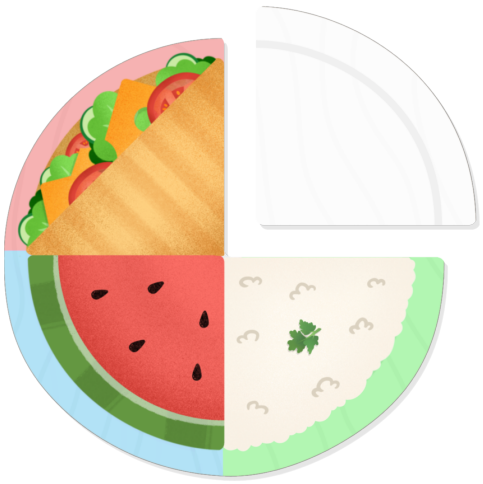
SMU
SINGAPORE MANAGEMENT UNIVERSITY



MEAL CALORIE TRACKER

CS461 - MOBILE & PERVASIVE COMPUTING AND APPLICATIONS

FINAL REPORT

GROUP 6		
NAME	**ID**	**EMAIL**
Aeole Jasmine Federico Montero	01395179	amontero.2019@scis.smu.edu.sg
Koh Jie Yin	01418737	jieyin.koh.2020@scis.smu.edu.sg
Lim Zhi Wei	01404534	zhiwei.lim.2020@scis.smu.edu.sg
Nicholas Wong Kai Wei	01375901	kwwong.2020@scis.smu.edu.sg
Yip Yi Mun	01420067	yimun.yip.2020@scis.smu.edu.sg

Executive Summary

In the realm of meal tracking applications, current solutions often lack seamless and personalized user experience. To bridge this gap, our innovative **MEAL CALORIE TRACKER** application aims to revolutionize this space through advanced functionalities. Featuring CameraX integration, intuitive food image recognition, QR Code Scanning, and personalized recommendations, our application not only enhances accuracy and streamlines meal logging but also ensures a user-friendly and tailored approach to individual dietary needs.

The report unfolds in a coherent sequence, commencing with the introduction, where we delve into the motivation behind our application's development. Subsequent sections provide a detailed explanation of the application's system design, implementation, and thorough testing procedures. We conclude by acknowledging challenges faced during development and presenting a roadmap for future enhancements.

Our application transcends the role of a simple calorie tracker, presenting itself as a holistic solution that transforms how users manage their dietary habits through enhanced engagement and personalization.

1. Introduction

MEAL CALORIE TRACKER is a specialized mobile application designed for meticulous nutritional tracking of user-consumed foods. Going beyond conventional tracking application functionalities, our application seamlessly integrates CameraX, food image recognition, and QR Code Scanning, allowing users to effortlessly capture and log meals with unparalleled accuracy.

Noteworthy features include personalized daily calorie intake recommendations based on individual profiles. The comprehensive dashboard not only records past meals but also provides users with insightful analytics, offering detailed view into their dietary habits. This feature empowers users to track nutritional intake over time and make informed comparisons with recommended calorie levels.

1.1. Background & Motivation

In the domain of nutritional tracking, accurately calculating the calorie content of meals has proven to be a challenging and time-consuming task. The prevalent method of manual logging, while widely used, not only requires a substantial time commitment but also frequently yields imprecise data.

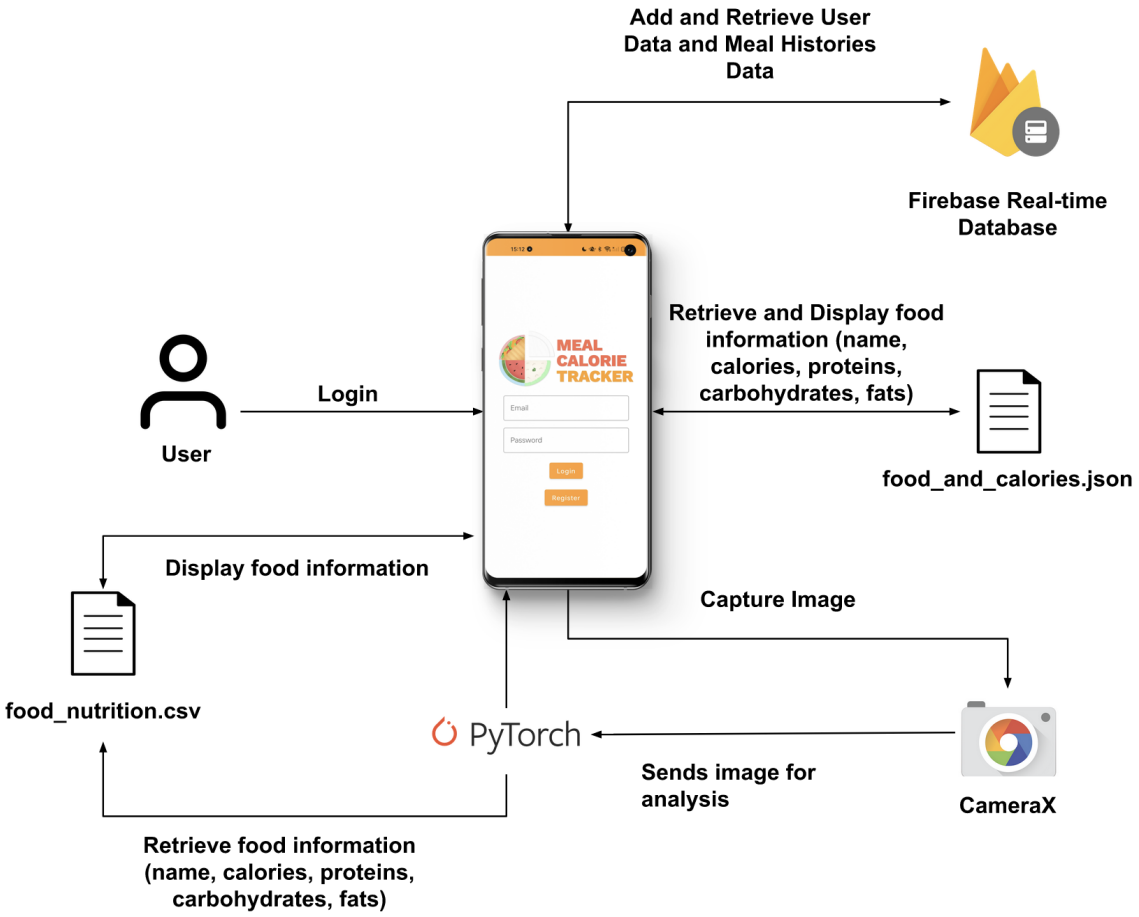
This challenge is further compounded by the diverse dietary goals individuals pursue, spanning from weight management and muscle gain to specific dietary restrictions. As a result, a considerable number of individuals opt to forgo the meticulous task of tracking their nutritional intake, which, in turn, contributes to challenges in maintaining balanced and healthy meal portions.

1.2. Objective

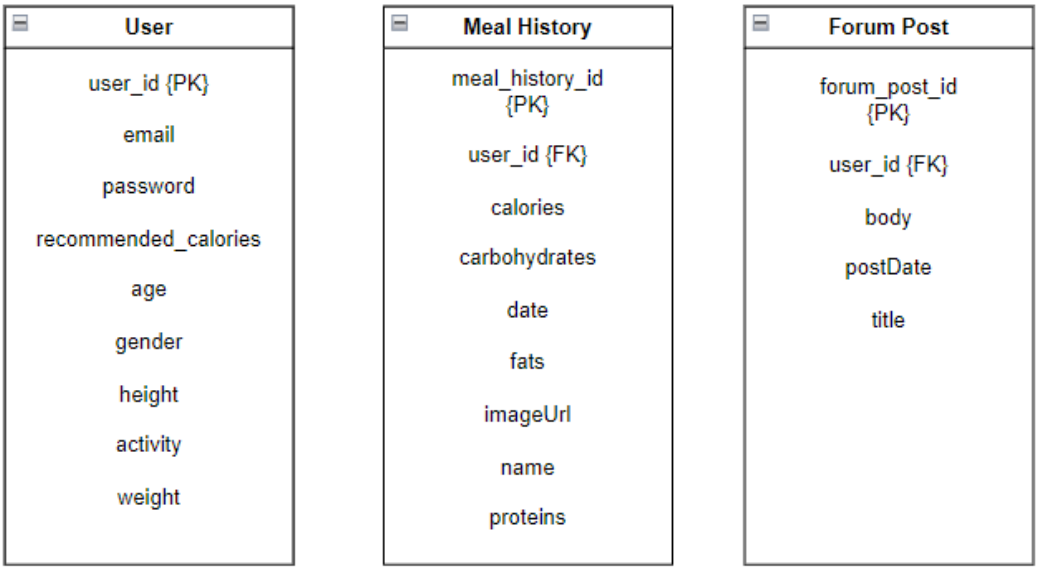
The genesis of **MEAL CALORIE TRACKER** stems from the identified gap in efficient and user-friendly meal calorie tracking. We aim to empower individuals by addressing the challenges associated with traditional tracking methods. Through the development of our app, we strive to offer a seamless and precise tool, transforming the often-burdensome task of calorie counting into an accessible and personalized experience for users.

