Problem Specification

User Requirement (UR), System Requirement (SR), Functional Requirement (FR), Non-Functional Requirement (NF).

Each system and functional requirement has been linked to a particular test in the test plan stated by a group of tests associated with a particular feature and then an id that associates best with meeting the associated requirement. From the previous iteration it was stated that **'Red UR'** are fundamental features where the majority of team effort was designated to and **'Blue UR '** being less fundamental.

UR: ***User Authentication Control****: The user must be able to log into their account*

a) (FR 1 a) has been met **('Beta-Testing (Ref UR 1 and 2) id 2',('Login UI and Register_Login_Manager Tests', id 1)) .**
b) (FR 1 b) has been met **('Beta-Testing (Ref UR 1 and 2) id 4')**
c) (FR 1 c) has been changed **('Beta-Testing (Ref UR 1 and 2) id 5')** to 'Suspension of User Login must require re-inputting login details'.
d) (SR 1 d, e) has been met and improved **('Login UI and Register_Login_Manager Tests', id 5)** to 'password length above 6 characters and no more than 255. Must include 1 @ character'
e) (SR 1 f) has been met **('Beta-Testing (Ref UR 1 and 2) id 4', 'Login UI and Register_Login_Manager Tests', id 4)**

The majority of functional and system requirements have been met for **UR 1**. **(UR 1 c)** has been altered due to agreement on requiring users to re-enter login details after a suspension improves security by ensuring authorised access and verifying user identity, accountability and compliance with regulatory standards. Furthermore, researching the sign-in system, it was found that the majority of systems asked for re-inputting data after a halt and therefore was adhered to. Testing progress was rigorous in terms of validation rules **(UR 1 d , e)** but lacked mocking of google service authentication due to lack of global research knowledge on required unit tests. As a result, beta-testing was conducted to account for this to determine whether the functional requirements have been met. In terms of implementation, the expected standards were created in order to differentiate between different users of our system. Therefore, this step gave the team the green light to implement user-directed features. For testing this system: Username: josh1@gmail.com, Password: josh@gmail.com.

UR: ***User Account Creation Control****: The user must be able to create an account*

a) (FR 2 a) has been met **('Beta-Testing (Ref UR 1 and 2) id 1')**
b) (FR 2 b) and (FR 2 c) have been met **('Beta-Testing (Ref UR 1 and 2) id 3')**. **(FR 2 c)** has been changed to 'Suspension of Create Account must require re-inputting sign up details'.
c) (SR 2 d e f) has been met (**'Login UI and Register_Login_Manager Tests, id 5')**
d) (SR 2 g) has been met (**"Beta-Testing (Ref UR 1 and 2) id 1')**

Similarly to UR 1, the majority of functional and system requirements have been met for UR 2. **(UR 2 c)** has been altered due to agreement on suspending account creation and mandating re-inputting signup details acts as an avoidance against fraudulent or duplicate accounts, promoting a more secure and authentic user base. Additionally, it ensures compliance with terms of service and upholds data accuracy by verifying users' commitment to providing genuine information.Testing lacked mocking of google service authentication due to lack of global research knowledge on required unit tests but validation rules were tested extensively **(SR 2 d e f)**. Beta-testing was developed to determine whether the functional requirements have been met through testing the backend database and widget features as a result of issue masking google auth **(Beta-Testing (Ref UR 1 and 2).** From an implementation standpoint, the implementation of this UR is fully functional with the appropriate validation rules and therefore allows for automatic account creation from manually inputting user credentials for the team's developers.

UR: ***User Profile****: The user must be able to create an account*

a) (FR 3 a) has been met ('**Profile Manager Test**', id 3)
b) (FR 3 b) has been met ('**Profile Manager Test**', id 5)
c) (SR 3 c) has been met gradually **(more than one significant test)** through ('**Profile Manager Test**', id 4, 6)
d) (SR 3 d) has been partially met ('**Profile Manager Test**', id 9)
e) (SR 3 e) has been met ('**Profile Manager Test**', id 6)

In context of the **UR 3**'s SR and FR for User Profile the majority of the requirements have been met successfully. There have been no changes to the functional or system requirements. Due to time constraints, the system requirement **(UR 3 d)** the validation of food restrictions against appropriate data structures remains unmet **(part of the requirement)**. While the majority of user and functional requirements for the User Profile have been successfully addressed, this specific aspect requires further iterations to implement effectively because we felt that an API should be used to address this issue instead of a static data structure which is prone to both developer accidents and security issues. Future iterations will prioritise the fulfilment of this requirement to ensure comprehensive functionality. In terms of development, the system is near fully functional, additional features to the requirement can be added such as implementation of user images and ability to reset their password.

