Orbital Splashdown READ.ME

Team Name	NullPointerException
Team ID	5193
Project Level	Apollo 11
Team Members	Aarnav Srinivasan Vaidyanaath Suresh

Motivation

Ever been confused about what to eat and where to eat inside NUS? You might be craving something or you might simply want to have a decent meal at stalls without breaking the bank. There are not many people you can ask to get to know everything about food at NUS. And it is practically impossible to visit each and every food stall/restaurant and gather the required information. Is there a way to get all this information easily?

Aim

We hope to provide real time menu data and availability status of food from all food-courts/ stalls in NUS, with additional features and information, through **our mobile application** Eat@NUS.

User Stories:

1. As a customer who wants to explore food options at NUS, I want to know the locations of all the food stalls/restaurants and all the available items in the menu in any stall.
2. As a customer, I want to be able to search for any food stall or dish, and filter them based on cuisine, price range, etc.
3. As a customer who has decided to buy a certain dish, I want to be notified about the availability of the dish in real time.
4. As a customer looking to try any dish, I should have the information on the ingredients and any allergens present in the dish and the calorie count, so as to make an informed decision.
5. As a customer who wants to provide feedback on food I have had, I want to be able to write a review about the quality of food for other customers to read.
6. As a food stall owner who has recently opened a stall, I should be able to create a new profile and update the display information.
7. As a food stall owner, I want to be able to update the opening and closing times of the stall so that it is reflected on the customer side of the app.
8. As a food stall owner who wishes to change menu items, I want to be able to edit, refresh, add food items and update their price and availability.

Scope of Project:

A **mobile application** that provides real-time menu data and availability of food across all food stalls inside NUS. The app has several features for both customers and food stall owners.

Customer features:

1. The **Search and Filter** feature enables customers to search for food stalls by name or search for food stalls that serve a particular dish. The customer can also filter the results by Cuisine, Price Range and Distance.
2. The **View Stall/ Dish Info** feature: The app displays any particular food stall's information including location, operating hours, menu items and its rating. It also displays the real-time availability, price, allergen information, calorie count and rating of a selected dish.
3. The **Write a Review** feature allows the customers to write reviews about the dishes.

Stall Owner features:

1. The **Edit Stall Info/ Menu Info** feature: Stall owners can edit the general information about their stalls, like location, operating hours. They can also add/ delete items in the menu.
2. The **Edit Dish** feature enables stall owners to edit the information of a particular dish, like name, price, description, allergen information and calorie count.
3. The **Toggle Dish Availability** feature allows stall owners to update the availability of a dish in real time, thus saving time for customers.

Additional Non-Functional Requirements:

1. **Latency:**

   Users will not like to wait to see the search results of food stall details and dishes, so the search functionality should be very fast.
2. **Availability:**

   High availability is desirable for the best customer experience.

Tech Stack:

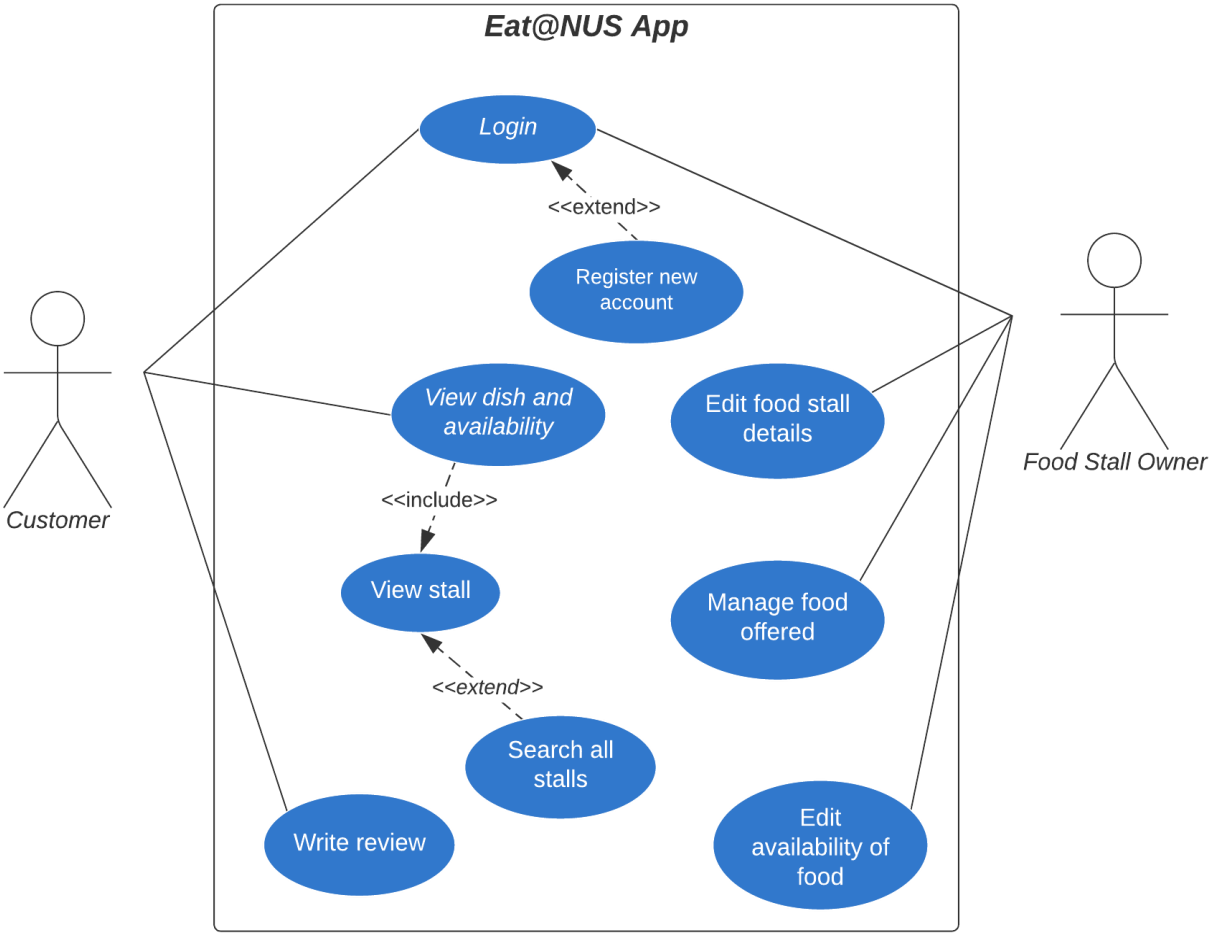
Tech	Remarks	Why did we choose this?
React Native	A cross platform javascript framework that will be used to develop the frontend of our mobile application	Single codebase for IOS and Android (we want our app to be used by all students at NUS) Code reusability + Large community support
Firebase Realtime Database	A NoSQL cloud database that will store data about the stalls, dishes, and their availability	Data is synced across all clients in Realtime (one of the main requirements of the app is that changes in the availability of a dish are immediately reflected on the customer-side) JSON data returned by the database is also easy to use
Firebase Cloud Storage	Cost effective object storage service to store the stall and dish images	Good integration with Firebase Realtime Database Strong security: has a declarative security model to control access
Jest	Jest is a testing framework built using JavaScript that will be used for system testing and unit testing	Has snapshot testing support: renders UI components, takes a screenshot and compares a recorded screenshot with changes made. Fast integration: tests are run in the command line. Easier to debug

