Problem Specification

'The core concept of this system is to mitigate the environmental impact by reducing food waste, aligning with sustainable practices. Stakeholders were identified through iterative interviews yielding key insights into user requirements'

Elicitation Process: Interviews:

The interview process targets individuals aged 15-80, with a focus on those aged 18-28 due to regional expectations. Short interviews were conducted for easy trend identification. About 70% of interviewees were young individuals due to our target market for the application (18-40). Gender representation was balanced, and the elicitation process involved two interview phases and a focus group. Summarised notes and interviewee classifications were documented after each interaction, and consent forms were signed by all participants.

Elicitation Phase: Data Collection:

Number	Gender	Age	Experience (Food Waste + Cooking)	Identified Features
1	Male	20	Does not use some ingredients due to being unaware of how to use them.	Top 10 Recipes, Add Own Recipes
2	Male	19	Generally quite good at using all ingredients in the kitchen.	Recipe generation.
3	Female	23	Throws away ingredients every so often due to overuse of the same ingredients.	Login Page, Creating an Account.
4	Female	20	Uses the majority of ingredients in the kitchen until out of use.	Recipe Generator
5	Female	19	Abundance of unused ingredients in the kitchen due to unsure what recipes use them.	About-Us Page
6	Male	24	Would like to improve cooking skills but is unaware which recipes to use and how to reduce their individual food waste.	Apple-Sign In, Google Sign-In. Food Notification.
7	Female	22	Well aware of how to minimise food waste through preparing small meal prep recipes but willing to improve their cooking skills	User profiles.
8	Male	38	Tends to throw away a couple of items contributing to food waste each week. Has experience of cooking food.	FAQs, Rating Recipes
9	Female	32	May throw food items away when expired as unaware how to add them into recipes with limited ingredients. Has limited cooking experience.	Add Own Recipes, Food Notifications
10	Male	39	Very good with limiting food waste and tends to use up all ingredients before consumer data. Has very high experience with cooking.	AI Help Chatbot

Phase One: Interview Structure:

Participants were recruited in person, and some follow-ups were conducted via phone call to arrange meetings. During interviews, the system's purpose and functionality were explained, accompanied by initial UI designs presented through phone visuals and team sketches. Standardised questions, including open and closed formats, were used to ensure data consistency.

Interview Area:

In the following image is the geographical area from which participants were sourced for gathering impressions on the group discussed application and to develop functional and non-functional requirements. Participants were approached and invited for participation in the interview for the desired application, employing a volunteer sampling method. This approach was selected for its cost-effectiveness in recruiting participants. To mitigate researcher bias, the abundance of phone numbers and participant information was given a number, randomised, and selected using a random number generator. This ensured the acquisition of a representative sample from the population. The number on the image represents the number of participants gathered in the designated region.

6
3
1
PORTSMOUTH CITY CENTRE
PORTSEA
FRATTON
SOMERS TOWN
SOMERSTOWN
SOUTHSEA
GUNWHARF QUAY
OLD PORTSMOUTH

Participants who scheduled a meeting were invited to join the study and presented with initial UI designs and ideas to gather first impressions of the system. The interview comprised a mix of open and close questions, along with follow-up questions. Key questions included:

1) **What features are important to be included in a recipe builder application to counteract food waste? Ingredients of food, use by dates, how many meals you can make from ingredients, prioritise mass ingredients**

2) Should the application be minimalistic and the approach should be consistent throughout the design?

3) Should users be able to have their own profile?
   - If so, what areas of control should they have on their profile data?

5) From a user point of view, which functionality features should be added for navigation?

6) Are there any features that should not be included, in relation to a recipe builder?
   - Such as, appropriate colours for users with visual impairments or navigation pages?

7) **Are there any safety features that should be installed in relation to food restrictions or expired foods?**
   - **If so, how should the application make the user aware of these issues?**

8) In terms of handling personal information, how should the application handle this information e.g., through encryption or preventing data breaches?
   - What data should be classes as voluntary in this application?

9) **What special features should be included in the application, as a way of tackling inclusivity?**

10) **Any questions or feature suggestions to be considered for the application?**

The list above covers UI features and important safety considerations discussed in the interview through these questions. The last question is designed to give participants an opportunity to discuss areas not covered explicitly, providing a more comprehensive understanding of the user requirements.

Interview Examination, Analysis, and Next Steps:

