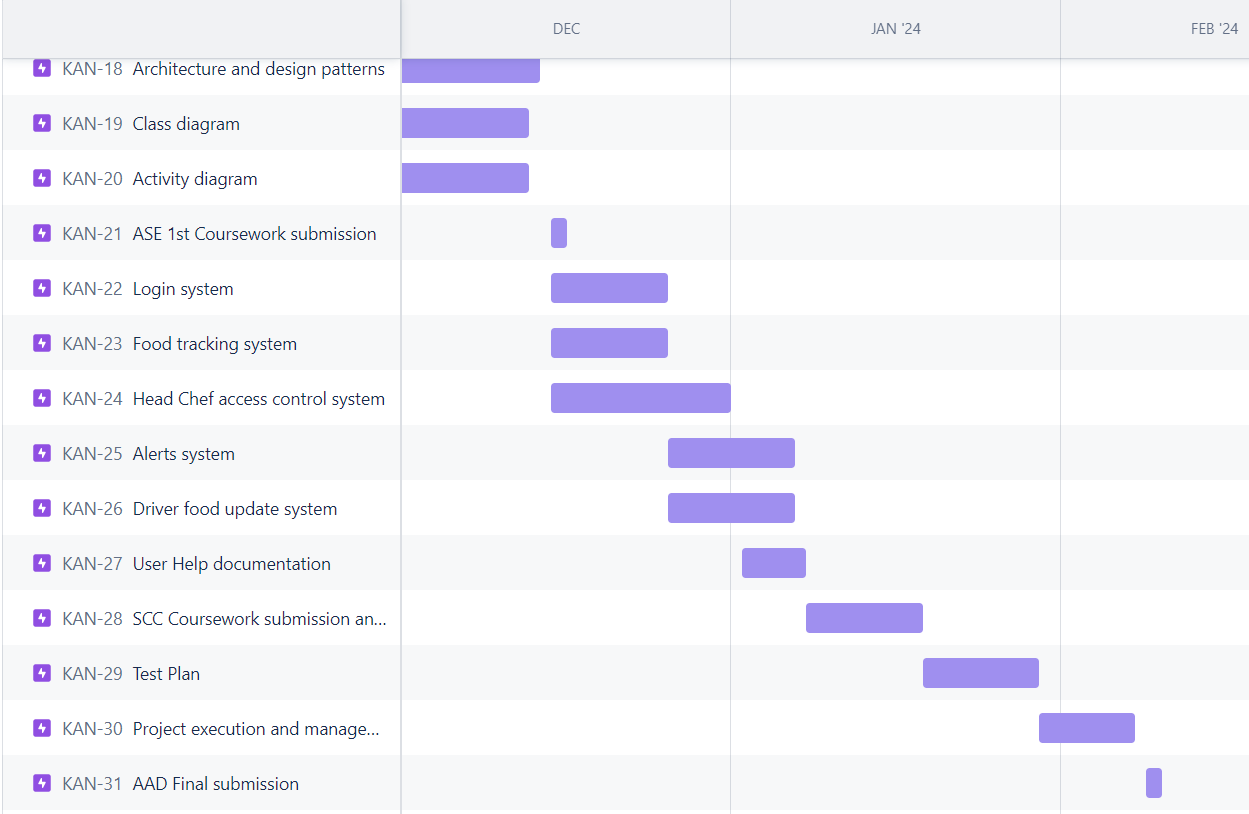
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Description** | **Probability** | **Impact** | **Mitigation Plan** |
| Strict task deadlines. | Setting strict deadlines may lead to team members having to work overtime. This increase in workload may cause stress and lower-quality work. Which may lead to a less-than-satisfactory product. | Moderate | Moderate | We plan to mitigate this risk by starting the project early. This will allow us to have more time. As well as this we will collaborate in creating a Gantt chart together so we can all agree on the deadlines. |
| Poor documentation. | Poor documentation can make it difficult for third parties to understand the application. Especially if a new member was added to the group or the clients wanted to understand the application in more detail. | Low | Moderate | We will mitigate this by documenting each part of the project thoroughly. This includes the plan, design, implementation and test phases. |
| Poor team communication. | Poor communication can lead to decreased productivity within the team. This will lead to a longer time taken to complete the project. | Low | High | To mitigate this risk we have created a team channel to communicate with team members. As well as this we will have bi-weekly meetings to inform each other of our progress. |
| Non-Contributing team members. | Team members may not contribute to the project or contribute less than other members. Leading to a larger workload for other team members. | High | High | To mitigate this risk we have create a procedure to deal with this situation. |
| Other Commitments. | Other commitments may mean we have less time for the project. These commitments may be coursework, exams, or personal commitments. | High | Moderate | To mitigate this risk we will take into account other commitments when planning the project timeline. |

**6.4.3. Economic Risk Assessment:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Description** | **Probability** | **Impact** | **Mitigation Plan** |
| High system maintanence costs. | If the project is created at a low standard, many errors may occur in the source code. This will likely result in higher maintenance costs. | Moderate | Low-Moderate | To mitigate this we will create the project at a high standard. This will lead to a robust programme, which will lead to lower maintenance costs. |

**6.5. Task estimation:**





**6.6. Procedure:**

The following is the procedure for tasks not being delivered on time:

* Contact team member after deadline if work no submitted.
  + Reasonable response:
    - Assign new deadline and assign support to team member
  + No response/ Unreasonable response:
    - Reassign task to someone else

The following is the procedure for dealing with team member not participating:

* Contact team member to discuss participation.
  + Reasonable response:
    - Provide support to member and reassign some tasks to other team members
  + No response/ Unreasonable response:
    - Reassign tasks to other team members

The following is the procedure for setting and confirming deadlines:

* Create a meeting for all team members
* Assign timeframe for each task
* Take into account other commitments/deadlines
* Make sure all parties agree to deadline.
* Set task deadlines

The following is the procedure for how to decide and reach census on what technologies to use:

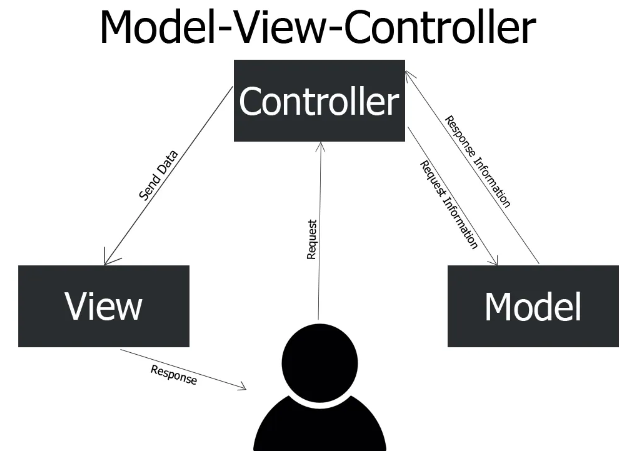
* Each member researches 1 technology our team should use.
* Set up a team meeting to discuss our findings.
* Each member has 10 minutes to discuss their findings.
* Then other members have 5 minutes to question the said member.
* Discuss to find a compromise.
  + If there is still disagreement by the end of the meeting:
    - Each member votes on a technology
    - Majority wins.

**7.** **Design Documentation:**

**7.1. Architecture and design patterns:**

**7.1.2. Architecture patterns:**

There are multiple Architectures that we discussed for our project. The main two we thought were beneficial to our project were the model-view-controller architecture pattern and the microservice architecture pattern.



*Figure 1: a diagram showing how the MVC model (Spinelli, J. ,2019)*

The model-view-controller, or MVC, architecture pattern separates the application into three components. Firstly, the Model represents the data and business logic of the application. Secondly, View which displays the user interface and interacts with users. Finally, the Controller this handles user input, manipulates the model and updates the view accordingly.

The View of the system is the GUI, graphical user interface. The user will interact with the GUI for each function e.g. sign up, add item etc. The GUI will be created using JavaFX and possibly CSS and HTML for style purposes. These languages will enable us to have a responsive, user-friendly interface.

The views are updated by the controller and retrieve data from the model. There are different views for each user role, e.g. Head Chef, Chef, Delivery person. It is through the views that the users can do certain actions, for example, add item, remove item, sign in etc.

The controller takes the action performed by the user in the view, such as user inputs and converts it into rules the model will process.

The model acts as the backend for the system and provides the core functionality. The model communicates with the database, for inserting users, items etc. Furthermore, it controls the permissions of each user.

The advantages of MVC are:

* The controller manages the functionality of the system, this allows for seamless addition and maintenance of multiple views. Changes made to a view does not affect the other views. This allows for multiple users to develop concurrently. Resulting in quicker development times.
* The MVC architecture can result in high cohesion. High cohesion contributes to the robust and easy-to-maintain code, as well as encouraging reusability. This is due to the logical grouping of actions in the architecture.
* The separation in model, views and controllers leads to minimal coupling. Low coupling can lead to the separation of elements. Which means errors in one element will not affect other parts of the code.
* Finally, each component in the MVC architecture can be tested independently. This makes it easier to identify and fix issues. This contributes to a more robust and reliable system.

The disadvantages of MVC are:

* The MVC architecture may introduce unnecessary complexity to an application. With simpler applications with limited functionality, the separation of model, view, and controller may be to complex. Hence, a more streamlined approach may be more beneficial.
* Another challenge to MVC is the learning curve it presents. Understanding and implementing the MVC structure may be confusing for developers to understand. This can slow down development time.

The microservices architecture pattern breaks down the overall software into small, independent services that communicate through APIs. Each service is focused on a specific action that the user needs to do. These APIs can be deployed and scaled independently.

For the microservices architecture, we will create the client application which will include the front end and mostly contain how the user interacts with the system. Then we will create the server which will contain all the microservices. For example, registering a user, login up, inserting food etc. This can then be deployed on a service such as Tomcat.

The advantages of microservice architecture are:

* Each microservice can be scaled independently based on each specific workload. This means if a lot of users are using one service the specific service can be scaled, This makes the application more resilient to crashing.
* Each microservice is completely separate from the others. This means if one microservice fails the whole system does not fail. This results in higher fault tolerance.

The disadvantages of microservice architecture are:

* Managing and coordinating the multiple services can be very complex. This increased complexity can lead to higher development times.
* Having microservices means that we need to make sure the data is consistent when sending data from client to server. If data is not consistent then the user will not be able to use the application as expected.

For the project, we will be opting for the MVC architecture over the microservice architecture. This is mainly because of the separation between the view, model, and controller. This will be beneficial for application due to the multiple views and model. The microservice architecture is less beneficial due to the coordination of multiple services being complex. This makes MVC architecture better than microservice architecture.

**7.2. Adopted error handling:**

Our chosen method of error handling is by using exception handling. This method allows us to catch errors and exceptions without the whole program breaking. Most modern programming languages rely on exceptions for dealing with errors (Cabral, B. and Marques, P., 2006) hence using exception handling will be highly beneficial to our program. Primarily, it allows for the isolation of error-handling logic from the main code. This results in more maintainable code. Furthermore, it allows developers to handle certain conditions without interrupting the running of the current program. Overall, it creates a more robust program that will be able to handle multiple faults without crashing.



**7.3. Coding standards and conventions:**

The main coding conventions that we are going to use are camel case naming convention, consistent indenting and formatting and robust error handling when creating the solution.

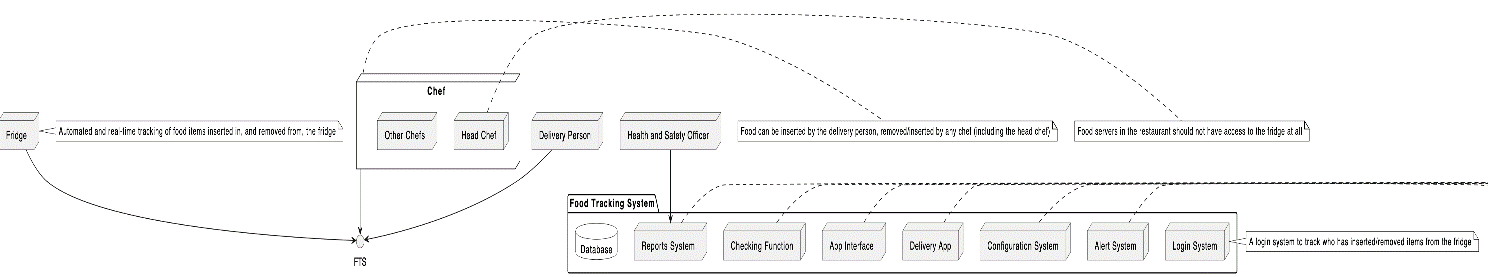
Camel case naming convention allows our code to become more readable and maintanis consistency across the codebase. By leveraging this convention, we can make variable names and method names easier to understand for all developers. This helps us to quickly grasp the names and methods of different functionality within the code. Consistent use of camel case also aligns with Java naming conventions, fostering a standardization that promotes collaboration and code comprehension among team members.

By using consistent indentation and formatting we were able to drastically improve code readability and comprehension. Consistent indentation enhances code structure and helps us visually identify code blocks and logical sections. By using proper indentation, we could easily discern between structures such as loops, conditionals, and method definitions. Moreover, well-formatted code allowed for an easier experience when reviewing code. This allowed us to find errors and correct them quickly. Indentation and formatting were crucial in creating the program for the FFSmart Fridge.

