EMETER

Technical Manual

Project Title
Demeter Food App

Supervisor:
Hyowon Lee

Date Completed:
20/04/2024

Group Members	Student ID
Kine Borromeo	19372763
Ronghui Lin	19354553

Table of Contents

1. Introduction

Demeter is an android mobile application that assists with meal preparation and grocery needs with improvements to user interaction through technologies like computer vision and barcode scanning. The app is designed to help you with keeping track of your pantry by keeping an ingredient list and it will suggest recipes to the user based on their dietary needs and ingredient requirements.
The application is built using Flutter Framework, which allows it to be compiled for multiple platforms in the future, programmed using Dart, a Object oriented programming language. It utilises technologies such as Firebase and supports ingredient input with item detection via computer vision as well as barcode recognition.

Our motivation for making this app was that there are many food recipe apps available on the market but none are like Demeter in the way it operates. Using our knowledge gained throughout our time in DCU in modules like (Search Technologies and the various Programming modules) as well as our individual research, we designed an app that could revolutionise the food application market.

In the link below, we have attached a link to our **updated blog** hosted on Notion detailing the many design changes and project milestones we have documented:
https://equal-bestseller-a50.notion.site/Demeter-Blog-a71931fabe5a40c692c55a84b317befb?pvs=4

A pdf version of our blog will be available in the Gitlab repository:
2024-ca400-borromk-2-linr-2 / docs / documentation /

1.2 Glossary

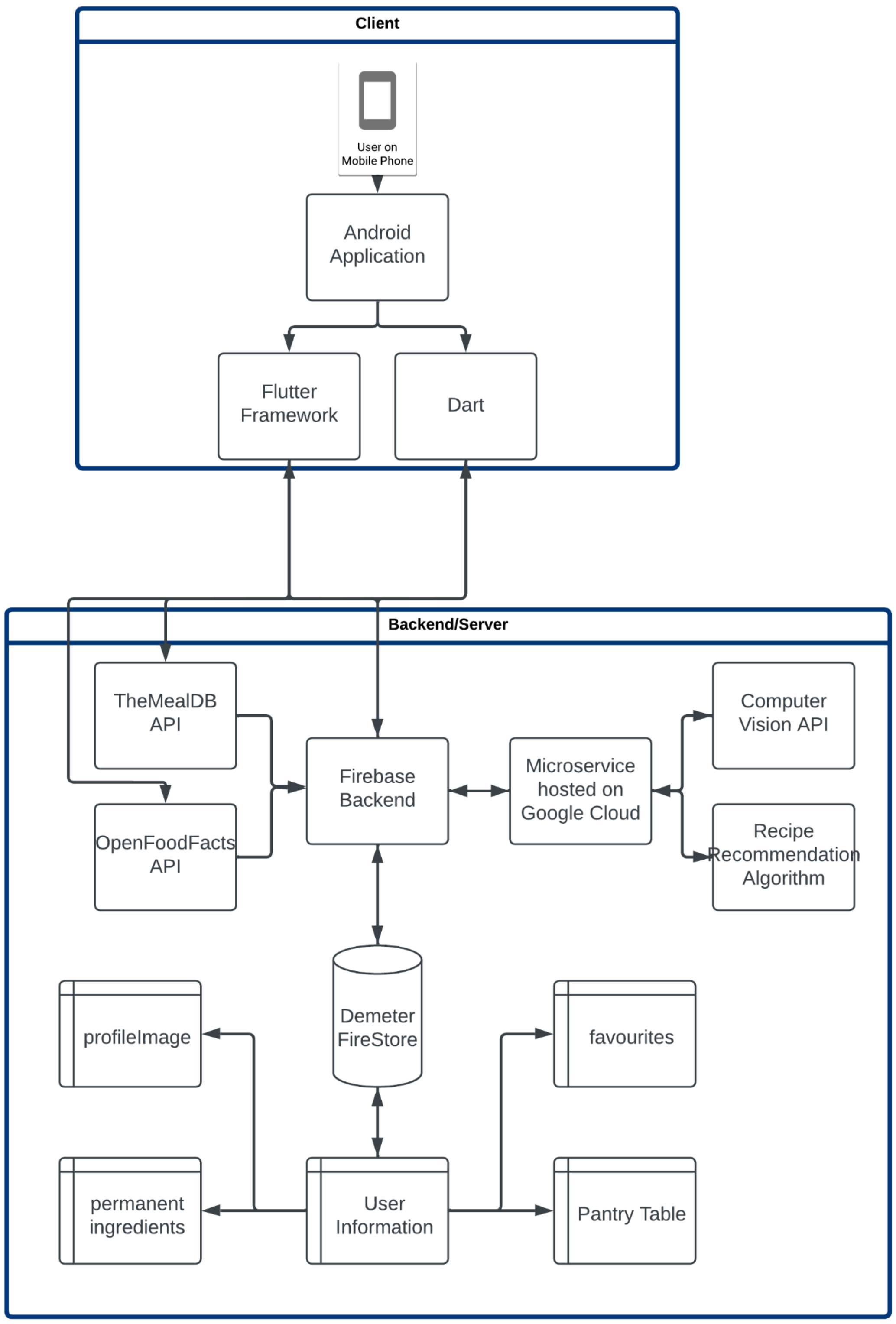
Term	**Definition**
UI	User Interface is the point of interaction between users and a computer, it consists of the visual and interactable elements that a user sees and can engage with.
API	Application Programming Interface is a set of rules and code that allows two

