COSC 499 – 001

Capstone Project

2022/2023 Winter Term 2

FINAL REPORT

EquiFood Team B

Group: Minh Bui, Neilansh Rajpal, Anamica Sunderaswara, Eunsuh Lee

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 it also enables consumers to enjoy the food at a subsidized price. As a 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.

1.2 User Groups

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.

    User Scenario:

    Laura is a busy student who just finished studying and is feeling too tired to cook tonight. She decides to order food using EquiFood which she recently downloaded on her smartphone.

    Laura opens the app on her smartphone and is greeted with a home screen featuring various food options and restaurants in her area.

    Laura scrolls through the available options and sees their discounted rates. She sees a restaurant she likes and decides that's what she wants for dinner.

    Once she's selected her meal, Laura clicks reserve.

    Once reserved, Laura is shown a confirmation screen that shows her how much time she has to go pick up her order and is able to cancel her order if she desires.

    Overall, EquiFood has made it easy and convenient for Samantha to order her favorite food and save some money.

2) The Food Provider:

- The food provider is able to post donations on the app 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.

User Scenario:

Dough Bro's is a recently approved restaurant on EquiFood that has been struggling without the amount of food waste they produce at the end of each night. Today is their first night adding donations to the application. Dough Bro's opens up the app and views the button prompting them to add a donation.

They have a lot of pizza slices left-over tonight, so Dough Bro's adds a donation offer for the pizza slices, supplying the app with all the details and giving them a 40% markdown. Once satisfied with their input, Dough Bro's uploads their donation and is able to view the confirmation of successful upload.

Now, Dough Bro's waits for a user on the individual end of the platform to reserve their order and come for pickup.

Overall, EquiFood helps restaurants like Dough Bro's get rid of their extra food without causing harm to the environment.

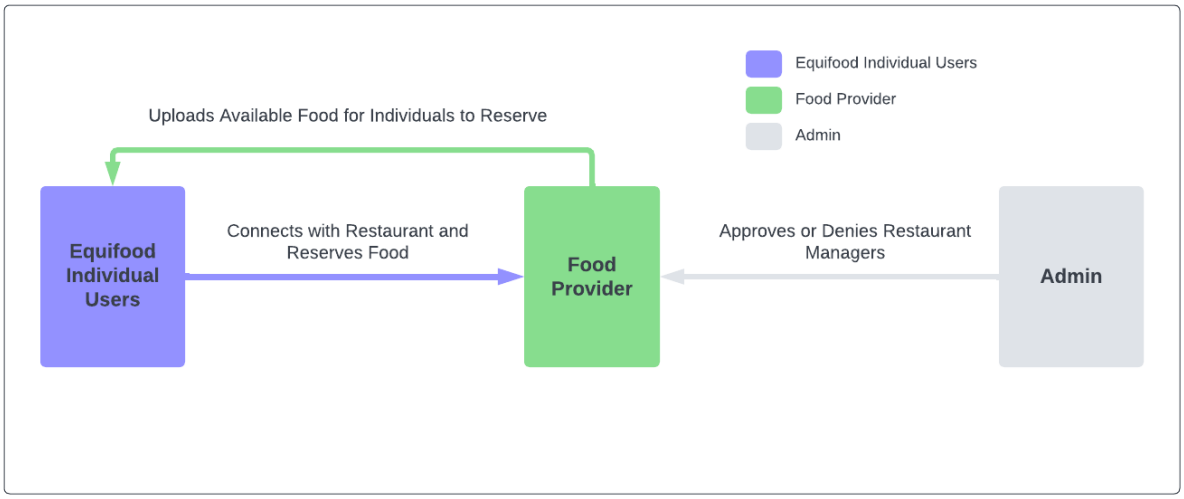
3) Administrators:
    - The administrator team's main purpose is to approve restaurants (and their managers) and view and post the donation amounts.

User Scenario:

An admin logs into the EquiFood app and is able to see the restaurants that are pending approval. The admin does further research on each restaurant with their own resources and returns back to EquiFood to decide if the applying restaurants fit EquiFood's values or not. If so, the admin approves the restaurant and the restaurant is added to the list of *Approved Restaurants*. If not, the admin has the power to deny the restaurant.