Finally, we are going to use robust error handling. Robust error handling is essential for writing reliable and resilient code. By using error-handing such as try-catch blocks and exception handling we can manage how the program will react to errors and exceptions. This will increase the reliability of the overall program. Moreover, error handling enables our team to diagnose and troubleshoot issues more efficiently, improving the debugging and maintenance process. By proactively addressing potential errors and failures. As a result, it is vital for our program.

**7.4. Physical deployment:**

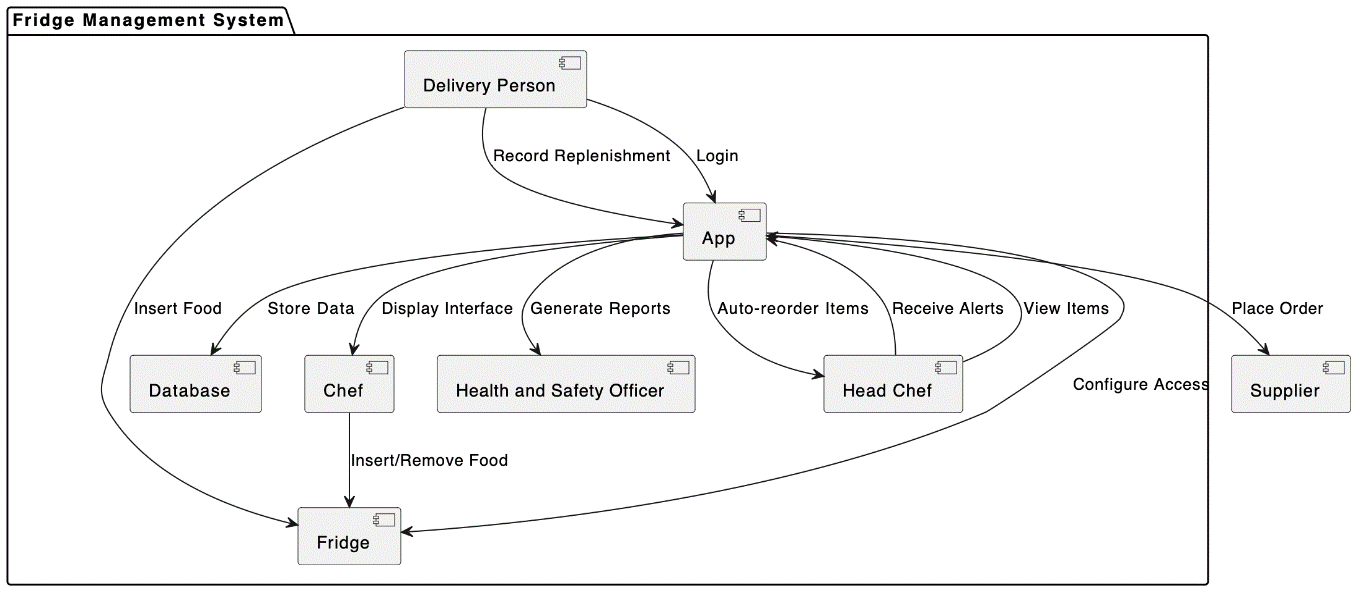
**7.4.1. Deployment diagram:**



In the deployment diagram for the FFsmart fridge system, chefs are depicted as individual nodes representing their computing devices, each designated for interacting with the fridge. Additionally, the fridge tracking system is portrayed as a distinct node within the diagram, signifying its separate entity in the system architecture. Surrounding these nodes are network connections facilitating communication between the chefs' devices and the fridge tracking system. These connections enable chefs to interact with the fridge tracking system, manage inventory, track food items, and receive alerts regarding item expiry or low quantities. The other actors can also interact with the fridge. for example, delivery people. All Food Tracking data is stored in a database.

**7.5. Structure diagrams:**

**7.5.1. Component diagram:**



The core component of the system is the Fridge, serving as the physical refrigeration unit for storing food items. With dependencies on other components, the Fridge relies on seamless integration to manage its contents effectively.

The Database component acts as the repository for storing comprehensive information about food items, including their names, quantities, expiration dates, and other relevant details. Interacting with the Chef component, it facilitates the management of food items within the fridge, enabling tasks such as adding, removing, and monitoring inventory.

Responsible for delivery and inventory updates, the Delivery Person component ensures the timely replenishment of food items in the fridge while updating the Database component with new inventory data.

The Store Data Display Interface component provides a user-friendly interface for monitoring food items within the fridge, displaying vital information such as quantities and expiration dates. It is utilized by the Health and Safety Officer and Head Chef components to oversee food management activities.

Generating insightful reports about inventory levels, expiration dates, and usage patterns, the Generate Reports component empowers decision-making processes for the Health and Safety Officer and Head Chef, ensuring informed management of food items.

The Health and Safety Officer component plays a critical role in ensuring compliance with health and safety regulations by monitoring food items within the fridge. It interacts with the Database, Store Data Display Interface, and Generate Reports components to oversee food safety and regulatory compliance.

The App component serves as the primary user interface, offering a range of features including auto-reordering, alerts, item viewing, access configuration, and order placement.

Automating the reordering process, the Auto-reorder Items component ensures that food items are replenished promptly when quantities run low or expiration dates near.

Sending timely alerts to users, the Receive Alerts component notifies them when food items are running low in quantity or approaching expiration.

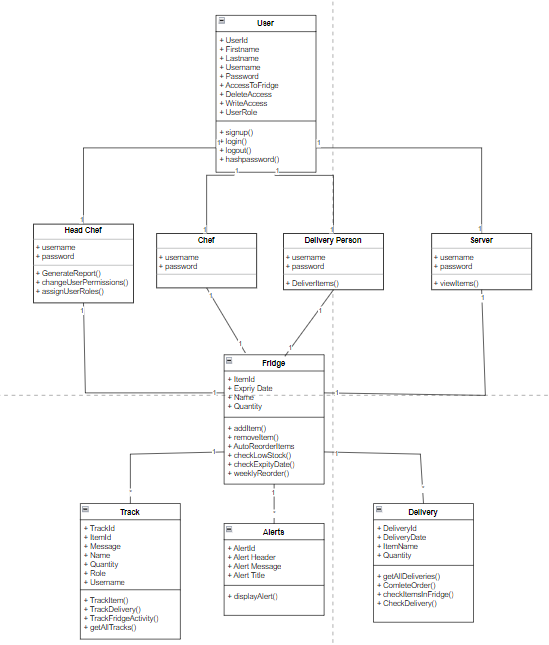
The Head Chef component holds responsibility for strategic decision-making regarding inventory, ordering, and usage of food items. It interacts with various components including the Database, Store Data Display Interface, Generate Reports, and App to access and manage food data effectively.

Empowering the Head Chef to configure access permissions for other users, the Configure Access component ensures proper user management within the system.

The Place Order component enables users to place orders for food items through the App, facilitating seamless communication with the Supplier Fridge Management System to transmit order details efficiently.

In summary, the system's architecture is designed to facilitate efficient management of food items within the Fridge through seamless interaction between various components, ensuring compliance with regulations, optimization of inventory, and enhanced user experience.

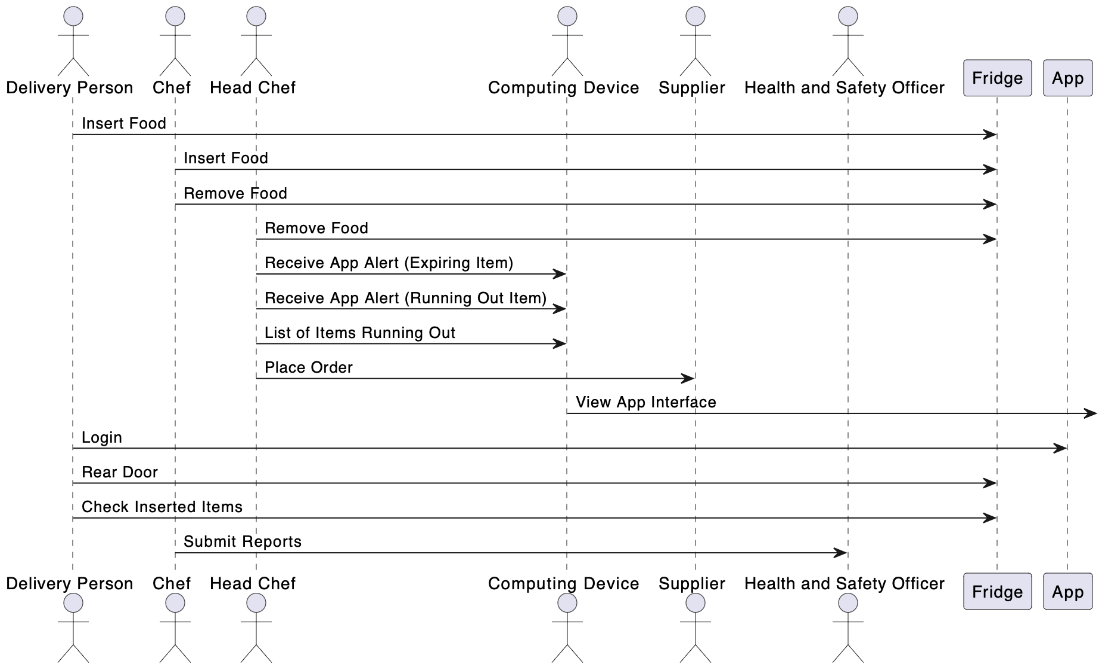
**7.5.2. Class diagram:**



The class diagram for the FFsmart fridge system shows the classes representing key entities and functionalities. The 'User' class represents individuals interacting with the system, distinguished by attributes like UserID, Username, and Role, with subclasses including 'Chef', 'HeadChef', and 'DeliveryPerson' reflecting distinct roles and responsibilities. 'Fridge' captures details such as ItemID, Name, and ExpiryDate. 'Alerts' handles system-generated alerts, and 'Delivery' handles replenishment orders. ‘Track’ handles all fridge activity and has the option to display it. The User class has a one-to-one relationship with the head chef, chef, delivery person and server whilst these classes also have a one-to-one relationship with the fridge. This is because each user can only be one user type and a user can only be connected to one fridge. Finally, the Track, Alerts, and Delivery classes have a one-to-many relationship with the fridge. This is because the fridge can get many alerts, track information and deliveries.

**7.6. UML diagrams:**

**7.6.1. Sequence diagram:**



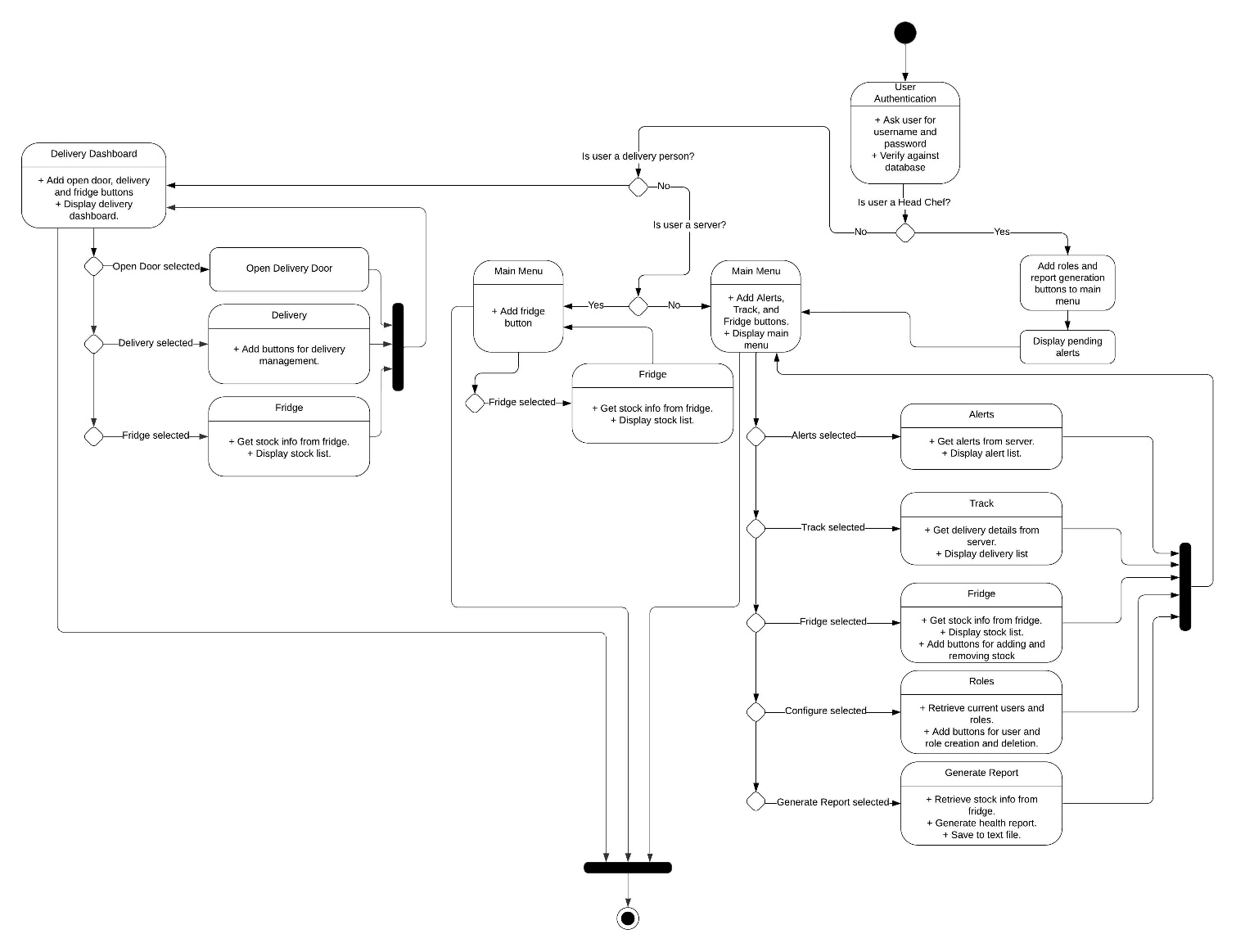
The sequence diagram provides a detailed insight into the operational flow of the FFsmart fridge system, showcasing the interactions between its key users and system functionalities. At its core are four primary actors: the System, responsible for automatic alerts, restocking, and reporting; the Delivery Person, tasked with stocking goods; the Head Chef, managing ordering and inventory; and the Chef, providing assistance in kitchen operations.

