Requirements Report

Equifood Team B

COSC 499 - 2022W

Anamica Sunderaswara, Eunsuh Lee, Minh Bui, Neilansh Rajpal

TABLE OF CONTENTS

1 GENERAL

1.1 High Level Description

Restaurants are one of the biggest producers of food waste, and Equifood is a mobile app designed to help combat that statistic. Through the app, restaurants are able to offer up their leftover food for free or at a significantly reduced price. Not only does this decrease the amount of food thrown away at the end of the night, but also enables consumers to enjoy the food at a subsidized price. By consequence, the app creates a symbiotic relationship between both consumers and restaurants.

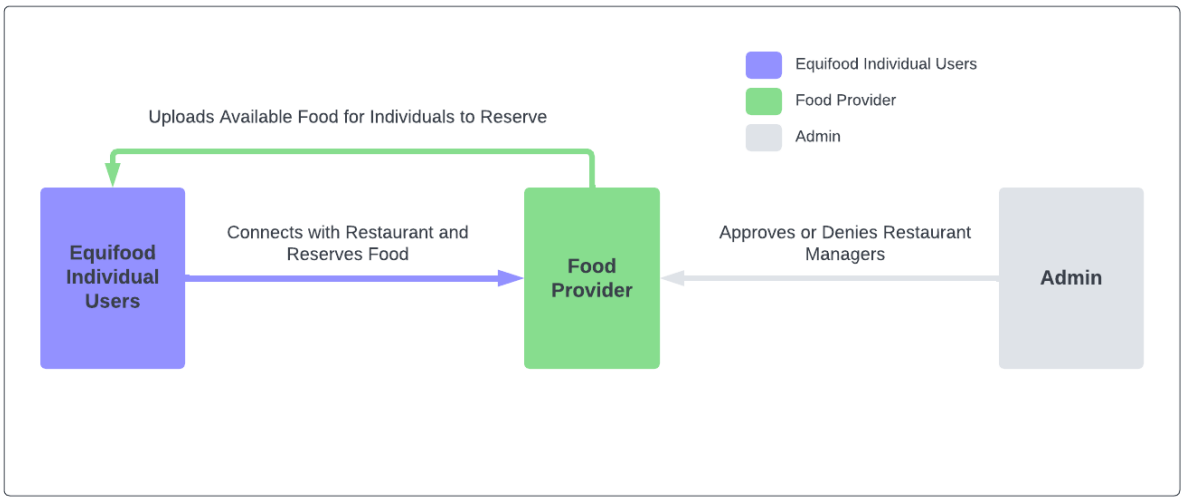
Additionally, to highlight the impact the app has on the community, Equifood also automatically tracks the actual dollar amount of food given away.

Equifood has three distinct user groups:

1) *The Individual:*
   The individual is able to log into the app and connect with a restaurant. Once connected, the individual is able to reserve food and commit to picking it up.
2) *The Food Provider:*
   The food provider is able to post donations on the app for which users can request to reserve. Additionally, the food provider can share the dollar amount of the donations they are providing with the administrator. Lastly, the food provider can also view and edit their information.
3) *Administrators*
   The administrator team's main purpose is to approve restaurants (and their managers) and view and post the donation amounts.

