Table of Contents

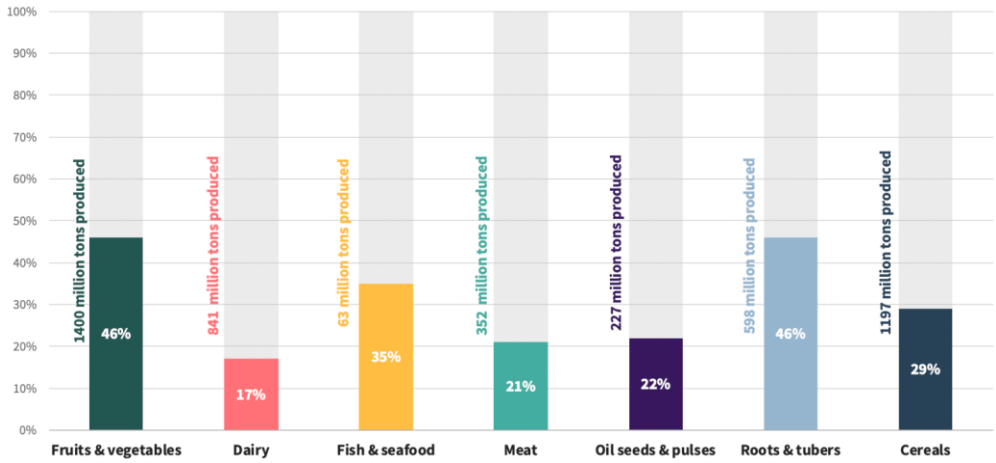
1 Background

Problem Statement: Home cooks lack resources and guidance when it comes to home cooking and meal preparation.

This problem statement became more relevant during the pandemic as it revived the dying trend of home cooking. This sudden rise in the number of people cooking at home has opened the market for applications to guide home cooks, especially the amateur users.

Research on Problem Statement:

1. A study conducted in 2013 by the Nutrition Journal has stated that there is a 16% at best and a 30% drop at the worst drop in people cooking and eating food at home (Tamarkin, D, 2017). Over the years, there has been a massive drop in food preparation at home.
2. Based on a survey conducted by ScienceDirect which had over 2400 respondents, the Covid lockdown led to a massive 42% rise in home cooks (ScienceDirect,n.d).
3. Many users are also facing difficulties when it comes to managing groceries and food wastage. With mismanagement of resources in the kitchen, many tend to overspend on groceries, waste food and even spend extra on transportation to get the resources. This eventually adds up to the overall expenditure per month causing many to deter from home cooking (Kliff,S, 2014)).
4. Lack of home cooking not just creates problems for individuals but also for the environment in general. Every year, an estimated 1.3 billion tonnes of food which is roughly one-third of the food produced for humans to consume goes to waste (ThinkEatSave, 2013).



This chart from TooGoodToGo shows the type of food waste that goes to waste and many of which are fresh produce (Roogoodtogo.com, 2016).

With one of the key problems being food wastage, users can key in the ingredients that they already have in their kitchen to find recipes. With this, the application will tackle the problem of food wastage. With this, users do not have to waste money on buying excessive ingredients which will go to waste if not used. This is especially applicable to fresh produce which will spoil quickly.

1.1 Introduction to our Application

HomeChef was primarily designed to help cooks of any experience be able to have a platform to find recipes and cook dishes based on the ingredients that are already available to them. This can also help to promote home cooking. With additional features like newsletters, videos, and calorie counters, users will not only be able to have more interactive, guided steps when cooking but also be able to keep track of their health.

For our application, we chose to focus on healthy eating due to the increasing need to advocate healthy living and lifestyle. We decided to include the nutrition details in all the recipes so that users are aware of the nutrition intake of the dishes they cook. Not only so, but a calories counter

is also included in the application for users to calculate the calories intake of what they had consumed. The reason why most people choose not to improve their diets even though they know about the importance of healthy eating is because they do not know the nutritional intake of the dishes and they are unsure of how to cook with their favourable ingredients. Hence, our application allows users to also be able to select their own ingredients and search up for matching recipes.

1.2 Scope of Application

The following explains the scope of the application.

1.2.1 Explore page

The Explore page consists of newsletters relating to healthy eating and food-related articles. The application's most rated recipes are also featured. There is a "View all" button where upon clicking on the button, users will be directed to the recipes page where all the recipes are categorised.

1.2.2 Login Page

The login and sign-up pages are pages that contain security features for users to access their accounts. The user's email will be used as the username for displaying on the profile page. Both pages are interconnected, as they can change between both pages easily with the link below the button.

The signup page is for first time users to register their account. The user's email will be used as the primary key and username for convenience, as people rarely change their email address.

The login page is a security feature for users to access their account and it requires the user's email and password for authentication.

1.2.3 Ingredient Search Page

In this page users can search for recipes based on ingredients they have at hand. They begin by selecting the ingredients that they already have. They can choose ingredients from different categories such as vegetables, fruits. Upon selecting all the ingredients, they can click the proceed button which will match the related recipes based on the ingredients selected.

1.2.4 Category Search Page

Recipe Search page consists of different tabs categorised to the various types of cuisines. These categories make it easier for users to search for their type of cuisines. Upon clicking on the categories, the application will redirect users to recipes from that category.

1.2.5 Calories Counter Page

Calories counter page lets users to search and add in the ingredients based on per serving. Following which it calculates the total calories intake.

1.2.6 Shopping List Page

The Shopping list page helps users prepare for the list of ingredients they need to purchase. The users can search the ingredients they require and add them to the list. They can also add the ingredients directly from the recipe page. The users can also edit and delete ingredients from the list. This provides a very integrated experience for the user.

1.2.7 Profile Page

Profile page allows users to access the login page and submit feedback on the application. For all the recipes that users favourited can be found in the collection page.

1.3 Aim of Project

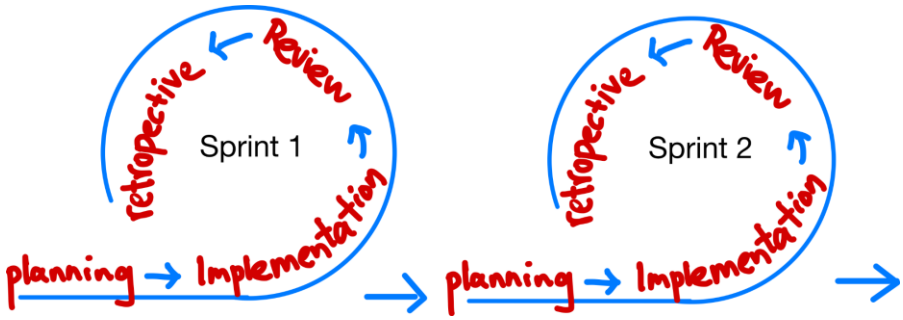
The main aim of our project is to create a meal prep application, HomeChef. This main purpose is to provide users with recipes based on the ingredients they have at hand. In addition to this, users can view weekly cooking newsletters, calculate their calorie intake and discover various new cuisines. This application would follow the stipulations mentioned in the midterm report. This would consist of codes for all the different pages, (GUI), and a record of how the application function. As well as, this report includes the planning process, analysis, evaluation, design specifications, prototyping and more.

1.4 Methodology that the team has decided

In our project we have decided to use the SCRUM framework and User-Centric Design(UCD) as mentioned in our midterm report. The SCRUM framework allows us to continuously modify and evolve our application based on users' inclinations and differing situations. We would be using sprints to help us keep track of our progress and record any changes made during the entire process.

1.4.1 Sprints

Sprints in agile are to break down a project into digestible chunks of time in which smaller tasks can be accomplished. When an agile team works on a project for a very long period it will become tedious, and they will feel are not making any progress even if they do. So, as an agile team, we are breaking down the project into manageable portions which allows us focus on our goals. This enables us to clearly know the progress we have done, to improve the efficiency and the quality of project. Although each sprint is planned individually, but the number and length of sprint should be decided at the outset. (Wrike.com, 2022)



Sprint in scrum

1.4.2 Why did we choose UCD?

We chose User-Centric Design(UCD) as our application revolves largely around user satisfaction. The application is community oriented as it requires the users to engage with our application to cater to their specific needs.

UCD can directly interact with users at an early stage, this can help to reduce the cost and save our working time. Besides this, UCD can also build a positive user experience, which increases user loyalty and boosts the reputation of the application against competitors. This also boosts competitiveness, which is one reason for choosing UCD. As through this, we will be able to interact with potential users and know their needs. Using this knowledge, we can make more improvements to ensure our products are up to date and remain relevant in the market. This enhances the chances of users choosing our products over the competitors.

We want our application to be easy-to-use and be of high quality. High quality in terms of providing a large database of recipes and ingredients for the users to choose and search from. As well as great UI/UX design for an easy-of-navigation for the user. Furthermore, to make our application more user-friendly, we decide to utilise an iterative development and test the application frequently. This is to ensure the application is easy to navigate for beginners and satisfy the needs of the expert planners as well.

1.5 Brief summary of group members

Product Owner	Kelly
Scrum Master	MingLin
Product Team	Kelly, Mingling, Wei Xian
Report Team	Kiran, Mukil, Wee Jie

1.5.1 Product Owner

Product Owner ensures that the collective vision and the goals towards the final product is always maintained and achieved. The Product Owner not only guides the team mates, but also considers and uses the end-user feedback to determine the next best action that can be taken to solve any issues and to keep the development of the project on track.

1.5.2 Scrum master

The Scrum master helps the product owner to convey information to the team and also leads the products team by ensuring that the tasks are executed properly and the key milestones are met. The Scrum master also holds the regular scrum meetings and aids in optimising backlog planning.

1.5.3 Team

The team operated in two different groups working in tandem to make sure that the product and report was progressing on task. The two teams include the Product team and Report team. The product team focused primarily on the application and the report team focused on the reports and testing of the application.

2 Planning and Research

2.1 Research undertaken to substantiate the project

Our mobile applications for the project are created and designed with React Native, Expo framework and Google Firebase for our database.

React Native and the Expo framework was chosen as the codes are applicable to both Android and IOS, hence only one set of codes is needed for the two different operating systems. This helps to reduce the time needed to build an application that is suitable to run on both Android and IOS. Not only so, using React Native with the Expo framework allows applications to reload immediately after changes are made. This helps developers in easing the testing and debugging process. In addition, React Native consistently released updates which allows developers to further improve the mobile application in the future.

Google's Firebase was chosen to manage all the stored data in the database. Also, with firebase authentication, it allows users to sign in with various options such as email and phone authentication. Most importantly, firebase is free of charge for the backend server. Therefore, it is one of the best options to use it as our backend. At last, React Native supports all services that Google Firebase had, this made the linking of our mobile application and the backend server much more convenient.

The scope of our project is as follow:

- Needs to have a sign-up page for new users
- Needs to have a login page for existing users
- Needs to have a search button when viewing multiple recipes and ingredients list
- Needs to have checkbox for user to track on the ingredients in recipe and shopping list
- Needs to have a button that enable user to edit the details for the shopping list item

- Needs to have a button that allows user to add the ingredients list to the shopping cart immediately (allows user to select or deselect)
- Needs to a counter for user to adjust the serving size when doing calories count and recipes
- Needs to have a collection page for user to view the saved recipes
- Needs to allow user to do multi select when selecting ingredients for matching recipe
- Needs to provide filter tab bar for the ingredient list when searching up for recipes with ingredients
- Needs to have a database to store all the content used for the application
- Needs to have a database to store all the customer's data

Our team came up with the following Gantt chart for the midterm report so that we can keep track of our progress.

Link to Gantt Chart :

[Agile gantt chart]

	TASK NAME	START	END	DURATION	MEMBERS
Analysis					
1	Brainstorming app Ideas	9 May	13 May	5	All
2	Define deliverable components	16 May	20 May	5	MingLin
3	Define scope	16 May	20 May	5	Mukil
4	Come up with specification	16 May	20 May	5	Mukil
5	Define time scale of work	23 May	27 May	5	Kelly
6	Define stakeholders	23 May	27 May	5	Kiran
7	Find challenged in domain	23 May	27 May	5	Kiran
Planning					
1	Define approach	30 May	3 Jun	5	WeiXian&Weejie
2	Market analysis	30 May	3 Jun	5	Kiran&Kelly
3	Wireframe design	30 May	3 Jun	5	Kelly&MingLin
4	Prototype	3 Jun	10 Jun	7	Kelly&MingLin
5	Assumption testing	13 Jun	17 Jun	5	Mukil
6	Finalise proposal	20 Jun	24 Jun	5	All
7	Design Data	27 Jun	30 Jun	4	All
Implementation					
1	Profile page	29 Jun	5 Jul	7	WeiXian
2	Login and Sign up Page	29 Jun	5 Jul	7	WeiXian
3	Home Page	4 Jul	10 Jul	7	Kiran
4	Recipe page	11 Jul	17 Jul	7	Mukil
5	Search page	11 Jul	17 Jul	7	MingLin
6	Calories counter page	18 Jul	24 Jul	7	Kelly
7	Shopping list page	18 Jul	24 Jul	7	Weejie
Testing					
1	Document issues	25 Jul	5 Aug	12	Kiran&WeiXian
2	Perform initial testing	1 Aug	5 Aug	5	MingLin
3	Correct issues	1 Aug	5 Aug	5	Mukil
4	Perform system testing	8 Aug	12 Aug	5	Kelly
5	Correct issues	8 Aug	12 Aug	5	Weejie
6	Finalise report	15 Aug	21 Aug	7	All

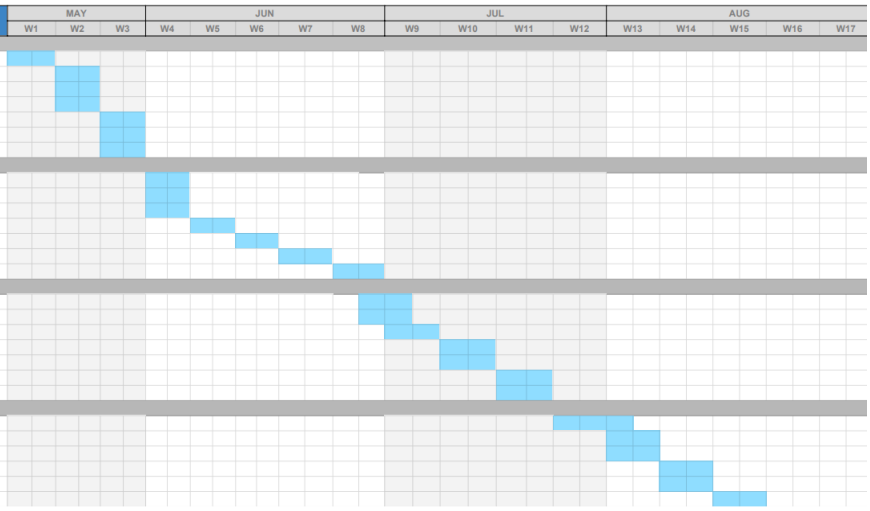


Figure 1:Gantt Chart

We had a better understanding of each other's ability after working together for midterms. Therefore, we rearrange our individual roles for the project development. Our team used a Kanban to provide a visual guide for progress tracking.

The Kanban format for our team is shown below and each box represents the main screen for each page. Sub function is listed in the sub task. Each box also consists of the subgroups that are in charge of that respective page.

Grey in the Kanban represents ‘Requested/Have not yet started’. Yellow in the Kanban represents ‘In Progress’ . Green in the Kanban represents ‘Done’

2.2 Kanban Board

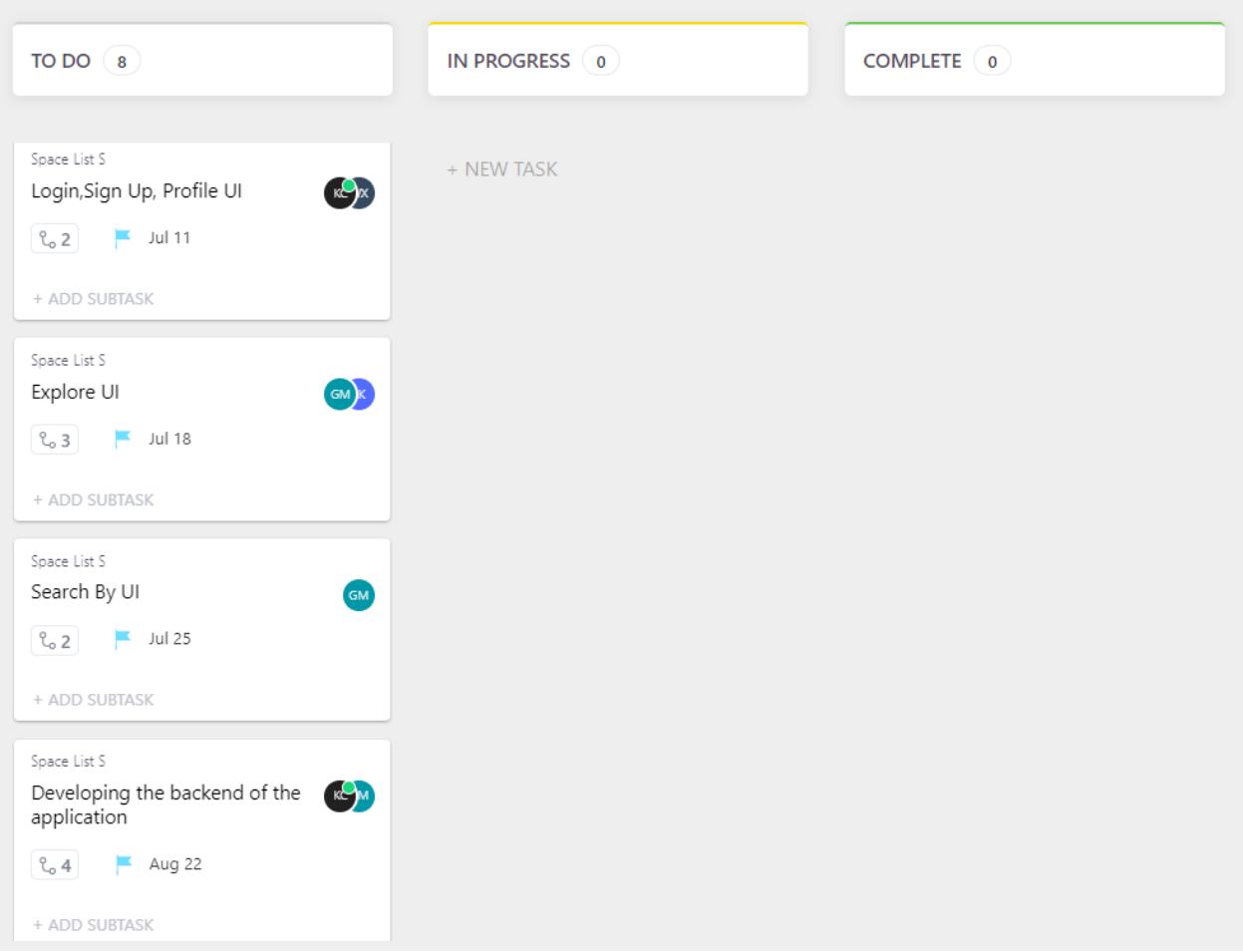
4 July

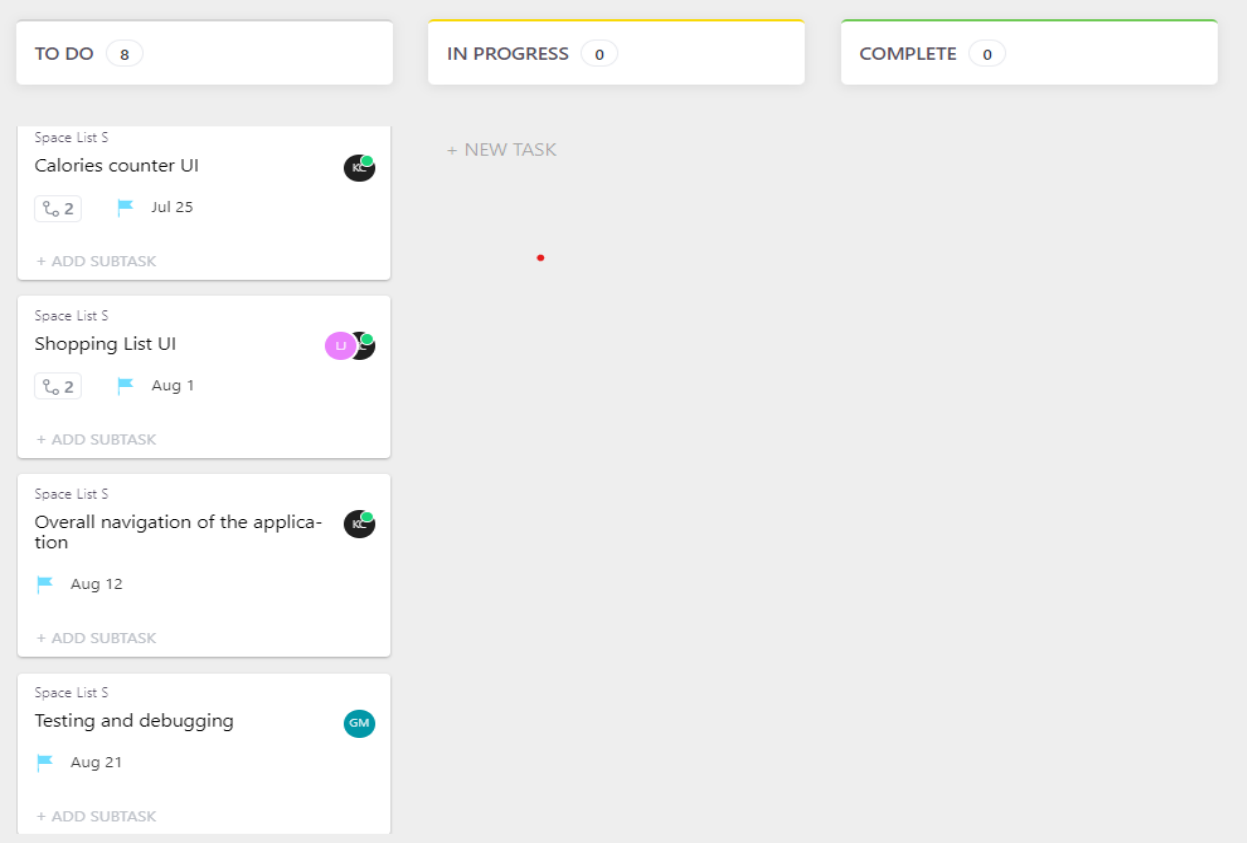
Kanban was used to monitor our progress. Each of us was assigned with individual tasks based on the Gantt chart and it was pinned on our Kanban.

We were broken up into subgroups according to our strength and competency, as well as the confidence for each task.

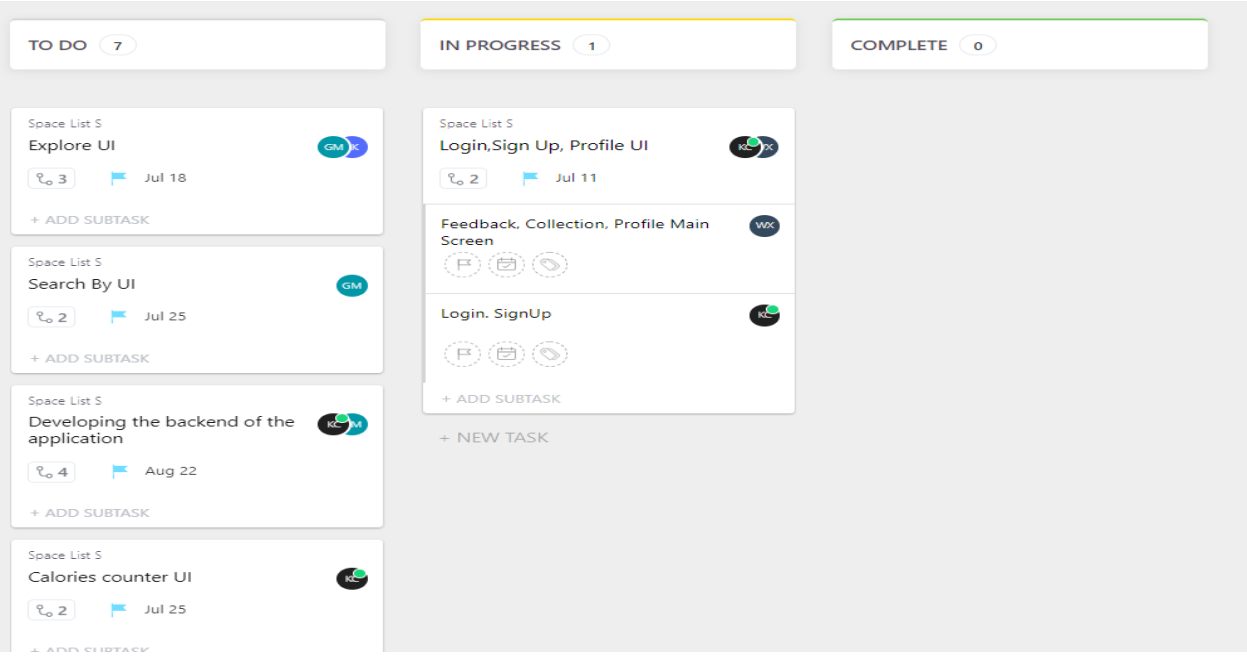
Coding was done on Visual studio code and codes were submitted to a shared repository on GitHub, so that everyone will have access to the changes of the code.

After assigning the tasks and deadlines, we proceed to code out the application. Our team is tasked to update the Kanban at the start of each week to monitor each other's progress.

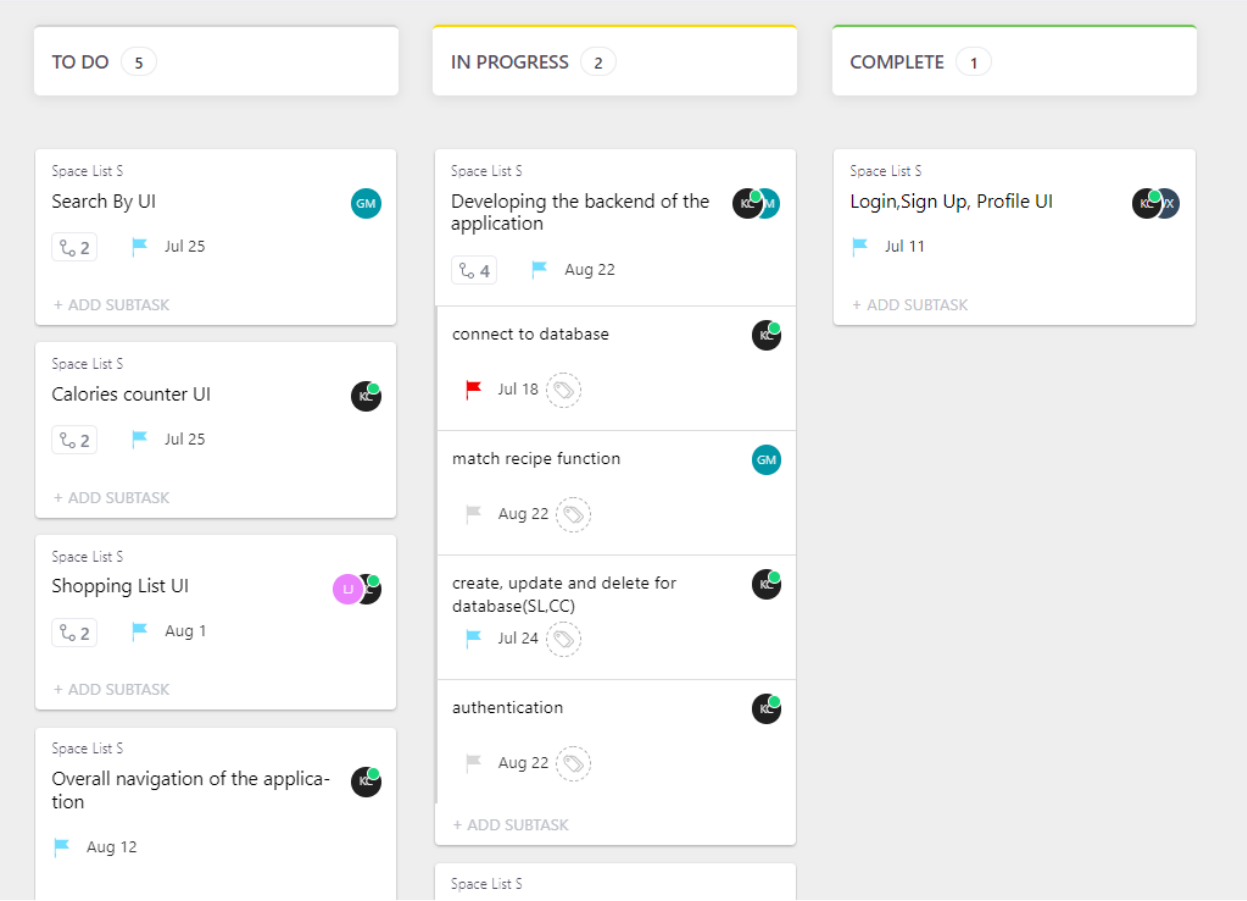


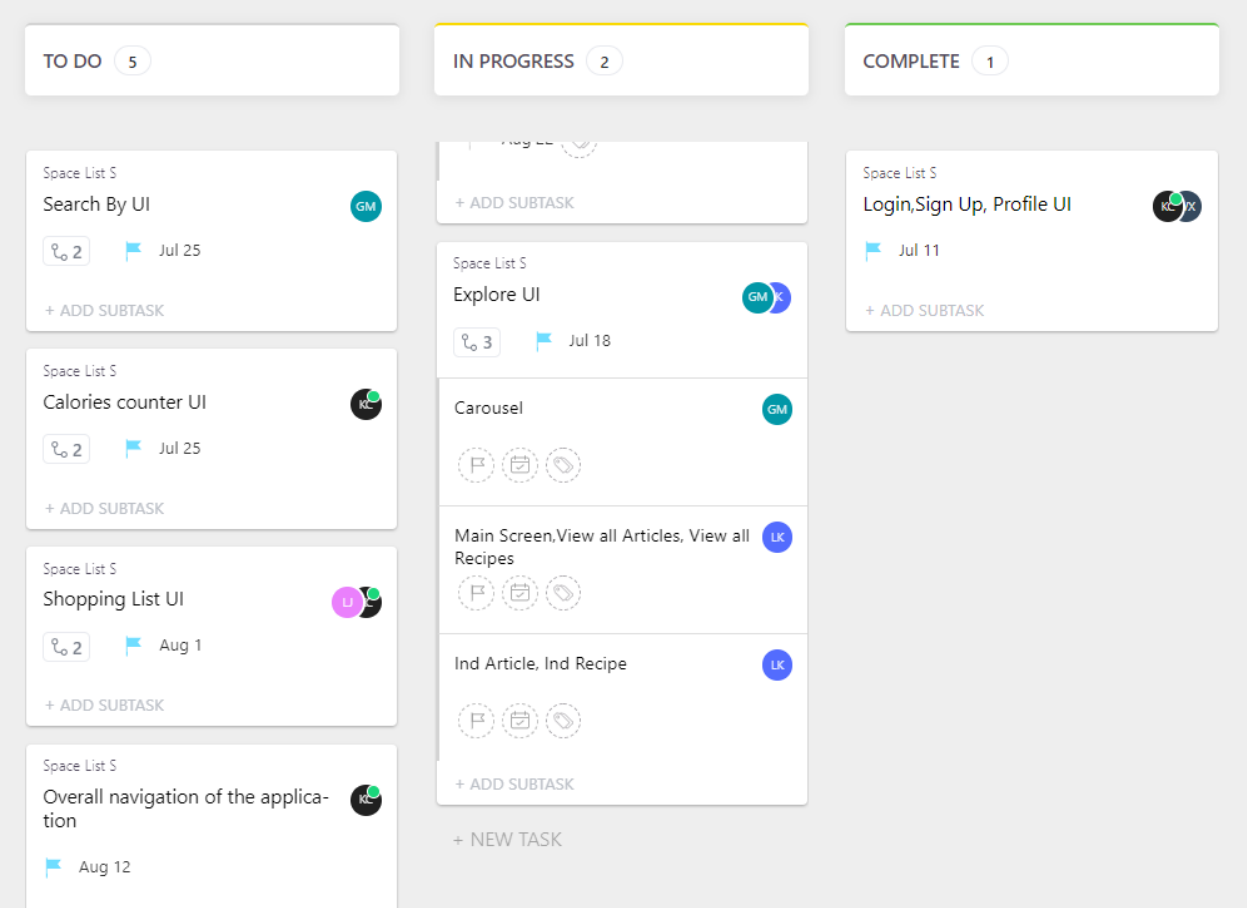


After taking a week to try and test out the react native code, our team had started on developing the UI for profile page.



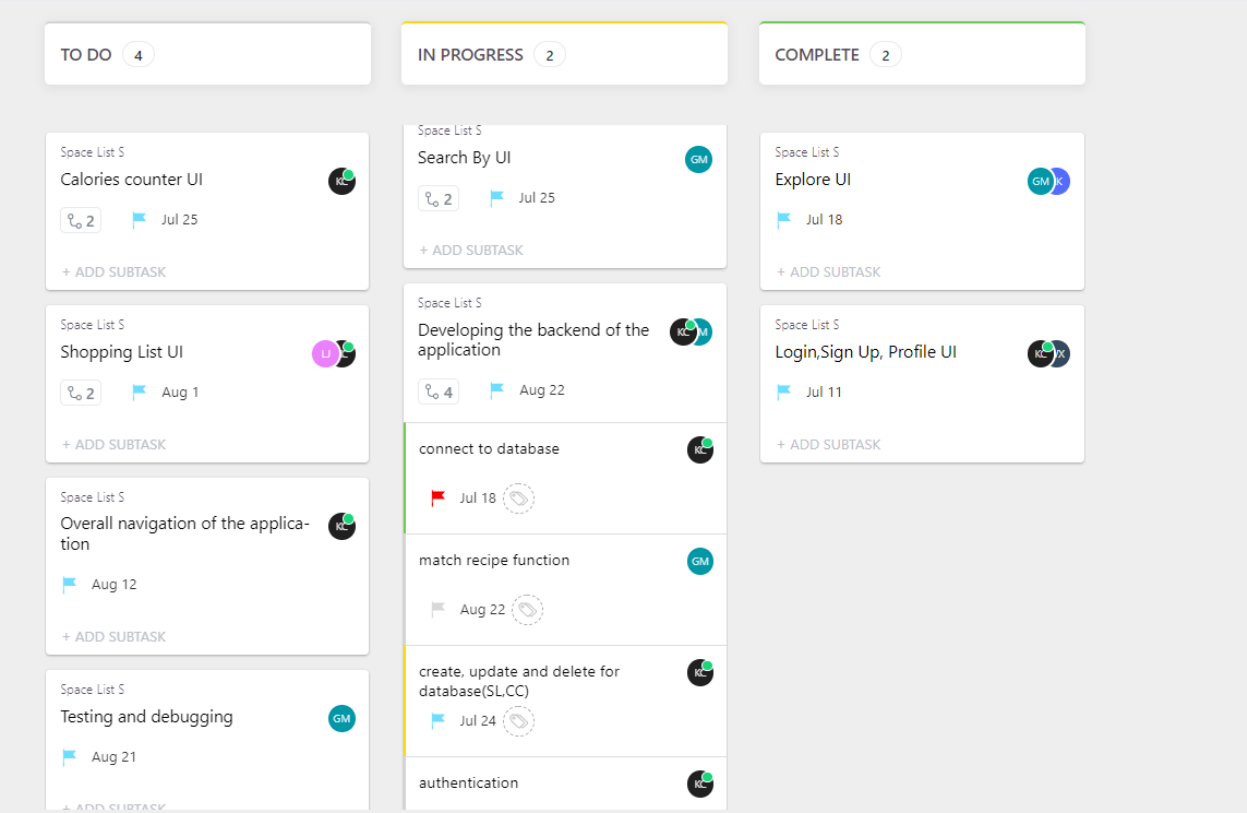
11 July


TO DO 5
IN PROGRESS 2
COMPLETE 1
Space List S
Search By UI
2
Jul 25
+ ADD SUBTASK
Space List S
Calories counter UI
2
Jul 25
+ ADD SUBTASK
Space List S
Shopping List UI
2
Aug 1
+ ADD SUBTASK
Space List S
Overall navigation of the application
Aug 12
Space List S
Developing the backend of the application
4
Aug 22
connect to database
Jul 18
match recipe function
Aug 22
create, update and delete for database(SL,CC)
Jul 24
authentication
Aug 22
+ ADD SUBTASK
Space List S
Space List S
Login,Sign Up, Profile UI
Jul 11
+ ADD SUBTASK



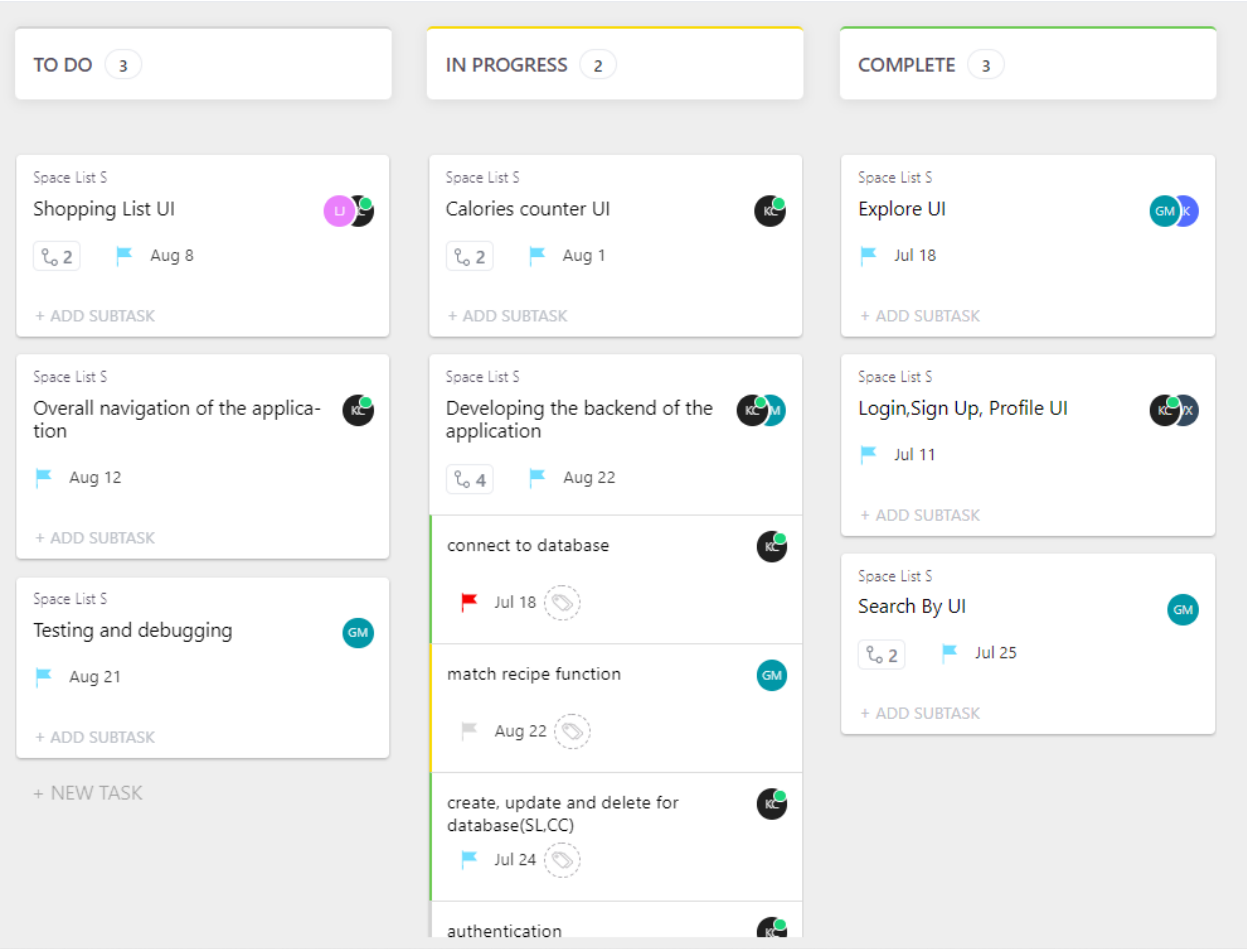
The user interface for login, sign up and the respective screens relevant to the profile page was completed in the first week. We were on track with our schedule. We start on the connection with the firebase while the others will continue to work on the UI for other pages.