	pieces of software to talk to each other and exchange information or services.
JSON	JavaScript Object Notation, is a text based human readable data format that is used to store and exchange information based on key-value pairs and arrays.
Frontend	Client side of websites and applications, it is the part that you can directly see and interact with.
Backend	The backend is the part that you don’t see or directly interact with, it generally focuses on things like storing data, processing data and security.
Flutter	Free and open-source framework by Google, used for building cross platform applications from just a single code base.
Dart	Object-oriented programming language developed by Google, similar to Java and C.
Firebase	Firebase functions like a Backend-as-a-Service, providing a collection of prebuilt tools and services which streamline the process of developing backends.
Firestore	Firestore is a NoSQL document based database provided by Firebase, it stores data in a format similar to JSON and is fully managed, so there is no need to worry about maintaining servers and is designed for high availability.
Microservice	Microservices are small independent software services that work together to form a larger application, focusing on doing one thing exceptionally well. They can be updated without affecting the rest of the application.
Google Compute	Google Compute Engine is a component within Google Cloud Platform, allowing you to run virtual machines on google's global

	infrastructure.
TF-IDF	Term Frequency - Inverse Document Frequency is a numerical statistic used in natural language processing and information retrieval, it determines how important a word is in a document relative to a collection of documents.
Cosine similarity	This is a metric used to measure the similarity between 2 non-zero vectors, in our case it measures the similarity between each recipe based on the ingredients.

2. System Architecture

2.1 Architecture Diagram



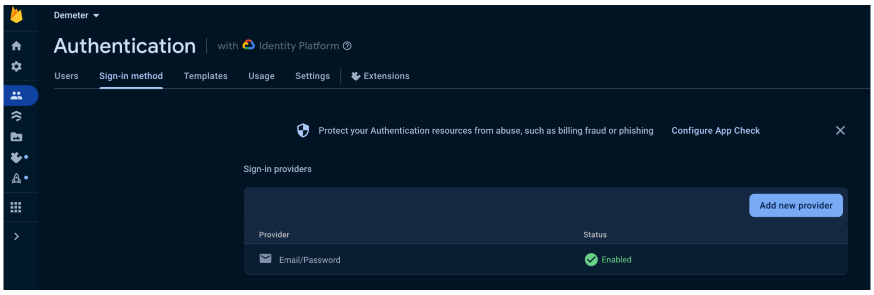
2.2 Implementation of Systems

2.2.1 Flutter/Dart

For our android application, we focused on using Flutter to create our application as it allows for a high degree of customisation and has the ability to compile to other operating systems such as iOS, Windows and Linux later. It is also written in Dart, an object oriented programming language developed by Google which has native integration with flutter which suited our use case.

2.2.2 Firebase / Cloud Firestore

Firebase is being used for our backend as it natively integrates well into our Flutter application allowing us to authenticate users and use their Firestore database. For user authentication, Firebase allows us to set tokens for each user to set their login states as well as the ability to allow users to change their emails and passwords. There are also checks added and Firebase implements their own email service, which allows us to bypass the step of setting up SMTP and allows us to automatically send emails about passwords or emails being reset. User data is also stored on Firebases integrated NoSQL database called Firestore, which we store Names, UUIDs, pantry ingredients, selected cuisines, permanent ingredients, profile pictures as well as favourited recipes. This allows us to serve the user with the necessary information that's required for our features to work in the application.



