

Requirements Report

EquiFood App - Group A

Abhiek Bist
Jake Daongam
Amrita Sidhu
Griffin Wilchuk

Our Software

The purpose of Equifood is to aid in reducing food wastage of restaurants. This app presents restaurant owners the idea of giving away excess food they have at the end of each day at discounted prices, or for free. Equifood administrators manually keep track of the money amount Equifood has saved individuals from restaurants. These are considered as donations.

Our software will allow restaurant owners to post their discounted/free offers for individuals to see and will automatically keep track of donation amounts.

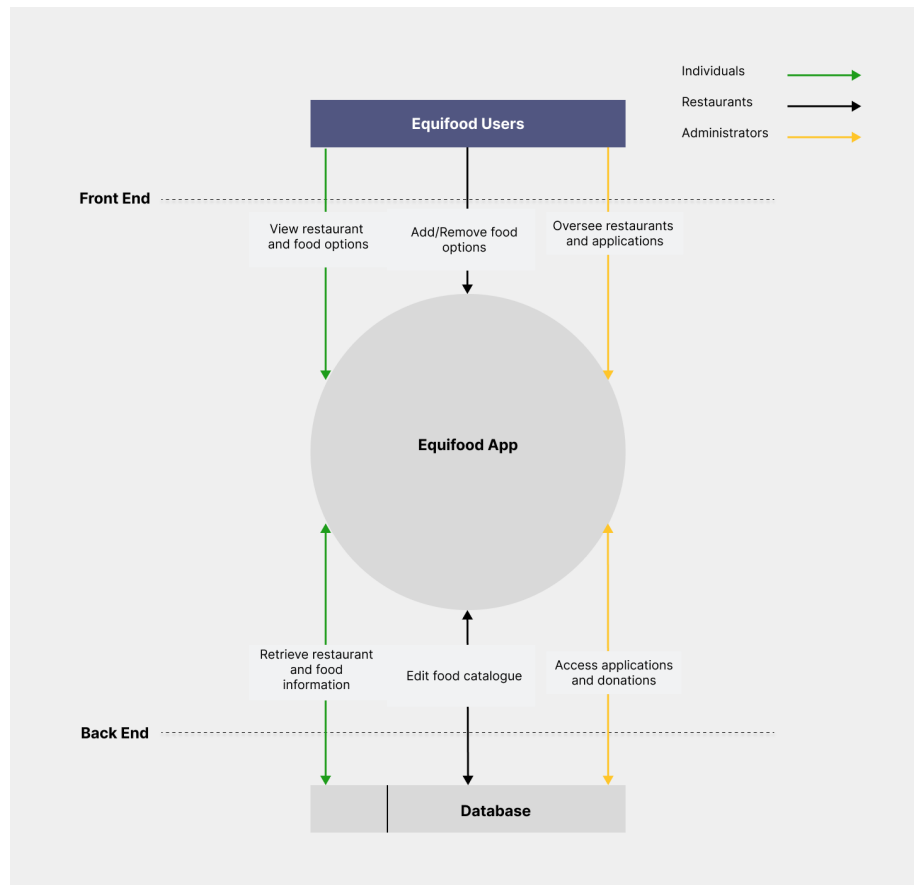
Target User Groups

Our target user groups consist of:

- The administrators, the individuals in ownership of Equifood.
 - Their purpose in using the app will be monitoring that restaurant owners are using Equifood as per their guidelines. They will have the option to approve, deny, or remove restaurants from this app.
 - Admin will have access to viewing donation amounts.
- The Restaurant Representatives.
 - The app will allow them to post information about their restaurant and food options available.
 - They will update total donations collected from their restaurant.
- Individuals.
 - Will be able to use the app to view available restaurants and food options.