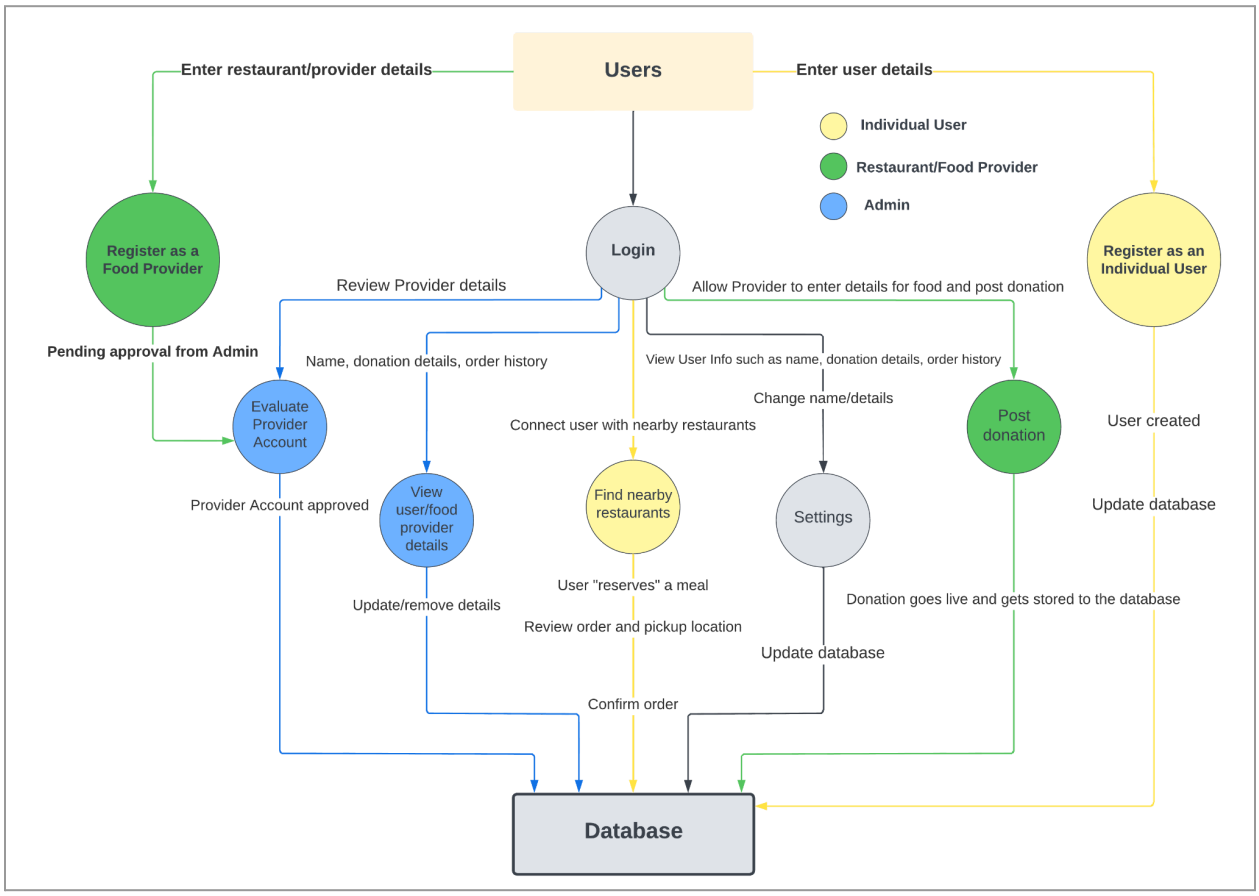
1.3 Data Flow Diagram Level 0

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.

1.4 Data Flow Diagram Level 1



- The Level-1 DFD above shows 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 Administrator**.**
- Registering as a Restaurant/Food Provider would require further approval from the Admin. The admin is provided with details of the unapproved Restaurant user, which helps them determine their legitimacy.

