F1V3GUY5

COMP3900 - Project Report
Recipe Recommendation System

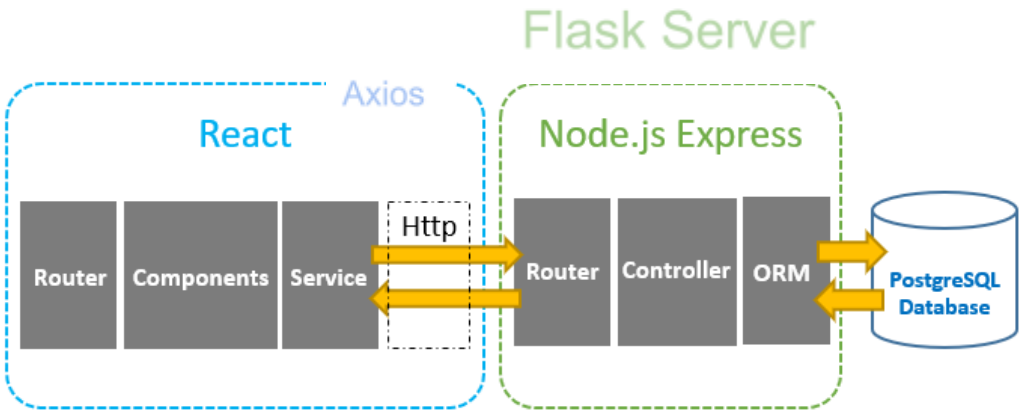
Team Name: F1V3GUY5

Scrum Master	**Edmond Lung(z5160227)**	z5160227@ad.unsw.edu.au
Programmers	**Liam Godfrey(z5256175)**	z5256175@ad.unsw.edu.au
	Bill Mo(z5305298)	z5305298@ad.unsw.edu.au
	Nathan Quan(z5207975)	z5207975@ad.unsw.edu.au

Project Report Submission Date: 05/08/2022

Table of Contents

Design and Architecture



Frontend Architecture

Our project employs a typical and standard API architecture to send requests with the use of APIs to a backend server. Requests are processed and a response is sent in return, declaring the status code, with data or without any error messages. The pages and components seen on our frontend architecture utilise the popular Javascript framework known as React. The system pages that you see when navigating through our website are separated into a pages folder and reusable components are placed in the components folder. To link all pages and navigate through the website, the user uses predetermined routes trafficked by React-Router-DOM. Each of these routes have levels of authentication access where logged in users are only able to access compared to using the website as a guest.

At the core of our frontend system, there are two main user flows within our system, namely recipe explorer (Guest) and logged in user. A recipe explorer is a user that can search, view recipes and its details on our recipe recommendation website. A logged in user is allowed to perform the same actions as a recipe explorer and additionally, able to customise their profile, create and edit recipes, bookmark, comment on and review recipes and also take advantage of tracking their dietary intake using the Diet & Metrics health system. A recipe explorer in this case is used as a base user with the basic functionality our system provides.

Dynamic rendering is used based on conditions within our system to only render certain components, with use of the frontend architecture, depending on the user's level of authentication.

Backend Architecture

The backend architecture was designed around utilising requests to the SQL database system to handle most of the data, with the Flask API as a go-between to interpret frontend requests, and sql responses. This is reflected in the limited local storage used in the backend implementation, and relying on updates to the SQL database to hold the data we need. This was intentional; to allow consistency between the locally run versions of the frontend (especially in testing).

The system was designed to have 7 primary objects, with 7 relational tables that connect information. The seven primary objects were the Users, Cuisines, MealTypes, Recipes, Ingredients, Allergens and Comments. The relational tables provided information needed to link (for example) a user with the allergens, or a recipe with the ingredients used. These were allergen-ingredients, recipe-ingredients, user-allergens, user-bookmarks (user-recipe), user-recentlyviewed (user-recipe), meal history (user-recipe) and recipe-rating (user-recipe).

In terms of actual backend API routes / functions, these were broken down into 5 categories, being: authentication, searching, contributing, health and customising. The authentication routes covered functions such as user login, register, asserting the user owns the recipe they are trying to edit, updating user data, as well as the forgotten password feature.

Searching routes covered the general list view searches, searching for recipes a user has uploaded, as well as the detailed view searching which has more information such as recipe instructions and health details.

Contributing routes were specific to users posting, editing, commenting on and reviewing recipes. These were the key functions that allowed for the community engagement that our project was centred around - user contribution.

Health routes were those that were targeted to our novel functionality of health diet and metrics. These included giving users the ability to set goals, recommending recipes based on what they had eaten compared to the daily average required intake of different food groups, calculating dietary imbalances etc.

Customising was originally planned to contain functionalities for user customisation, such updating allergens, changing their username, bookmarking recipes etc, but after design changes, these were portioned throughout authentication routes / requests for other pages to lessen the requests sent to the backend for a faster user experience.

Additionally, the backend makes use of the third party Google API to send emails via communicating with the frontend and PostgresQL database to handle all business logic.

Description of Functionalities

All users are able to browse recipes based on the ingredients they currently have in their pantry by selecting them in a drop down menu on the search page.

Logged in users can enjoy extra features such as the ability to post/share their own recipes. They also have access to health tracking features. Logged in users can also mark recipes as eaten. Our system processes that information to figure out any dietary imbalances and we recommend recipes to bring their diet back into balance.

To achieve these goals, our team broke the project down into the following core objectives:

Recipe Explorers must be able to:

- **RE1:** Search for recipes.
- **RE2:** View recipes.
- **RE3:** Create and Register an account.
- **RE4:** Log into an account.

Logged in Users should additionally be able to:

- **LU1:** Create and share recipes.
- **LU2:** Comment on, Bookmark and Review other recipes.
- **LU4:** View a more Customised list of recommended recipes.
- **LU5:** Track Calories and Nutrients consumed.
- **LU6:** Customise their profile.
- **LU7:** Create a Health Goal.

Searching and Viewing recipes

As a first-time visitor to the website, you are greeted with the option to either continue as a guest or signing up as a user and logging in to see the recipes available on the platform. Both types of users we define as recipe explorers (base user) and logged in users have the ability to use the dynamic search functions which include search via the following: recipe name, ingredients, cuisine and mealtype. User's that submit any search criteria in these fields will be presented with recipes that fit the search query(s) and be able to view them in further detail. This satisfies objectives **'RE1: Search for recipes'** and **'RE2: View Recipes'** as it allows any type of user on our system to search for recipes of their choosing and select to view to attain the information on the ingredients, instructions and quantities needed to make use of this recipe.

Registration and Authentication

From a retention standpoint, we understood the premise of allowing guests to use our platform for the basic functionality of searching and viewing recipes. However, creating an account with their credentials unlocks aspects of our system that guests cannot access such as the profile screen, ability to create and share recipes on the platform and comment on, bookmark and review recipes etc (The remaining functionalities are listed above in core objectives). Upon creating an account, the user is automatically navigated to the login screen after a prompt and can log in with the registered credentials. Logged in users are greeted with a welcome and can freely traverse the functionalities (as above). This satisfies objectives: **'RE3: Create and Register an account'** and **'RE4: Log into an account'** as it allows users to gain a higher level of authentication and make full use of the 'Logged In' core objectives.

Creation of Recipes

In order to promote people to use this website, we want to make sure we have a vast selection of recipes available to recipe explorers. In order to address this issue, we allow users to create and share their own recipes. Logged in users can create their own recipes on the "My Recipes' page by clicking 'Create a Recipe' and filling in the form. If the form is filled in correctly, a recipe is created and stored in the database and is automatically viewable when a search criteria matches. Additionally, creators of recipes can view the recipe on the 'My Recipes' page. This is an integral part of our application as it gives users the power to share their ideas and recipes they've seen/cooked themselves. This satisfies objective **'LU1: Create and Share recipes'** as the collaboration of many of these users will grow the systems database of recipes to be shared to all.

Engaging with recipes

Allowing authenticated users to engage directly with the recipe and with other users is a key functionality we have taken into consideration to build retention on the platform. Users can easily bookmark recipes with a star button on either the search result page or on the page displaying the recipe details if they wish to come back to it later. Bookmarked recipes are stored in the profile section in the 'Favourite Recipes' Section. Within the later page, users can view comments, leave a comment and review out of 5 stars to promote engagement and the sharing of feedback. This satisfies objective **'LU2: Comment on, Bookmark and Review other recipes'** as users are given the ability to engage in different mediums on any recipe of their choosing.

Track Health Metrics, Caloric Intake and Health Goals

Our application allows users to track key health metrics and caloric intake by making use of the 'Eaten' function on each recipe. When a recipe is eaten, it is marked in the database and the backend server calculates the nutrients and calories consumed. This can be seen on the Diet/Metrics page, where authenticated users are able to see a breakdown on the calories and nutrients consumed over the past week. This makes diet management more simplified for the user. An additional component to the eaten function allows users to view recommended recipes on the Diet/Metrics Page which lists three recipes

In order to incentivise a healthy diet, users are able to set Goals for how much they'd like to limit their caloric intake either on a daily or weekly basis. Users can do this in the diet/metrics section of their profile. Users can also mark recipes as eaten. Our system processes that information to figure out any dietary imbalances and we recommend a customised list of recipes to bring their diet back into balance. (e.g. If not consuming enough protein, more proteinous recipes will be recommended).

This satisfies the following objectives **'LU5: Track Calories and Nutrients Consumed'** and **'LU7: Create a Health Goal'** and **'LU4: View a more customised list of recommended recipes'**.

Customisation

The Health epic story includes a **novel idea** that we have not seen used anywhere else in a recipe recommendation system. While research suggests that this isn't a new idea (with the large variety of health tools out there such as MyFitnessPal), this is new in the recipe recommendation system, and new with our type of implementation. The novel implementation part of this idea is recommending recipes based on their goal (as well as to those without a goal) and keeping track of nutrients consumed. Other similar tools simply record the nutrients consumed when you input it into the app, but won't necessarily tell you what you should eat in order to reach your goal / a healthy nutrient balance.

Additionally to this, authenticated users are able to change their username, password and a security customisation feature that allows users to secure their account if they have forgotten their password. Since it is changing an aspect specific to their account, it falls under customisation. In relation to dietary requirements, users can select the allergies they have in the profile page that appends to a list and is stored in the database. Recipes that are searched on the platform which include their allergy such as shellfish are excluded.