18 July



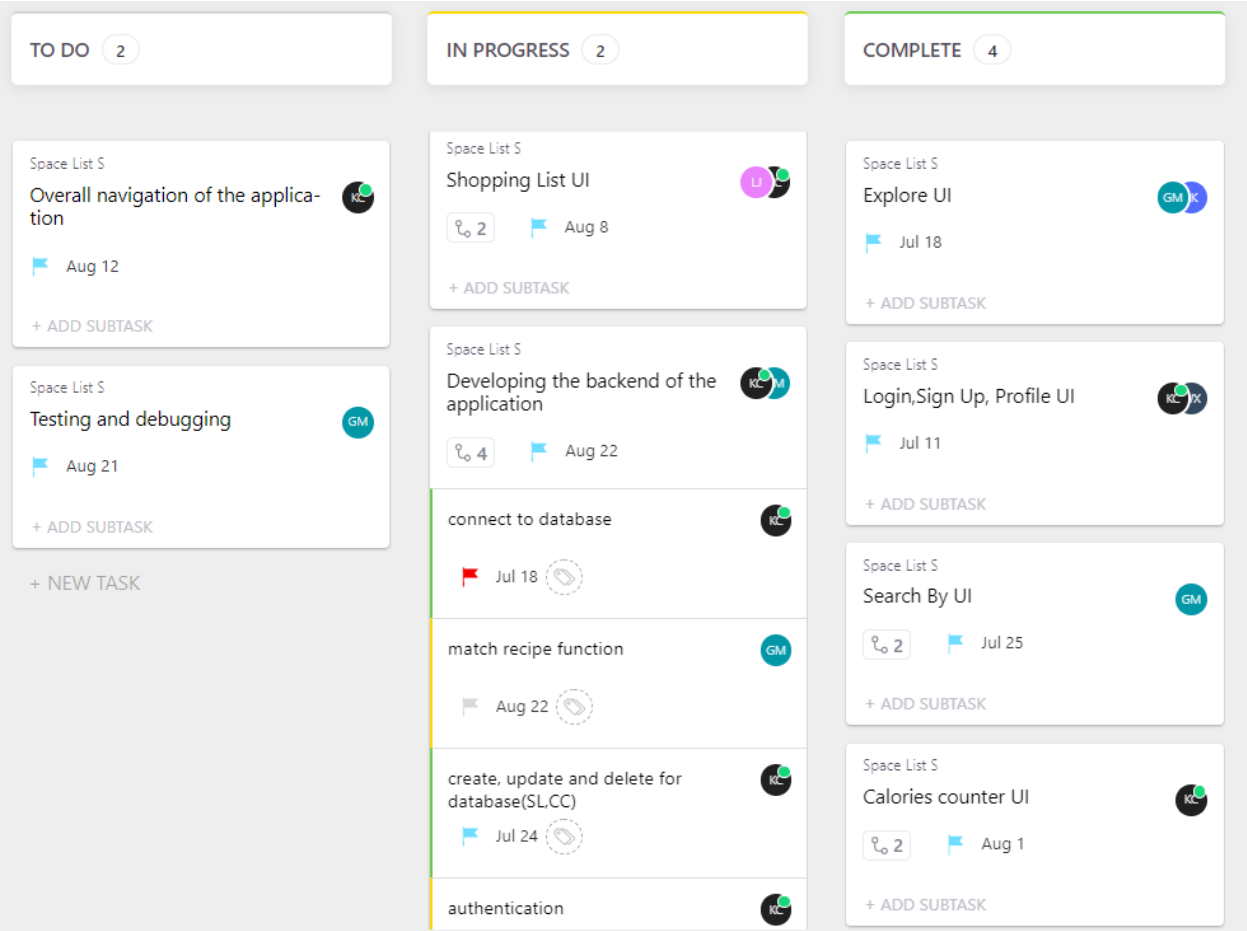
In our second meeting, we found ways to read the data from firebase in our react native app. However, some difficulties were found in combining the UI with the database. More time was required to complete. The others will continue to work the UI for the rest of the pages.

25 July



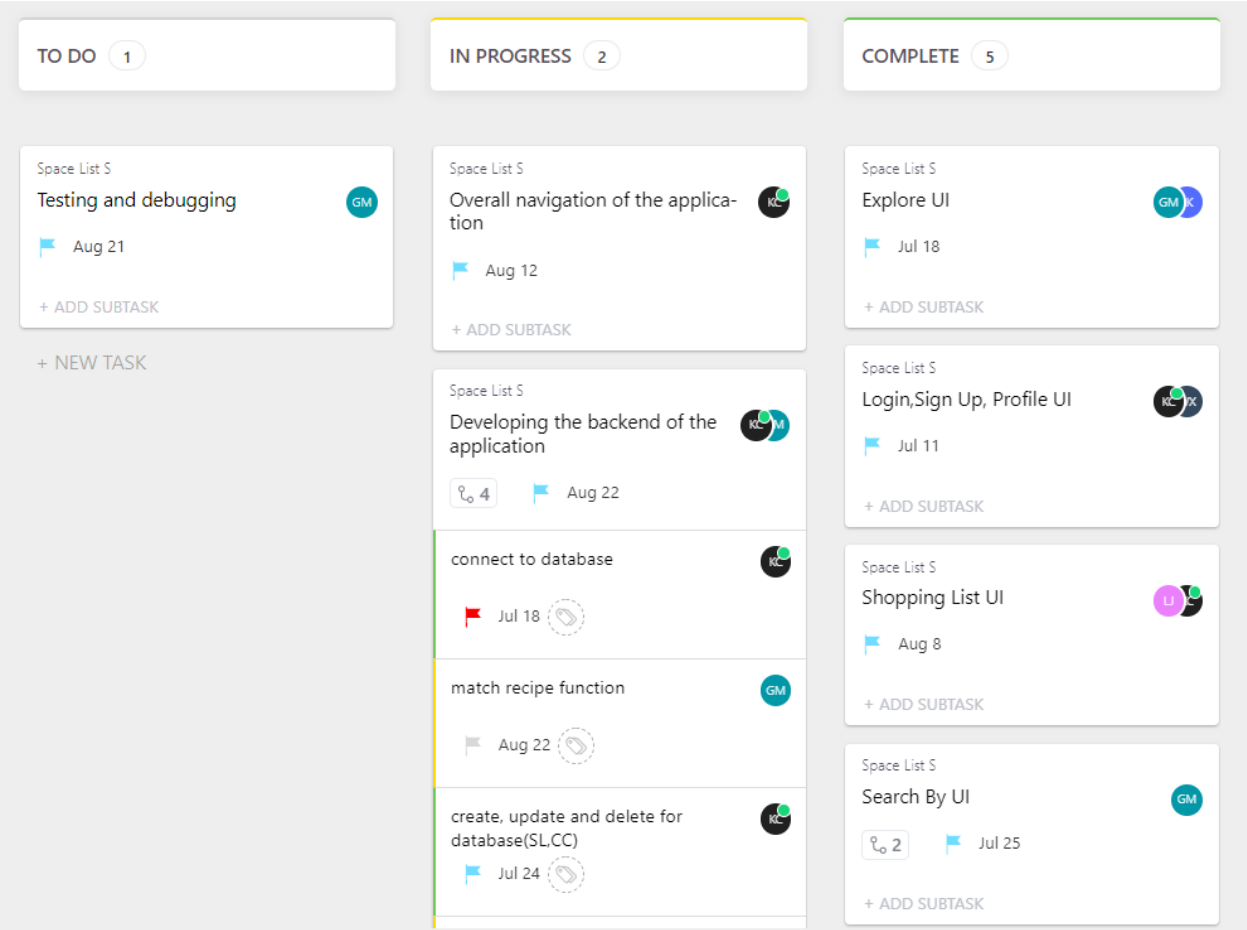
The UI of the for Explore, Profile and Search By are completed. However, the function for matching ingredients with recipes is not complete as we faced some difficulties in it. We managed to do the CRUD operations with firebase, and we integrated it in the calories counter page. We faced some difficulties in it, but we still had time before the deadline. Hopefully it will get done on time.

1 Aug



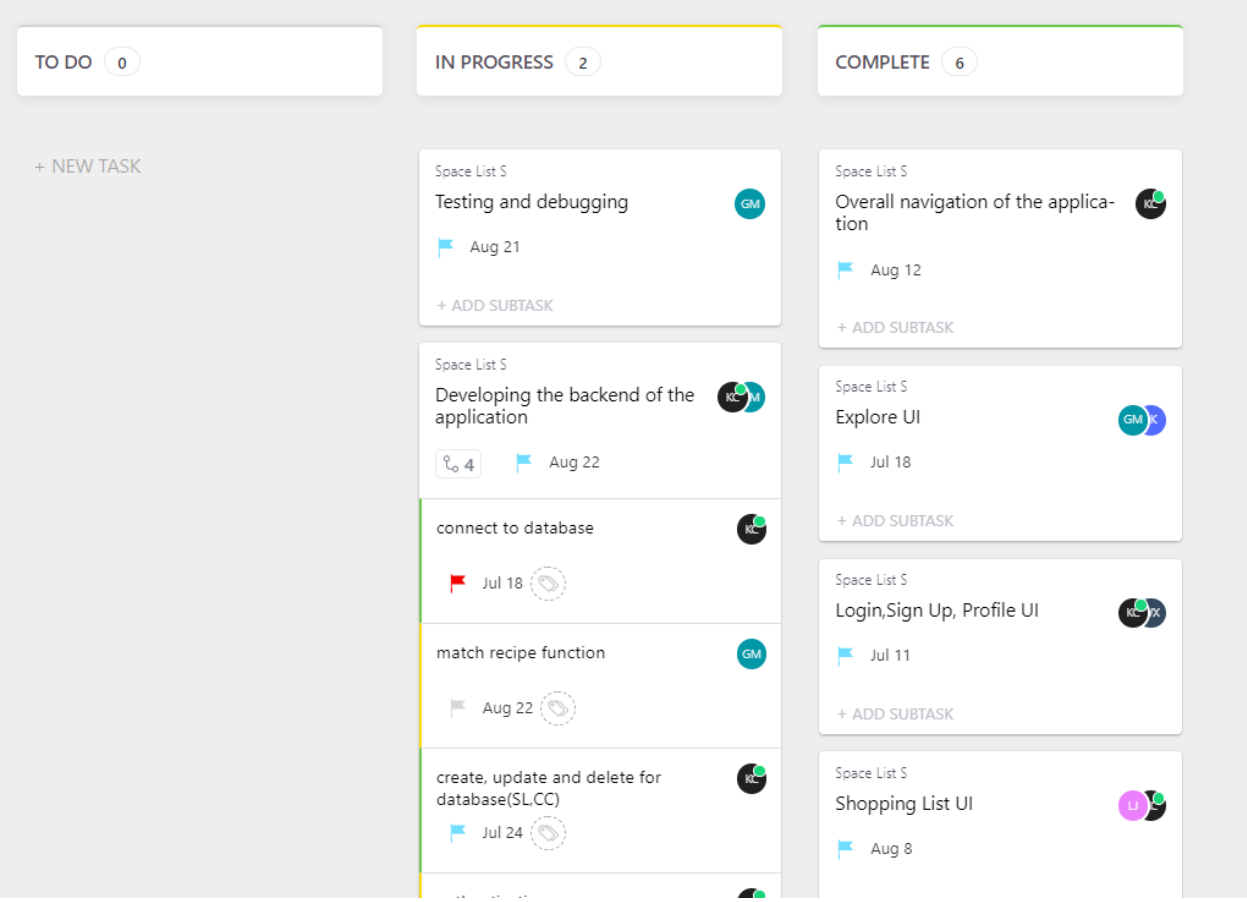
We managed to solve the errors for the calories counter page, and we proceed to use the same operation for the shopping list. Our team still did not manage to get the match recipes done; hence the rest of the teammates are helping out on searching up on the authentication and matching function.

8 Aug



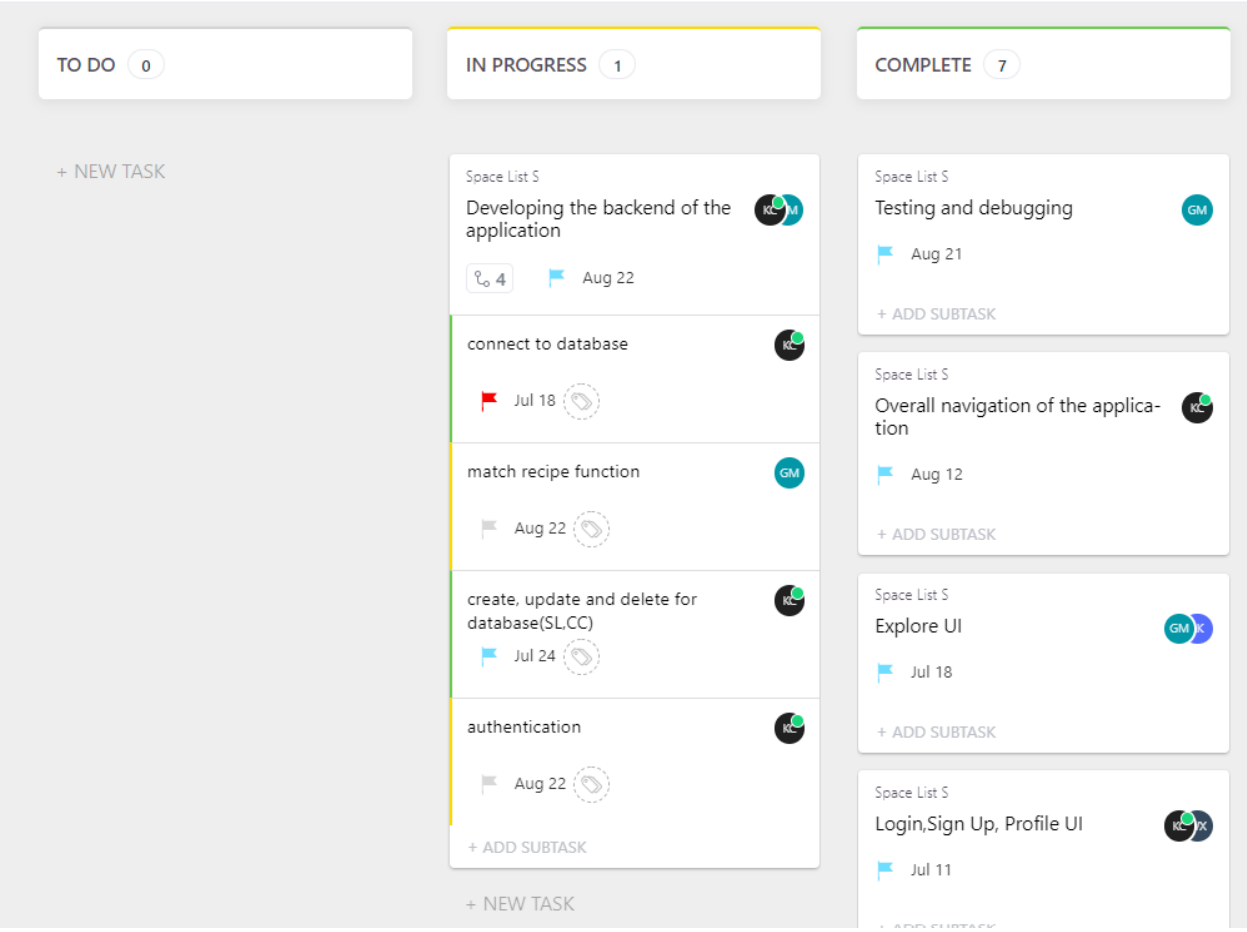
All the UI for the application is done, but the backend side still faces problems. For example, matching the ingredients to their respective recipes and authentication. Hence, we decide to continue to work on the navigation of the application since all the UI is done.

15 Aug



The overall navigation is done, and we start on testing some of the completed functions and debug if there is any error. While the others will continue to work on the reports and also the incomplete functions.

22 Aug



All the navigation and UI are done. There were not many errors to debug, hence we feel that we can complete the project by our stipulated deadline. However, more time is required to develop some of the backend logic of the application. Hence, our team decided to fall back onto our contingency plan, which is to use a static database of the items for the matching ingredients search function. This will simulate the process of the application and focus more on improving the user interface of our application.

28 Aug

The integration and the navigation of our whole mobile application was completed, and this is the end of our Kankan board. We proceed to let our users try out the application and make necessary changes based on the feedback received. Also, work on the documentation for the project with the remaining time.

2.3 Sprint Backlog

The sprint backlog are tasks that were identified by our team which we plan to complete during each scrum sprint. Our team came up with user stories that were deemed necessary to fulfil each story before the implementation stage. Each member was assigned tasks based on our strength. We estimated the time needed for us to complete our tasks and the deadline was set so that we can have buffer time to spare for our final report.

For higher priority tasks, members in charge are required to update the backlog daily on time spent doing the task and their progress. Team meetings were conducted weekly to check on each other's progress. Once the tasks are completed, members will mark the work as completed and write out the time frame they finished it. The Kanban is also updated accordingly.

After several discussions, our team decided that the development of the backend logic for matching recipes and authentication will be pushed to further development, beyond the final submission deadline. Any excess time will be spent on improving the user interface so that user engagement will increase.

ID	User Story	Tasks	Priority	Assigned to	Status	Estimated effort	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
1	As an administrator, I need to ensure that I have a database to store all my data used for the application	Create a database on Google Firebase	1st	Kelly	Completed (11/7-18/7)	9	0	0	2	1	0	0	0	0	0	0
		Configure our project and link it to the database	2nd	Kelly	Completed (11/7-18/7)		0	0	0	2	2	2	0	0	0	0
2	As a user, I need to be able to sign up and login to my account	Create sign up page user interface	1st	Wei Xian	Completed (4/7-11/7)	14	2	2	0	0	0	0	0	0	0	0
		Create login page user interface	2nd	Wei Xian	Completed (4/7-11/7)		0	0	2	2	0	0	0	0	0	0
		Code the backend part of login and sign-up page to be to send information to the database	3rd	Kelly	Further Development		NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
3	As a user, I need to view the all the available recipes in the application	Create Home page	1st	MingLin	Completed (11/7-18/7)	25	2	2	2	0	0	0	0	0	0	0
		Create View all recipes page	2nd	Kiran	Completed (11/7-18/7)		0	0	0	3	0	0	0	0	0	0
		Crete View Individual recipe page	3rd	Mukil	Completed (11/7-18/7)		0	0	0	0	2	2	2	0	0	0
		Create View all articles page	4th	Kiran	Completed (11/7-18/7)		0	0	0	0	2	0	0	0	0	0
		Crete View Individual article page	5th	Kiran	Completed (11/7-18/7)		0	0	0	0	0	2	0	0	0	0
		Code the backend of the Individual recipe page	6th	MingLin	Further Development		NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
4	As a user, I need to count the ingredients calories and adjust the serving accordingly.	Create Calories Counter Page	1st	Kelly	Completed (25/7-1/8)	13	2	2	0	0	0	0	0	0	0	0
		Create backend of the calories counter (searchable dropdown, counter)	2nd	Kelly	Completed (25/7-1/8)		0	0	3	2	2	2	0	0	0	0
5	As a user, I would like to search up for recipes with my available ingredients and cook the dish	Code Search by Ingredients page	1st	MingLin	Completed (18/7-25/7)	30	2	2	2	0	0	0	0	0	0	0
		Code Search by category page	2nd	MingLin	Completed (18/7-25/7)		0	0	0	2	2	0	0	0	0	0
		Code the backend of the matching recipes	3rd	MingLin	Further Development		NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
6	As a user, I would like to compile a shopping list of groceries to purchase on the next visit to grocery store	Code Shopping List page	1st	WeeJie	Completed (1/8-8/8)	9	2	3	0	0	0	0	0	0	0	0
		Code the backend of the adding item into shopping list and editing it	2nd	Kelly	Completed (1/8-8/8)		0	0	2	2	0	0	0	0	0	0
7	As a user, I would like to view on the saved recipes and send feedback on the application	Create Profile page	1st	WeiXian	Completed (4/7-11/7)	16	2	0	0	0	0	0	0	0	0	0
		Create collection page and feedback form page	2nd	Mukil	Completed (4/7-11/7)		0	2	2	0	0	0	0	0	0	0
		Code the backend of the collections page and feedback form page	3rd	WeiXian	Further Development		NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
8	As a developer, I need to ensure that my application is error free, and it can run smoothly for all users	Debugging and troubleshooting of the application to ensure that there are no errors or bugs	1st	MingLin	Completed (15/8-21/8)	11	4	4	3	0	0	0	0	0	0	0

3 Prototyping and Iteration

We began by creating 3 sets of prototypes before we started developing the final deliverable. The first prototype, the low-fidelity digital sketches were drafted out using concept sketching which contains the basic features of the application.

The second prototype, the medium-fidelity wireframes which were converted from the digital sketches into digital diagrams using Figma, making them prominent and presentable. This was essential to give potential users a clear representation of the application and gather feedback for improvements. These improvements can then be implemented before working on the final prototype.

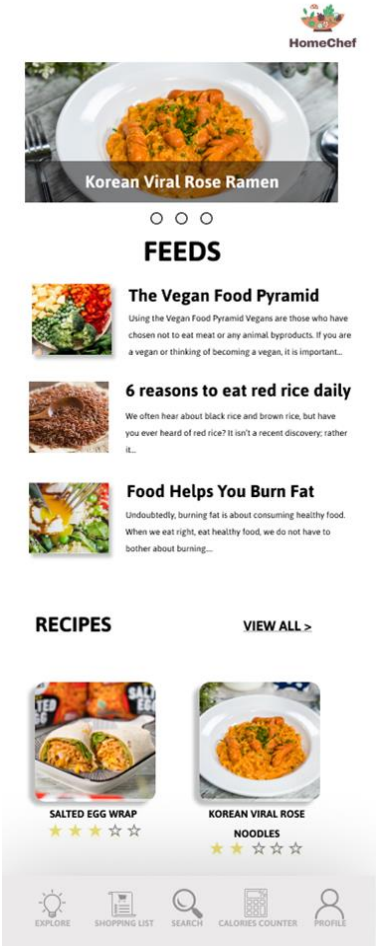
The final prototype, the high-fidelity prototype, is designed through Figma using the technique of clickable wireframes. This prototype is made by taking into consideration user feedback from the medium-fidelity wireframes. Thus, this provides a better illustration of our ideas for the application.

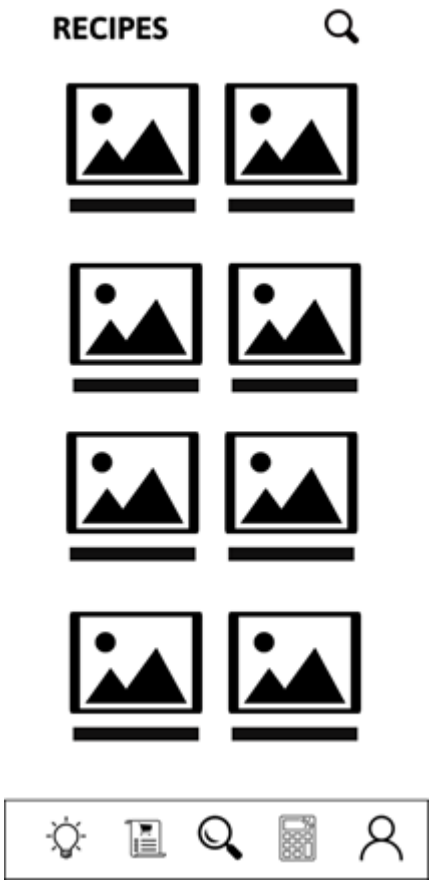
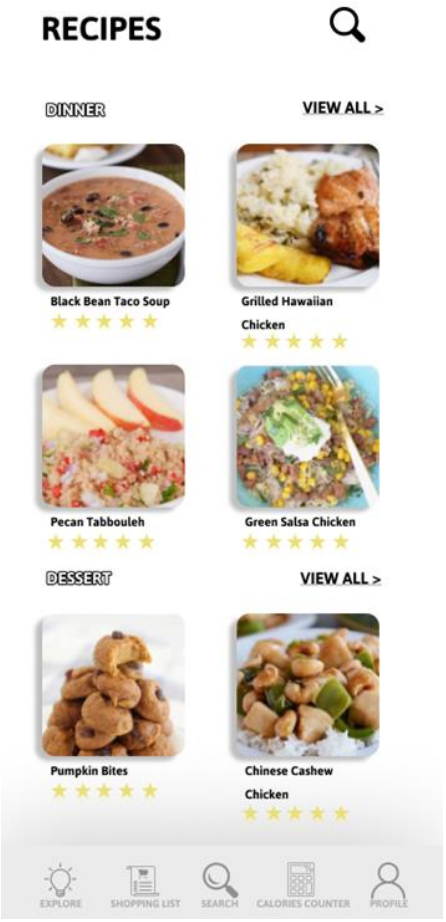
Before we began working one the high-fidelity prototype, we gathered some feedback on the medium-fidelity prototype. As mentioned in the document that we submitted in the midterm, there are a few main topics we have discussed were:

- The flow of the application
- User-friendliness
- Suggestions or improvements to be made

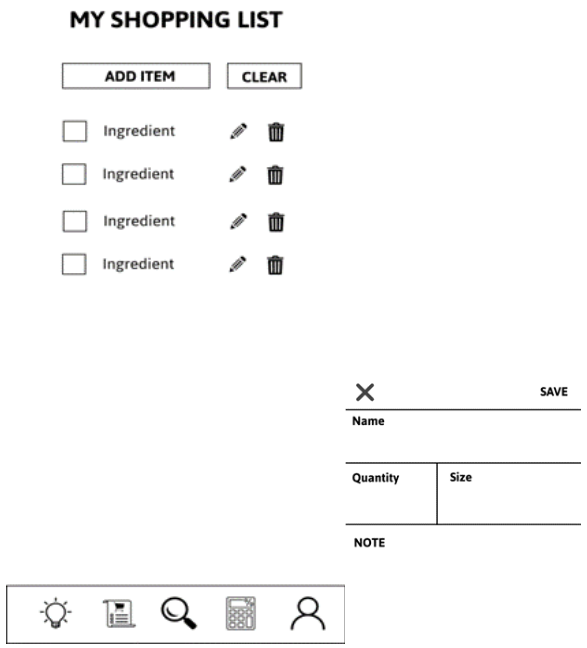
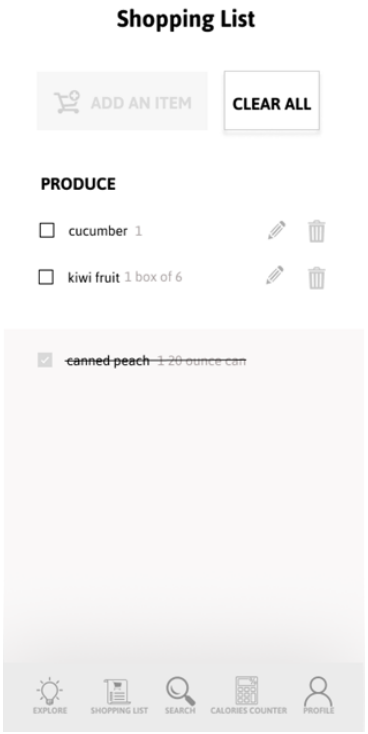
While the users appreciated the flow of the application, we have also gathered other feedback from them, so that we can clearly know our weaknesses and improve them.

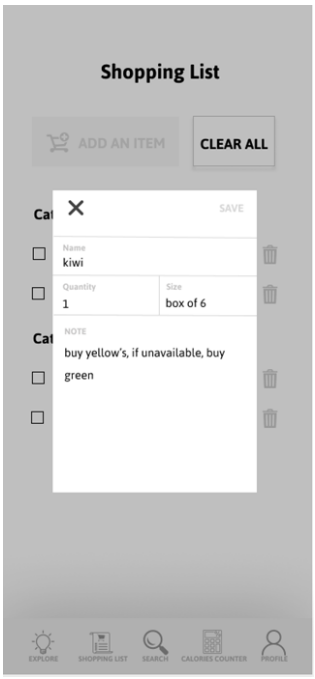
For example:

<table>
<tr><th colspan="2">Explore page</th></tr>
<tr><th>Before improvement</th><th>After improvement</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">User feedback:
- Provide ratings below each recipe
Improvement:
- We have added the ratings below each recipe.
- The number of stars shows the ratings of each recipe, there is a maximum of five stars. This will let the users efficiently choose the recipe that they prefer.</td></tr>
</table>

<table>
<tr><td colspan="2">Recipes page</td></tr>
<tr><td>Before improvement</td><td>After improvement</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">User feedback:
- Categorise the recipes for easier navigation

Improvements:
- We have categorised the recipe, so the user can easily choose the recipe that they want such as Main Dish, Side Dish, Dessert, and Appetisers.</td></tr>
</table>

Shopping list page	
Before improvement	**After improvement**
	

User feedback:

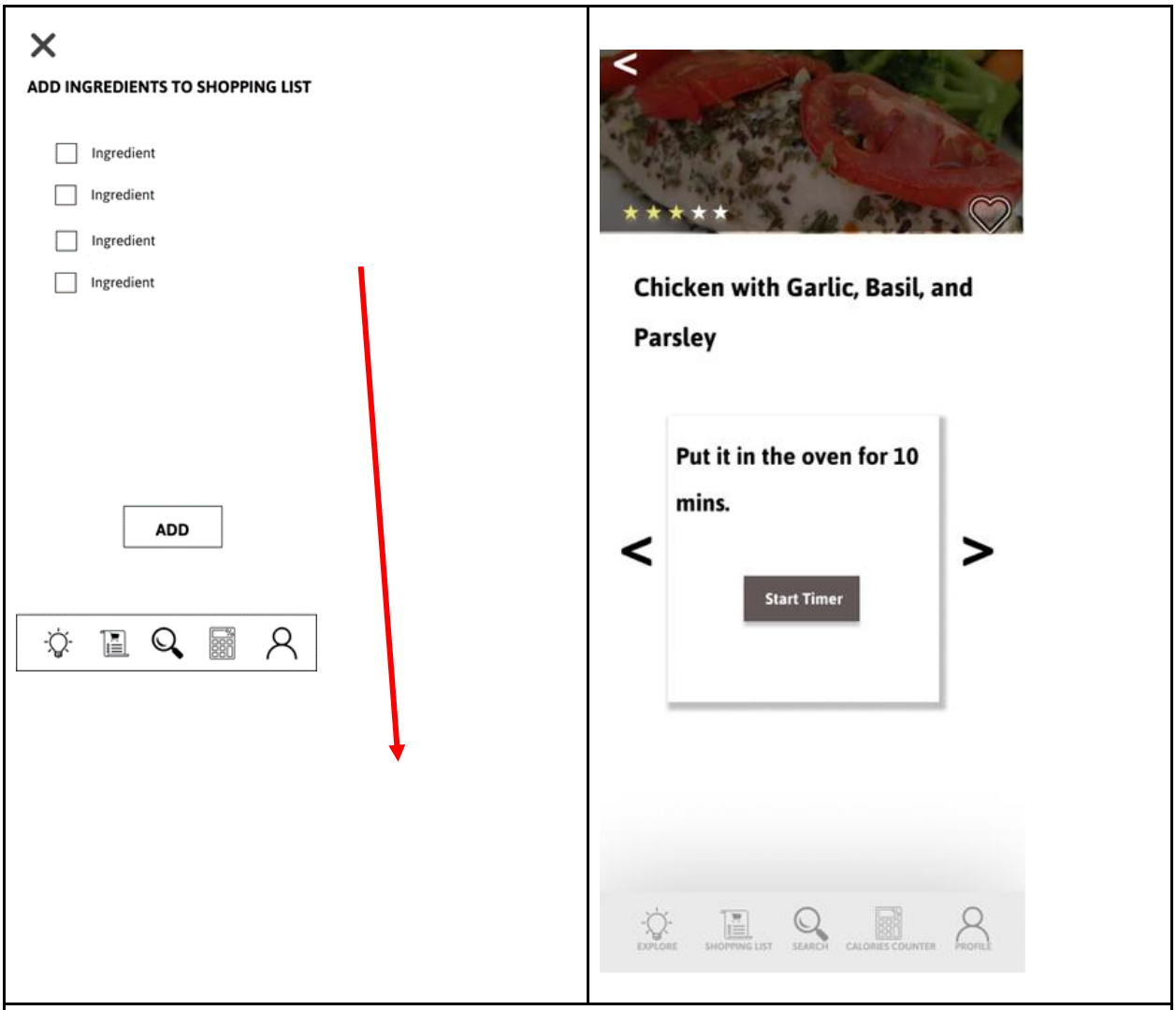
- Separate the list based on completed and uncompleted.

Improvement:

- We have added a function that can separate the list based on completed and uncompleted.
- The product that is marked as completed by the user will be separated, this will led the user to distinguish it at a glance.

Recipes page

Before improvement	After improvement
	



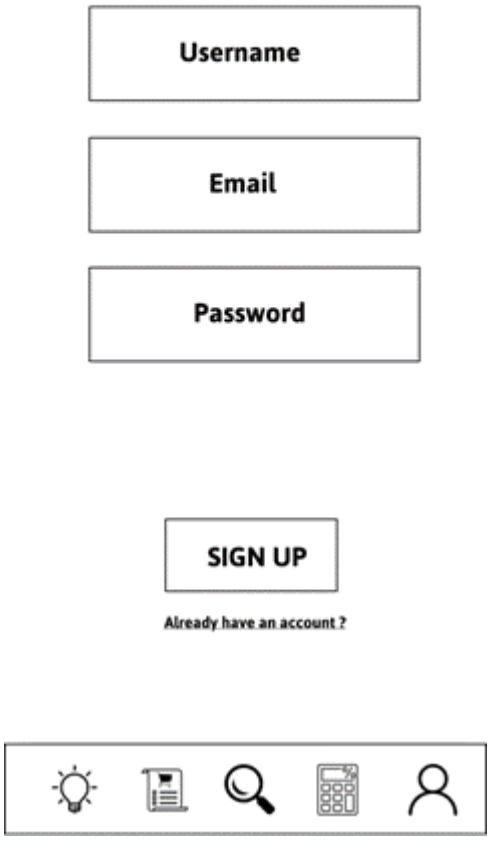
User feedback:

- Have a ratings or/and leave comments section
- Allow users to like the recipes
- Present the cooking steps in step by step format

Improvements:

- We have added the ratings and like function for the users, users are able to rate or like their favourite recipe.
- The cooking step also presented in step by step format, this is clearer to the users.

Profile page	
Before improvement	After improvement
	
User feedback: - Have a collection tab of saved or liked recipes Improvements: - The collection tab to view all the recipes that the user has liked is also implemented.	

<table>
<tr><td colspan="2">Sign Up page</td></tr>
<tr><td>Before improvement</td><td>After improvement</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">User feedback:
- Have a placeholder to double confirm the password when signing up

Improvements:
- A placeholder to double confirms the password was implemented, this will prevent the user lost their account by entering the wrong password during registration.</td></tr>
</table>

After we received much constructive feedback, we ended up having to prioritise some specs over others to deliver in the end. Here are the points we need to consider when submitting our application:

1. Our team has some basic knowledge of JavaScript, so we have chosen React Native as the framework because it is based on JavaScript. However, we don't have the relative knowledge to develop with it, so we need to consider the complexity of implementing it.

2. We need to allocate the time for learning the knowledge about React Native to make sure we can submit our final deliverable. Besides that, we also need to ensure our deliverable is bug-free and pass the test after user testing.

Therefore, our team has finalised some functional specifications, they are:

1. Allow users to register to the application

2. Allow the registered users log into the application

3. Allow users to rate the recipes inside the application

4. Allow users to read the recipe that they choose

5. Allow users to search the recipe by ingredient

6. Allow users to search the recipe by category

7. Allow users to add, remove and edit the shopping list

8. Allow users to calculate the calories

The technical specifications are as follows :

1. Platform to be used on iOS and Android
2. Framework for development : React Native and Figma framework
3. Language : JavaScript
4. Backend: Firebase

3.1 Testing of input and output during development

'Form validation' functionality

If the users didn't fill up the form's necessary fields, the application page will pop up a message that requested users to ensure all fields have been filled, otherwise, users are not able to submit the form.