Across the diagram, the system's functionalities are depicted through a series of sequential events, highlighting the steps involved in executing various tasks within the FFsmart fridge system. These functionalities are organized into seven primary use cases: authenticating users, viewing inventory, updating inventory, creating reorder requests, generating expiration alerts, generating low stock alerts, and creating usage reports.

Each actor's interaction with these use cases is clearly illustrated, delineating their roles and responsibilities within the system. For instance, the System autonomously generates alerts, while the Delivery Person's interaction primarily revolves around authentication and updating inventory. Moreover, relationships between use cases are demonstrated, such as the "View Inventory" use case extending the "Update Inventory" use case, indicating a logical dependency between these functionalities.

Overall, the sequence diagram provides a comprehensive overview of the FFsmart fridge system's operational workflows, enabling stakeholders to understand the expected system capabilities and roles of each user. By organizing logically connected functions and depicting essential relationships, the diagram defines the scope of the system and represents its fundamental functional needs in an abstract manner. It serves as a visual description of primary user goals and system operations, facilitating quick comprehension and alignment among stakeholders involved in the system's design and implementation.

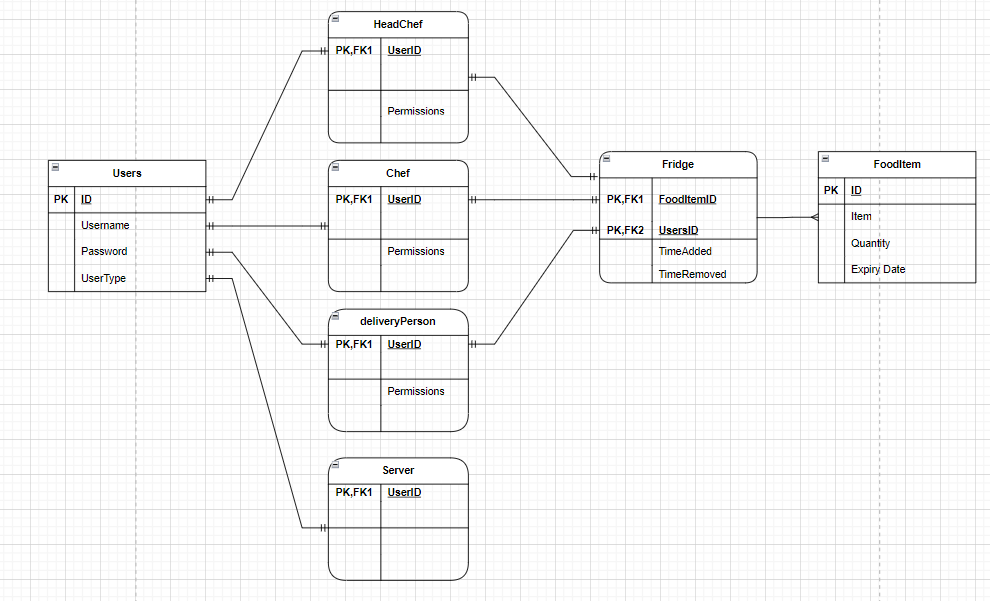
**7.6.2. Activity diagram:**



The activity diagram depicts the flow of the FFsmart application. The program begins with user authentication, and the rest of the program cannot be accessed until this is complete. There are 4 main user roles that dictate the access level of a logged in user: Head Chef, who has access to the entirety of the program’s functionality, Delivery Persons, who have access to the delivery dashboard, servers, who are only given access to the Fridge functionality, and other chefs who have all access bar report generation and role management.

Functionality access is determined through the user authentication method; the role of the logged in user is retrieved and the relevant menus are shown. The possible selections for each menu are demonstrated through the use of decision cascades.

**7.7. ERD diagrams:**



When a first enters the application they are prompted to sign up for an account. This will save the information in a database. The Users entity has 3 different user types which have a one-to-one relation to the user. Head chef which has their priority permissions and view. They can do actions that other user types cannot, for example, generate a safety report. The chef has permission to interact with the fridge unless the permissions are removed by the head chef. The delivery person can only insert it into the fridge. The Server has no access to the fridge. The user can add items to the fridge or remove items based on the permissions. The user types (head chef, chef, delivery person) have a one-to-one relationship with the fridge. This means each user account is connected to one fridge. Finally, the fridge and food items have a one-to-many relationship as many food items can be placed into the fridge.

**8.** **User Help Documentation:**

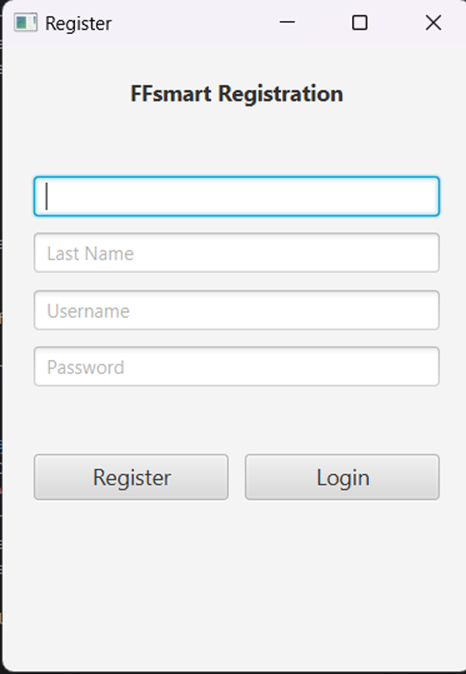
This is a guide to using the new FFSmart Fridge application. This guide will show news users how to access the services and each key feature of the application. Furthermore, the guide highlights the different features of each role in the application.

**How to install**

To use the application the latest Java development kit (JDK) must be installed, and JavaFX SDK's latest version must be installed. IntelliJ is recommended to run the program in development.

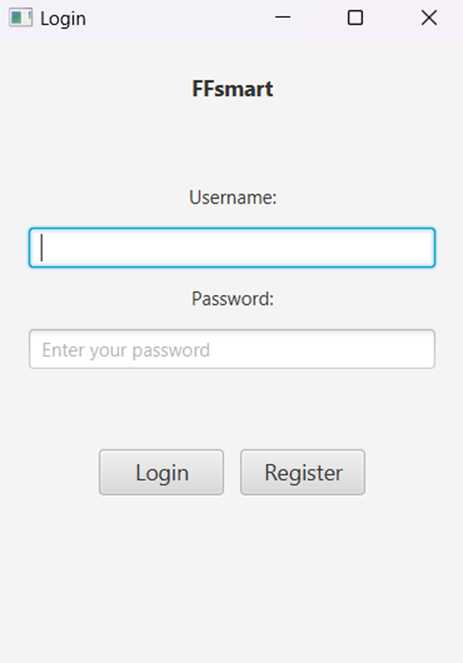
**Sign Up**

New users will be prompted to register. This uses a combination of a first name, last name, username, and password. Depicted in the figure below. The username must be unique, or an error alert will be displayed, and the user will not be able to register. Once has user has signed up, they will have to wait for the head chef to assign their role, e.g. Chef, Delivery person etc.



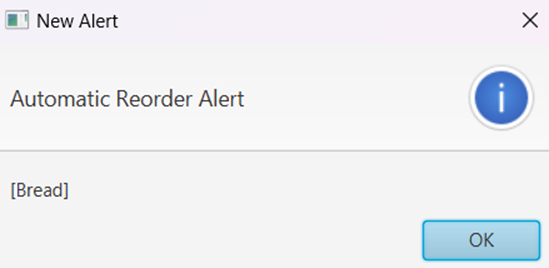
**Login**

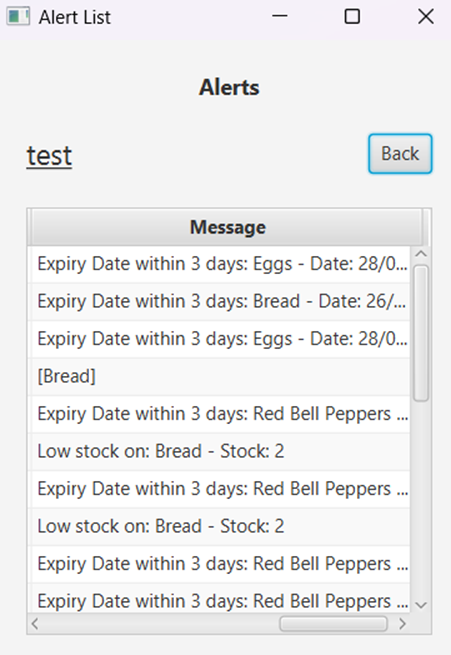
Once a user has registered and a role has been assigned, they will be allowed to log in. This is done by using the correct username and password combination. This will then send the user to the specific dashboard based on their user role.



**Alerts**

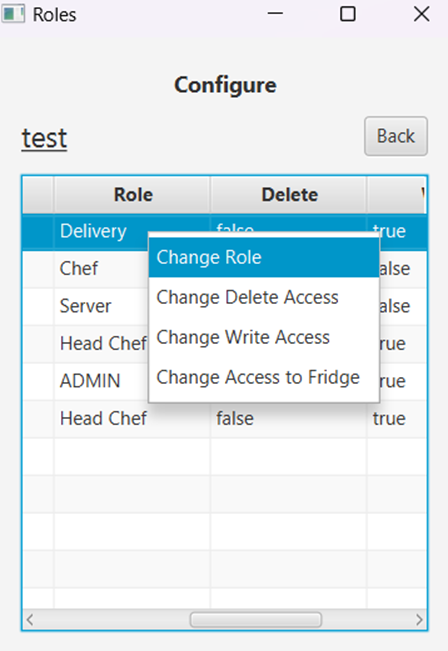
When a Head Chef has logged in, they may potentially be greeted with Alerts. These Alerts tell the Head Chef crucial information. For example, if food has been reordered, if food in the fridge is close to the expiration date, if food is low in stock, who has inserted/removed items in the fridge and if a delivery driver has not inserted the correct items into the fridge. These allow the head chef to act on the alerts. In the dashboard, there is a button which says ‘Alerts’ which allows the head chef to see all previous alerts.





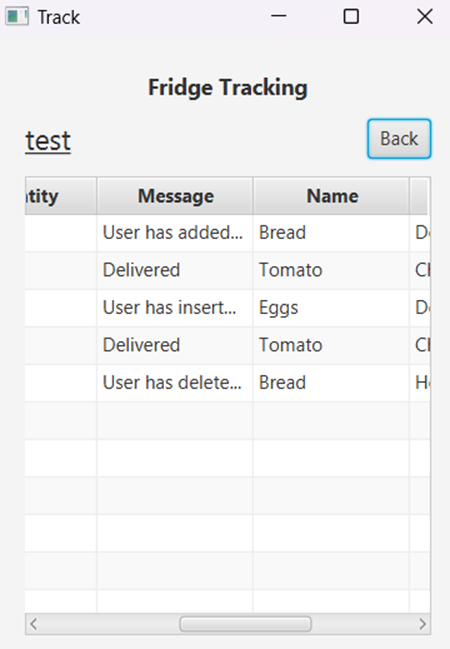
**Assign User Roles/ Edit User Permissions**

The Head Chef has the further ability to assign the user roles, as expressed in the sign-up section, and edit user’s permissions. There are 4 roles in total: delivery person, chef, server, and head chef. The Head chef role is the only role that can access this permissions page. The Head Chef can assign delete, insert, and access permissions to each user. This will limit an individual’s ability to add items to the fridge, remove items from the fridge, and even open the fridge.