UR: ***Dynamic Recipe Generation****: Users must be able to generate top 10 recipes based on selected ingredients and food restrictions.*

a) (FR 4 a) has been changed ('**Api Search Test**', id 2) to 'Top 5 Recipes'
b) (FR 4 b) has been met ('**Api Search Test**', id 2, '**Food Notification Manager Tests**', id 2)
c) (FR and SR 4 c, d) has not been met
d) (SR 4 e) has been met (**"Food Notification Manager Test"**, id 2,)
e) (SR 4 f) has been changed ("**Api Search Test**", id 2") to 'Personalised Recipe Generation Engine'
f) (SR 4 g) has been met ("**Open Food Api Tests**", id 1, **Food Notification Manager Tests**', id 2)

The **UR 4** of Dynamic Recipe Generation has been met to a high extent along with a multitude of changes to SR and FR. Due to time constraints and constraints with API usage (e.g., paying for a payment plan), as well as platform to develop. The generation of **(UR 4 a)** has been changed to top 5 recipes due to device size but has been met from a development standpoint as expired ingredients **(UR 4 b, e)** have been filtered out from Food Notifications **(UR 5)** successful completion. **(UR 4 c and d)** has not been incorporated into the system, as a consequence of the previous data SR of UR 3 associated with food restriction validation rules. However, recipes can be generated through user inputted ingredients from the 'Add Ingredient page'. The system is a personalised engine meeting partially to **(UR 4 f)** but doesn't incorporate food restrictions into its search engine due to API payment plans, to overcome this filtering issue of inputted ingredients it will be required that the user should take caution when entering ingredients. An additional API 'open_food_facts' has been added to achieve the system requirement used to validate whether an ingredient is valid therefore completing UR **(4 g)** and making the requirement highly significant in the accuracy of our recipe generation algorithm. From a testing perspective, the API usages of both **'open_food_facts and 'Adaman API'** have been highly thoroughly stated in **'Api Search Test'**, "**Open Food Api Tests**" therefore providing important functionality for our system.

***UR**: Food Warning Notifications: User must be notified on expired food items that have been selected for Recipe Generation*

a) (FR 5 a) has been met **('Food Notification Manager Test', id 1)**
b) (SR 5 b) has been met partially **('Food Notification Manager Test', id 5)**
a) (SR 5 c) has been met **('Food Notification Manager Test', id 2)**

The **UR 5** of Food Warning Notifications is that the majority of SR and FR have been fulfilled in terms of notifying the user and removal of expired ingredients **(UR 5 a, c)** when incorporating **'Ingredient Manager'** functions e.g., **"addIngredients()"** to add ingredients to the google cloud. However, **(UR 5 b)** has been met partially as an efficiency notification can be derived to the interface but an algorithm efficiency system cannot be derived without a large amount of data analytics to determine our recipe generation algorithm efficiency. Although, it was decided that the most important information was removing expired ingredients from the recipe generation **(UR 4 g)** to prevent health concerns with users. From a development standpoint, the food warning notifications have been successfully implemented with removing expired ingredients. However, efficient notifications have been derived to the client interface without a yet-to-be efficiency calculation system. On the other hand, testing has been rigorous with removing expired ingredients but lack of testing on evidence of the food notification being elicited to the interface which requires an **"existence test"** in future iteration.