Unit testing of all functionalities in the application

3.1.1 Test name: Unit testing of Login functionality

Test case #1	**Scenario**	**Steps**	**Expected output**	**Actual Output**	**Pass/ Fail**
1.1	User will be able to login at login page	1. There are 2 boxes to fill in. Email and Password. That is for people that have already signed up. 2. There will be a login button. Click the login button after the email and password is filled.	Pop up message shows that the user had successfully logged in. Users will be directed back to the explore page.	As expected	Pass

1.2	Users will not be able to submit if there are blank fields to login.	1. Nothing or Invalid data is filled in email or password.	An error message will return replying ‘ **Password is a required field**’ or ‘ **Email is a required field**’ depending on which box is empty.	As expected	Pass

3.1.2 Test name: Unit testing of Sign-up functionality

Test case #2	**Scenario**	**Steps**	**Expected output**	**Actual output**	**Pass / Fail**
2.1	The user can sign up on the app	1.Tap the **“Do not have an account?”** hyperlink. 2. Type the Username, Email, Password into the respective fields. 3.Tap the **Sign-Up** button.	Once all the fields have been filled and the **Sign-Up** button has been tapped by the user, the user is directed to the explore page	As expected	Pass

2.2	Blank fields are not accepted to Sign Up	1. Click the **Sign-Up** button without typing any information in the fields on the sign up page.	Display the error messages "**Username is required**", "**Email is required**", "**Password is required**".	As expected	Pass
2.3	The invalid format will be rejected	1. Type the invalid format email (eg. without the @hotmail.com) in the respective fields	Display the error message "**Invalid email**"	As expected	Pass
2.4	The password must at least 8 digits long	1. Key in the password at least 8 digits in the respective fields	Display the error message "**Password must at least 8 digit**"	As expected	Pass

2.5	The confirmed password is the same as the password typed above	1. Key in the same password at the confirmed password input box	Display the error message "**Password is not the same**"	As expected	Pass

3.1.3 Test name: Unit testing of Recipe.js

Test case #3	**Scenario**	**Steps**	**Expected output**	**Actual Output**	**Pass/ Fail**
3.1	User can like the recipe at the receipt page	1. Clicking the like button on the picture. It will save the recipe to the collection tab.	Once the like button is triggered, a pop out message will be shown with the message "**Added to collection**"	As expected	Pass
3.2	User can rate the recipes	1. Drag the rating stars to rate the recipes	When user released the drag, a pop out message will be shown with the message	As expected	Pass

			“**Thank you for rating**”		
3.3	User can adjust the serving size	1.Click on the counter to adjust the serving size	Clickable counter	As expected	Pass

3.1.4 Test name: Unit testing of ShoppingList.js

Test case #4	**Scenario**	**Steps**	**Expected output**	**Actual output**	**Pass / Fail**
4.1	The users can add new ingredients into the shopping list	1.Click on the add ingredient tab 2. Search the specific product on the shopping list page 3. Select the specific product.	Item is added to the respective collection in firebase	As expected	Pass
4.2	The users can clear all items in shopping list	1. Tap the **CLEAR ALL** box on the shopping list page	Remove all the items in the shopping list and in firebase	As expected	Pass
4.3	The users can edit the shopping list details	1. Tap the edit icon beside the items in the shopping list. 2. Edit the Name, Quantity, Size and Note	The data in firebase is updated and shows the updated details of the items after save at	As expected	Pass

		3. Press Save after done editing	the shopping list		
4.4	The users can remove the selected items	1. Swipe right and click on the delete button	The data in firebase is removed and the selected items will be removed on the shopping list	As expected	Pass

3.1.5 Test name: Unit testing of CaloriesCounter.js

Test case #5	**Scenario**	**Steps**	**Expected**	**Actual Output**	**Pass/ Fail**
5.1	Count the total calories for the ingredients added to the list	1.Click on the add ingredient tab 2. Search on the ingredients 3. Select the ingredient. 4. Adjust the serving size of the ingredients added	Total calories are shown with adding up all the calories of the ingredients added	Expected	Pass
5.2	Customise each of the ingredient calories	1. Click on either add and minus button to adjust the serving size.	The calories shown for each ingredient is adjusted according to the serving size. Serving size cannot be decreases when serving size reaches 1	As expected	Pass

5.3	The users can remove the selected items	1. Swipe right and click on the delete button	The data in firebase is removed and the selected items will be removed on the shopping list	As expected	Pass

3.1.6 Test name: Unit testing of Profile.js

Test case #6	**Scenario**	**Steps**	**Expected**	**Actual Output**	**Pass/ Fail**
6.1	Clicking my collection	1. Clicking on the My collection will change the page to my collection page.	Contain the list of recipes added to the collection page	As expected	Pass
6.2	Clicking on to login	1. Clicking on the login page will allow users to access the login page.	It will contain a hyperlink allow users to quickly access to their login page	As expected	Pass
6.3	Clicking Feedback	1. Clicking on the login page will allow users to access the feedback page.	It will allow users to send us feedback.	As expected	Pass

3.1.7 Test name: Unit testing of Explore.js

Test case #7	**Scenario**	**Steps**	**Expected output**	**Actual output**	**Pass / Fail**
7.1	The users can read the feeds	1.Tap any feeds on the explore page	Shows the respective feeds to the users	As expected	Pass
7.2	Users can view all the recipes or all the articles respectively	1. Tap the view all hyperlink on the explore page	Shows all the available recipes or articles to the users	As expected	Pass
7.3	Access the content of the recipes or articles listed	1. Click on the item to access the content	Content of the selected item is shown	As expected	Pass
7.4	Swipe left or right the carousel	1. Swipe left or right 2. Click on the dot to access the respective item immediately without swiping	Swipe able carousel	As expected	Pass

3.1.8 Test name: Unit testing of SearchByCat.js and SearchByIngre.js

Test Case #8	**Scenario**	**Steps**	**Expected output**	**Actual output**	**Pass / Fail**
8.1	Users have two options when they want to search for recipes, either search by category or by ingredients	1. Tap the **Search icon** at navigation bar	Display two options which are “Search by ingredients” and “Search by Category”	As expected	Pass
8.2	The users can search the recipes based on the selected ingredients	1. Tap the search icon 2. Choose the “search by ingredients” section 3. Select one or more ingredients 1. Filter the ingredients with the category tab	A pop out message is shown to ask user if they want to go back to edit or view the search result	As expected	Pass

8.3	The users can search the recipes based on the cuisine category	1. Tap the search icon 2. Choose the “search by category” section 3. Click on the category	Recipes with selected category is displayed	As expected	Pass

4 Design Specifications

4.1 Document Control

Title: HomeChef Design Specification

Owner	Current Version	Last Changed On	Approved By
Kelly	Version 1.11	30/08/22	Kelly

4.2 Distribution List

This table explains how each member is involved in creating the application, while giving clear insight to how each role is being involved

Function of Application	**Frontend / Backend**	**Member Involved**	**Role**
Explore Page/All articles Page/ Articles content Page/ All Recipes Page	Frontend	MingLin and Kiran	Scrum Master and Team member
	Backend	Kelly	Product Owner
Individual Recipe Page	Frontend	Mukil	Team Member
	Backend (Attempted)	MingLin	Scrum master
Shopping List Page	Frontend	WeeJie	Team Member
	Backend	Kelly	Product Owner
Calories Counter Page	Frontend	Kelly	Product Owner
	Backend	Kelly	Product Owner

Search By Category Page	Frontend	MingLin	Scrum master
	Backend	Kelly	Product Owner
Search By Ingredient Page	Frontend	Kelly	Product Owner
	Backend (Attempted)	MingLin	Scrum master
Profile Page	Frontend	WeiXian	Team Member
	Backend (Attempted)	Kelly and WeiXian	Scrum master

4.3 Record of Revision

This table records improvements that have been made by the team to the application such that stakeholders can easily track any changes

Revision Date	Description	Section Affected	Changes Made By	Version after Revision	Approved By
1/08/22	Link articles and recipes page with its content	Explore Page	MingLin	1.01	MingLin
5/08/22	Add liked button for recipe	Recipe Page	Mukil	1.02	MingLin
5/08/22	Add Rating for Recipes	Recipe Page	MingLin	1.03	MingLin
6/08/22	Add checkbox for ingredient listed	Recipe Page	Kelly	1.04	Kelly
13/08/22	Add icon for ingredient listed	Search By Ingredients Page	Kelly	1.05	Kelly
13/08/22	Improved Search By Ingredients Page by adding	Search By Ingredients Page	Kelly	1.06	Kelly

	more categories				
17/08/22	Changes delete button to swipe to delete	Calories Counter Page and Shopping List Page	Kelly and WeeJie	1.07	Kelly
18/08/22	Add counter to adjust serving size	Recipe Page	Kelly	1.08	Kelly
23/08/22	Add back button to go back previous page	All Articles Page, All Recipes Page	Kiran	1.09	MingLin
26/08/22	Add close button to close the search	Shopping List Search Page and Calories Search page	Kelly	1.10	Kelly
30/08/22	Password must be at least 8 digits long	Sign Up Page	Kelly and WeiXian	1.11	Kelly

4.4 Introduction to Design Specifications

Design Specification aims to work as a centralised technical document that indicates a list of points relating to our application. It consists of a system design to show how our application will be developed to meet our objectives and satisfy the needs of the user. System design specifics all the aims and description of our application structure in the proposal that we plan to execute and develop.

In our midterm's reports, several user stories are created in the planning stage. Some of it was chosen by our team to work upon high levels of detail before expanded upon in design specification.

System-based design specifications are organised as follow:

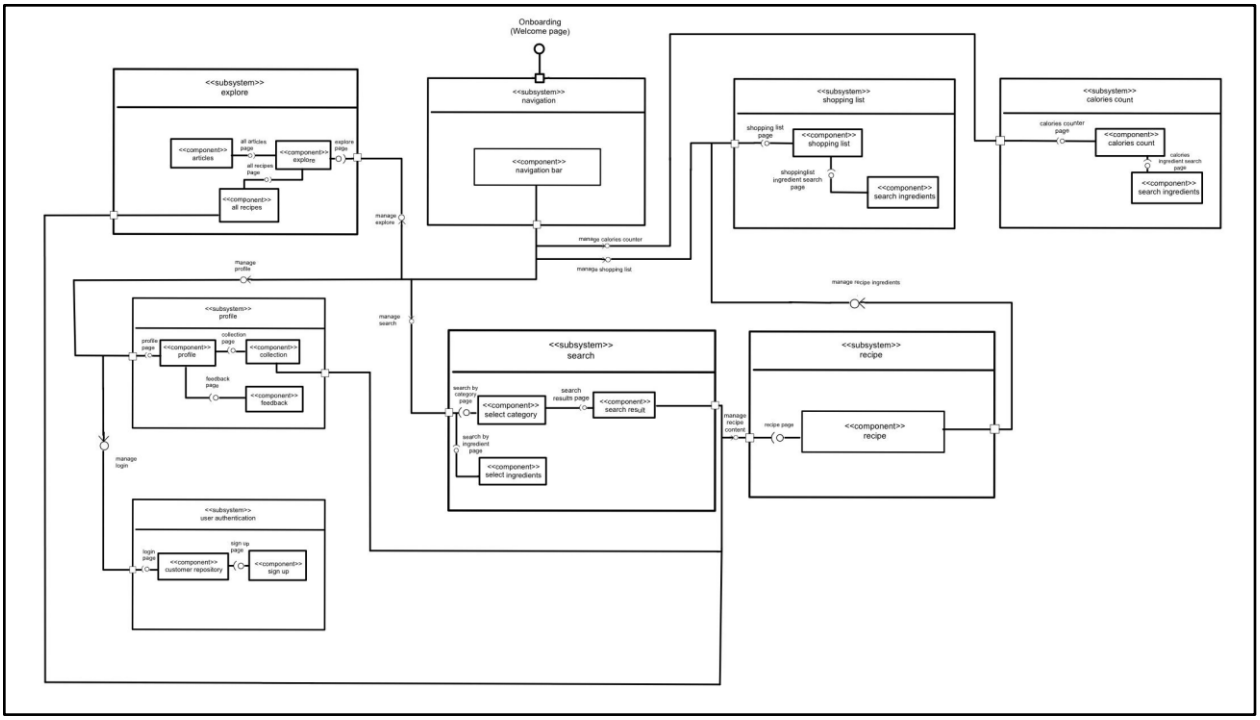
1. System architecture to show relationships between subsystems
2. Module design to show interrelation between system components
3. User interface to explain the reasoning for the interface for each screen
4. Semantics data models to map out the database

After design specifications are made, our team has created the program specified in our design specification. The application is shown to the stakeholders for gathering feedback and updating the user story. This will allow us to improve our application to satisfy their needs better.

4.4.1 System architecture

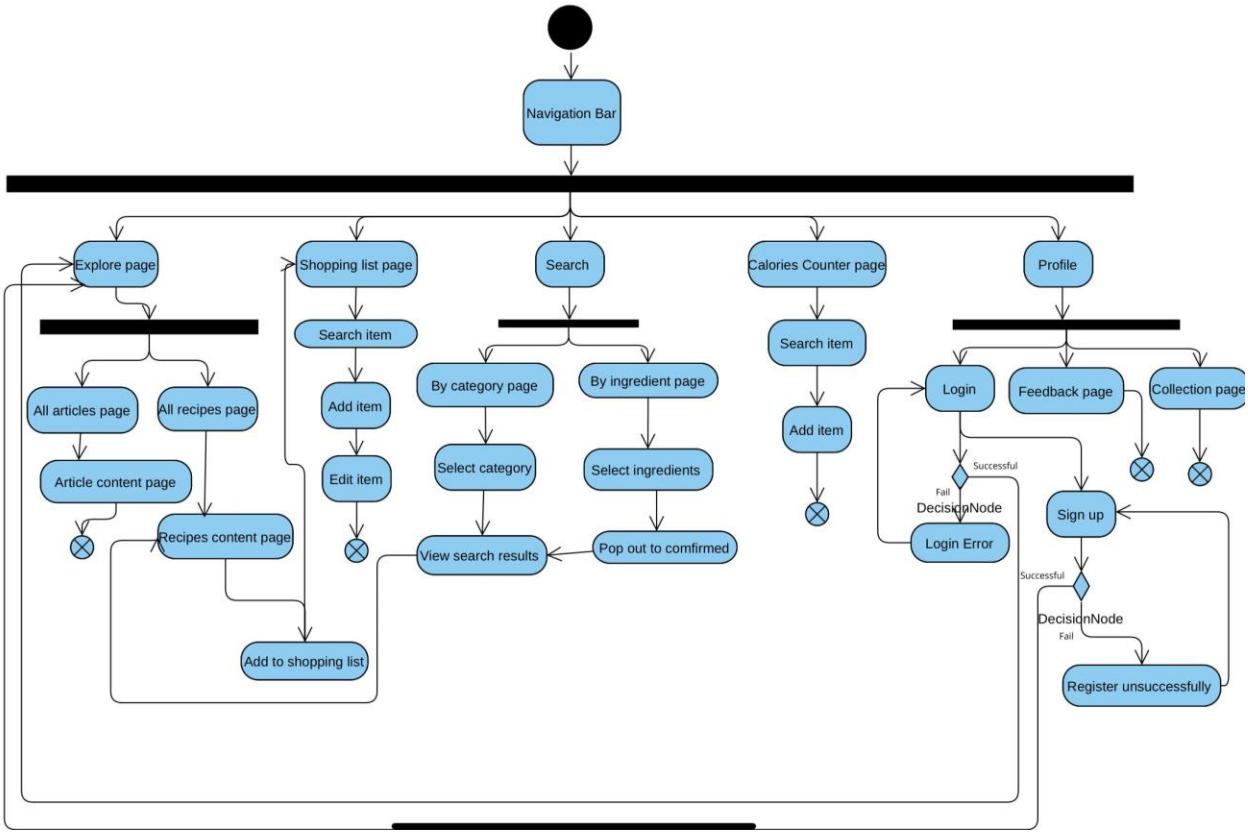
Architectural design specification is a technical document that includes how a software system is developed to achieve the aims described in the requirements. It is important to lay out the details for what developers must build in the future. It is a model that conceptualises the structure, behaviour, and the view of a system. The subsystems define the relationships. It stipulates what each system can provide as a service and how each component is limited in their functionality.

Since we are in Agile, the system architecture should not be written in stone. The design should involve many times as necessary to accommodate the changes when there is a better way to do it to achieve the user's goals. Below is the component diagram to show how we have chosen our system.



4.4.2 Module Design

Module design explains the behaviour and relationship between components and modules involved in our application's system. It helps to illustrate the steps of a user case diagram in higher detail as it shows all the possible flow of events when using our application.



Legend of activity diagram

Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Action	Used to represent the executable sub-areas of an activity
	Control Flow / Edge	Used to represent the flow of control from one action to the other
	Activity Final Node	Used to mark the end of all control flows within the activity
	Flow Final Node	Used to mark the end of a single control flow
	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
	Fork	Used to represent a flow that may branch into two or more parallel flows

The following includes the scenarios on how a user might use HomeChef, to better explain the possible flows throughout the activity diagram.

4.4.2.1 User case 1:

The new user wants to access the application

When a user accesses the application, they would have to login into their account on the login page. They will key in their credentials, their email and password, then click log in which would give them access to their application. However, if a new user is accessing the application, they need to sign up, by clicking on the “sign up” button at the bottom of the login page. After which, the user has to enter their email and create a password which follows a set of guidelines;at least 8 digits long. They would have to re-enter the password to confirm the details, following which their account is created. The user can now log into the application using the account details and access the features available in HomeChef.

4.4.2.2 User case 2

A new user wants to search up a recipe to cook

The new user must begin by opening the application and creating an account in the application. After successfully signing up they would be led to the explore page. The explore page displays a newsletter which contains updates on the newly added and recommended recipes. The new user can browse through the explore page for suggestions on recipes. The user can then click on the “Search” icon and select “Recipes” which will lead them to the search by recipes page. Here, the user can search for recipes based on various categories available. When the user decides on the category, recipe suggestions based on the chosen category will be displayed. Upon selecting the recipe, the user will be led to the recipe page. In the recipe page, there are step by step instructions on how to cook the meal, calorie intake, time taken to make the meal and more details and the rating of recipes.

4.4.2.3 User case 3

A frequent user wants to check their calorie intake

The user would log into the application to access the account. They would then navigate to the “Calculator” icon which would lead the user to the calorie counter page. The user has to search and add the ingredients they have based on per serving size. Users are able to adjust the ingredient serving size accordingly. The user can also view the calorie intake of the recipes when they view the recipe of the meal chosen.

4.4.2.4 User case 4:

A frequent user wants to search by ingredient

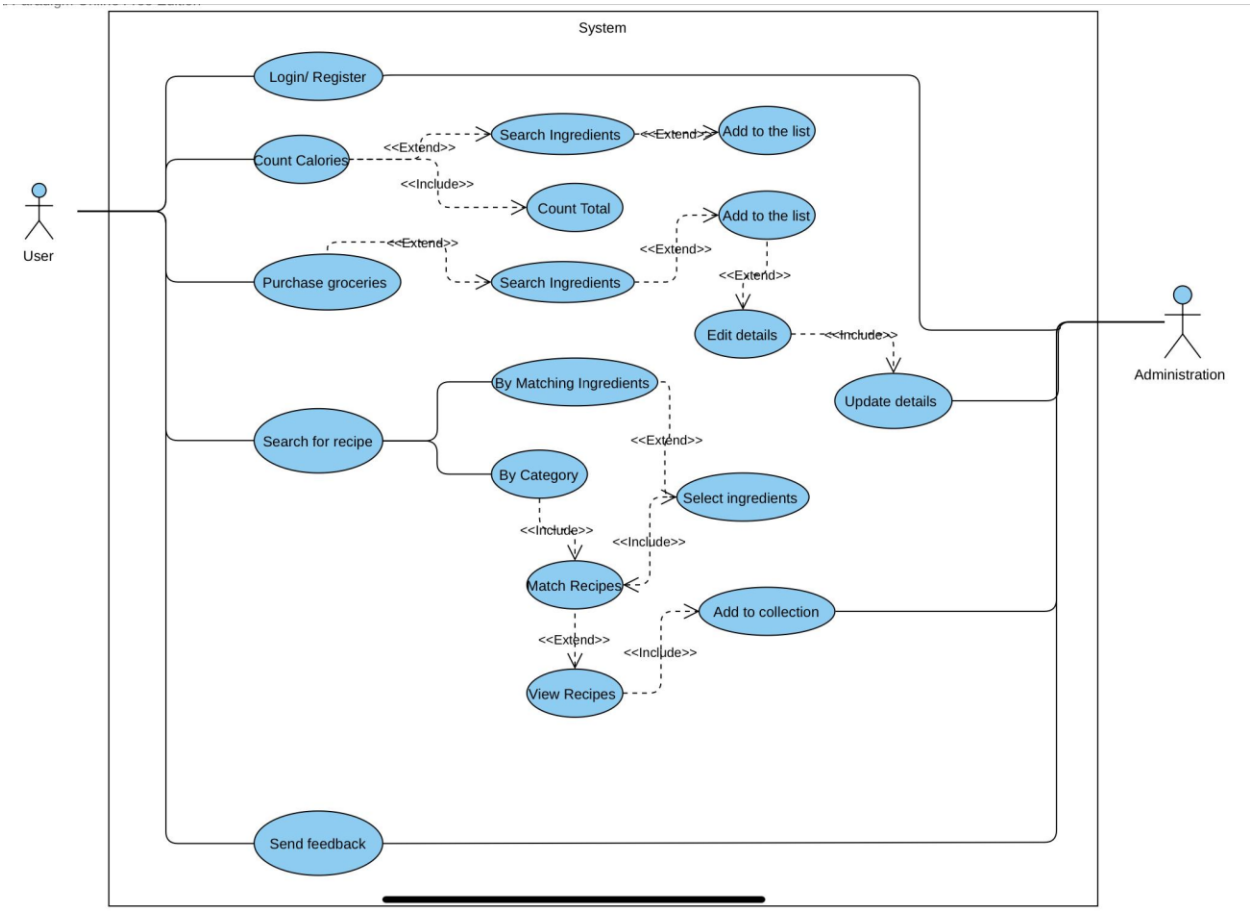
The user can click on the “ the search” icon in the middle of the app. Once it is tapped there will be 2 choices for the user to pick. Either search by category or ingredient. User will proceed to click on the ingredients. In the search by ingredient page, users can proceed on searching for the desired ingredient by tapping the box on top of the screen. User can also use the category tab to see the ingredients available in the selected calories. After finish selecting, a pop out will appear for user to view the search result after clicking on the match button.

4.4.2.5 User case 5:

A frequent user wants to list out their groceries to prepare for shopping of groceries the next day

Users can go to the shopping list tab and click on the add an item button to search for items. Upon clicking on the item, it will be added to the list. Users can edit and item details such as quantity, size or even put a note. Users can delete the selected item by swiping right and clicking on the delete button.

4.4.3 Use Case Diagram



The figure above shows the flow of the application which is like the use case diagram that we have depicted in midterm.

Users can login or register, while administrators would check the authentication for login and registration. When users wish to search for a recipe, they can choose the option to either search the recipe by category or ingredients. If they wish to search it by ingredients, they can proceed to select the ingredient in the ingredients list. They will then be shown the related recipes with the ingredients selected.

If users wish to save the recipe for quicker access next time, they can save it to their collection. The administrators will then update the user collection tab, so that the recipe will appear in their account's collection tab when they use the app in the future.

When users add the item to their shopping list, they can edit the quantity and size, and put a note for the item. If users wish to do calories count on a list of ingredients, they can do so in the calories counter. It allows users to adjust the serving size of each ingredient and the total calories will be updated accordingly. In any case, if a user wants to send feedback after using the app, they can send it at the tab found in the profile page. Administration will take their feedback into account for improvement of application upon receiving it.

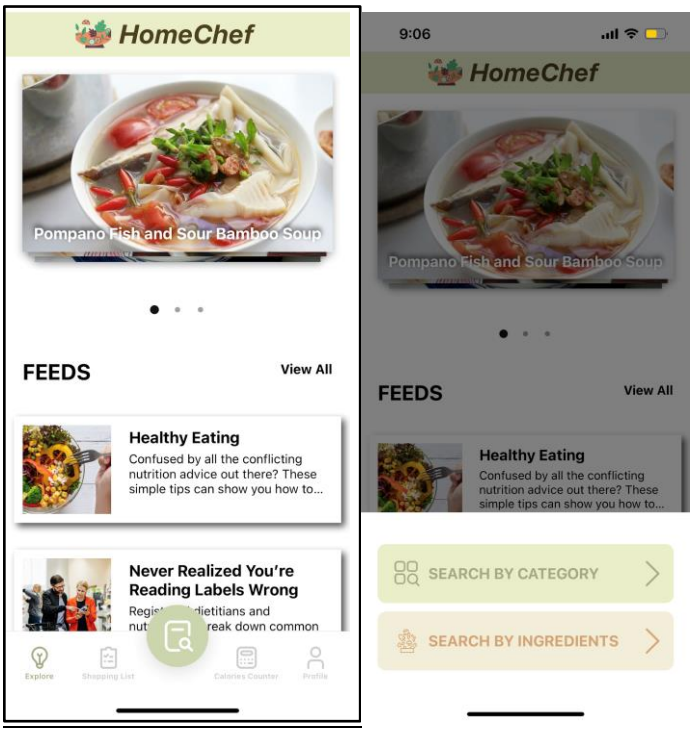
4.4.4 User Interface

The User Interface (UI) refers to the point of interaction between the computer and a user in a given application or website. Our aim is to create an effective UI that facilitates efficient user interaction between the users and our app by providing clean and purposeful design and responsiveness. It should provide the best user experience by being intuitive and easy to use, while satisfying the requirements of the users, so that we can expand our app's conversion rates.

Initially, our team planned to use basic colour and design for our application, but we had decided to use the natural colour series such as green, beige and brown colour for some part of our application. This is because we try to promote healthy eating and these colours simulate health and nature.

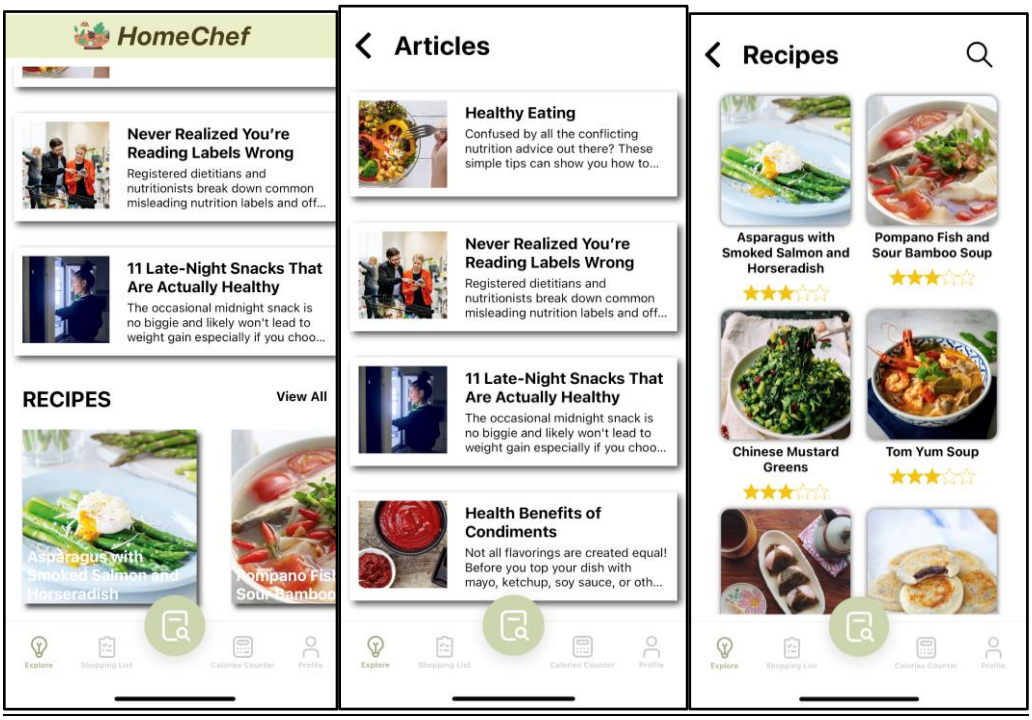
The following pages describe the purpose of each interface used in our application and its role within sub-systems and other external systems.

4.4.4.1 Explore Screen



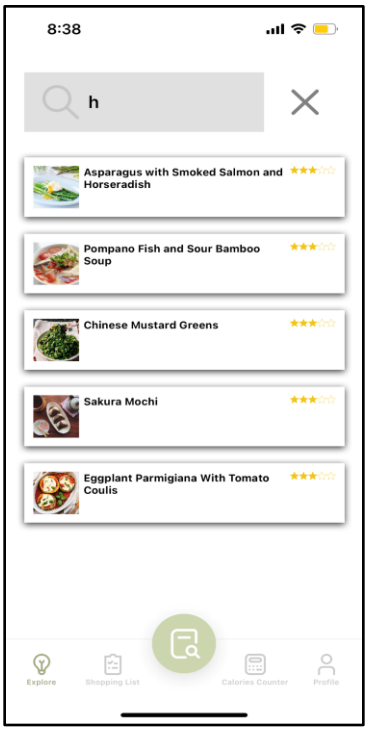
The navigation bar contains the redirecting links to the "Explore", "Shopping List", "Search", "Calorie Counter" and " Profile" pages. It is available at every page of the screen. We have chosen grey colour for the tab icon at the navigation bar and the colour of the tab icon will change when user on the respective pages. The search icon is different from the rest as that is the main function of the application. Upon clicking the search tab, two options will be available for user to select.

The carousel at the top is used for rendering popular recipes in the database which contain the image and the title of the recipe. Users can swipe right to view for more items stacked below. They are also able to swipe back to see the previous item. The three dots below the image are for users to directly access the pages they want without having to keep swiping.

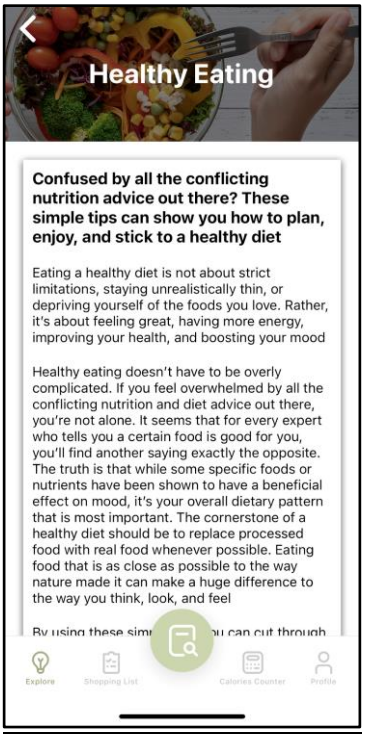


Explore screen also consists of the interface which displays newsletter and the recently added recipes. There is a view all button at both sections for viewing more items for the respective section.

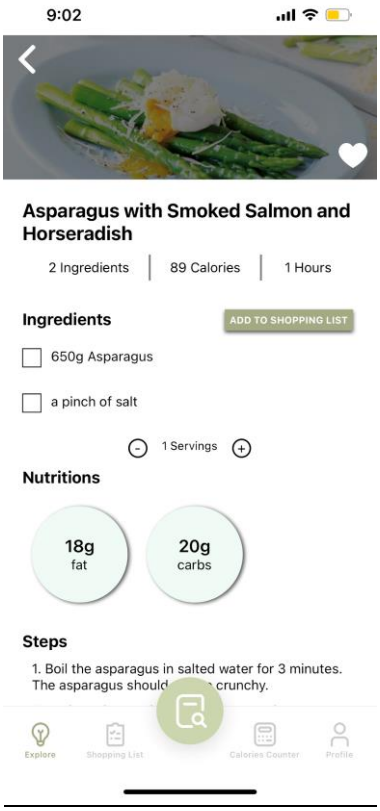
There is a search button for users to search for desired recipes without scrolling through the whole section.



Upon typing text at the search bar, a list of related recipes will be shown with its image, title and rating. When the user clicks on the item, it will direct the user to the recipe content. The close button is for the user to navigate back to the previous page.

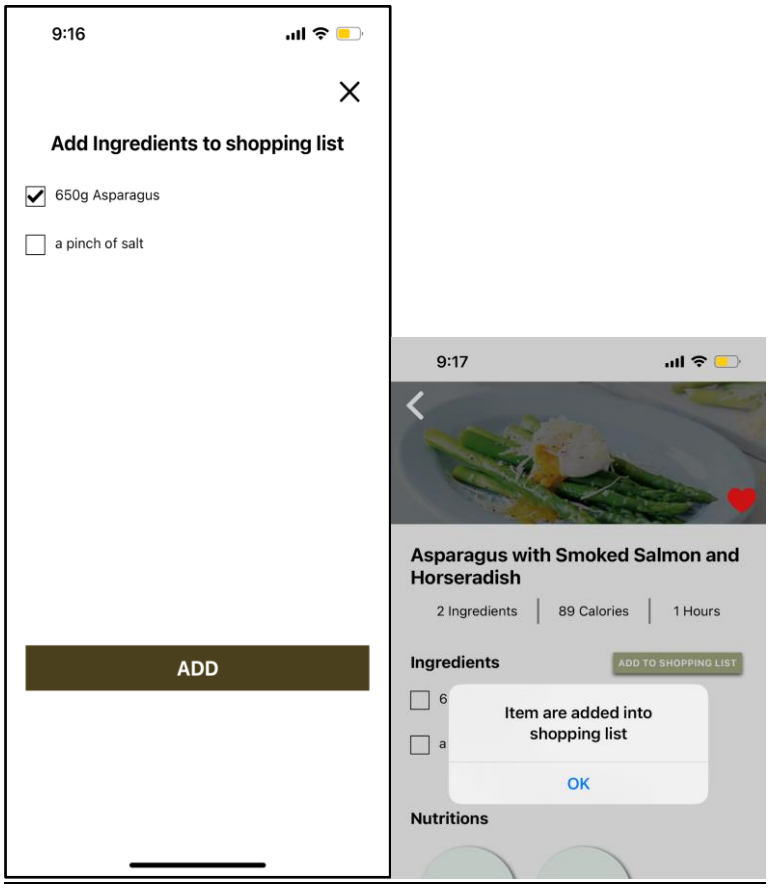


In the article content page, users will be able to see the title placed within the article image with a black overlay. This will allow the user to see the title clearly. The subheading and the paragraph text is found below the image.

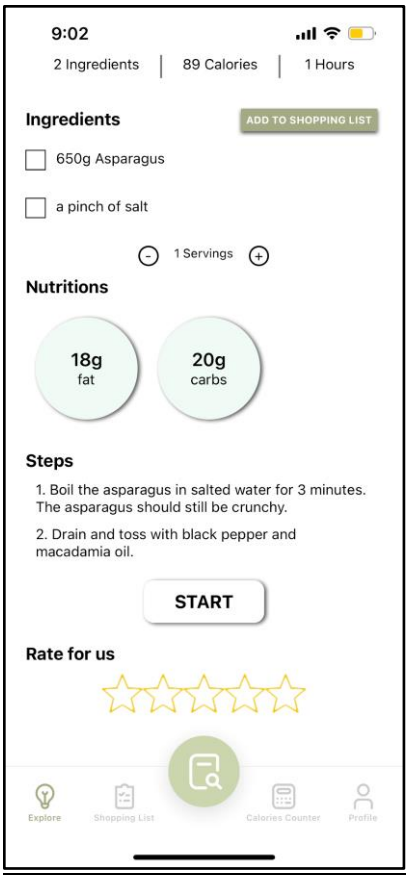


In the recipe content page, it includes the number of ingredients, total calories count for the dish, and the time taken to complete the dish at the top. Proceeding, the list of ingredients needed is listed with a checked box. This allows users to track the ingredients they have easily.

The add to shopping list button is for users to directly add the ingredients from the recipe to the shopping cart. Users can select and deselect the list of ingredients before adding it.



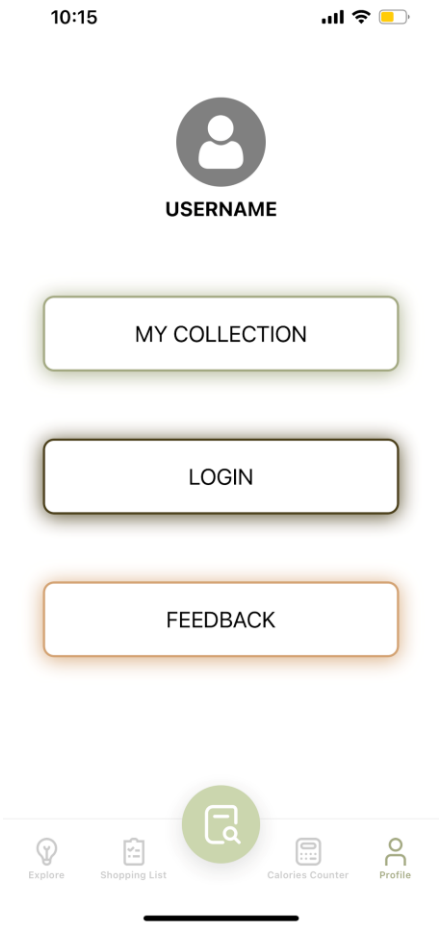
A pop out message will be shown when the user clicks on the add button.



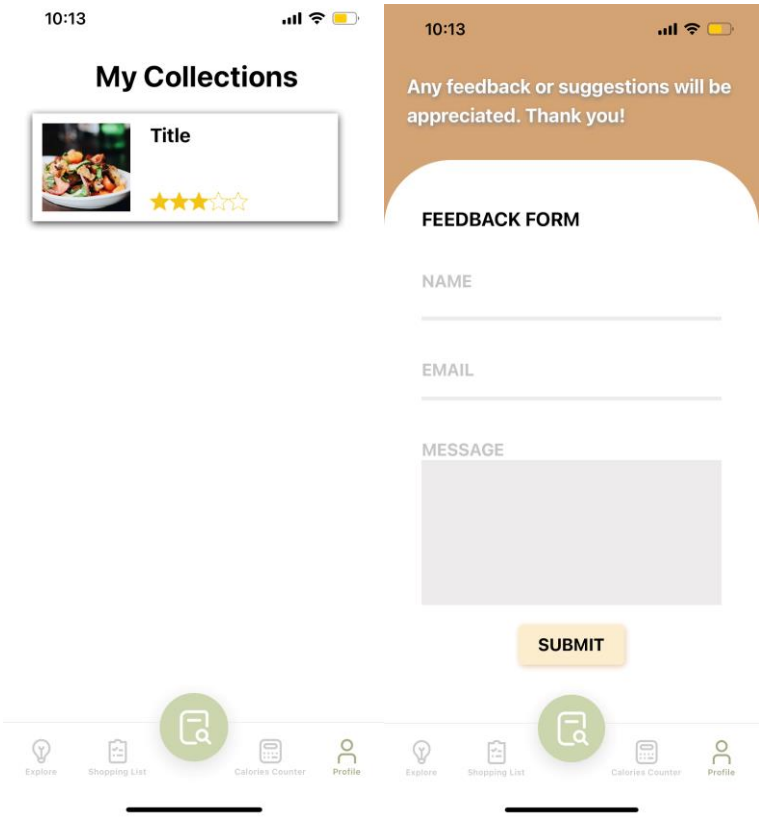
There is a counter for users to adjust the serving size of the recipes. The list of nutritional facts of the recipe is found below the ingredient list as well as the facts. User can also rate the recipes.

There is a back button for users to go back to the previous page and a heart button for users to add the recipe to their collection. An alert box will be shown upon clicking.