Timeline:

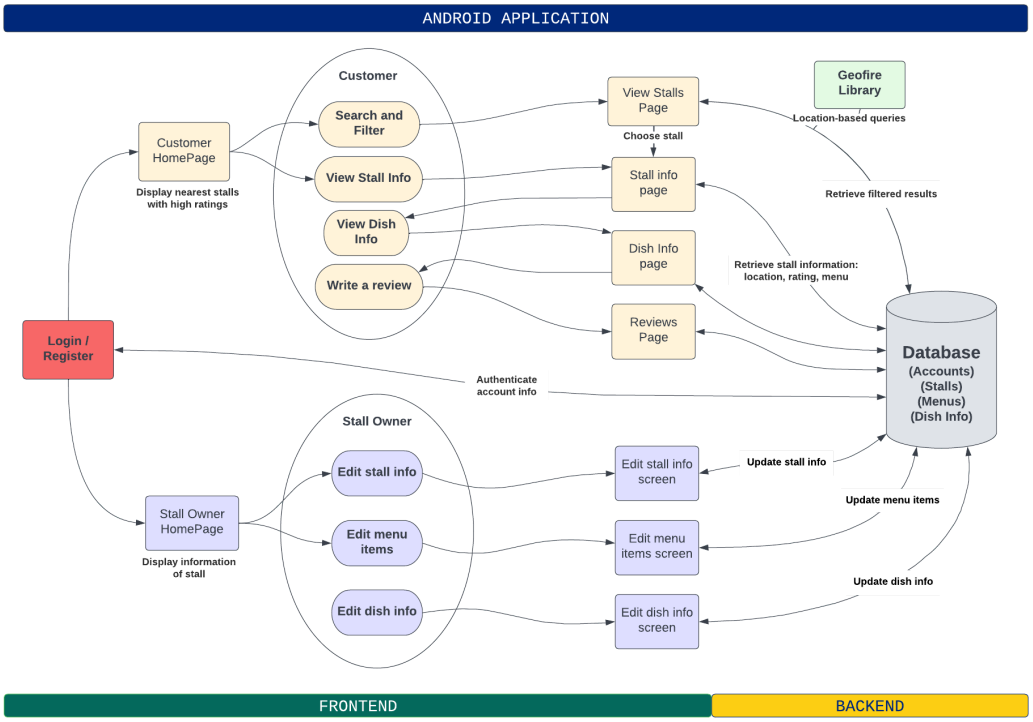
Features that are completed (as of MS3):	Additional features that were removed due to time constraints (but can be implemented in future updates):
1. View Stall/ Dish Info —---------- MS2 2. Edit Stall/Dish Info 3. Edit Menu Items + toggle availability 4. Search and Filter results 5. Write a review —---------- MS3	1. Bookmark feature 2. Notification feature

Use Case diagrams:

Based on the user stories, we created a use-case diagram below that illustrates how the users of the Eat@NUS app might interact with the system.



Technical Proof of Concept:



[Click here for Google Drive link of high quality image]