Our overarching goal is to provide a user-friendly solution that not only enhances accuracy but also encourages sustained engagement in the pursuit of healthier dietary habits.

2. System Design Overview



2.1 Database Schema



3. Implementation Details

Our application consists of the following features:

1. User Profile
2. CameraX Integration
3. Food Image Recognition
4. QR Code Scanning
5. Food Search
6. Manual Input
7. User Statistics
8. Meal History and Filtering
9. Personalized Recommendation
10. Discussion Forum

Our food datasets, namely **`food_and_calories.json`** and **`food_nutrition.csv`** are stored as text files within the application's resources. Specifically, they reside in the asset folder of the Android project.

The decision to opt for local storage over a remote database like Firebase was driven by the goal of minimizing network latency, especially during the loading of the food dataset within the application. By keeping the datasets locally, we ensure quicker access to essential information without relying heavily on network communication.

Over the next few sections, we will be explaining each of the features in detail.

3.1 User Profile

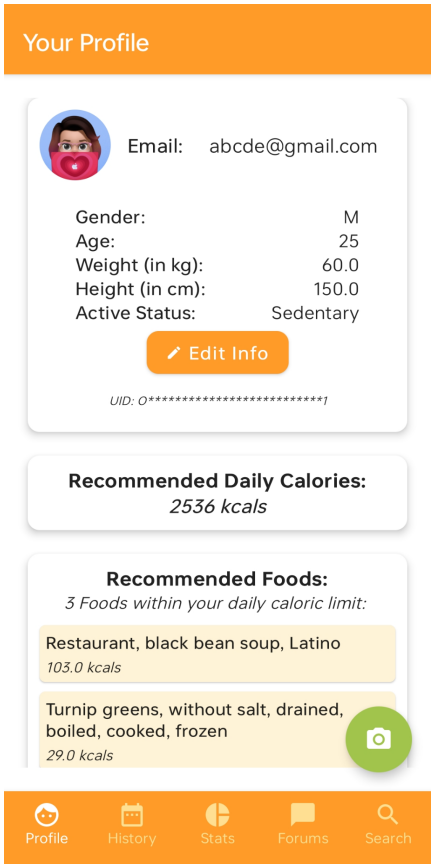


Fig 1: User Profile Page

The introduction of the user profile feature was designed to support the registration and login process in our application. In addition, it serves as a crucial asset for several other functionalities that necessitate access to user sessions and data. To achieve this, we collect essential user credentials, specifically 'Email' and 'Password,' which are securely stored in **Google Realtime Firebase**.

We used **FirebaseUtil (Firebase Utility)** in our application for user authentication services, to ensure the validity of the user's email and password input. Upon successful registration or login, the user's data is stored in **SessionManager**, allowing for effortless data retrieval and efficient session management.

Upon successful authentication, the user will receive a success welcome toast message and be redirected to their profile page. In case of authentication failure, an error toast message will be displayed instead.

3.2 CameraX Integration

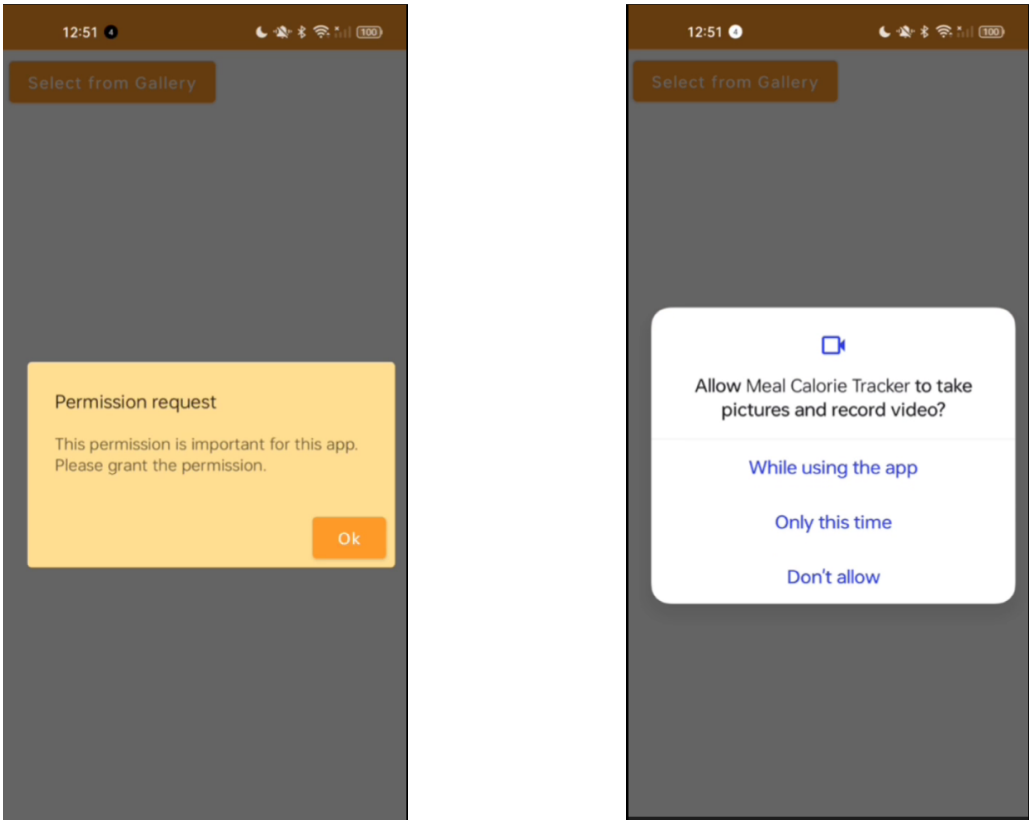


Fig 2: CameraX Integration requires the user to grant Camera access

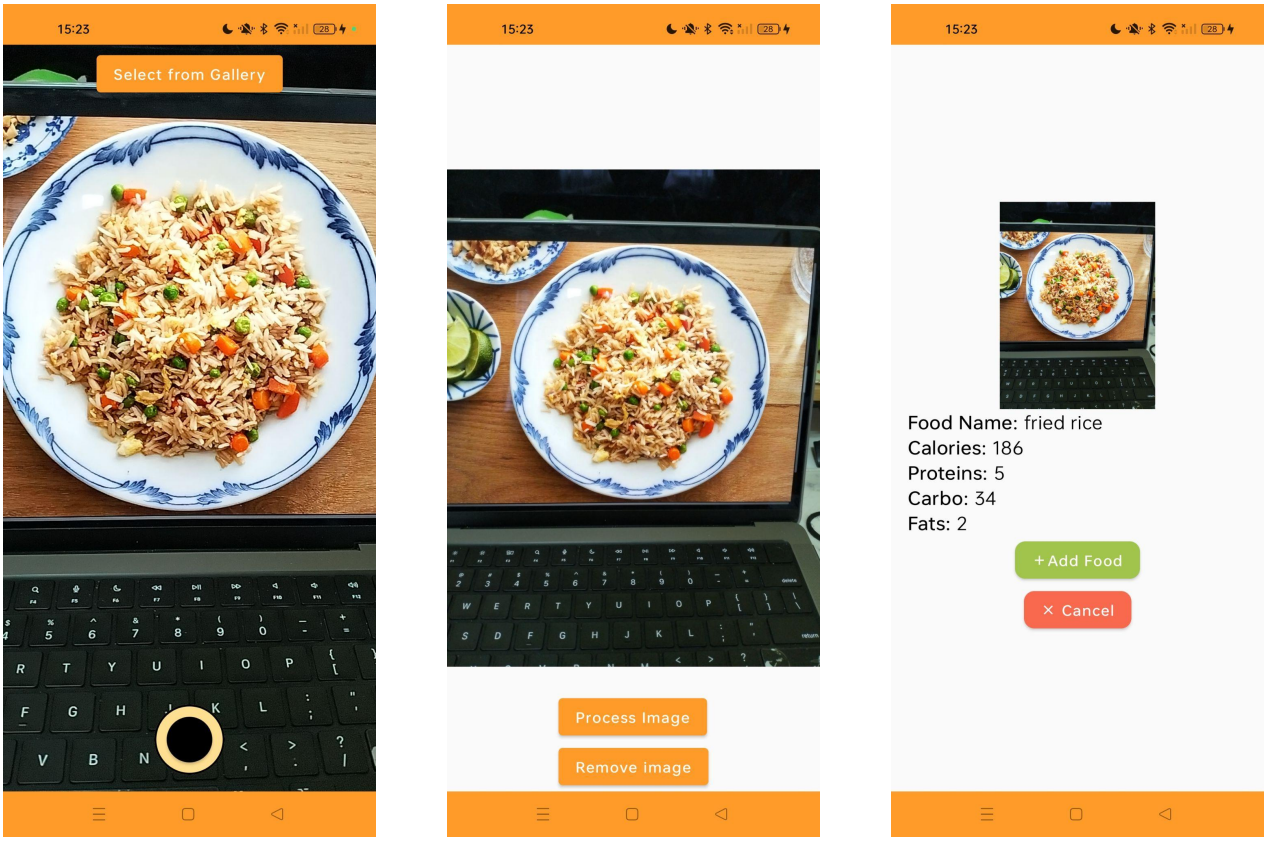
CameraX Integration forms the foundation of the Food Image Recognition feature in our application. This integration enables users to utilize their device's camera to capture images of their meals or of QR codes with food information in it effortlessly.