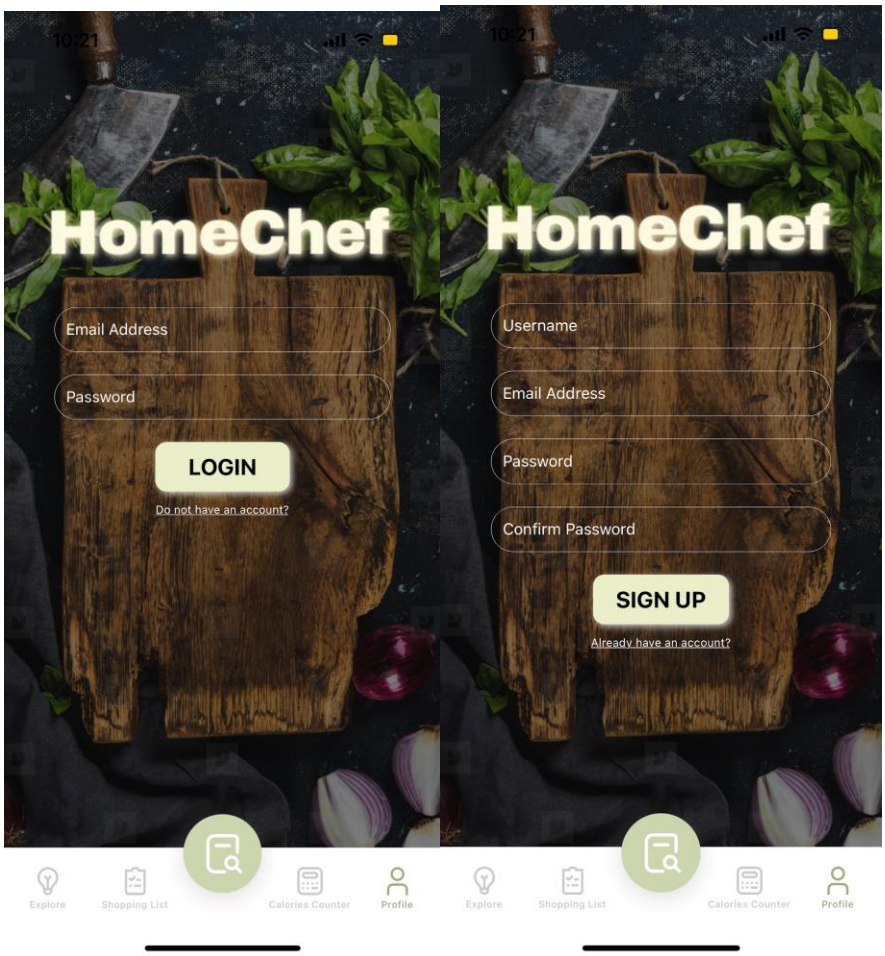
4.4.4.2 Profile Screen



In the profile screen, we have the tabs for my collection which store all the recipes that users saved, a login tab for users to login to their account and a feedback tab for users to send feedback about the application to the administrator.



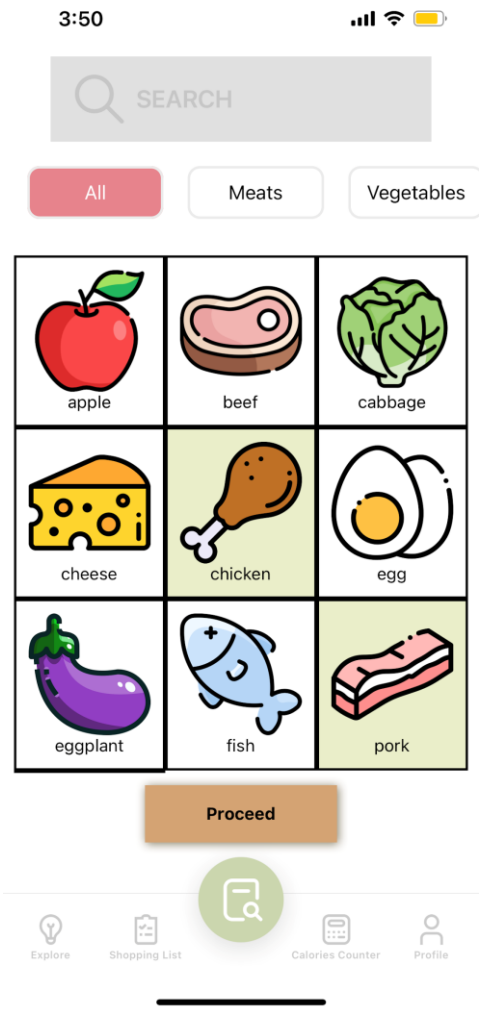
In the collection tab, the image, title, and the rating of the recipes will be shown while in the feedback tab the feedback form requires the user to key in their name and email with the feedback they intend to write.



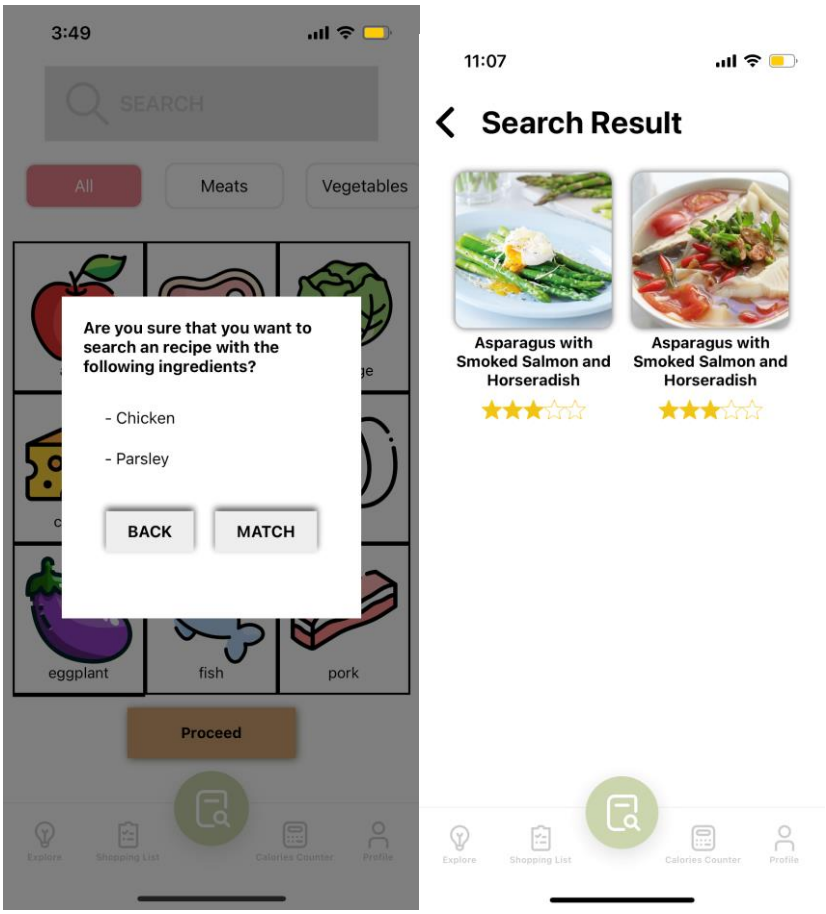
For the login tab, it required the user to key in their registered email and password. For new users, they can access the sign-up page by clicking onto the “do not have an account” and username, email, password and confirmed password are required to register their account.

Error messages will be shown if the text input key is in the wrong format such as password must have 8 characters etc. Upon successfully clicking on the login or sign-up button, they will be directed to the home page.

4.4.4.3 Search by Ingredient Screen

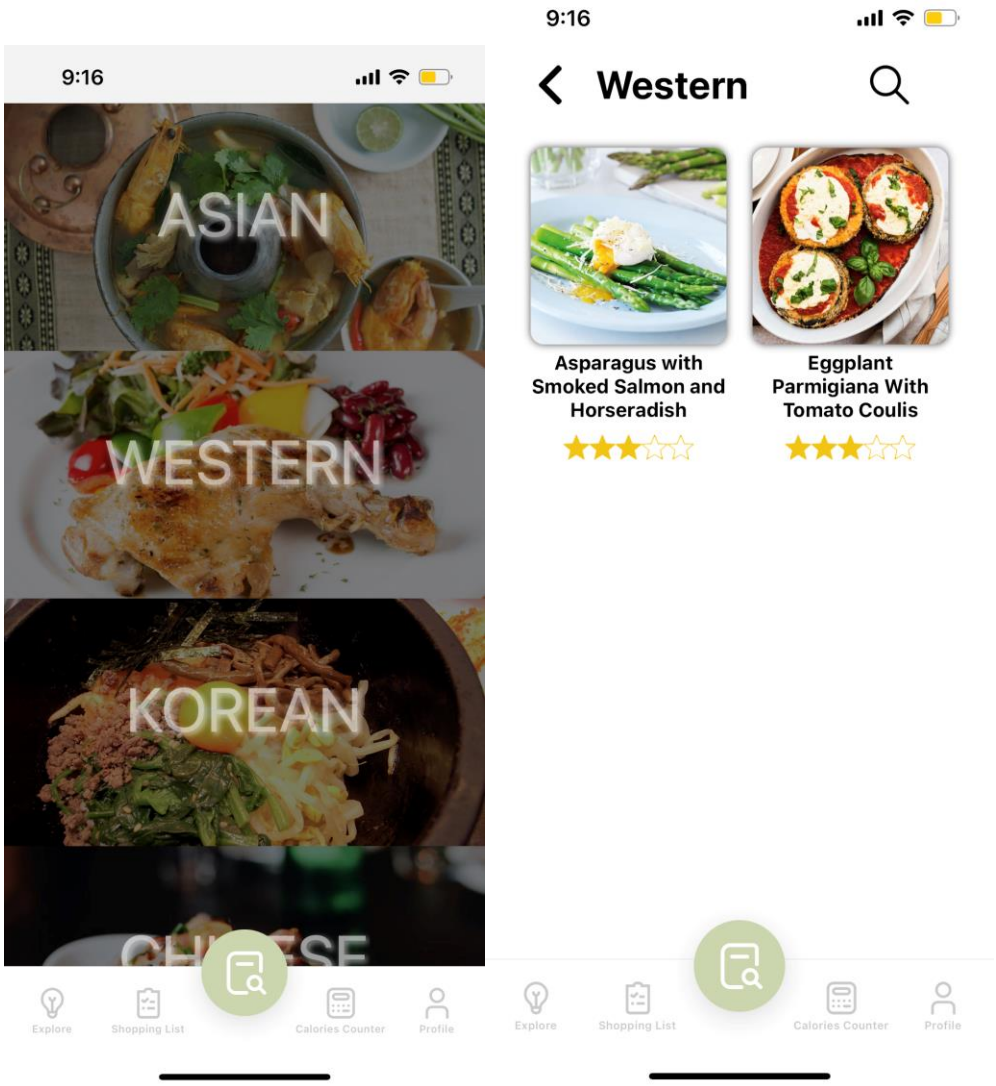


Users can use the search bar or the category filter to search and view the ingredient list for selection. The ingredient is presented with the related icon and its name for better visualisation. The background of the selected category tab and ingredients selected is changed.



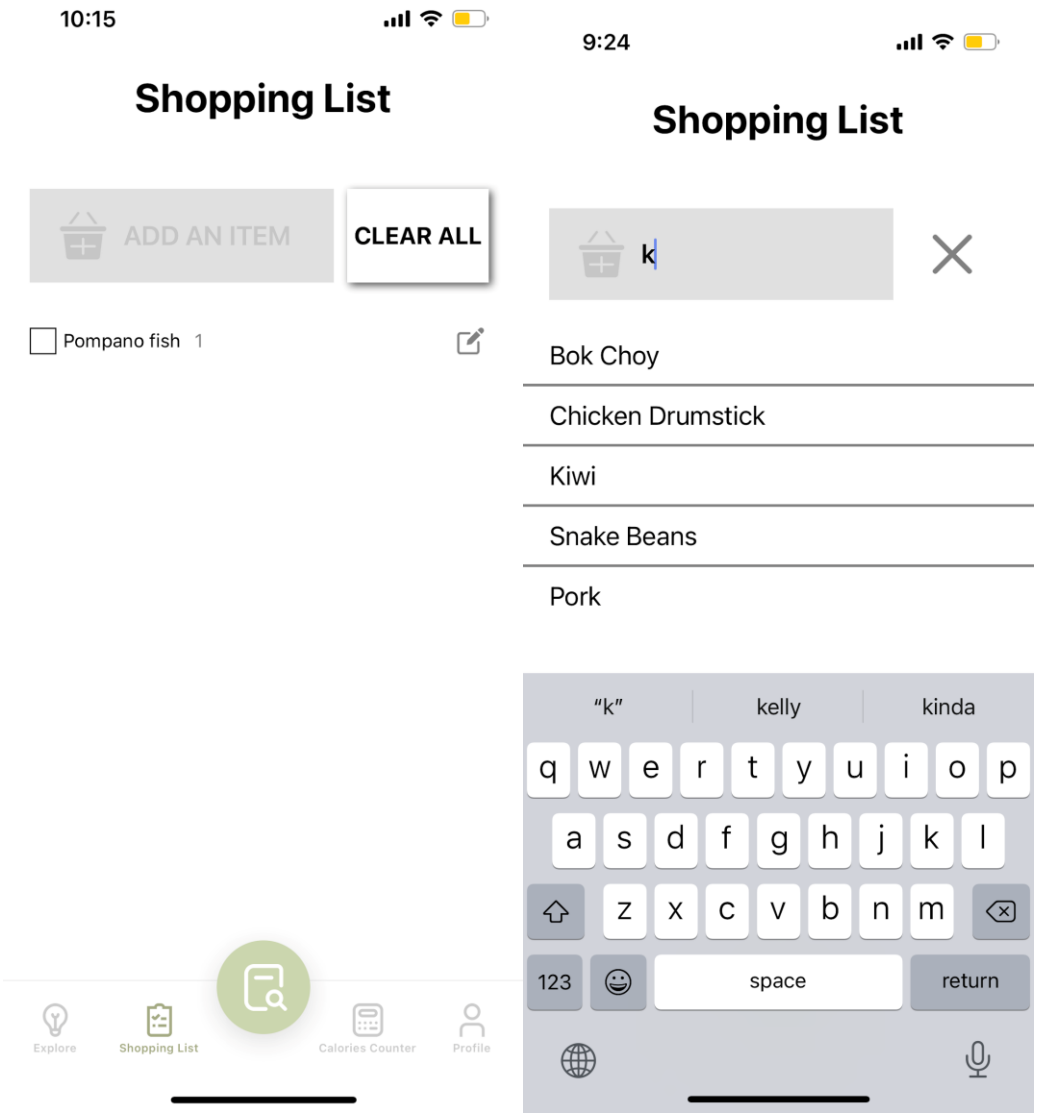
After selection, the user can click “Proceed”, which will prompt the user to check the list again for confirmation before showing the related recipes. Users will have to click on the match button to view the search result.

4.4.4.5 Search by Category

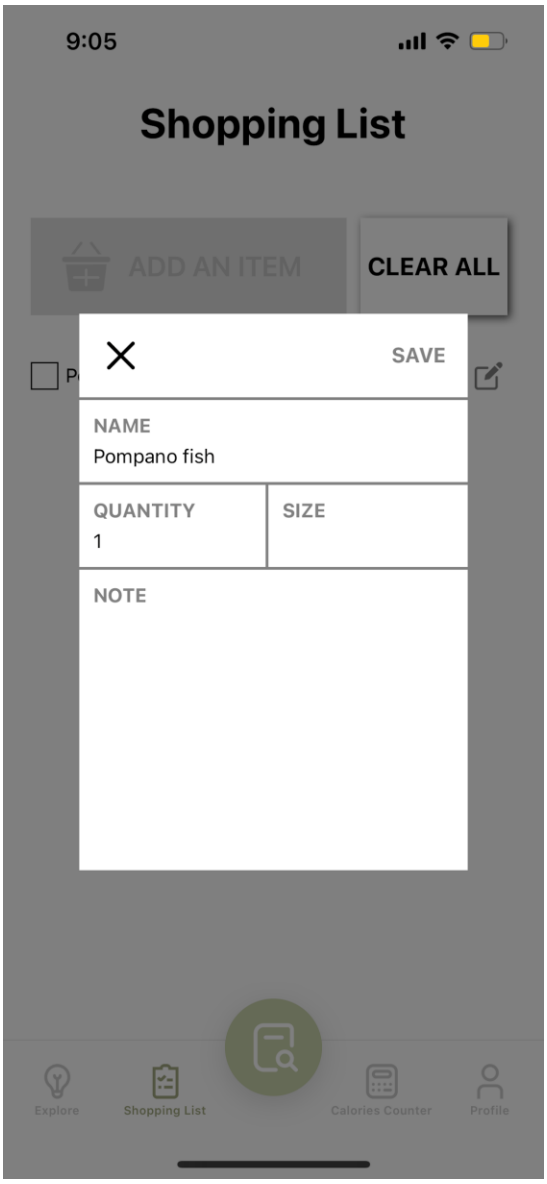


A list of cuisine categories is shown for the user to select. Users can select the desired category to view the recipes included in the selected category.

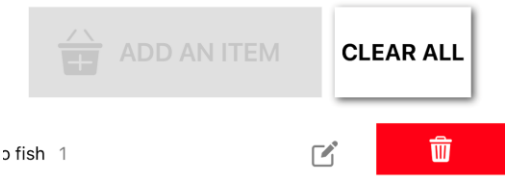
4.4.4.6 Shopping list Screen



To add an item into the shopping list, users need to click on the add an item button to access the search page. Users will need to search for the ingredients, and it will be added upon clicking.



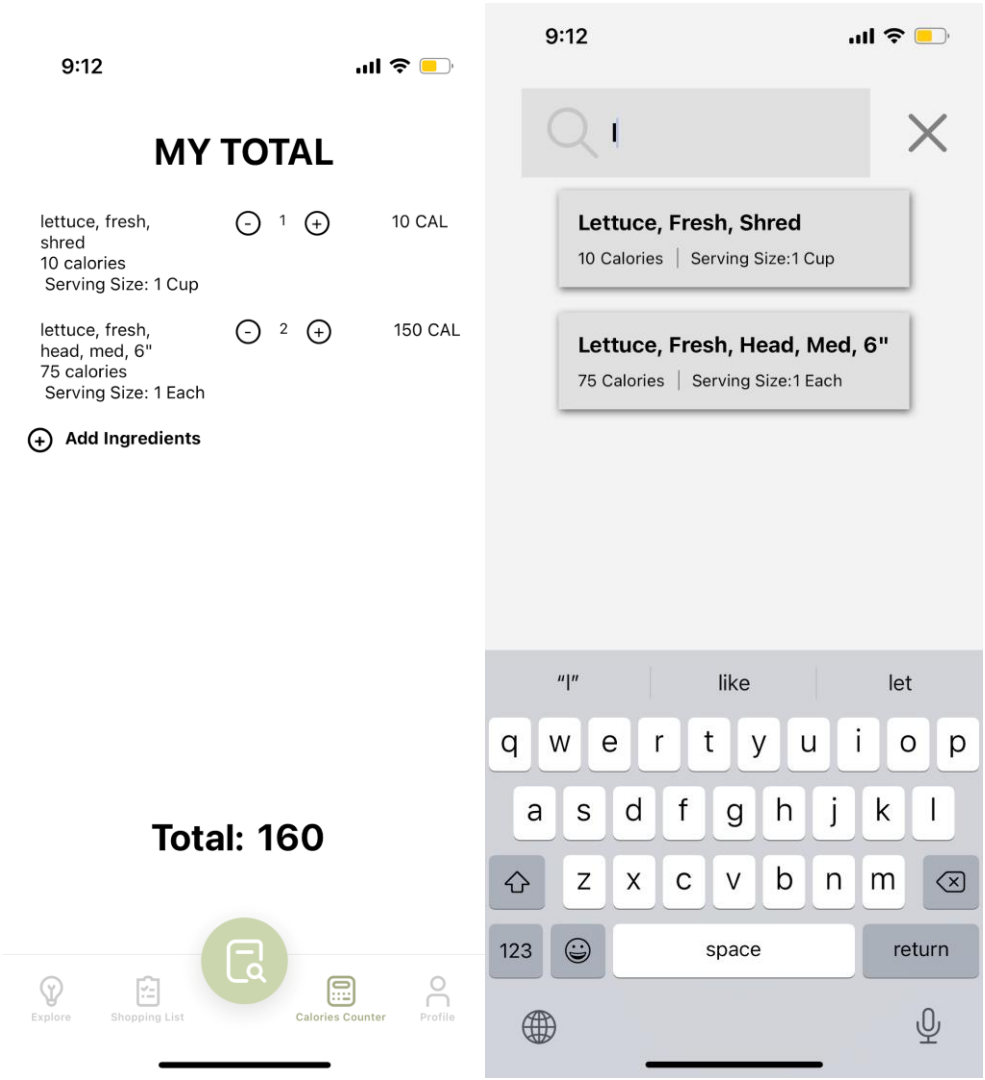
Every added item will have an edit button which allows the user to edit the quantity, size and notes. The quantity and size will be shown with the ingredient.

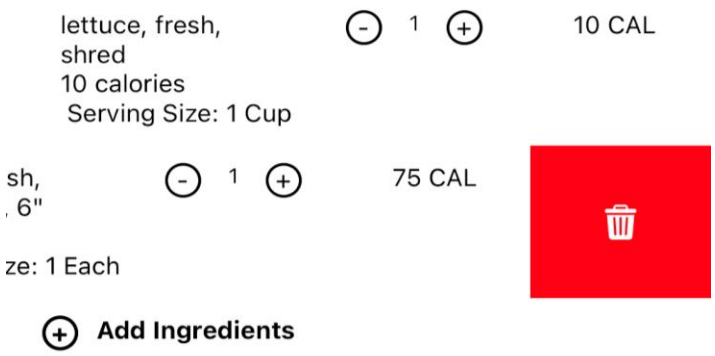


If a user wishes to delete the added ingredient, they need to swipe right and click on the delete button. The clear all button will delete everything added in the shopping list.

4.4.4.7 Calories Counter Screen

Users will need to click on the add ingredient button and search for ingredients to be added. Users are allowed to adjust the serving size, and the number of calories will be calculated accordingly. All the ingredients added are included in the total calories count.





Each ingredient list also includes a swipe to delete button.

4.4.5 Semantic Data Model

The semantic data model is a high-level semantics-based database description and structuring formalism for databases. The semantic data model specifications, which is a document to explain databases, can be used to further explain entities within the HomeChef environment, the taxonomies, and classifications of said entities, as well as the interconnection among them structurally. Hence, it allows us to augment the usability and efficiency of our database system in the database design process. This includes allowing our team to see the various ways that our database can accommodate the derived information from our audience, as well as its requirements to process this information. Our team has chosen two ways to organise our database, in the context of our application.

Our Team decided to adopt a dynamic database that imports data info like recipes and ingredients into the app from Firebase, which is a web database storage by google. By using this, there is no need to store data in local files or in the .js files itself as an array. Data can be pulled directly from firebase where all the tabs that use the same function are able to have access to the data. Any changes made to the database will be updated in real time to all devices that have

access to the app. Future update features will only app functionality but not impact the database used.

Our Team has also decided to adopt a static database to store some of the data, the reason why we have chosen to implement a static database over a dynamic database is due to time and complexity constraints. Hence, we have focused more on the database for the ingredients and recipes, as the main deliverable for our app is to be able to provide meal prep information to the user. Despite the limitations mentioned we hope to complete the backend work required for our application with working dynamic databases that stores the cooking information that bundles with user data.

5 System Development

As mentioned in the midterm report, our team decided to follow through with the agile using sprints and user centred design methodologies, so our group is split into subgroups each focusing on a feature of the application.

5.1 Technical Responsibilities

Functionality of the application	**Member in charge**
Explore page	MingLin and Kiran
Shopping List page	Weejie and Kelly
Search page (By Category, By Ingredients)	MingLin and Kelly
Calories Counter page	Kelly
Profile page, Login and Signup page	WeiXian and Kelly
Recipe page	Mukil
Backend of development	Kelly and MingLin

Overall navigation of the application	Kelly
Connect with firebase with the application	Kelly
Integration and debugging of the application	MingLin

Using the sprints, we do have meetings weekly to discuss and update everyone process and review each other work. We gave each of the subgroups a time frame to finish the task that was assigned to them.

We decided to use OneDrive sharing where we can upload the file and shared it among our members. When someone upload the file, Kelly or MingLin will download and see if it run as per normal and merging will be done afterwards.

However, to ease the development process for future version control we decided to use a GitHub repository (link in appendix), the one in charge of managing the repository is Kelly and MingLin as they in charge of most of the technical stuff.

The link in appendix shows the application under a YouTube link.

5.2 Development of the code

The language that our team undertook is JavaScript and the framework used is React Native and Expo. The reason why we chose Expo as it is a framework and universal platform for React applications. Our team took a week to understand the basics of react native and Expo before we started on our project as it was a whole new experience for all of us. For the backend of the project, we used Google firebase to store our data. The IDE we used is Visual Studio Code.

5.3 Introduction of the application's file structure

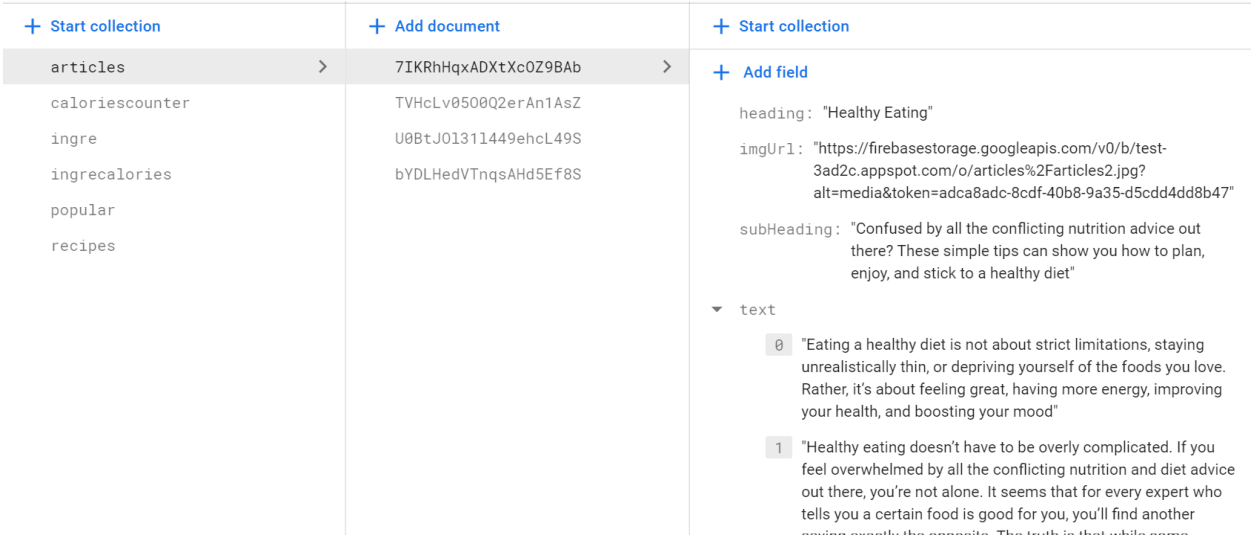
Our team used the high-fidelity prototype as a reference to develop the code. We placed some of the components that will be reused in the components folder. Components help us to design and style the user interface and it is frequently used when we are designing the screens. Refer to the appendix for the libraries used, the description of components used and the purpose of each screen in the screen folder.

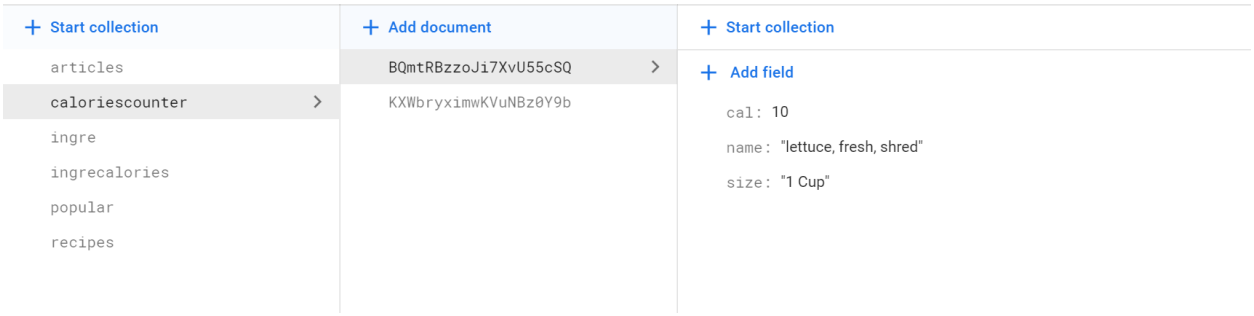
5.4 Description of Code

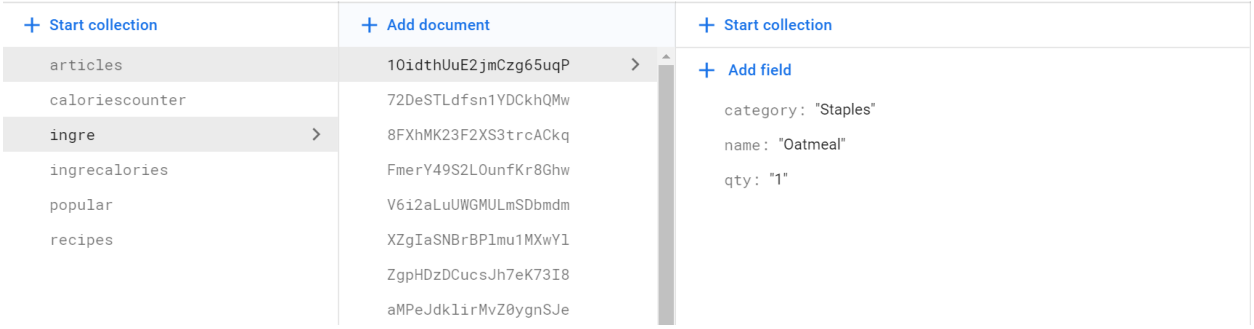
HomeChef's development was facilitated with the iPhone emulator. Due to time constraints and the huge scope of our application, the team prioritised developing the UI of the application as this would allow users to be aware of the unique functionalities and have a rough understanding of how the whole application works. Our team had also done up some of the simple functionalities for the application such as counting the calories for every ingredient according to its serving size and total it up. Subsequently, our team would work on the development of the complex functionalities such as matching the selected ingredients to the recipes in our database.

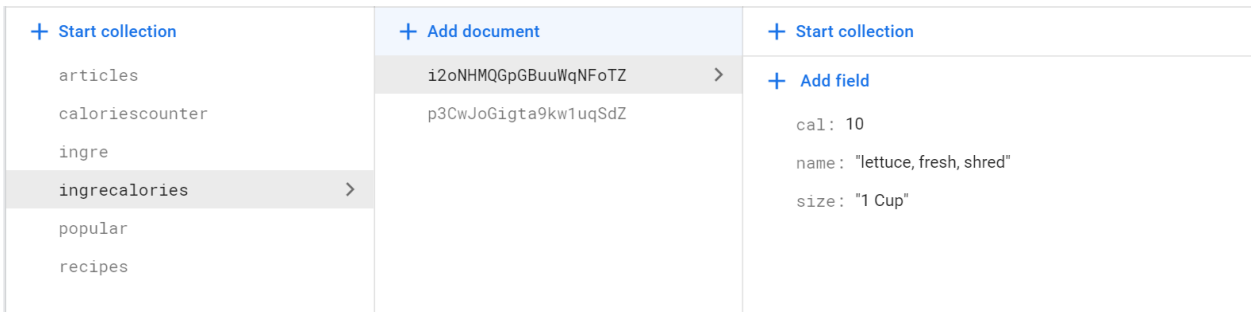
Our team has created a database in cloud Firestore, which allows us to perform CRUD (Create, Read, Update, Delete) operations with React Native.

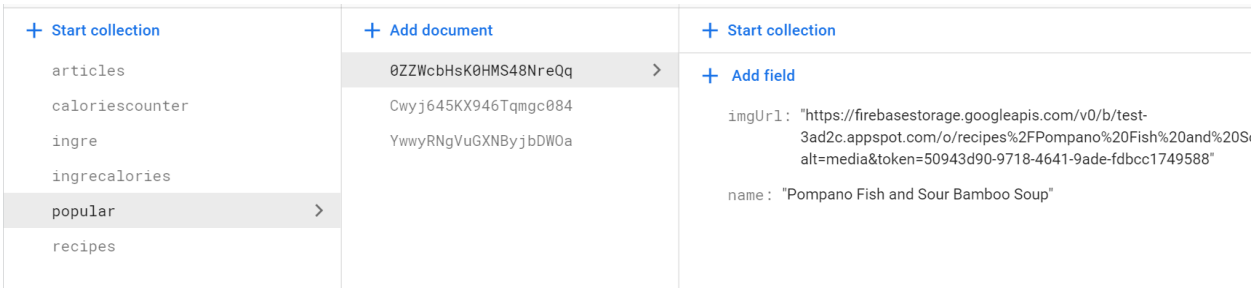
Database used in Firestore:

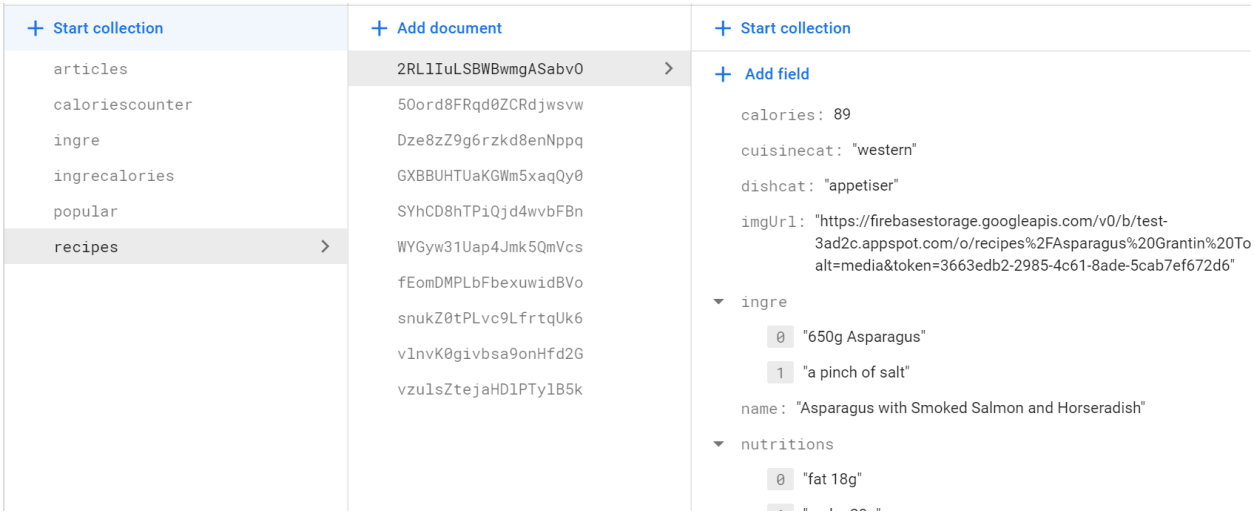

Start collection
articles
caloriescounter
ingre
ingrecalories
popular
recipes
Add document
7IKRhHqxADXtXcOZ9BAb
TVHcLv05O0Q2erAn1AsZ
U0BtJO1311449ehcL49S
bYDLHedVTnqsAHd5Ef8S
Start collection
Add field
heading: "Healthy Eating"
imgUrl: "https://firebasestorage.googleapis.com/v0/b/test-3ad2c.appspot.com/o/articles%2Farticles2.jpg?alt=media&token=adca8adc-8cdf-40b8-9a35-d5cdd4dd8b47"
subHeading: "Confused by all the conflicting nutrition advice out there? These simple tips can show you how to plan, enjoy, and stick to a healthy diet"
text
0 "Eating a healthy diet is not about strict limitations, staying unrealistically thin, or depriving yourself of the foods you love. Rather, it's about feeling great, having more energy, improving your health, and boosting your mood"
1 "Healthy eating doesn't have to be overly complicated. If you feel overwhelmed by all the conflicting nutrition and diet advice out there, you're not alone. It seems that for every expert who tells you a certain food is good for you, you'll find another

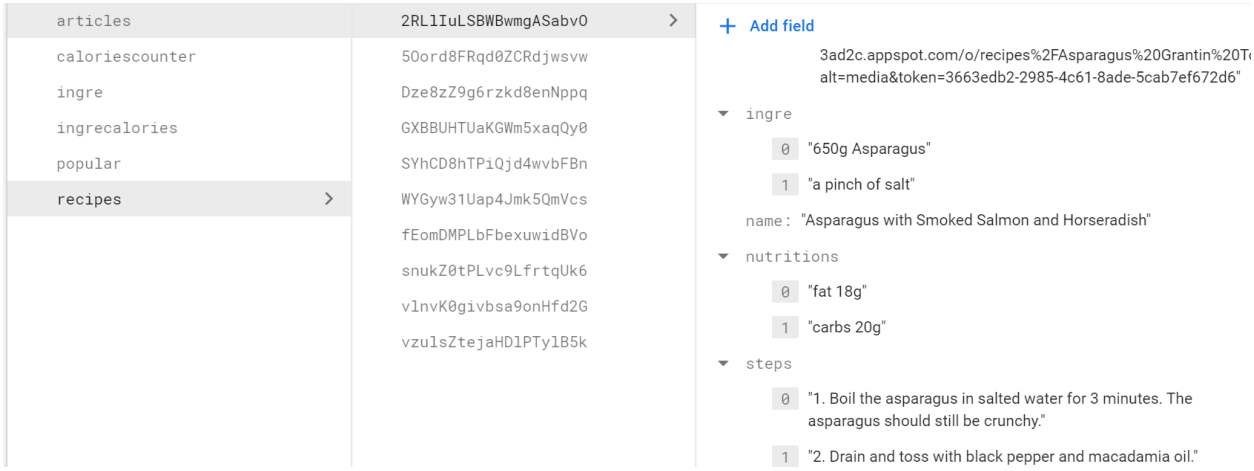

Start collection
articles
caloriescounter
ingre
ingrecalories
popular
recipes
Add document
BQmtRBzzoJi7XvU55cSQ
KXWbryximwKVuNBz0Y9b
Start collection
Add field
cal: 10
name: "lettuce, fresh, shred"
size: "1 Cup"


Start collection
articles
caloriescounter
ingre
ingrecalories
popular
recipes
Add document
1OidthUuE2jmCzg65uqP
72DeSTLdfsn1YDCkhQMw
8FXhMK23F2XS3trcACkq
FmerY49S2LOunfKr8Ghw
V6i2aLuUWGMULmSDbmdm
XZgIaSNBrBP1mu1MXwY1
ZgpHDzDCucsJh7eK73I8
aMPeJdklirMvZ0ygnSJe
Start collection
Add field
category: "Staples"
name: "Oatmeal"
qty: "1"


Start collection
articles
caloriescounter
ingre
ingrecalories
popular
recipes
Add document
i2oNHMQGpGBuuWqNFoTZ
p3CwJoGigta9kw1uqSdZ
Start collection
Add field
cal: 10
name: "lettuce, fresh, shred"
size: "1 Cup"







We will first need to connect to the Firestore to do the CRUD operations. The config.js file is used to export the Firestore instance which will allow us to use it to query the database. The config file is used to configure firebase within our application.