***UR**: **About-Us Page**: Users should have access to company-information.*

a) (FR 6 a) has been developed
b) (FR 6 b) has been developed
c) (SR 6 c) has been developed
d) (SR 6 d) has been changed 'System must statically managed team members profile through structured code"
e) (SR 6 e) has been changed 'System must statically ensure bio is less than 200 characters and messages to users are 300 characters or less' and has been developed
f) (SR 6 f) has been developed

The UR 6 of About-Us Page is that the majority of SR and FR have been met partially on a functional and developmental level. Through the lack of validation rules implemented as the team agree that this UR is a less fundamental UR, **(SR 6 d, e)** has been changed so that within future iterations this validation system can be implemented in full. The testing for this feature was not rigorous and does not meet the testing expectations required by the specification due to time allocation on the project's goal **(UR 4)**. As a result, structural tests should be developed as well as functional and validation tests in future iterations.

***UR**: **Recipe Rating**: User should be able to rate recipes on the system and provide viable feedback*

a) (FR 7 a, b, c, d) has not been met

In previous team meetings, regarding the prioritisation of development and testing efforts, with a decision made to prioritise the implementation of the 'Recipe Ratings' feature **(UR 7)**. This choice stemmed from the assessment that the generation of recipes held greater significance compared to the proposed community-oriented aspect outlined in previous iterations of the specification. Consequently, due to time constraints and the unavailability of team members to undertake additional development tasks within the current schedule, it was collectively agreed that UR 7 would not be pursued at this time. This strategic decision ensures that available resources are allocated efficiently to meet the primary objectives of the project within the given timeframe. Furthermore, due to a team member resigning from the group, this increase of workload caused a reassessment on the distribution of work.

*UR: **Add Own Recipes**: User should be able to enter their own recipes to the system*

a) (FR 8 a) has been met **('Add Recipe Manager Tests', id 4)**
b) (FR 8 b) has not been met
c) (FR 8 c) has been met **('Add Recipe Manager Tests', id 4, "Open Food Api Tests", id 2)**
d) (FR 8 d) has been partially met **('Add Recipe Manager Tests', id 2, 5)**

The UR of Add Own Recipes **(UR 8)** has been met to a moderate extent. **(UR 8 a and c)** were attained through testing the firebase backend to determine if user saved information can be displayed and inputted into the interface. Furthermore, the user-friendly form inputs were validated using the **'open_food_facts api'** to meet the requirement **(UR 8 a)** of approving user-submitted recipes in the backend for frontend processing. **(UR 8 b)** was not attained through lack of the development of an automatic administrative system due to the workload proposed on validating user ingredients for recipe generation **'open_food_facts_api'** and **'Adaman API'**. Moreover, as a group it was determined that the **new functional requirement of removing recipes** was more important to prevent redundant recipes being stored in the google cloud. Due to lack of administrative power mentioned in **UR 5,** the validation system determined by **(UR 8 e)** which was partially met but lacks content moderation and quality assessment tools to complement **(UR 8 b)**. To conclude **UR 8**, from a development standpoint this has been met moderately which allows a community-feature of adding user-submitted recipes and removal of recipes that they have developed. In future iterations, this can then be used to develop the **(UR 8)** feature. From a testing standpoint, 'Add Recipe Manager' backend has been tested thoroughly to allow for frontend processing of this system, therefore meeting the requirements through mocking the backend database to determine if recipes can be added and deleted for example.

*UR: **FAQ Page**: User should have the ability to submit and view FAQ results or queries*

a) (FR 9 a) has been met (**"FAQ Page Test", id 1)**
b) (FR 9 b) has been developed.
c) (FR 9 c) has been developed.
d) (FR 9 d, g) has not been met.
e) (FR 9 e) has been developed.

f) (FR 9 f) has been met partially (“**FAQ Page Test”, id 2)**

The UR of the FAQ Page's FR and SR has met extensively in terms of implementation standards. Due to the lack of collaboration, testing the FAQ_Page has minimal unit tests to determine the quality and validation of this feature. **(UR 9 a)** has been represented as a true test to determine that there is a strict format and organisation for the development of user queries and answered questions therefore meeting the requirement. Further to this, questions are strictly under 200 characters on the homepage through structural tests but lack functionality of validation rules **(FR 9 f)** therefore has been met partially. To conclude, from a developmental perspective, the system can add user queries that are validated and added to the frontend when manual engineering has been altered setting the query to true but this lacks an automatic administrative system **(UR 9 d)**. From a testing perspective, this has not been thoroughly tested due to lack of contribution from confidential team members, as a result this will be met in future iterative prototypes to properly meet the requirements.