We have implemented this feature in Kotlin, leveraging the **CameraX API** of Android Studio, known for its backward compatibility and ease of use. CameraX simplifies the complex camera functionalities into an easy-to-use and consistent API that works across most Android devices.

The application requests camera permissions to access the device's camera hardware. Upon permission grant, the application sets up a live camera preview for the user. We use the permissions API to ensure a seamless experience that adheres to Android's best practices and user privacy guidelines.

Through the page, users interact with a custom camera interface that includes a preview and a capture button. After capturing the image, it is stored locally, and the URI is passed to the next page for further processing.

3.3 Food Image Recognition



1. Take photo / Select from Gallery
2. Confirm image to process / Choose different image
3. Discard or add the food to their meal history

Fig 3: Progression of taking image of food to automatic detection

Machine Learning Algorithms

The Food Image Recognition feature enables users to automatically identify food items from images. This feature uses advanced machine learning algorithms powered by PyTorch and the Vision Transformer (ViT) architecture to analyze food images and predict their categories.

We utilize the **`google/vit-base-patch16-224`** model pre-trained on the ImageNet-21k dataset for its robustness and efficiency. This model has been further fine-tuned on the Food101 dataset to specialize in food classification tasks.

The Food101 dataset, which includes 101,000 images across 101 food categories, serves as our training and validation dataset. We preprocess images using the **`ViTImageProcessor`** to conform to the model's input requirements, such as size normalization and pixel value scaling.

Our model is trained using a custom PyTorch training loop. We employ a cross-entropy loss function suitable for multi-class classification and the Adam optimizer with a learning rate of '1e-3'. The model trains for '5' epochs with a batch size of '8', ensuring adequate learning while managing computational resources.

```
num_epochs = 5
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-3)
train_dataloader = DataLoader(dataset, shuffle=True, batch_size=8)

model.train()
model.to(device)
bar = tqdm(range(num_epochs * len(train_dataloader)))
for epoch in range(num_epochs):
    for batch in train_dataloader:
        batch_x = batch['image'].to(device)
        batch_y = batch['label'].to(device)

        gc.collect()
        torch.cuda.empty_cache()

        batch_outputs = model(batch_x)
        batch_y_hat = batch_outputs.logits
        batch_loss = loss_fn(batch_y_hat, batch_y)
        batch_loss.backward()

        optimizer.step()
        optimizer.zero_grad()
        bar.update(1)
```

For inference, we use a TorchScript-traced version of our model, which offers portability and optimization for production environments. We evaluate the model's performance on the validation split of the Food101 dataset, emphasizing the model's accuracy in classifying a diverse range of food images.

Our evaluation metric is accuracy, which is the proportion of correctly predicted images over the total number of images. In our validation tests, the model achieved an accuracy of **87.72%**, demonstrating its effectiveness in food image recognition tasks.

Integration into Mobile App

The Food Image Processing page is the pivotal class that bridges the camera functionality with the food image recognition. It initializes with an image URI delivered through an intent from the Camera Capture page, embodying a seamless transition from image capture to analysis.

Upon receiving the image URI, the page invokes the image processing routine. For integration into the mobile application, the trained PyTorch model is converted into a TorchScript format with a **`.pt`** file extension, known as **`model.pt`**. This conversion process optimizes the model for mobile environments, ensuring compatibility and efficiency when running on a wide array of Android devices. TorchScript provides a serialized representation of the model that can be executed independently of the Python runtime, which is essential for deployment within the app's infrastructure.

The **`model.pt`** file is then incorporated into the mobile application's backend, specifically within the Food Image Processing page. When the user captures an image, it is passed to this page through an intent as an image URI. The page utilizes Glide to convert the URI to a Bitmap, which is then preprocessed and fed into the TorchScript-traced model for inference. The inference results, indicating the type of food and its nutritional information, are displayed to the user on the page.

This process is asynchronous and robust, utilizing Glide to fetch the image as a Bitmap. This bitmap is then used for food identification: the image is passed through the pre-trained and fine-tuned PyTorch model.

The model, specifically tailored for food classification, predicts the food item depicted in the image. The predicted item is then used to fetch detailed nutritional information from a local asset, **`food_and_calories.json`**. This local JSON file stores an array of food items along with their corresponding nutritional data, which includes calories, proteins, carbohydrates, and fats content. Such an approach ensures that the data retrieval is swift and does not depend on network availability, making the application more responsive.

Once the food item is recognized, and its nutritional information is obtained, the mobile application updates to display this information to the user. The interface provides two distinct paths: users can add the recognized food item to their meal history or opt to cancel. This can be useful for the cases where the food image recognition is inaccurate.

The former action triggers an integration with Firebase, wherein the food item, along with its nutritional data and image URI, is stored in the Firebase Realtime Database and the image is uploaded to Firebase Storage. This integration ties the data to the authenticated user, ensuring a personalized and secure user experience.

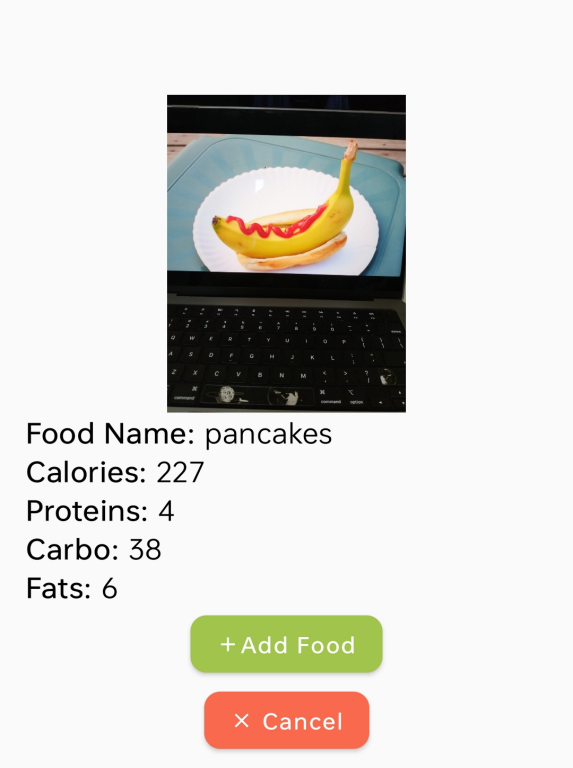


Fig 4: For food images that have been wrongly identified, users can discard the image and retry capturing another food image