This satisfies the objective **'LU6: Customise their profile'** as it affords authenticated users the ability to fine tune their profile to their specific needs and be offered a tailored user experience when on utilising the platform for its health tracking, goal setting and customisation.

Third Party API/Libraries

MaterialUI

Material UI (Google, 2014) is a user interface component library for React.

MaterialUI is essential to our application as it allows us to construct attractive, consistent components for faster and easier web pages development while adhering to modern web design principles. It helps ensure that our application is fast and responsive. Our system has implemented many of MUI's components and material-icons. The MultipleSelect, Login and Register are examples that contain and reference the MaterialUI library.

Material UI is released under the MIT licence.

React-Router-dom

React-router (Remix Software, 2019) is the main library and react-router-dom is used in browser based web applications generally.

React-router-dom is pivotal to our application as it empowers us to use a single page application with only one page, App.js, and different components which get displayed based on routes.

React-router-dom is licensed under Facebook Inc. Documentation is licensed under CC BY 4.0.

ReactJS

ReactJS (Facebook, 2022) is a JavaScript library used for building reusable UI components.

React is essential to our application as it allows us to create frameworks and user interfaces that render dynamically as the users and database change over time. As more recipe explorers sign up to the system, it is important to account for the scalability of our system by using a simple programming model that brings outstanding performance.

React is licensed under Facebook Inc. Documentation is licensed under CC BY 4.0

Google APIs

Google APIs provide access to a wide set of google services ranging from Google Drive, Gmail, to other cloud services.

Our application needs to send emails to users who have forgotten their password with a new temporary one.

We send emails from allofrandomness@gmail.com. To do that, our server needs to be granted certain access permissions to that google account. These permissions are granted through configuration via Google Cloud Platform and via OAuth. Both of these are implemented through this API.

Python3 Flask

Flask (Ronacher, 2010) is a micro web framework that was utilised on the backend for building the web application.

Flask was essential to our design and was used to allow the frontend to communicate with backend scripts / the database, as well as provide authentication for the users with the help of the flask jwt extension.

Flask is licensed under a three clause BSD Licence.

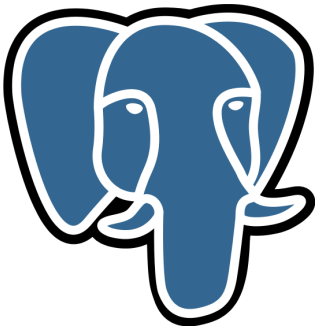
USDA Ingredients

FoodData Central is an integrated data system that provides expanded nutrient profile data and links to related agricultural and experimental research.

USDA Ingredients (U.S. Department of Agriculture, 2019) was essential to our database design and was used to allow users of our website the ability to search over 1000 ingredients and track nutrient profile data of every ingredient. We are able to use their FoodData Central as they are a public domain, not copyrighted and require no permission to use the API.

USDA Ingredients is not licensed.

PostgreSQL Server



Postgres (The PostgreSQL Global Development Group, 2019) is an open-source object-relational database system and was the database system used on the backend to store persistent information such as user login credentials, recipes, ingredients, user allergens etc.

PostgreSQL is released under the PostgreSQL Licence, a liberal Open Source licence, similar to the BSD or MIT licence.

Implementation Challenges

Scope Estimations

Our team encountered severe implementation issues as the sprint deadlines drew closer and towards each project demo. This was mainly in part due to poor scope estimation and inability to make correct assumptions when estimating time, resources needed and effort needed to implement our user stories. Looking back, when planning the sequence of implementation of our user stories, we should have adopted a Fibonacci Sequencing or at the very minimum allocated 'story points' to each of the user stories so we would know which functionalities required a smaller or larger amount of time.

As features became more complex, our completion of user stories for each sprint was beyond the deadline and that reflected a lack of consideration of time needed to complete the user stories within each sprint. This caused a flow-and-effect on the last sprint and a majority of our implementation was in the final few days as incomplete user stories from the previous sprint were moved into it.

Towards the end of the project, we as a team discussed the estimations we made on a novel functionality, being the shopping API from Woolworths and Coles we had scoped to implement. However, upon further research, we were unable to do so as there was no API available and we had to compromise by leaving out the functionality entirely of our project. Initially, we aimed to dream big and bring a high level of novelty to our recipe recommendation systems but our scope estimations were far from correct.

Bug Fixes and Merge Conflicts

With the development of an application involving multiple streams of implementation, consistent testing must be done to ensure the project is delivered with the highest quality and outcome. Upon coding and implementing our application, many bugs and issues arose. Our developers initially implemented all functionalities within their own separate branch with poor naming standards that had no relevance to what was being implemented.

Initially, we had structured the implementation phases during the sprints to resolve, test and fix bugs and issues ourselves until they were resolved before merging to the main branch. We realise this was a monumental issue as this would sometimes break the main branch, making it uncompilable and unrunnable, halting our project progress. Debugging and attempting to resolve merge conflicts created confusion through many instances and we believed that it was best practice moving forward to implement pair programming sessions to allow code to be double checked and assure quality before pushes and merges.

Ultimately, throughout the project it was difficult to maintain a working main branch that worked at all times. This was due to our inexperience with component testing and our ability to produce high quality code in a short time frame. Each of the team members experienced issues relating to managing and installing dependencies on their local systems to be able to run either the frontend or backend, halting more progress being made. Our team was too rushed into developing functionality that we neglected proper checks and branch management that were expected of final year computer science students.

Unfamiliar Tools

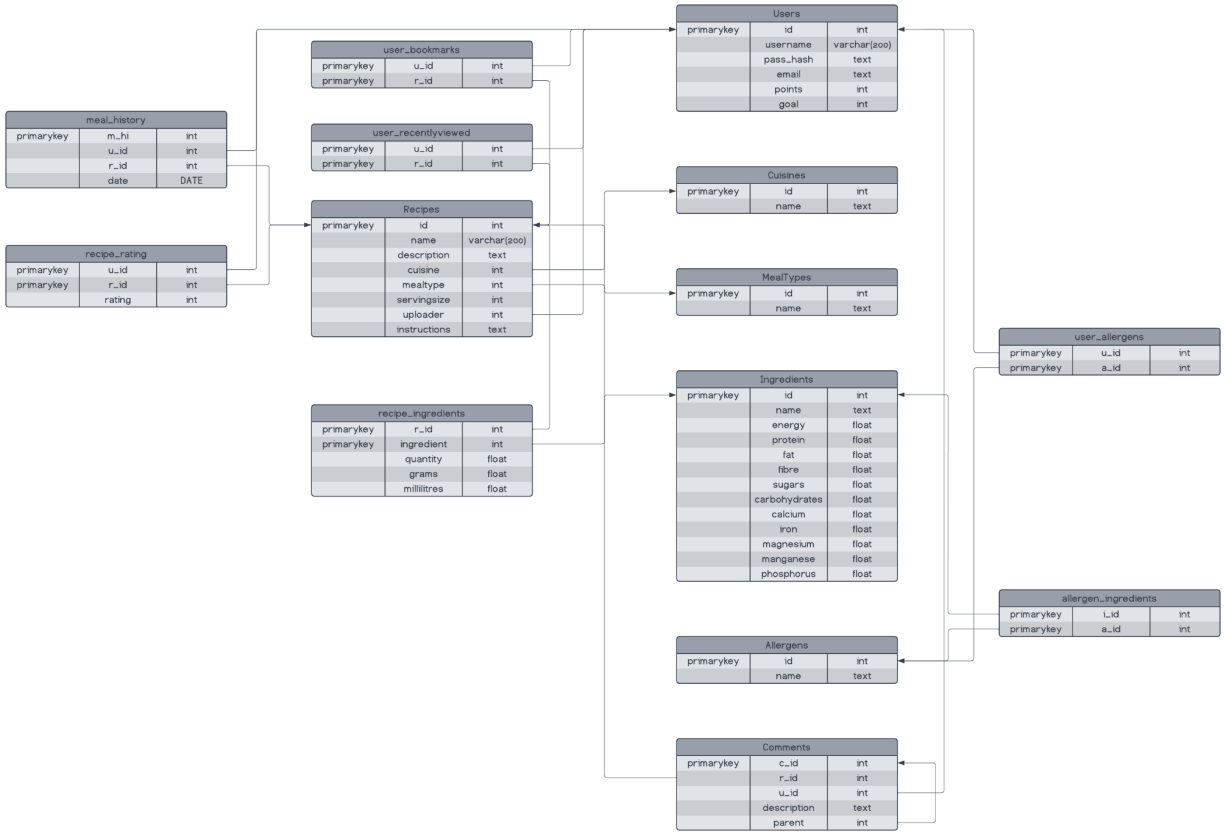
While certain tools such as the use of postgres, psycopg2 (python integration to postgres server) and flask had been used before in other courses, they hadn't been used to the same extent as required in this project. Security features such as the use of JWT tokens, proper use of flask, error handling, and figuring out how to best communicate with the database to avoid as many sync errors as possible proved to be a challenge. Setting up an external psql server took time to learn how to do, as well as to ensure that correct ports were open, or permissions were given to the user to allow integration with the backend.

There was also a learning curve for some members due to the lack of number of people with frontend experience, meaning that other members needed to learn aspects of coding in react / frontend coding.

Algorithmic Complexity

Holding all persistent data in an external SQL server proved to have challenges in regards to obtaining information, evident in the search function, which required small changes to be made to the SQL query based on what fields are present which, unfortunately, SQL does not necessarily fully support. To get around this, in the search_general function (for example), an SQL query was dynamically created based on the input supplied. The reason for doing this rather than pulling all data and sorting it on the backend with python (which would have arguably been easier) was purely due to the speeds at which SQL can achieve the search compared to python. We wanted to design a system that, at a larger scale, would still run efficiently, which is why we opted for utilising optimised SQL queries for the majority of our data retrieval, only pulling the data we needed.

Database Design



With our design centred around the use of persistent data (for consistency across multiple instances of the web app being run), SQL and a good database design was critical. In the initial planning phase, this was one of the primary focuses of our backend design. The challenges associated with this, however, were primarily due to how we chose to address issues we'd seen in other recipe recommendation systems, such as recipes being unavailable because the website hosting the information was down, by hosting everything on our own server. With this, we needed a way to import data quickly, easily, and without fail, as well as to back up the database in case something went wrong.