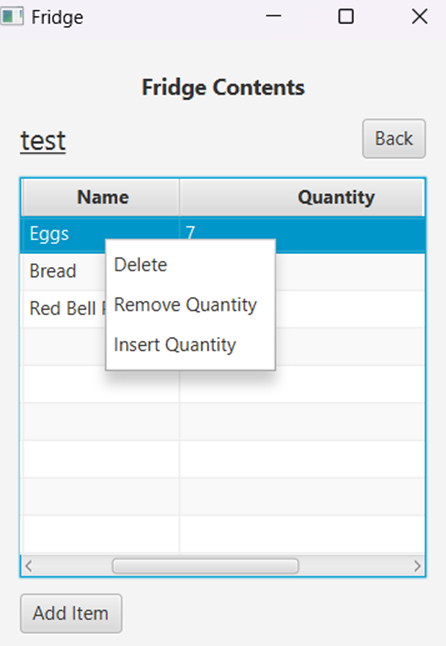
**Track Fridge contents**

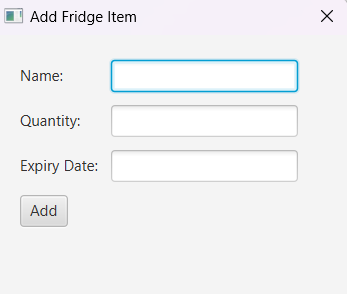
Users with the Access permission enabled can see the contents of the fridge by clicking the fridge button. This will show the item, quantity, and expiry date of the specific items. The Head Chef has more information shown in the Alerts section which shows who has inserted or removed items from the fridge. If the specific item is low in quantity, it will be automatically reordered on the coming Monday. If an item is within 3 days of the expiry date an alert will be sent to the head chef and it will have to be manually removed.



**Insert Remove/Item into Fridge**

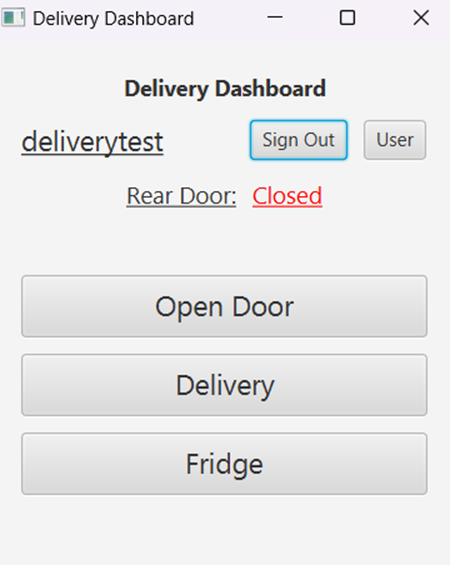
Users with delete and insert access will be able to insert and remove fridge contents. Deleting can be done by simply clicking the item needed and selecting the delete option, or by choosing the insert option and entering the item details.





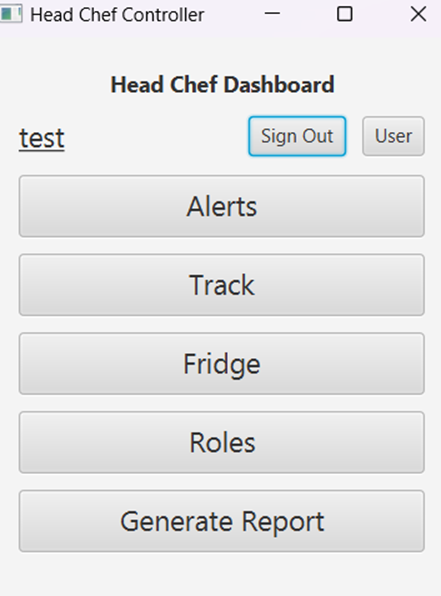
**Access Fridge**

To open the fridge door the access fridge button can be selected. This allows the users to open the rear of the fridge door.



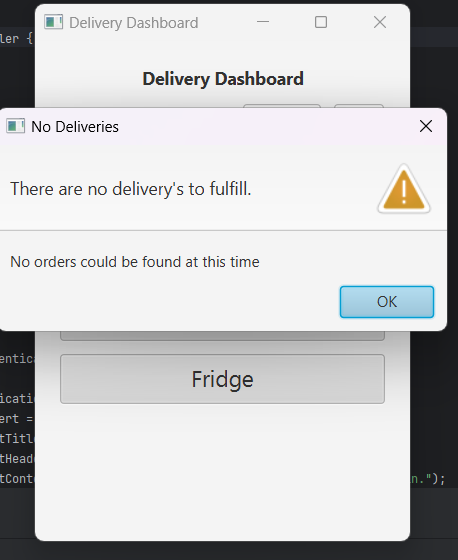
**Generate Report**

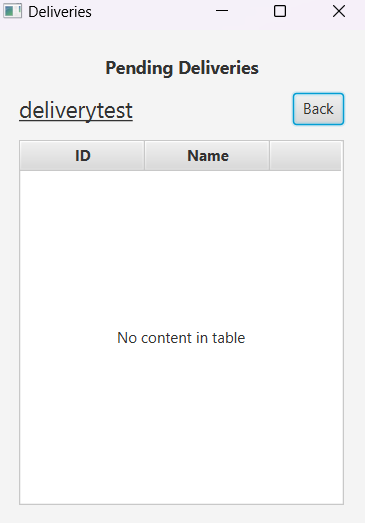
The Head Chef can generate a Health and Safety report for the Health and Safety officer. This is done by pressing the generate report button. This report contains important information about the information inside the fridge. This is generated into a text file and can then be printed or sent to the Officer via email.



**Delivery checking function**

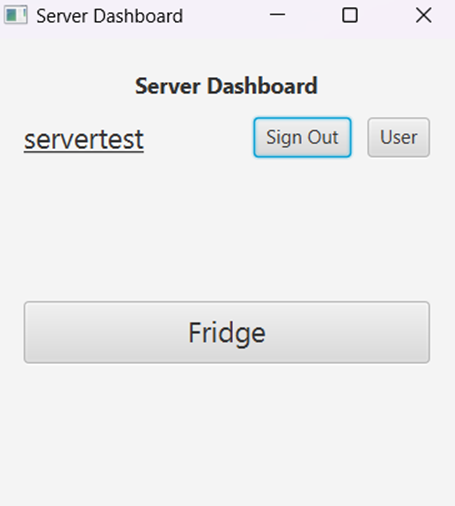
The checking function in the FFsmart fridge system serves the purpose of verifying pending deliveries and ensuring the accurate completion of deliveries. It examines incoming delivery records against scheduled deliveries to ascertain if any pending items remain. Additionally, it validates the delivered items against the expected items, confirming that the correct items were inserted into the fridge, thereby maintaining inventory accuracy and operational efficiency.





**Servers UI**

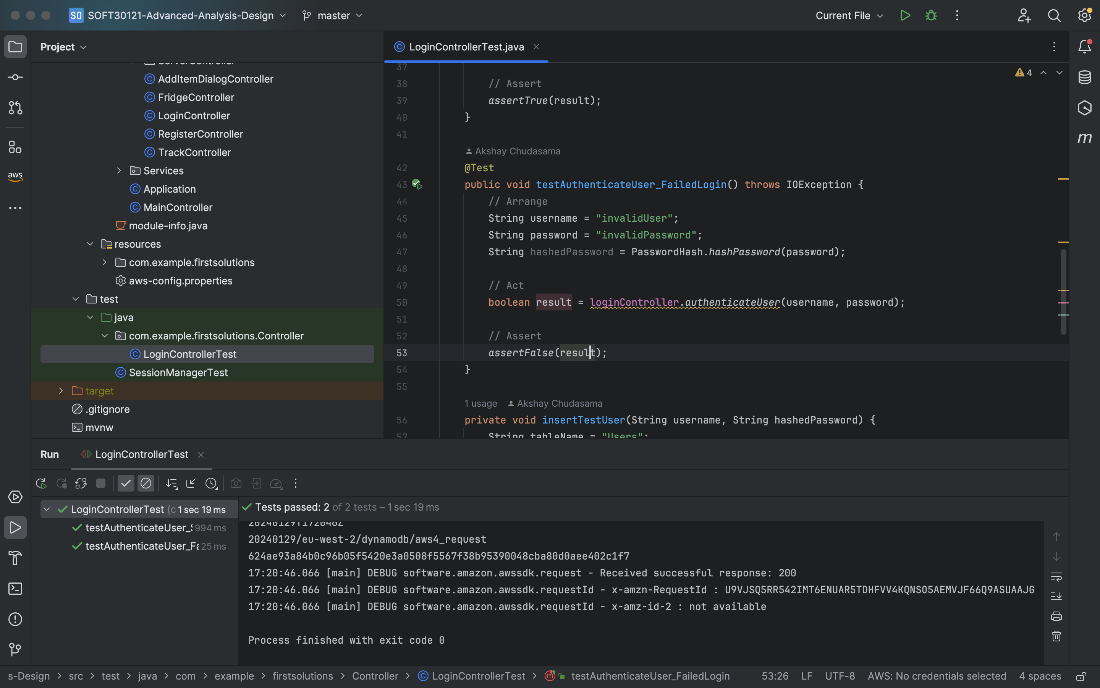
The servers can view the contents of the Fridge, but they are not able to interact with the fridge.