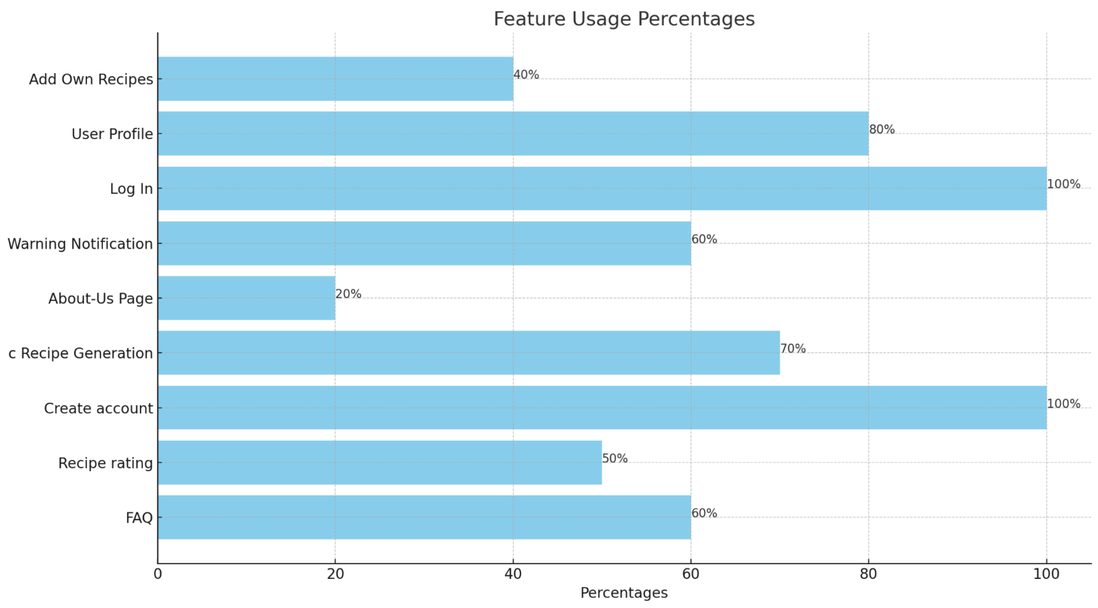
Audience feedback on the envisioned application functionality and initially concept was positive. Suggestions for potential functionalities and navigation enhancements, such as notifications for expired food items, user profiles, and user-contributed recipes, were proposed during interviews (refer to data collection table). While these recommendations formed a strong foundation for understanding user requirements, a challenge arose in consolidating feedback and producing an overview of the initial system ideas. Limited comprehensive feedback on newly introduced features prompted the initiation of focus groups as a strategic approach to address these issues.

Final Elicitation Phase: Focus Groups:

In this phase, follow-up studies were conducted with previous interviewees to gather feedback on newly collected system requirements and guide future development. Contact was made via phone, and Zoom calls were arranged for discussion on data and UI designs, using a similar set of questions and structure as outlined previously. The average duration of each Zoom call ranged from 15-20 minutes. However, 2 out of 10 previous interviewees did not respond due to our follow-up study.

During Zoom calls, computer-based sketches of the design were presented, accompanied by detailed summaries of each feature suggestion. Participants provided initial feedback on integration preferences and shared views on benefits and drawbacks, considering factors like time consumption and potential confusion. To conclude, close questions were used for a comprehensive analysis, such as 'Is this feature necessary for the system>' Participants were prompted to sign consent forms at each phase, and they were given the option to participate in testing during the implementation phase, contingent upon their consent.

Elicitation Analysis and Results:



After the elicitation process, the collected Zoom call data underwent summarised analysis. Features with over 50% participant support were considered fundamental, while those below 50% were deemed less crucial, with potential consideration for future iterations. Subsections of functional and non-functional requirements present statements gathered from the elicitation phase. Consequences from all Zoom calls interviews collectively describe opinions, including benefits and drawbacks. Due to time constraint, implementation within the team's timeframe appeared unlikely. Consequently, a prioritised hierarchy of requirements was established based on the percentage of participants in the focus group study supporting each feature (stated in the graph above).

To aid feature hierarchy development, a comprehensive risk assessment study was conducted for each feature. This involved analysing potential challenges, dependencies and uncertainties that could impact the project. Documented risks influenced downgrading some fundamental requirements to less critical and excluding the 'AI chatbot' feature. This process contributed to the final selection of requirements for the WasteAway application.

Functional Requirements:

User requirements have been translated into system requirements. For traceability, each requirement references the original data collection table, indicating the participant ID who recommended it. This ensures clarity about the origin of each feature, emphasising its identification as a user requirement. Risk assessment and consequences were considered, although not explicitly shown in the hierarchy.

UR: User Requirement
SR: System Requirement
Red: Fundamental functional system requirement
Blue: Less crucial functional system requirement

1) *UR: **User Authentication Control**: The user must be able to log into their account **(Ref 3,6)***
    a) FR: Ability to trigger authentication: system shall provide functionality to initiate the user authentication process upon receiving a valid request.
    b) FR: Ability to terminate authentication: system should terminate an ongoing authentication process, providing an immediate halt to user access validation.
    c) FR : Ability to resume authentication after a halt: system must support the ability to resume the process, allowing users to complete the authentication.

d) FR: System must take user input; email and password and compare against log-in records in Firebase database.
e) FR: System must validate information. All log-in information must be provided by the user to log-in to the system.
f) SR: System validates user information and forwards the user to the homepage (recipe generation screen). If invalidates data then an appropriate error message is elicited back to the client interface.