To fetch the data in Firestore the following code is used:

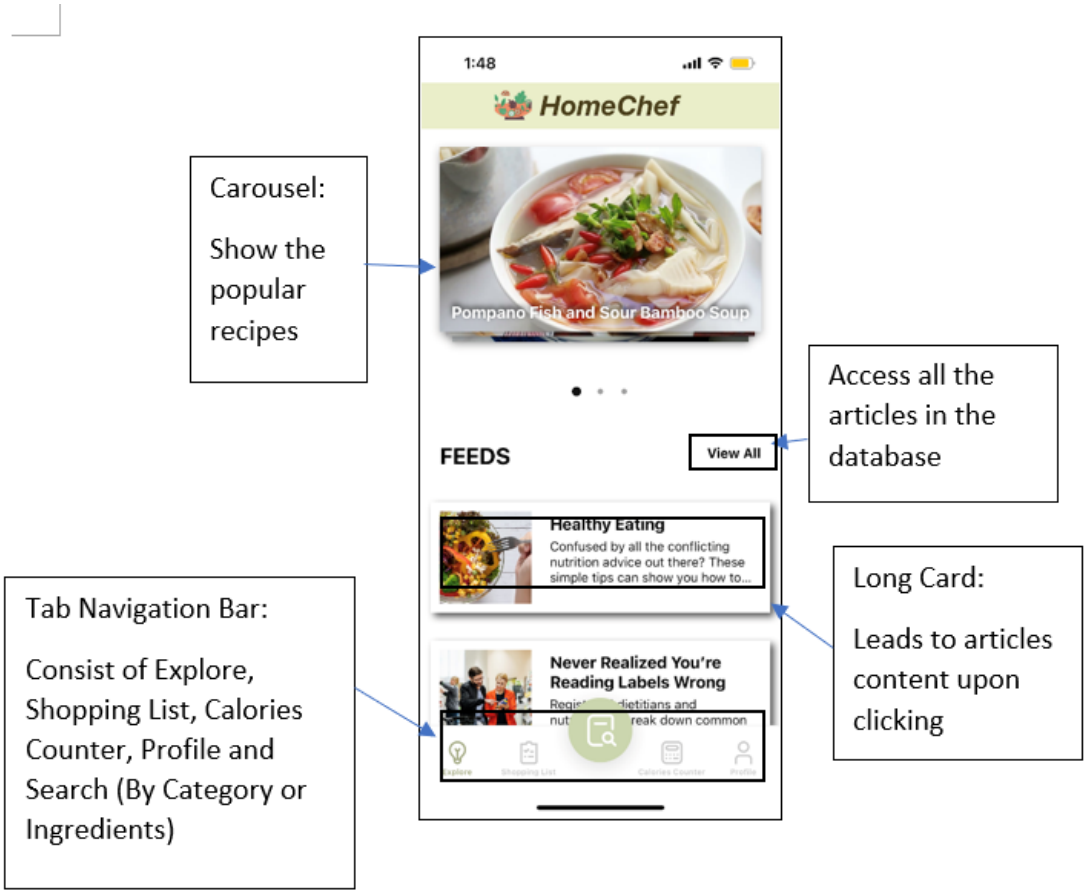
```
import { firebase } from "../config";

const AllArticlesPage = ({ navigation }) => {
  const [articles, setArticles] = useState([]);
  const articlesRef = firebase.firestore().collection("articles");

  useEffect(() => {
    (async () => {
      articlesRef.onSnapshot((querySnapshot) => {
        const articles = [];
        querySnapshot.forEach((doc) => {
          const { heading, subHeading, imgUrl, text } = doc.data();
          articles.push({
            id: doc.id,
            heading,
            subHeading,
            imgUrl,
            text,
          });
        });
        setArticles(articles);
      });
    })();
  }, []);
```

First, import the firebase function from the config file. Next, declare the initial state of the variable as an empty array and create the reference of the collection we want to fetch the data. We fetch the data in real-time by using the querySnapshot function. We declare an empty array and push the data we want to it. Finally, set the variable state as the array after looping every element in the collection. This code is used across many screens where data from Firestore is needed.

The description of the code for the user interface of the page:

5.4.1 Explore.js

The screenshots of homepage:



```jsx
import React from "react";
import { View, Text, StyleSheet, Dimensions, Image } from "react-native";

export const SLIDER_WIDTH = Dimensions.get("window").width;
export const ITEM_WIDTH = Math.round(SLIDER_WIDTH * 0.9);

const CarouselCardItem = ({ item, index }) => {
  return (
    <View style={styles.container} key={index}>
      <Image source={{ uri: item.imgUrl }} style={styles.image} />
      <View style={styles.txtCont}>
        <Text style={styles.header}>{item.name}</Text>
      </View>
    </View>
  );
};
```

The Carousel Card Item implements the look of the cards in the carousel. It returns a component that will display the item passed as props. The item name is displayed at the bottom of the picture.

```jsx
import React, { useEffect, useState } from "react";
import { View } from "react-native";
import Carousel, { Pagination } from "react-native-snap-carousel";
import CarouselCardItem, { SLIDER_WIDTH, ITEM_WIDTH } from "./CarouselCardItem";

import { firebase } from "../config";


const CarouselCards = () => {
  const [carousel, setCarousel] = useState([]);
  const carouselRef = firebase.firestore().collection("popular");

  useEffect(() => {
    (async () => {
      carouselRef.onSnapshot((querySnapshot) => {
        const carousel = [];
        querySnapshot.forEach((doc) => {
          const { name, imgUrl } = doc.data();
          carousel.push({
            id: doc.id,
            name,
            imgUrl,
          });
        });
        setCarousel(carousel);
      });
    })();
  }, []);
  const [index, setIndex] = React.useState(0);
  const isCarousel = React.useRef(null);
```

```jsx
return (
  <View>
    <Carousel
      layout="tinder"
      layoutCardOffset={9}
      ref={isCarousel}
      data={carousel}
      renderItem={CarouselCardItem}
      sliderWidth={SLIDER_WIDTH}
      itemWidth={ITEM_WIDTH}
      onSnapToItem={(index) => setIndex(index)}
      useScrollView={true}
    />
    <Pagination
      dotsLength={data.length}
      activeDotIndex={index}
      carouselRef={isCarousel}
      dotStyle={{
        width: 10,
        height: 10,
        borderRadius: 5,
        marginHorizontal: 0,
        backgroundColor: "rgba(0, 0, 0, 0.92)",
      }}
      inactiveDotOpacity={0.4}
      inactiveDotScale={0.6}
      tappableDots={true}
    />
  </View>
);
};
```

The Carousel component from the react-native-snap-carousel package was used to implement the carousel. Tinder layout was used to present the stacked card-like layout. Pagination is added so that users can skip to a certain item in the carousel without having to swipe continuously.

```
{articlesdata.map((item) => {
  return (
    <View key={item.id}>
      <LongCard
        image={{ uri: item.imgUrl }}
        heading={item.heading}
        subheading={item.subHeading}
        onPress={() => {
          navigation.navigate("IndArticle", { paramKey: item });
        }}
      />
    </View>
  );
})}
```

```
const [articles, setArticles] = useState([]);
const articlesRef = firebase.firestore().collection("articles");

useEffect(() => {
  (async () => {
    articlesRef.onSnapshot((querySnapshot) => {
      const articles = [];
      querySnapshot.forEach((doc) => {
        const { heading, subHeading, imgUrl, text } = doc.data();
        articles.push({
          id: doc.id,
          heading,
          subHeading,
          imgUrl,
          text,
        });
      });
      setArticles(articles);
    });
  })();
}, []);

const articlesdata = articles.slice(0, 3);
```

```
export default function LongCard({ image, heading, subheading, onPress }) {
  return (
    <TouchableWithoutFeedback onPress={onPress}>
      <View style={styles.container}>
        <Image source={image} style={styles.image} />
        <View style={styles.txtCont}>
          <Text style={styles.headingTxt}>{heading}</Text>
          <Text numberOfLines={3} style={styles.subTxt}>
            {subheading}
          </Text>
        </View>
      </View>
    </TouchableWithoutFeedback>
  );
}
```

LongCard were used to display each of the article items. The filtered 3 items from the articles data will be map and present each of it in the explore page. For each object in the articles array will have id, heading, subheading, image url and text. We chose to create the card as it would be easier to display the information in a list on the explore page and there will be a clear distinction

between articles. The Longcard takes in the image, title, and subtitle. It will pass the selected item's data to IndArticle upon pressing.

```
<View style={styles.sectionCont}>
  <View style={styles.sectionHeader}>
    <Text style={styles.txtTitle}>FEEDS</Text>
    <TouchableOpacity
      onPress={() => navigation.navigate("Articles")}
      style={styles.viewAllBtn}
    >
      <View>
        <Text style={styles.viewAllTxt}>View All</Text>
      </View>
    </TouchableOpacity>
  </View>
```

```
<SafeAreaView>
  <View style={styles.headerCont}>
    <View style={styles.backBtn}>
      <GoBack onPress={() => navigation.goBack()} color={"black"} />
    </View>

    <Text style={styles.headerTxt}>Articles</Text>
  </View>
  <ScrollView>
    {articles.map((item, index) => {
      return (
        <View key={index}>
          <LongCard
            image={{ uri: item.imgUrl }}
            heading={item.heading}
            subheading={item.subHeading}
            onPress={() => {
              navigation.navigate("Article", { paramKey: item });
            }}
          />
        </View>
      );
    })}
  </ScrollView>
</SafeAreaView>
```

The "view all" button will navigate to AllArticles screen. Long card was also used for displaying each item and it works the same as the one in explore page.

```jsx
import React from "react";
import {
  Image,
  SafeAreaView,
  ScrollView,
  StyleSheet,
  Text,
  TouchableWithoutFeedback,
  View,
} from "react-native";

import GoBack from "../compenents/GoBack";

const ArticlePage = ({ route, navigation: { goBack } }) => {
  const dataSource = route.params.paramKey;

  return (
    <View style={{ flex: 1, backgroundColor: "white" }}>
      <SafeAreaView style={{ marginBottom: 100 }}>
        <ScrollView>
          <View style={styles.imgCont}>
            <Image source={{ uri: dataSource.imgUrl }} style={styles.img} />
            <View style={styles.imgOverlay} />
            <Text style={styles.title}>{dataSource.heading}</Text>
          </View>
          <View style={styles.backBtn}>
            <GoBack onPress={() => goBack()} color={"white"} />
          </View>

          <View style={styles.contentCont}>
            <Text style={styles.subHeading}>{dataSource.subHeading}</Text>
            {dataSource.text.map((item, index) => {
              return (
                <View key={index}>
                  <Text style={styles.txt}>{item}</Text>
                </View>
              );
            })}
          </View>
        </ScrollView>
      </SafeAreaView>
    </View>
  );
};
```

```jsx
import React from "react";
import { TouchableWithoutFeedback, Image, StyleSheet } from "react-native";

const GoBack = ({ onPress, color }) => {
  const styles = StyleSheet.create({
    img: {
      width: 30,
      height: 30,
      tintColor: color,
    },
  });
  return (
    <TouchableWithoutFeedback onPress={onPress}>
      <Image source={require("../assets/back.png")} style={styles.img} />
    </TouchableWithoutFeedback>
  );
};
```

For individual articles, the user will go back to the previous page upon clicking the back button which was imported from components. The title is placed within the centre of the article's image with an opaque black overlay. The subheading and text is placed in a scroll view as contents in the articles might be too long to view within the screen.

```
<View style={styles.sectionCont}>
  <View style={styles.sectionHeader}>
    <Text style={styles.txtTitle}>RECIPES</Text>
    <TouchableOpacity
      onPress={() => navigation.navigate("Recipes")}
      style={styles.viewAllBtn}
    >
      <View>
        <Text style={styles.viewAllTxt}>View All</Text>
      </View>
    </TouchableOpacity>
  </View>
</View>
<FlatList
  data={recipedata}
  keyExtractor={(item) => item.id.toString()}
  renderItem={({ item }) => (
    <TouchableWithoutFeedback
      onPress={() => {
        navigation.navigate("IndRecipe", { paramKey: item });
      }}
    >
      <View style={styles.item}>
        <View style={styles.itemCont}>
          <Image
            source={{ uri: item.imgUrl }}
            style={styles.itemImg}
          />
        </View>
        <Text style={styles.title}>{item.name}</Text>
      </View>
    </TouchableWithoutFeedback>
  )}
  horizontal
/>
```

For the recipes section, the item is displayed with the flatlist component from react native so that we can list each of the items one after another. We set the data in the flat list to recipedata, the key extractor to the data id and the rendered item, item which is used in the renderItem function. In the function it takes in the item name and imageurl. The flatlist is set to horizontal scroll where all the items are placed horizontally.

When the item is pressed, it will navigate to the recipes screen to view the content of the selected recipe and all details of the selected item will also be transferred over. There is also a view all button where it allows the user to navigate to the AllRecipe screen to view all the recipes from the database which work the same as the articles' one.

```
<View style={styles.headerCont}>
  <View style={styles.backBtn}>
    <GoBack onPress={() => navigation.goBack()} color={"black"} />
  </View>

  <Text style={styles.headerTxt}>Recipes</Text>
  <TouchableWithoutFeedback>
    <Image
      source={require("../assets/search.png")}
      style={{
        width: 30,
        height: 30,
        alignSelf: "center",
        right: 40,
        position: "absolute",
      }}
    />
  </TouchableWithoutFeedback>
</View>
{/* present recipes */}
<View style={{ alignItems: "center", height: "83%" }}>
  <FlatList
    data={recipes}
    keyExtractor={(item) => item.id.toString()}
    numColumns={2}
    key={(item) => item.id.toString()}
    renderItem={({ item }) => (
      <Card
        title={item.name}
        image={{ uri: item.imgUrl }}
        onPress={() => {
          navigation.navigate("Recipe", { paramKey: item });
        }}
        rate={3}
      />
    )}
  />
</View>
```

```
import { Rating } from "react-native-ratings";

export default function Card({ title, image, onPress, rate }) {
  return (
    <TouchableWithoutFeedback onPress={onPress}>
      <View style={styles.container}>
        <View style={styles.imgCont}>
          <Image source={image} style={styles.image} />
        </View>
        <View style={styles.detailContainer}>
          <Text style={styles.title}>{title}</Text>
        </View>

        <Rating
          imageSize={20}
          startingValue={rate}
          readonly
          style={styles.rating}
        />
      </View>
    </TouchableWithoutFeedback>
  );
}
```

A card component was used in the AllRecipe page where it displayed the item image, title and rating. We defined the key as item id and the numcolumns for the flatlist function as 2 so that there are two items in a row. It will navigate to the recipe screens and pass all the details when the card is pressed.

5.4.2 ShoppingList.js

```
const [shoppingList, setShoppingList] = useState([]);
const shoppingListRef = firebase.firestore().collection("shoppinglist");

useEffect(() => {
  (async () => {
    shoppingListRef.onSnapshot((querySnapshot) => {
      const shoppingList = [];
      querySnapshot.forEach((doc) => {
        const { name, qty, size, note } = doc.data();
        shoppingList.push({
          id: doc.id,
          name,
          note,
          size,
          qty,
        });
      });
      setShoppingList(shoppingList);
    });
  })();
}, []);
```

For the shopping list, there is item id and item name in the ingreShop array which is fetched from firebase.

```
<View style={{ marginTop: 10, padding: 20, flexDirection: "row" }}>
  <TouchableWithoutFeedback
    onPress={() => {
      navigation.navigate("ShopListSearch");
    }}
  >
    <View
      style={{
        width: "65%",
        height: 70,
        backgroundColor: "#e0e0e0",
        justifyContent: "center",
      }}
    >
      <Text
        style={{
          paddingLeft: 70,
          fontSize: 20,
          fontWeight: "600",
          color: "#c6c6c6",
          textTransform: "uppercase",
        }}
      >
        add an item
      </Text>
      <Image
        style={{
          position: "absolute",
          width: 40,
          height: 40,
          marginLeft: 20,
          tintColor: "#c6c6c6",
        }}
        source={require("../assets/addingre.png")}
      />
    </View>
  </TouchableWithoutFeedback>
```

When the search button is pressed, it will navigate to the shoplistsearch screens where there is a search button component for the user to search within the filterdata. The list of the ingredients for users to search up on is in the ingreShop array.

```
const [filterData, setFilterData] = useState(ingreShop);
const [search, setSearch] = useState("");
const [masterData, setmasterData] = useState(ingreShop);

const searchFilter = (text) => {
  if (text) {
    const newData = ingreShop.filter((item) => {
      const itemData = item.name ? item.name.toUpperCase() : "".toUpperCase();
      const textData = text.toUpperCase();
      return itemData.indexOf(textData) > -1;
    });
    setFilterData(newData);
    setSearch(text);
  } else {
    setFilterData(masterData);
    setSearch(text);
  }
};
```

We set the initial state of the filterData and the masterData as the ingreShop and search as an empty string. When the text is keyed onto the search bar, it will set the search state as the text typed and make it all to uppercase. Then, filter out the matching ingredients and set the filterdata as the filtered items. If there isn't any matching item it will set the filterdata to masterdata.

```
const addIngre = ({ item }) => {
  firebase
    .firestore()
    .collection("shoppinglist")
    .add({
      name: item.name,
      qty: "1",
      size: "",
      note: "",
    })
    .catch((error) => console.log(error));
};
```

```
const ItemRender = ({ item }) => (
  <TouchableWithoutFeedback
    onPress={() => {
      addIngre({ item });
      goBack();
    }}
  >
    <Text style={{ fontSize: 20, paddingLeft: 20, paddingVertical: 10 }}>
      {item.name}
    </Text>
  </TouchableWithoutFeedback>
);

const ItemSeperator = () => {
  return (
    <View
      style={{
        backgroundColor: "grey",
        height: 2,
      }}
    />
  );
};
```

```
<View style={{ marginTop: 10, padding: 20, flexDirection: "row" }}>
  <SearchBtn
    width={"75%"}
    image={require("../assets/addingre.png")}
    value={search}
    onChangeText={(text) => searchFilter(text)}
    placeholder="add an item"
  />
  <TouchableWithoutFeedback onPress={() => goBack()}>
    <Image
      source={require("../assets/close.png")}
      style={{
        width: 30,
        height: 30,
        tintColor: "grey",
        alignSelf: "center",
        marginLeft: 30,
      }}
    />
  </TouchableWithoutFeedback>
</View>
<FlatList
  data={filterData}
  renderItem={({ item }) => <ItemRender item={item} />}
  keyExtractor={(item) => item.id}
  ItemSeparatorComponent={ItemSeperator}
/>
```

When the item is pressed, the addIngre function will add the name and set the qty to 1 and the size and note to empty string. Each of the items listed in the filterdata will be separated with ItemSeperator where it is just one single grey coloured line.

```
const deleteAll = () => {
  shoppingList.map((item) => {
    firebase.firestore().collection("shoppinglist").doc(item.id).delete();
  });
};
```

```
<TouchableWithoutFeedback onPress={() => deleteAll()}>
  <View
    style={{
      padding: 5,
      backgroundColor: "white",
      shadowColor: "black",
      shadowOpacity: 0.7,
      justifyContent: "center",
      marginLeft: 10,
      shadowOffset: { width: 2, height: 2 },
    }}
  >
    <Text
      style={{
        fontSize: 18,
        textTransform: "uppercase",
        fontWeight: "bold",
      }}
    >
      clear all
    </Text>
  </View>
</TouchableWithoutFeedback>
```

For the clear all button, it will loop through all the items in the shoppingList and delete it from firebase based on the item's id.

```
const [shoppingList, setShoppingList] = useState([]);
const shoppingListRef = firebase.firestore().collection("shoppinglist");

useEffect(() => {
  (async () => {
    shoppingListRef.onSnapshot((querySnapshot) => {
      const shoppingList = [];
      querySnapshot.forEach((doc) => {
        const { name, qty, size, note } = doc.data();
        shoppingList.push({
          id: doc.id,
          name,
          note,
          size,
          qty,
        });
      });
      setShoppingList(shoppingList);
    });
  })();
}, []);
```

The items shown on the shopping list are all fetched from the shoppinglist collections in firebase.

```
return (
  <Swipeable renderRightActions={leftSwipe}>
    <View key={index}>
      <View
        style={{
          flexDirection: "row",
          padding: 10,
          alignItems: "center",
          marginLeft: 10,
        }}
      >
        <Checkedbox />
        <Text style={{ margin: 5 }}>{item.name}</Text>
        <Text style={{ margin: 5, color: "grey" }}>{item.qty}</Text>
        <Text style={{ margin: 5, color: "grey" }}>{item.size}</Text>
        <TouchableWithoutFeedback
          onPress={() => {
            modaltxt({ item });
            setModalVisible(true);
          }}
        >
          <Image
            source={require("../assets/edit.png")}
            style={{
              width: 20,
              height: 20,
              tintColor: "grey",
              right: 30,
              position: "absolute",
            }}
          />
        </TouchableWithoutFeedback>
      </View>
    </View>
  </Swipeable>
```

For each of the item, it is mapped from shoppinglist and display it with the item name, qty, size, note button and a swipe able delete button.

```
<Modal
  //animationType={"slide"}
  transparent={true}
  visible={modalVisible}
  onRequestClose={() => {
    console.log("Modal has been closed.");
  }}
>
  <View
    style={{
      flex: 1,
      alignItems: "center",
      justifyContent: "center",
      backgroundColor: "rgba(0,0,0,0.5)",
      padding: 100,
    }}
  >
    <View
      style={{
        width: 280,
        height: 400,
        backgroundColor: "white",
      }}
    >
      <View
        style={{
          flexDirection: "row",
          padding: 10,
          margin: 10,
          alignItems: "center",
        }}
      >
        <TouchableWithoutFeedback
          onPress={() => {
            setModalVisible(!modalVisible);
          }}
        >
          <Image
            source={require("../assets/close.png")}
            style={{
              width: 20,
              height: 20,
            }}
          />
        </TouchableWithoutFeedback>

        <TouchableWithoutFeedback
          onPress={() => {
            updateItem(selId);
            setModalVisible(!modalVisible);
          }}
        >
          <Text
            style={{
              right: 5,
              position: "absolute",

              fontSize: 15,
              textTransform: "uppercase",
              color: "grey",
              fontWeight: "bold",
            }}
          >
            Save
          </Text>
        </TouchableWithoutFeedback>
      </View>

      <View
        style={{
          padding: 10,
          borderTopColor: "grey",
          borderTopWidth: 2,
        }}
      >
        <Text
          style={{
            color: "grey",
            fontWeight: "bold",
            textTransform: "uppercase",
            marginBottom: 5,
          }}
        >
          Name
        </Text>
        <TextInput defaultValue={nameInput} editable={false} />
      </View>
```

```
<View
  style={{
    borderTopColor: "grey",
    borderTopWidth: 2,
    flexDirection: "row",
  }}
>
  <View style={{ padding: 10 }}>
    <Text
      style={{
        color: "grey",
        fontWeight: "bold",
        textTransform: "uppercase",
        marginBottom: 5,
      }}
    >
      quantity
    </Text>
    <TextInput
      defaultValue={qtyInput}
      onChangeText={(text) => setQtyInput(text)}
      onEndEditing={() => setQtyInput(qtyInput)}
      editable
    />
  </View>
  <View
    style={{
      marginLeft: 40,
      borderLeftColor: "grey",
      borderLeftWidth: 2,
      padding: 10,
    }}
  >
    <Text
      style={{
        color: "grey",
        fontWeight: "bold",
        textTransform: "uppercase",
        marginBottom: 5,
      }}
    >
      size
    </Text>
    <TextInput
      defaultValue={sizeInput}
      onChangeText={(text) => setSizeInput(text)}
      onEndEditing={() => setSizeInput(sizeInput)}
      editable
    />
  </View>
</View>
<View
  style={{
    padding: 10,
    borderTopColor: "grey",
    borderTopWidth: 2,
  }}
>
  <Text
    style={{
      color: "grey",
      fontWeight: "bold",
      textTransform: "uppercase",
      marginBottom: 5,
    }}
  >
    note
  </Text>
  <TextInput
    keyboardType="default"
    returnKeyType="done"
    blurOnSubmit={true}
    onSubmitEditing={() => {
      Keyboard.dismiss();
    }}
    onChangeText={(text) => setNoteInput(text)}
    onEndEditing={() => setNoteInput(noteInput)}
    defaultValue={noteInput}
    editable
    multiline
  />
</View>
</View>
</View>
</Modal>
```

When the note button is pressed, it will call out a modal where the transparent is assigned as true so the previous screen still can be seen.

```
const [modalVisible, setModalVisible] = useState(false);
const [nameInput, setNameInput] = useState("");
const [qtyInput, setQtyInput] = useState("");
const [sizeInput, setSizeInput] = useState("");
const [noteInput, setNoteInput] = useState("");
const [selId, setSelId] = useState([]);

const modaltxt = ({ item }) => {
  setNameInput(item.name);
  setQtyInput(item.qty);
  setSizeInput(item.size);
  setNoteInput(item.note);
  setSelId({ item });
};
```

The visible is assigned as modalVisible where the initial state was set to false, and the state will changed to true when the button is pressed. In the modal, we used textInput to place the name, qty, size and note of the item. The editable for name's textinput is set to false as we only allow user to edit the qty, size and note. To update the edited text, we first declare the input for each detail as empty string. Then when the text is changed, we set the input as text type, also onEndEditing the input is set with the final input after the user finishes editing.

```
const updateItem = ({ item }) => {
  firebase.firestore().collection("shoppinglist").doc(item.id).set({
    name: nameInput,
    qty: qtyInput,
    size: sizeInput,
    note: noteInput,
  });
};
```

When the save button is pressed, the updateItem function will be called to set the updated text with all the respective input and set the firebase item according to the item id.

```
const deleteItem = ({ item }) => {
  firebase.firestore().collection("shoppinglist").doc(item.id).delete();
};
```

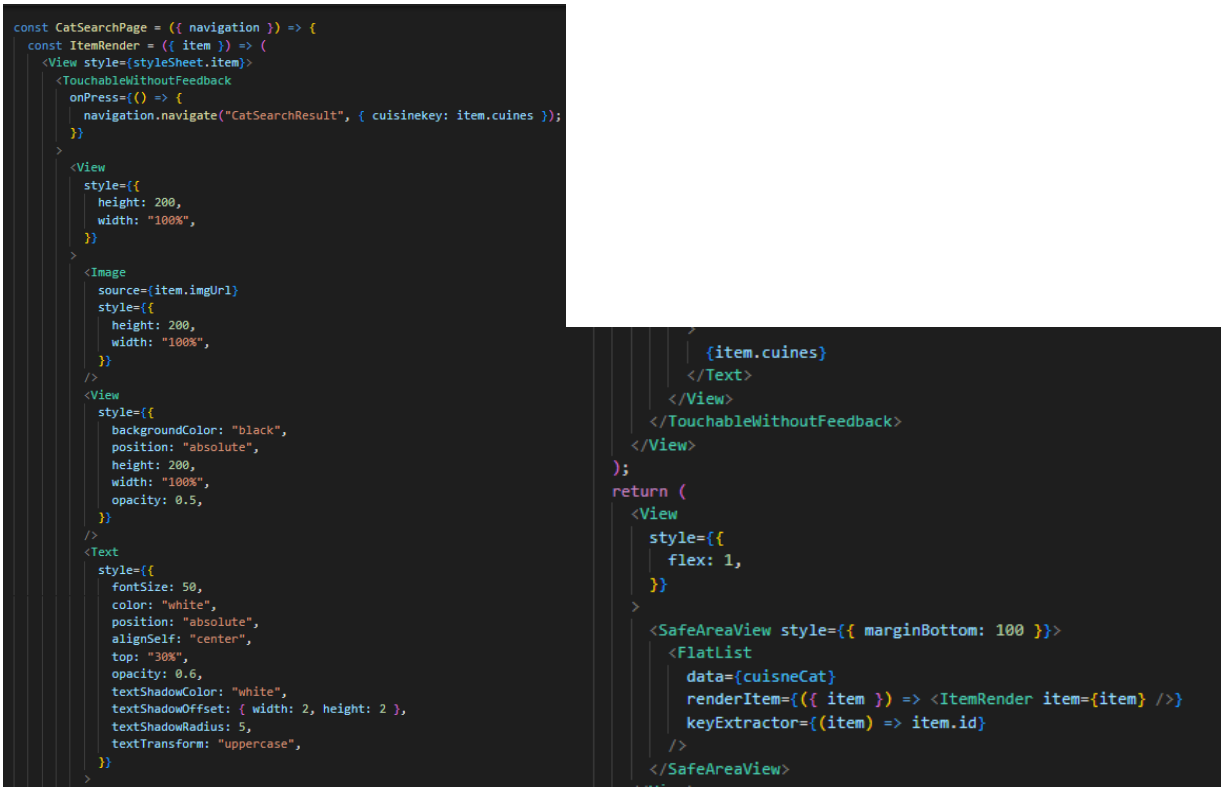
```
{shoppingList.map((item, index) => {
  const leftSwipe = (progress, dragX) => {
    const scale = dragX.interpolate({
      inputRange: [-150, 0],
      outputRange: [1, 0],
      extrapolate: "clamp",
    });
    return (
      <TouchableWithoutFeedback
        onPress={() => deleteItem({ item })}
        activeOpacity={0.6}
      >
        <View style={styleSheet.deleteBox}>
          <Animated.Image
            source={require("../assets/delete.png")}
            style={{
              width: 30,
              height: 30,
              tintColor: "white",
              transform: [{ scale: scale }],
            }}
          />
        </View>
      </TouchableWithoutFeedback>
    );
  };
```

When the item list is swiped, a delete button will be shown with red background. We set selId as empty string and when the button is pressed it will set the selId to the selected item id. Then we call the deleteItem function and delete the item from the firebase according to the id.

5.4.3 SearchByCat.js

```
const cuisneCat = [
  {
    id: 1,
    cuines: "asian",
    imgUrl: require("../assets/asian.jpg"),
  },
  {
    id: 2,
    cuines: "western",
    imgUrl: require("../assets/western.jpg"),
  },
  {
    id: 3,
    cuines: "korean",
    imgUrl: require("../assets/korean.jpg"),
  },
  {
    id: 4,
    cuines: "chinese",
    imgUrl: require("../assets/dish.jpg"),
  },
];
```

We have a constant name cuisineCat which lists out the category name and the image url. The flatlist component is used to display each category.



```
const CatSearchPage = ({ navigation }) => {
  const ItemRender = ({ item }) => (
    <View style={styleSheet.item}>
      <TouchableWithoutFeedback
        onPress={() => {
          navigation.navigate("CatSearchResult", { cuisinekey: item.cuines });
        }}
      >
        <View
          style={{
            height: 200,
            width: "100%",
          }}
        >
          <Image
            source={item.imgUrl}
            style={{
              height: 200,
              width: "100%",
            }}
          />
          <View
            style={{
              backgroundColor: "black",
              position: "absolute",
              height: 200,
              width: "100%",
              opacity: 0.5,
            }}
          />
          <Text
            style={{
              fontSize: 50,
              color: "white",
              position: "absolute",
              alignSelf: "center",
              top: "30%",
              opacity: 0.6,
              textShadowColor: "white",
              textShadowOffset: { width: 2, height: 2 },
              textShadowRadius: 5,
              textTransform: "uppercase",
            }}
          >
            {item.cuines}
          </Text>
        </View>
      </TouchableWithoutFeedback>
    </View>
  );
  return (
    <View
      style={{
        flex: 1,
      }}
    >
      <SafeAreaView style={{ marginBottom: 100 }}>
        <FlatList
          data={cuisneCat}
          renderItem={({ item }) => <ItemRender item={item} />}
          keyExtractor={(item) => item.id}
        />
      </SafeAreaView>
```

The cuisineCat is used as the data and for each of the rendered items, the item is passed to the ItemRender function. In the ItemRender function, we will have an image placed as a background and there is also a black overlay so that text can be seen clearly. The cuisine title from the cuisineCat is placed at the middle of the image. When the item is pressed, it will navigate to the CatSearchResult and pass the selected item.cuisine to the page.

```
const recipeRef = firebase.firestore().collection("recipes");
const [recipe, setRecipe] = useState([]);
const choosenCat = route.params.cuisinekey;
//const [filterRecipe, setFilterRecipe] = useState([])

useEffect(() => {
  (async () => {
    recipeRef.onSnapshot((querySnapshot) => {
      const recipe = [];
      querySnapshot.forEach((doc) => {
        const {
          calories,
          cuisinecat,
          dishcat,
          imgUrl,
          ingre,
          name,
          nutritions,
          steps,
        } = doc.data();
        recipe.push({
          id: doc.id,
          name,
          cuisinecat,
          dishcat,
          imgUrl,
          ingre,
          calories,
          nutritions,
          steps,
        });
      });
      setRecipe(recipe);
    });
  })();
}, []);

const filterRecipe = recipe.filter((item) =>
  item.cuisinecat.includes(choosenCat)
);
```

```
<SafeAreaView style={{ marginBottom: 100, marginTop: 30 }}>
  <View style={styles.headerCont}>
    <View style={styles.backBtn}>
      <GoBack onPress={() => navigation.goBack()} color={"black"} />
    </View>

    <Text style={styles.headerTxt}>{choosenCat}</Text>
    <TouchableWithoutFeedback>
      <Image
        source={require("../assets/search.png")}
        style={{
          width: 30,
          height: 30,
          alignSelf: "center",
          right: 40,
          position: "absolute",
        }}
      />
    </TouchableWithoutFeedback>
  </View>
  {/* present recipes */}
  <View style={{ alignItems: "center" }}>
    <FlatList
      data={filterRecipe}
      keyExtractor={(item) => item.id.toString()}
      numColumns={2}
      key={(item) => item.id.toString()}
      renderItem={({ item }) => (
        <Card
          title={item.name}
          image={{ uri: item.imgUrl }}
          onPress={() => {
            navigation.navigate("Recipes", { paramKey: item });
          }}
          rate={3}
        />
      )}
    />
  </View>
</SafeAreaView>
```

In the CatSearchResult page, as usual we will fetch the data from firebase. In the recipe constant, we have document id,name,cuisine category, dish category, img url, ingredient list, calories, nutritions and steps. Then declare another constant called filterRecipe where it will filter out the recipe array with the same selected category passed from previous screens. The recipe is presented the same as how recipes were presented in the Recipe page.

5.4.4 SearchByIngre.js

```
const listTab = [
  { cat: "All" },
  { cat: "Meats" },
  { cat: "Vegetables" },
  { cat: "Fruits" },
  { cat: "Dairy" },
  { cat: "Seafood" },
];

const ingrelist = [
  {
    id: 1,
    name: "apple",
    icon: require("../assets/category/apple.png"),
    cat: "Fruits",
  },
  {
    id: 2,
    name: "beef",
    icon: require("../assets/category/beef.png"),
    cat: "Meats",
  },
  {
    id: 3,
    name: "cabbage",
    icon: require("../assets/category/cabbage.png"),
    cat: "Vegetables",
  },
  {
    id: 4,
    name: "cheese",
    icon: require("../assets/category/cheese.png"),
    cat: "Dairy",
  },
  {
    id: 5,
    name: "chicken",
    icon: require("../assets/category/chicken.png"),
    cat: "Meats",
  },
```

```
<SearchBtn
  width={"80%"}
  image={require("../assets/search.png")}
  placeholder="Search"
/>
```

We have a search bar which is imported from component. We have a constant array with the ingredients name, icon url, category and id. There is also another constant for the ingredients category name.

```
const [cat, setCat] = useState("All");
const [datalist, setDatalist] = useState(ingrelist);
const [test, setTest] = useState(true);
const setCatFilter = (cat) => {
  if (cat !== "All") {
    setDatalist([...ingrelist.filter((e) => e.cat === cat)]);
    setTest(false);
  } else {
    setDatalist(ingrelist);
    setTest(true);
  }
  setCat(cat);
};
```

```
<ScrollView horizontal={true}>
  {listTab.map((e) => (
    <TouchableOpacity
      style={[styles.btnTab, cat === e.cat && styles.btnTabActive]}
      onPress={() => setCatFilter(e.cat)}
    >
      <Text
        style={[styles.textTab, cat === e.cat && styles.txtTabActive]}
      >
        {e.cat}
      </Text>
    </TouchableOpacity>
  ))}
</ScrollView>
```

The ingredients category is shown by using map function, by looping each of the category in the listTab and each tab is created with touchable opacity and the category name is included in it. When it is pressed, it will call the setCatFilter where it will change the datalist according to the selected category name. The list tab is presented horizontally.

```
<View style={{ alignSelf: "center", height: "60%" }}>
  <FlatList
    data={datalist}
    keyExtractor={(e, i) => i.toString()}
    numColumns={3}
    key={(item) => item.id.toString()}
    renderItem={({ item }) => (
      <IngreListItem name={item.name} icon={item.icon} />
    )}
  />
</View>
```

Flatlist component is used to render the data in the ingredient array. For each of the item, it will call the ingrelistitem where item name and icon was used. For each of the ingredients it is shown as a container with the icon and category name in it. We included icon because it is more visually appealing compared to just text alone.

```
export default function IngreListItem({ name, icon }) {
  const [selIngre, setSelIngre] = useState(false);
  const [selIngreBg, setSelIngreBg] = useState("white");

  return (
    <TouchableWithoutFeedback
      onPress={() => {
        if (selIngre == "white") {
          setSelIngre("#e9edc9");
        } else {
          setSelIngre("white");
        }
      }}
    >
      <View
        style={{
          alignItems: "center",
          padding: 10,
          borderColor: "black",
          borderWidth: 2,
          backgroundColor: selIngre,
        }}
      >
        <Image source={icon} style={{ width: 100, height: 100 }} />
        <Text>{name}</Text>
      </View>
    </TouchableWithoutFeedback>
  );
}
```

When the item is selected the background colour will change to show the user which ingredients they had selected.

```
const [modalVisible, setModalVisible] = useState(false);
```

```
<TouchableWithoutFeedback onPress={() => setModalVisible(true)}>
  <View
    style={{
      alignSelf: "center",
      marginTop: 10,
      paddingHorizontal: 50,
      paddingVertical: 15,
      backgroundColor: "#d4a373",
      shadowColor: "#4c411a",
      shadowRadius: 4,
      shadowOffset: { width: 1, height: 1 },
      shadowOpacity: 0.7,
    }}
  >
    <Modal

      transparent={true}
      visible={modalVisible}
      onRequestClose={() => {
        console.log("Modal has been closed.");
      }}
    >
      <View
        style={{
          flex: 1,
          alignItems: "center",
          justifyContent: "center",
          backgroundColor: "rgba(0,0,0,0.5)",
          padding: 100,
        }}
      >
        <View
          style={{
            width: 280,
            height: 300,
            backgroundColor: "white",
          }}
        >
          <View style={{ padding: 20 }}>
            <Text style={{ fontSize: 15, fontWeight: "700" }}>
              Are you sure that you want to search an recipe with the
              following ingredients?
            <View style={{ marginLeft: 20, marginTop: 20 }}>
              <Text style={{ fontSize: 15, paddingVertical: 10 }}>
                - Chicken
              </Text>
              <Text style={{ fontSize: 15, paddingVertical: 10 }}>
                - Parsley
              </Text>
            </View>
            <View
              style={{
                marginTop: 20,
                flexDirection: "row",
                alignSelf: "center",
              }}
            >
              <TouchableWithoutFeedback
                onPress={() => setModalVisible(false)}
              >
                <View
                  style={{
                    paddingVertical: 10,
                    paddingHorizontal: 20,
                    backgroundColor: "#ededed",
                    shadowColor: "black",
                    shadowOpacity: 0.7,
                    margin: 10,
                  }}
                >
                  <Text
                    style={{
                      fontSize: 15,
                      fontWeight: "bold",
                      textTransform: "uppercase",
                      textAlign: "center",
                    }}
                  >
                    Back
                  </Text>
                </View>
              </TouchableWithoutFeedback>
              <TouchableWithoutFeedback>
                <View
                  style={{
                    paddingVertical: 10,
```

We set the modalVisible state as false and it will be set to true when the proceed button is pressed. In the modal we have the list of item that had been added and also there are button to go back to previous screen or match button to match the recipes.