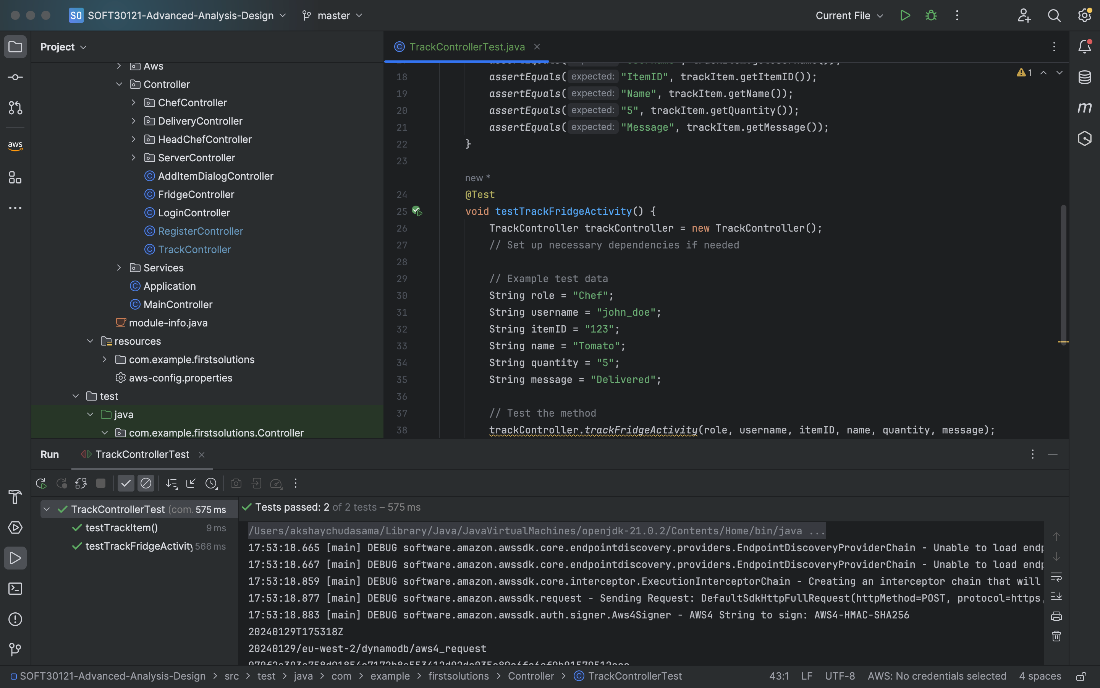
**9.** **Test Plan:**

**9.1. Unit tests:**

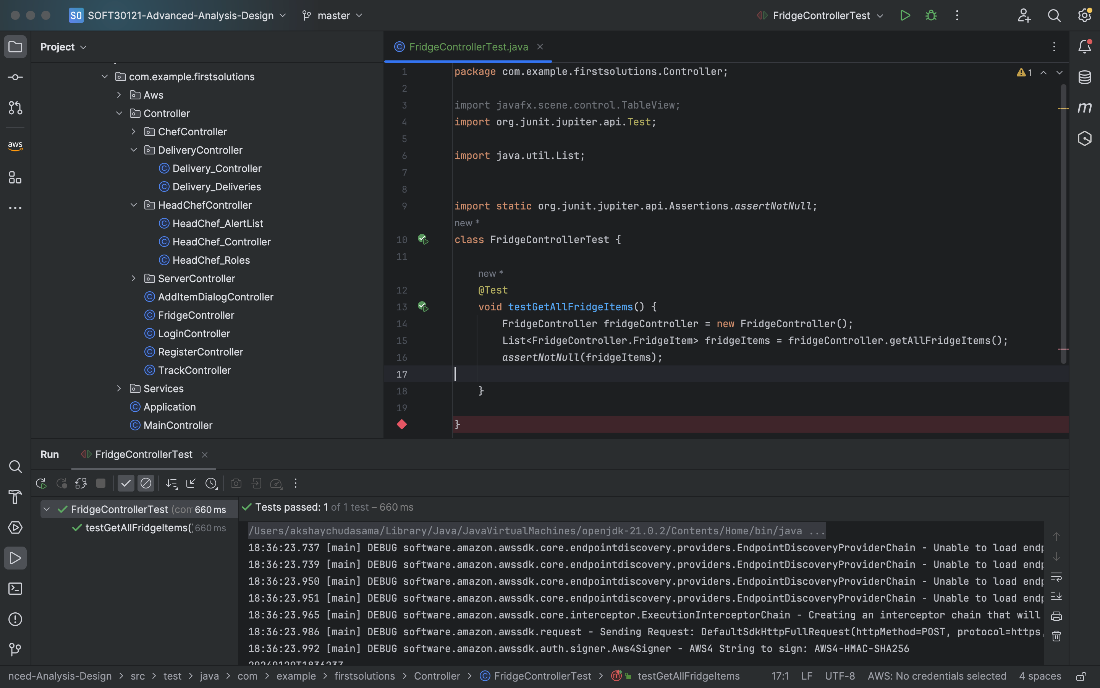
**LoginController Junit test:**



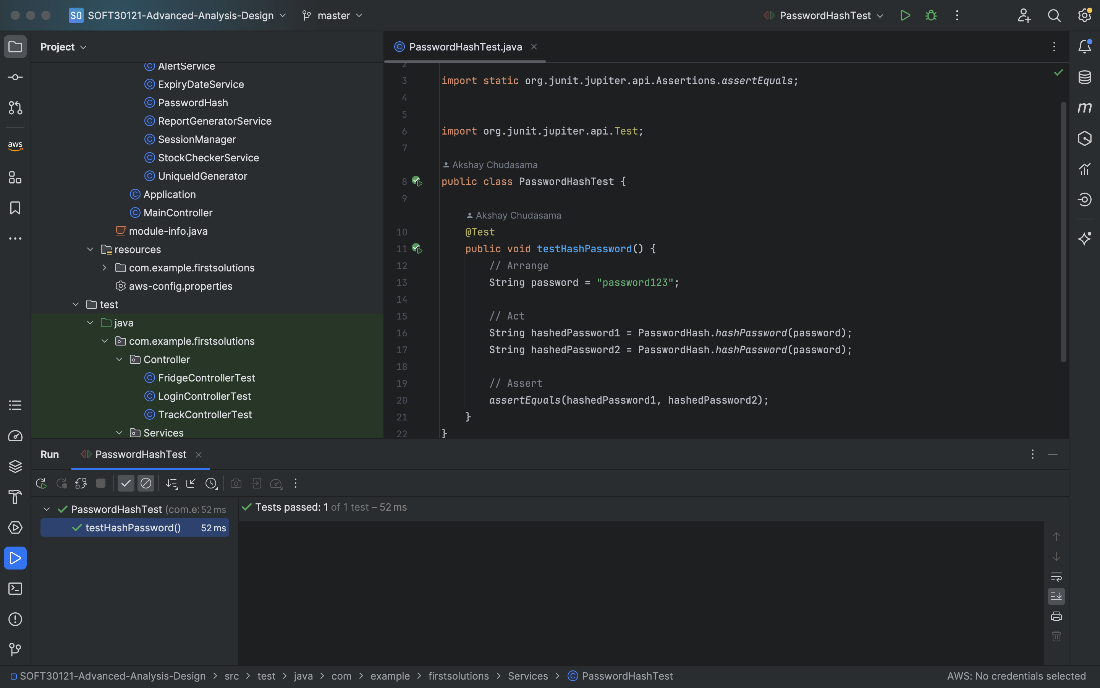
TrackController Junit test:



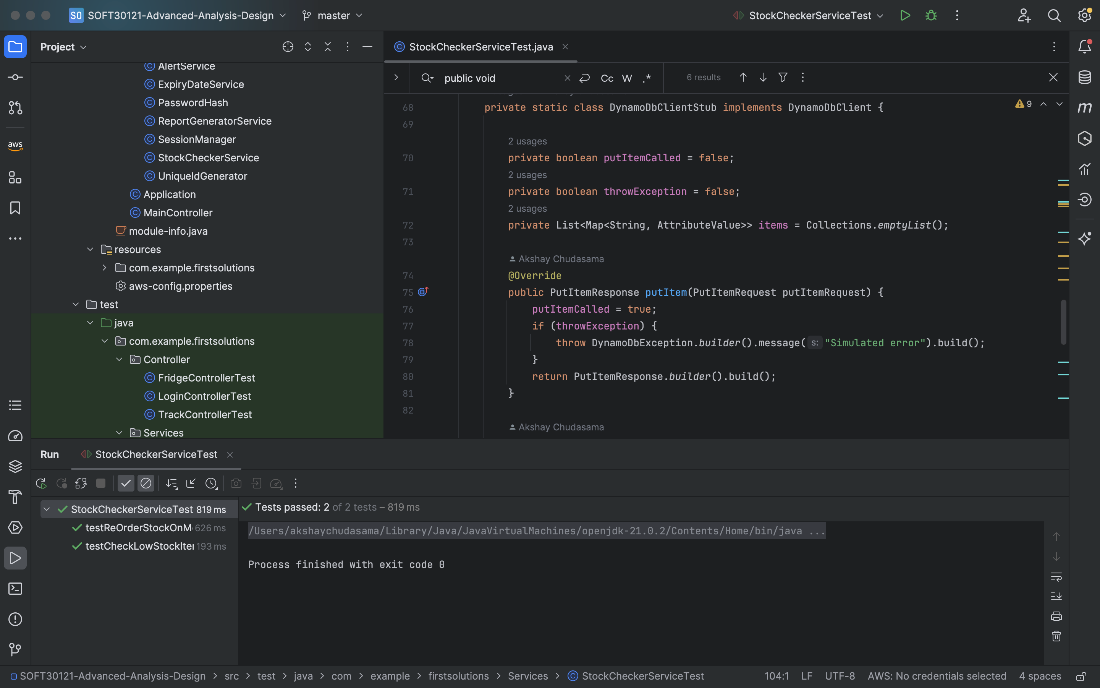
FridgeController Junit test :



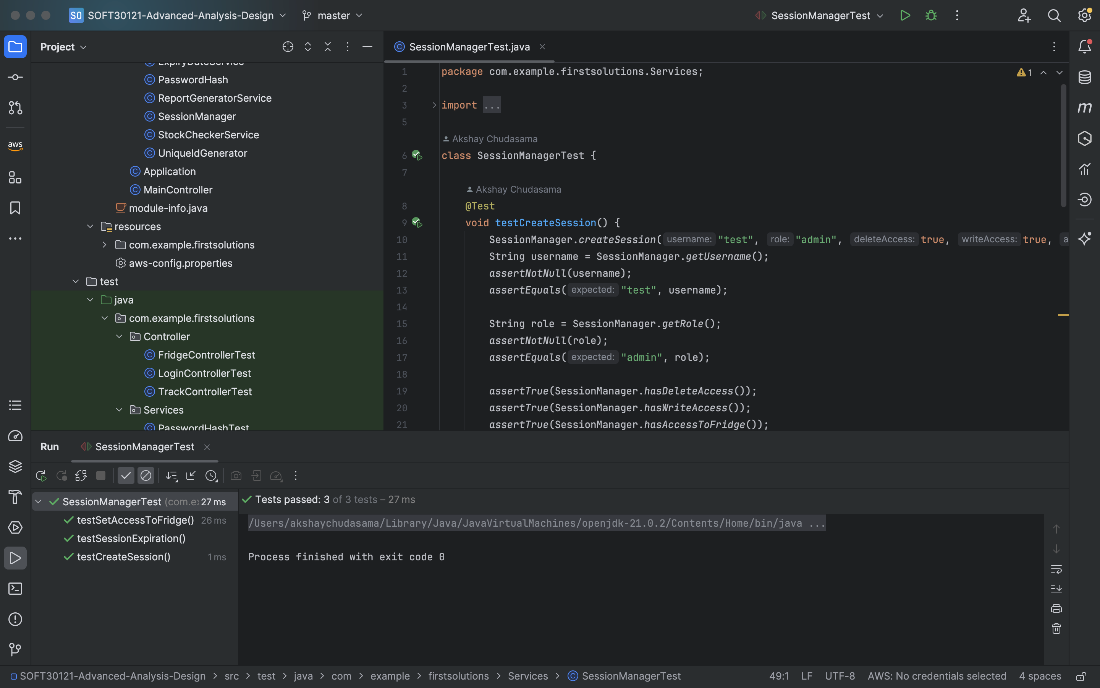
Passwordhashing Junit test :



Stockcheckerservice junit test:



Sessionmanger junit test :



**9.2. User Acceptance tests:**

|  |  |  |
| --- | --- | --- |
| **Functionality** | **Expected** | **Pass/Fail** |
| Automated and real-time tracking of food items inserted in, and removed from, the fridge | Automated and real-time tracking of food items inserted in and removed from the fridge involves developing a system that detects changes in the fridge's contents. This system collects data on item type, quantity, and timestamps, processing it in real-time to update inventory status. Users accessing features like viewing current contents, Alerts and managing inventory | Pass |
| A login system to track who has inserted/removed items from the fridge | Create somewhere to view what uers have removed or inserted in the firdge with a timestap of when they have removed the item or put an item in the fridge | Pass |
| Keeps track of the name, quantity and expiry dates of all items in the fridge and pushes app alerts to the computing devices of the head chef 3 days before the upcoming expiry of a given item | Tracking if a item is expiring and giving only the headchef the alerts on all devices and making sure e.g if the same item is in the fridge but has a later expiry only the one that is expiring is alerted to the headchef. | Pass |
| Alerts the same app for the head chef if items are about to run out of quantity 3 days before they do | Alerts for only the headchef to see what contents are expiring in the firge in the 3 days this can be done through a button or view only headchef can see or use | Pass |
| configure/change what users have access to the fridge and whether they are allowed to insert and remove items or not | Users will have a specific role that will restrict access to certain parts the app which then they can add or remove items. | Pass |
| An easy-to-use app interface display for each use | Interface that allows the app to work on any device such as tablets, laptops and mobiles | Pass |
| Reports to submit to the health and safety officer | Send a detailed report to submit to the the health and saftey officer which includes alerts of epiring foods items in the fridge what food laws we have to comply to when the report was generated. | Pass |
| Provides a list of items to the head chef that are close to running out in the fridge, and on each Monday auto-reorders the items which will/have run out | Headchef can add items to a list that needs to be reordered on a monday and add a specific item on a reorder which then when the item is running low can be automatically reordered | Pass |
| Food servers in the restaurant should not have access to the fridge at all | Fiid servers should not have the ability to open the fridge at all. | Pass |

**9.3. Test summary:**

**LoginController Junit test explanation** :

**Test Methods:**

The class contains two test methods annotated with **@Test**, each focusing on a specific aspect of the **authenticateUser** method:

**testAuthenticateUser\_SuccessfulLogin:**

**Arrange:**

Sets up test data, defining a valid username and password and hashing the password using the **PasswordHash.hashPassword** method.

**Act:**

Invokes the **authenticateUser** method of the **loginController** instance with the prepared test data.

**Assert:** Verifies that the method returns **true**, indicating a successful login attempt.

**testAuthenticateUser\_FailedLogin:**

* **Arrange:** Defines an invalid username and password and hashes the password.
* **Act:** Invokes the **authenticateUser** method with the invalid credentials.
* **Assert:** Verifies that the method returns **false**, indicating a failed login attempt.

**insertTestUser Method:**

The **insertTestUser** method is a helper method used in the tests to insert a test user into the DynamoDB table. It constructs an item map representing user attributes (e.g., "Username," "Password," "UserRole") and performs a **putItem** operation on the DynamoDB table using the **dynamoDbClient**. This method ensures that the DynamoDB table has the necessary test data for the login tests.

**Explanation of Passing the Test:**

* **Successful Login Test:**
* If the **testAuthenticateUser\_SuccessfulLogin** method passes, it indicates that the **authenticateUser** method in the **LoginController** class successfully authenticates a user with valid credentials. The test checks if the method returns **true** when given a correct username and password combination.
* **Failed Login Test:**
* Conversely, if the **testAuthenticateUser\_FailedLogin** method passes, it indicates that the **authenticateUser** method correctly rejects invalid credentials. The test checks if the method returns **false** when given an invalid username and password combination.

**Significance of the Tests:**

These tests are crucial for ensuring the correctness of the login authentication logic implemented in the **LoginController** class. They validate that the method functions as expected, allowing valid users to authenticate while rejecting invalid attempts. Additionally, passing these tests serves as a form of regression testing, ensuring that future modifications to the code do not inadvertently break the login functionality. It provides a reliable and automated means to verify the correctness of user authentication within the application.

**Trackcontroller Junit test explanation**:

**Test Methods:**

**testTrackItem:**

**Purpose:**Verify the correctness of the **TrackItem** class constructor and getter methods.

**Details:**

**Arrange:** Create an instance of **TrackItem** with specific property values.

**Act:** Use the getter methods (**getTrackID()**, **getRole()**, etc.) to retrieve the properties.

**Assert:**

Ensure that the retrieved values match the expected values.

This test ensures that the **TrackItem** class is properly initialized and its getters provide the expected results.

**testTrackFridgeActivity:**

**Purpose:**

* Check the behavior of the **trackFridgeActivity** method in the **TrackController** class.

**Details:**

**Arrange:**

* Create a **TrackController** instance.
* Set up any necessary dependencies if needed (not explicitly shown in the provided code).
* Define example test data (role, username, itemID, name, quantity, message).

**Act:**

* Call the **trackFridgeActivity** method with the example test data.

**Assert:**

* Currently, there are missing assertions that should be added based on the expected behavior of **trackFridgeActivity**.
* Assertions could include checks on state changes, return values, or any other relevant outcomes based on the logic inside **trackFridgeActivity**.