*2) UR: **User Account Creation Control**: The user must be able to create an account **(Ref 3)***

a) FR: Ability to initiate account creation, upon receiving valid user details.
b) FR: Ability of temporary suspension of an ongoing account creation process, providing users with options to resume later.
c) FR: Ability of system to resume the process after suspension, allowing users to complete the account creation.
d) FR: System must take user input email, password and validate information accordingly.
e) FR: System must check email is not duplicated in the Firebase database (as this is considered as unique information). Furthermore, that the email is actually a valid email e.g., contains '@' '.com'.
f) FR: System must ensure password length is above 6 characters and contains at least 1 symbol.
g) SR: System should store username and password into the backend database.

*3) UR: **User Profile**: The user must be able to create an account **(Ref 7)***

a) FR: Ability to create and manage a profile with personal information and preferences.
b) FR: Ability to provide secure authenticate and authorise mechanisms to safeguard user profile.
c) FR: System must include a module for users to create and manage their profiles, allowing them to input and update personal information and preferences.
d) FR: System must validate user inputs; user bio should be less than 200 characters and data type should be a string, food restriction should be a string and should be a valid food restriction defined by a list of food restrictions data structure.
e) FR: System should store user inputs (user inputs; user bio, food restriction, location, and bio) into the backend.

*4) UR: **Dynamic Recipe Generation**: Users must be able to generate top 10 recipes based on selected ingredients and food restrictions. **(Ref 1,4)***

a) FR: Ability to dynamically generate a list of top 10 recipes based on user food restrictions and selected ingredients.
b) FR: Ability to take user input; ingredient name, ingredient expiry date to generate recipes.
c) FR: Ability to allow users to select ingredients for recipe generation.
d) FR: System must ensure the user has selected at least 5 valid ingredients and type of ingredients to produce an appropriate recipe.
e) FR: System must consist of the ability to validate ingredient names, and ensure that expiry date is not passed the current date time.
f) FR: System must integrate a personalisation engine that tailors recipe recommendations to each user's unique food restrictions.
g) FR: System must validate whether ingredients are valid to use in the feature e.g., if an ingredient is expired, shouldn't use this ingredient in the recipe generation **(5)**.

*5) UR: **Food Warning Notifications**: User must be notified on expired food items that have been selected for Recipe Generation **(Ref 6,9)***

a) FR: Ability to provide notifications about expired food items that have been selected.
b) FR: System must consist of a method to analyse recipe ingredients and user profile to identify and notify potential allergens if the recipe generation algorithm is no longer efficient.
c) FR: System must consist of a module that identifies ingredients that have expired from ingredients attached to their profile based on the current date and time and notify the user of this issue.

*6) UR: **About-Us Page**: Users should have access to company-information. **(Ref 5)***

a) FR: Ability to display relevant information about the platform and mission, along with the team behind the idea.
b) FR: Ability to include features such as team member profiles or contact forms to enhance user engagement.
c) FR: System must consist of a module that displays the team's mission, and contact details.
d) FR: System must manage team members profiles, allowing for easy updates and additions. E.g., bio, messages.
e) FR: System must ensure bio is less than 200 characters long and is a string data type. System must ensure messages to users are 300 characters or less and is a string data type.

f) FR: System must integrate a contact form module that enables users to interact with the team and submit inquiries directly. Contact form consists of a link to an email so that users can reach out to team members.

7) *UR: **Recipe Rating**: User should be able to rate recipes on the system and provide viable feedback **(Ref 8)***
   a) FR: Ability to rate recipes on a scale or provide viable feedback.
   b) FR: System must implement a user-friendly mechanism for users to input ratings and feedback for recipes.
   c) FR: System must ensure recipe ratings consist of less than a 200 character review and is a string data type.
   d) FR: System must develop a module that calculates aggregate ratings (out_of_five) for each recipe based on the ratings provided by users.

8) *UR: **Add Own Recipes**: User should be able to enter their own recipes to the system **(Ref 1,9)***
   a) FR: Ability to review and approve user-submit recipes to ensure quality and appropriateness.
   b) FR: System must implement a user-friendly form or module that allows users to easily submit their own recipes which must be validated.
   c) FR: System must consist of a validation system for approving user-submitted recipes (recipe name, list of ingredients, approximation of cooking time, cooking method), including tools for content moderation and quality assessment.

9) *UR: **FAQ Page**: User should have the ability to submit and view FAQ results or queries **(Ref 8)***
   a) FR: Ability to organise common user queries and concerns.
   b) FR: Ability to search and quickly find answers to specific questions.
   c) FR: Ability to add an FAQ.
   d) FR: Ability to allow administrators to easily add, edit, or remove FAQ entries.
   e) FR: System must develop a module that organises common user queries and concerns in a structured format.
   f) FR: System must ensure FAQ query (query_question, query_content) submitted are less than 200 characters and in a string format.
   g) FR: System must produce a module for administrators to review newly submitted FAQ questions.

Non-Functional Requirements:

Compatibility:
- SR: The system must be compatible with iOS devices.
- SR: The system must be able to integrate third party services.