5.4.5 CounterCalories.js

```
<View>
  <TouchableWithoutFeedback
    onPress={() => {
      navigation.navigate("IngreSearch");
    }}
  >
    <View
      style={{
        flexDirection: "row",
        marginLeft: 20,
      }}
    >
      <View
        style={{
          width: 20,
          height: 20,
          borderRadius: 20 / 2,
          borderWidth: 2,
          borderColor: "black",
          alignItems: "center",
          justifyContent: "center",
        }}
      >
        <Text
          style={{
            fontWeight: "bold",
          }}
        >
          +
        </Text>
      </View>
      <Text
        style={{
          marginLeft: 10,
          fontWeight: "bold",
        }}
      >
        Add Ingredients
      </Text>
    </View>
  </TouchableWithoutFeedback>
</View>
```

```
<SearchBtn
  width={"70%"}
  image={require("../assets/search.png")}
  value={search}
  onChangeText={(text) => searchFilter(text)}
  placeholder="Search"
/>
```

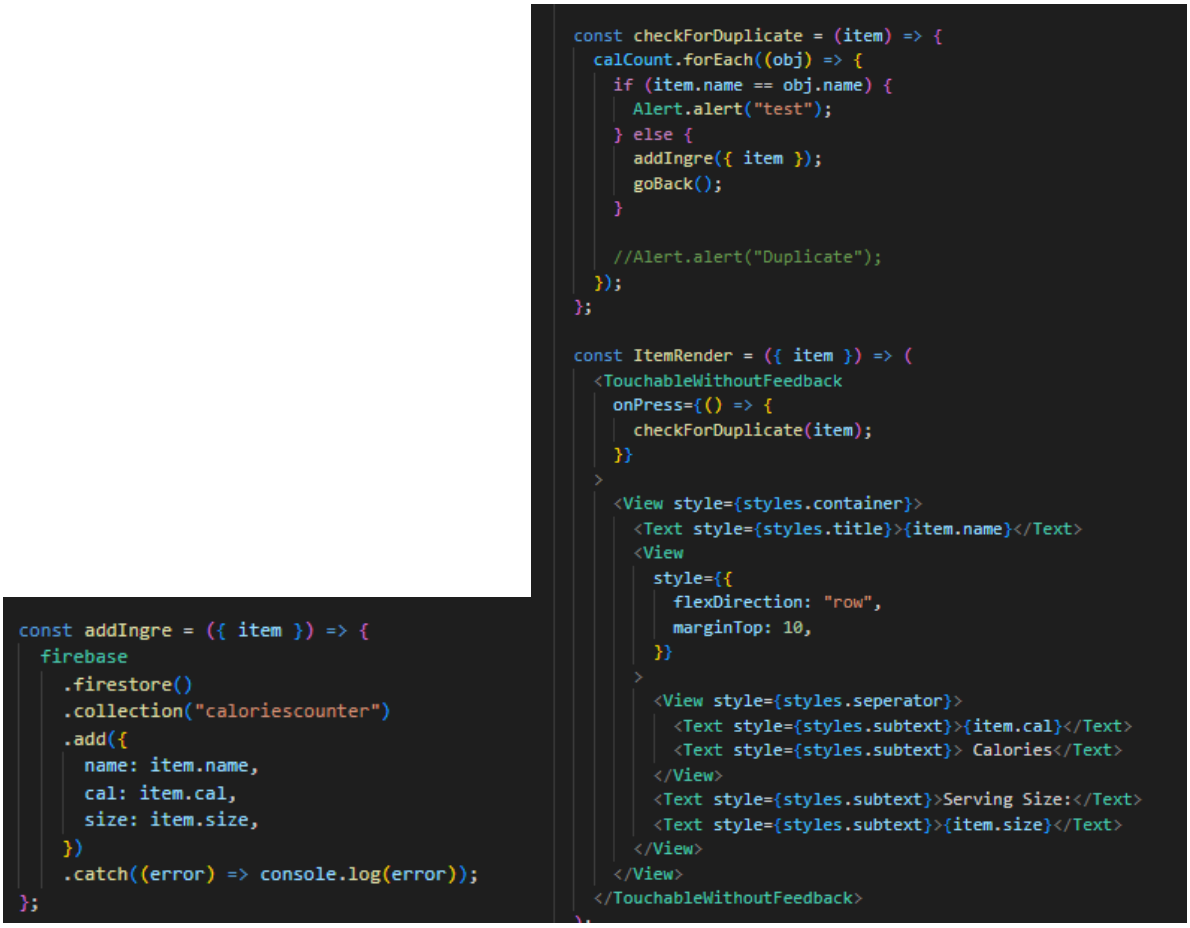
When user pressed on the “add ingredients” button it will navigate to CaloriesIngreSearch. In this screen, it contain a search bar which is included in the components folder.

```
import React from "react";
import { View, Text, TextInput, StyleSheet, Image } from "react-native";

export default function SearchBtn({
  value,
  onChangeText,
  placeholder,
  width,
  image,
}) {
  return (
    <View style={[styles.searchBarCont, { width: width }]}>
      <TextInput
        value={value}
        onChangeText={onChangeText}
        placeholder={placeholder.toUpperCase()}
        style={styles.searchBar}
        placeholderTextColor="#c6c6c6"
      />
      <Image style={styles.image} source={image} />
    </View>
  );
}
```

For the search bar btn, it was created with a textinput where it will display the keyboard and allow user to type. The height of the search bar is fixed, but user are able to customise the placeholder text, the image included in the search bar and also when the text changed.

```
const [filterData, setFilterData] = useState(ingreCal);
const [search, setSearch] = useState("");
const [masterData, setmasterData] = useState(ingreCal);

const searchFilter = (text) => {
  if (text) {
    const newData = ingreCal.filter((item) => {
      const itemData = item.name ? item.name.toUpperCase() : "".toUpperCase();
      const textData = text.toUpperCase();
      return itemData.indexOf(textData) > -1;
    });
    setFilterData(newData);
    setSearch(text);
  } else {
    setFilterData(masterData);
    setSearch(text);
  }
};
```

The search bar search through the ingreCal list and filter the matching variable out from the list. Do note that the textInput and the word search will both be uppercased.



```
const addIngre = ({ item }) => {
  firebase
    .firestore()
    .collection("caloriescounter")
    .add({
      name: item.name,
      cal: item.cal,
      size: item.size,
    })
    .catch((error) => console.log(error));
};
```

```
const checkForDuplicate = (item) => {
  calCount.forEach((obj) => {
    if (item.name == obj.name) {
      Alert.alert("test");
    } else {
      addIngre({ item });
      goBack();
    }

    //Alert.alert("Duplicate");
  });
};

const ItemRender = ({ item }) => (
  <TouchableWithoutFeedback
    onPress={() => {
      checkForDuplicate(item);
    }}
  >
    <View style={styles.container}>
      <Text style={styles.title}>{item.name}</Text>
      <View
        style={{
          flexDirection: "row",
          marginTop: 10,
        }}
      >
        <View style={styles.seperator}>
          <Text style={styles.subtext}>{item.cal}</Text>
          <Text style={styles.subtext}> Calories</Text>
        </View>
        <Text style={styles.subtext}>Serving Size:</Text>
        <Text style={styles.subtext}>{item.size}</Text>
      </View>
    </View>
  </TouchableWithoutFeedback>
```

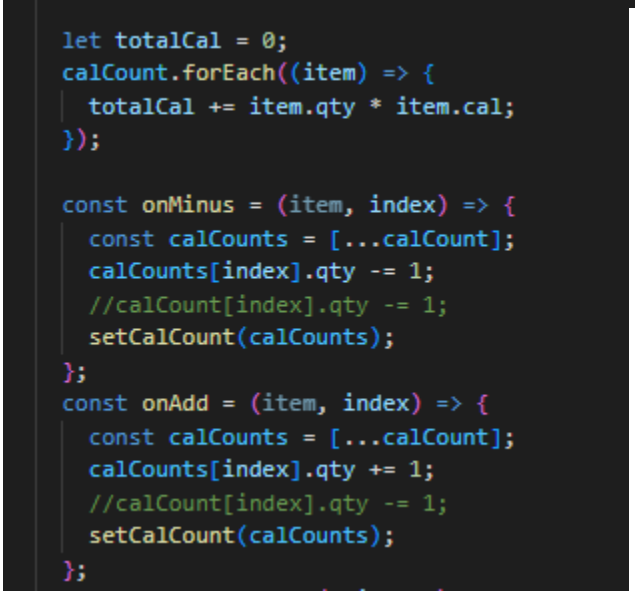
When the item in the search result are pressed, it will call the addData function where it will add the data to the the collection called caloriescounter in firestore.

```
const [calCount, setCalCount] = useState([]);
const calCountRef = firebase.firestore().collection("caloriescounter");

useEffect(() => {
  (async () => {
    calCountRef.onSnapshot((querySnapshot) => {
      const calCount = [];
      querySnapshot.forEach((doc) => {
        const { name, cal, size } = doc.data();
        calCount.push({
          id: doc.id,
          name,
          cal,
          size,
          qty: 1,
        });
      });
      setCalCount(calCount);
    });
  })();
}, []);
```

It will then proceed back to the main page of the Calories counter. The data displayed on this page are fetched from the calorie counter collection.

```
export default function CalListItem({ item, onAdd, onMinus }) {
  // const [count, setCount] = useState(1);
  const total = Number(item.cal) * item.qty;
  return (
    <View
      style={{
        // top: 70,
        marginLeft: 20,
        flexDirection: "row",
      }}
    >
      <View style={{ width: "40%", padding: 10 }}>
        <View>
          <Text>{item.name}</Text>
        </View>
        <View style={{ flexDirection: "row" }}>
          <Text>{item.cal}</Text>
          <Text> calories</Text>
        </View>
        <View style={{ flexDirection: "row" }}>
          <Text> Serving Size: </Text>
          <Text>{item.size}</Text>
        </View>
      </View>
      <View style={{ marginHorizontal: 20, top: 10 }}>
        <View style={{ flexDirection: "row" }}>
          <TouchableWithoutFeedback disabled={item.qty == 1} onPress={onMinus}>
            <View
              style={{
                backgroundColor: "white",
                width: 20,
                height: 20,
                borderRadius: 20 / 2,
                borderColor: "black",
                borderWidth: 1.5,
                marginRight: 15,
              }}
            >
              <Text
                style={{
                  textAlign: "center",
                  fontSize: 14,
                }}
              >
```

```
        </View>
      </TouchableWithoutFeedback>
      <Text
        style={{
          fontSize: 12,
        }}
      >
        {item.qty}
      </Text>

      <TouchableWithoutFeedback onPress={onAdd}>
        <View
          style={{
            backgroundColor: "white",
            width: 20,
            height: 20,
            borderRadius: 20 / 2,
            borderColor: "black",
            borderWidth: 1.5,
            marginLeft: 15,
          }}
        >
          <Text style={{ textAlign: "center", fontSize: 14 }}>+</Text>
        </View>
      </TouchableWithoutFeedback>
    </View>
  </View>

  <View style={{ marginLeft: 30, flexDirection: "row", top: 10 }}>
    <Text>{total.toString()}</Text>
    <Text> CAL</Text>
  </View>
</View>
```

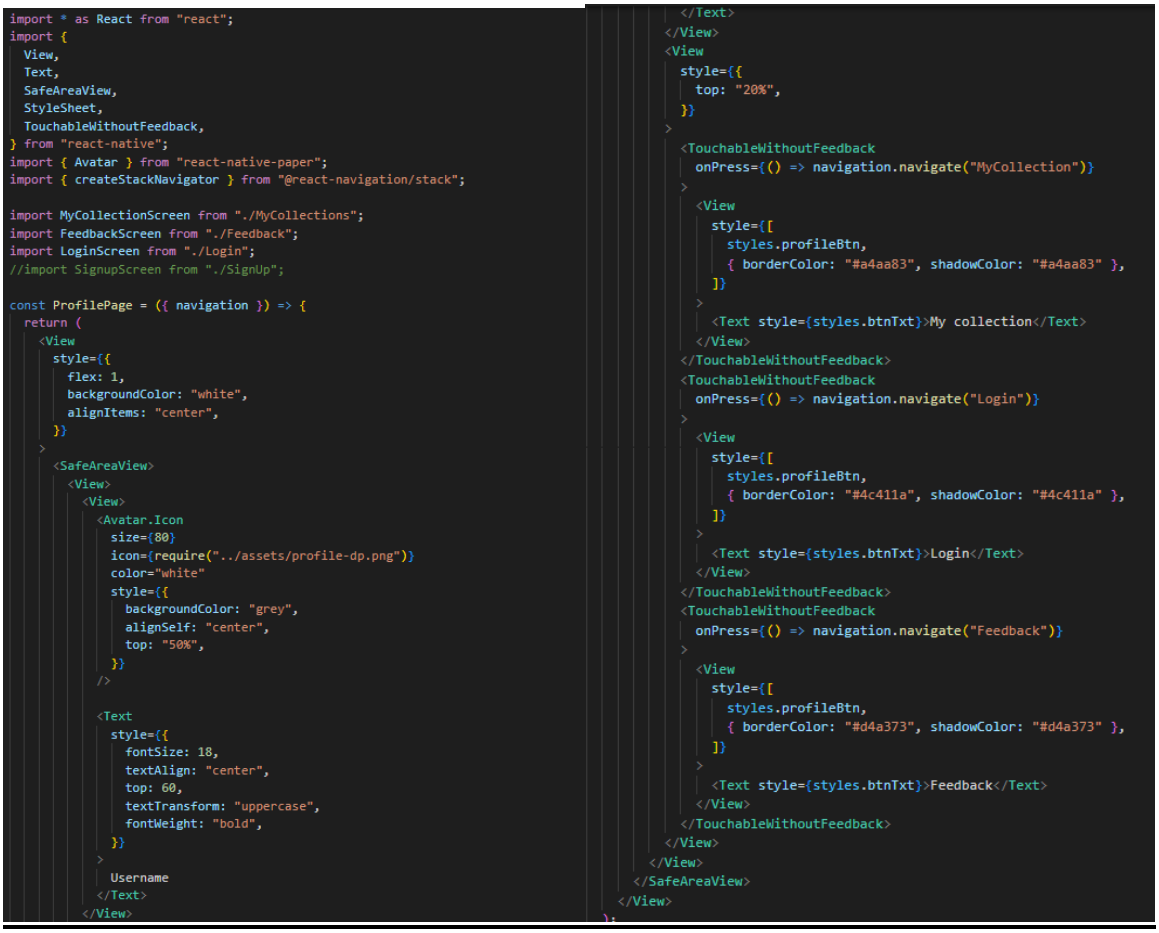


```
let totalCal = 0;
calCount.forEach((item) => {
  totalCal += item.qty * item.cal;
});

const onMinus = (item, index) => {
  const calCounts = [...calCount];
  calCounts[index].qty -= 1;
  //calCount[index].qty -= 1;
  setCalCount(calCounts);
};
const onAdd = (item, index) => {
  const calCounts = [...calCount];
  calCounts[index].qty += 1;
  //calCount[index].qty -= 1;
  setCalCount(calCounts);
};
```

For each of the item, there is a counter from callistitem component and total calories of the ingredients according to its serving size. When the minus button is pressed it will call the minus function where it will minus the quantity and update the quantity in the array. Same goes to the add button but it will call the add function instead. There is a swipe to delete button which works the same as the one in shopping list.

The cal for each item is calculated by multiplying the quantity and the item calories. The overall calories count is found by looping through every item in the array and multiple the quantity by the calories then totals it up.

5.4.6 Profile.js

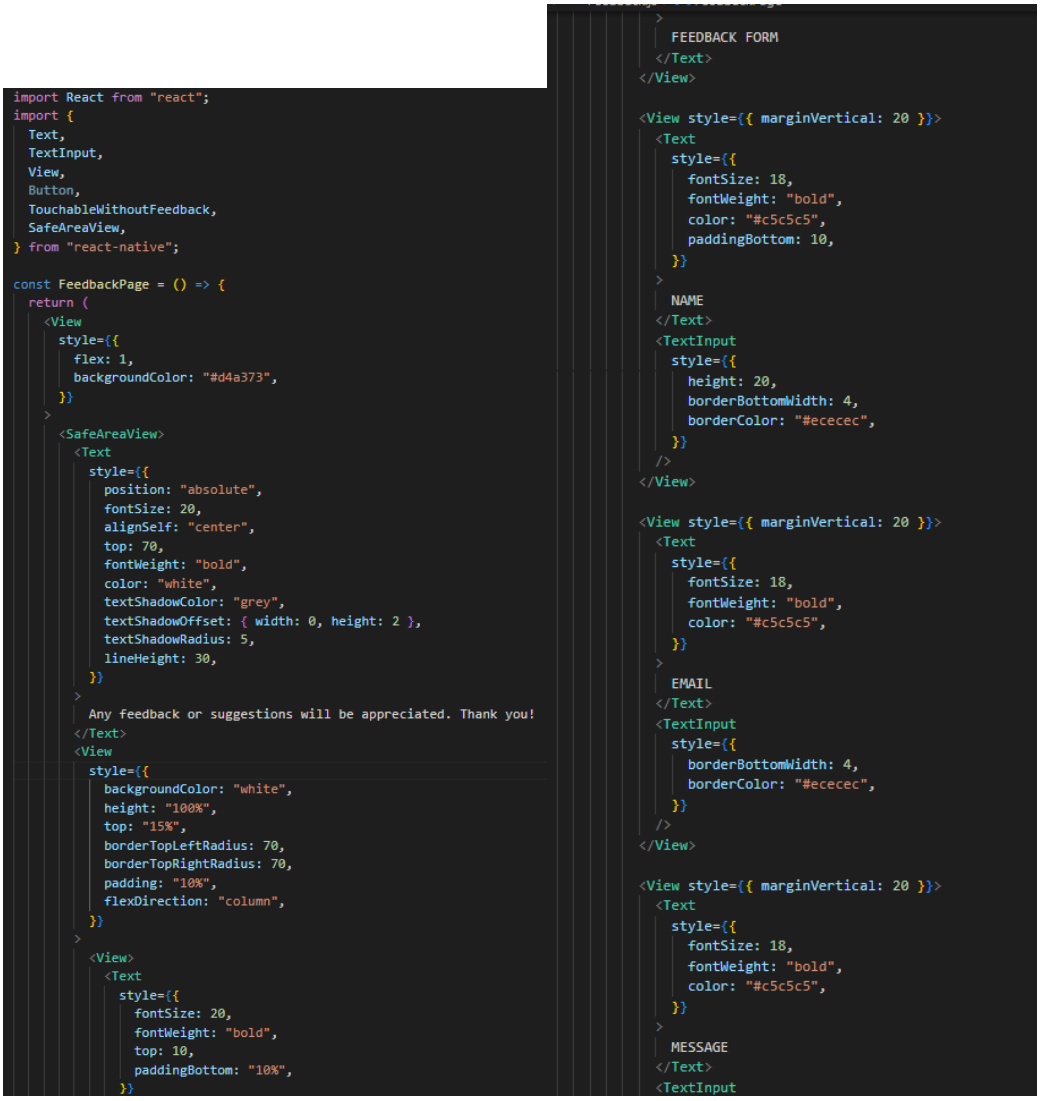


```
import * as React from "react";
import {
  View,
  Text,
  SafeAreaView,
  StyleSheet,
  TouchableWithoutFeedback,
} from "react-native";
import { Avatar } from "react-native-paper";
import { createStackNavigator } from "@react-navigation/stack";

import MyCollectionScreen from "./MyCollections";
import FeedbackScreen from "./Feedback";
import LoginScreen from "./Login";
//import SignupScreen from "./SignUp";

const ProfilePage = ({ navigation }) => {
  return (
    <View
      style={{
        flex: 1,
        backgroundColor: "white",
        alignItems: "center",
      }}
    >
      <SafeAreaView>
        <View>
          <View>
            <Avatar.Icon
              size={80}
              icon={require("../assets/profile-dp.png")}
              color="white"
              style={{
                backgroundColor: "grey",
                alignSelf: "center",
                top: "50%",
              }}
            />

            <Text
              style={{
                fontSize: 18,
                textAlign: "center",
                top: 60,
                textTransform: "uppercase",
                fontWeight: "bold",
              }}
            >
              Username
            </Text>
          </View>
          </Text>
        </View>
        <View
          style={{
            top: "20%",
          }}
        >
          <TouchableWithoutFeedback
            onPress={() => navigation.navigate("MyCollection")}
          >
            <View
              style={[
                styles.profileBtn,
                { borderColor: "#a4aa83", shadowColor: "#a4aa83" },
              ]}
            >
              <Text style={styles.btnTxt}>My collection</Text>
            </View>
          </TouchableWithoutFeedback>
          <TouchableWithoutFeedback
            onPress={() => navigation.navigate("Login")}
          >
            <View
              style={[
                styles.profileBtn,
                { borderColor: "#4c411a", shadowColor: "#4c411a" },
              ]}
            >
              <Text style={styles.btnTxt}>Login</Text>
            </View>
          </TouchableWithoutFeedback>
          <TouchableWithoutFeedback
            onPress={() => navigation.navigate("Feedback")}
          >
            <View
              style={[
                styles.profileBtn,
                { borderColor: "#d4a373", shadowColor: "#d4a373" },
              ]}
            >
              <Text style={styles.btnTxt}>Feedback</Text>
            </View>
          </TouchableWithoutFeedback>
        </View>
      </View>
    </SafeAreaView>
  </View>
);
```

The profile page contains the user's profile picture and user's username. There is also the Mycollection button, Login Button and Feedback button.

```
import React from "react";
import {
  Text,
  View,
  StyleSheet,
  ScrollView,
  Image,
  SafeAreaView,
  TouchableWithoutFeedback,
} from "react-native";
import { Rating } from "react-native-ratings";

const MyCollectionPage = () => {
  return (
    <View
      style={{
        flex: 1,
        backgroundColor: "white",
      }}
    >
      <SafeAreaView>
        <Text style={styles.titleTxt}> My Collections </Text>
        <ScrollView style={{ top: 30, height: "100%" }}>
          <TouchableWithoutFeedback>
            <View style={styles.svCont}>
              <Image
                source={require("../assets/dish.jpg")}
                style={styles.dishImg}
              />
              <View>
                <View style={styles.txtCont}>
                  <Text style={styles.dishTitle}>Title</Text>
                </View>

                <Rating imageSize={20} startingValue={3} readonly />
              </View>
            </View>
          </TouchableWithoutFeedback>
        </ScrollView>
      </SafeAreaView>
    </View>
  );
};
```

Onclick to mycollection button will open up the mycollection.js, the name, rating and image of the recipes will be shown.



```
import React from "react";
import {
  Text,
  TextInput,
  View,
  Button,
  TouchableWithoutFeedback,
  SafeAreaView,
} from "react-native";

const FeedbackPage = () => {
  return (
    <View
      style={{
        flex: 1,
        backgroundColor: "#d4a373",
      }}
    >
      <SafeAreaView>
        <Text
          style={{
            position: "absolute",
            fontSize: 20,
            alignSelf: "center",
            top: 70,
            fontWeight: "bold",
            color: "white",
            textShadowColor: "grey",
            textShadowOffset: { width: 0, height: 2 },
            textShadowRadius: 5,
            lineHeight: 30,
          }}
        >
          Any feedback or suggestions will be appreciated. Thank you!
        </Text>
        <View
          style={{
            backgroundColor: "white",
            height: "100%",
            top: "15%",
            borderTopLeftRadius: 70,
            borderTopRightRadius: 70,
            padding: "10%",
            flexDirection: "column",
          }}
        >
          <View>
            <Text
              style={{
                fontSize: 20,
                fontWeight: "bold",
                top: 10,
                paddingBottom: "10%",
              }}
            >
              FEEDBACK FORM
            </Text>
          </View>

          <View style={{ marginVertical: 20 }}>
            <Text
              style={{
                fontSize: 18,
                fontWeight: "bold",
                color: "#c5c5c5",
                paddingBottom: 10,
              }}
            >
              NAME
            </Text>
            <TextInput
              style={{
                height: 20,
                borderBottomWidth: 4,
                borderColor: "#ececec",
              }}
            />
          </View>

          <View style={{ marginVertical: 20 }}>
            <Text
              style={{
                fontSize: 18,
                fontWeight: "bold",
                color: "#c5c5c5",
              }}
            >
              EMAIL
            </Text>
            <TextInput
              style={{
                borderBottomWidth: 4,
                borderColor: "#ececec",
              }}
            />
          </View>

          <View style={{ marginVertical: 20 }}>
            <Text
              style={{
                fontSize: 18,
                fontWeight: "bold",
                color: "#c5c5c5",
              }}
            >
              MESSAGE
            </Text>
            <TextInput
```

Onclick to the Feedback button will open up the feedback.js. Text Input box is used for users to input their name, email and message.

```
<Formik
  validationSchema={loginValidationSchema}
  initialValues={{
    username: "",
    email: "",
    password: "",
    confirmPassword: "",
  }}
  onSubmit={() => navigation.navigate("Explore")}
>
  {({ handleSubmit, isValid }) => (
    <>
      <Field
        component={CustomInput}
        name="email"
        placeholder="Email Address"
        keyboardType="email-address"
      />

      <Field
        component={CustomInput}
        name="password"
        placeholder="Password"
        secureTextEntry
      />

      <TouchableWithoutFeedback
        onPress={handleSubmit}
        disabled={!isValid}
        //onPress={() => console.log("Helo")}
      >
        <View
          style={{
            paddingHorizontal: 20,
            paddingVertical: 10,
            backgroundColor: "#e9edc9",
            width: 120,
            borderRadius: 10,
            alignSelf: "center",
            top: 10,
            shadowColor: "white",
            shadowOpacity: 0.8,
            shadowOffset: { width: 2, height: 2 },
          }}
        >
```

Onclick to the login button will open up the login.js. Formik is used to keep track of all the forms for us and gives us validation out of the box while up is used for form validation.

```
const LoginPage = ({ navigation }) => {
  const loginValidationSchema = yup.object().shape({
    email: yup
      .string()
      .email("Please enter valid email")
      .required("Email is required"),
    password: yup
      .string()
      .matches(/\w*[a-z]\w*/, "Password must have a small letter")
      .matches(/\w*[A-Z]\w*/, "Password must have a capital letter")
      .matches(/\d/, "Password must have a number")
      .matches(
        /[!@#$%^&*()\-_"=+{}; :,<.>]/,
        "Password must have a special character"
      )
      .min(8, ({ min }) => `Password must be at least ${min} characters`)
      .required("Password is required"),
  });
```

A constant loginValidationSchema is imported from Yup where it will do a preliminary check and the front end to see if the basic requirements for email and password are in correct format before the user submits the form. We made both input compulsory to be filled. The input for email must be in the email address format while the password input must be at least 8 characters and include both uppercase and lowercase, numbers and special characters. This is for the security feature.

```
<TouchableWithoutFeedback
  onPress={() => navigation.navigate("Signup")}
>
  <Text
    style={{
      fontSize: 10,
      textDecorationLine: "underline",
      textAlign: "center",
      color: "white",
      top: 20,
    }}
  >
    Do not have an account?
  </Text>
</TouchableWithoutFeedback>
```

Users will be directed to the sign up page when they click on the do not have an account button.

```
const signUpValidationSchema = yup.object().shape({
  username: yup.string().required("Full name is required"),
  email: yup
    .string()
    .email("Please enter valid email")
    .required("Email is required"),
  password: yup
    .string()
    .matches(/\w*[a-z]\w*/, "Password must have a small letter")
    .matches(/\w*[A-Z]\w*/, "Password must have a capital letter")
    .matches(/\d/, "Password must have a number")
    .matches(
      /[!@#$%^&*()\-_"=+{}; :,<.>]/,
      "Password must have a special character"
    )
    .min(8, ({ min }) => `Password must be at least ${min} characters`)
    .required("Password is required"),
  confirmPassword: yup
    .string()
    .oneOf([yup.ref("password")], "Passwords do not match")
    .required("Confirm password is required"),
});
```

Sign up page is similar to the login's, but the validation schema is set to signupvalidationscheme. There are two extra fields where the username is required and the confirmed password field must be field and have the same input as the password input.

```
const CustomInput = (props) => {
  const {
    field: { name, onBlur, onChange, value },
    form: { errors, touched, setFieldTouched },
    ...inputProps
  } = props;

  const hasError = errors[name] && touched[name];

  return (
    <>
      <TextInput
        style={[styles.textInput, hasError && styles.errorInput]}
        value={value}
        onChangeText={(text) => onChange(name)(text)}
        onBlur={() => {
          setFieldTouched(name);
          onBlur(name);
        }}
        {...inputProps}
        placeholderTextColor="white"
      />
      {hasError && <Text style={styles.errorText}>{errors[name]}</Text>}
    </>
  );
};
```

CustomInput specifies the style of the input text box.

5.4.7 Recipe.js

```
const [heartColor, setHeartColor] = useState("white");
```

```
<TouchableWithoutFeedback
  onPress={() => {
    if (heartColor == "white") {
      setHeartColor("red");
      Alert.alert("Added to collection");
    } else {
      setHeartColor("white");
      Alert.alert("Removed to collection");
    }
  }}
>
  <Image
    source={require("../assets/liked.png")}
    style={{
      width: 30,
      height: 30,
      position: "absolute",
      tintColor: heartColor,
      right: 10,
      bottom: 10,
    }}
  />
</TouchableWithoutFeedback>
```

```
<TouchableWithoutFeedback onPress={() => goBack()}>
  <Image
    source={require("../assets/back.png")}
    style={{
      width: 30,
      height: 30,
      position: "absolute",
      tintColor: "white",
      top: 10,
      left: 10,
    }}
  />
</TouchableWithoutFeedback>
```

Whenever the heart is pressed, it will check the tintcolor of the heart and change it accordingly. There is a back button, and it will direct back to the previous page when it is pressed.

```
const [modalVisible, setModalVisible] = useState(false);
```

```
<Modal
  //animationType={"slide"}
  transparent={true}
  visible={modalVisible}
  onRequestClose={() => {
    console.log("Modal has been closed.");
  }}
>
  <View
    style={{
      flex: 1,
      backgroundColor: "white",
      padding: 20,
      paddingTop: 100,
    }}
  >
    <TouchableWithoutFeedback
      onPress={() => {
        setModalVisible(!modalVisible);
      }}
    >
      <View
        style={{
          position: "absolute",
          right: 30,
          top: "10%",
        }}
      >
        <Image
          source={require("../assets/close.png")}
          style={{
            width: 20,
            height: 20,
          }}
        />
      </View>
    </TouchableWithoutFeedback>
```

```
{dataSource.ingre.map((item) => {
  return (
    <View
      style={{ flexDirection: "row", paddingVertical: 10 }}
    >
      <CheckedBox />
      <Text
        style={{
          paddingBottom: 10,
          paddingLeft: 10,
        }}
      >
        {item}
      </Text>
    </View>
  );
})}
<TouchableWithoutFeedback
  onPress={() => {
    Alert.alert("Item are added into shopping list");
    setModalVisible(false);
  }}
>
  <View
    style={{
      backgroundColor: "#4c411a",
      paddingHorizontal: 20,
      paddingVertical: 10,
      //width: 100,
      top: "50%",
    }}
  >
    <Text
      style={{
        fontSize: 20,
        fontWeight: "bold",
        textAlign: "center",
        textTransform: "uppercase",
        color: "white",
      }}
    >
      Add
    </Text>
  </View>
</TouchableWithoutFeedback>
</View>
</Modal>
```

The modal visible for add to shopping list is set to true when it is pressed. The ingre array in datasource is map and each item is rendered with a checkbox component and its ingredients name. The counter works the same as the one on the calories counter page.

The nutrition facts is presented with the flatlist where it will render each of the items in the datasource's nutrition array and show it in a circle container. The item in the array will be split where it will be split with a space. Index 1 of the split item will be shown in bold. The rating used a rating component.

5.5 User Testing

A group survey was carried out to test the usability of the application. The following are the question used based on the System Usability Scale (SUS).

1. I think that I would like to use the system frequently

2. I found the system unnecessarily complex

3. I thought that the system was easy to use

4. I needed guidance to be able to use this system

5. I find the application is well integrated

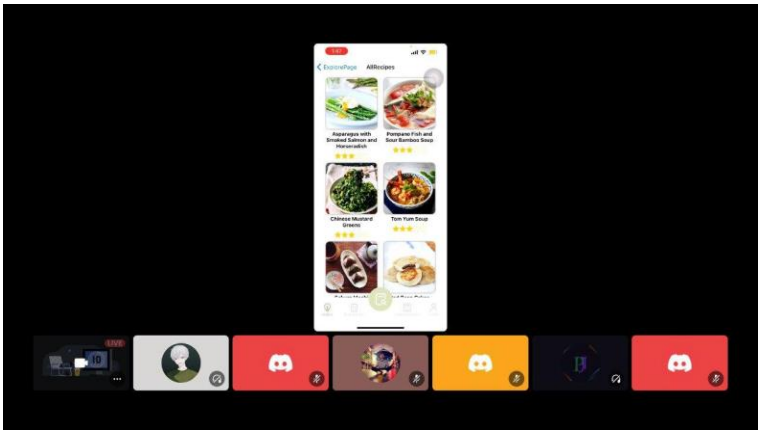
6. I thought that there was too much inconsistency in this system

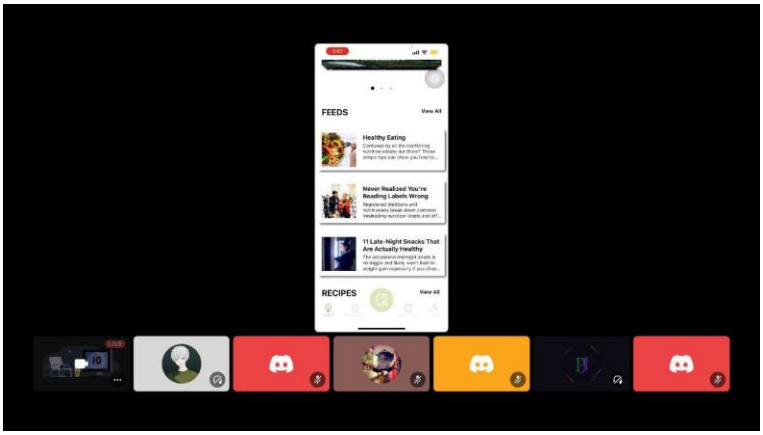
7. I believe most people would learn that to use this system very quickly

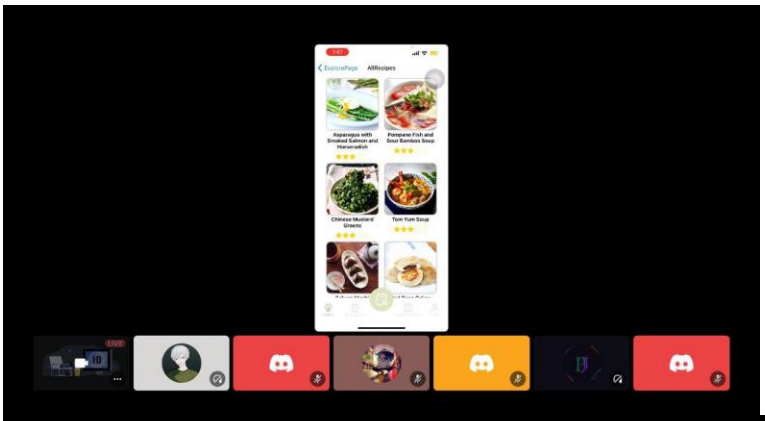
8. It was easy to navigate around the application

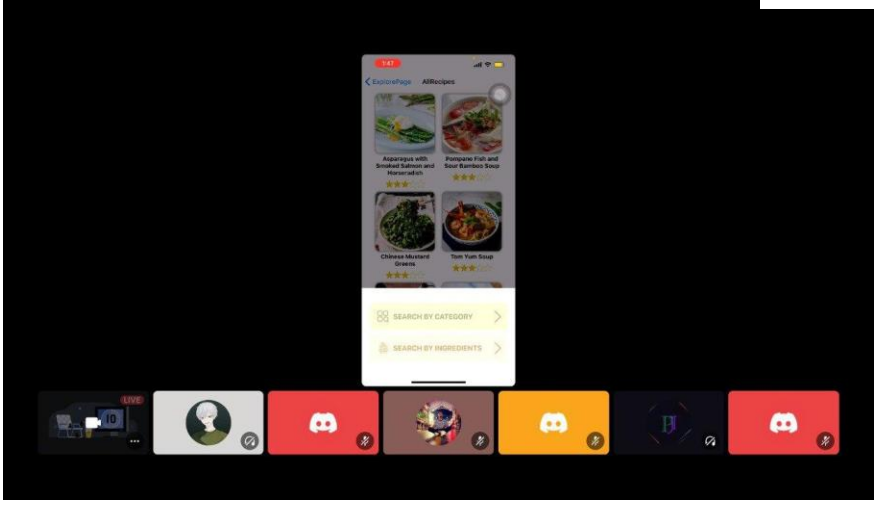
9. I felt very confident using the system

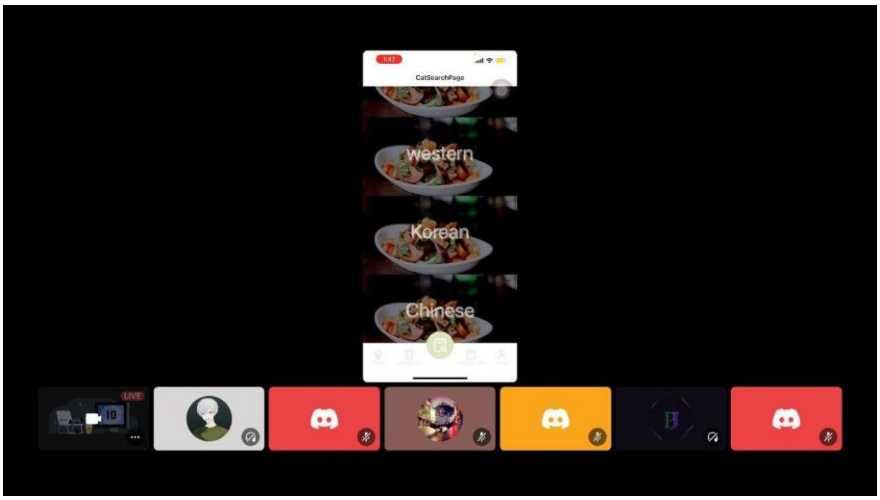
10. There was a lot to learn before I could get going with this system.