3.4 QR Code Scanning

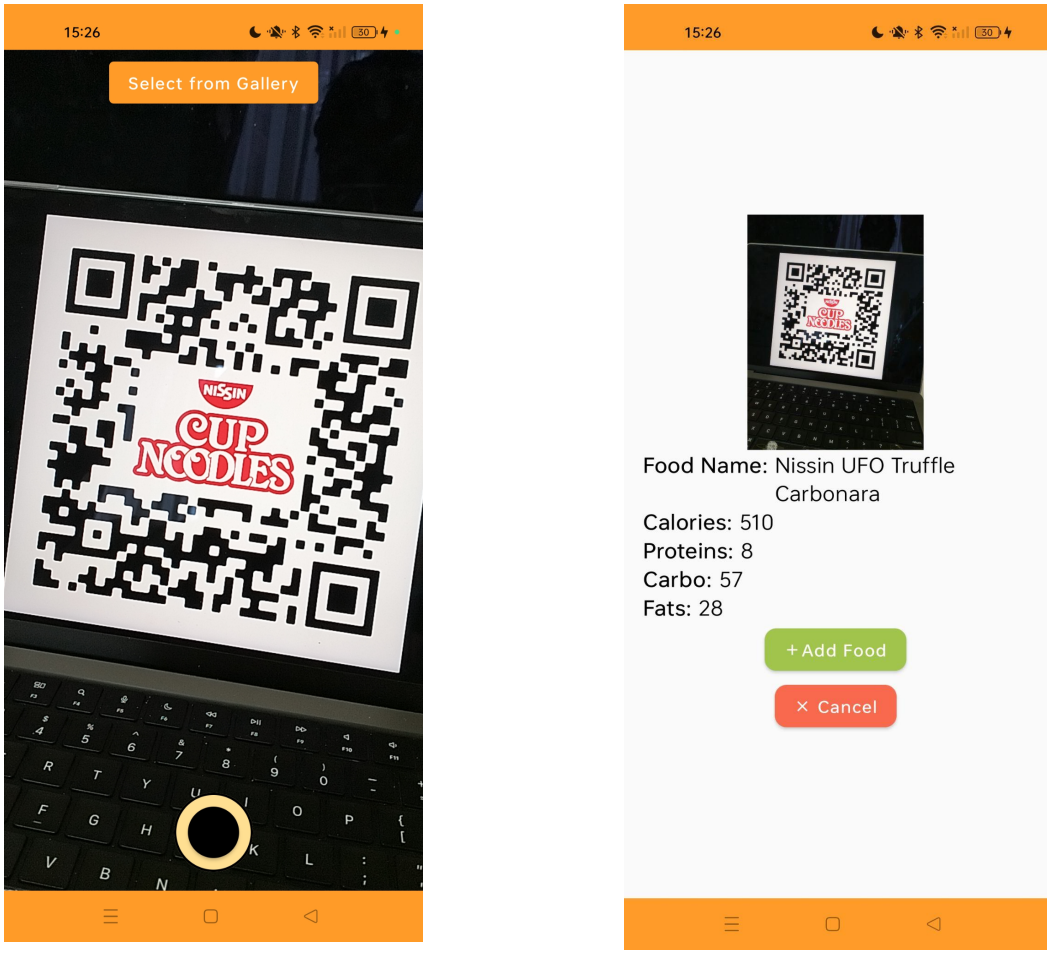


Fig 5: Progression of taking image of QR code to automatic detection of food item

The QR Code Scanning feature allows the user to scan QR codes that contain food information in this format:

`[Food Name], [Calorie Count], [Protein Count], [Carbohydrates Count], [Fat Count]`

After capturing an image from the Camera, the URI is passed to the Food Image Processing page, where it detects whether a QR code is present in the image. If a code is detected, it checks whether it follows the above format. If there is no code detected or the content of the code does not match the above format, it moves onto the Food Image Recognition feature detailed previously.

3.5 Food Search

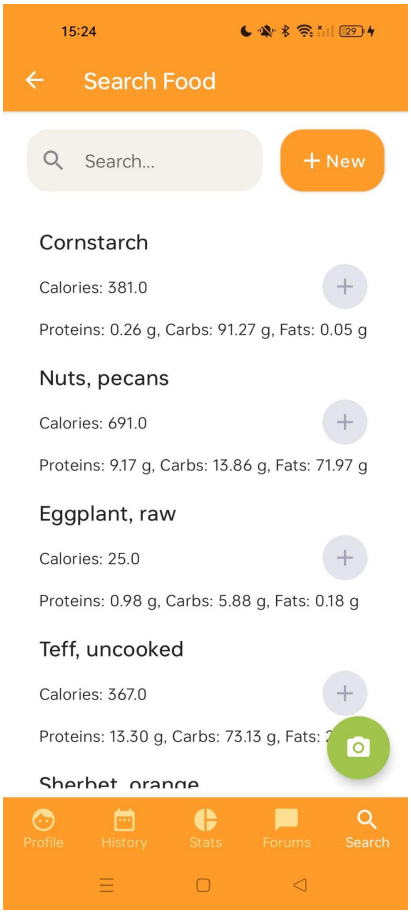


Fig 6: Food Search Page

The Food Search feature allows users to find specific foods from a list, addressing the limitations of the Food Image Recognition and QR Code Scanning features. These limitations include foods not covered in the training dataset or those challenging for automated identification.

The feature's screen comprises a search bar and a list of food items, each accompanied by nutritional information (calories, proteins, carbohydrates, and fats) and an add button.

The search bar dynamically filters the list based on the user's input, employing a case-insensitive approach for accurate results regardless of the input format.

The list of food items is sourced from **`food_nutrition.csv`**, initially stored in the assets folder. During initialization, the application checks for the existence of this dataset in the application's data folder (internal storage).

If present, it reads the data locally. If not, it reads the CSV file from the asset folder, populates the food list, and simultaneously writes the CSV data to the app's data folder for subsequent use. It's important to note that **`food_nutrition.csv`** is a dataset obtained from Kaggle, providing nutritional values for various common foods and products.

This strategic approach optimizes performance by enabling quick access to food information while also accommodating potential updates to the dataset. It ensures a robust and flexible Food Search feature for users.

3.6 Manual Input

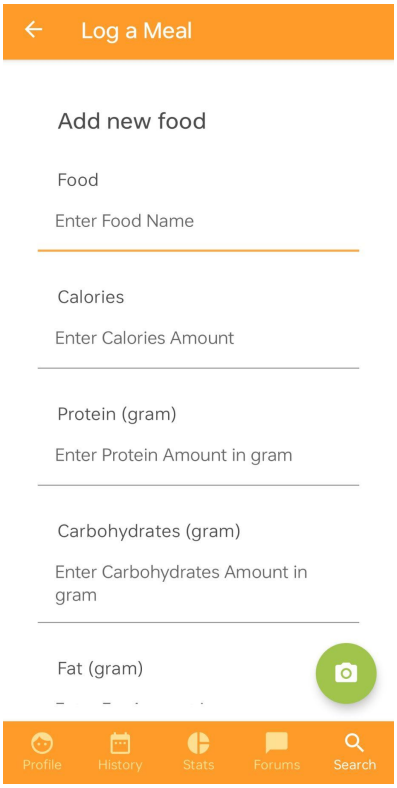


Fig 5: Manual Input

The Manual Input feature, strategically crafted to complement the Food Search, Food Image Recognition, and QR Code Scanning functionalities, empowers users to manually input food items along with their corresponding nutritional information. This capability proves invaluable in scenarios where automated recognition falls short or when the desired food is absent from the existing list.

The user-friendly interface of this feature incorporates input fields for Food Name, Calories, Protein, Carbohydrates, and Fats.

It's worth noting that, with the exception of the Food Name, these input fields exclusively accept numerical values, allowing for precise data entry with decimal points. Protein, carbohydrates, and fats should be specified in grams.

Upon entering the data, the system conducts a validation check to determine if the food already exists in the application's dataset, sourced from **`food_nutrition.csv`**. If a match is found, a prompt toast message promptly notifies the user of the food's presence in the database. Conversely, if the food is not part of the dataset, the application seamlessly adds the new food and its corresponding details to the CSV file.

With a successful addition, a confirming toast message appears, and users are seamlessly redirected to the Food Search feature's screen, where they can search for the newly created food.

It's essential to highlight that the Manual Input feature seamlessly interacts with the same nutritional dataset utilized by the Food Search functionality, ensuring data consistency across various aspects of the application. This dataset, initially stored in the assets folder, is dynamically checked during initialization. If it exists in the application's data folder (internal storage), the application reads the data locally.

Otherwise, it reads the CSV file from the assets folder, populates the food list, and simultaneously writes the CSV data to the app's data folder for subsequent use. This strategic approach optimizes user experience and flexibility while maintaining a unified dataset across functionalities, offering a cohesive and efficient solution for manual food input within the application.