Performance:
- SR: The system must respond to actively occurring processes within a maximum response time of _ seconds to ensure progress of the system.
- SR: Account creation and profile management processes should be designed to handle a simultaneous load of 100 users without significant degradation in performance
- SR: Recipe generation is efficient, taking a few seconds max (internet speed dependent)
- SR: Recipes are accurate and load in less than a few seconds

Security:
- SR: Authentication and authorisation mechanisms must adhere to industrial-standard protocols to safeguard user authentication and profile information.
- SR: User data, profiles, submitted recipes, and ratings, must be stored and transmitted securely using encryption methods to protect against unauthorised access.

Scalability:
- SR: Systems must be designed to scale horizontally to accommodate an increasing number of users, recipes and interactions without compromising performance.

Availability:
- SR: System must ensure a minimum of 99% uptime to provide users with continuous access to authentication, account creation, and critical functionality.

Reliability:
- SR: Features, e.g., dynamic recipe generation and user-specific notifications must comply with data privacy standards to protect user information
- SR: System logic for sorting recipes is efficient for quick loading.

Usability:
- SR: The UI across all features should follow accessibility standards to ensure a positive UX for individuals with disabilities.
- SR: Navigation menu is intuitive with a user-friendly layout and useful tooltips.
- SR: System should score highly in usability testing.
- SR: System complies with WCAG for accessibility requirements

- SR: Interface is easy to use for new users and users with disabilities (through use of appropriate colour schemes which follow accessibility guidelines).
- SR: Submitting information through user-friendly forms.

Compliance:

- SR: System must comply with relevant data protection regulations to safeguard user data and ensure legal and ethical use (e.g., GDPR, HIPAA)

Quality Assurance:

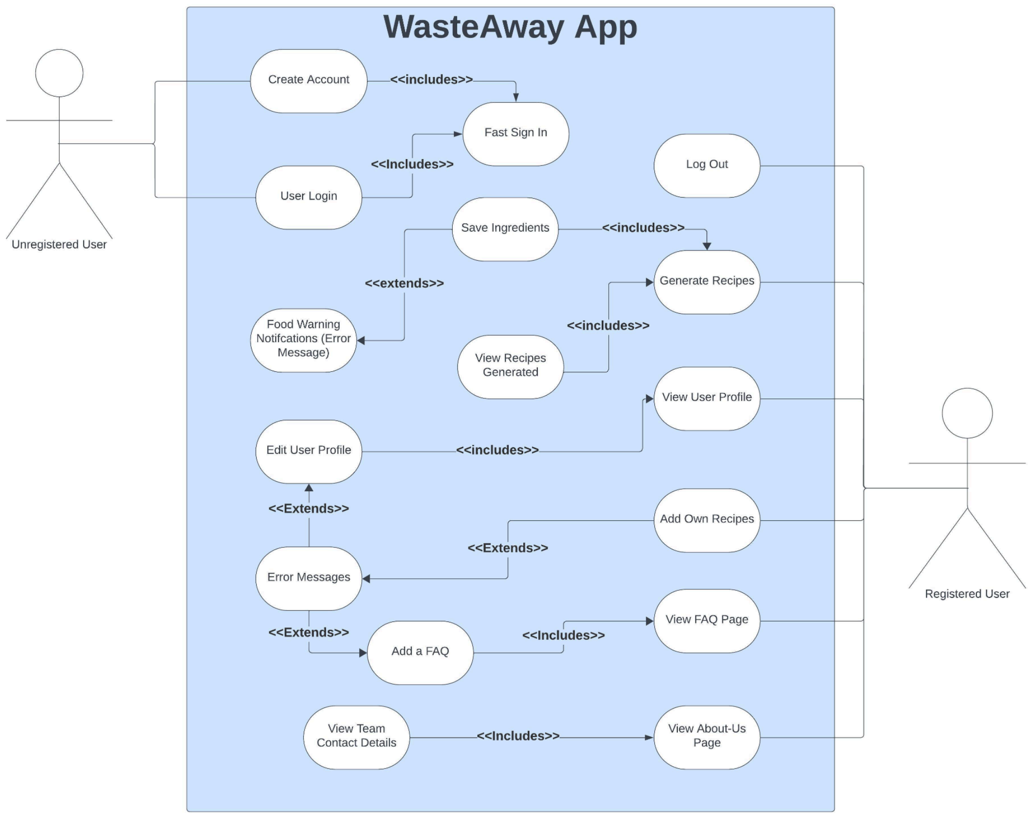
- SR: Rigorous testing procedures, e.g., security testing, must be conducted to ensure the overall reliability and integrity of the system.

Design

This section outlines design considerations for the iOS application and its integration with the Firebase database. Grounded in system requirements, these decisions form the basis for user cases, interfaces, and methods, with procedural aspects represented through models for the subsequent implementation stage.

Use Case Diagram:

Explores functionalities specified in the problem specification (1-9) to serve as the foundation for design and implementation. Each use case references the problem specification, identified by the user requirement ID shown in the functional requirements. The application is currently designed for registered and unregistered actors. The strategic arrangement of use cases influences the UI design and flow, detailed in the subsequent section.



