High Level Description - Ana

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 types of users:

1) The Individual:
   the individual is able to log into the app, connect with a restaurant and obtain food
2) The Restaurant:
   the restaurant is able to post donations on the app and manage their information
3) Administrators
   the admin team approves of restaurants and views donations amounts

Functional Requirements (minh)

- Use Case - All Users
    - Create an account
    - Login to the account
    - Edit account details
    - Enable Notifications

- Use Case - Individual
    - View available donations
    - View donation details and price
    - Reserve / remove one or more items from current cart
        - Has to be from the same restaurant
    - Order review
    - Display pickup location on map
        - Not a delivery app! So no need for tracking
    - Confirmation
        - Call restaurant option
        - (Reservation countdown)
    - Favorites (important for notification)
    - (No payment system! The user will pay in person as the system only provides a way to notify them that there is leftover. Therefore the app is not responsible for any refunds as well)

- Use Case - Restaurant
    - Post donation (description, thumbnail, donated price, actual price)

        - Edit donation
        - Remove donation
            - When cancel, the user's reservation will be cancelled as well
        - Mark donation as complete

- Admin View
    - View pending restaurant details and contact information
    - Approve / Deny restaurant managers
    - Remove existing restaurant from app
    - Stats
        - View donation amount from each restaurant
        - View total donation amount

Milestones (minh)

Milestone 1 - 12/10 and 14/10

- Requirements Presentation
- Wireframes

Milestone 2 - 23/11 and 25/11

- Peer Testing #1 + Video Demo #1
- Individual's Use Case
- Authentication (Complete)

Milestone 3 - 1/3 and 3/3

- Peer Testing #2 + Video Demo #2
- Restaurant's Use Case (Complete)
- Admin's Use Case (Complete)

Milestone 4 - 12/4 and 14/4

- Final Report + Final Video
- Maps
- Notification Settings
- Profile Management
- Code polishing