3.7 User Statistics

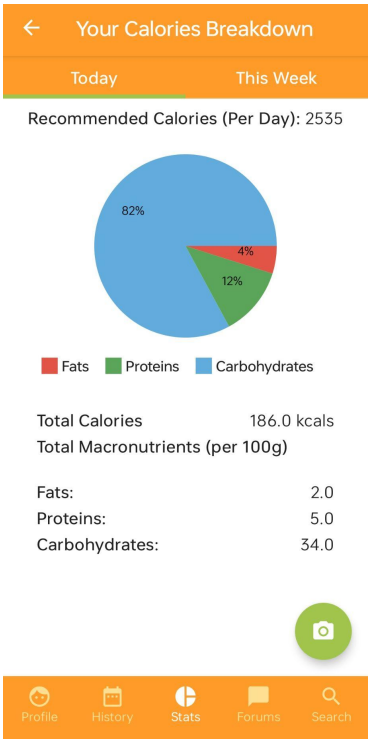


Fig 7: User Statistics for Today

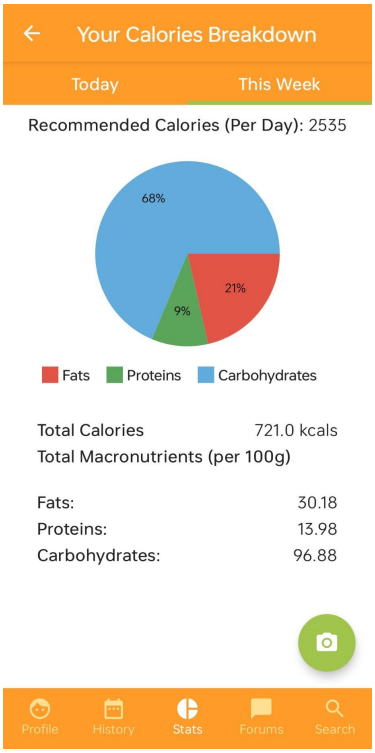


Fig 8: User Statistics for This Week

Our mobile app's User Statistics feature is a comprehensive dashboard that provides users with insights into their dietary habits. Through an interactive and informative interface, users can track their nutritional intake over time, compare it with recommended values, and make informed decisions about their eating habits.

The feature dynamically fetches the user's recommended daily calorie intake from Firebase, enabling a personalized experience. Users can immediately gauge how their current intake stacks up against their nutritional goals.

With the integration of DateFilterTabs, users can filter their nutritional data based on date ranges, such as **Today** or **This Week**. This flexibility allows for granular tracking of dietary patterns over different periods.

The statistics page boasts a custom PieChart composable that visually breaks down the percentage of macronutrients consumed. This visual aid is instrumental in helping users quickly understand their nutritional balance at a glance.

For a more detailed analysis, MealStatsContent presents the total macronutrients consumed and compares the calorie intake against the recommended amount. This section aims to educate users on their nutritional intake and guide them towards healthier choices.

To encourage consistent interaction, it will display “No records found” to prompt users to log their meals whenever there's an absence of data. This gentle reminder is key to motivating users to maintain a regular logging habit, thereby maximizing the benefits of the statistics feature.

3.8 Meal History and Filtering

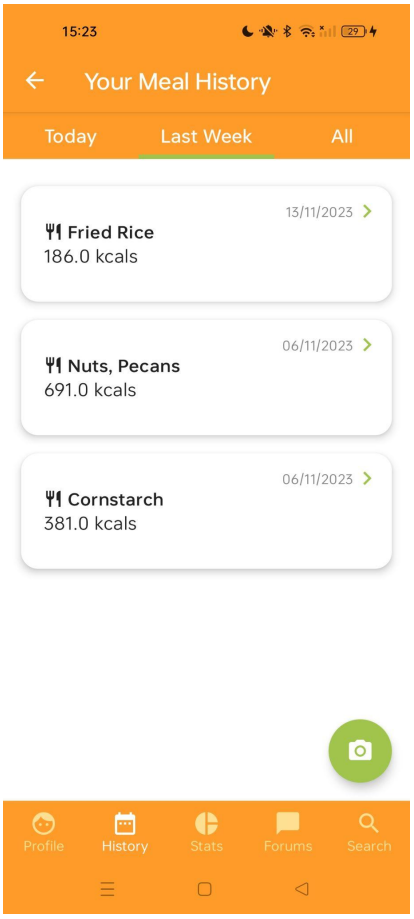


Fig 9: Meal History List for Last Week

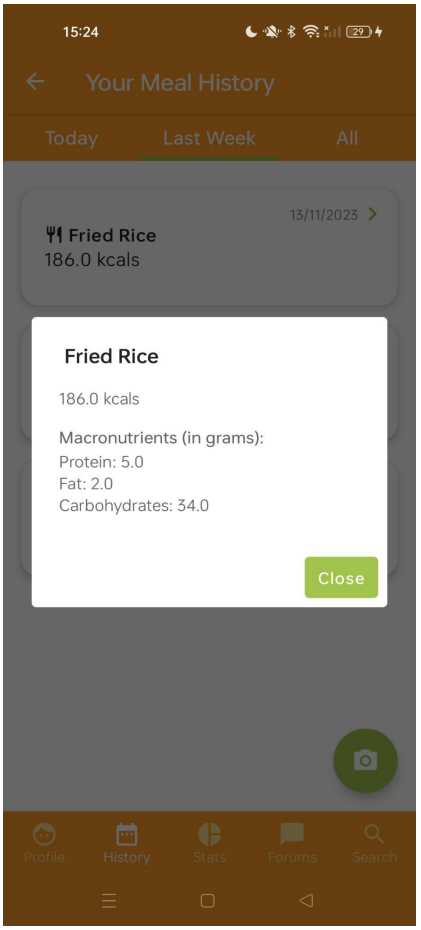


Fig 10: After clicking on a Meal History entry, additional information can be seen

The meal history feature allows users to view their existing meal entries. Upon entering the page, we use firebase authentication to get the current user's ID and display all meal entries in the firebase database that corresponds to the ID. Each meal entry will be displayed in a box showing the entry's name, date and caloric information. Users can click on each box to view all additional macronutrient information like the amount of protein, fat and carbohydrates in the meal.

It is done in this manner to declutter the main meal history page, allowing users to quickly and easily identify a specific meal they might be looking for and key information like its caloric content, while still giving them the option to view additional information. Meal entries are sorted by date descending so that users can view their latest meals first.

Users can also sort their meal history page to display only the meals from today, all of last week, or all their existing meal histories. This is easily done through three tabs at the top of the page. This feature gives users the flexibility and ease to view relevant information specific to any use case they might have.

3.9 Personalized Recommendations

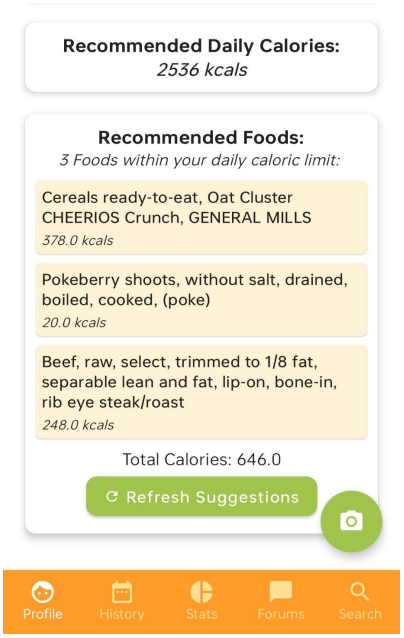


Fig 11: Personalized Recommended Daily Calorie Intake and Foods

The personalized recommendation feature is designed to provide users with:

1) a recommended daily calorie intake to help users know their calorie limits for the day

2) customized food suggestions on what they can eat for the day, taking into account that these foods' calories still fall within their recommended daily calorie intake.