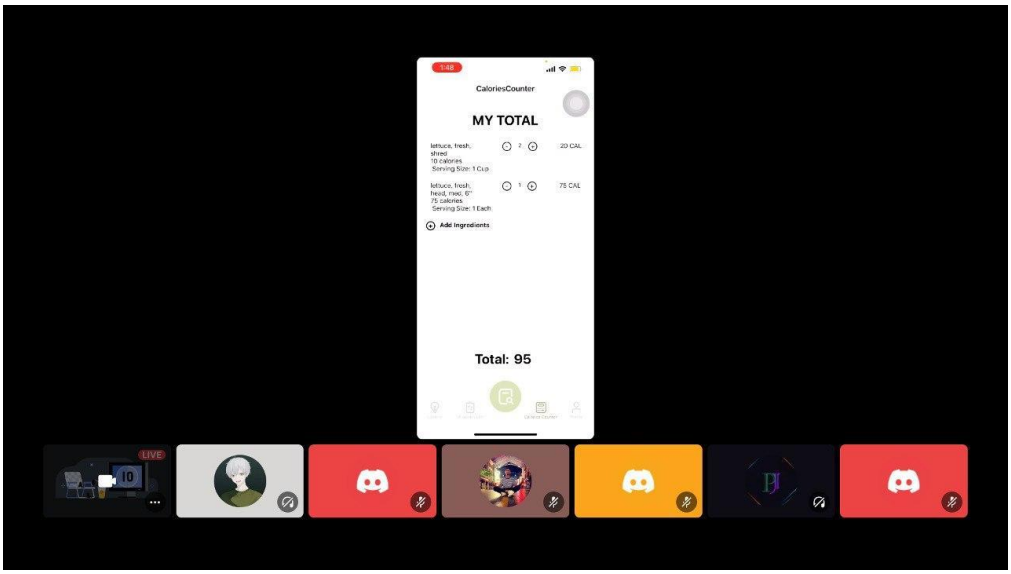

ExplorePage
AllRecipes
Asparagus with Smoked Salmon and Horseradish
Pompano Fish and Sour Bamboo Soup
Chinese Mustard Greens
Tom Yum Soup

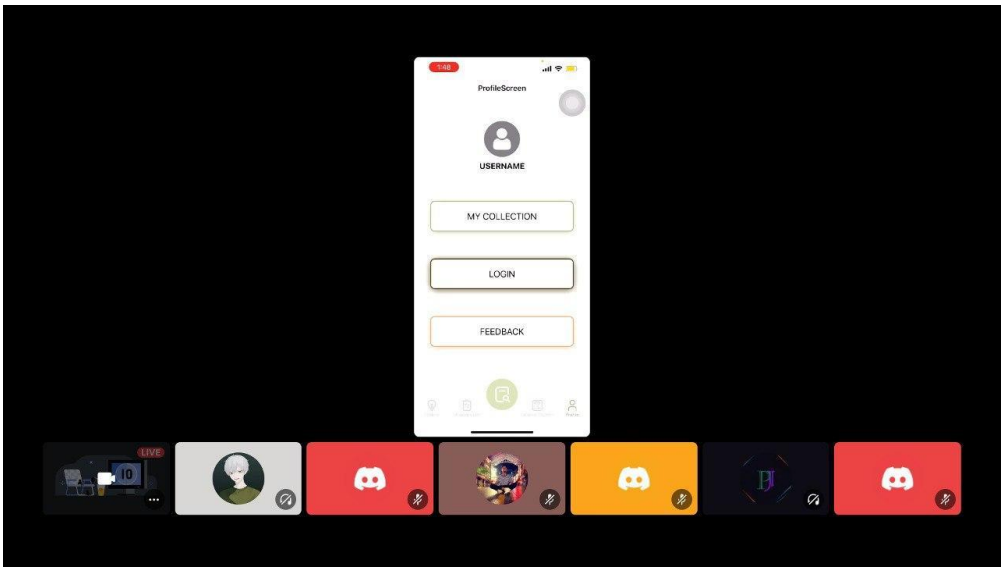

FEEDS
View All
Healthy Eating
Never Realized You're Reading Labels Wrong
11 Late-Night Snacks That Are Actually Healthy
RECIPES
View All

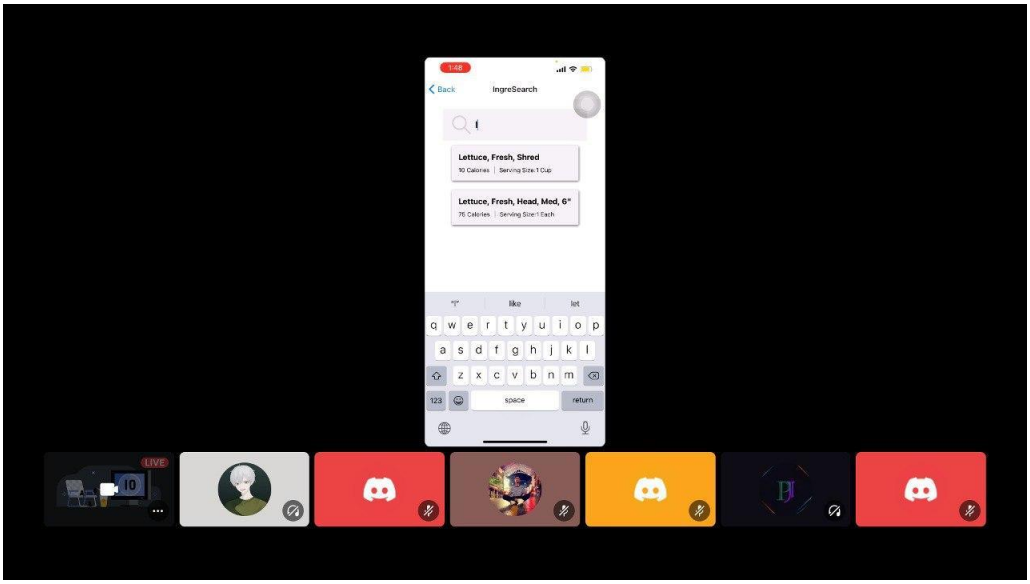


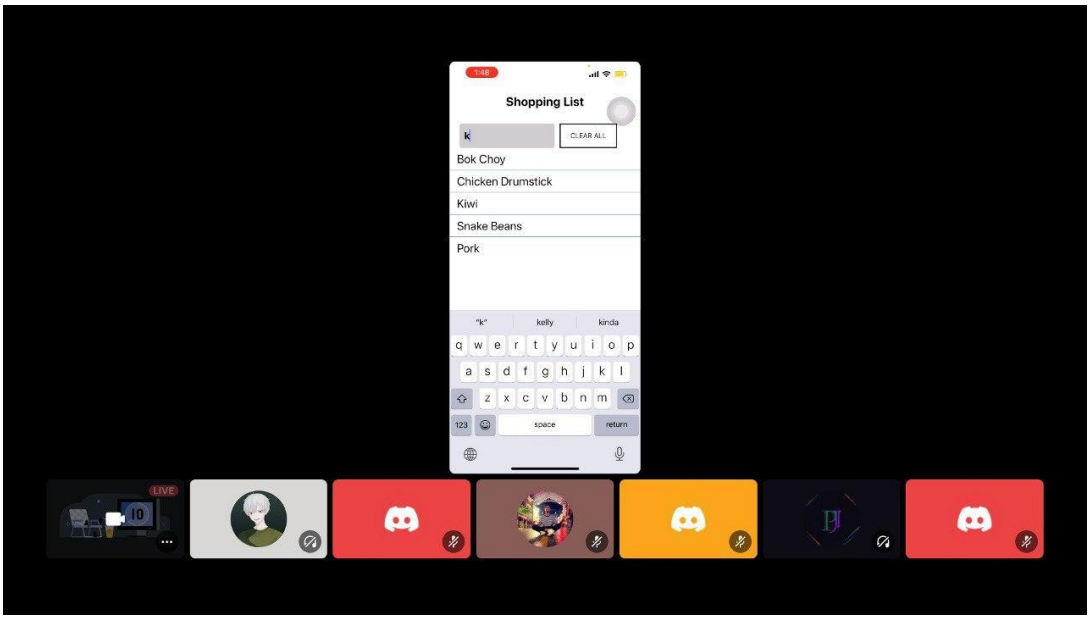

SEARCH BY CATEGORY
SEARCH BY INGREDIENTS

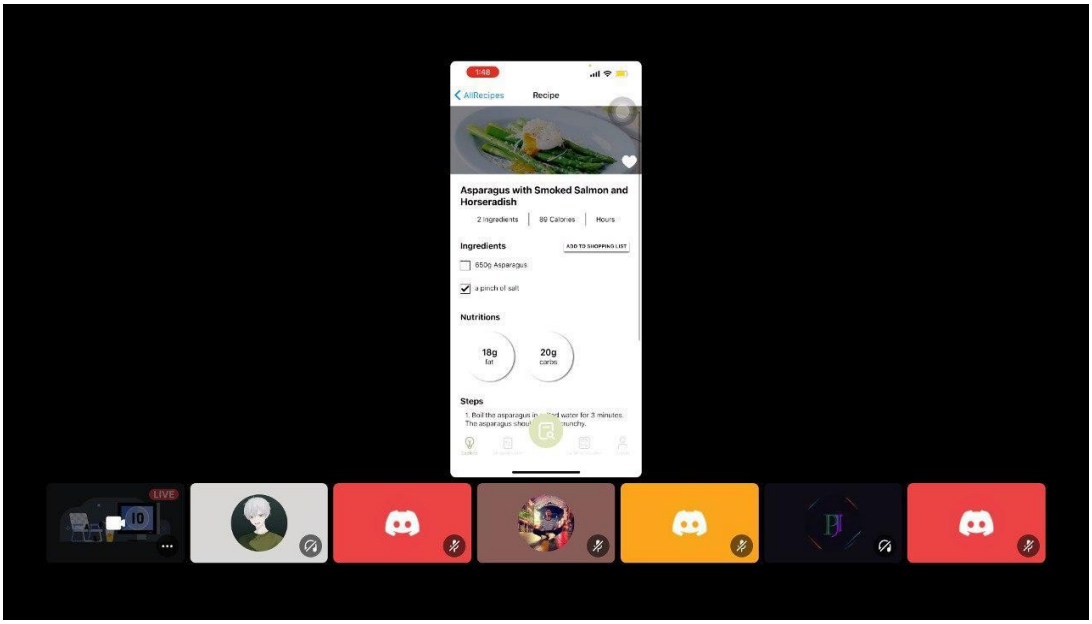

CatSearchPage
western
Korean
Chinese

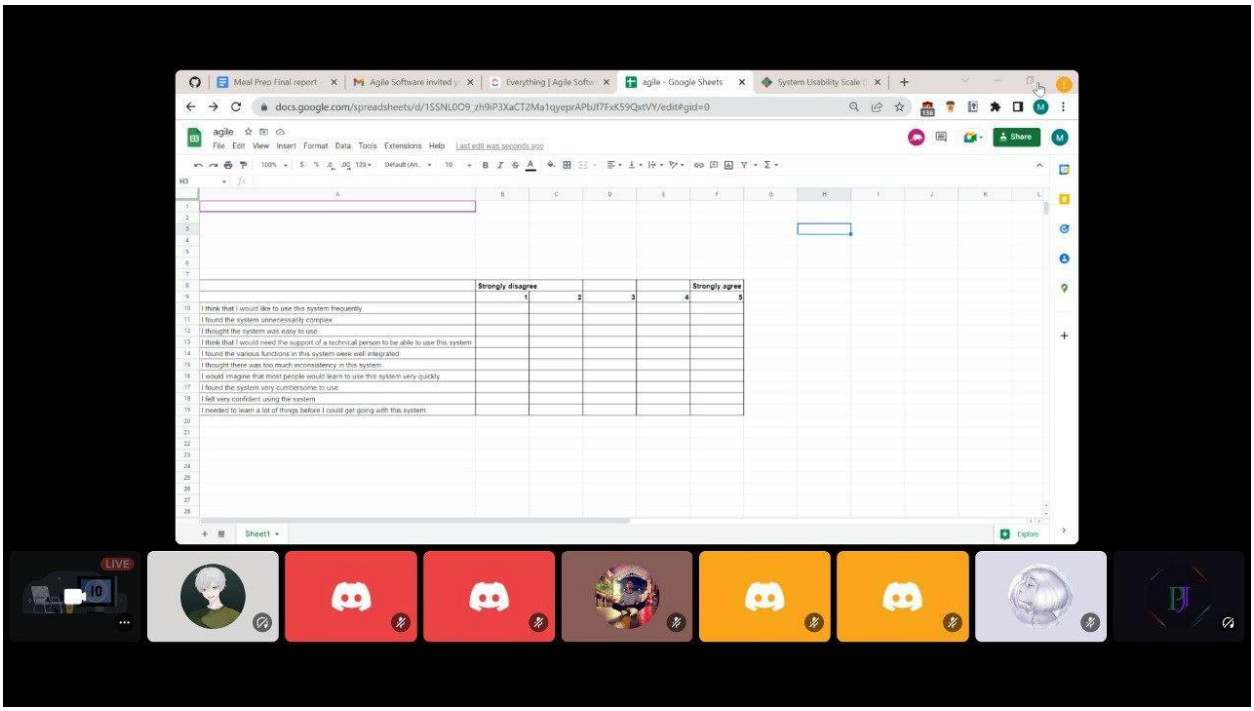

CaloriesCounter
MY TOTAL
Total: 95

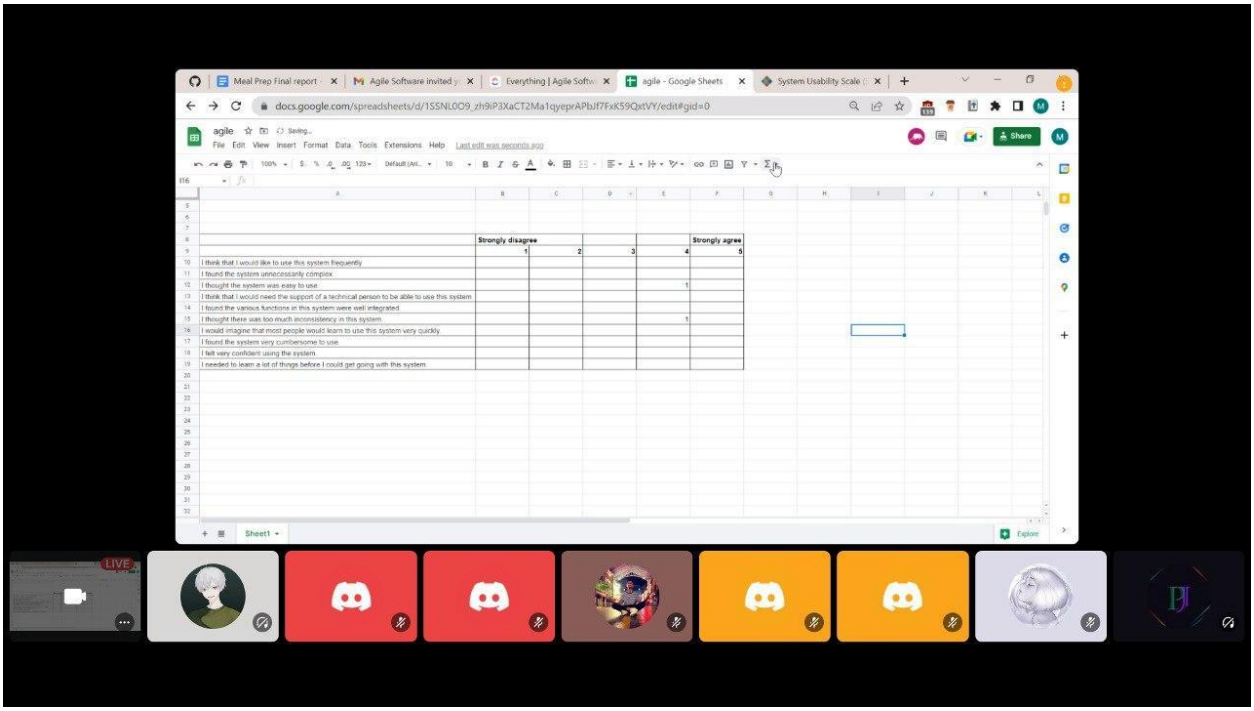

ProfileScreen
USERNAME
MY COLLECTION
LOGIN
FEEDBACK


Back
IngreSearch
Lettuce, Fresh, Shred
Lettuce, Fresh, Head, Med, 6"


Shopping List
CLEAR ALL
Bok Choy
Chicken Drumstick
Kiwi
Snake Beans
Pork


AllRecipes
Recipe
Asparagus with Smoked Salmon and Horseradish
2 Ingredients
80 Calories
Hours
Ingredients
ADD TO SHOPPING LIST
650g Asparagus
a pinch of salt
Nutritions
18g fat
20g carbs
Steps


docs.google.com/spreadsheets/d/1SSNL0O9_zh9iP3XaCT2Ma1qyeprAPbJf7FvK59QxtVY/edit#gid=0
agile
File Edit View Insert Format Data Tools Extensions Help
Share
Strongly disagree
Strongly agree
1
2
3
4
5
I think that I would like to use this system frequently
I found the system unnecessarily complex
I thought the system was easy to use
I think that I would need the support of a technical person to be able to use this system
I found the various functions in this system were well integrated
I thought there was too much inconsistency in this system
I would imagine that most people would learn to use this system very quickly
I found the system very cumbersome to use
I felt very confident using the system
I needed to learn a lot of things before I could get going with this system
Sheet1



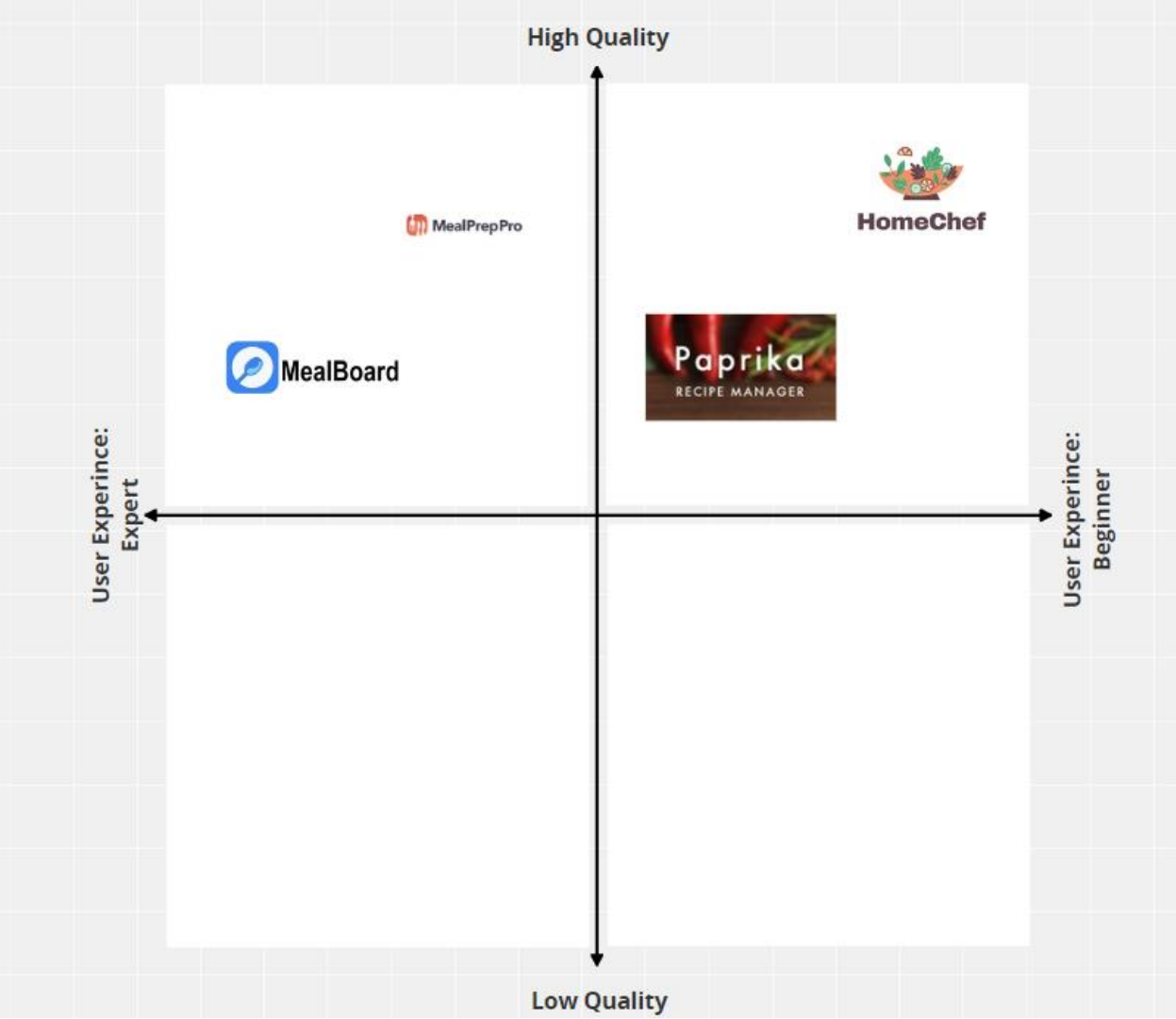
The feedback we received:

1. To allow users to export the shopping list
2. To allow users to add in their own recipes
3. Enable users to save calorie counter calculator for future use

6 Analysis

6.1 Market positioning map

For the analysis of HomeChef, we begin by identifying the applications with similar functions available and comparing them against our application. The applications we have chosen are MealPrepPro, Paprikaapp, and MealBoard.



We have used two parameters to analyse and evaluate the application. The two parameters are Quality and User Experience. Quality refers to the collection of recipes and features available for users. User Experience refers to the experience level of users in the kitchen.

HomeChef is ideal as the application has a wide range of recipes from different regions that cater to different cooking styles. Our application also caters to users with all levels of experience in the kitchen.

6.2 Comparison against similar applications in the market:

6.2.1 MealPrepPro

Application to plan out meals based on diet

Limitations of MealPrepPro:

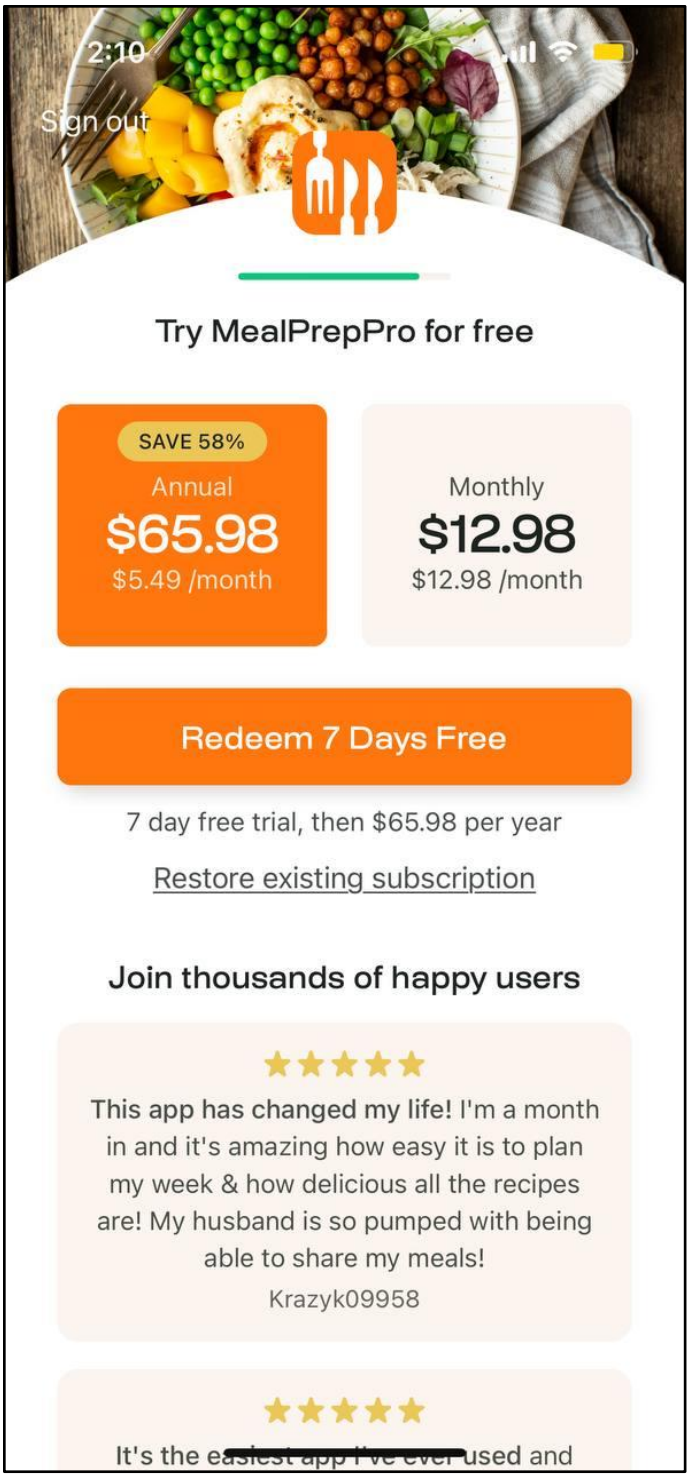
1) Monetisation of the application

As mentioned previously in the mid-term report, HomeChef, focussed solely on catering to the users hence there we would not be monetising the application. Most of our primary competitors focus on a membership plan to fully optimise the application. An example of this would be MealPrepPro, which targets the same category of users. This application provides similar functions to HomeChef, however, a user has to subscribe to MealPrepPro.

When a user downloads the application, they are prompted to sign up to be a member, following which they would have to key in the bank details to get the 7-day free trial which allows them to experience all the functions of the application. Following this, they would have to enrol on one of the limited prices plans the application provides, in order to have full access to the application.

Whereas HomeChef offers the same functions with no charges. The users are given complete access to all the features of the application. This allows them to experience and plan their needed without worries about having to make a payment or missing out on aspects of the application.

Furthermore, the application is more catered towards batch cooking. Where the user cooks larger quantities of food to be stored and consumed later.


2:10
Sign out
Try MealPrepPro for free
SAVE 58%
Annual
$65.98
$5.49 /month
Monthly
$12.98
$12.98 /month
Redeem 7 Days Free
7 day free trial, then $65.98 per year
Restore existing subscription
Join thousands of happy users
This app has changed my life! I'm a month in and it's amazing how easy it is to plan my week & how delicious all the recipes are! My husband is so pumped with being able to share my meals!
Krazyk09958
It's the easiest app I've ever used and

2) Limited to IOS users

The application is only available to iOS as of now, which restricts the majority of the user from not being able to use it. According to global statistics, as of July 2022, 71.38% of the global population uses android while 27.5% of the population uses iOS.

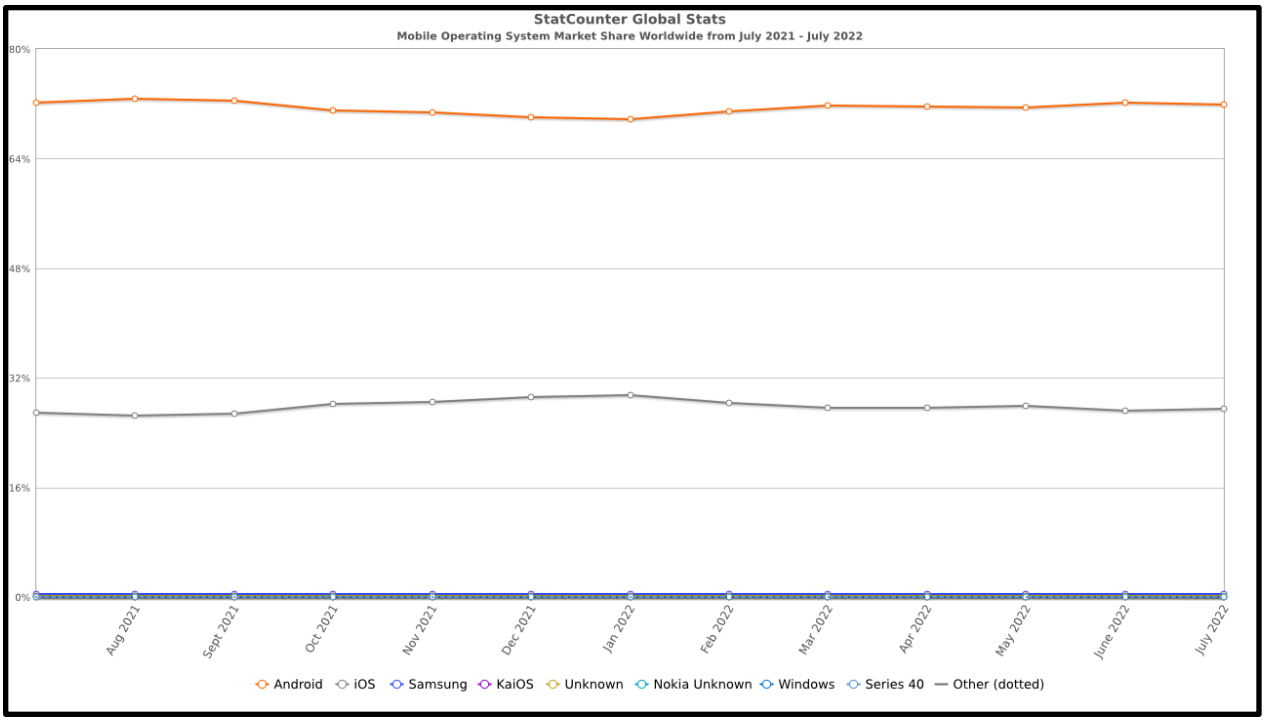
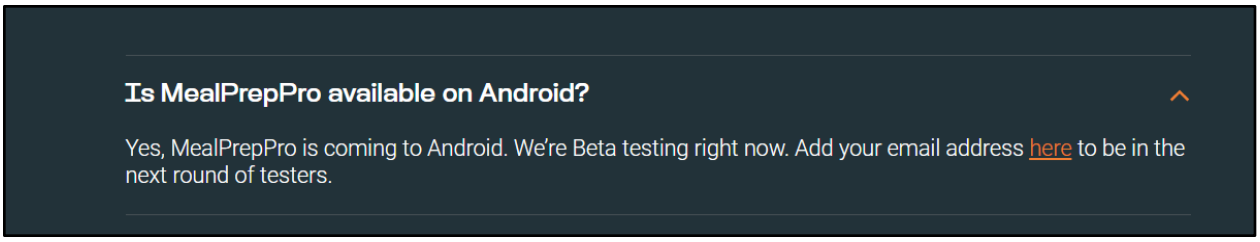
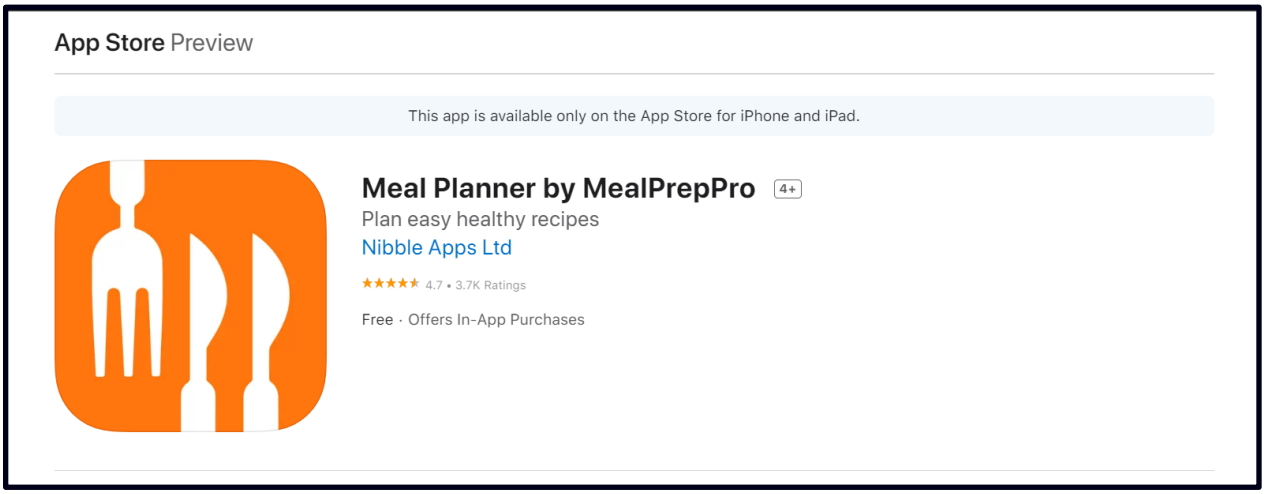


Figure 1

This drastically reduces the size of the target audience and makes it difficult for a user to locate an application which caters to their needs in android.





We took into account this based on user feedback and decide to make HomeChef, available on both android and iOS devices. The application will be fully functioning in both systems.

6.2.2 Paprika:

Application that aids in organising recipes, planning out meal schedules and making shopping lists.

Limitations of Paprika:

1) No recipes instructions video

The application only focuses on aiding users to structure their meal plans and does not provide any help on how to make the meals. This defeats the purpose of organising recipes since most of the users are new home cooks or users looking for a quick recipe. With reference to this, we can

conclude that users would require help in planning their meals and at the same time need aid in making the recipes.

HomeChef centres around serving the user's needs. The application was created to cater for users; new home cooks and users who are in a crunch for time and more. Through different user stories, we ensured to include step-by-step instructions on how the users can cook the recipes. This saves additional time as the users do not have to toggle between applications to find the instructions for the recipes, they saved for the meal plan.

2) Restricted availability

Most competitors focus on a membership plan in order to fully optimise the application. Paprika does the same. The application requires the user to pay in order to download the application. This prevents the user from experiencing the application.

Furthermore, the application is only available to iOS users. As mentioned previously this largely constrains the target audience who are primarily Android users as well.

Even though Paparika provides better user experience and has high quality with reference to the market analysis map above. This is futile if the application is inaccessible to most users. As statistics show that most users are more willing to download an application if it free on both Andriod and iOS devices(Satista,2022). In addition to this there is a larger user market for Andriod compared to iOS devices owners as well.

Hence by taking into consideration the points mentioned above, HomeChef has been developed to be a free application which will be available to both Andriod and iOS users.

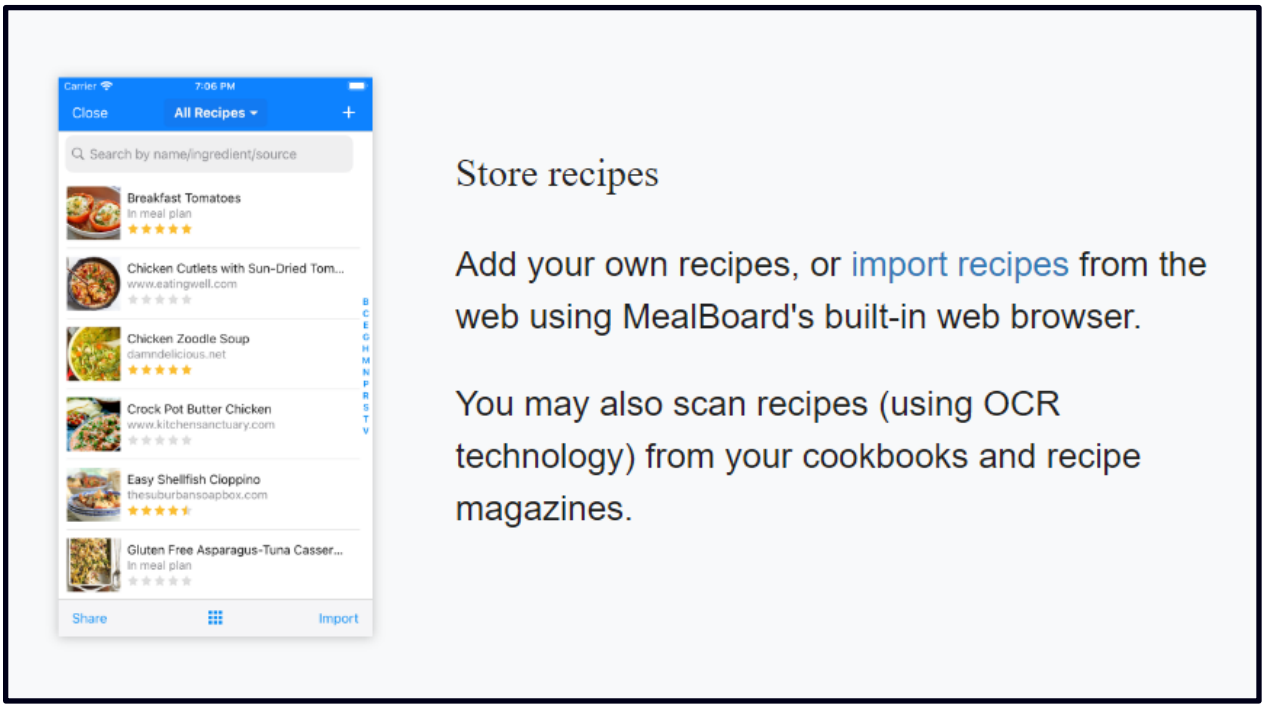

2:12
paprika
Cancel
Paprika Recipe Manage...
Organize your recipes
S$ 6.98
67
Sync with all your devices
Save your favorite recipes
Create grocery lists
Recipe Keeper
Meal planner & shopping list
GET
19
In-App Purchases
Today
Games
Apps
Arcade
Search

6.2.3 MealBoard

Application that helps in meal planning, organising shopping lists and recipes. It allows tracking inventory.

Limitations of Meal Board:

MealBoard is an application that is primarily for arranging and planning meals to aid the user in preparing for their day. However, the application does not have a database of recipes or ingredients. This is disadvantageous as the application relies heavily on user input. The user has to upload their recipes or import recipes from the web using the built-in extension by MealBoard. If the user fails to input this data, the application would be deemed ineffective. As the application aims to be a meal planning application and using the data input for the meal planning (the recipes), the other features would be functional. An example would be creating a shopping list, which a user would usually create using ingredients they do not have to create recipes. Hence, the lack of recipes is a disadvantage to many users.



Furthermore, the application does not provide the nutritional value of the meals. Nutrition is an essential part of meal preparation, especially in this increasingly health-conscious society. Consumers are taking into account the type and quantity of nutrition and calories in the meals they consume on a daily basis. Hence the lack of nutritional information regarding meals and recipes is disadvantageous to the consumer and creator.

In HomeChef, there is an extensive recipe database, from different categories and cuisines. This enables the user to locate the recipes they are looking for and also provides them with endless options of recipes to choose from. Adding on, all the recipes and ingredients in the application are supported by the respective nutritional value and calorie count per serving. This aids the user in planning their meals and being health conscious.

MealBoard is only available to iOS users. This narrows the target audience and prevents a large portion of the users from accessing it. As mentioned previously, the majority of the consumers are android users hence this is disadvantageous to the creators as well.