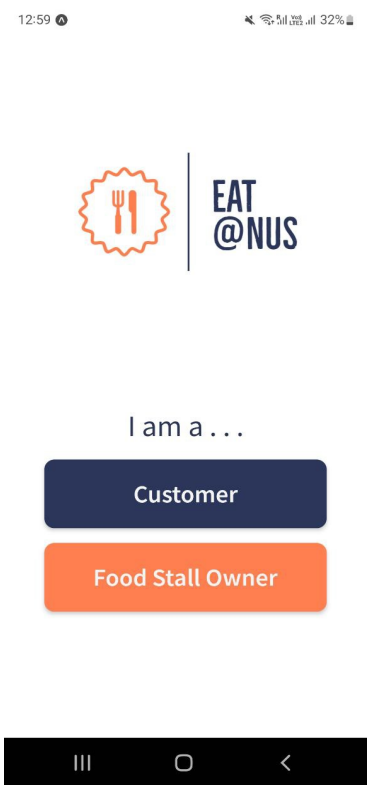
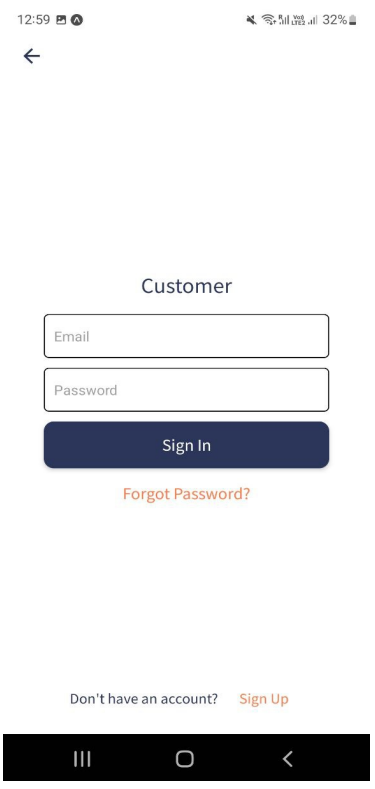
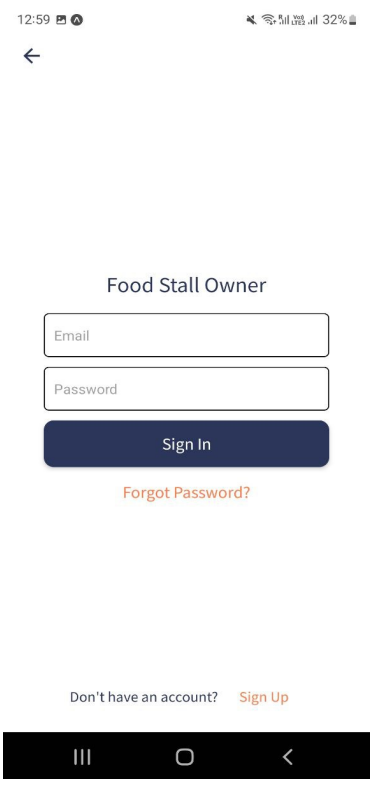
Explanation:

Eat@NUS is a mobile application built using React Native. The app has different features and interfaces, for customers and food stall owners.

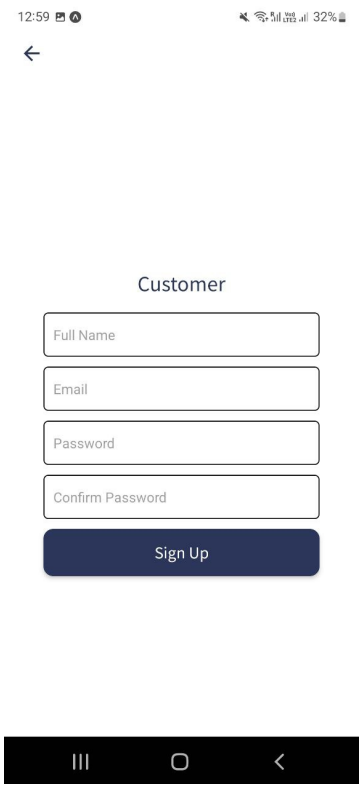
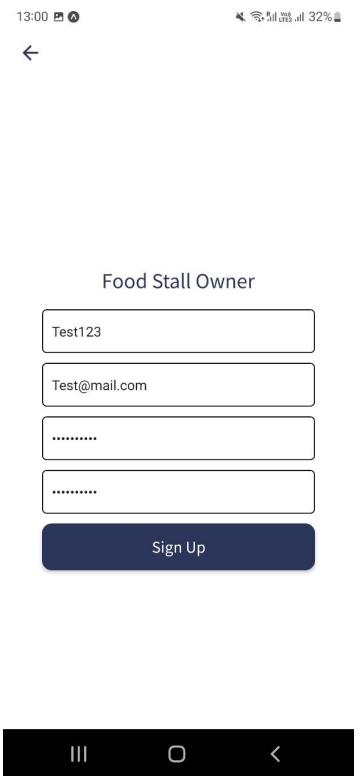
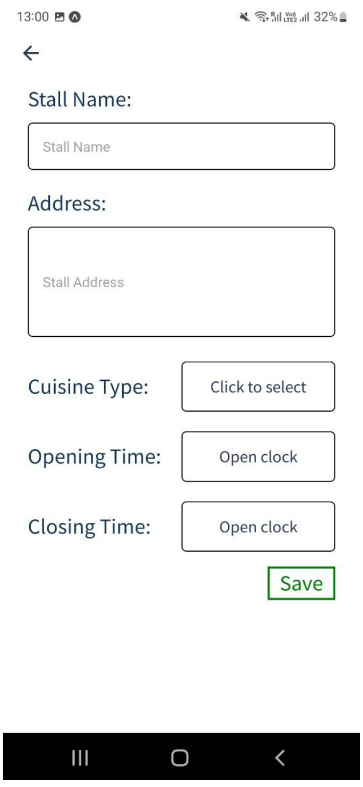
The customer is able to access all the stalls and popular dishes on the homepage. The customer may choose to search for a particular dish or stall and filter results based on several metrics. The search and filter queries are performed by the firebase database and data is returned to the app in the form of a json object which can be rendered appropriately. On clicking a stall button, the customer is brought to the stall screen where there is a list of dishes along with their information as well as availability. Clicking on a dish brings the user to the dish screen which displays more information about the dish.