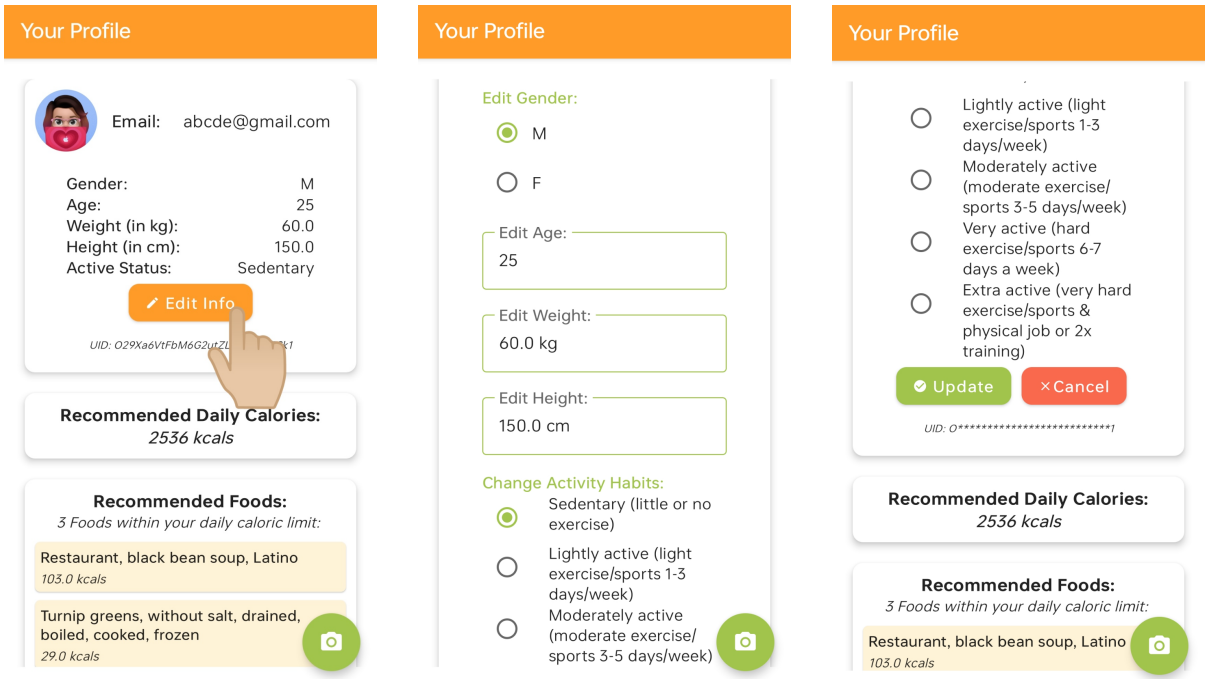


Fig 12. Progression of step for users to update their user information

Users have the option to update their information (such as Age, Weight, Height, Activity Habits) within their profile, to ensure their profiles remain accurate and up-to-date. When these details are updated, our application calculates a new daily calorie intake recommendation based on the updated information.

Additionally, the application generates three food recommendations from the list of food items contained in **`food_nutrition.csv`** file, also ensuring that it aligns with the specified calorie intake limit. If a user happens to dislike any of these suggestions, they can easily refresh the recommendations by clicking a button, and our application will generate another set of three recommendations randomly.

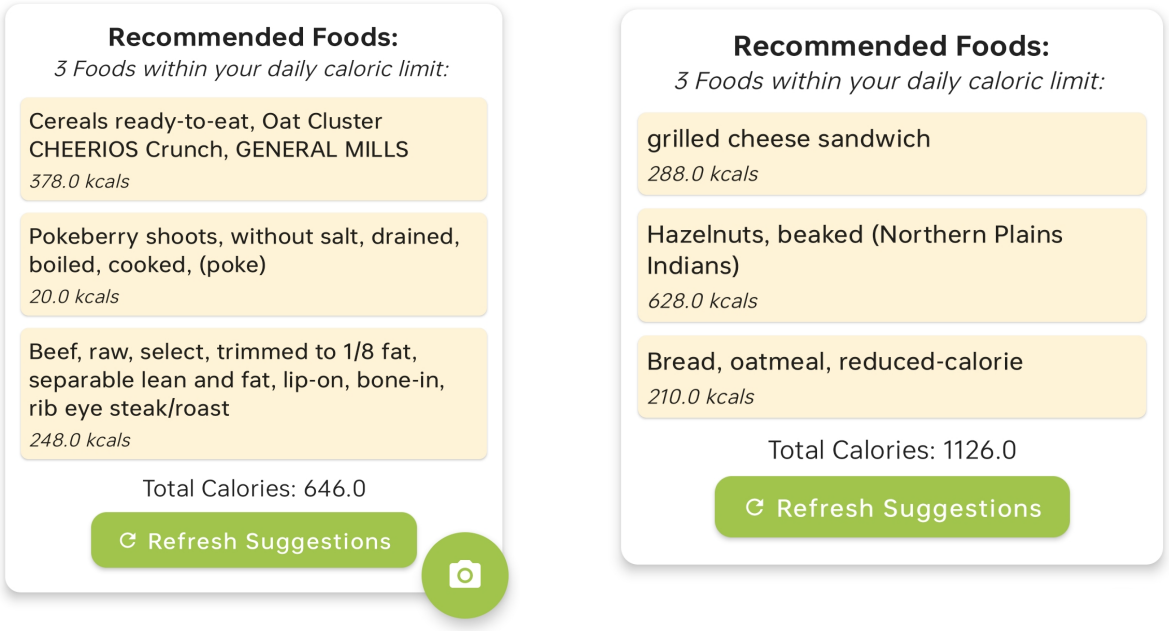


Fig 13. Users can generate new Recommended Food suggestions by clicking "Refresh Suggestions"

3.10 Discussion Forums

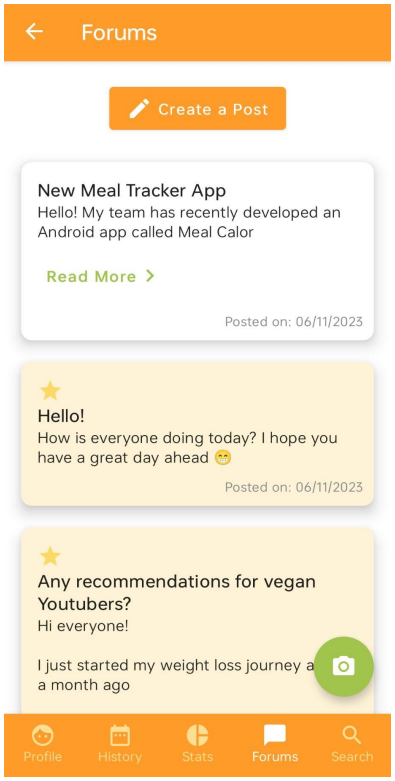


Fig 14. Discussions Forum Page

Healthy living and eating is a journey best done with others.

The discussion forum feature allows users to share any content in the form of forum posts with other users of the app. For example, users can share about their weight loss journeys, insights about certain diets, and delicious recipes that they created, to name a few.

Upon clicking the forums tab, users will be presented with a page of posts, each with a bolded title, a post body and date posted. The list is sorted by date descending, allowing users to view the latest posts.

If the post body is too long, only a part of the content is shown, and users can click on “read more” to read the full post. This makes the page faster to scroll to a post that the user is interested in.

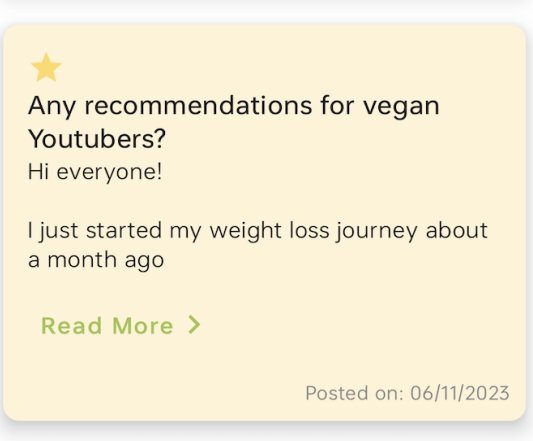


Fig 15. Long Forum Post on first load of page

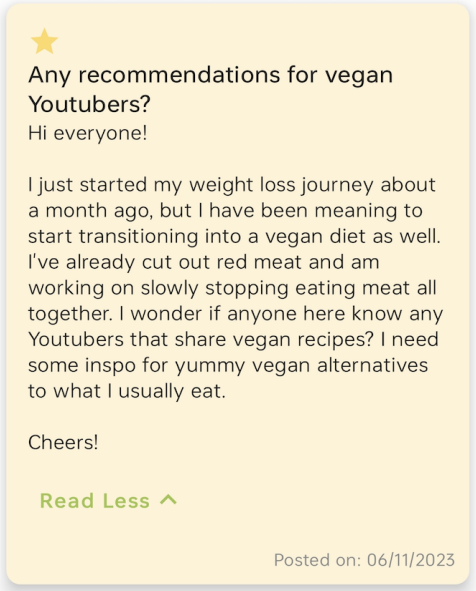
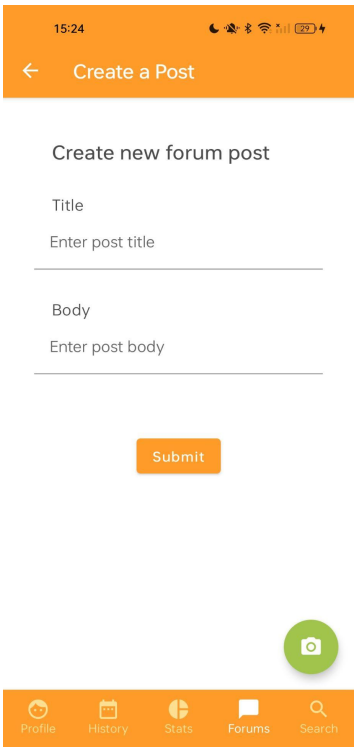