Protect your Authentication resources from abuse, such as billing fraud or phishing

Sign-in providers

Provider	Status
Email/Password	Enabled

Firebase - User Authentication

```
void _showRecipePopup(dynamic mealData) async{
 final String userId = FirebaseAuth.instance.currentUser!.uid;
 bool isFavourited = await _isFavourite(mealData['idMeal'], userId);
 showDialog(
```

showRecipePopup function (line 540 of dashboard_page.dart)

2.2.3 TheMealDB API

This open source API is what we use in order to incorporate the 303+ recipes into our application. We have paid for a premium version of the API for 3 Euros which allows us to have unrestricted access to the API. This is used across the app when users either search for recipes or request recipe details. It is one of the main APIs used across the app, it also allows us to string together filters in order to fetch recipes. Details such as ingredients, quantity, recipeIDs as well as thumbnails are saved to our firebase backend, especially when users favourite items or add ingredients to their pantry.

```
// 3. Check for Empty Favorites and Fetch From MealDB if Needed
if (formattedRecipes.isEmpty) {
 // Fetch random meals from MealDB
 const randomMealsUrl =
'https://www.themealdb.com/api/json/v2/9973533/randomselection.php';
 final response = await http.get(Uri.parse(randomMealsUrl));

 if (response.statusCode == 200) {
   final data = jsonDecode(response.body);
   final meals = data['meals'] as List<dynamic>;
```

code calling MealDB API (line 968 of dashboard_page.dart)

2.2.4 OpenFoodFacts API

This API is mainly used for our barcode scanning functionality, when users turn on the functionality and scan a barcode, we pass the decoded barcode into an API GET request to OpenFoodFacts, which is an open source database of over 2.9 million food items. We retrieve the product name as well as the quantity in order for users to confirm and add the item to their pantry, allowing them to filter the recipes with those ingredients.

```
void _fetchProductDataFromOpenFoodFacts(String barcode) async {
 final apiUrl =
'https://world.openfoodfacts.net/api/v2/product/$barcode?fields=product_name,
product_quantity,product_quantity_unit';

 try {
   final response = await http.get(Uri.parse(apiUrl));

   if (response.statusCode == 200) {
     final productData = jsonDecode(response.body);
     _showProductConfirmationDialog(productData);
   } else {
     // ignore: use_build_context_synchronously
     _showAlertDialog(context, 'Product not found, try again');
   }
 } catch (e) {
   // ignore: use_build_context_synchronously
   _showAlertDialog(context, 'Error fetching product data, try again');
 }
}
```