To overcome these challenges, a script was written on the server hosting the psql instance that ran every **x** minutes (this was changed multiple times) to backup all data from the server, append any new information to the .csv files in backend->data->*.csv, and then reload the server (this was the reason for a user who only pushed changes to the psql_server branch). The reason for this reload was intended to allow new entries to be added to the .csv files and then uploaded to the server on the next cycle, rather than having to send a request to the psql server. This, however, had unintended consequences in how different fields were automated. One such consequence was that the automated counter for the user id (and other automated id's) would be reset at each reload, meaning that a new user could not be created because of clashing keys. To explain this further in an

example, assume there are 5 users in the database each with unique ids from 1-5 that are generated automatically from the server when other fields are filled out. The psql server is ready to assign a new user the id of 6 (since it's counted 1-5 so far), but then the server is reloaded, the counter is reset to 1, but there are still 5 users in the database, each with ids up to 5.

To overcome this issue, rather than reloading the database every 30 minutes, we simply opted to back it up every 30 minutes rather than reload the server. New information was uploaded through the web app frontend (when that was set up) to first ensure that it was working correctly, but also to allow easy data upload and non-clashing data.

Functions Optimising Run Time

Intake Overview: Our application helps users maintain a balanced diet by recommending recipes to eat. Users are able to mark recipes as "eaten". Our system then reviews the user's past nutrient intake to find any imbalances. Once imbalances are found, our system then recommends recipes to bring the user's diet back into balance.

This process has been optimised by running the bulk of the logic in SQL instead of python. This is made possible by our database design which separates recipes, recipe ingredients and ingredient details into separate tables and linking them with references. Recipe ingredients allow us to quickly query all recipes and find what ingredients they contain. The ingredients table contains all the nutritional values. This design allows us to perform table joins and row summation in a manner that allows recipes to be ranked from best to worst match.

Search: Searching for specific recipes was optimised by using a dynamic SQL statement as opposed to filtering results in python. This allows for faster search queries, and a scalable approach to gathering search results. This 'dynamic SQL' query was achieved by appending search terms to the original SQL query based on user supplied input, allowing the SQL search to only return the necessary information to the backend API server, to then be forwarded onto the frontend.

As mentioned earlier, the primary purpose for this was for scalability, to allow optimised speeds when there are thousands of recipes to search for. The alternative (being pulling all recipes and filtering out results in python) would not allow users to search the database efficiently as the size of the quantity of recipes increased.

User Documentation

Setup and Configure

This project requires npm, flask and pip (to install packages).

Ubuntu / Debian-based distros (aptitude package manager)

```
apt update && \
apt install -y npm \
python3-pip \
python3-venv \
python3-flask
```

Once these packages are installed, the frontend and backend will need to be run separately.

Frontend

Navigate to the frontend folder

```
cd frontend
```

and run the following to install required packages (this only needs to be run once)

```
npm install
```

After the required npm packages have been installed, to start the frontend server:

```
npm install
```

Backend

In a separate shell, navigate to the backend folder

```
cd backend
```

From here, on Unix based systems, we can either start the program (using a virtual environment) by running the start.sh executable. Or, to install the required packages, run the following:

```
python3 -m pip install -r requirements.txt
```

The above only needs to be run once. Once this has installed successfully, simply type:

```
flask run
```

To start the backend.

Open the program

Once `npm` start is run, the site should be launched in your default browser, but in the instance that it's not, navigate to `http://localhost:3000`

Additional Information

Emails are sent from allofrandomness@gmail.com and the server gets access permission through OAuth. This permission is persistent as long as two files are present in /backend/src/gmail: "client_secret.json" and "session_credential.json".

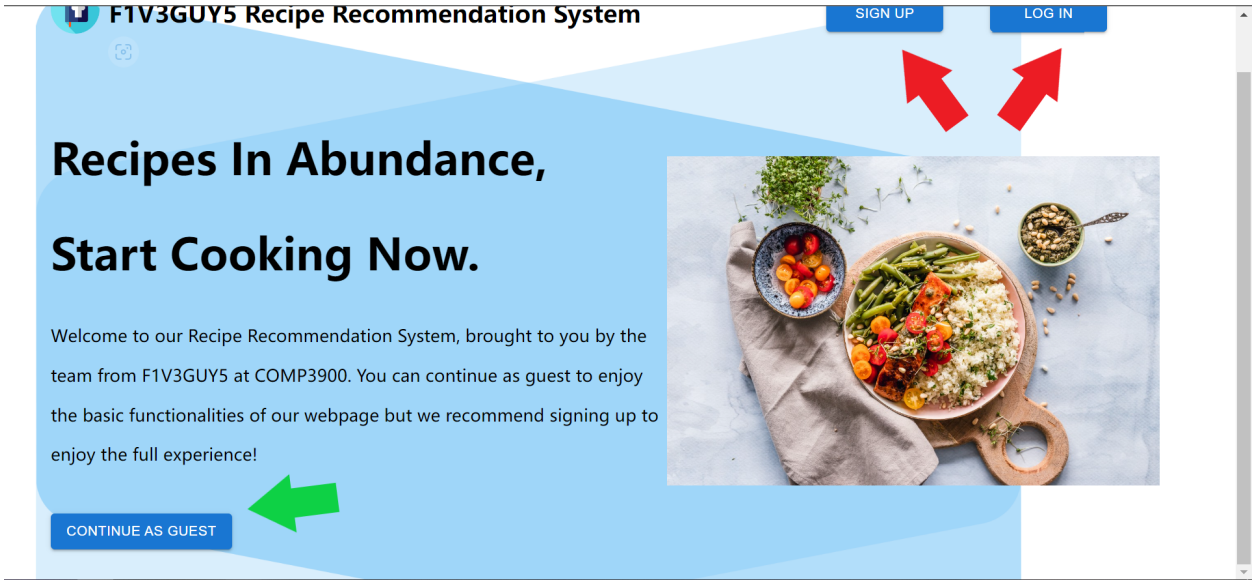
We will make sure these two files are present and the tokens within are still active when we submit our source code. However in the off chance the tokens do expire, follow the steps below:

1. Open the backend server (http://127.0.0.1:5000/) in the browser
2. Navigate to (http://127.0.0.1:5000/test) through this link or by clicking "Test an API request" on the index page
3. This will take you to a gmail sign on page. Sign in to this google account:
    - Email: allofrandomness@gmail.com
    - Password: iamrandom123#
4. The server is now ready to send emails.

How to Use the System

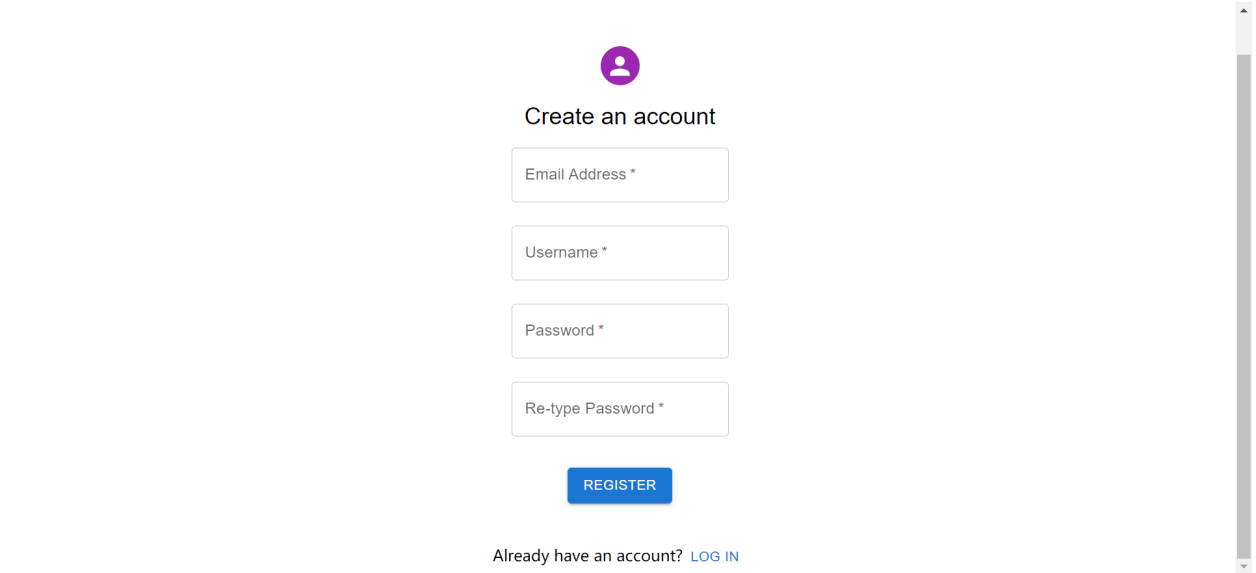
Authentication

1. To use our system as a logged in user, click LOG IN to log in if you have an account. Otherwise, click SIGN UP to create an account.
2. Alternatively, you can follow the green arrow to continue as a guest, though you can only use limited functions.