Mockups and UI Prototype

The **Landing Page** displays the name of the app and contains two buttons: customer and food stall owner. The user is expected to select their role. When pressed, this leads to the **sign in screen** where the user enters their credentials. There is a link to the sign up page which will take the user to a form. The **forgot password** link sends a password-reset link via the registered email to the user. After verification, the users are guided to their respective homepages.

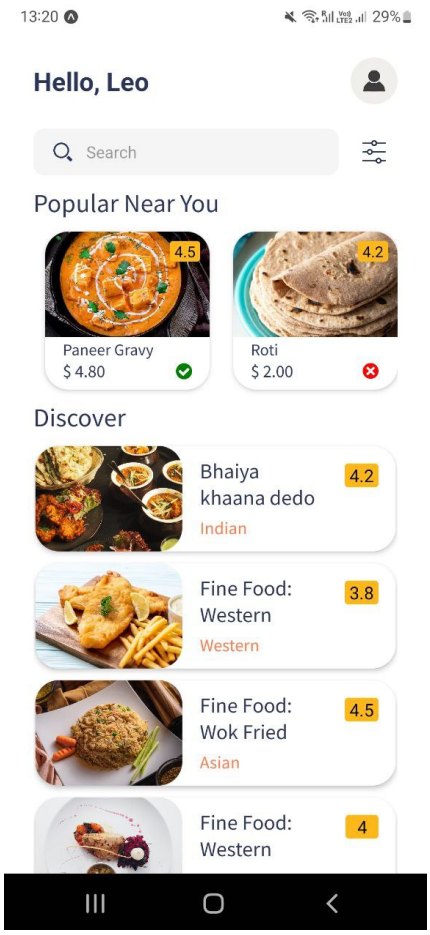
LANDING	CUSTOMER SIGN IN	FOOD STALL OWNER SIGN IN
		
• Name with logo • 2 buttons (touchable opacity components) • User chooses their role	• Text input fields for email and password • Sign in button (Integrated with firebase auth) • Forgot password button • Button links to Sign Up page	(Same as customer sign in page, except this screen checks if your account is registered as a food stall owner)

The user is expected to enter their details such as full name, email and password to **register a new account**. If the user is a food stall owner, he/she has to additionally **register the stall** by entering its details. The **cuisine type** is selected from a dropdown menu of available options. The **opening and closing times** can be selected with a datetime picker that pops up when the placeholder button is selected. (Refer to video submission for demonstration). *For security reasons and to prevent spam, food stall owners will be asked to present their SFA licence when our admin requests*.

CUSTOMER SIGN UP	FOOD STALL OWNER SIGN UP	FOOD STALL OWNER REGISTER STALL
		
• Text input fields for full name of user, email address, password and confirm password • Sign Up button • Signing up successfully leads to customer homepage	• Same as customer sign up • After sign up, there is an additional step to register stall details	• Text input fields for name of stall and address • **Dropdown select** to pick cuisine type • **DateTime Picker** to select operating hours

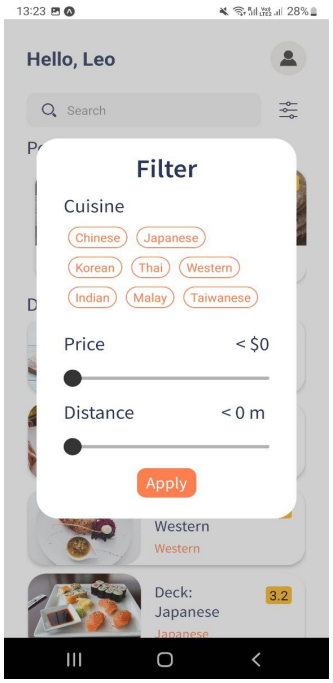
Customer-side features and interfaces:

The **Customer Homepage** displays a list of popular dishes (dishes with the highest ratings) and a list of stalls, retrieved from the database. The name of each popular dish, its price, rating and availability can be clearly seen from this screen. Clicking on a popular dish takes the user to the dish screen (displayed later). The names, cuisine types and ratings of the stalls are also shown for the user to explore according to their tastes. Clicking on a stall takes the user to the stall screen.



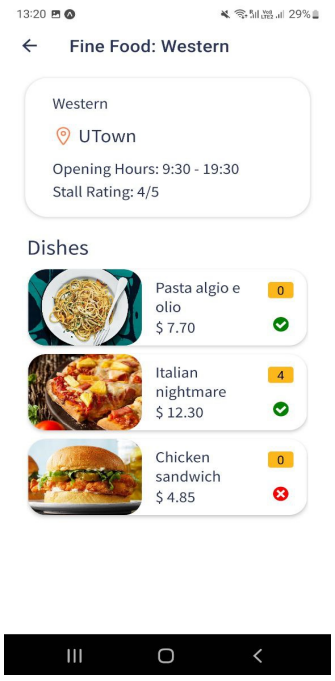
- Greeting message
- Sign Out Button (Top Right - Profile icon)
- A horizontally scrollable display of the most popular dishes
- A vertically scrollable display of the stalls for the user to explore.
- A search bar where the user can search for any specific dish or stall.
- Filter button

The **Search and Filter** feature allows the user to search for a stall by its name or the list of stalls that serve a particular dish. The user can also filter the search results by Cuisine, Price Range and Distance.



- A textbox to allow user input (Search Bar)
- The filter window contains tags for Cuisine types and two sliders, one each for Price Range and Distance.
- A button, which when pressed, takes the user input and the selected filters to obtain desired results using Firebase.

The **View Stall Info** feature allows the customer to view information about a stall, like the location, operating hours, customer rating and menu items.