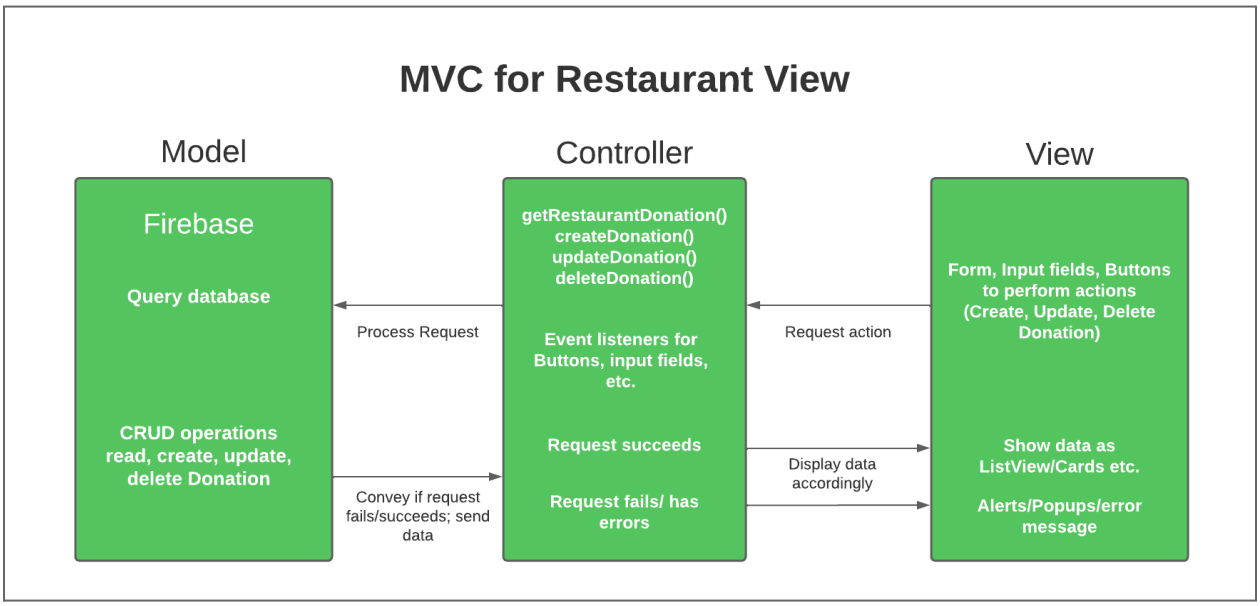
Different options are provided after logging in, depending on the user type:

- Individual user: Connect with nearby restaurants, select an available meal, view meal details and “reserve it”, and view their orders and statistics.
- Restaurant/Food Provider:
    - Approved user: Post donation, edit and delete an existing donation, view statistics.

    - Unapproved user: Displays a message that advises the Restaurant user to wait for the Admin's approval.
- 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 and metadata about the donation, and meals reserved by the users just to name a few. These would also be available for the food provider to view.

2 SYSTEM DESIGN



- The app employs a Model-View-Controller, or simply MVC, architecture.
- The functionality offered by the app differs based on the user type (Regular, Restaurant or Admin User).
- As the name suggests, this architecture divides up the app into 3 basic components.

2.1 Model

- We are employing Firestore as a means of storing data.

- The Firebase architecture offers built-in functions that help us manage data. This could be thought of as the Model component since it interacts with the database.
- It supports *CRUD* operations and has mechanisms to *view*, *query* or *manipulate* data.
- To exemplify, after a donation is created by the Restaurant User, it can be edited or deleted. Such actions are possible through the Model component, which performs the appropriate action based on the information received from the Controller end.

2.2 Controller

- This component is responsible for receiving requests from the user.
- In our case, the requests could be thought of as "inputs" from the user. The input is processed by the Controller logic which then interacts with the Model to perform the correct action.
- For example, to create a donation, the Restaurant User enters the input fields on the appropriate page/form and requests to create a donation by clicking the "Create" button.
- The Controller logic processes the inputs and interacts with the Model Component (firebase), which executes the appropriate action (i.e, to create a donation and add its details to the database).
- The Controller also checks for errors after data is returned and interacts with the View controller to show it correctly, thereby acting as a Mediator between the Model and the View.

2.3 View

- In simple words, this is the UI of the app with which the user interacts.
- The app features a minimalistic UI which changes appearance dynamically through state-handling.
- We also have some mechanisms to enable the UI to keep "listening" for changes in the database and update the screens accordingly. This updates the screens in real-time.
- For example, if a new Donation is created on the Restaurant User's end, it will appear immediately on the Regular User's Dashboard. In case there are no donations at a given time, the Dashboard displays the appropriate message.

- As previously noted, the Controller interacts with the View to tailor the appropriate content, based on the success/failure of the user's request.

3 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.

3.1 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 favorites 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

3.2 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.
- Finally, the app should run without performance issues on the latest mobile devices.