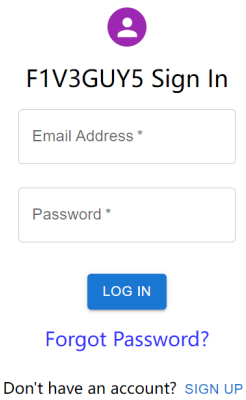
Sign up

1. Fill in all the required fields.
2. Click REGISTER.



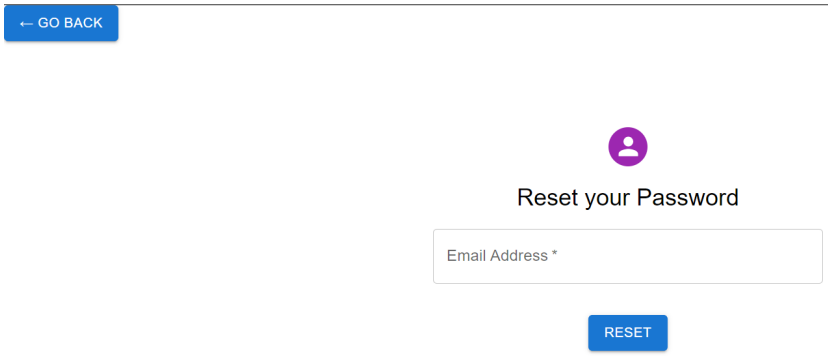
Log in

1. Fill in your email address and password.
2. Click LOG IN.

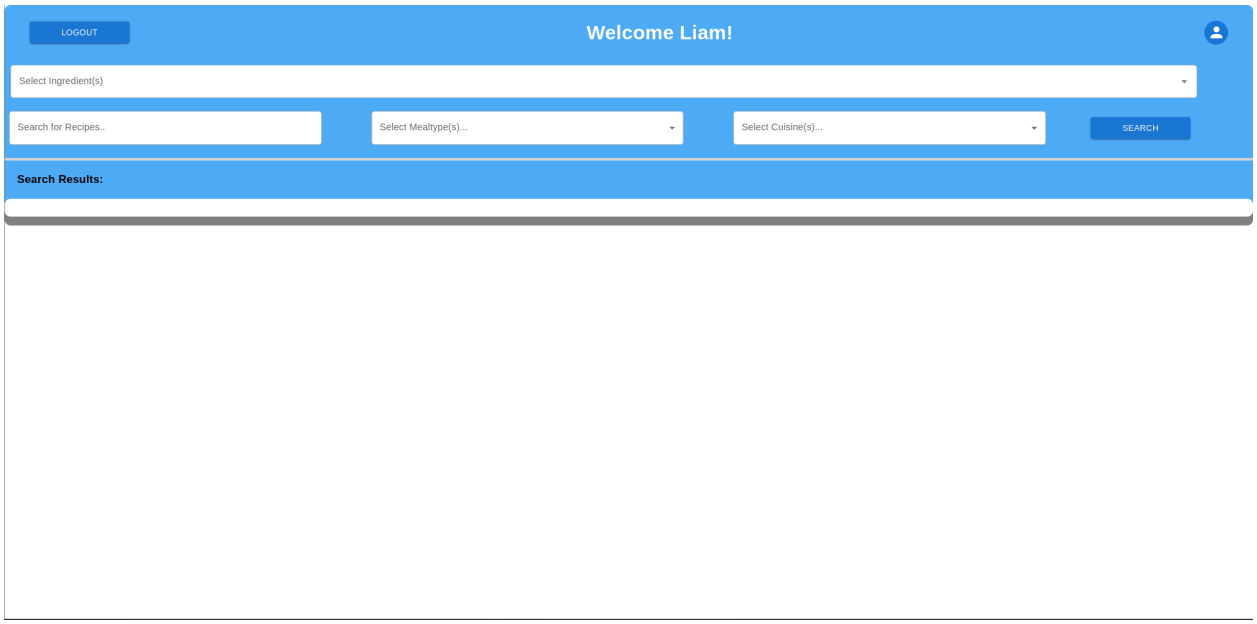


Reset password

1. Click Forgot Password on login page.
2. Fill in your email.
3. Click RESET to receive a verification message



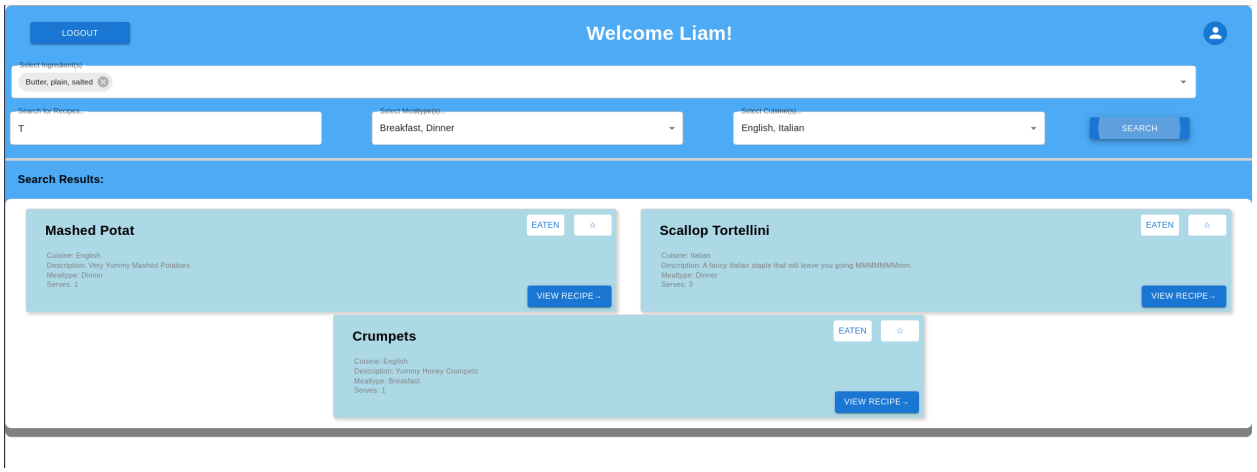
Searching



There are several combinations of searches available. Each of the fields in the search can include as many selections as the user would like, or none at all.

To grab all recipes in the database:

1. Click on the SEARCH button



To search for a specific recipe:

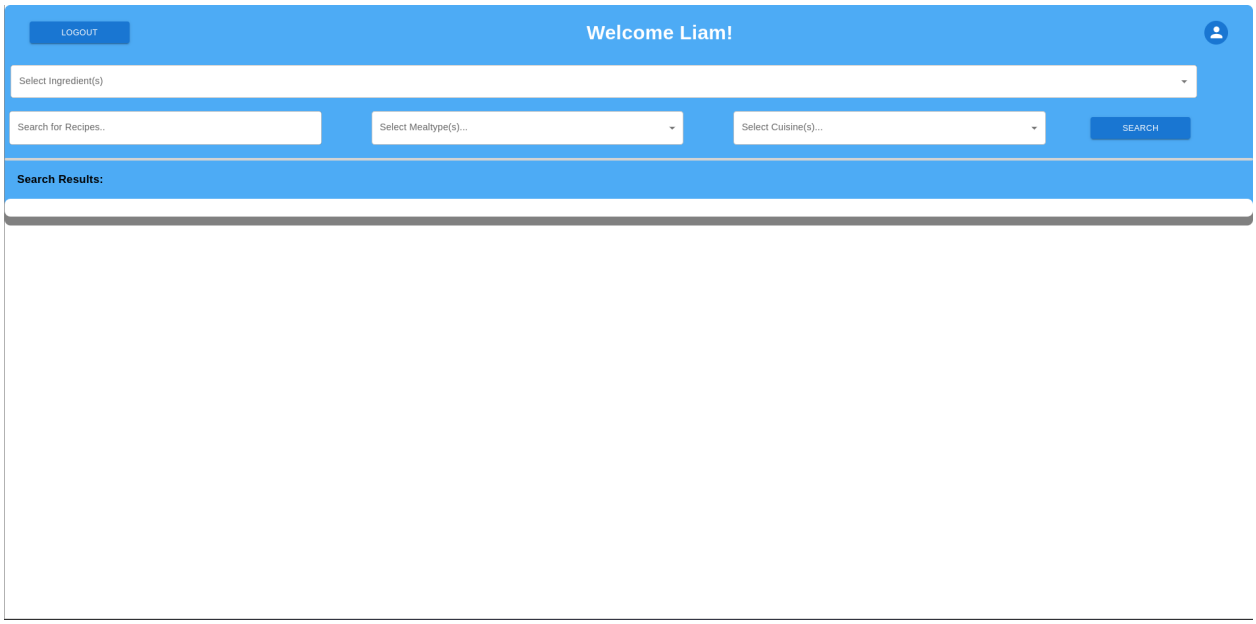
1. Click on 'Select Ingredient(s)'
2. Select the ingredient(s) you'd like to search for (if any).
    a. TIP: You can also type in the name of the ingredient to find it faster!
3. Click on the 'Search for Recipes' textbox and type in any part of the name of the recipe you'd like to search for (e.g. a search like 'ota' would return 'Mashed Potat')
4. Click on 'Select Mealtypes' and select any (multiple select) meal types you would like to include in your search filter.

5. Click on ‘Select Cuisines’ and select any (multiple select) cuisines you would like to include in your search filter
6. Click on the SEARCH button