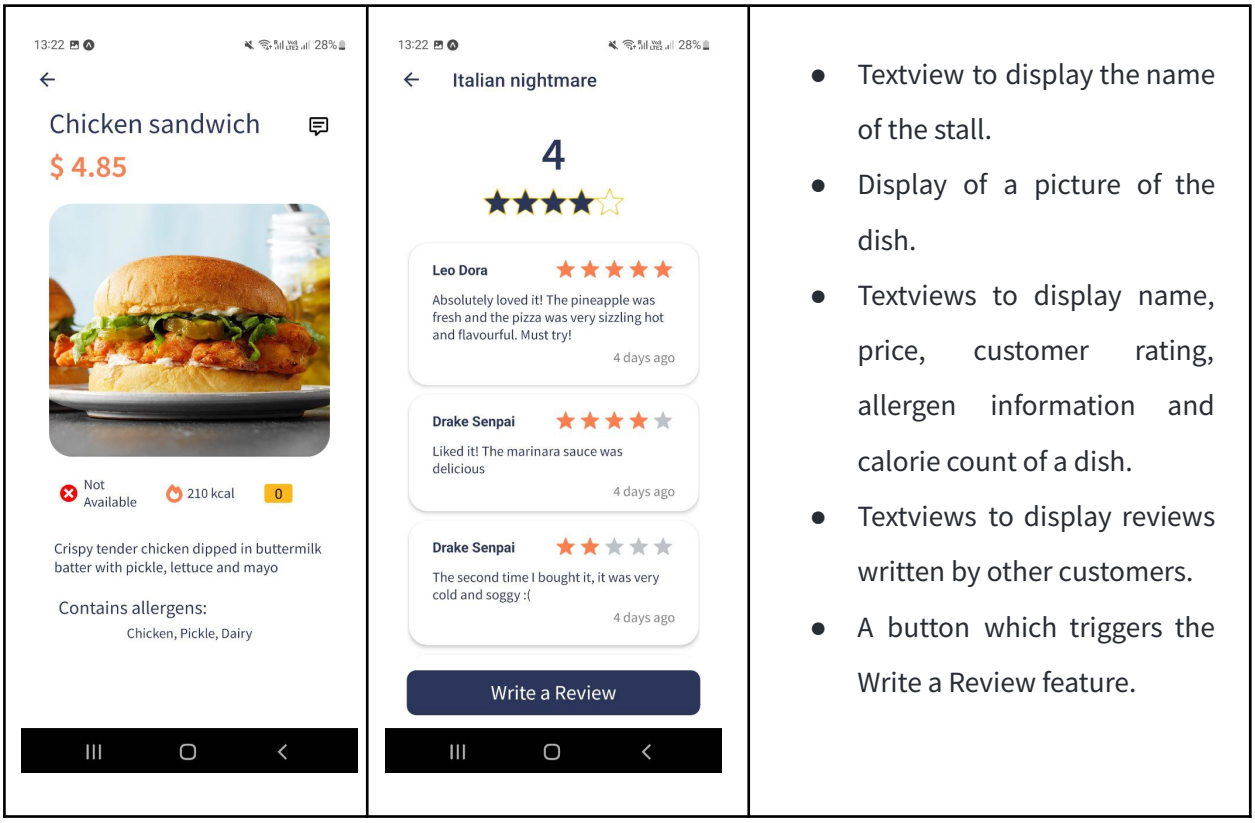


- Textview to display the name of the stall.
- Display of pictures of the stall.
- Textviews to display the general information of the stall.
- Textviews to display names and prices of dishes in the menu, which when clicked upon, triggers the View Dish Info feature
- Green tick indicates that the dish is available, red cross means that the dish has run out or is unavailable

The **View Dish Info** feature allows the user to view information about a particular dish served in a particular stall. The user can view a picture of the dish, the real-time availability of the dish, and in addition to it, get to know of any possible allergens present in that dish and the approximate calorie count. The user can also view the rating and read reviews of the dish written by others.

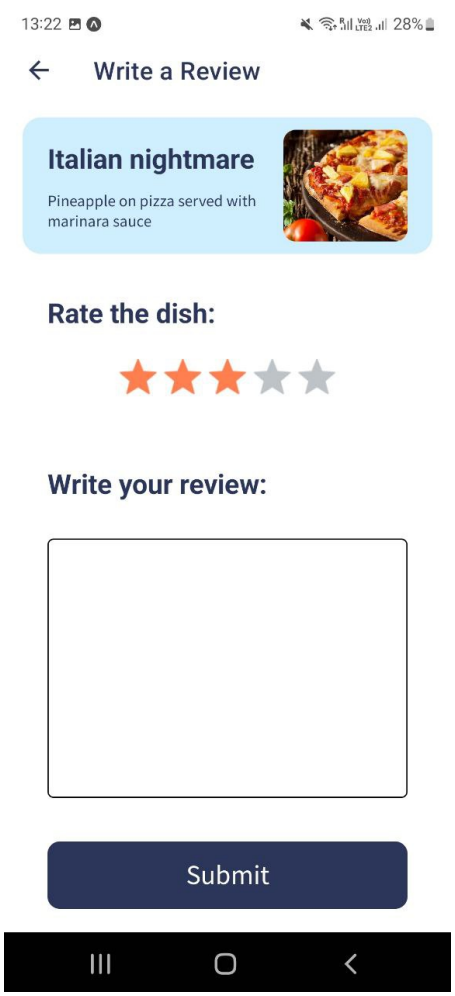
Clicking on the “comment-box” shaped button on the top right of the page leads the user to the **reviews page** where he/she can read reviews left by other users. The time since the review was written is also provided to give the user a better idea of how the food tastes now.