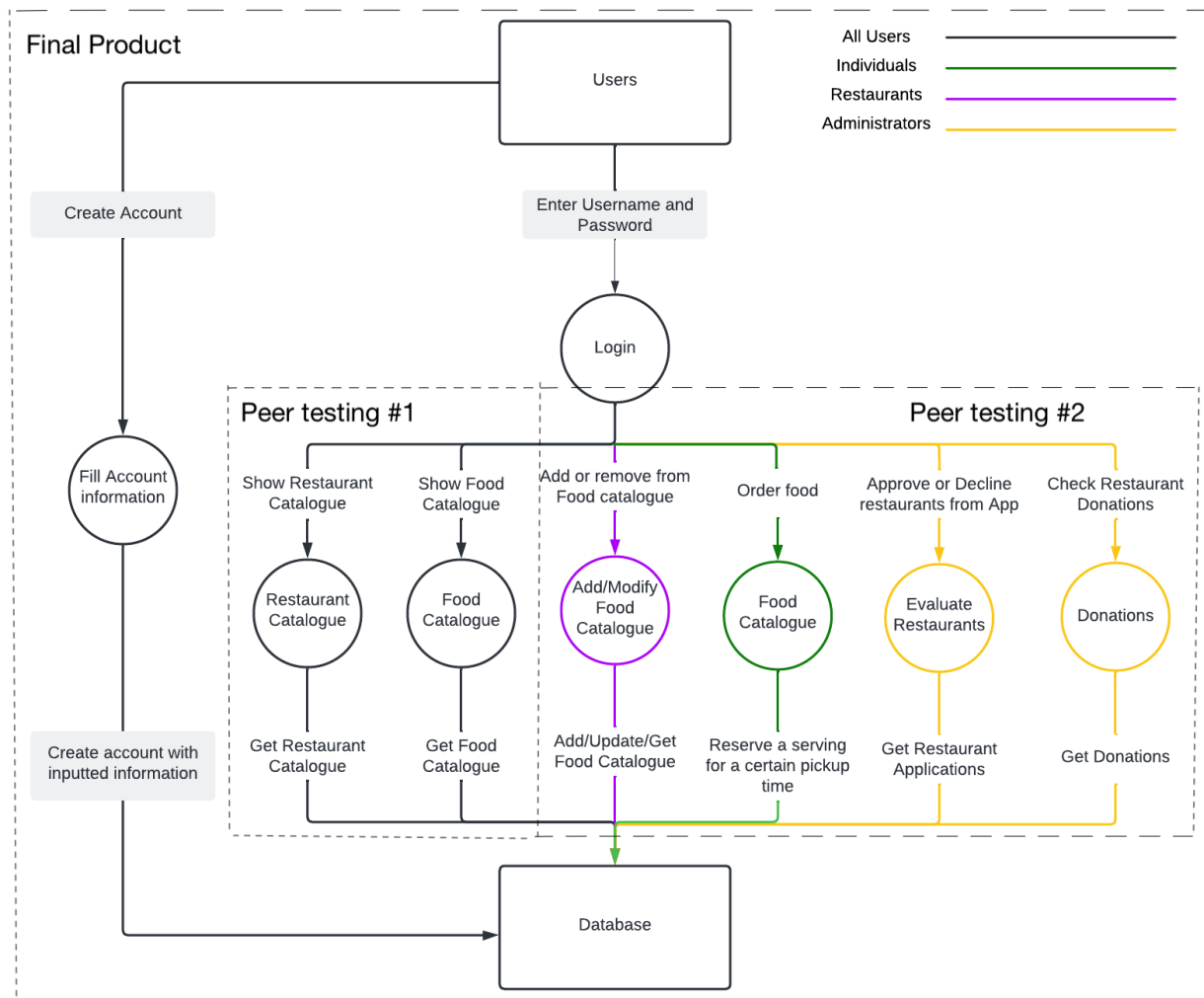
Data Flow Diagrams

Level 0 DFD:



The level 0 DFD is a very high level overview of how data will differentiate between the target user groups. The individuals will primarily be focused on selecting a restaurant to get food from, and selecting whichever food they want from that restaurant. This requires the app to have a front end that displays the food and restaurant options, as well as a backend that retrieves that information from the database. We will be focusing on implementing a basic front end and back end for the individual features for Peer Testing #1. The Restaurants will primarily be controlling the food options and inventory levels for their own store page. This will require a different UI to allow them to easily edit their store page, which will also require some extra back end features to allow the editing of the database. This will be a goal for Peer testing #2. Administrators will be using the app to moderate restaurants and approve/decline incoming restaurant applications. They will also require a different UI and database privileges to give restaurants permissions to edit. This will also be a goal for Peer testing #2.