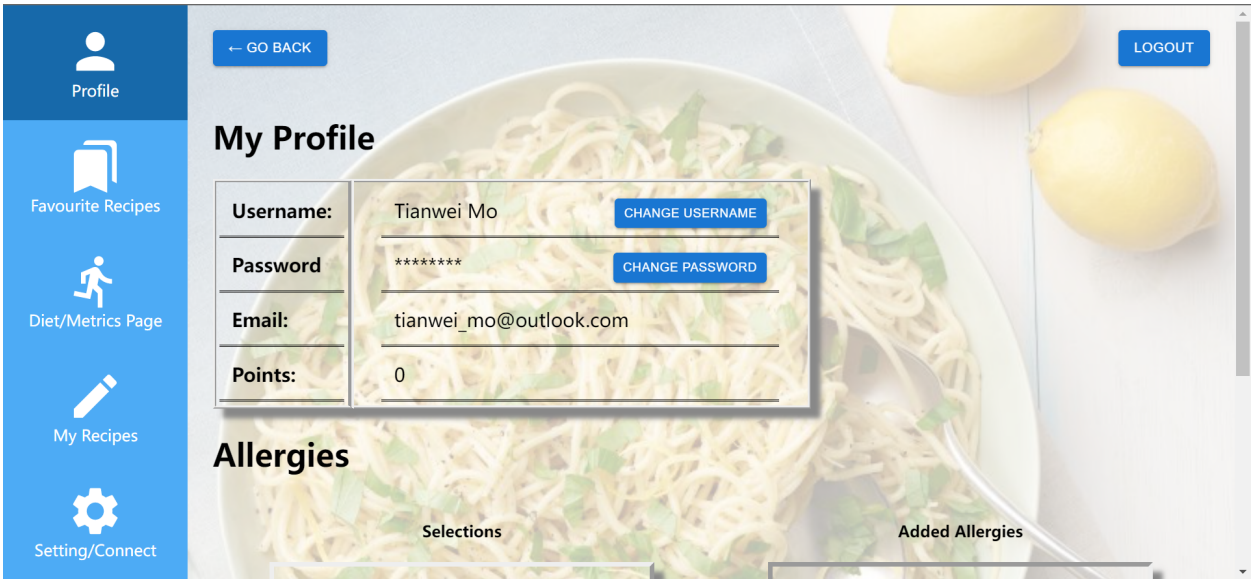
Customise profile

Change username and password

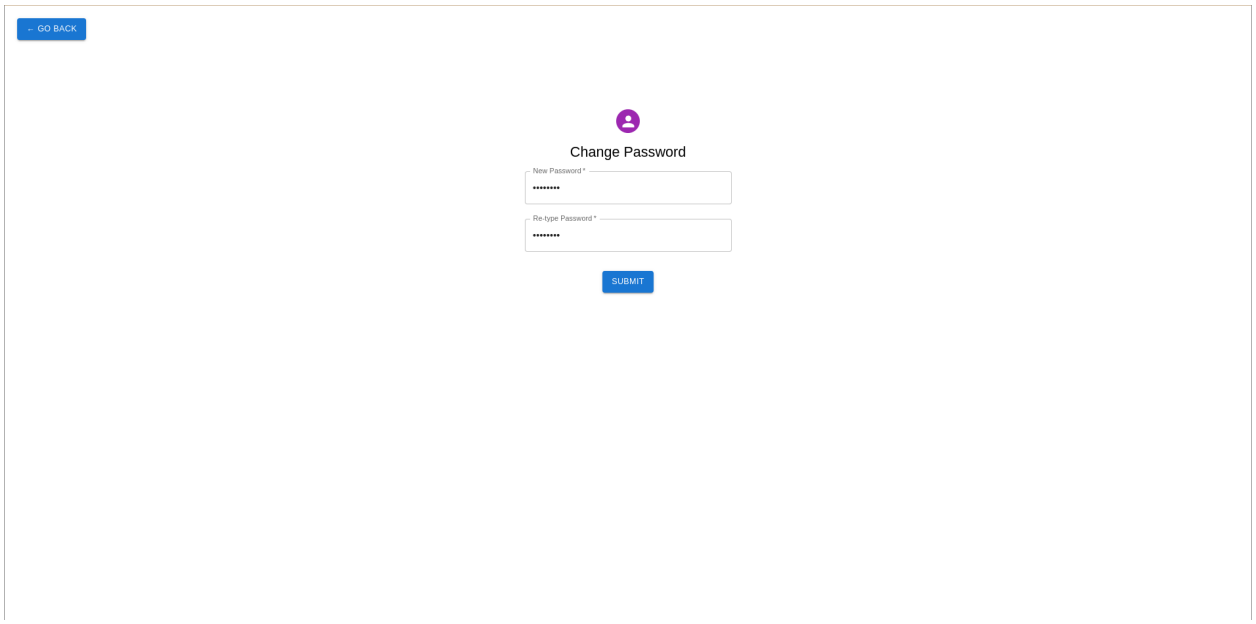
1. Click the icon on the top right of the dashboard.

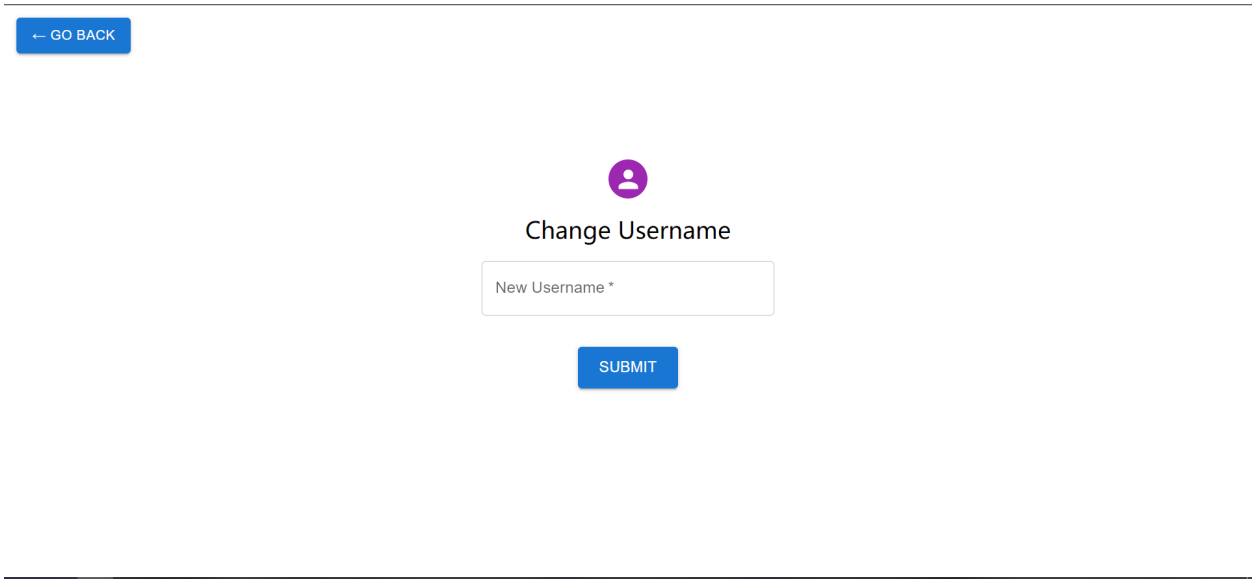


2. If no information shows up, refresh it.
3. Click CHANGE USERNAME if you want to change your username or CHANGE PASSWORD if you want to change your password.



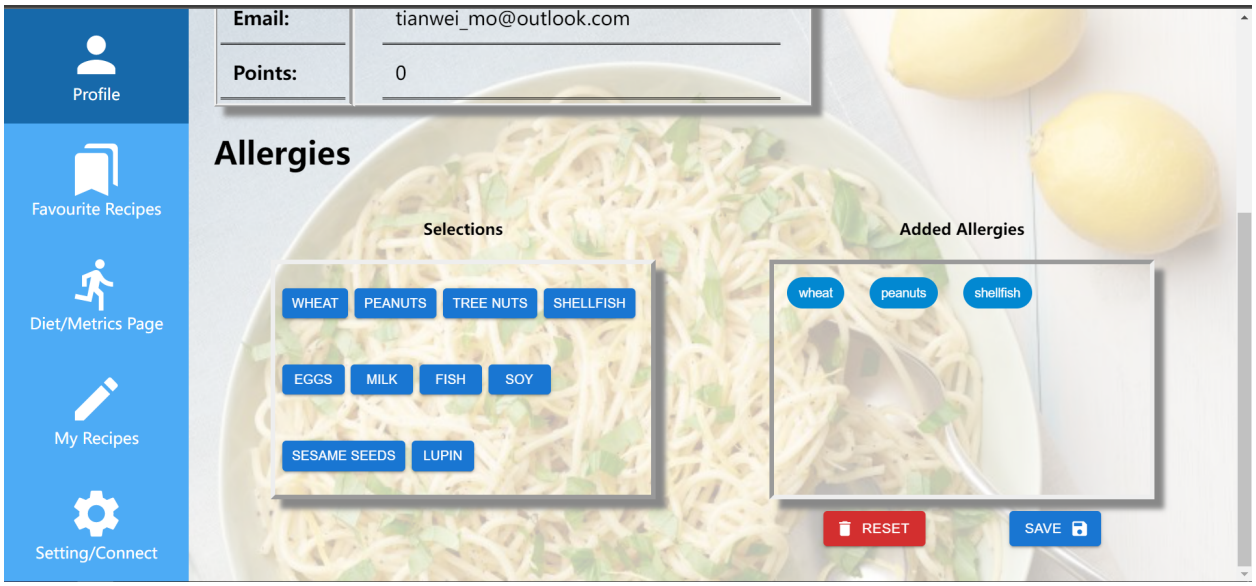
4. Fill in your new username or password.
5. Click SUBMIT and you will receive an alert to navigate you back to profile page if success.





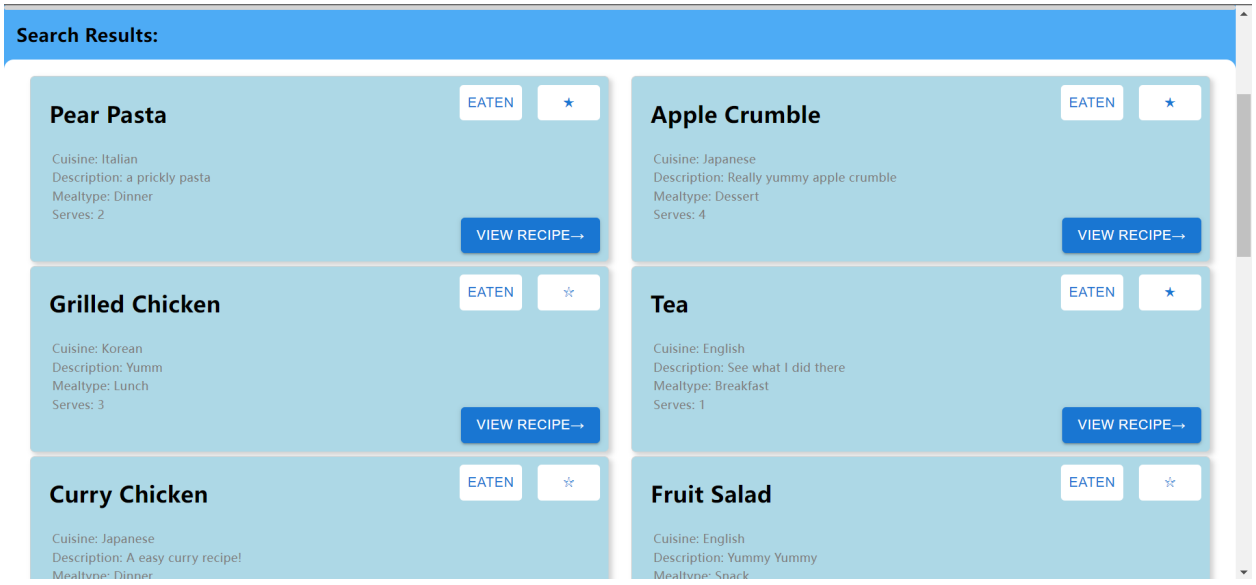
Change Allergens

1. On profile page, click any allergens to add from the selection box.
2. Click SAVE if you are satisfied with changes. Otherwise click RESET to clean your allergen box.