Compatibility NF Requirement:

a) SR has been met **(Integration with Third Party Services),** evidence **('Add Recipe Manager Tests', id 4**, **'Food Notification Manager Tests', id 2)**
b) SR has been changed (**IOS devices and Chrome Services**) evidence **('Majority of Tests Run on Google Chrome and Some Tests on Recent IOS device'**)

Performance NF Requirement:

a) SR has not been met **(Maximum response time of 5s of actively occurring processes)** evidence **("Api Search Test", id 3)**
b) SR has been partially met and changed to 'Backend database should be designed to handle a large load of data from over 1000 users without degradation in performance'.
c) SR has been met **(Recipe generation is efficient, taking a few seconds max)**, evidence (**"Api Search Test", id 3**)
d) SR has been met and changed **(Recipes are accurate)**, evidence (**"Api Search Test", id 3**)

Security NF Requirement:

a) SR has been met **(Adhere to industrial-standard protocols to safeguard user authentication and profile information)** evidence **(("Profile Manager Test", id 6), ('Beta-Testing (Ref UR 1 and 2) id 2'))** through Google Cloud service usage.
b) SR has not been partially met **('User data, profiles, submitted recipes, and ratings, must be stored and transmitted securely using encryption')** evidence (**'Profile Manager Test'**, id 5)

Scalability NF Requirement:

a) SR has been met through google cloud storage and authentication displayed in **"User Authentication Control Tests"** allow the system to become more scalable to new users and recipes **"Add Recipe Tests".**

Availability NF Requirement:

a) SR has not been met

Reliability NF Requirement:

a) SR has been changed to **('User-specific notifications in terms of prevention of recipes being generated with expired ingredients to prevent health concerns of users ')**, evidence (**'Api Search Test'**, id 2 , **'Food Notification Manager Tests', id 2**)
b) SR has been met **('System logic for sorting recipes is efficient for quick loading.')**, evidence (**'Api Search Test'**, id 2, 3")

Usability NF Requirement:

- As a result of unit tests based on usability being difficult. The UX designer **'Alex Pearson'** provided the following table to determine whether the usability of our system has met the following non-functional requirements.

Specification Standards	Comments
SR has not been met (**'Positive UX for individuals with disabilities.'**)	No accessibility features for those with physical difficulties e.g., voice recognition and gesture detection. However, good colour schemes for visual imparities.
SR has been met **('User-friendly layout and useful tooltips')**	User friendly layout has been approved through ", useful tooltips has been attained through youtube video that has been tested thoroughly tested ("video_player_test")
SR has not been met (**''Score highly in usability testing'**)	Usability score of 5/10
SR has not been met(**'Complies with WCAG for accessibility requirements'**)	No usability with physical disabilities.
SR has been met partially **('Interface is easy to use for new users and users with disabilities')**	Use of appropriate colour schemes and youtube link to introduction ("See video_player.dart") for new users but no usability with physical difficulties.

Compliance NF Requirement:

a) SR has been met partially **("Profile Manager Test", id 6)** due to handling of user profile information that is stored safely in the backend. THis will require a professional review to determine if this NF requirement has been met appropriately.

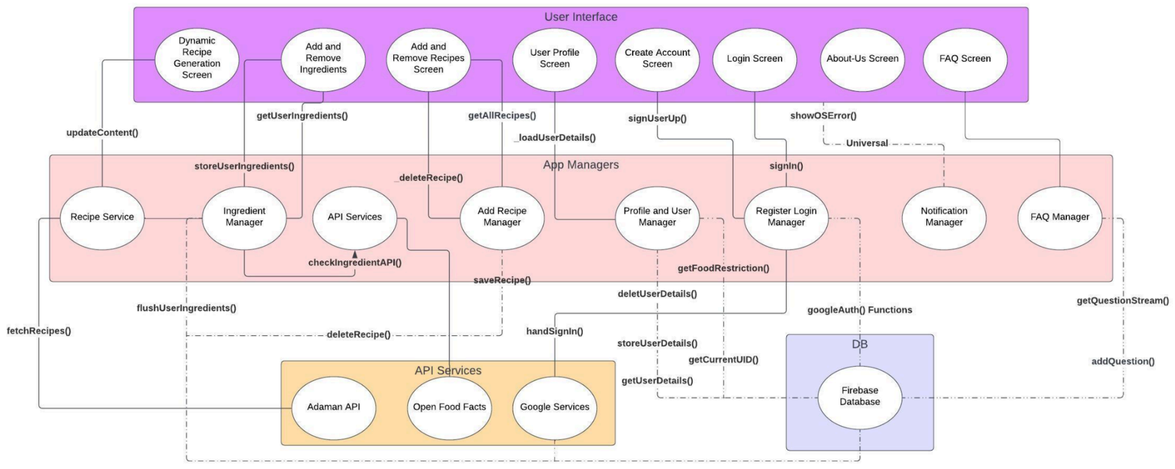
Quality Assurance NF Requirement:

a) SR has been met partially **(See All User Requirement Tests)**, majority of fundamental UR components have been met extensively on a testing level.

To summarise our assessment of NF requirements, the bulk of them have been addressed through the utilisation of Google Cloud mocking services as detailed earlier. This gives us confidence in the scalability and security of our system, particularly concerning user data, thanks to the integration with trusted third-party resources. However, it's essential to note that a comprehensive evaluation by a qualified professional is necessary to ascertain the full extent to which our system meets these requirements. Our system's performance is notably efficient, evidenced by rapid API request speeds when fetching recommended recipes. While user experience (UX) enhancements were prioritised during development, it's evident that non-functional aspects were not given equal attention. Hence, in future iterations, it's imperative to elevate the priority of non-functional requirements to ensure the system's reliability upon deployment to the public.

Design

The figure below illustrates the reverse engineered model of our source code for the application. This will be contrasted with the previous model of our design stated in the first iteration.



The architecture is organised into three layered components: Frontend: UI components (Purple), App manager (Pink) and Backend: API services (Yellow) and DB (Blue). Dotted relationships refer to the interactions and relationships between the module and Firebase backend.

The following components have remained unchanged from the previous iteration model.

- **Google Services** facilitates the fast sign in API to allow users to sign in through faster methods.
- **FAQ manager** facilitates the validation of user entered FAQ questions to the cloud which can then be used for admin authentication to allow new queries to be added to the system.
- **About-Us Screen** provides information about the application, purpose, and team behind it and their contact information for user querying.
- **Create Account and Login Screens** provide users to create accounts and log in securely.
- **Firebase Backend** provides backend functionality for all components of the system.
- **Register Login Manager** still remains as a single modular structure to allow facilitating the similar google services and firebase backend authentication for both 'Create Account Screen' and 'Login Screen'
- **User Profile screen** maintains the sensitive user information that the user can alter and change.

The following components and relationships have been changed from the previous iterative model.

- In terms of interface components 'Add Remove Ingredients Screen' and 'Add Remove Recipes Screen' have been split into seperate interface components due to simpler resource management and simplifying a complex task.

- 'Add Remove Ingredients Screen' relationship has been added to the 'Ingredient Manager' to facilitate the separation of the screen's backend functionality from 'Add Remove Recipe Manager' thus reducing the workload on 'Ingredient Manager' reducing its functions and procedures complexity'. Moreover, the 'Add Remove Recipes Screen' has been added to the new manager 'Add Recipe Manager'.

- A new relationship between the ingredient manager and the API service has been created so that there is better modularity and splitting of API functions so that these API functions can become more reusable between different modules. Thus allowing for separate usages of the 'Open Food Facts API' and allows the system to become more scalable in the future.

- The dynamic recipe generation screen has then been split from the 'Ingredient Manager' to a new 'Recipe Service' manager to ensure that the service can use the functions from the ingredient manager to reduce the workload on the 'Ingredient Manager' which already has the highest complexity of all modules. Furthermore, this relationship will allow for the new sub-module to be facilitated as a separate entity. Furthermore, this separation allows for multiple usages of the Andaman API with new future features but also allows for content filtering to be applied without increasing 'Ingredient Manager's file size.

- The Notification Manager module has been added as it was agreed on by the team that a manager could facilitate the majority of notifications to warn the user across all screens. This module includes the use of food notification warning notifications and efficiency notifications. This will improve the scalability of the system when facilitating more interface screens in the future.