The user can write their own review of the dish by clicking on the write a review button.



- Textview to display the name of the stall.
- Display of a picture of the dish.
- Textviews to display name, price, customer rating, allergen information and calorie count of a dish.
- Textviews to display reviews written by other customers.
- A button which triggers the Write a Review feature.

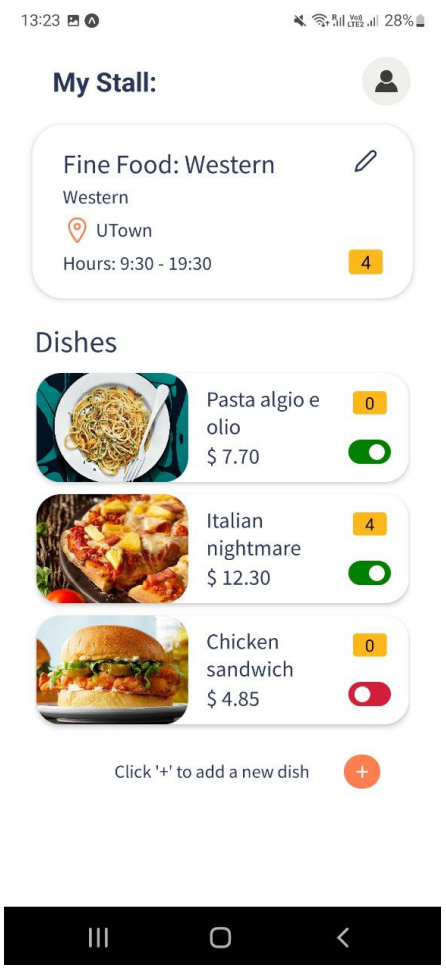
The **Write a Review** feature allows the user to share his/her thoughts about the dish.



- Panel to show the dish being reviewed: Name, description, image
- Star rating (default value = 3)
- Textbox to allow user input to write a more detailed review.
- Submit button adds the review to the list of reviews under this dish and the dish rating is subsequently adjusted.

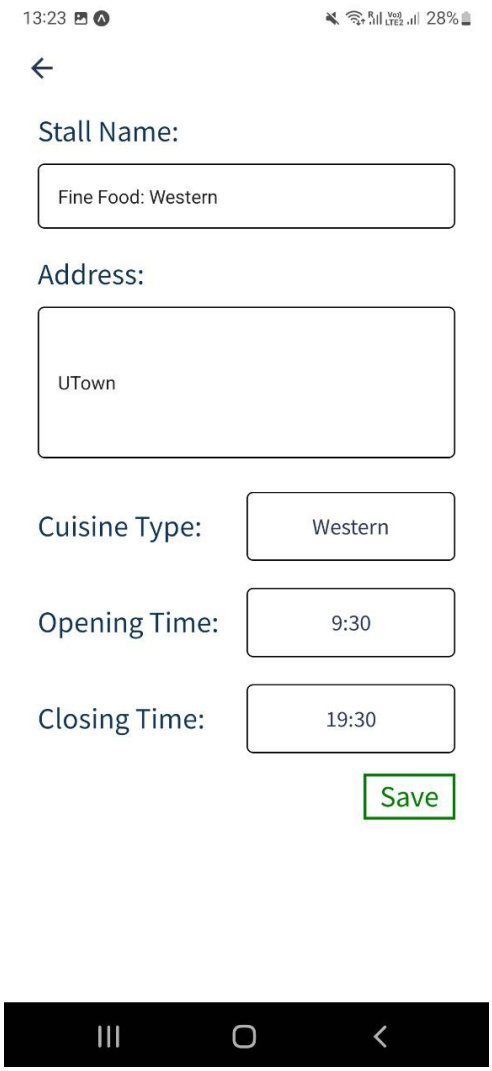
Stall Owner-side features and interfaces:

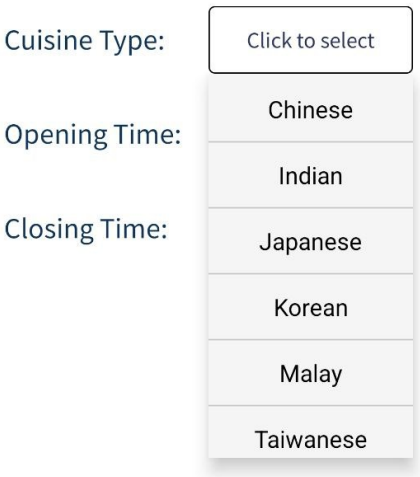
The **Stall Owner Homepage** displays all the information about the food stall associated with the account used to login, pulled from the database. The displayed content includes general information about the stall (like location, customer rating, operating hours, etc.), and the menu.

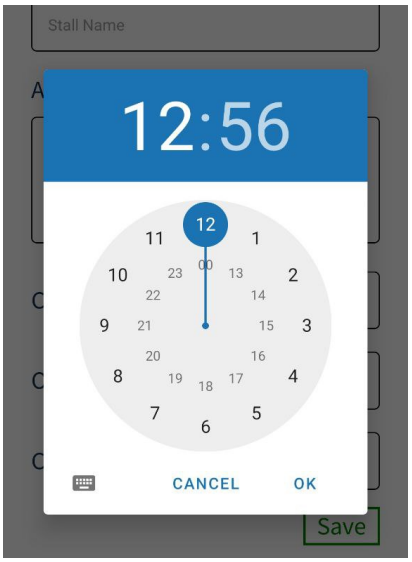


- Textviews to display stall name, customer rating of the stall, stall location, operating hours.
- A button to trigger the Edit Stall Info feature.
- Buttons to display the dishes in the menu of the stall, which when clicked, navigates to the dish screen with more detailed information about the dish.
- 'Add dish' button at the footer of the list, which when clicked navigates to the add dish screen.
- Switches to toggle availability of a dish.