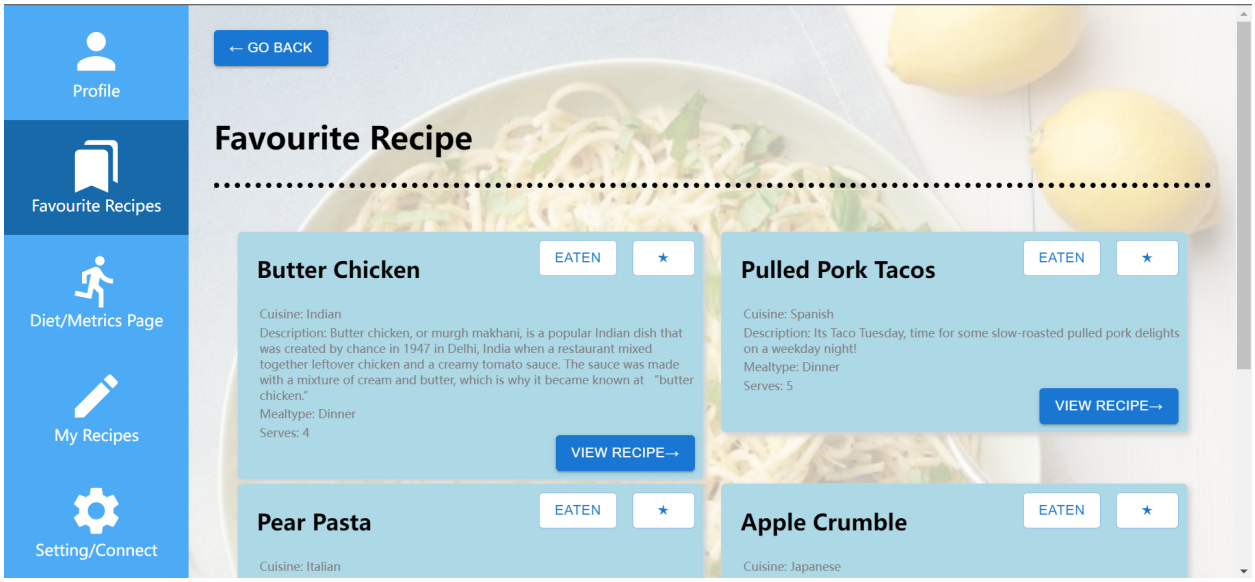


Bookmark recipes

1. To bookmark or unbookmark recipes on the dashboard, click the star button on the top right of the recipes.

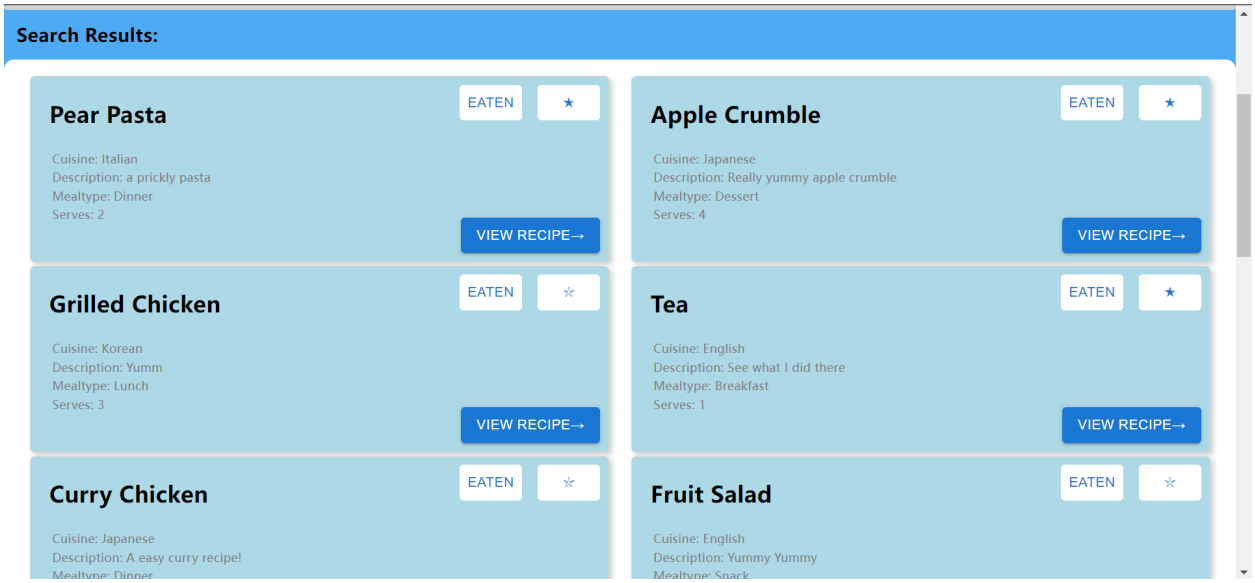


2. On the profile page, Click Favourite Recipes on the navigation bar to see your bookmarked recipes.
3. Should you want to unbookmark any recipe, click on the star to remove them.

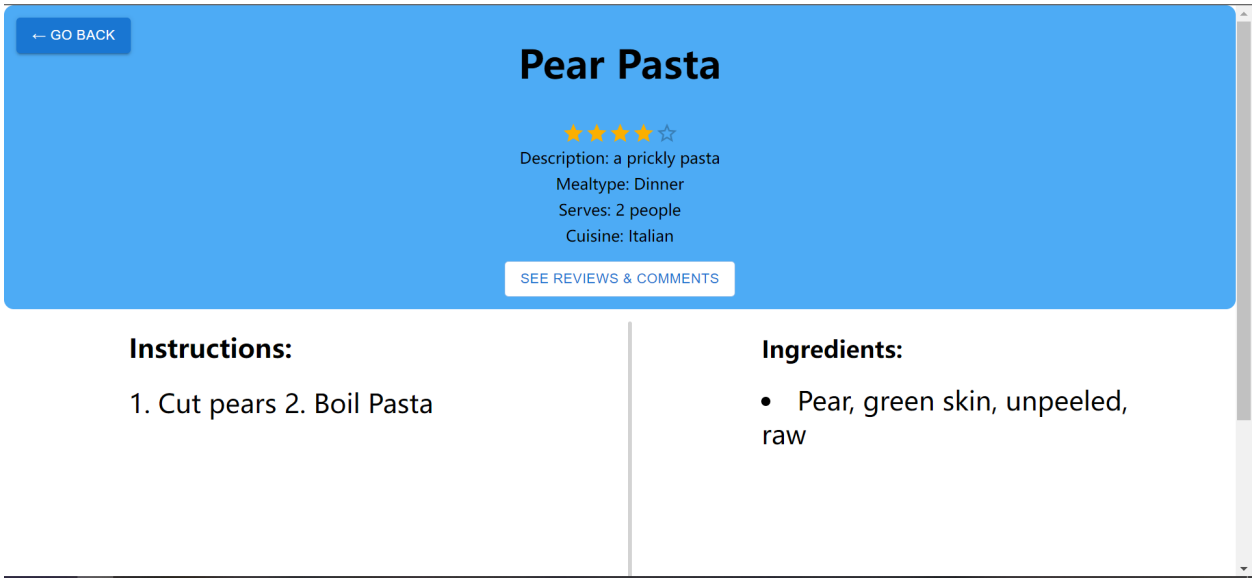


Comment and rating

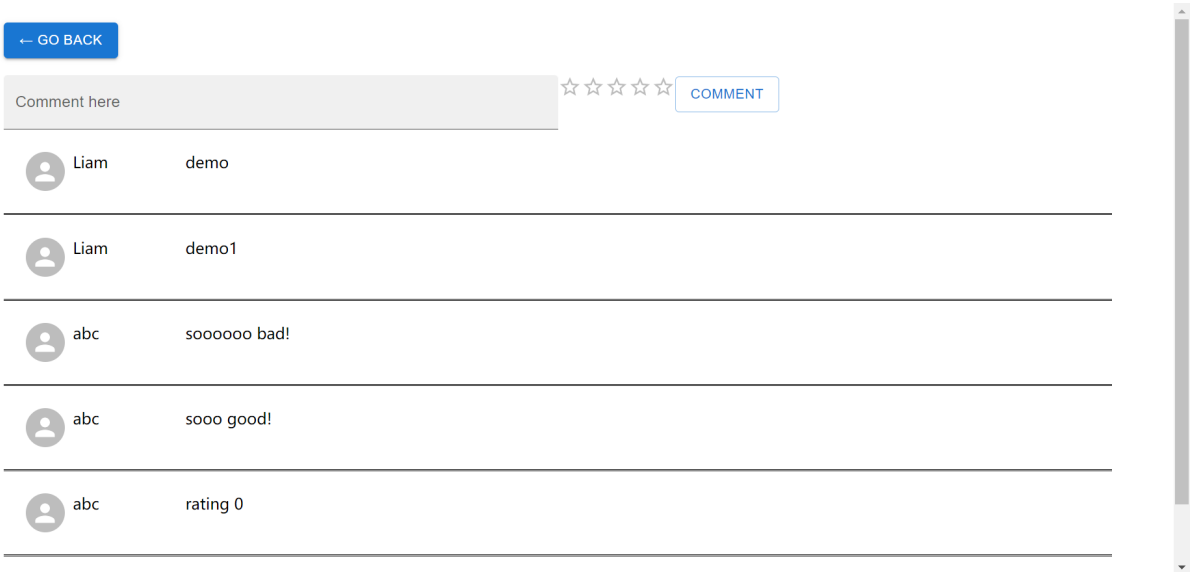
1. On the dashboard, click VIEW RECIPE to see recipe details.