**Significance of the Tests:**

**testTrackItem:**

* Ensures the proper functioning of the **TrackItem** class.
* Validates that the class can be instantiated with specific values and that the getter methods correctly retrieve those values.
* Helps catch any issues with the initialization or accessor methods.

**testTrackFridgeActivity:**

* Validates the functionality of the **trackFridgeActivity** method in the **TrackController** class.
* Provides a way to test the method with example data to ensure it behaves as expected.
* The missing assertions need to be added to confirm the correctness of the method's outcomes.
* This test is essential for verifying that the **trackFridgeActivity** method works as intended and maintains expected behavior.

**FridgeController Junit test explanation** :

This test is designed to ensure that the **getAllFridgeItems** method in the **FridgeController** class functions correctly by retrieving a list of fridge items. Here's what happens during the test:

* **Arrange:**
  + An instance of **FridgeController** is created.
  + The **getAllFridgeItems** method is called to retrieve a list of fridge items.
* **Act:**
  + The test checks if the retrieved list is not null using **assertNotNull**.
* **Assert:**
  + If the test passes, it confirms that the **getAllFridgeItems** method is working as intended, providing a non-null list of fridge items.

**If the testGetAllFridgeItems Test Fails:**

* **Possible Scenarios:**
  + The **getAllFridgeItems** method might not be implemented correctly.
  + There could be an issue with the data retrieval or processing logic inside the **getAllFridgeItems** method.
  + The **FridgeController** might not be properly initialized or configured.
* **Implications:**
  + If the test fails, it indicates that the **getAllFridgeItems** method is not behaving as expected, and it might not be providing a valid list of fridge items.
  + The application's fridge-related functionality relying on this method could be compromised, leading to potential errors or inaccuracies in displaying fridge items.
* **Actions to Take:**
  + The developer needs to investigate the test failure, examining the implementation of the **getAllFridgeItems** method.
  + Potential issues could include incorrect data retrieval, processing errors, or problems with the initialization of the **FridgeController**.
  + Debugging and reviewing the relevant portions of the code will be essential to identify and address the root cause of the failure.

**Significance of Addressing Failures:**

* Resolving test failures is crucial for maintaining the reliability and functionality of the application.
* Fixing issues uncovered by the test ensures that the **FridgeController** can correctly retrieve and provide the list of fridge items, preventing potential errors or disruptions in the application's fridge-related features.
* Addressing failures promptly contributes to the overall stability and quality of the application.

**Passwordhashing junit explanation:**

### **Test Method:**

**testHashPassword:**

**Purpose:**

Verify that the **hashPassword** method consistently produces the same hash for a given input password.

**Details:**

**Arrange:** Set up the test by defining a sample password ("password123").

**Act:**

* Invoke the **hashPassword** method twice with the same input password.
* Capture the resulting hashed passwords (**hashedPassword1** and **hashedPassword2**).

**Assert:**Ensure that the two hashed passwords are equal.

This assertion confirms that the hash function is deterministic, meaning it produces the same output for the same input.

**Significance of the Test:**

**Deterministic Hashing:**

The test is checking if the **hashPassword** method is deterministic. In other words, when given the same input (password), it should always produce the same hashed output.

This property is crucial for password hashing in security applications. Users with the same password should have the same hash stored in the system.

**Consistency:**

Consistent hashing ensures that even if the **hashPassword** method is called multiple times with the same input, the resulting hash should remain unchanged.Consistency is vital for the integrity of password storage and verification systems.

**Sessionmanger test explanation:**

### Test Methods:

1. testCreateSession:

Purpose: Verify that the createSession method in SessionManager sets up a session with the correct attributes.

Details:

Arrange: Call createSession with specific parameters (e.g., username, role, access privileges).

Act:

Retrieve session information using SessionManager methods (getUsername(), getRole(), etc.).

Assert:

Check that the session attributes match the expected values. Ensure that access privileges are correctly set (delete access, write access, access to the fridge).

2. testSetAccessToFridge:

Purpose: Confirm that the setAccessToFridge method correctly updates the access to the fridge for a specific session.

Details:

Arrange: Create a session using createSession. Get the current session ID.

Act: Verify initial access to the fridge is true. Call setAccessToFridge to change access to false.

Assert: Ensure that access to the fridge has been successfully updated.

3. testSessionExpiration:

Purpose: Check if the session expiration functionality works as expected.Details:

Arrange: Create a session with a specified expiration state (false in this case). Get the current session ID.

Act: Verify that the session is initially not expired. Logout the user using logoutUser to mark the session as expired.

Assert: Ensure that the session is correctly marked as expired after the user logs out.Significance of the Tests:

Session Management:

Validates the correctness of the SessionManager in handling session creation, access control, and expiration.

Access Control:

Tests whether setting access privileges, specifically for the fridge in this case, works as intended.

Session Expiration:

Ensures that the session expiration mechanism correctly identifies when a session is expired after a user logs out.

**Stockmangement test explanation:**

### **Test Methods:**

**testReOrderStockOnMonday:**

**Purpose:**

Verify that stock is reordered automatically on a Monday.

**Arrange:**

Set up the test by creating a **StockCheckerService** with a **DynamoDbClientStub**.

Set the current date to a Monday (February 12, 2024).

**Act:**

Call the **reOrderStock** method with a list of items.

**Assert:**

Check that the **putItem** method of the DynamoDbClientStub was called.

Assert that the result of the reorder is true.

Verify that the console output contains the expected message.

**testCheckLowStockItemsWithNoLowStock:**

**Purpose:**

Verify that the **checkLowStockItems** method behaves correctly when there is no low stock.

**Details:**

**Arrange:**

Set up the test by creating a **StockCheckerService** with a **DynamoDbClientStub**.

Populate the DynamoDB stub with an item with sufficient quantity (7 in this case).

**Act:**

Call the **checkLowStockItems** method.

**Assert:**

Assert that the console output does not contain a message about low stock.

Ensure that the **putItem** method of the DynamoDbClientStub was not called.

### **Supporting Methods:**

**setUp:**

**Purpose:**

Initialize necessary objects before each test.

**Details:**

Create a **DynamoDbClientStub**, a **StockCheckerService** instance, and redirect console output to a **ByteArrayOutputStream**.

**assertConsoleOutputContains and assertConsoleOutputDoesNotContain:**

**Purpose:**

Helper methods to assert the presence or absence of specific messages in the console output.

**Details:**

Used to validate the content of the console output during test execution.

**createDynamoDbItem:**

**Purpose:**

Helper method to create a DynamoDB item with a specified item name and quantity.

**Details:**

Used to set up test data for the DynamoDbClientStub.

**DynamoDbClientStub:**

**Purpose:**

Implementation of a stub for the DynamoDB client to simulate interactions.

**Details:**

Keeps track of whether the **putItem** method was called.

Allows setting items to be returned during a scan.

Can simulate throwing an exception if needed.**Signifcance of the TestsAutomatic Reorder on Monday:**

Ensures that the **reOrderStock** method correctly triggers a reorder on Mondays.

Verifies that the correct message is output to the console.

**Low Stock Check:**

Validates the **checkLowStockItems** method, ensuring it correctly identifies low stock conditions.

Verifies the absence of unnecessary DynamoDB interactions when stock levels are sufficient.

**10.** **Project execution and management:**

**10.1. Team organisation:**

In organising our team to create the final product we kept the roles assigned in section 6. Jared being the Project Manager, ensures each member is contributing to the project’s success, overseeing the project’s trajectory resource allocation and timeline. Dami being the Lead Programmer, divides the project into multiple subtasks in our implementation phase and assigns tasks to individuals. Furthermore, checking code quality and making sure the GitHub repository is always deployable. Akshay took the responsibility of the Software Tester, he made sure our unit tests were all successful before allowing us to make new functionality. Finally, he made sure the program was fully working and that there were no bugs. Lastly, Max was our Analyst, he delved into the intricacies of the program and guided us with crucial insights. We used meetings to ensure everyone was on task and making progress. Without these meetings the project would not be as successful as it has been,

Meeting minutes:

Meeting 1 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Identify Primary roles * Complete Introduction of project * Complete Non-Functional Requirements * Complete Functional Requirements * Complete Interfaces section |
| Tasks completed | * Identify Primary roles * Complete Introduction of project * Complete Non-Functional Requirements * Complete Functional Requirements |
| Tasks incomplete | * Complete Interfaces section |

Meeting 2 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Complete Use case models * Complete Misuse cases * Complete Interfaces section * Complete Project Plan section * Have meeting with Client * Submit by 17th November |
| Tasks completed | * Complete Use case models * Complete Misuse cases * Complete Interfaces section * Complete Project Plan section * Have meeting with Client * Submit by 17th November |
| Tasks incomplete |  |

Meeting 3 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Architecture and design patterns * Error handling * coding standards and conventions * Auto-generated content * Complete ERD Diagram * Complete Class Diagram |
| Tasks completed | * Architecture and design patterns * Error handling * coding standards and conventions * Auto-generated content * Complete ERD Diagram * Complete Class Diagram |
| Tasks incomplete |  |

Meeting 4 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Component Diagram * Activity Diagram * Deployment Diagram * Sequence Diagram * Login Controller * Register Controller * Head Chef View and functions |
| Tasks completed | * Component Diagram * Deployment Diagram * Sequence Diagram * Login Controller * Register Controller * Head Chef View and functions |
| Tasks incomplete | * Activity Diagram |

Meeting 5 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Make Fridge Tracking system * Make Auto restock * Make Alerts system * Create chef view and functions * Create delivery person view and functions * Create server view * Activity Diagram |
| Tasks completed | * Make Fridge Tracking system * Make Auto restock * Make Alerts system * Create chef view and functions * Create delivery person view and functions * Create server view |
| Tasks incomplete | * Activity Diagram |

Meeting 6 Participants: Jared, Dammy, Akshay

|  |  |
| --- | --- |
| **Agenda** | **Action** |
| Tasks Identified | * Junit tests * Activity Diagram * Project execution and management * Final checks * Review |
| Tasks completed | * Junit tests * Activity Diagram * Project execution and management * Final checks * Review |
| Tasks incomplete |  |

**10.2. Methodology used:**

During our project, we opted for the agile methodology, specifically the Kanban framework. The Kanban framework emphasises continuous delivery and adaptive planning which resonated well with our project. The incremental nature of Kanban allowed us to prioritise tasks effectively and respond quickly to changes in requirements or priorities. Furthermore, the visualisation of tasks on the Kanban board allowed for transparency and easier communication within the team. This meant that using the agile Kanban board was very beneficial for our project.

However, while Kanban was very effective in managing our workflow, we encountered challenges in accurately estimating task dependencies and completion times. The lack of sprint boundaries meant that some tasks took much longer than expected and if other tasks depended on that particular task then progress was stagnant. This lead to minimal productivity at times as all we could do was wait. This resulted in increased time to complete the project.

Despite these limitations, the flexibility and responsiveness Kanban provides enabled us to address issues promptly and adapt our approach iteratively. In conclusion, Kanban effectively facilitated our project's development process by promoting transparency and adaptability, addressing challenges related to task estimation improvement in future projects.

**10.3. Software tools used:**

Original project plans and project goals will have to be changed to address the dynamics caused by uncertainty and to maximize project success (Dvir D, et al 2004). This is the case with many large and smaller projects. Throughout each phase of the software development lifecycle, our team remained committed to most of the tools decided in section 6, however, we did have an additional tool. Similarly, as we progressed through the project, particularly during the design phase, we encountered deviations.

Below in Figures 1 to Figure 7, shows all the tools used during the project. All of them correlate to the tools chosen in the Project Plan. They were the most beneficial tools to use as we have all had experience in each of these tools hence it will allow for seamless collaboration and increased productivity. However, we used an additional tool for creating diagrams, in addition to Lucidcharts we used Draw.io, Figure 4. Draw.io was beneficial to use as it helped us to draw diagrams specific diagrams with more detail as there is no shape limit like in LucidChart e.g. Class and ERD. Furthermore, we can create as many documents on Draw.io as we want to, unlike LucidChart where it was three editable maximum. This made Draw.io a needed tool in our development.

Initially, we crafted use cases, class diagrams and entity relation diagrams to conceptualise our system architecture. However, as we started coding in the implementation phase, we found ourselves changing these components. We did this so the diagrams could align with our evolving project implementation. This approach enabled us to maintain coherence between our initial designs and our current solution. This resulted in a refined end product.