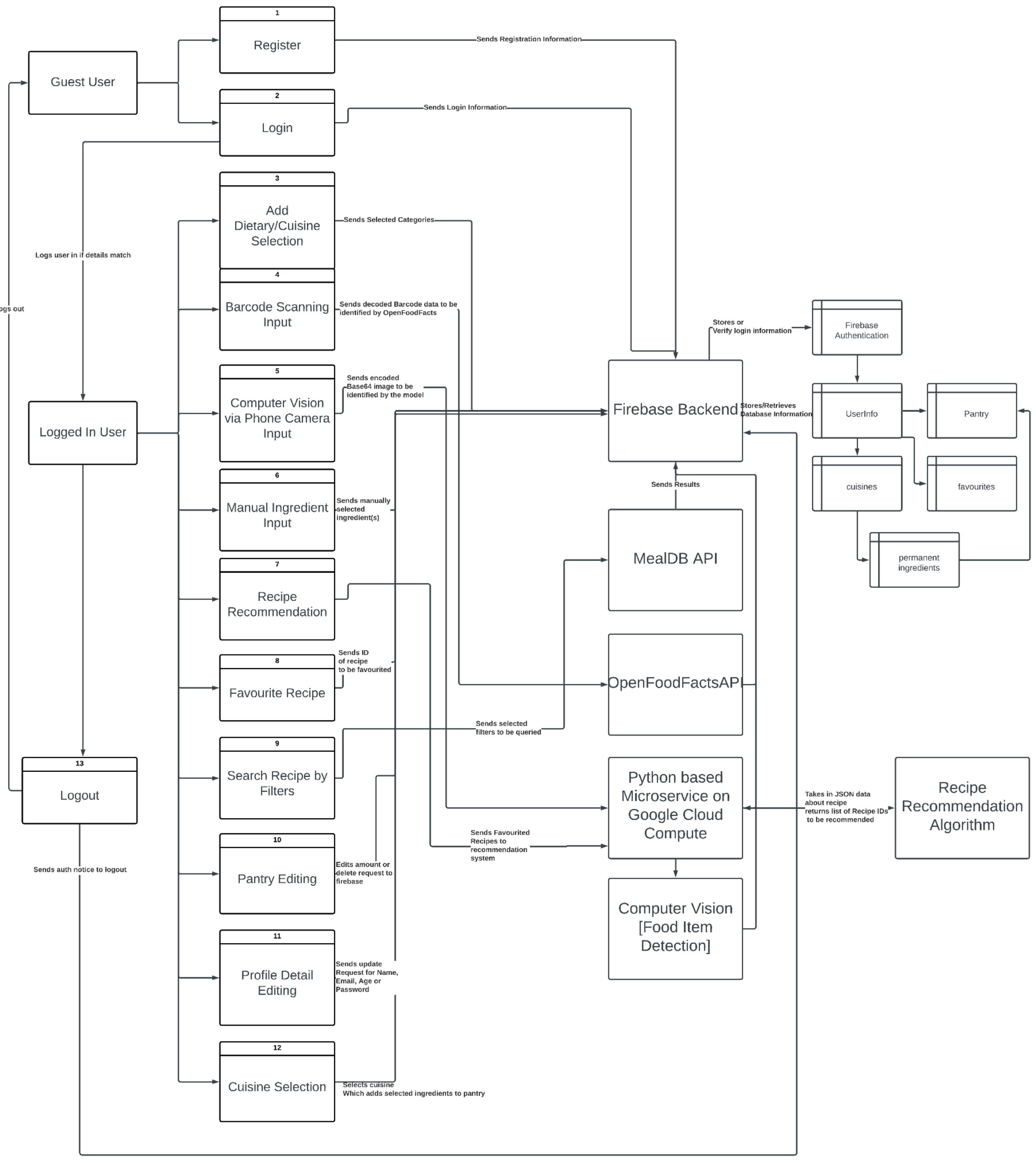
code calling OpenFoodFacts API (line 968 of dashboard_page.dart)

2.2.5 Microservice on Google Cloud

A python based flask microservice is built and hosted on Google Cloud Platform using their Compute Engine instance, this microservice hosts two main services, one of which is for the computer vision functionality, where the user takes a picture before its send as a Base64 image to the microservice, this is then decoded and passed to Clarifai where it uses a Food Item Recognition model in order to detect the item. We then filter it through a minimum confidence level filter before returning the detected item back to the user. It also hosts our recipe recommendation algorithm, which uses TF-IDF and cosine similarity in order to retrieve recommended recipes to the user.

3. High-Level Design

3.1 Data Flow Diagram


Guest User
1
Register
2
Login
Sends Registration Information
Sends Login Information
Logs user in if details match
Logs out
3
Add Dietary/Cuisine Selection
Sends Selected Categories
4
Barcode Scanning Input
Sends decoded Barcode data to be identified by OpenFoodFacts
5
Computer Vision via Phone Camera Input
Sends encoded Base64 image to be identified by the model
6
Manual Ingredient Input
Sends manually selected ingredient(s)
7
Recipe Recommendation
8
Favourite Recipe
Sends ID of recipe to be favourited
9
Search Recipe by Filters
Sends selected filters to be queried
10
Pantry Editing
Edits amount or delete request to firebase
11
Profile Detail Editing
Sends update Request for Name, Email, Age or Password
12
Cuisine Selection
Selects cuisine Which adds selected ingredients to pantry
Logged In User
13
Logout
Sends auth notice to logout
Firebase Backend
Stores or Verify login information
Stores/Retrieves Database Information
Firebase Authentication
UserInfo
Pantry
cuisines
favourites
permanent ingredients
Sends Results
MealDB API
OpenFoodFactsAPI
Python based Microservice on Google Cloud Compute
Sends Favourited Recipes to recommendation system
Takes in JSON data about recipe returns list of Recipe IDs to be recommended
Recipe Recommendation Algorithm
Computer Vision [Food Item Detection]