2. The average rating for recipes are shown here.
3. Click SEE REVIEWS & COMMENTS.

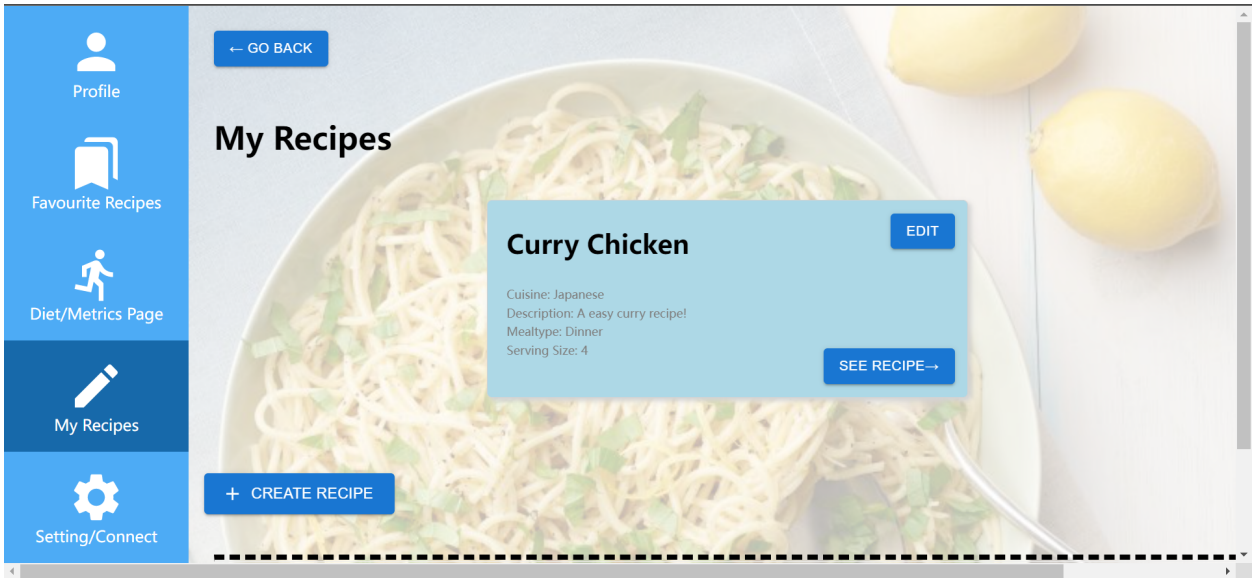


4. Comments and reviews for the recipe are displayed here.
5. Add your comment in the text field.
6. Click stars to rate the recipe.
7. Click COMMENT to submit your comment and rating.

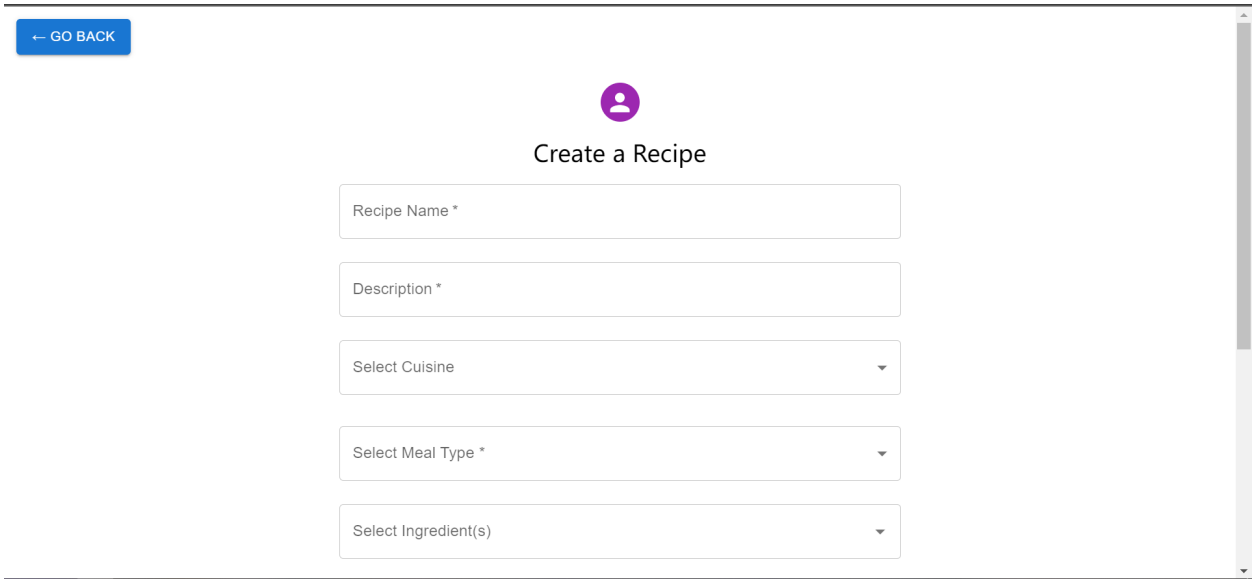


Create recipes

1. On the profile page, click My Recipes to navigate to My Recipe page.
2. Click CREATE RECIPE to open a form.

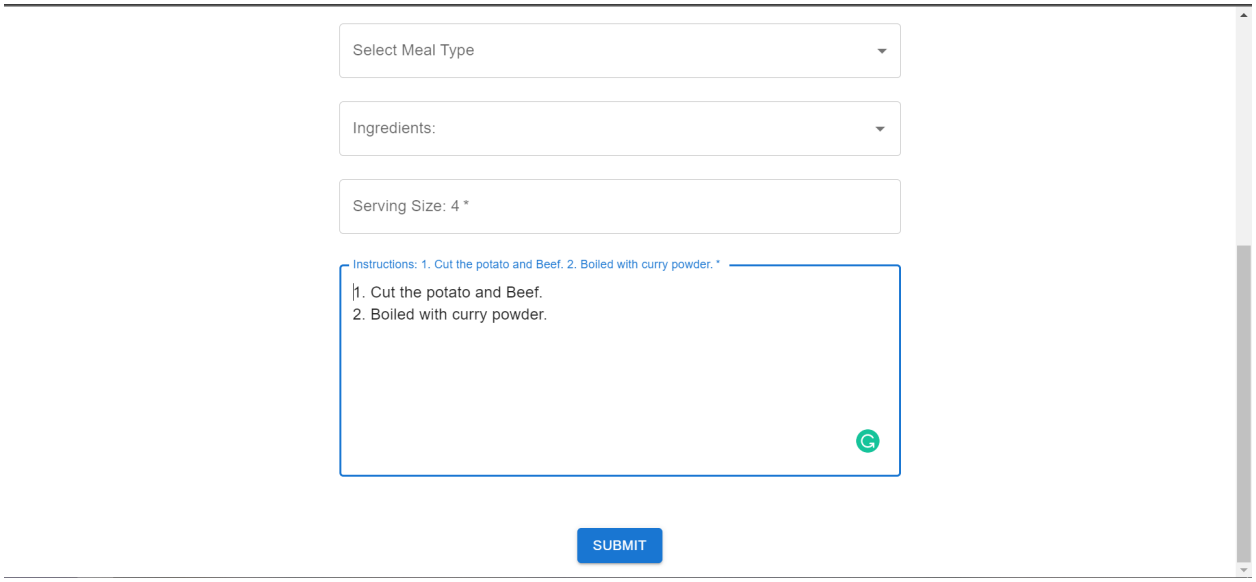


3. Fill in all required fields and click SUBMIT at the bottom to post the recipe.



Edit recipes

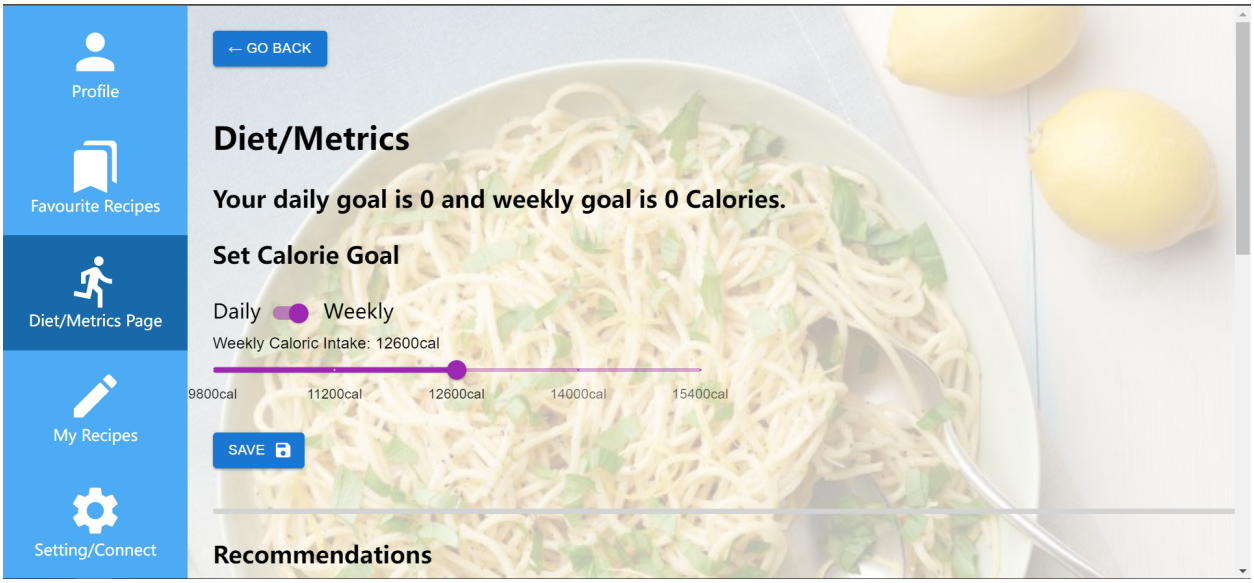
1. Get to My Recipe page.
2. For any recipe you've created, click EDIT on the top right of the recipe.
3. Fill in the form and click SUBMIT on the bottom to save the changes.