‘Pre-Configurations are made visible within the architectural design thus illustrating some validation methods the architectural managers might utilise’

Use Case 1: Recipe Generation (4)	
Actors	**Registered** WasteAway User
Description	The user seeks to obtain a curated set of top 10 recipes. To achieve this, the user selects ingredients through the select ingredients form. Triggers recipe generation process. System generates recipes based on selected ingredients and food restrictions attached to the user profile.
Pre-condition	User is presented on the home page of the application. User has already inputted ingredients and food restrictions (Save Ingredients, Edit User Profile).

Post-condition	User presented with a curated set of top 10 recipes.
Data	**Input**: User tick boxes on select ingredients form. Activates initiation process (widget button). **Output**: Generate 10 recipes, Error message.
Valid Case	User is logged in, ticks boxes of ingredients (**getIngredients()**) they have inputted previously (front-end). The ingredient manager verifies the selected ingredients against configured rules. The ingredients, user's food restrictions are sent to the **Recipe API** to retrieve recipes to provide a response to the user (**getRecipes()**). The recipes are organised accordingly and displayed to the client interface.
Error Case	Erroneous Data: The user selects ingredients in submission form. User imitates the generation process. **Ingredient manager** validates data against preconfigured rules; error is elicited. 'Not enough ingredients or ingredient types provided' to the client interface. **Users are notified** of the issue type and can select more ingredients for the generation process and re-initiate the process.

Use Case 2: Save Ingredients (4)	
Actors	**Registered** WasteAway User
Description	The user inputs a list of ingredients through the client side application on (Recipe Generation Page). The ingredients are then attached to the user profile in a record file in the Firebase database backend.
Pre-condition	User is logged in and presented on the homepage of the application.
Post-condition	User's submitted ingredients are stored in backend and attached to user profile (UID)
Data	**Input**: Ingredient name, Ingredient expiry date. **Output**: Success Message / Error Message
Valid Case	User inputs a list of ingredients along with equivalent expiry dates. User triggers the submission of the ingredient list. This is validated by the **ingredient manager** against preconfigured rules. User (**UID**) is retrieved from Firebase database (**getUID()**). The list of ingredients is then submitted to the backend attached to the user's records (**storeIngredients()**). Adjusting the list of ingredients that the user has access to. A success message is then elicited to the client interface 'Ingredients have been added'.
Error Case	Erroneous Data: User inputs a list of ingredients along with equivalent expiry dates and triggers submission. **Ingredient manager** elicits error due to expired item entered into ingredient list "Expired Item Entered'. The user can then modify the error on the form and resubmit. This process will repeat but for the alternative cases of 'Incorrect Ingredient Names', 'Empty Inputs' this will be handled by the ingredient manager and elicit errors to the client interface to ensure integrity of information in the backend. Replicated Ingredient: Similar scenario, the user submits ingredients. Upon submission, the **ingredient manager** checks for duplicate ingredients on the submission form. If identified, a notification is generated 'Duplicate Ingredient Located' on the client interface. The user is prompted to review and address the replicated entries on the form.

Use Case 3: Edit User Profile (3)	
Actors	**Registered** WasteAway User
Description	The user interacts with the user profile functionality to manage and view their personal information within the application.
Pre-condition	The user is logged into the application and navigates to the user profile page.
Post-condition	The user successfully updates user profile information and backend Firebase updates this information using user (**UID**). The user can now view this live update on their profile.
Data	**Input**: User profile details **(name, email, password, food restriction, bio)** **Output**: Updated view on user profile description
Valid Case	Users access their user profile section within the application. The user is presented with an interface displaying their current profile information. User decides to update certain details, such as name, email, food restriction etc. The user submits the information through the provided form through initiation of a button. The **account manager** validates the data against preconfigured rules. If successful, the manager gets user UID (**getUID()**) and updates the user profile record (**sendUserDetails()**) to the backend Firebase database.