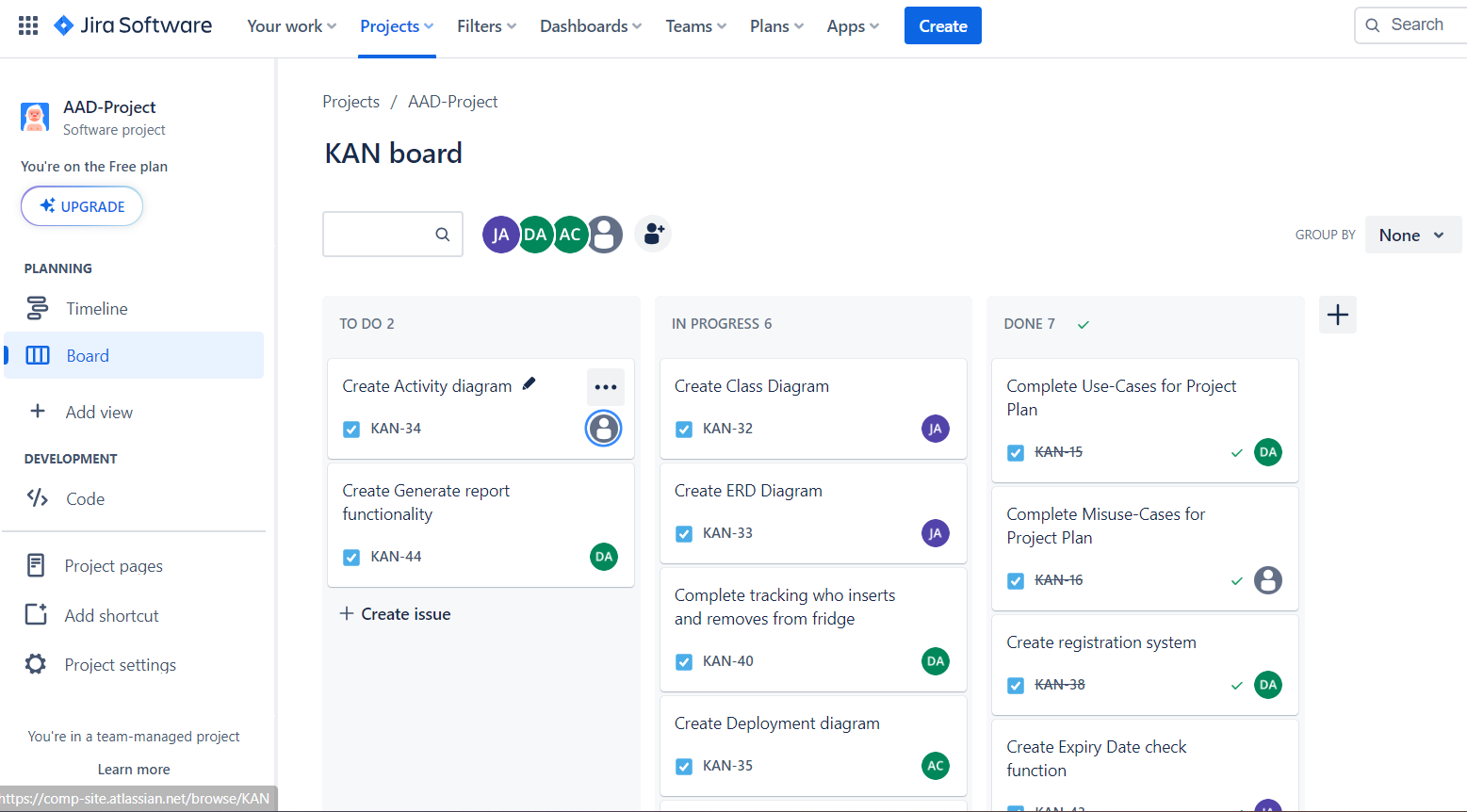


Figure 1: Jira Kanban board

We used Jira Kanban board to track our tasks as depicted in the Project Plan.

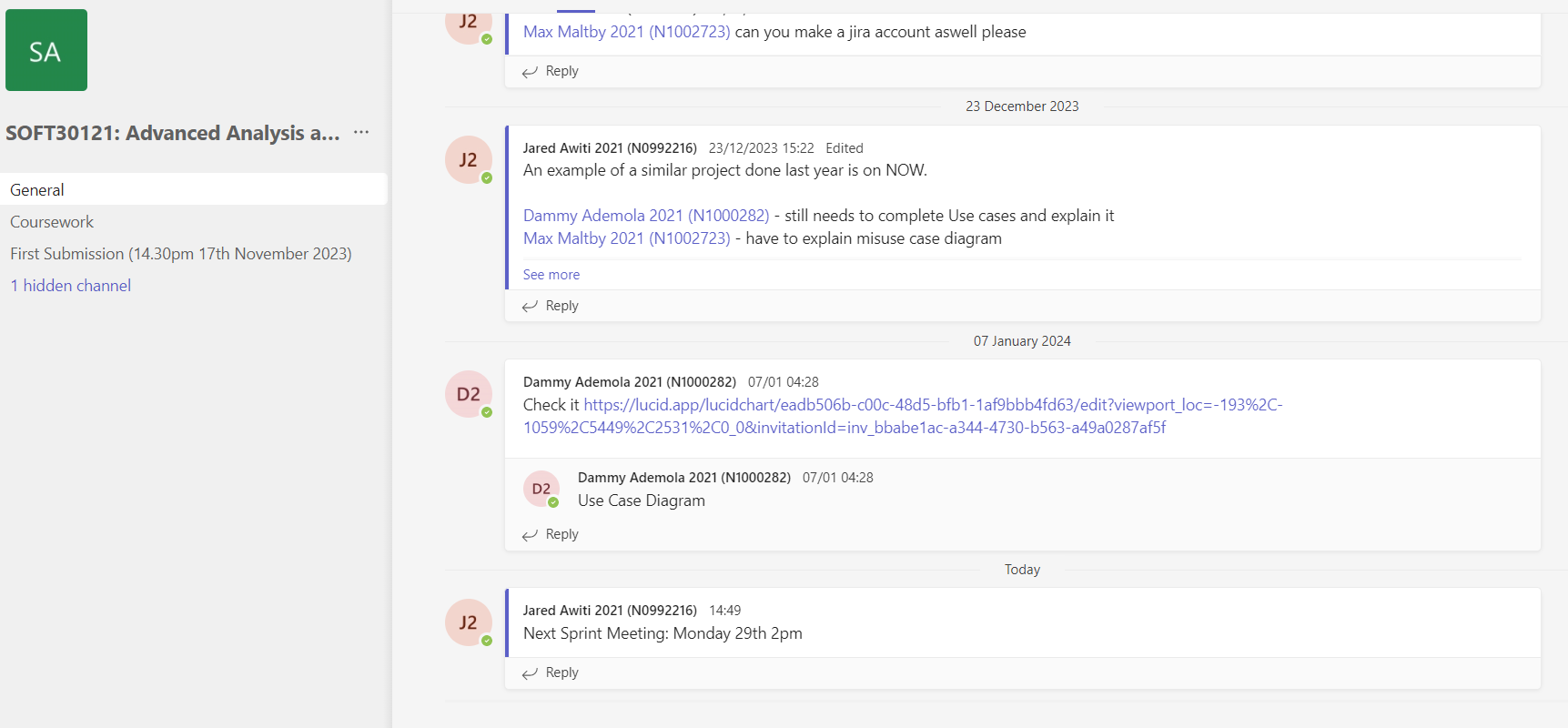


Figure 2: Teams communication channel

We used Teams to communicate and share files as shown in the Project Plan.

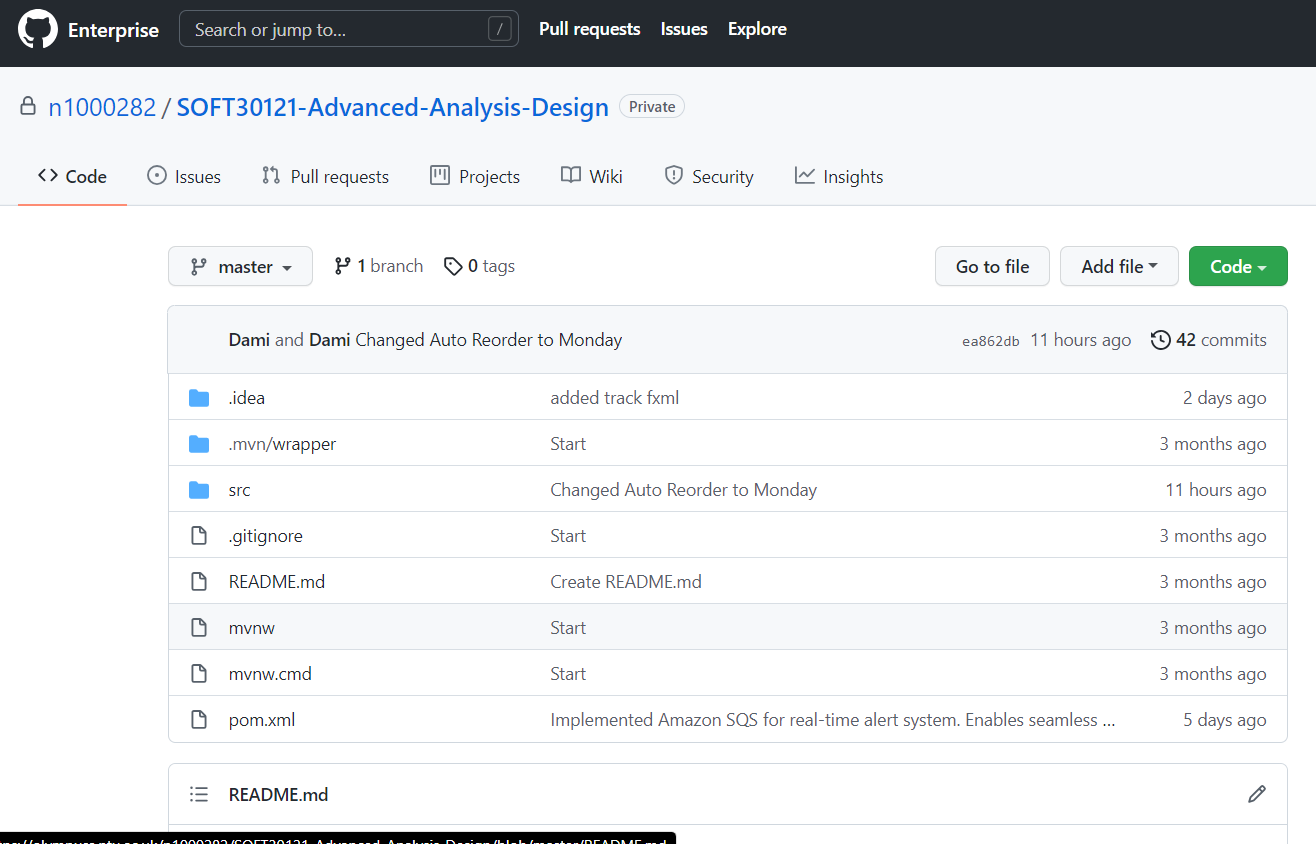


Figure 3: Github project

We used Github which is a version control software to collaboratively code and store our code as conveyed in the Project Plan.

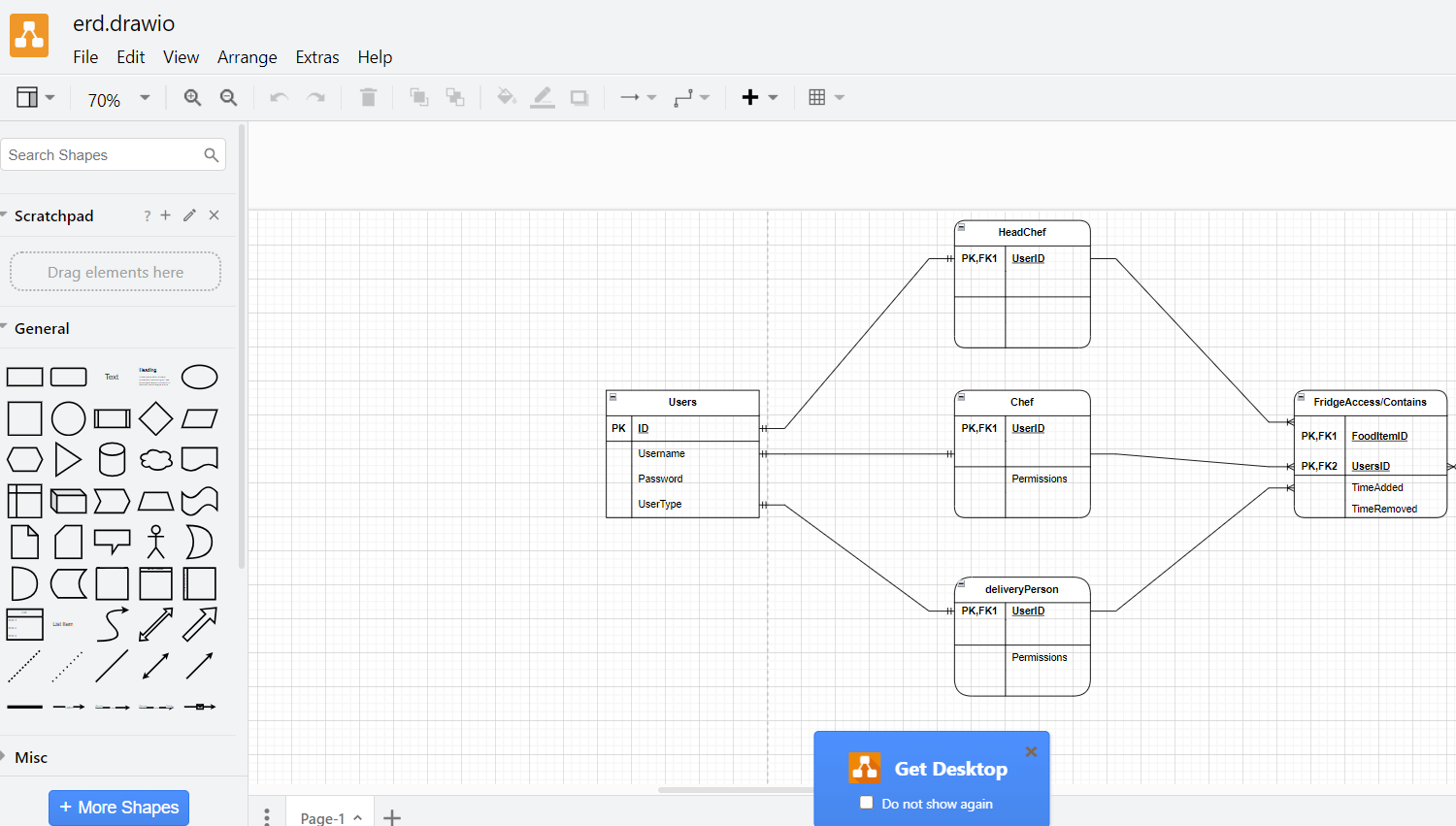


Figure 4: Draw.io

We used Draw.io to create some graphs.