HomeChef is made to be accessible to both iOS and android users

7 Evaluation

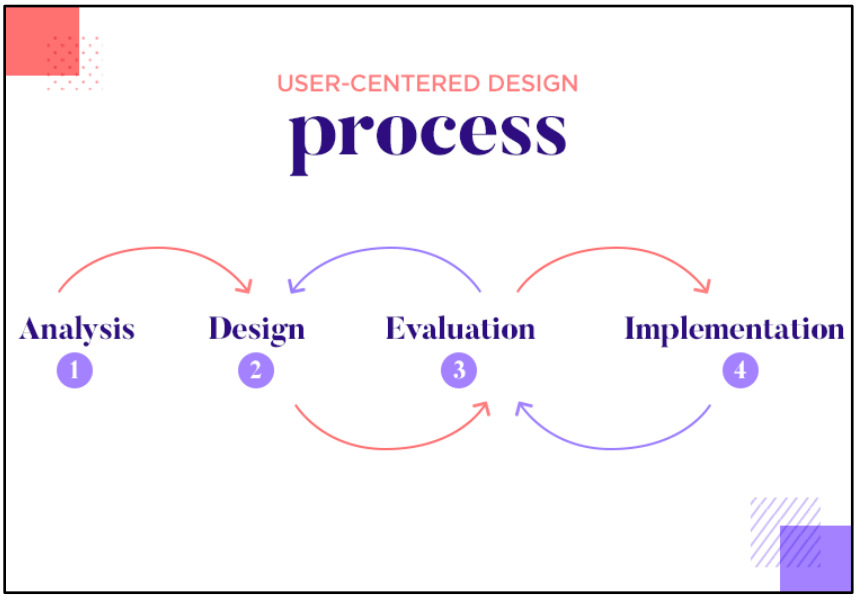
7.1 Evaluation of the group project

For the entire process of working on the project, the team was generally split into two teams. The application team was primarily working on the application itself. The report team was focusing on working on the report while also assisting the application team with coding. Such support was to gather recipes for the application and also work on the designs and the structure of the application.

By choosing agile, the team was able to produce the deliverables in successful incrementation. This also allowed for the team to respond to feedback quickly to understand key errors and issues with the application throughout the development. The team was also able to evaluate the

progress and feedback and use the results to further plan and rectify any errors and make consistent progress throughout.

By choosing User Centric Design (UCD), the team primarily focused on designing the application based on the general user's needs. The team primarily focused on adopting an iterative development and frequent testing to make sure that the application is always user friendly. The chart from Justinmind visually explains the general process that went into the UCD methodology.



1. To analyse the problem statement raised and spot key issues to tackle in our product to aid the users.
2. Designing the product based on the user needs
3. Evaluation of the product and reverting back to the design to improve
4. Implementation of the product while evaluating the process

This creates a cycle of procedures to ensure that our design and implementation addresses the user's needs.

7.2 Improvements to HomeChef

An improvement that can be done to the application is to create a social interaction aspect to it. Currently the application is very static when it comes to user interactions with other users. With an addition of a social media component to it, HomeChef will be able to build an online community for users to meet like-minded chefs and discuss recipes and get feedback. With an addition of such features, the users can share photos and videos of their cooking process and also give feedback on the recipes. This will make the application more interactive and engaging. This will create a potential to pull in more users for the application as they will be drawn in from the recommendations from their family and friends.

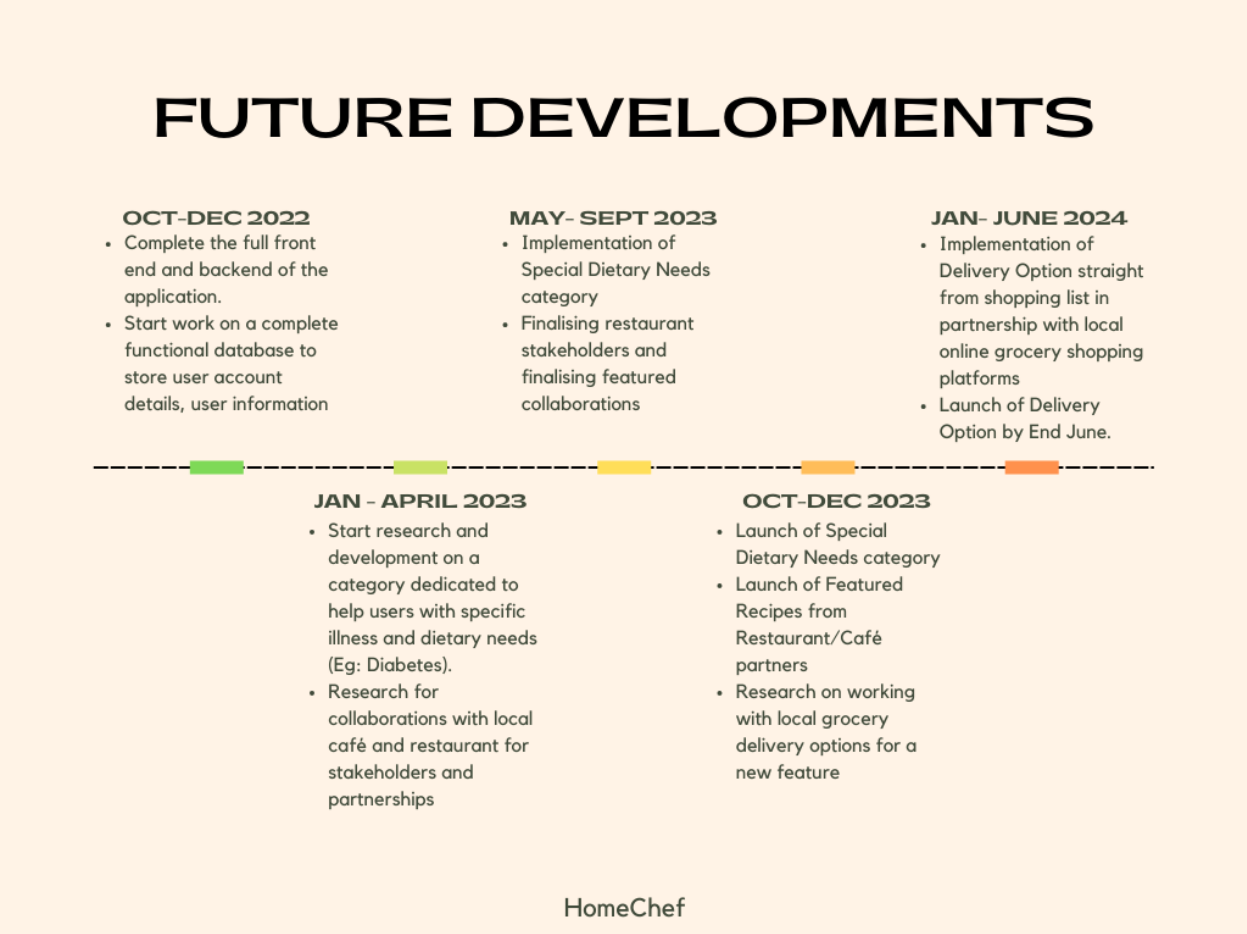
Another improvement that will not only add value but also increase an environmental awareness to the users will be to have a food wastage tracker. Food wastage is a common problem as addressed in the introduction. We can have a food wastage section for users to where users can analyse the groceries bought, their expected expiry date and if the users have utilised the grocery by that date. In this way, the users can get an understanding of their food usage and wastage habits.

7.3 Limitation of HomeChef

The technical features left to be implemented are as follow:

1. Adjust the ingredient list according to the serving size
2. Avoid duplicate to be added for both shopping list and calories counter
3. Match selected ingredients to its related recipes
4. Add to collection function
5. Authentication for login and sign up (Firebase)

7.4 Timeline for Future Developments



7.5 SWOT Analysis

7.5.1 Strength

HomeChef is an easy to use, functional application with more diversity in recipes and health tracking features compared to other products currently in the market. By giving users all the various options packaged into a single application, users have a complete experience.

HomeChef caters to a wider range of users when it comes to recipes. It covers both regional and international cuisines. This not only allows users of different regions to use the app, it also allows local users to try out different recipes from around the world. This makes the application have a more international appeal.

7.5.2 Weakness

Currently the application is not fully developed with a database. This hence makes all the features static and features like users being able to create their own account are all in prototyping stage and not fully functional.

A weakness of the application is the social component of it. The app does not let users communicate and connect with other users of the application. Right now, the application is only designed for a certain user and his/her needs. It does not cater to the users' need to interact with other users for collaborations and support.

7.5.3 Opportunities

As stated in the diagram above under Timeline for Future Developments, the team is aiming to work on three major developments.

1. Creating a Special Dietary Needs category for users with illnesses that requires very specific dietary needs. Under this section the recipes featured will be moderated to help them with their dietary needs and dietary planning.

2. Featuring Local restaurants and cafes. By forming partnerships with local businesses, HomeChef will be able to get stakeholders which will bring in revenue for the application as we feature these eateries on our application. This collaboration can include special recipes and videos from these restaurants.

3. Adding a Delivery Option in the shopping cart. By working with a local grocery delivery platform, users will be able to not only add their needed ingredients in the shopping cart, they can order their groceries from the application and get it delivered to their homes.

7.5.4 Threats

Meal preparation and health tracker applications market is on the rise with many applications being introduced. This increases the opportunity for users to pick from a wide range hence creating more competitors for HomeChef.

Currently, HomeChef is a free to use application. Hence, there is currently no sustainable source of income for the application to be maintained and updated. With a lack of revenue coming in, the application might not be able to have funds for further developments and even the proper functioning of the application itself.

8 Conclusion

In conclusion, the HomeChef application is a good support system for amatuer home cooks to aid them in the kitchen. The product is not just a catalogue for recipes. HomeChef is equipped with relevant features like calorie counter and shopping list to aid users in both pre and post cooking. With the ideology, **"To eat is a necessity, but to eat intelligently is an art.".** We strongly believe that HomeChef will allow users to learn the art of cooking while having a positive mindset and approach towards healthy and sustainable eating habits.

9 References

Tamarkin, D. (2017). *How and Why Home Cooking is Dying in America*. [online] Epicurious. Available at: https://www.epicurious.com/expert-advice/what-i-mean-when-i-say-home-cooking-is-dying-article [Accessed 3 Aug. 2022].

Kliff, S. (2014). *The problem with home-cooked meals*. [online] Vox. Available at: https://www.vox.com/2014/9/26/6849169/the-problem-with-home-cooked-meals [Accessed 3 Aug. 2022].

ThinkEatSave. (2013). *Worldwide food waste*. [online] Available at: https://www.unep.org/thinkeatsave/get-informed/worldwide-food-waste#:~:text=Roughly%20one%2Dthird%20of%20the,tonnes%20%2D%20gets%20lost%20or%20wasted. [Accessed 3 Aug. 2022].

Toogoodtogo.com. (2016). *What food is wasted?* [online] Available at: https://toogoodtogo.com/en-us/movement/knowledge/what-food-is-wasted [Accessed 3 Aug. 2022].

Sarda, B., Delamaire, C., Serry, A.-J. and Ducrot, P. (2022). Changes in home cooking and culinary practices among the French population during the COVID-19 lockdown. *Appetite*, [online] 168, p.105743. doi:10.1016/j.appet.2021.105743.

Justinmind.com. (2020). *User-centered design: a beginner's guide*. [online] Available at: https://www.justinmind.com/blog/user-centered-design/ [Accessed 1 Sep. 2022].
Wrike.com. (2022). *What Is a Sprint in Agile?* [online] Available at: https://www.wrike.com/project-management-guide/faq/what-is-a-sprint-in-agile/ [Accessed 1 Sep. 2022].

Statista. (2022). *Android & iOS free and paid apps share 2022 | Statista*. [online] Available at: https://www.statista.com/statistics/263797/number-of-applications-for-mobile-phones/ [Accessed 4 Sep. 2022].

10 Appendix

Appendix 10.1 Libraries and Dependencies used

Libraries/ Dependencies Used	**Purpose**
React navigation/ Stack and Tab Navigation	Navigation of the application
React Native Ratings	Used for the rating display
React Native Snap Carousel	Used for the carousel display
Formik	Styling the form
Yup	Validation of the form field
Firebase	A realtime dynamic database that would help to store the data

Appendix 10.2: Components used

Components in the folder	**Function**
CalListItem	Cal List item is the layout for listing the item at the calories counter page. It consists of ingredients details, a counter and the total count for the selected ingredients
Card	Card is the layout card we used to display the item on the recipes page, It consists of the dish image, the title and the rating Found in AllRecipes
CarouselCard	CarouselCard is the navigation of all the carousel card item
CarouselCardItem	CarouselCardItem is the layout of the carousel used on the explore page. It consists of the image and title.
Category	The layout of the ingredient's category.
CheckBox	CheckBox is a checked box for checking on the ingredients list in the shopping list and recipe page

CustomInput	Custom input is the layout of the text input for login and sign up form
GoBack	GoBack is a button to navigate back to the previous screen
IngreListItem	IngreListItem is the layout of the ingredient listed at the search by ingredients page. It consists of a select button and the ingredient name.
LongCard	Card is the layout card we used to display the item in the recipes page, It consist of the dish image, the title and the rating
SearchBtn	SearchBtn is the layout of a search bar that is used to search for respectives data
TabModal	TabModal is a pop-out option for search tab
TabNavigation	Tab Navigation is a navigation bar that appears in every screen for users to access Explore, Shopping List, Calories Counter, Profile and the Tab Mobal for search.

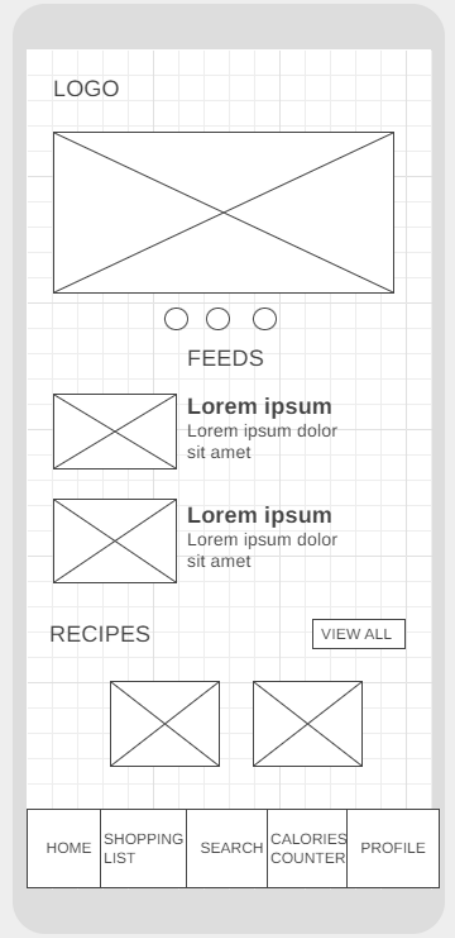
Appendix 10.3: Purpose of Screens

Screen name	Purpose

AddIngreToshop.js	For the user to search up and add the ingredients to the shopping list
AllArticles.js	Show all the Articles in the database
AllRecipes.js	Show all the Recipes in the database
Article.js	The content of selected Articles
CaloriesCounter.js	Display all the selected ingredient for calories count
CaloriesIngreSearch.js	For the user to search up and add the ingredients to the calories counter
CatSearchResult.js	Display the recipes for selected category
Explore.js	For the user to access the articles and recipes
Feedback.js	For the user to send feedback
Login.js	For the user to login to their account
MyCollections.js	Show all the recipes that user had saved
Profile.js	For the user to access the collection, feedback and login page

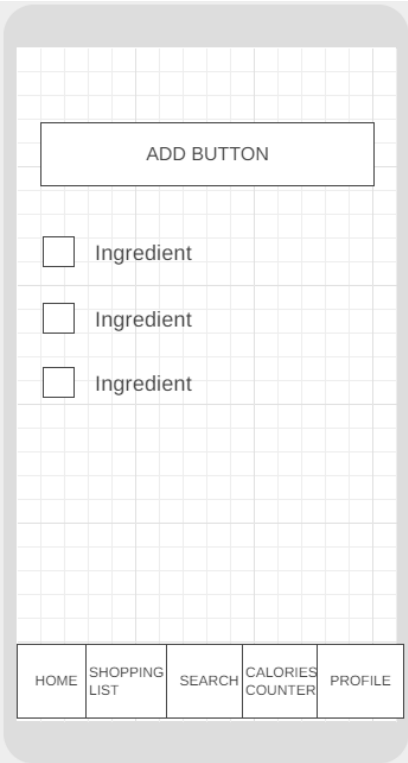
Recipes.js	The content of selected Recipes
SearchByCat.js	For a user to choose the category of the recipes they wished to search up on
SearchByIngre.js	For the user to view and add the ingredients for matching of recipes
SearchResults.js	For viewing the matching related recipes
ShoppingList.js	Display the selected ingredients that are added to the shopping list
SignUp.js	For the user to register for an account

Appendix 10.4: Low fidelity



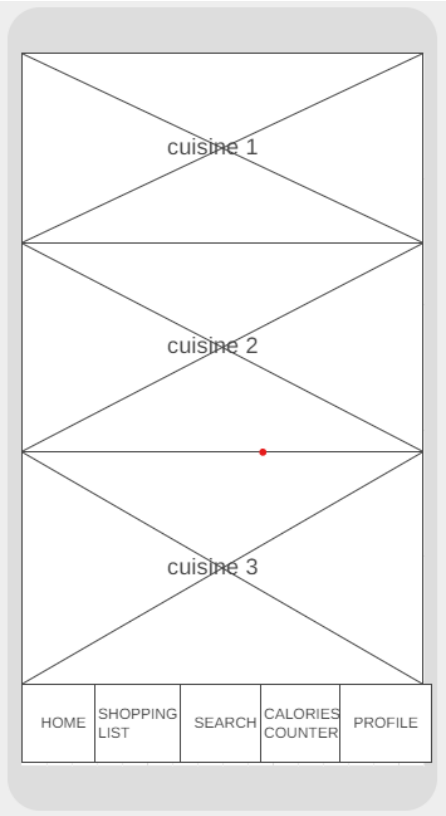
This is the homepage which consists of

- Navigation bar at the bottom which upon clicking would lead to different pages
    - Direct to Homepage
    - Direct to Shopping list
    - Direct to Search page, a pop out buttons will allow users to choose if they want to search by recipes based on categories or ingredients selected
    - Direct to Calories counter
    - Direct to user Profile page
- A slideshow of recently added recipes and most searched recipes
- Food articles
- A preview of recipes
- “View all” button which leads to all the recipes available



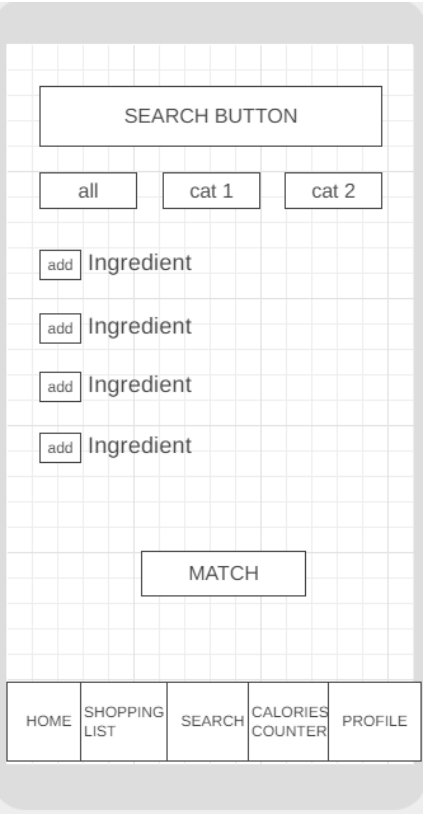
This is the shopping list page which consists of

- An “Add” item button to look for ingredients to be added to the shopping list
- A check box to tick off the items that have been brought
- Navigation bar at the bottom which upon clicking would lead to different pages



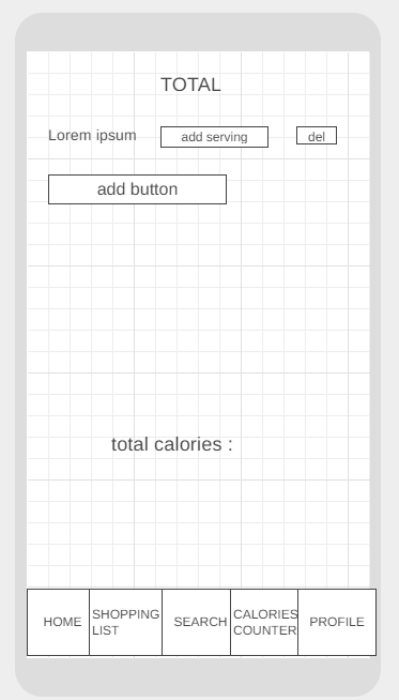
This is the recipe search page (by category) that consists of

- A list of different cuisine tabs
- Navigation bar at the bottom which upon clicking would lead to different pages

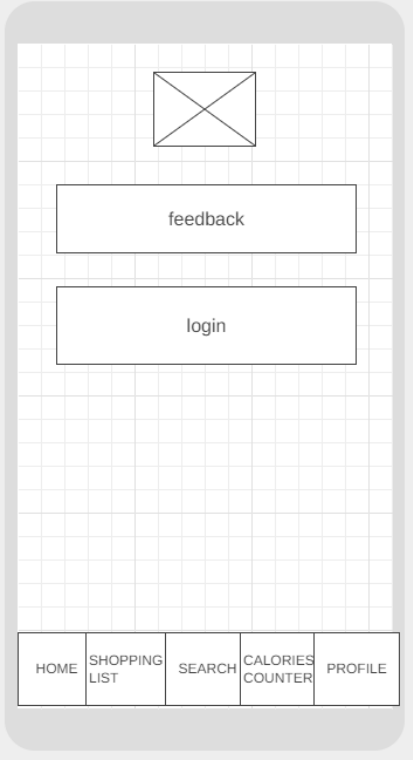


This is the ingredient search page (by ingredients) that consists of

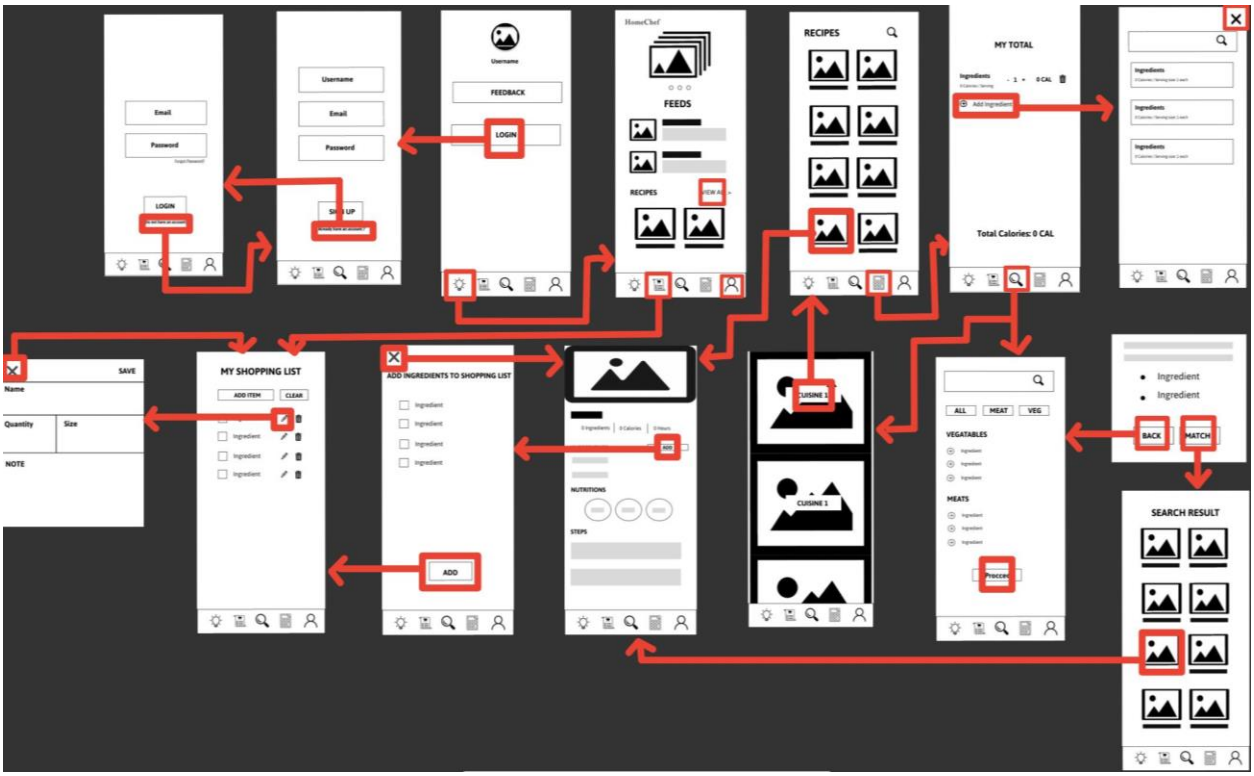
- A search bar to search for ingredients
- Different categories of ingredients tab
- A list of ingredients
- “Match up” button to correlate recipes based on the ingredients the user selected
- “Add” button for each ingredient to be added to their list
- Navigation bar at the bottom which upon clicking would lead to different pages

	This is the recipe page which consists of - The details of the recipes such as ingredients, nutritions and steps - “Add” button to add the list of ingredients from the recipe to the shopping list - Navigation bar at the bottom which upon clicking would lead to different pages
	This is the calories counter page which consists of - “Add” button to add the ingredients - “Add Serving” button to increase the serving size of the ingredients - “Del” button to delete the ingredients added - A calculation of the total calorie count for all the ingredients added in - Navigation bar at the bottom which

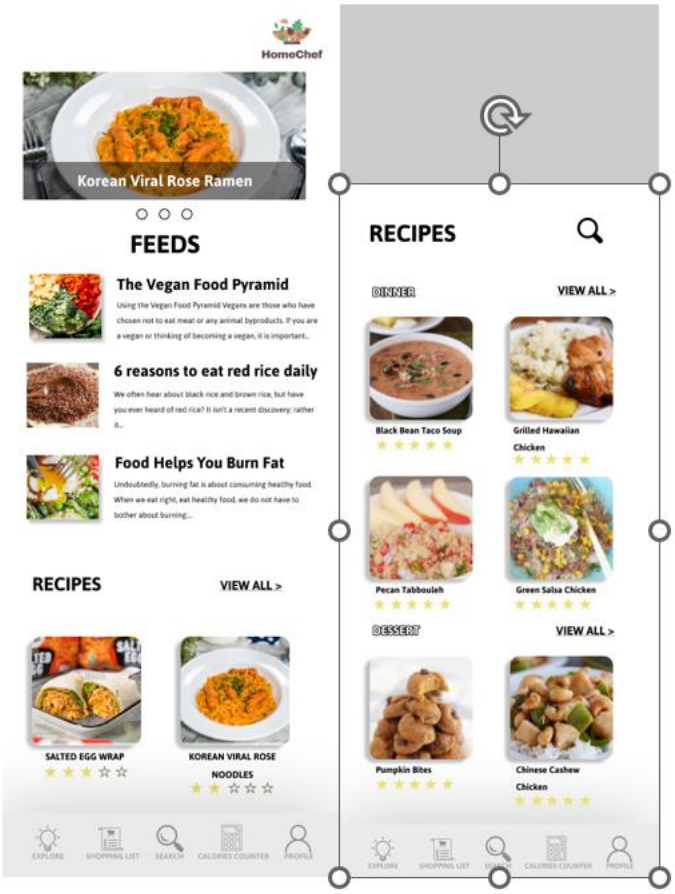
	upon clicking would lead to different pages

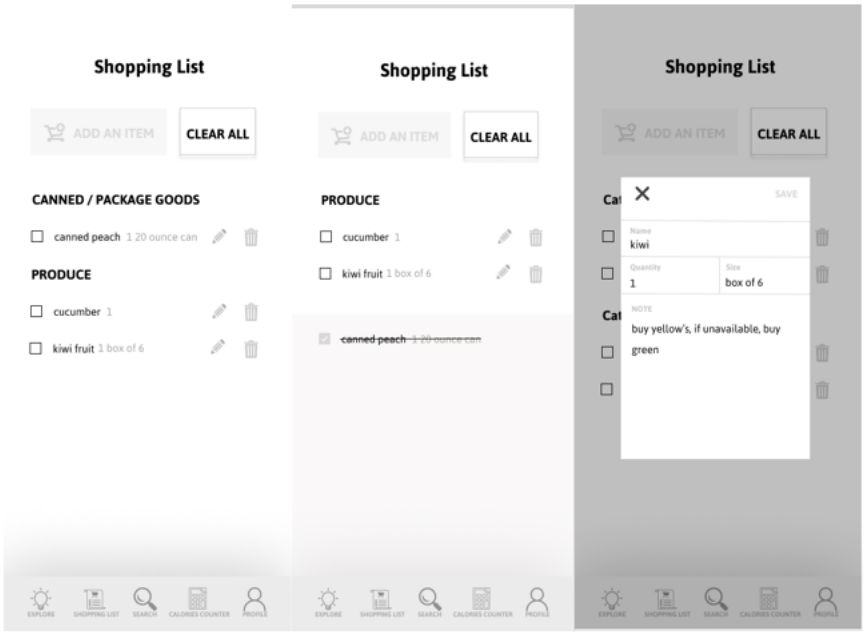
	This is the profile page which consists of - “Feedback” button for the users to provide feedback - “Login” button for the users to log in or logout of their account - Navigation bar at the bottom which upon clicking would lead to different pages

Appendix 10.5: Medium fidelity

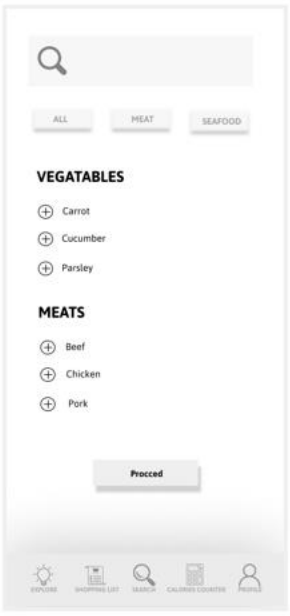


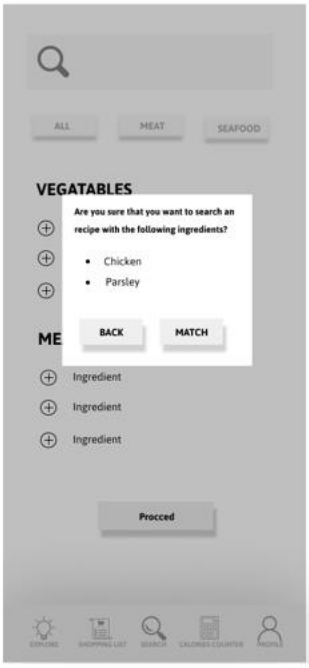
Appendix 10.6: High fidelity

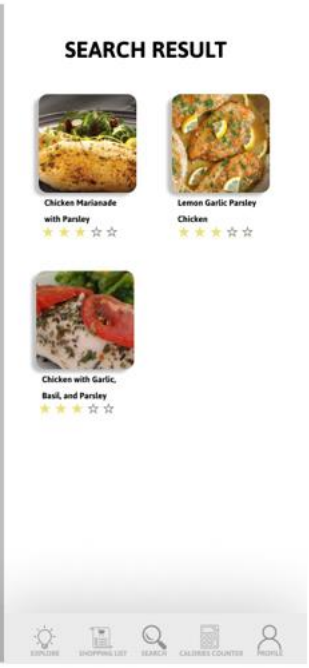


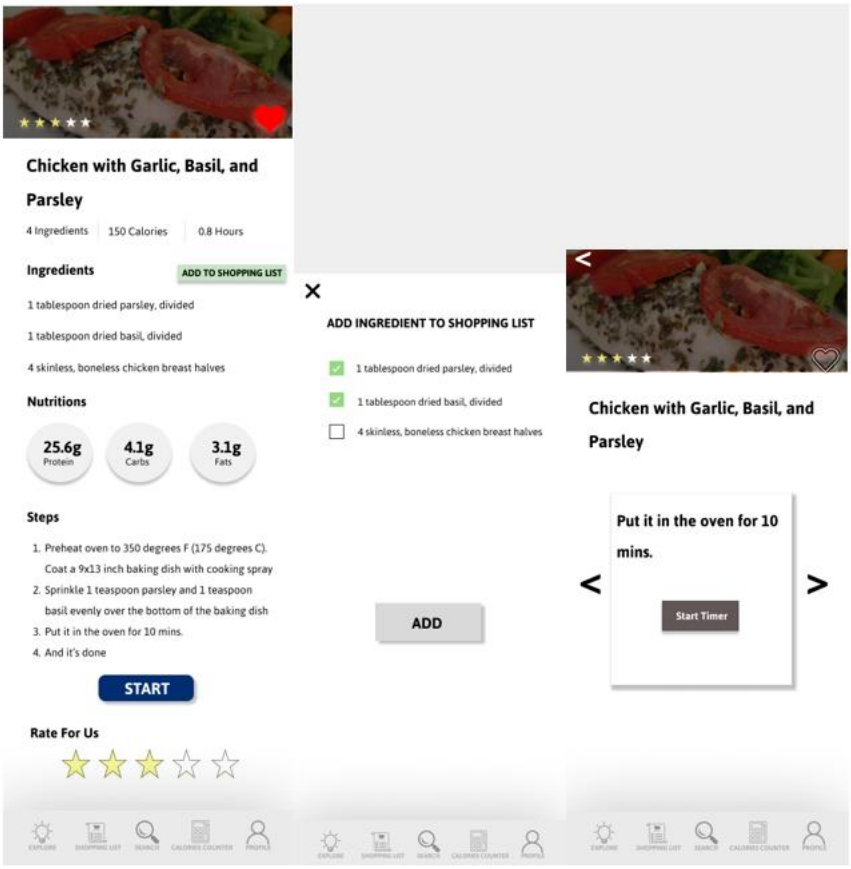


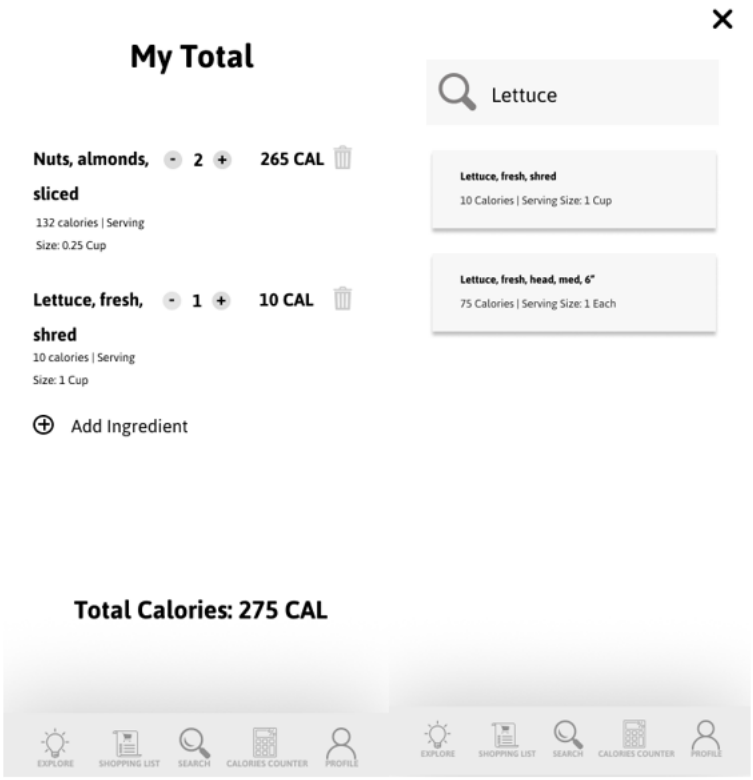

ASIAN
HEALTHY
KOREA
by ingredients
by category

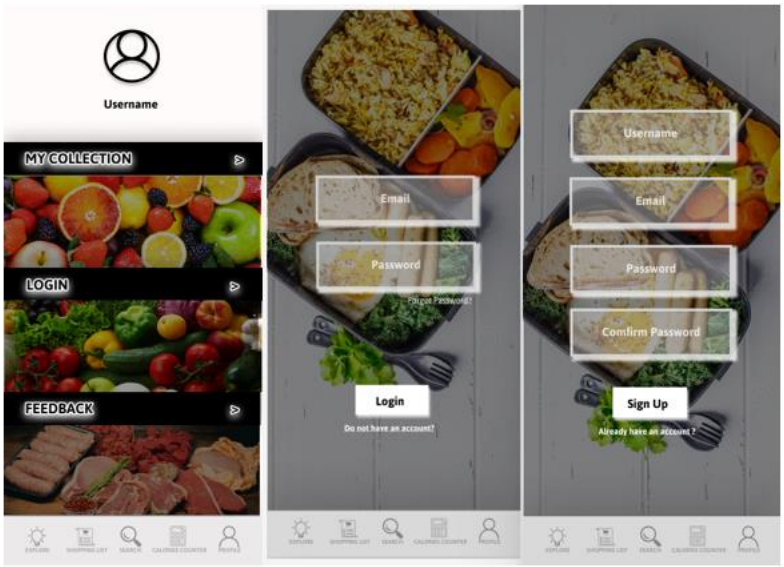

ALL
MEAT
SEAFOOD
VEGATABLES
Carrot
Cucumber
Parsley
MEATS
Beef
Chicken
Pork
Procced


ALL
MEAT
SEAFOOD
VEGATABLES
Are you sure that you want to search an recipe with the following ingredients?
Chicken
Parsley
BACK
MATCH
Ingredient
Ingredient
Ingredient
Procced


SEARCH RESULT
Chicken Marianade with Parsley
Lemon Garlic Parsley Chicken
Chicken with Garlic, Basil, and Parsley


Chicken with Garlic, Basil, and Parsley
4 Ingredients
150 Calories
0.8 Hours
Ingredients
ADD TO SHOPPING LIST
1 tablespoon dried parsley, divided
1 tablespoon dried basil, divided
4 skinless, boneless chicken breast halves
Nutritions
25.6g Protein
4.1g Carbs
3.1g Fats
Steps
1. Preheat oven to 350 degrees F (175 degrees C). Coat a 9x13 inch baking dish with cooking spray
2. Sprinkle 1 teaspoon parsley and 1 teaspoon basil evenly over the bottom of the baking dish
3. Put it in the oven for 10 mins.
4. And it's done
START
Rate For Us
ADD INGREDIENT TO SHOPPING LIST
1 tablespoon dried parsley, divided
1 tablespoon dried basil, divided
4 skinless, boneless chicken breast halves
ADD
Chicken with Garlic, Basil, and Parsley
Put it in the oven for 10 mins.
Start Timer





Appendix 10.7: Link

YouTube link for high fidelity: https://youtu.be/OMGtHkwon10

YouTube link for the application: https://youtu.be/yWkfxKMY29w

GitHub link : https://github.com/kellychin714/HomeChef-final.git

Appendix 10.8: Members Role in Report

Parts of the Report	**Member**
Background	Kiran
Planning and Research	Mukil
Prototyping and Iteration	Wei Xian & Wee Jie
Design Specification	Kelly and MingLin
System Development	Kelly and MingLin
Analysis	Mukil
Evaluation	Kiran
Conclusion	Everyone