4 SOFTWARE IMPLEMENTATION

4.1 Technical Specification

The EquiFood app employs the following stack:

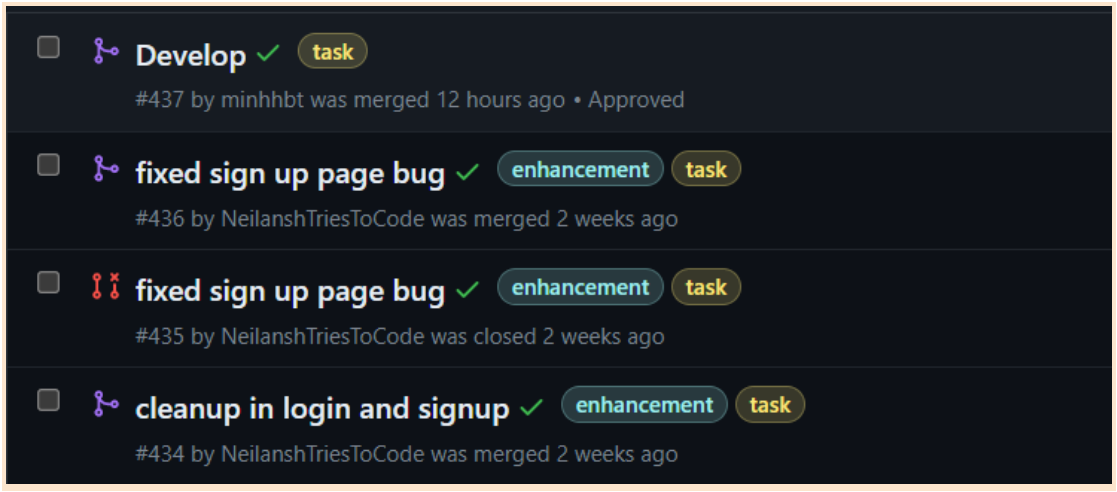
1. Flutter: A frontend developing framework used to develop cross-platform apps for Android and iOS devices. Offers built-in UI Widgets, downloadable dependencies and easy integration with other services. This framework is built upon the dart language.
2. Firebase: Used to host our database. Also offers authentication services to manage users. Easy setup for Android and iOS apps.

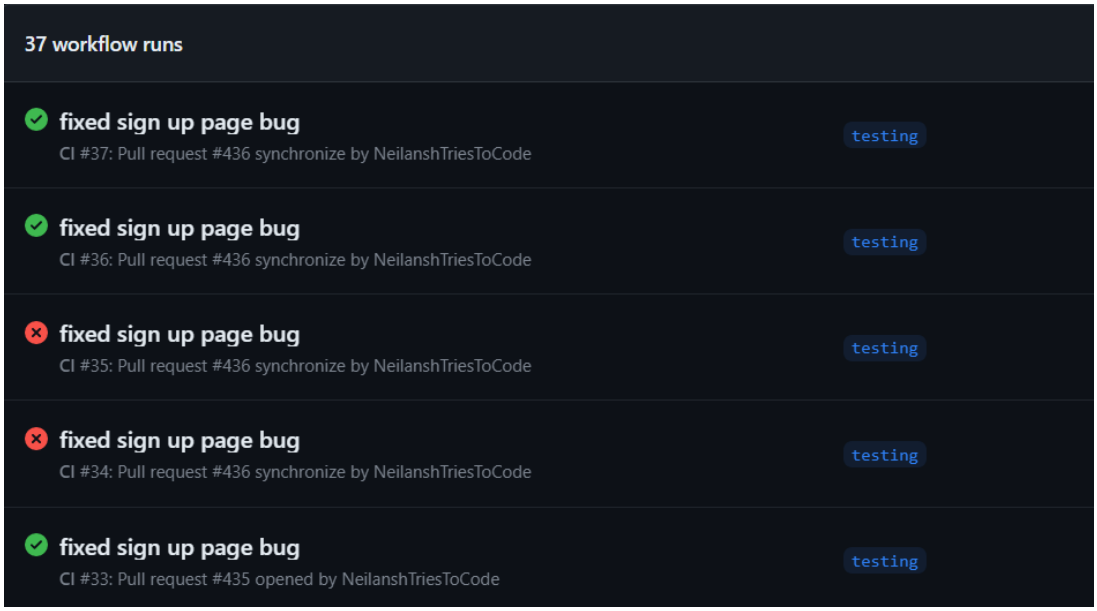
4.2 Updated Test Report

We employed the built-in test library from Flutter to write tests and further set up GitHub Actions to run them on every pull request made to the *develop* branch. A summary of the tested features is provided below:

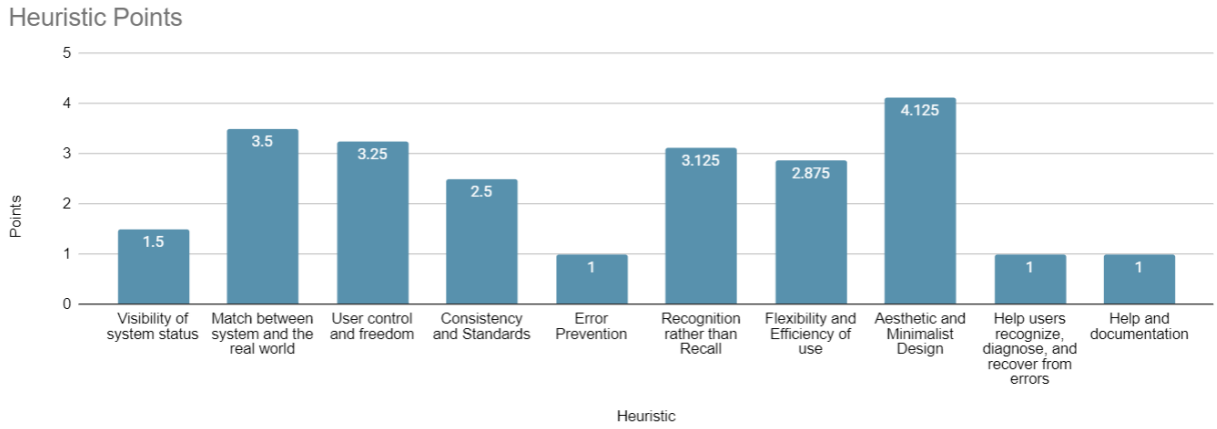
Sr. No.	Feature	Description	Pass/Fail
1.	Sign-up Page	Ensure UI elements such as input fields, Buttons are rendered appropriately.	Pass
2.	Restaurant Sign-up page	Ensure error mechanisms (alerts/dialogs) are in place when input fields are empty/invalid.	Pass
3.	Login Page	Ensure UI widgets are rendered appropriately.	Pass
4.	Login Page	Ensure feedback elements show up if the user's inputs are empty/invalid on the login page.	Pass
5.	Dependencies	GitHub Actions also check if all the dependencies are in place and not outdated.	Pass

Below are a few screenshots depicting some tests:

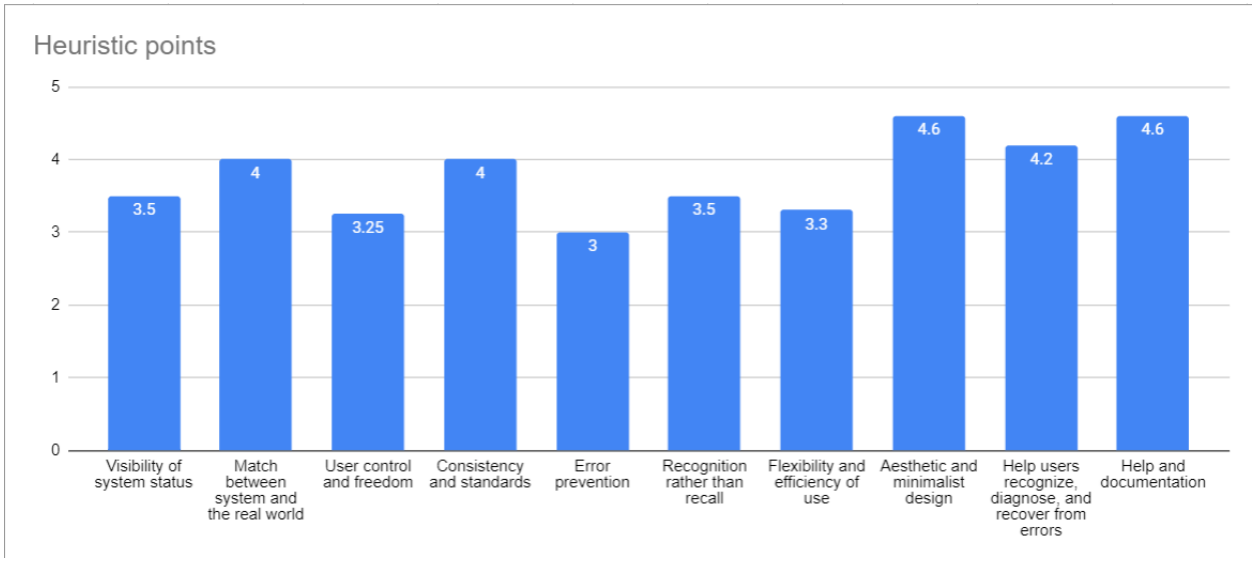




We made changes to our UI mainly based on the heuristic feedback from our peers. Most of the drastic changes are from Peer Test #1.



In our Peer Test #2, we mainly made subtle refinements to our app and the results were much more positive in terms of how the app was more intuitive and better in terms of appearance.



4.3 Unimplemented Requirements

Sr. no.	Feature	Status
1.	Reserve/Cancel donations	Complete
1.1	Add quantity field while reserving donation	Not complete
2.	Push notifications (Individual and Restaurant Users)	Not complete

3.	Update settings (all user types)	Not complete
4.	Read data for Statistics (all user types)	Not complete

4.4 Known Bugs

Problem	**Bug Severity**
Display not tested on Android because the team members only have iOS devices.	High
If a user reaches the reservation screen, and another user reserved that item first. There's no code to deal with the double reservation.	High
Map API call is not working as expected when passing a specific location to be called	High
Firebase backend code only makes it possible to have 1 admin account	Medium
Installation steps are really simple on Windows but are tricky on Mac, because the instructions for the terminal depends on the user's OS version so they might have to do some extra research.	Medium
If the user does not attach an image link to their user account or donation, the app will display a red error until it is added in the backend.	Medium

5 PROJECT HANDOVER

5.1 Project Link

The source code can be found on our GitHub repository.

The EquiFood promo video can be found in this private link.

5.2 Installation Details

Prerequisites:

- Flutter SDK, JAVA, Visual Studio and Android Studio should be installed on your machine.
- Visual Studio Code is recommended as a code editor, but other code editors can be used as well.
- For *Windows*, make sure you have installed the required software and configured your environment variables properly.
- For *macOS*, Xcode needs to be installed additionally.
- You can find more details on how to install them on the official Flutter website: link

Step 1: Clone the GitHub repository

- Open your terminal and navigate to the directory where you want to clone the app's repository.
- Run the following command:

```
$ git clone https://github.com/COSC-499-EquiFood-B/EquiFoodApp-B
```

Step 2: Install dependencies

- Navigate to the root directory of the cloned repository in your terminal.
- Run the following command to install the app's dependencies:

```
flutter pub get
```

Step 3: Run the app

- Make sure you have an emulator or a physical device connected to your machine.
- Run the following command to build and run the app:

```
flutter run
```

5.3 Maintenance Issues

1. Compatibility issues: The app may encounter compatibility issues with different versions of mobile operating systems or with other software used by the user.

2. Firebase: We are currently employing the free-tier from Firebase. Once the app is deployed, the client may have to upgrade to accommodate more users and requests as well as other features.
3. Deployment: Android and Apple developer accounts would have to be created. It would also be necessary to go through the instructions, costs and guidelines to ensure that the app meets the criteria for deployment on the platforms.

5.4 Testing of Existing Features*

Firebase: Our initial plan was to test Firebase operations (CRUD operations) by using dummy credentials from our database. However, the recommended way of testing Firebase is using an external library that offers a mock database. Due to complexities that would have been introduced (affecting nearly all our Widgets) and time constraints, we could not properly test database-related operations.

**Any testing of existing features remaining that have NOT been done.*

5.5 Remaining Features

1. Notification System - While our app has the functionality to reserve/cancel orders, we could not have notification mechanisms in place due to time constraints and complexity. In future updates of the app, notifications such as push notifications and email could be set up. A possible scenario where this could be used is to send individual users pick-up reminders about their reserved meals or notifications about offers/discounts, etc.
2. Quantity options - Allow users to choose quantity/meal servings when reserving donations and make the change accordingly on the backend.
3. Payment system - In future updates, we can add a payment option using payment gateway APIs such as Square to handle payments.
4. Timer - Right now we are displaying the time by which the user should pick up the donation but having a timer countdown down would be a nice addition to have.