Fig 15. Long Forum Post on clicking “Read More”

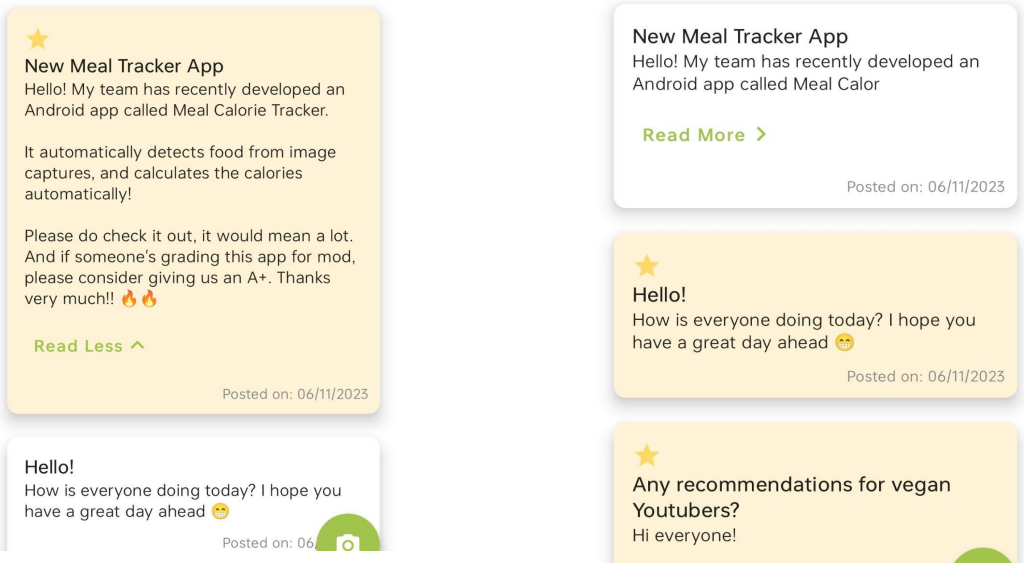


Clicking on "**Create a Post**" at the top of the page brings users to a new page where they can input a title and body, allowing them to create their very own post.

Upon submitting, the firebase database populates the posts collection with a new document, including the post title, body, date and userID.

Fig 16. Creating a Forum Post

The userID is stored and used to distinguish between the current user's posts and other users' posts. Any post made by the user has a different background color from posts made by other users. This allows users to more easily view what posts interest them.



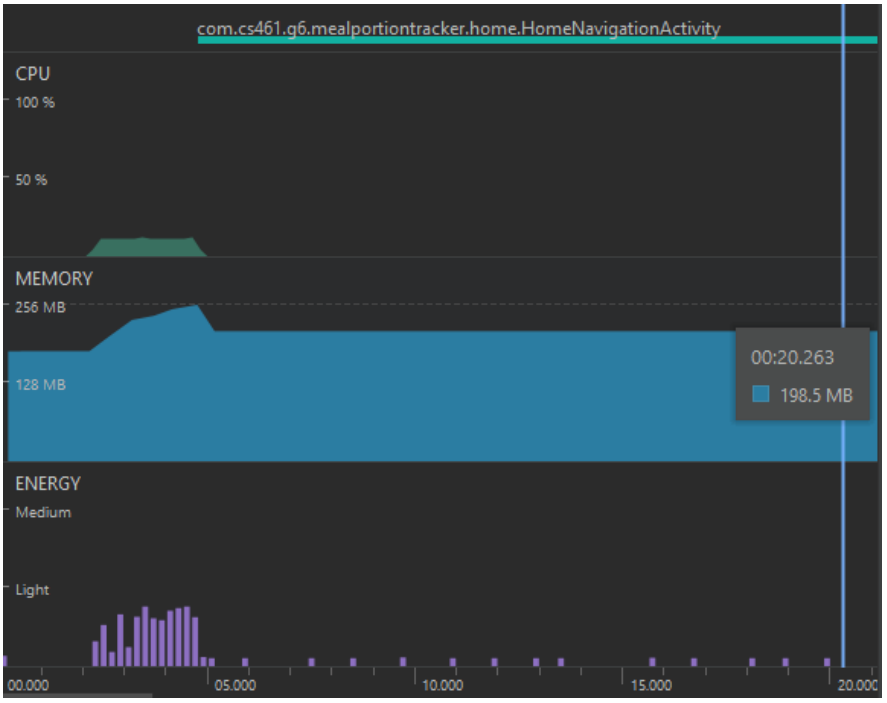
On abcde@gmail.com account

On bg@gmail.com account

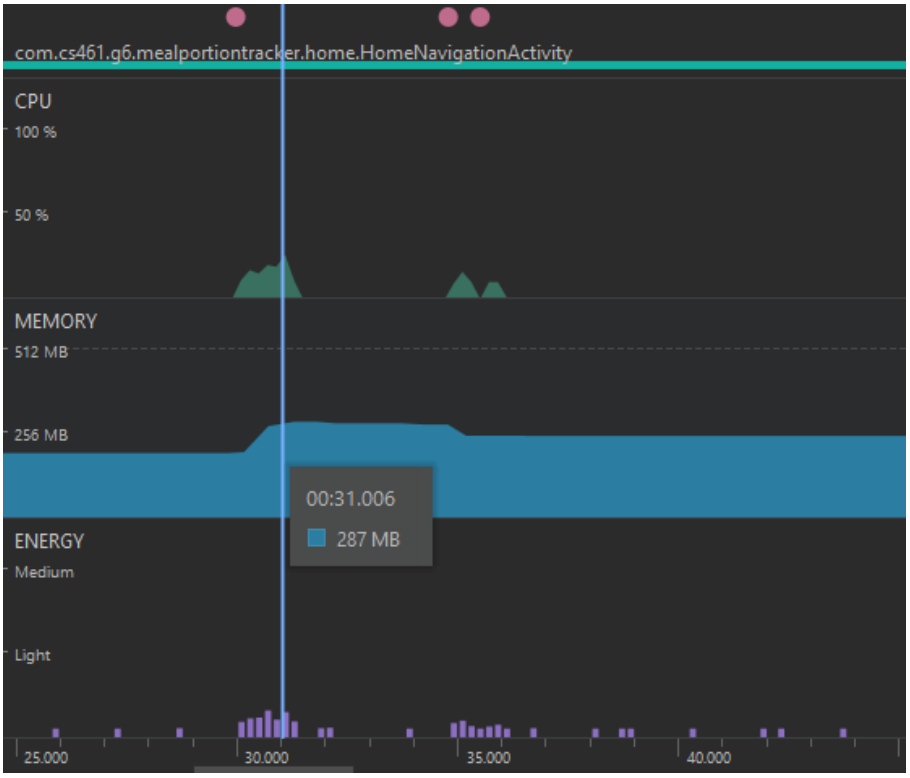
Fig 18. Users can identify their own forum posts as it will be marked by a Star symbol and be highlighted in yellow.

4. Application Testing & Comprehensive Analyses

We used Android Profiler while testing our application to record the application's resource utilization metrics, allowing us to understand its performance in real time.