The **Edit Stall Info** feature enables the stall owner to edit the general information of the stall. Any changes made will be reflected in the Customer-side interface of the app.

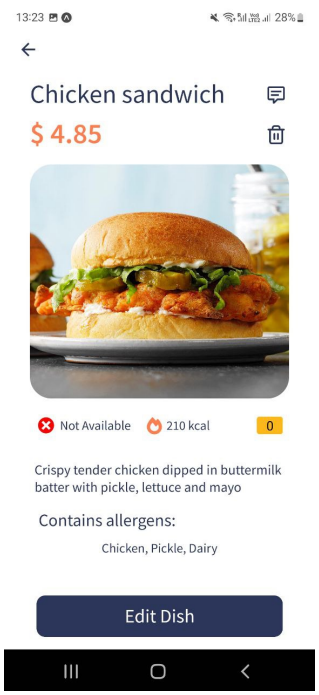


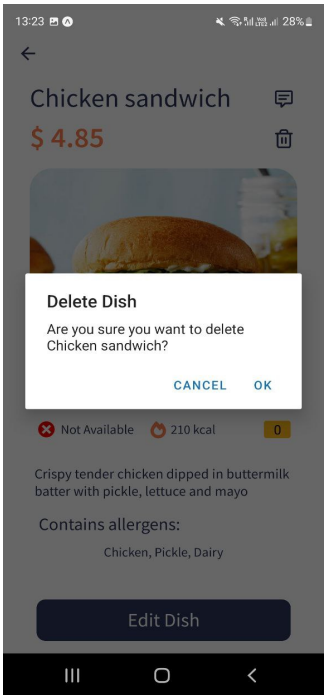




- Textboxes with the pre-existing data, to allow user input.
- Same text inputs and dropdown to select cuisine type, datepicker to select time.
- A button, which when pressed, saves the changes to the database, through Firebase.

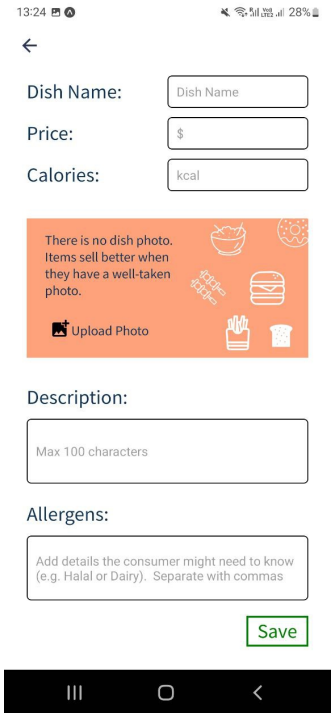
The **Edit Menu Items** feature enables the stall owner to add/delete items to/from the menu. Any changes made will be reflected in the Customer-side interface of the app.





Delete Items

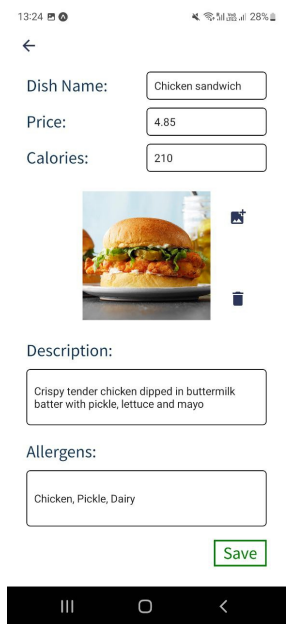
- Reach the dish screen by clicking on the dish from the homepage
- A delete button (trash icon), when pressed, shows an alert and deletes all information about the dish from the database.



Add Items

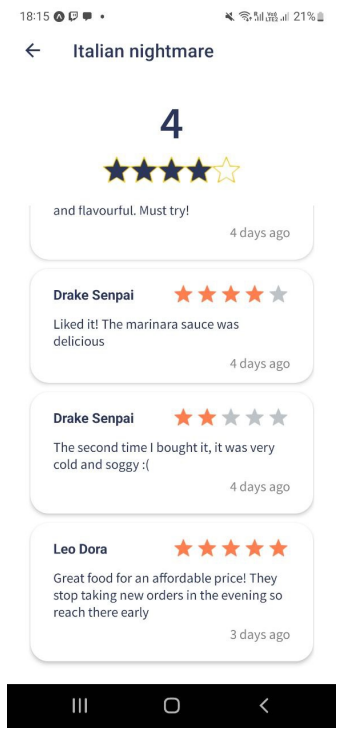
- Textboxes allow users to input dish name, price, calorie count, description and allergen info.
- Picture input for the stall owner to add a picture of the dish.
- A button which when pressed, saves the data to the database, through Firebase.

The **Edit Dish Info** feature enables the stall owner to edit the information about a particular dish, such as the name, price, allergen information and calorie count. Any changes made will be reflected in the Customer-side interface of the app.



- Identical to add dish info screen but the data of the dish being edited is loaded into the input fields.
- Edit dish image button
- Delete dish image button

The stall owner will also be able to read the reviews for the dishes written by customers in the **View Dish page**.



- Textviews to display customer reviews, pulled from the database, through Firebase.
- Unlike a customer, a food stall owner cannot write a review.

Problems Encountered

1. **KeyBoardAvoidingView:**

   Encountered a situation where the keyboard to enter the user credentials kept covering the input fields when they were clicked.