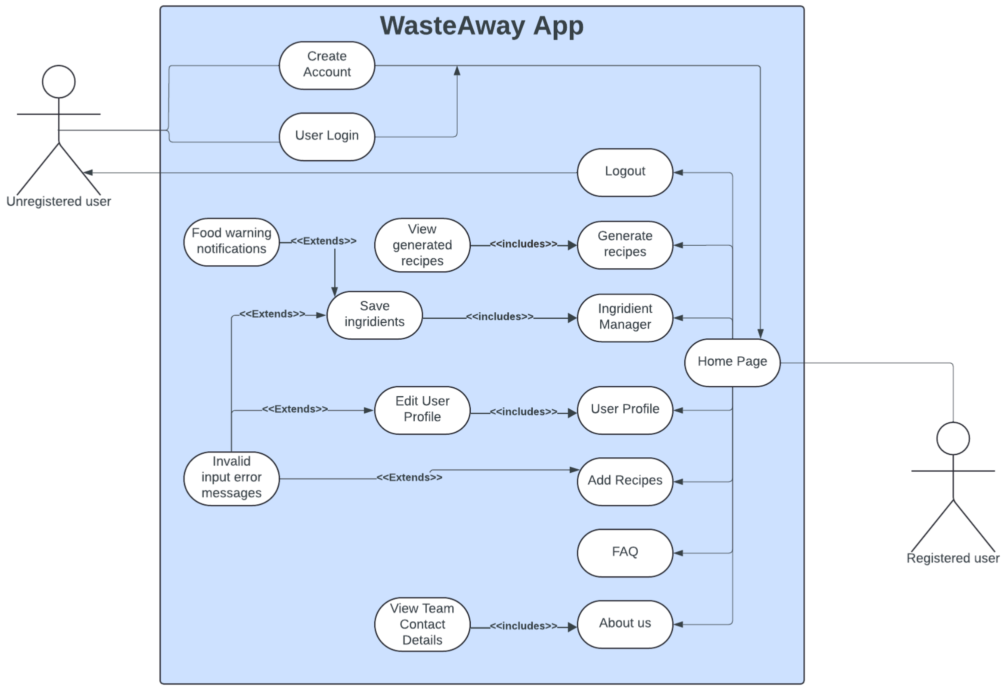
- The new ‘Open Food Facts API’ has been added to facilitate a highly reliable source of information to validate user-inputted ingredients so that inaccurate generation of recipes cannot occur. As a result, this improves the validity of the recipe generation screen instead of using a static data structure which could be prone to redundant information.

- Apple Services was removed due to the lack of wider knowledge on how to link apple services to flutter applications. Furthermore, the complexity of this task would have undermined attaining the fundamental requirements of our application.

- The ‘Create Account and Login Account Screens' were separated as a result of reducing the file size of code as both of the login and create account systems were highly complex along with the use of google services API and interacting with the backend database. Therefore, this was broken down into submodules for improved modularity of the system and developer scalability.

- **The “Profile and User Manager” has been combined through the complexity of the design module above but has been separated into different-submodules** to allow the Profile Manager to facilitate the obtaining of profile related information from the backend. And the ‘User Manager’ to facilitate obtaining user related information for further modules e.g., getUID(). Therefore, all modules have access to the google auth information in order to communicate with unique user records e.g., storing ingredients specific to that user, or storing a food restriction associated with that user.

The following use case diagram illustrates the system’s navigation and functionality features:



Implementation

This simplified section elicits the code documentation (containing the test plan) and repository links. As well as the youtube demo for the application, and a back-up google docs test plan. To run the automated test, clone the repository using git clone https://github.com/RhysRoo/SDrecipe.git , run ‘Flutter pub get’ and then ‘Flutter test’

Link to Video Demo: https://youtu.be/g7YnB9r9-YY

Link to GitHub Repository: https://github.com/RhysRoo/SDrecipe

Link to Code Documentation (readthedocs): https://sdrecipie-docs.readthedocs.io/en/latest/