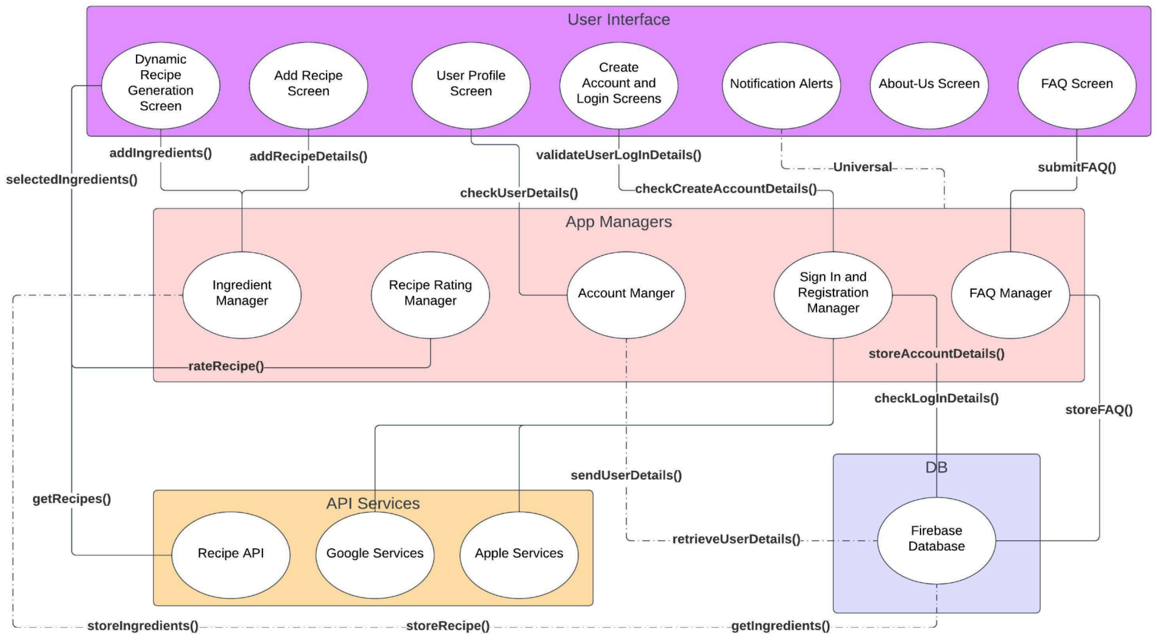
Error Case	<u>Erroneous Data</u>: User accesses the user profile section. User initiates the edit profile form to update certain details. The **account manager** validates data against preconfigured rules. Error is elicited 'Invalid Name Entry' in the scenario of empty inputs or password is not long enough 'Invalid Password Entry' for example. The user is notified of the error, client side. The user corrects the form and resubmits. <u>Authentication Error</u>: User attempts to access or modify their profile through the edit user profile page. However, the user's session has expired. The application prompts the user to log in again to access the profile functionality.

Use Case 4: Food Warning Notifications (5)	
Actors	**Registered** WasteAway User
Description	The registered user aims to generate recipes using the recipe generation. Notification of expired ingredients is elicited to the client interface, the backend Firebase database then removes these expired ingredients from the system.
Pre-condition	Users are logged into their account and navigated to the Recipe Generation Page. User has previously entered ingredients (Save Ingredients)
Post-condition	User is notified of any expired ingredients, and the recipe generation proceeds with updated data.
Data	**Output**: Warning Notification
Valid Case	Users log into their account and are presented with a generation page. User activates a button to select the ingredients for the feature. The system retrieves the ingredients (**getIngredients()**) from the Firebase database using the user **UID**. The **ingredient manager** checks the expiration date of each ingredient against the current date. If no ingredients have expired then the user is forwarded to the form for selecting ingredients.
Error Case	<u>Expired Ingredient</u>: User logs into their account and presents with a generation page. User activate button to select ingredients for the feature. System retrieves ingredients (**getIngredients()**) from the Firebase database (**UID**). The ingredient manager identifies expired items. The system notifies the user of the expired items, using the ingredient name and ingredient expiry. The user then has an option to continue without the expired ingredients or to halt the process to add more ingredients.

Use Case 1: Add Own Recipes (8)	
Actors	**Registered** WasteAway Users
Description	The user intends to add their own recipes to the application.
Pre-condition	The user is logged into the application and is on the page dedicated to adding new recipes.
Post-condition	The user's custom recipe is successfully added to the application's recipe database.
Data	**Input**: Recipe details (recipe name, ingredients, cooking method). **Output**: Success Message / View recipe
Valid Case	The user is navigated to the Add Own Recipes page of the application. The user fills in the required details for the new recipe (**addRecipeDetails()**). The user submits the form which is validated by the ingredient manager. Upon successful validation, the user's custom recipe is added to the application's recipe database (**storeRecipe()**). A confirmation message is elicited to the client interface and the user can view their added recipe.
Error Case	<u>Erroneous Data</u>: User fills in the required details for the recipe. User submits the form containing recipe details. The recipe manager identifies the errors in the entered data e.g., missing fields. An error message is displayed on the client interface. The user reviews information and corrects their form.

<u>Architectural Design:</u>

'This system architecture is designed to provide a robust and scalable framework for the WasteAway application'



The architecture is organised into three layered components: Frontend: UI components (Purple), App manager (Pink) and Backend: API services (Yellow) and DB (Blue). These components work cohesively to ensure the effective operation of the system. The following description will elicit each architectural component's purpose and interaction with the system. In the following section, is a comprehensive description of the system architecture and their relationship with other modules along with an ID corresponding back to the user requirements.

Components and Modules:

Dynamic Recipe Generation Screen (DRGS) (4): Displays top 10 recipes based on user's food restriction and database stored ingredients. Allows users to input ingredients.

- addIngredients() procedure: ingredient name, ingredient expiration date for validation.
- selectedIngredients() procedure: list of ingredients elicit list of recipes from Recipe API.

Add Recipe Screen (8): Allows users to input and add new recipes to the system.

- addRecipeDetails() procedure: recipe name, list of ingredients for validation.

User Profile Screen (3): Shows individual user profiles, including personal details and bio etc.

- checkUserDetails() procedure: user's name, email, password, food restriction for validation.