DFD Level 1



The DFD level 1 is similar to the level 0 dfd but is a little more detailed in the data flows. It breaks down the features by milestones:

1. Peer testing #1 milestone requires the app to have a basic UI that can get restaurant and food catalogues from a database and display them to the user. The goal is to just have the functionality down for peer testing #1 and to iterate on the design for peer testing #2 for a better user experience.
2. Peer testing #2 adds in the functionalities for individuals, restaurants, and administrators. The individual functionality is to be able to order and reserve a certain pickup time for the food they select. The restaurant functionality is to be able to add or remove from their restaurant store page, and the administrators functionality is to be able to oversee restaurants, approve or decline restaurant applications, and to check overall donation amounts from restaurants. Once

again, the goal for these are to just get the functionalities down for peer testing #2, then to iterate on the design for the final milestone for a better user experience.

3. The final milestone will include adding in the login page, populating the database, and ensuring that the final design is nice to use for the users.
 - An extra component that could be implemented would be a review system, where individuals would be able to review restaurants and give them a score. This would help keep restaurants accountable for the quality of the food they provide on the EquiFood app.

Functional Requirements

Milestone 1 (Peer Testing #1): Implement basic front end and back end to allow users to see and navigate restaurant and food options. This includes basic implementation of:

- Restaurant browsing screen which showcases all restaurants in block style
- Food browsing screen which showcases all foods in block style
- Create basic database for food and restaurant options

Milestone 2 (Peer Testing #2): Populate database, improve navigation, implement food reservation

- Add onto database with information about accounts, donations, and more food and restaurants options
- Food reservation will require:
 - Food portion inventory incrementer
 - Customer pickup system
- Create page for restaurants to add and remove foods from their page
- Create page for administrators to review restaurant applications, also give them the ability to remove restaurants from the app
- Create page to allow administrators to see donation totals for all restaurants

Milestone 3 (Final Product): Implement login page and registration page. If time permits, implement a restaurant rating system. Then extensively test, document, and prepare for handoff.

- Create login page for different users
- Create registration page for users

Non-functional Requirements & Environmental Constraints:

- Effective communication with the client. Preferably weekly.
- Team members level of familiarity with the coding languages and development environments chosen, or willingness to learn.
- Team members personal laptop restrictions to software or memory we may require for development.
- The reliability of software will be determined by running it many times with the expectation of accurate results 90% of the time.
- The robustness of software that in the case that it cannot interpret the user's request, it declines gracefully.
- The adaptability of code by:
 - Organising code with comments for further clarity.
 - Detailed documentation for potential further development of the software.
- The usability of software by implementing design that is easy to understand by users with minimal understanding of technology.

Tech Stack Identification

Our client has minimal technological background, as such we presented 3 options to the client. Below are all 3 options with the pros and cons of each as well as our rationale for our final decision. (all prices are converted to canadian dollars)

Technology	Product	Cost
Frontend	React and Node	Free
Backend	Express	Free
Database	MongoDB	Free (\$15 a month if more storage is needed)

This stack is commonly referred to as the MERN stack. It is highly documented and used in a variety of established apps. React, node and Express are all free allowing us to build a rock solid Frontend and Backend without any cost. MongoDB is used for the Database as it is extremely robust and well-documented. If the app grows rapidly, larger storage capabilities can be hosted for \$15 a month. Apple's yearly developer fee of \$135 and Androids one time \$35 fee also contribute to our total estimated cost of \$35(Android's fee) + \$135 yearly(Apple's fee) + \$15 a month(expanded storage).

Technology	Product	Cost
Frontend	React and Node	Free
Backend	Express	Free
Database	MariaDB	Free

Our second stack is a slight variation of the MERN stack. MongoDB is replaced with MariaDB, an open source database. Although MariaDB does not have as solid of an infrastructure as MongoDB it is much quicker. Once again, Apple and Androids fees still apply here. Our total estimated cost is \$35 + \$135 yearly. These costs are solely the developer fees. **This is our chosen stack.**

Technology	Product	Cost
Frontend	Flutter	Free
Backend	Firebase	Free (\$5 a month if more storage is needed)
Database	Firebase	Free (same scaling cost as above)
Design	Flutterflow	\$560 for two months of team access

Our third and final stack runs on google's Firebase. Firebase is extremely well documented and trusted by many companies for their backend and hosting. Our Frontend would be made using Flutter. Flutter is a high level program for developing clean looking and fast UI's quickly. The addition of Flutterflow is included here. This is an extension of Flutter which allows users with little design knowledge to make very professional looking applications. Although useful, it is very costly at \$70 per user, per month. Our estimate calls for 2 months of Flutterflow for each member of our team leading to a cost of \$560. Firebase's hosting capabilities can be vastly expanded for approximately \$5 a month, this cost is also included in our estimate. Our total cost estimate for this stack is: \$560(Flutterflow) + \$35(Android fee) + \$135 yearly(Apples fee) + \$5 a month(Firebase).