Link to Code Documentation Repository: https://github.com/RhysRoo/SDRecipie_docs

Link to Back-up Test Plan:https://docs.google.com/document/d/e/2PACX-1vQsVzmZua2yJZgMdotSoZEBsC8wMQ5ZwQj8ZSaBFhSlrLW0kJfV3GhFt3bg1GRvKcJHYBpW169QB2rq/pub

Implementing the system proved to be a formidable challenge owing to the intricate nature of user requirements and the diverse array of functionalities expected from the system. One of the major hurdles encountered was replicating the backend features of the system for testing purposes. To address this, the team acquired the 'Mockito' library, enabling rigorous technical testing and the simulation of complex functions to ensure seamless integration with Google Cloud services. However, another significant obstacle surfaced when attempting to mock Google authentication, a notoriously difficult task. To surmount this, the team leveraged MockUser functions, imported specifically for this purpose, ultimately enabling smoother authentication processes within the system. Despite the complexities encountered, these strategic solutions facilitated progress and ensured the successful implementation of the system.

Implementing the system encountered additional challenges with Google sign-in authentication, which proved intricate to mock effectively for testing purposes. Moreover, the adoption of SHA 256 for Apple sign-in added another layer of complexity, ultimately leading to the decision to remove Apple sign-in from the system. This strategic move allowed the team to prioritise fundamental features without compromising on quality or diverting resources towards resolving compatibility issues. Despite the setbacks, the team's focus on overcoming authentication obstacles and streamlining functionality ensured the successful implementation of the system, demonstrating adaptability and problem-solving prowess throughout the development process.

The functionality and testing of the 'Rating Page' and 'FAQ Page' proved challenging due to team members facing time constraints, resulting in incomplete tasks and a large distribution of information among the remaining team members. This fragmented approach to development adversely impacted the prototype, particularly with regards to these specific features. The limited availability of team members hindered thorough testing and optimization, causing delays and setbacks in ensuring the robustness and effectiveness of the 'Rating Page' and 'FAQ Page' functionalities. Despite these obstacles, the team persevered, making concerted efforts to prioritise tasks and allocate resources efficiently to mitigate the impact of time constraints on the overall development process.

Testing

Displayed below is a simplified version of a test plan which has been written on the 'Test' folder the github repository stated below. This is extremely simplified as requirements meet the specifications in above. More extensive testing can be run through the automated testing and be seen for all UR and NF components met.

Link to Code Documentation (readthedocs): https://sdrecipie-docs.readthedocs.io/en/latest/
Link to Back-up Test Plan:https://docs.google.com/document/d/e/2PACX-1vOsVzmZua2yJZgMdotSoZEBsC8wMO5ZwOj8ZSaBFhSlrLW0kJfV3GhFt3bg1GRvKcJHYBpW169QB2rq/pub

To run the automated test, clone the repository using git clone https://github.com/RhysRoo/SDrecipe.git , run 'Flutter pub get' and then 'Flutter test' to initiate the required tests. The back-up youtube will describe the process further as there is no link to automated test plan.

Critical Analysis

Leadership:
Although we never formally appointed a group leader (scrum master), one of our experienced team members, Josh, naturally assumed the role. He demonstrated a clear vision and drive for the project, steering us in the right direction, such as guiding the transition from implementation to testing to prevent scope drift. While task assignments were made during group meetings based on individual strengths and confidence levels—some excelling in backend tasks while others in frontend—the project leader, Josh, played a pivotal role in ensuring collaboration of the team.

In instances where team members encountered roadblocks, Josh stepped in to provide assistance, keeping the project on course. If anyone faced challenges with a particular aspect of the software, Josh would offer guidance and, if necessary, rearrange tasks to optimise efficiency. Moreover, Josh facilitated conflict resolution, leveraging his authority to foster a healthy and productive team dynamic. Our team has operated seamlessly with this structure since the start of the semester, with Josh's leadership proving instrumental in our success.