In the Data Flow Diagram above, it shows how a typical user could interact with the application's frontend through their phone and what typical requests or details would be sent to the system. It shows how systems interact with one another when the user starts an interaction. It also shows the availability of each feature depending on the level of access the user has, for example a guest user that's not logged in only has access to register and login, while a logged in user has access to the rest of the application except register and login, however they gained access to the log out functionality instead. This Data Flow Diagram also shows that data or requests flow to another component of the system, how it's then processed and how it's handled and passes it onto the next process.

3.2 Language Choice

Dart was chosen as our programming language as we were also developing using the Flutter framework, the combination of the two allows us to develop native compiled applications for both Android and iOS from a single codebase, which would reduce development time. Dart also compiles to native ARM code for mobile devices, which in combination with flutters optimised rendering engine, allows us to enable smooth animations with minimal performance overhead, this will be delved deeper in the testing section.
Another reason was due to Flutters hot reload functionality, allowing us to see the changes in the app almost immediately, which allows us to iterate through the UI Design.
Dart also benefits from great support within popular IDEs which we were planning to use such as Android Studio and Visual Studio Code.
While we were researching, we found out that Dart uses Ahead-of-Time compilation, which means apps compile down to native machine code which ensures fast start-up times and smooth performance. Ease of use was also one reason as the language has syntax similar to C, Javascript and Java. All of which we have used over the course of this degree.

4. Testing

Testing in our application was conducted through the use of Unit Tests, Widget tests and Integration tests, as well as adhoc testing as we developed and using Firebases' Test lab.
For dart we used packages like `flutter_test`, `flutter_test_driver` and `mockito` in order to test our flutter apps.

Unit tests:
We wrote unit tests for functions like fetch meal details and fetch recipes, which verify that the http response is returning the correct json data in the correct format.

Widget tests:
We wrote widget tests for our landing page and both our register and login pages in order to test various scenarios in both such as empty text fields, invalid email formats and invalid passwords and it also tests for the error messages that should appear like pop ups or hint text.

Integration tests:
We wrote an integration test that combines our landing page, registration page, login page and profile completion dialog when you first login to the application and arrive at the dashboard page. This test automatically locates the text fields in each page and fills in the required details before proceeding to the next. This test combines text field finding, alert dialog finding and submission as well as checkbox selection.

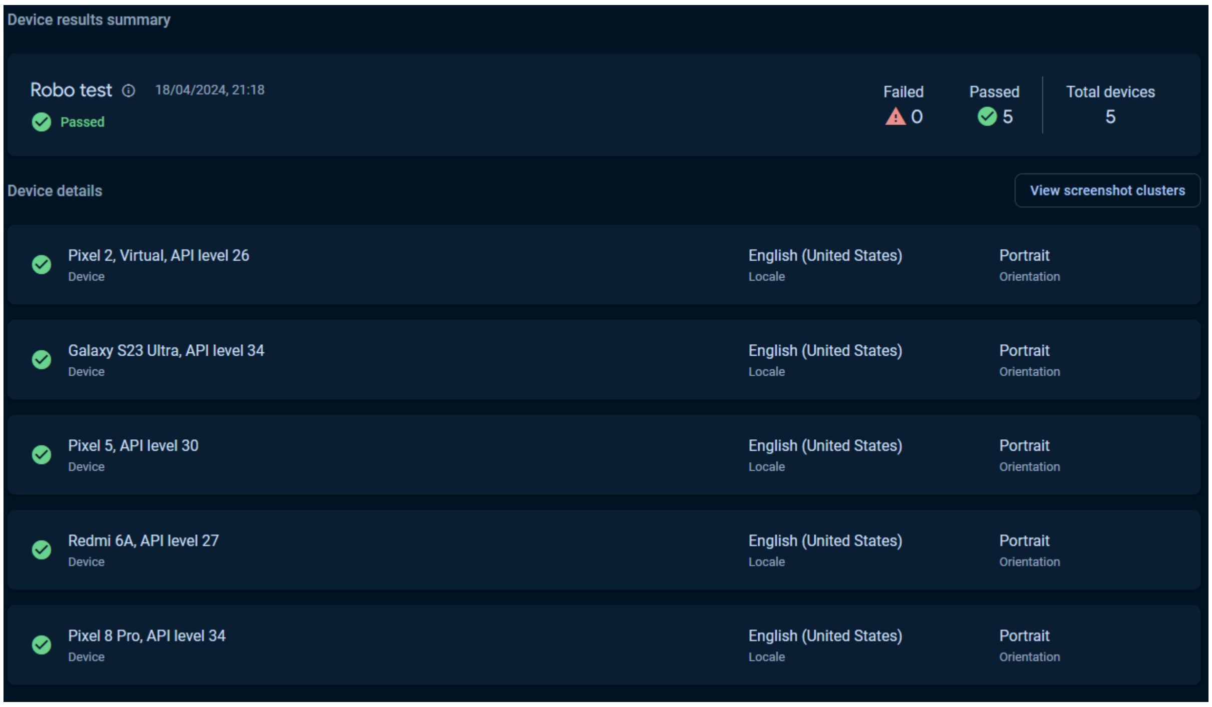
There were however some issues with writing additional tests, the reason mainly being a majority of our functions were tightly coupled with firebase, as such isolating specific components for unit testing becomes difficult, while there are third party libraries that can be used to “mock” firebase services, this caused us discrepancies compared to our actual firebase environment. Firebase services also involve multiple asynchronous calls and creating mocks for these would be time consuming and complex.