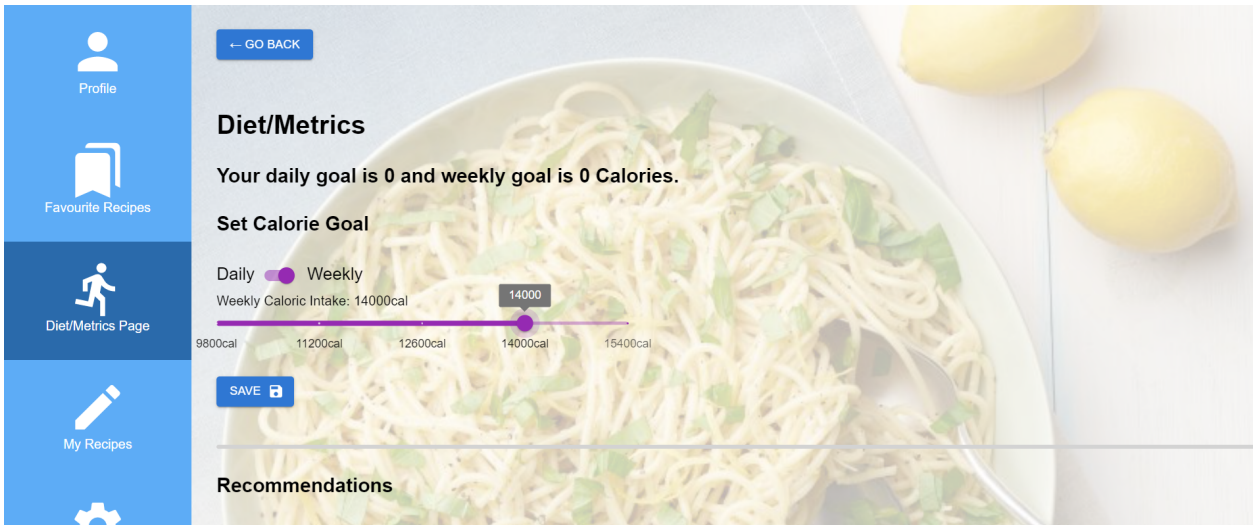
Diet and Metrics

Set Health Goal

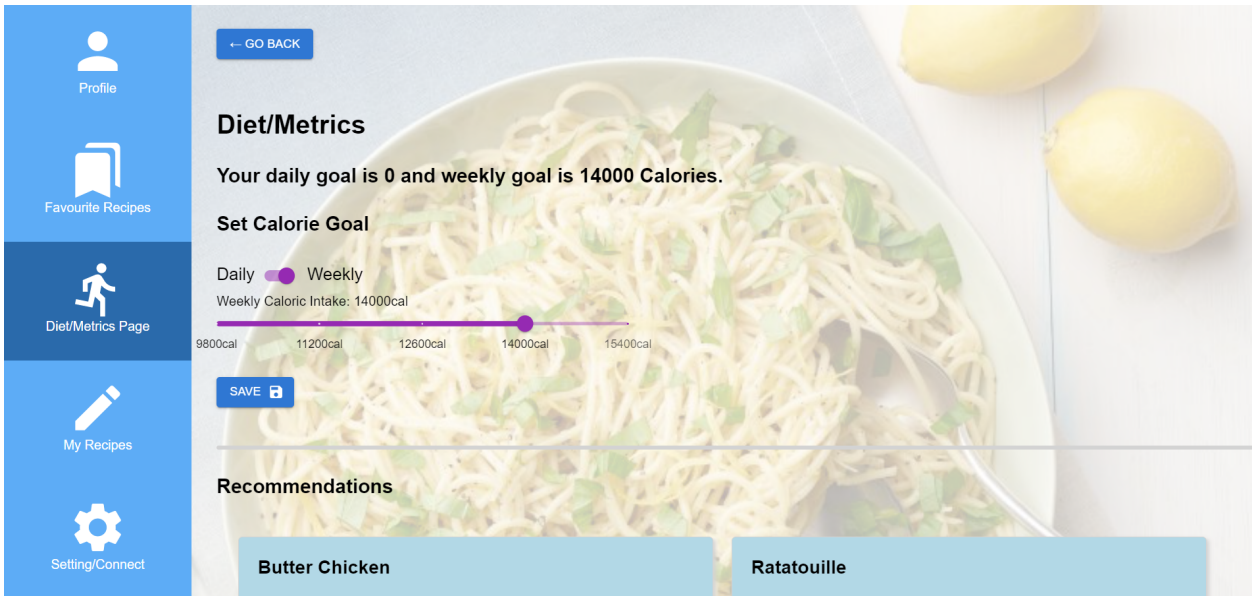
1. If you are on the Go to Diet/Metrics Page by



2. Change the switch from Daily to Weekly or vice versa depending on your goal timeframe.
3. Move the slider to your prefered caloric intake for that timeframe.
4. Click ‘Save’.

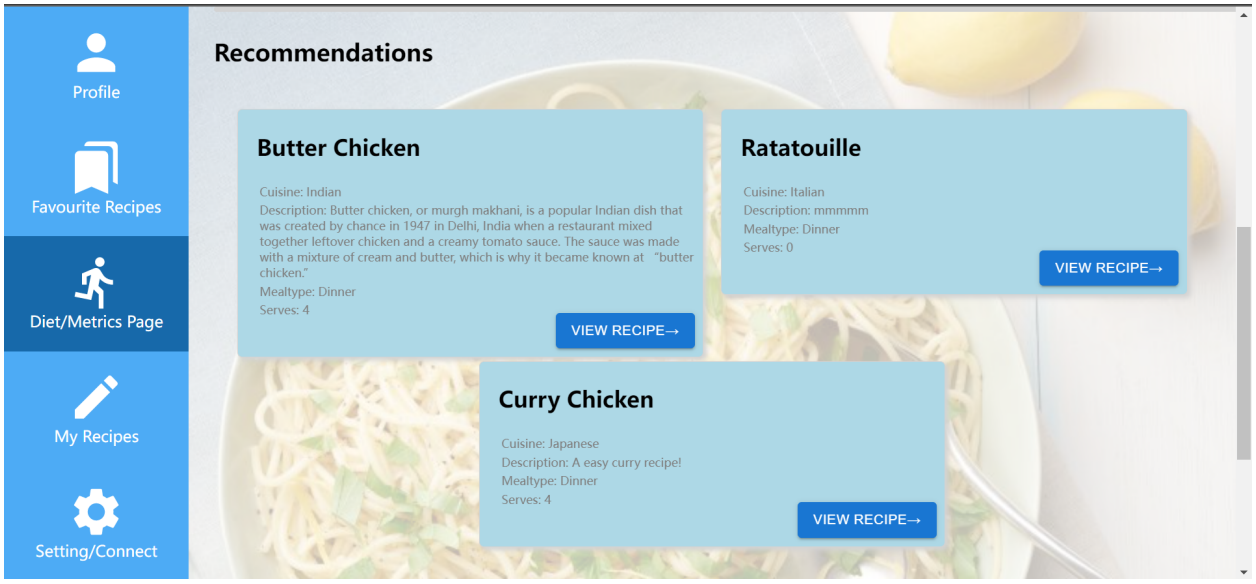


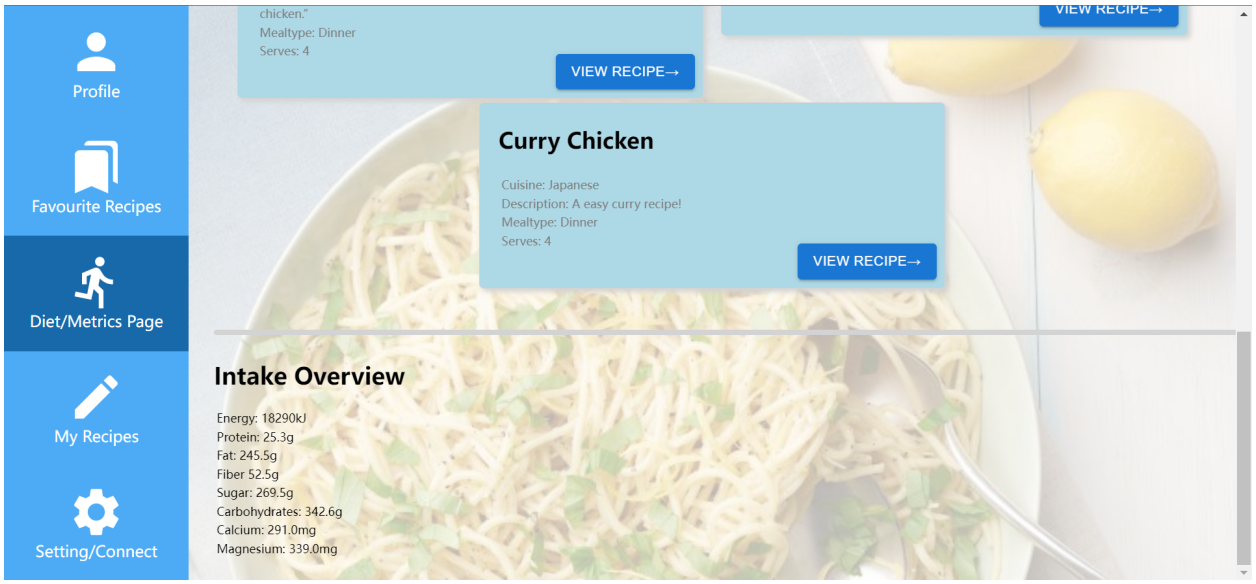
5. An update will show in the daily goal or weekly goal.



Recommendations/Intake Overview

1. Scroll down to the recommendations section on the Diet/Metrics Page.
2. Recommendations and Intake overview are presented when the page is loaded.
3. click 'View Recipe' if you would like to view the recipe details.





References

Google (2014). *MUI: The React component library you always wanted*. [online] mui.com. Available at: https://mui.com/.

Remix Software (2019). *React Router: Declarative Routing for React*. [online] ReactRouterWebsite. Available at: https://v5.reactrouter.com/web/guides/quick-start.

www.tutorialspoint.com. (n.d.). *Basics of React.js Routing*. [online] Available at: https://www.tutorialspoint.com/basics-of-react-js-routing.

Facebook (2022). *React – A JavaScript library for building user interfaces*. [online] Reactjs.org. Available at: https://reactjs.org/.

Ronacher, A. (2010). *Welcome to Flask — Flask Documentation (2.2.x)*. [online] flask.palletsprojects.com. Available at: https://flask.palletsprojects.com/en/2.2.x/

U.S. Department of Agriculture (2019). *FoodData Central*. [online] Usda.gov. Available at: https://fdc.nal.usda.gov/.

The PostgreSQL Global Development Group (2019). *PostgreSQL: The world's most advanced open source database*. [online] Postgresql.org. Available at: https://www.postgresql.org/.