Create Account and Login Screens (1,2): Enable users to create accounts and log in securely

- validateUserLoginDetails() procedure: email, password for validation.
- checkCreateAccountDetails() procedure: email, password for validation.

Notification Alerts (5): UI component that produces relevant notifications (Food Notifications), important information (Error Notifications) across the entire application.

- Universal (Notifications are elicited on client interface from all application manager)

About-Us Screen (6): Provides information about the application, purpose, and team behind it and their contact information.

FAQ Screen (9): Displays a FAQ section to address and search common user queries.

- submitFAQ() procedure: FAQ query, FAQ description for validation.

Ingredient Manager (4, 5, 8): Validation of ingredient and recipe data, including recipe names, ingredient names, and ingredient expiration dates. Objective is to prevent the storage of incorrect information in the Firebase database and ensure that inaccurate data is not presented on the client interface.

- storeIngredients(), storeRecipe() procedures: storing into a database using UID after validation process.
- getIngredients(): uses UID to retrieve a list of ingredients for use in DRGS.

Recipe Rating Manager (7): Responsible for ensuring a fair and accurate assessment of recipe rating (aggregate functions). Goal is to validate and judge recipe ratings in a transparent manner. This process guarantees that user-provided ratings are considered valid (rating description, UID, number_of_stars) and provides an explanation for each rating, contributing to a more robust and trust-worthy recipe evaluation system.

- rateRecipe() procedure: validates feedback from a recipe generated by Recipe API.

Account Manager (3): Responsible for overseeing user account-related functions. This includes tasks such as profile creation, and account management. Role is to ensure secure handling of user data (username, password, food restriction, bio), interaction between users and system's back and frontend.

- sendUserDetails() procedure: stores user details after correct validation of user inputs.
- retrieveUserDetails() procedure: retrieve user information using UID, to display on the User profile screen.

Sign In and Registration Manager (1,2): Tasked with facilitating user access to the system. Oversees user sign-in procedures through API services (Apple and Google services), and manual sign-in and registration procedures. Ensuring data is validated (username, password) before submitting to the Firebase Database for example.
- storeAccountDetails() procedure: user details are stored into Firebase database (manual registration) after correct validation.
- checkLoginDetails() function: validated user details are checked against user records, return True if details are valid or False if invalid.

FAQ Manager (9): Responsible for organising and presenting Frequently Asked Questions (FAQs) within the system, ensuring accuracy and user-friendliness to enhance user understanding and satisfaction. Validates data such as (Question name, User UID, Question content) before submitting the question to the Firebase database.
- storeFAQ() procedure: validate FAQs are stored in the format of a list to the database using UID.

Recipe API (4, 5, 7) : Facilitates access to recipe-related data and functionalities with the system.
Google and Apple Services APIs (1,2): Integrates services from Google and Apple, allowing users to sign-in and create accounts through these popular service providers.
Firebase Database API (*): Manages communication with the Firebase Database (NoSQL), enabling efficient storage and retrieval of data within the system.

For alternative models, consider a microservices architecture, where each component is more loosely coupled and can be developed, deployed, and scaled independently. This could provide more flexibility and resilience but can at the cost of increased complexity in terms of network interactions and data consistency. The layered architecture simplifies the development process by clearly defining and separating the UI from business logic and data storage, making it easier to maintain and update the system. However, it could potentially create a monolithic application that might be less scalable and more difficult to deploy in a distributed environment compared to microservices. To comply with the specification, the architecture should ensure that all data flows are secure, the validation of data is thorough to prevent incorrect information from being stored or presented, and that the system is capable of scaling to handle a large number of user requests without performance.

Preconfigured Rules:

'These are some of the configured rules that are mentioned for the above use cases for each of the app manager's mentioned'

Manager Name	Input	Validation Method	Other Validation Methods
Ingredient Manager	Ingredient name, ingredient expiry date, food_restriction, etc. List of ingredients and recipes.	**Rule**: Number and type of ingredient selected is viable for recipe generation. **Rule**: Verify that the selected ingredients align with the user's food restriction.	**Rule**: Validate submitted ingredients against a set of valid ingredients. **Rule**: Validate identify and reject ingredients with expiry dates that have passed the current date.
Account Manager	Email, password, food restriction, bio etc.	**Rule**: Validate user inputs such as password length: contains a symbol, greater than 6 characters and less than 20. Email contains '@' and is a valid email.	**Rule**: User's session is active **Rule**: User's food restriction is validated against list of food restrictions.
Recipe Rating Manager	Rating description, Rating stars.	**Rule**: Validate user inputs (Rating description less than 200 characters, Rating stars less than or equal to five).	**Rule**: User's rating should be reviewed by admins before displaying to other user's on the application.

Implementation

Version Control Setup:
- **Git and GitHub** ensure a centralised location for codebase management and version tracking.
- Issue Tracking: Jira for prompt issue tracking and resolution
- **Link to Repository**: https://github.com/RhysRoo/SDrecipe

Programming Languages:
- **Dart** is the primary programming language, chosen for its compatibility with Flutter.
- **Flutter**, a UI toolkit, is employed for cross-platform mobile application development. The choice ensures a consistent and visually appealing user experience (UX) across iOS platforms.