Ad-hoc testing was performed as we developed functionality and wrote functions, this was aided by Flutters hot reloading feature which allowed us to instantly see our changes.
Informal user testing was also conducted and additional navigation options were added from the feedback given as well as small quality of life features like user’s initially swiping left and right to navigate between the various views.

We also utilised firebases’ test lab and robo lab to test if our application can run on various android versions and what their performance is. We ran the rest on various android versions and API levels ranging from API 26 to API 34, which stands for Android 8 to Android 14.

Performance Data of 5 minute robo test:

Device + API Level	**CPU Average % Usage**	**CPU Peak % Usage**	**Average RAM Usage (Megabytes)**
Redmi 6A, API level 27	15-20% Average	52.5%	95 Megabytes
Pixel 5, API level 30	4-9% Average	21.09%	166 Megabytes
Pixel 8 Pro, API level 34	4-6% Average	14.38%	273 Megabytes



Device results summary

Robo test 18/04/2024, 21:18
Passed

Failed	Passed	Total devices
0	5	5

Device details

View screenshot clusters

Device	Locale	Orientation
✓ Pixel 2, Virtual, API level 26	English (United States)	Portrait
✓ Galaxy S23 Ultra, API level 34	English (United States)	Portrait
✓ Pixel 5, API level 30	English (United States)	Portrait
✓ Redmi 6A, API level 27	English (United States)	Portrait
✓ Pixel 8 Pro, API level 34	English (United States)	Portrait

5. Problems & Resolutions

5.1 Extensive coupling between App code and Firebase

This problem is mainly due to multiple functions having to interact with our firebase instance, as many of our features require calling our Firestore in order to get a read or update request. This led to problems down the line when we began to write unit tests for our application, as with much of our code being linked to firebase, mocking the interactions in order to write unit tests became increasingly more complex and difficult. The solution would be to refactor a majority of our functions, which we actually began to do with `meal_service.dart` and `auth_service.dart`, however it's a very time consuming task and requires us to extensively test each refactored component, as such it's very much work in progress.

5.2 Navigational Options

Initially, we only included a tap to navigate option, as we utilised a navigational hot bar on the bottom of the screen, while this was functional and was working well. In our informal user testing, we realised there were multiple users who attempted to swipe to navigate through the app, especially to move from one page to another as they saw the available pages in the navigation bar. However the way we implemented navigation logic in the navigation bar did not allow for swiping functionality, as such we had to refactor the navigation logic in order to make both navigation options to work. There was however an unexpected problem, where when you navigate to the profile page and then back out, you would end up on a blank page. This was later fixed by adding an index selection and jump to page check once the back button was tapped.