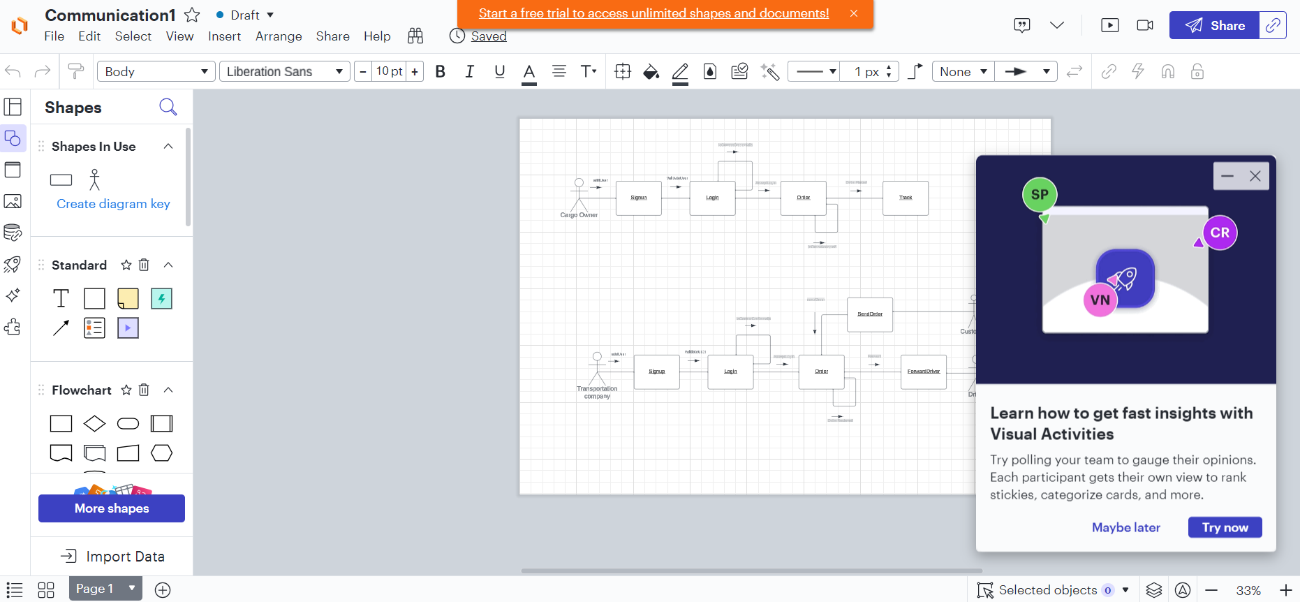


Figure 5: LucidChart

We used LucidChart to create some graphs as depicted in the Project Plan.

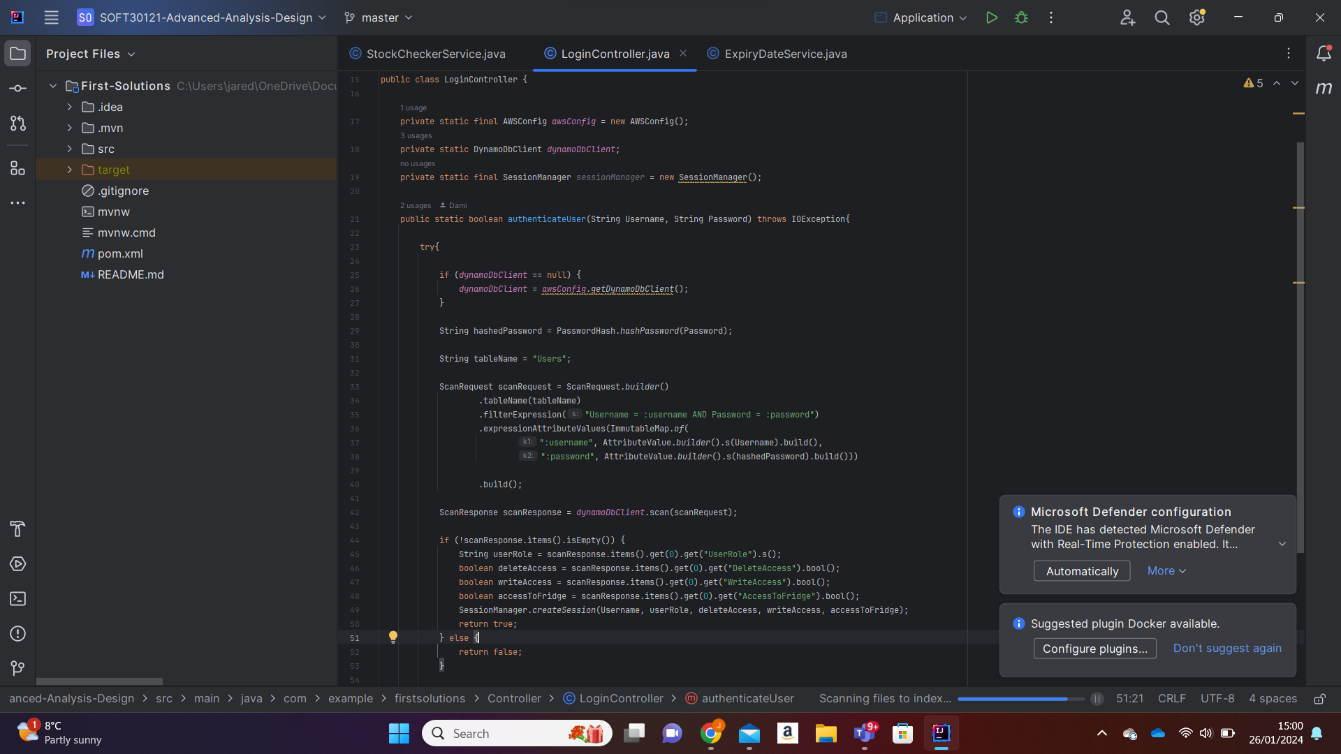


Figure 6: Intellij

We used IntelliJ to code as expressed in the Project Plan.

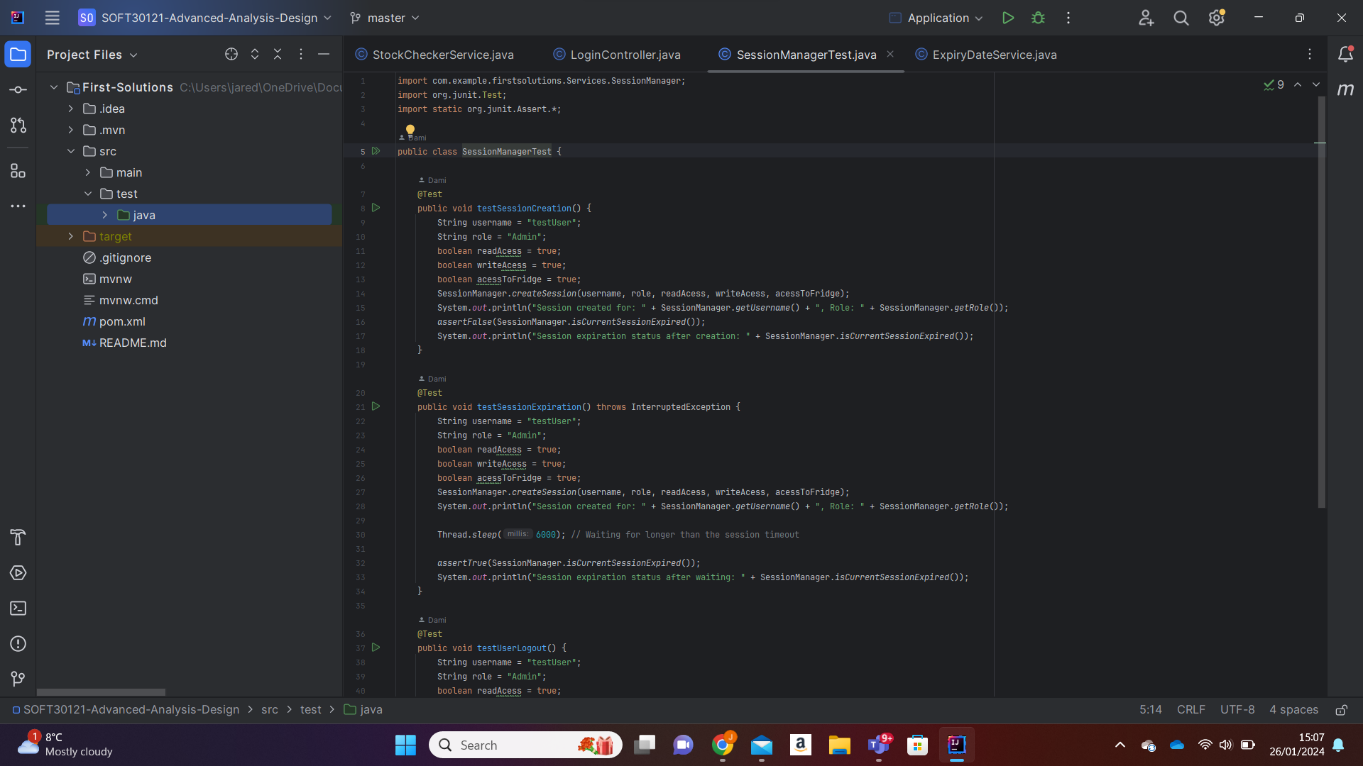


Figure 7: JUnit

We used JUnit to unit test the program as proclaimed in the Project Plan.

**10.4. Conclusion:**

Overall we believe this project was a huge sucess. We managed to create a user-friendly application for the FFSmart Fridge within the alloted time. All requirement were achieved and we truly believe taht this application will help all restraunts with the FFSmart fridge become more productive and safer. Which will result is a more profitable resturant.

**11.** **References:**

Adenowo, A.A. and Adenowo, B.A., 2013. Software engineering methodologies: a review of the waterfall model and object-oriented approach. *International Journal of Scientific & Engineering Research*, *4*(7), pp.427-434.

Cohen, D., Lindvall, M. and Costa, P., 2004. An introduction to agile methods. *Adv. Comput.*, *62*(03), pp.1-66.

Cabral, B. and Marques, P., 2006, November. Making Exception Handling Work. In *HotDep*.

Dvir, D. and Lechler, T., 2004. Plans are nothing, changing plans is everything: the impact of changes on project success. *Research policy*, *33*(1), pp.1-15.

Gaba, I. (2023) *Agile vs waterfall : Which methodology is right for you?*, *Simplilearn.com*. Available at: <https://www.simplilearn.com/tutorials/agile-scrum-tutorial/agile-vs-waterfall> (Accessed: 24 October 2023).

Good, L. (2023) *Pros and cons of Jira Software for Project Management (2023)*, *project*. Available at: <https://project-management.com/the-pros-and-cons-of-using-jira-software/> (Accessed: 30 October 2023).

Karrenbauer, J., Wiesche, M. and Krcmar, H., 2019. Understanding the benefits of agile software development in regulated environments.

Spinelli, J. (2019) *MVC(model, view, controller):*, *Medium*. Available at: <https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5> (Accessed: 23 January 2024).

**Appendix A: Contribution**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Primary Role** | **Areas worked on** | **Contribution rating** |
| Jared Awiti | Project Manager | Project Plan, Functional requirements, non-functional requirements, ERD Diagram, Class diagram, User acceptance testing, Project execution and management, User help documentation, stock checker system, expiry date system | 4 |
| Akshay Chudasama | Software tester | User Interface, Deployment diagram, Component diagram, Sequence diagram, Tracking system, Unit testing, User Acceptance testing, Interview | 4 |
| Dammy Ademola | Software developer | Use case, Introduction, Unit testing, login system, register system, Tracking system, Auto reorder system, sessions controller, delivery function, checking function. | 4 |
| Max Maltby | Analyst | Misuse cases,  Activity diagram | 2 |

Name Signed by:

|  |  |
| --- | --- |
| **Name** | **Signed by** |
| Jared Awiti | Jared Awiti |
| Akshay Chudasama | Akshay Chudasama |
| Dammy Ademola | Dammy Ademola |
| Max Maltby | Max Maltby |