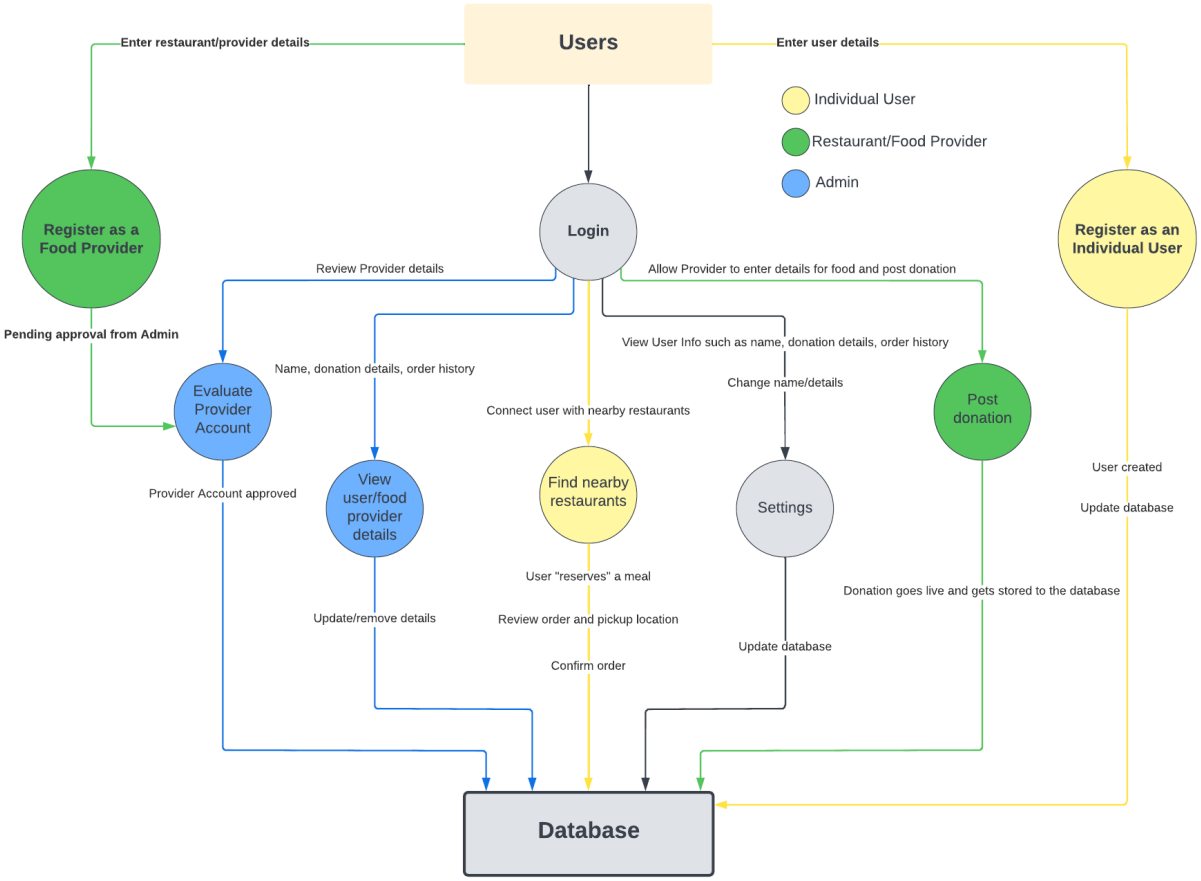
Non-Functional Requirements

- User friendly - mobile first design, intuitive interface, buttons, headings, etc.
- Responsive – users should not have to wait a long time for data to be processed.
    - Load dashboard within 10 seconds after opening the app
    - Refresh new donations every 5 minutes
    - Email confirmation should be sent within 10 seconds after reserving item
- Cross platform - Available on both iOS and Android with a maintainable code-base.
- Scalable - Back end should be able to scale up if there's a surge of new users.

- Performance - should be able on recent phone models, for iOS then it should run on the Iphone X towards the latest version.

Level 0 DFD (ana)

Level 1 DFD (neilansh)



- Here is the Level-1 DFD, showing the processes that run after an action is executed while using the app.
- As you can see, there are 3 types of users: Individual, Food Provider and Admin.
- Registering as a Food Provider/Restaurant would require further approval from the Admin.
- After logging in different options are provided depending on the type of user.
- The database will be storing information such as user details, donation details and history to name a few.
- We have defined our milestones based on the features here. So each milestone will cover one or more of the processes listed here.
- Moving on to the next slide, we'll look at the Tech Stack.

Tech Stack (neilansh)
(Make sure to have the table in the slide)

- Three options: React Native (framework), Flutter (SDK) and Ionic (framework).
- All 3 tools are cross platform and therefore support iOS and Android.
- As the table suggests, some important considerations taken into account were: performance, how developer-friendly it is, reliance on 3rd party APIs and memory usage.
- 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. Ionic supports live loading so it executes the whole app every time to test small changes, so development could slow down.
- 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 Flutter features that appealed to us:
    - Easier to learn.
    - Developer-friendly, and is 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. We will also be making use of the **Google Maps API**.

Testing Options (eunsuh)

https://docs.flutter.dev/testing

- Continuous integration testing (Flutter test library & github action workflow)
    - Unit testing
        - For all code beside the UI widgets
        - One set of uni tests usually test a single class
    - widget test (riverpod)
        - For testing a single widget
    - integration test
        - For testing large parts of app from the user perspective
- Regression testing (flutter: golden_toolkit package, reset-all)

For the continuous integration testing, our software developer kit, Flutter, has an atomatiated test library called flutter_test. Flutter breaks down testing into 3 stages; unit test, widget test and integration test.
To start, the Unit test checks to see if there is any error thrown at the coding phase. It usually tests a single function, method, or object. For unit tests, it could test a mock up user, before the actual user registers into the system.
Widget tests check if the widget's UI shows up correctly corresponding to what it is supposed to show up and we will also use Riverpod for the widget test [to help?]. For example, if a user clicks a restaurant to see more details, the widget test will check that it shows the right restaurant's information.
We then have an Integration test which is the testing of the user's perspective, the entire flow of action, design and redirection performed. For example, registering as a new user, choosing a restaurant based on their location on a map, and reserving a food item using the app will be counted as integration testing.
Regression testing is used to make sure that when new code entered does not break or disable anything we were working on beforehand. This is overall important to our project because we require continuous updates and improvement to our code and work. [Plan to avoid break down of code]

For the continuous integration testing, our software developer kit, Flutter, has its own atomatiated test library. Flutter breaks down testing into 3 stages; unit test, widget test and integration test. Unit test done in coding phase to make sure there isn't any error. Flutter test library and github action workflow will be used to do the unit test.
Widget test will be done using flutter widget testing and riverpod. In this stage, we are testing UI components so the widget works as expected.
For the integration testing, Flutter integration testing library and github action workflow will be used same as unit test. Integration test is testing a big chuck of an app from the user's perspective to see if the entire flow of the app is working.

In the regression testing, 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 improvement to our code and work.