5.3 Flutter Animations and Integration test

While we were attempting to run our integration test for our landing page, registration, login and profile completion, an issue occurred that stopped the integration test from running. After some investigation, it seemed that the driver was waiting for animations to finish running and the app to “stabilise” before running the taps and text entering, however as we utilise infinitely running floating bubbles as a background, the animation never stops and as such stopping our integration test. In order to solve this issue, we used “`waitUntilFirstFrameRasterized`” method, which allowed us to wait until the initial UI of our app is fully rendered, by guaranteeing that at least the basic interface can be interacted with, then we made another change where the driver tests are wrapped in “`await driver.runUnsynchronized() async {`“, which allowed our tests to running without waiting for animations to stabilise. However, we accounted for animation loading anyways by adding additional delays in between each action. There were some issues with finding certain text fields, however this was solved by adding key values to the components in the main code.

6. Future Work

6.1 iOS Support

As mentioned before, our application was developed using the Flutter Framework which is compatible with iOS devices so it would not be too time consuming or tedious to be able to launch Demeter as an iOS application due to flutter’s declarative UI paradigm and hot reload feature rapidly increasing development time. During our investigation of our project we learned that flutter also includes a community-rich selection of iOS plugins that eases the platform-specific features if required. However there are a few extra steps to consider when launching for iOS devices such as XCode installation on a MacOS system as flutter may require a newer version of XCode tools in order to use flutter tools properly. Although it is not required to have a physical iOS / Android device in order to develop and test Demeter, it is certainly much faster when a physical device is available to developers.

6.2 Voice Recognition / Processing

From investigating the ways users could input ingredients into Demeter, the barcode scanning option was the most accurate and fastest way to achieve this, from our testing we found that it would fetch the correct ingredient every time. The computer vision / ingredient recognition feature came in second as it would sometimes not manage to fetch the correct ingredient. If the two previous methods did not work, the user could still manually enter their ingredients and their quantity. Voice recognition was another method that we aimed to deliver in the final application but we were not able to integrate. A way in which we could implement this is by using the `speech_to_text` library in flutter which is suited for short phrases such as ingredient names and recipes. Adding additional code to our `food_detection.py` file to handle audio that it is sent would be our first step and will require further research into Natural Language Processing in order to add a list of ingredients or singular ingredients to the Pantry as well their quantities. By adding this feature we could help users who may have impairments that could hinder their ability to input a lot of ingredients into Demeter.

6.3 Nutrition Details

For future development, we would like to implement more information about the recipes such as calories, track if the user has any allergies and to alert them if a dish contains a specific allergen. By integrating more APIs to Demeter like Edamam to provide details about the individual ingredients, although we must consider the API's accuracy cost as well as usage limits We can also allow users to enter their allergies on a list when they first log in and in the Profile page as well. This will enable users to get warnings whenever they get a dish that could potentially be harmful to them. By adding additional information like calories, we can potentially develop a Demeter wearable application that can work in tandem with the mobile application.

7. Installation Guide

First clone from our repository:
https://gitlab.computing.dcu.ie/borromk2/2024-ca400-borromk-2-linr-2.git
Then in order to run the application from command line to test and run:

```
-- Make sure you are in the Demeter directory , pwd
../../2024-ca400-borromk-2-linr-2/src/Demeter

-- Execute flutter clean to clean cache and to fix build problems
flutter clean

-- Execute flutter pub get to build Demeter's dependencies from pubspec.yaml
```