1.2 Data Flow Diagram Level 0

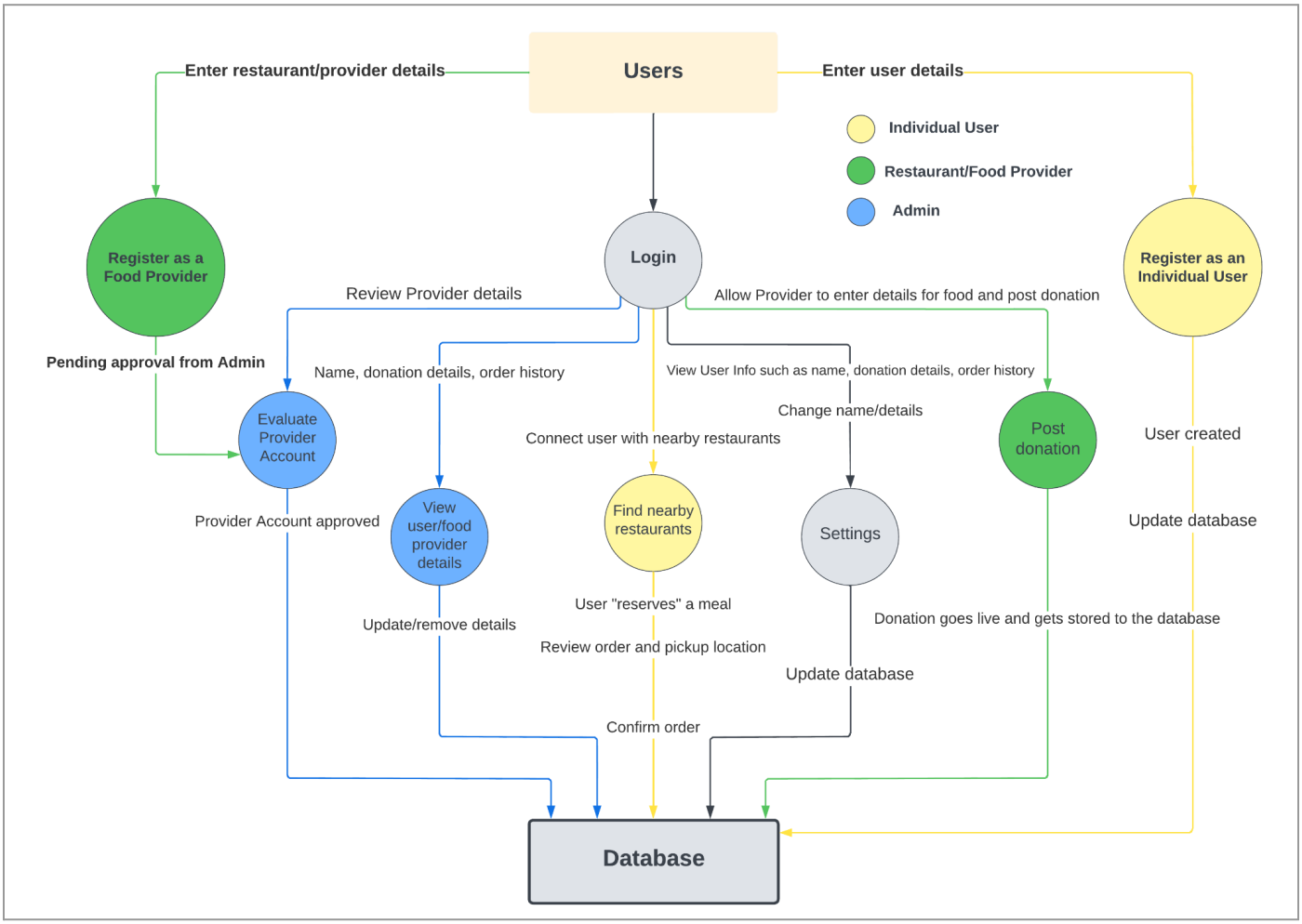
Below is the visualization of the Data Flow Diagram Level 0



The Data Flow Diagram illustrates the three user groups: EquiFood's Individual Users, Food Providers, and Administrators, as well as their most basic functionalities. The Individual User's main purpose is to connect with restaurants and reserve food. Next, the food provider is tasked with uploading available food for individuals to reserve. The Admin is meant to approve or deny restaurant managers.

We plan to work on all three user groups simultaneously, however, we plan to have the Individual's use case complete by Milestone 1, and the Food Provider and Administrator use case complete by Milestone 2.

1.3 Data Flow Diagram Level 1



The figure above depicts a Level-1 DFD, showing the processes that run after an action is executed while using the app.
Users are distinguished on the basis of 3 types: *Individual*, *Food Provider* and *Admin*.

Registering as a *Food Provider* would require further approval from the Admin. After a provider has been approved, their details get stored in the database.

Different options are provided after logging in, depending on the user type:

- Individual User: Connect with nearby restaurants, select an available meal, “reserve it”, change user settings, and view order details.

- Food Provider: Post donation, current donations, update a donation, change settings, view donation details such as history.
- Admin: Evaluate food providers, view details for food providers, edit/remove options, and donation statistics among many more.

The database will be storing information such as user/provider details, donation details and such as order information, donation history, and amount in dollars saved to name a few. These would also be available for the food provider to view.

Moreover, we have defined our milestones based on the features listed here. So each milestone will cover one or more of the processes mentioned here.

1.3 Development Milestones

- Milestone 1: We will have the Requirements Report ready by 21st October 2022.
- Milestone 2: We plan to have the User Interfaces for Individual User and Food Provider ready. This will also include the authentication feature. Deliverables for this milestone would be a video demo (#1) and Peer-testing (#1) to test the UI and ensure a smooth experience. The date set for this milestone is 25th November 2022.
- Milestone 3: We intend to have the screens for all the user types ready. The app for all 3 user types should be functioning. This will also feature backend integration. Deliverables will include a Video Demo (#2) and Peer-testing (#2) to test the working of the app across all user types. It is scheduled to be delivered in the first week of March 2023.
- Milestone 4: Some additional features will be added such as Profile Settings, Notifications, adding the Google Maps API for dynamic maps. Deliverables would include a Final Report and a Video displaying the fully-functional app. The deadline set for this milestone is 14th April 2023.