Progress Monitoring:
We developed using Scrum, an agile software development methodology using Jira. Sprints were used to break down the task into small segments by producing incremental deliverables enabling more manageable subtasks. When subtasks were marked as complete and added to the done board, the team was made aware of these changes (assigning each user to a specific task). Keeping a consistent log is good as it prevents conflicts from occurring and stops tasks from getting performed more than once. Scrum helped us stay focused on the projects and ensured we had concrete deliverables. On the first teaching block, we used a Snapchat group chat, which was our initial form of communication as it was quick and easy to set up and invite everyone. But as a team, we hit a few roadblocks along the way. The first was chat history: by default, all Snapchat chats expire after 24 hours, and it is messy to find old chats if they are saved. Secondly, you can't attach documents in Snapchat group chats, which can make it hard to keep the team informed of important documents. From teaching block 2, we decided to use Slack. This is a communication piece of software that is used in the industry by many software development companies, and we thought it would be good practice to learn and utilise this platform for our project. Individual Branches: Each team member would use their own individual branches, making it easy to distinguish work from each other, allowing features and bug fixes to work in isolation from the main. The rest of the team were able to keep track of these key steps as the contributors would see pull requests. This visibility is crucial for managing, checking quality assurances and reviewing the progress of development tasks keeping to project standards before completing.

Conflict Resolution:
To address conflicts stemming from communication breakdown and task allocation issues, it's important to identify the underlying causes and implement a comprehensive resolution strategy. Open and honest dialogue among team members was encouraged during team-meetings and using Slack, providing a platform for everyone to voice their concerns and perspectives. Transitioning to more effective communication channels, such as face-to-face meetings or video conferences, enhanced clarity and alignment of team goals and opinions. Establishing clear meeting protocols, including the documentation of meeting minutes and assigned tasks, allowed for transparency and accountability within the team. Task allocation should be based on individual strengths and weaknesses, with opportunities for skills development and project improvement.

Encouraging problem-solving by creating a culture where team members feel empowered to raise issues and propose solutions collaboratively. Facilitating conflict resolution through active listening sessions, aiming to find mutually acceptable solutions. Implementing a feedback session weekly as a mechanism allows for continuous improvement and adjustment of conflict resolution strategies based on team dynamics and project needs. By embracing these conflict resolution techniques, the team can effectively navigate challenges and foster a more collaborative and a productive working environment.

To mitigate queries and conflicts arising from GitHub code contributions, our team implemented a systematic reviewing scheme for committing and merging code into the main repository. This approach provided an avenue for all team members to offer their insights and suggestions on each individual's commit before integration into the official repository. By allowing thorough feedback from multiple perspectives, we effectively addressed redundant code and minimised conflicts within the codebase. This reviewing process not only enhanced the quality of our code but also improved collaboration and knowledge sharing among team members. As a result, our GitHub workflow became more streamlined and efficient, facilitating smoother project progress and reducing the likelihood of future conflicts.

Contributions Table

Team member name	Detailed contributions per chapter	Contribution Percentage Overall	Signature
Joshua Varney (up2138017)	Chapter 1: Problem specification: 100% Chapter 2: Design: 100% Chapter 3: Implementation: 100% Chapter 4: Testing: 100% Chapter 5: Critical analysis: 100%	100%	[illegible]
Rhys Parsons (up2125866)	Chapter 1: Problem specification: 100% Chapter 2: Design: 100% Chapter 3: Implementation: 100% Chapter 4: Testing: 100% Chapter 5: Critical analysis: 100%	100%	[illegible]
Harry Miller (up2120303)	Chapter 1: Problem specification: 100% Chapter 2: Design: 100% Chapter 3: Implementation: 100% Chapter 4: Testing: 100% Chapter 5: Critical analysis: 100%	100%	HMiller
Khadija Baffa (up2157506)	Chapter 1: Problem specification: 100% Chapter 2: Design: 100% Chapter 3: Implementation: 100% Chapter 4: Testing: 0% Chapter 5: Critical analysis: 100%	60%	[illegible]
Matthew Bowers (up2112240)	Chapter 1: Problem specification: 100% Chapter 2: Design: 100% Chapter 3: Implementation: 100% Chapter 4: Testing: 100% Chapter 5: Critical analysis: 100%	100%	[illegible]
Cindy Murimi (up2060987)	Chapter 1: Problem specification: 0% Chapter 2: Design: 0% Chapter 3: Implementation: 0% Chapter 4: Testing: 0% Chapter 5: Critical analysis: 0%	0%	[illegible]