Our client preferred we use one of the two MERN stack variations as the Flutter stack was simply too expensive. Our team decided to use the MariaDB variation. We prefer MariaDB as its relational model is crucial for our queries and additions of new available products. Its open source nature is also a massive upside. Express was chosen for its concise code and phenomenal scalability. Similar to Express, React and Node were chosen for their scalability and speed along with a great amount of documentation and support material.

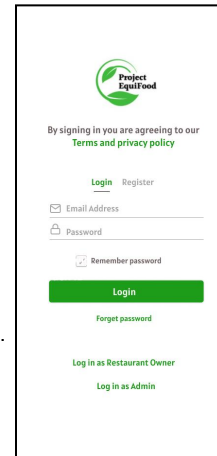
Prototype Version 1

After our initial discussions with our client, we produced our first prototype with the following attributes:

Loading Screen:

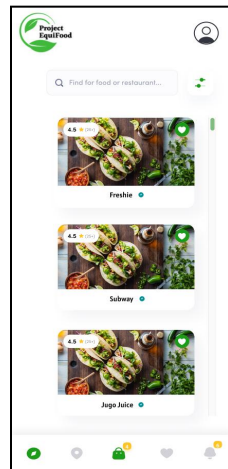


Login Screen:



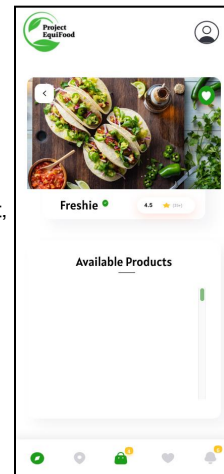
Includes the option to log in as restaurant owner or admin.

Main browsing screen:

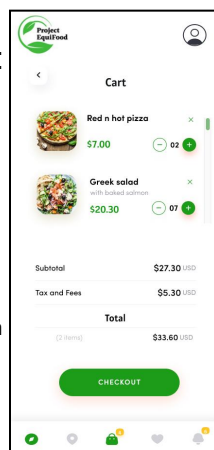


Includes a block style scroll with a search bar. Options at the bottom include navigation to main screen, location selection, Cart, Favourites and, Notifications.

After selecting an establishment, this screen displays their available products.

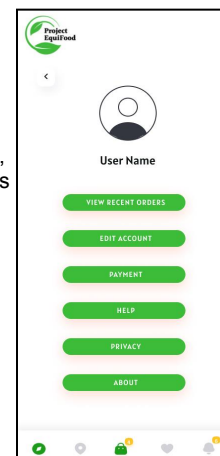


Checkout screen:



This screen displays the users cart and selected items with confirmation to reserve them for pickup. (all payment is handled at the establishments.)

Personal Profile:



The user profile page allows payment methods to be added, account changes, recent orders and more

We chose a very minimalistic design as this is more suitable for a greater range of users.

Upon further discussions with our client, we understood our client invisions this app as more of a dashboard rather than an interactive app.

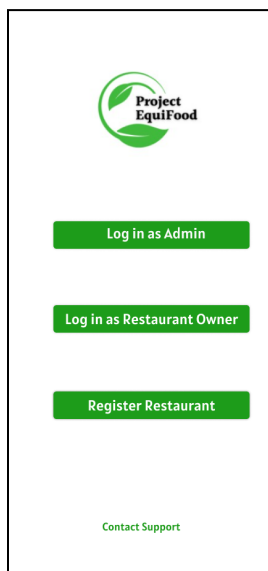
Prototype Version 2

For our second prototype, we have maintained the loading screen, main browsing screen, and available products screen from Prototype Version 1. We have removed the checkout and the personal profile screen, and updated the Login screen as per our understanding of the clients requirements.