2 REQUIREMENTS

After discussing with the client, we have decided to build the system only for mobile users given the time constraint of our project. The team also proposed a list of requirements that the system should have to solve the main tasks of retrieving and providing food donations. First are the functional requirements which are features that the application must-have for the three user groups to use the system as intended. Next, the non-functional requirements will include external constraints on a broader scope and how we should approach the project from a technical standpoint to satisfy multiple requirements.

2.1 Functional Requirements

Firstly, the admin which is our client for the time being, should be able to:

- Approve restaurants after they have registered to the app for the first time.
- Get notified when a new restaurant requires approval.
- Remove existing restaurants from the database.
- View the total donation value that each restaurant has contributed.
- View the accumulated donation value for all restaurants, filtered by a time frame.

For the second user group, which consists of the food providers, the required features are:

- Post a donation that contains a description, thumbnail, price and quantity of items.
- Edit the donation details.
- Remove an existing donation.
- Mark donations as completed after someone has picked them up.
- Get notified when an item has been reserved.

For the most common user group, which is the day-to-day user, some of the requirements are to:

- View available donations from the dashboard
- View the donation details
- Reserve a donation for pickup

- Have a pickup timer to avoid users who do not eventually pick up the item, order by mistake or prank calls.
- Get notified when the reservation is about to expire.
- Review the reservation and pick-up location on Google Maps
- Have the option to contact the restaurant
- Add restaurants to a favourites list and get notified when they add new donations.

Furthermore, all of these three user groups must be able to:

- Create an account
- Login into an existing account
- Edit their account details

2.2 Non-Functional Requirements

For the non-functional requirements of the EquiFood app:

- The app is cross-platformed and there should be a single code base that compiles natively for both iOS and Android.
- The app should be user-friendly and layouts, buttons and headings should be minimal and intuitive
- The app should also be scalable and can handle a surge of new users since the main time frame for distributing leftovers is late at night.
- The fourth requirement is to be responsive, and it shouldn't take more than 10 seconds to load the main dashboard.
- And finally, the app should run without performance issues on the latest mobile devices.

3 TECH STACK

3.1 Frameworks Comparison

We considered three options: React Native (framework), Flutter (SDK) and Ionic (framework). These tools are cross-platform. So a single copy of the code will support both Android and iOS.

Some considerations that formed the basis of the comparison were:

- Performance
- How developer-friendly it is
- Reliance on 3rd party APIs
- Memory usage

The table below summarises our comparison.

Framework	Main Language	Cross Platform	Hot reloading	Performance	Development Speed	3rd Party Plugin/API reliance	Space Usage
React Native	JS/TS	☑	☑	Fast	Fast	More	Less
Flutter	Dart	☑	☑	Faster	Faster	Least	More
Ionic	JS/TS	☑	☐	Slow	Fast	More	Less

- React Native and Ionic are built on *JS* and can also be used with *HTML/CSS*. Flutter is built on *Dart*, which is another programming language.
- React Native and Flutter support *hot reloading* which will help us in testing features quickly. This also speeds up the development process. Ionic supports *live reloading* which requires the whole app to be executed every time to test even a small change, so development could slow down a bit.
- While there is not a noticeable difference performance-wise, Flutter is still a bit faster due to *Dart* being “closer” to native code. React Native on the other hand, makes use of a “bridge” that allows it to convert its components into native components - resulting in a “native” feel to the app.
- React Native and Ionic have a few built-in UI components. Additional 3rd-party APIs can be installed using the *npm* package manager. As there is less variety, Native and Ionic may have a considerable dependence on plugins. Flutter comes with a variety of built-in plugins powered by Google, which reduces the dependence on 3rd-Party APIs. This makes it space-efficient.

3.2 Features of Flutter

After weighing the pros and cons, we decided to proceed with Flutter as the tool to be used for developing the app. Here are some practical features offered by Flutter.

- Easier to learn.
- Developer-friendly, and thoroughly documented.
- Slightly better performance-wise.
- Has several built-in plugins and UI Components from Google, thereby reducing dependence on 3rd party APIs and making app memory efficient.
- Provides testing options which we'll look at in the next slide.

The team agreed upon using *Firebase* as the database. Flutter supports easy integration with Firebase.
We will also be making use of the *Google Maps* API, which, again, is supported by Flutter.