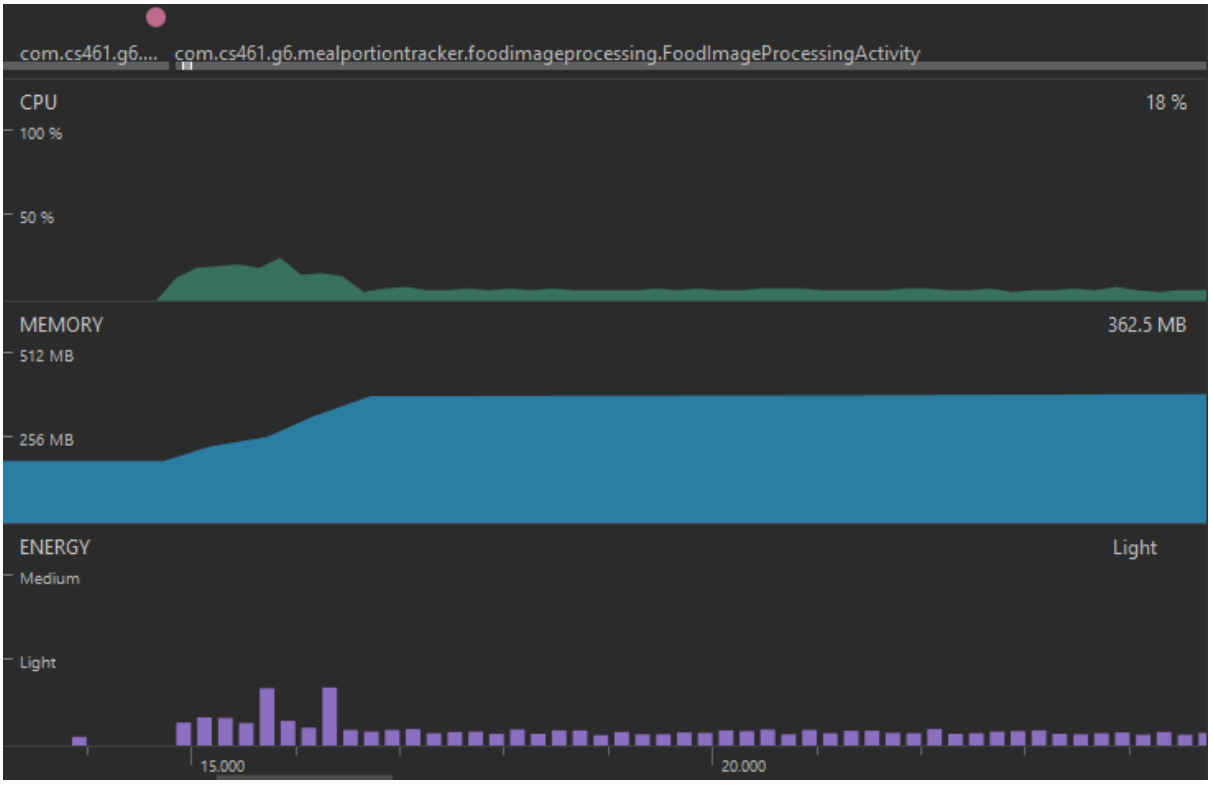


Above is the result of the profiler. On the left most of the graphs, when the app has just started running, there is a slight spike in metrics, but when left on the profile page, they stabilize into a baseline state of 0% CPU utilization, 200MB of memory consumption, and none or light energy usage.



Clicking on other activities like Meal History, Statistics, Forums, and Food Search, as well as their own tabs like Statistic's Today and This Week, causes a spike in memory to around 280-300MB, CPU utilization of around 10-20% and light energy consumption.

When left on these activities, they return a baseline of 230-250MB, 0% CPU utilization and none to light energy usage. From these results, we can see that most of the app's activities except the one later mentioned all record light battery consumption and low resource utilization. Hence, the application is unlikely to experience performance issues.



The only deviation from the relatively equal utilization records of the other activities is the food image processing feature. The pink dot is the action where a user would click **"Process Image"** after snapping a picture of their meal. CPU utilization spikes to around 25%, memory climbs to 360MB and energy consumption becomes constantly light instead of intermittently light, but still remains light overall. Even though higher than the rest, we can see that this feature is still very light on the system resource utilization, allowing our app to run smoothly.

5. Conclusion

Acknowledging the pressing concern of unhealthy eating habits and their detrimental impact on personal well-being, we emphasize the urgency of taking action. With this in mind, the team has committed to creating an inventive mobile application designed to tackle the calorie intake challenge, named **MEAL CALORIE TRACKER**. This user-friendly, precise, and individualized solution empowers individuals to effectively oversee and control their calorie consumption.

The **MEAL CALORIE TRACKER** app signifies a transformative shift in the way individuals approach their dietary choices. Our calories detection application is a robust tool equipped with a variety of features aimed at assisting users in monitoring and optimizing their nutritional intake. These features include a User Profile for personalization, CameraX Integration for capturing food images, Food Image Recognition, QR Code Scanning for food item identification, Food Search for fast information retrieval, and Manual Input for custom data entry. Our application also offers User Statistics for monitoring and data analytics, Meal History and Filtering for historical data management, Personalized Recommendation for tailored food suggestions, and a Discussion Forum for knowledge exchange and community interaction.

By delivering a seamless and personalized experience for calorie tracking, our application reimagines how people manage their calorie intake, making it effortless to maintain a balanced and health-conscious diet.

5.1. Challenges and Limitations

5.1.1 Limitations while Finding Suitable Dataset for Portion Detection

In the context of food portion processing, a major hurdle we faced was the absence of a suitable training dataset tailored for the specific task of portion detection. The most promising dataset we came across contained only a minimal variety of food items placed on standard plates, and unfortunately, we couldn't locate a dataset that encompassed dynamic portions such as smaller plates, bowls, and similar variations. This unfortunately hindered our ability to develop the intended model for this aspect of the project, and thus we had to change the scope of our project.

5.1.2 Challenges in Sourcing for Suitable Databases

One of the challenges when developing the food image processing feature, was the limited availability of comprehensive food databases that encompass a wide range of foods and their variations. Even when databases are available, it is incompatible with our food recognition feature, where it is unable to retrieve the corresponding food item from the databases.

This limitation had a marked impact on the precision of our food image processing capabilities. To address this obstacle, we had to devise alternative methods for food detection or data input.

5.1.3 Challenging to Implement New Technology and Packages:

The team decided to use the Jetpack Compose UI Toolkits due to its huge availability of samples and frameworks that could assist in building our application. However, since we were unfamiliar with this toolkit, we had to invest time in learning and familiarizing with these tools.

Moreover, the integration of CameraX into our application presented a significant learning curve. The existing codebase was predominantly written in Java, a language with which our team was not particularly familiar. Adapting to this code and converting it to Kotlin required a substantial investment of time, impacting our productivity. Additionally, the Machine Learning Algorithm process was a challenge as we faced issues like the dataset is too large to be pushed to Github.

5.2. Future Work

5.2.1 Implementation of Portion Detection

Our initial concept for the application involved automating the detection of food portions on a plate. Nevertheless, as previously mentioned, due to our restricted dataset, we are unable to realize this feature within our current timeframe. However, we remain committed to introducing the portion detection functionality in the future, as we are convinced that it offers a convenient means for our users to assess and control their food consumption with precision.

5.2.2 Improving Dataset Models

If feasible, our goal is to prioritize the acquisition of larger and more comprehensive datasets, with the ultimate aim of establishing our proprietary food database. The enhancement of our dataset size and specificity holds substantial potential for significant improvements in our application.

Presently, our team uses different datasets for food image recognition and personalized recommendations, partly due to the limitations outlined earlier. Our aspiration is to enhance and unify these datasets, creating a unified and extensive repository for the entire system to draw from. Additionally, we are looking to implement an iterative machine learning model capable of incorporating user-provided manual input to further refine food image recognition. This iterative approach will allow us to continuously enhance the model's accuracy and effectiveness.

5.2.3 Expand and Improve on our Current Features

The unfortunate reality of time constraints has curtailed our capacity to bring certain enhancements to fruition. Our primary goal is to extend the dimensions of our 'User Statistics' feature by integrating additional charts and statistical components. We understand that furnishing users solely with visual representations of macronutrient percentages may fall short of their expectations. Consequently, our aspiration is to introduce supplementary statistics, including predictive analytics, calorie comparisons, and insights on calories burned.

Regarding the 'Personalized Recommendation' feature, our team envisions refining the recommendation algorithm to tailor food suggestions more precisely. This refinement will be based on a user's meal history data, ultimately culminating in recommendations for specific food items and even dining establishments where these recommended foods can be enjoyed.

Finally, concerning our discussion forum, we have plans to enhance its functionality by introducing features such as the ability to 'like' and 'comment' on discussion posts. Additionally, we aim to create discussion groups where members can engage in deeper exchanges of ideas and insights on particular topics or subjects.

5.2.4 Integration with Health Promotion Board (HPB)

HPB has started projects such as Healthy365 and LumiHealth, to encourage Singaporeans to be more healthy by gamification and offering monetary rewards. While these apps have basic calorie tracking and Apple Watch activity tracking, it can be further improved by having an even more seamless experience, where users only have to take a photo of their food.

6. References

Glide v4 : Fast and efficient image loading for Android. (n.d.).
https://bumptech.github.io/glide/

google/vit-base-patch16-224-in21k · Hugging Face. (n.d.).

https://huggingface.co/google/vit-base-patch16-224-in21k

Vision Transformer (VIT). (n.d.).

https://huggingface.co/docs/transformers/model_doc/vit