Changes include:

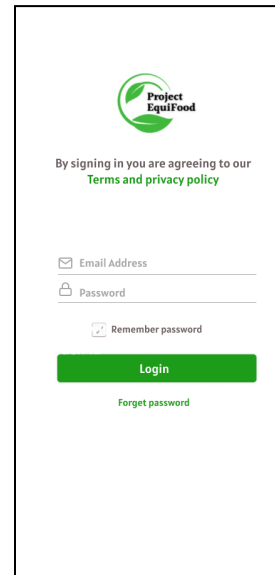
Login options screen that will display after the loading screen. allows users to select whether they would like to log in as Admin, or restaurant owner, or whether they would like to register their restaurant.

This page offers the option to contact support if necessary.



The mockup shows a white screen with the 'Project EquiFood' logo at the top. Below the logo are three green buttons: 'Log in as Admin', 'Log in as Restaurant Owner', and 'Register Restaurant'. At the bottom, there is a green link that says 'Contact Support'.

Updated Login:



The mockup shows a white screen with the 'Project EquiFood' logo at the top. Below the logo is a line of text: 'By signing in you are agreeing to our Terms and privacy policy'. Below this are two input fields: 'Email Address' and 'Password'. There is a checkbox labeled 'Remember password'. Below the input fields is a green 'Login' button. At the bottom, there is a green link that says 'Forget password'.

Our app will also include a Restaurant Owner view and an Admin View with similar layout and design of our prototype. We will create these upon further clarification from the client.

Questions we have for the client that we will be clarifying in the next meeting are:

- Will restaurant owners be manually updating the quantity/ food selection they have available?
- Will individuals have the option for reserving a food item for ~ 30 mins. (This would be a very helpful functionality as individuals will feel that it is worthwhile to commute to the restaurant IF it is guaranteed that there will be food available.)
- Would you like for the donations to also be visible on the front end?

Testing

We will be using jest to verify the system behaves as expected by testing for various inputs for expected outputs.

To unit and load test our API Endpoints we will also be conducting tests using Postman.

Once all the functional requirements of a milestone are complete, we will adopt a non-functional testing approach to verify that the application performs well.

The UI, functionality and performance of each feature will be tested using emulators like android studio.

A UX and a technical review will be required before every merge to master.

Integration Testing

We will know, from our functional requirements, what the expected output for every input is, therefore we will be writing tests while treating our system as a black box and ensure that all the individual components of our App are integrated properly.

Continuous Integration

To ensure continuous integration and deployment throughout the development cycle, we will be using GitHub actions to set up a CI/CD pipeline.

We will also be setting up staging and a production server. We will first merge our code changes to staging to ensure that our app doesn't break, if successful, the code will be shipped to production.

Regression Testing

We will set up pre-commit hooks to ensure that all tests pass and the app builds before commits are staged. This will make sure that all our test cases pass before our changes are merged into production/staging and minimize the chance of breaking the app

Peer Evaluation Question Responses

Q: How are you handling payment methods on your app? Are you using a third party?

A: All transactions will be handled by the restaurants and patrons directly. This application is in no way a business and will handle zero transactions. The purpose is to simply link potential customers to the available establishments.

Q: How will the platform profit if no payments or transactions are made on the platform?

A: The Equifood app is not for profit and is in no way a business therefore no payment or transactions will be handled by the application

Q: How exactly do you plan on stress testing using postman? Does postman have a built-in feature for multiple web requests?

A: Postman's Collection Runner can be used to load test an API by specifying a large number of iterations and running a number of such collections in parallel.

Q: In your presentation, you mentioned transactions/donations, how do you plan to process those while maintaining security?

A: As stated above, no transactions will be handled through the application. As for donations, when a meal is purchased its sale price will be subtracted from its retail sale price. This difference will be the "donation" and will be added to the running total.

Q: Is the food inventory done through real time updating.

A: Yes, restaurants will be incharge of updating their total available stock each day. When a meal is reserved and collected it will be deducted from the total stock.

Q: In case there is a large number of users that use the app at night, would the database be able to handle these multiple requests?

A: MariaDB/MySQL can handle at default 150 connections simultaneously. As connections will be completed and terminated quickly we do not expect this number to be reached.

Q: What steps will be taken to ensure the hygiene of the restaurant's leftovers?

A: All affiliated establishments are expected and liable to conform to all Worksafe and Food safe laws and guidelines.