Devices Targeted:
- **Uses iOS emulators** for development and testing.

Database Management:

- **Firebase database API (NoSQL)** for real-time data storage and retrieval.
- Real-time synchronisation for scalability

Libraries:

- **Provider** package is utilised for state management within the Flutter framework. Ensuring data flow and updates to the application (e.g., changes in user profile)
- **Firebase** packages, including **Firestore** and **Authentication**, for database interaction and user-auth.

Feasibility Check:

- **Confirm Flutter supports iOS platforms** to deploy the WasteAway app deployment.
- **Flutter and Dart API Changes** (Review release documentation to ensure application remains compatible with latest features and updates)
- **Flutter Version Compatibility**: Ensure that Flutter is compatible with the latest Dart SDK.
- **Assess security practices of Flutter and Dart**; **regularly update IDE** to benefit from the latest security enhancements.

Testing

'The test strategy provides a brief overview of the testing approach, and test inputs specify the scenarios used for testing each system requirement'.

Requirement ID	**Test Strategy**	**Test Inputs**	**System Requirement (ID-Functional Requirement)**
0.01	Integration Testing	Valid and invalid user credentials, interruptions during authentication, resuming authentication.	User Authentication Control **(1)**
0.02	Validation Testing	User details with valid and invalid email formats, duplicate emails addresses, password strength tests.	User Account Creation Control **(2)**
0.03	Data Verification Testing	Profile creation with various character lengths, input types and predefined data structures for food restrictions	User Profile Management **(3)**
0.04	Functional Testing	A minimum and excess number of ingredients, expired ingredients, ingredient names with typos.	Dynamic Recipe Generation **(4)**
0.05	Notification Testing	Profiles with expired ingredients, profiles with unknown allergens, recipes using expired ingredients	Food Warning Notifications **(5)**
0.06	Content Verification Testing	Team members profile updates, bio character limits, message character limits, contact form functionality.	About-Us Page Functionality **(6)**
0.07	Recipe Rating and Feedback	Recipe rating submissions, feedback character lengths, aggregate rating calculations.	Recipe Rating and Feedback **(7)**
0.08	Submission Testing	Recipe submissions with various ingredient lists, recipe names, and cooking methods, content moderation scenarios	User Recipe Submission **(8)**
0.09	Search Functionality	FAQ entry additions edit, deletions, search functionality with typical user queries.	FAQ Page Interactivity **(9)**

Critical Analysis

Leadership:

- **Regular check-ins and open dialogue** have created a unified team with a shared vision for the WasteAway application.
- **Adaptability in leadership style and active encouragement of team input** have promoted a sense of ownership among team members. Furthermore, adaptability allowed for accommodating diverse working preferences and skill sets within the team (e.g., working in different groups and communicating between groups)
- **Clear communication channels through regular Discord meetings and library meetings every Thursday** ensured the team was aligned with project objectives. This created a sense of unity and shared responsibility among team members.
- **Agile refinement**, includes continuous refinement of the project backlog, ensuring that user stories and tasks are well-defined, prioritised, and aligned with project objectives.

Progress Monitoring:

- **Regular reviews occurred over Discord** which have proven effective in evaluating completed tasks, identifying challenges, and adjusting the project plan and areas of focus.
- **Implementation of a robust version control system** (as mentioned), will allow for tracking changes, resolving conflicts and maintaining a cohesive codebase.

- **Jira will be used along with GitHub** in order to visualise tasks, prioritise activities and maintain a clear roadmap.
- **Knowledge sharing sessions** were enforced, these sessions, led by team members, covering new emerging ideas and participant feedback for example. This approach kept the team at the forefront of the software development project, enhancing quality and efficiency of work.

Conflict Resolution:
- Our approach to conflict was **proactive and collaborative**. Establishing a culture that encourages open communication has been paramount. Team members were encouraged to express concerns, and conflicts, when they arose, were addressed directly in meetings.
- Facilitating **constructive conversations**, ensuring that all perspectives were heard, and guiding the team towards mutually agreeable solutions. Turning conflicts into opportunities for learning and potential growth of the system. These diverse viewpoints enhanced WasteAway's overall quality.
- A **structured escalation process** was maintained, ensuring that issues were elevated to the appropriate level for resolution while maintaining a positive and collaborative team environment.
- **Continuous feedback loops** were enforced, these loops include regular group sessions between team members, creating a safe space for open dialogue. This ongoing communication ensures that concerns are addressed promptly.

Participation Form

Team member name	Detailed contributions per chapter	Percentage contributions overall	Signature
Joshua Varney (up2138017)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	[illegible]
Rhys Parsons (up2125866)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	[illegible]
Harry Miller (up2120303)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	HMiller
Khadija Baffa (up2157506)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	[illegible]
Matthew Bowers (up2112240)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	[illegible]
Cindy Murimi (up2060987)	Chapter 1: Problem specification Chapter 2: Design Chapter 3: Implementation Chapter 4: Testing Chapter 5: Critical analysis	15	[illegible]