4 FEATURE TESTING

Continuous integration testing is done to detect any bugs, crash which will help reduce risk as development continues. This step helps speed up the progress as we, the developers, do not have to manually locate the error in each step we make in the development of our app.

For the continuous integration testing, Flutter's automated test library, GitHub action workflow and Riverpod would be used. Flutter breaks down testing into 3 stages: unit test, widget test and integration test.

4.1 Unit Testing

Unit testing is done in the coding phase to make sure there isn't any error. The unit test divides code into small portions and tests a method, class, or object. For example, getting input, checking input is right, and printing output would be tested in this stage. The Flutter framework provides default testing tools written in the Dart language, and the GitHub Actions workflow uses the .NET package to generate testing.

4.2 Widget Testing

The next stage of the continuous integration testing will be the widget test where we are testing UI components so the widget works as expected. The widget test will be done using flutter widget testing and Riverpod. Riverpod does not need to be reimplemented, since it implements retrieving and caching data well.

4.3 Integration Testing

For the integration testing, the intergration_test package from Flutter and GitHub Actions workflow will be used similarly to the unit test. An integration test is testing a big chunk of an app from the user's perspective to see if the entire flow of the app is working. In the other words, API will be mainly tested in this stage, for example, the user registration and log-in and out, connecting the restaurant with the map will be done.

4.4 Regression Testing

The regression testing is the final coding phase test to make sure the code from the previous version does not cause any bugs for every update. We will use Flutter testing tool kits called golden_toolkit package and reset-all functions. This is overall important to our project because we require continuous updates and improvements to our code and work.

4.5 Usability Testing

For the usability test, the developer will invite participants to observe how they would be able to follow the app. The participant will receive a scenario to perform.

For the app we will have two tasks:

1) *Individual users* - who try to order a food item
2) *Restaurant owners* - try to upload their restaurant information and confirm an order.

While participants are performing, the researchers will record and ask questions to participants after their tasks are done to get any comments or feedback to update the design of the app to be more friendly to users.

5 QUESTIONS AND FEEDBACK

5.1 Answers From The Team

1. *How will you verify that your application works on a variety of devices, does flutter provide emulators?*

- One of the main reasons we chose Flutter as our Tech Stack is because it has a facilitated process for testing our application on a variety of devices. Flutter provides an emulator through Android Studio. This ensures that we are able to test our application for Android Devices. Additionally, Flutter has a Default Inspector to test our code for iOS devices. To have the application working on both Android and iOS devices was a part of our functional requirements, thus corresponding testing features were crucial.

2. *Is this platform like an "ordering" service platform then? Users "buy" online then pay in person and collect the food in person? Since there will be no delivery or transactions made online.*

- That is exactly what the premise of our application is. Users are allowed to reserve food through the app, and then have the responsibility of collecting and paying for the food in person. The client is strict about the application not being a business but rather simply a liaison between a food provider and an individual consumer.

3. *Why use Firebase as your database considering it is decently priced?*

- Firebase has set limits for data storage. We will be charged after it crosses the basic limit, which is about 10 GB. Additionally, it provides other basic features covered under the basic “Spark” plan, such as Authentication and Notifications, which we plan to implement during Milestone 4. Not to mention the easy integration with Flutter, since they both are from Google.

4. *Could you go into more detail about how you plan to secure user logins?*

- We have planned to employ the authentication services provided by Firebase. It allows signing in with email, Google, iOS and other third-party services such as Facebook.

5. *How do you plan on implementing push notifications? Does this require native code or does Flutter support this?*

- While this feature remains one of the “good-to-haves”, we plan on using the Firebase Cloud Messaging (FCM) Services offered by Firebase, which will help us send notifications to users.

6. *You mentioned showing where the restaurant is on Google Maps, will you leave it up to Google Maps to recommend how to get to the restaurant? Will restaurants have a description on their profile recommending how to get there?*

- Google Maps provides a variety of different APIs to choose from, each catering to a different purpose. We will be employing the Places API which will render a map showing the restaurant as a marker. On clicking it, the restaurant details such as name and address would be displayed.

7. *Who will be the main admin of the software/program, the developers or the stores?*

- The developers will be having admin privileges to the app. We will also be discussing with our client about others from their team who may be granted admin privileges as well.

8. *What do you have in plan for your app to be compatible with older mobile devices?*

- As per our project requirements, the app will be developed to support iOS and Android systems. The app will be developed using Flutter, which supports iOS 11 (2017) and later for Apple devices, Android 4.1 (2012) and later versions for Android devices.

9. *Will the individual be able to see on their account how much food waste they’ve helped reduce?*

- Yes, we will be adding a similar feature so that the users, including food providers, will be able to see the amount of food they have saved till then.