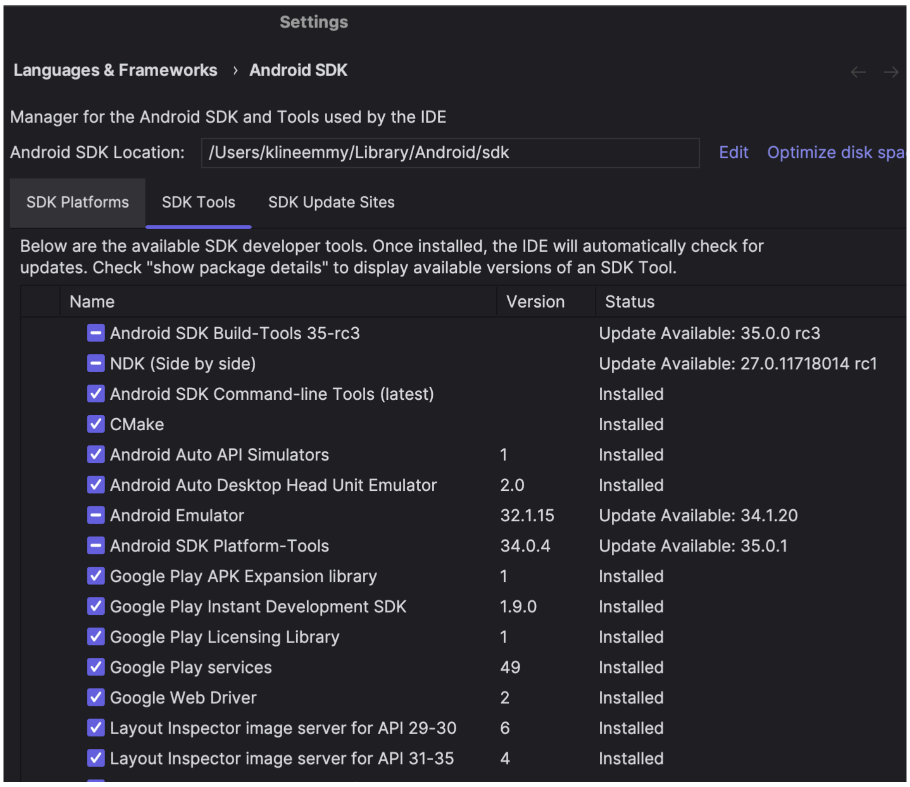
```
flutter pub get

-- Execute flutter run to build Demeter in debug mode
flutter run
```

7.1. Installation

7.1.1 General Prerequisites (if looking to develop on app or debug app):

- Demeter was developed using Android Studio Iguana | 2023.2.1
    - https://developer.android.com/studio/releases



Settings

Languages & Frameworks › Android SDK

Manager for the Android SDK and Tools used by the IDE

Android SDK Location: /Users/klineemmy/Library/Android/sdk Edit Optimize disk spa

Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
− Android SDK Build-Tools 35-rc3		Update Available: 35.0.0 rc3
− NDK (Side by side)		Update Available: 27.0.11718014 rc1
✓ Android SDK Command-line Tools (latest)		Installed
✓ CMake		Installed
✓ Android Auto API Simulators	1	Installed
✓ Android Auto Desktop Head Unit Emulator	2.0	Installed
− Android Emulator	32.1.15	Update Available: 34.1.20
− Android SDK Platform-Tools	34.0.4	Update Available: 35.0.1
✓ Google Play APK Expansion library	1	Installed
✓ Google Play Instant Development SDK	1.9.0	Installed
✓ Google Play Licensing Library	1	Installed
✓ Google Play services	49	Installed
✓ Google Web Driver	2	Installed
✓ Layout Inspector image server for API 29-30	6	Installed
✓ Layout Inspector image server for API 31-35	4	Installed

    - These are the SDK tools used on Android Studio
    - VSCode also has excellent Android development tools.

- The appropriate flutter version for your laptop/PC can be found here
    - https://flutter-ko.dev/get-started/install

The application was mainly built and tested for optimum performance from Android devices ranging from **Android 10 (API 29) to **Android 14 (API 34).** While it may be possible to run on lower versions of android, we cannot guarantee compatibility.**

7.2. APK Building & Installation:

```
- After running flutter run -> flutter pub get, we can install the app as an apk to our
laptop / PC

flutter install
OR
flutter build apk
```

- The `Demeter.apk` file will be available for download to the user and they will have to allow camera access when using the application.
- A **separate** completed `.APK` will be available for download if users do not wish to compile and build their own from downloading the source code, which users can just download onto their android phone and install with ease.

7.3 Running the microservice

- Ensure you are in the microservices directory.
- Ensure you have at least Python 3 installed.
- Create an .env file in the directory with the following details from your registered account on Clarifai

```
PAT = 'INSERT-PAT-HERE'
USER_ID = 'INSERT-ID-HERE'
APP_ID = 'INSERT-APP-ID-HERE'
MODEL_ID = 'food-item-recognition'
MODEL_VERSION_ID = '1d5fd481e0cf4826aa72ec3ff049e044'
```

- After that, open your terminal, and install the requirements.txt file using this command:

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

- Afterwards you can run

```
- python3 food_detection.py
```