   We solved the issue by creating a custom container component using KeyBoardAvoidingView and modified the style for both IOS and Android. This led us to create custom components for all the other containers too enabling code reusability.

2. **Loading fonts asynchronously:**

   Encountered an error when I tried to use custom fonts. The screen component was being rendered before the fonts could be loaded. Sometimes, this kept crashing the app and raising an error stating 'Fonts not loaded with loadAsync'.

   The issue was resolved after using the useFonts hook provided by expo.

3. **Outdated dependency used by expo modules:**

   Received error messages indicating that the app was using the deprecated version of asyncStorage library even though none of the dependencies required it. Later, we realised that the expo modules were using them.

   Could not be solved even after updating the expo-cli to the most recent stable version.

4. **Database Structure:**

   Only having had experience with SQL databases, we initially implemented a structure for the database where all information related to a stall or dish was stored under its name/ id. But this decision raised performance issues since all the child nodes of a stall were fetched from the database even when we required only a single attribute like name.

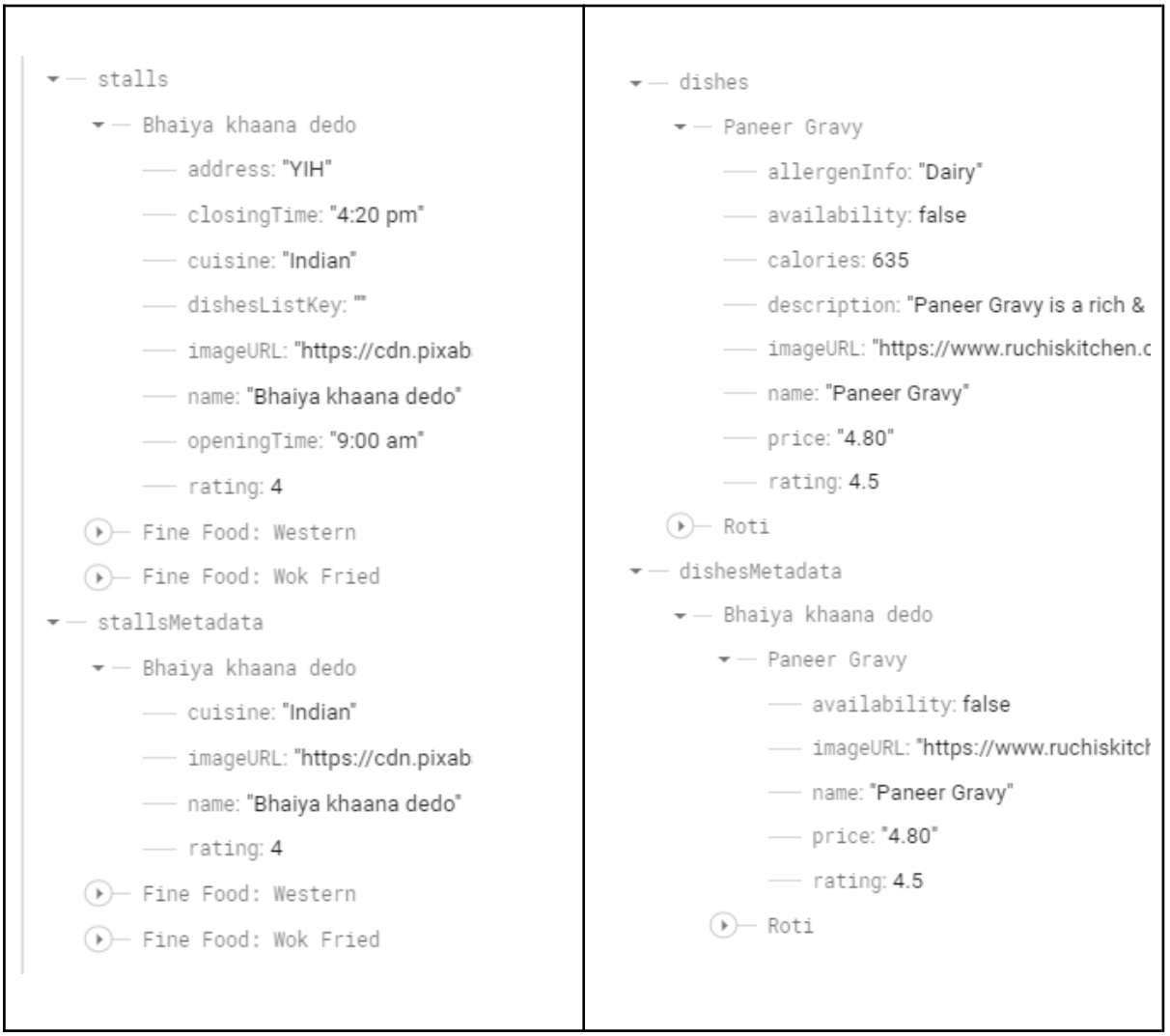
We restructured the database prioritising performance. We decided to duplicate the data (duplication is not bad in NoSQL) and separate it into metadata (some information that must be shown) and the actual data (contains the same info as metadata and all the other data).

Data Structure Design

On the customer homepage, only some parts of the information about each stall are required (name, rating, cuisine type). So we made a separate node that stored this information and also had an ‘id’ key which could be used to access the same stall in other nodes. We called this the ‘stallMetadata’ subtree.

The same id can then be used to access more information about the stall under the ‘stalls’ subtree when the user then clicks on the button. This prevents firebase from fetching long lists of data unnecessarily. The same idea was adopted for showing dish info: data was duplicated into ‘dishMetadata’ and ‘dishes’.

The example below with dummy data displays how our realtime database looks like.



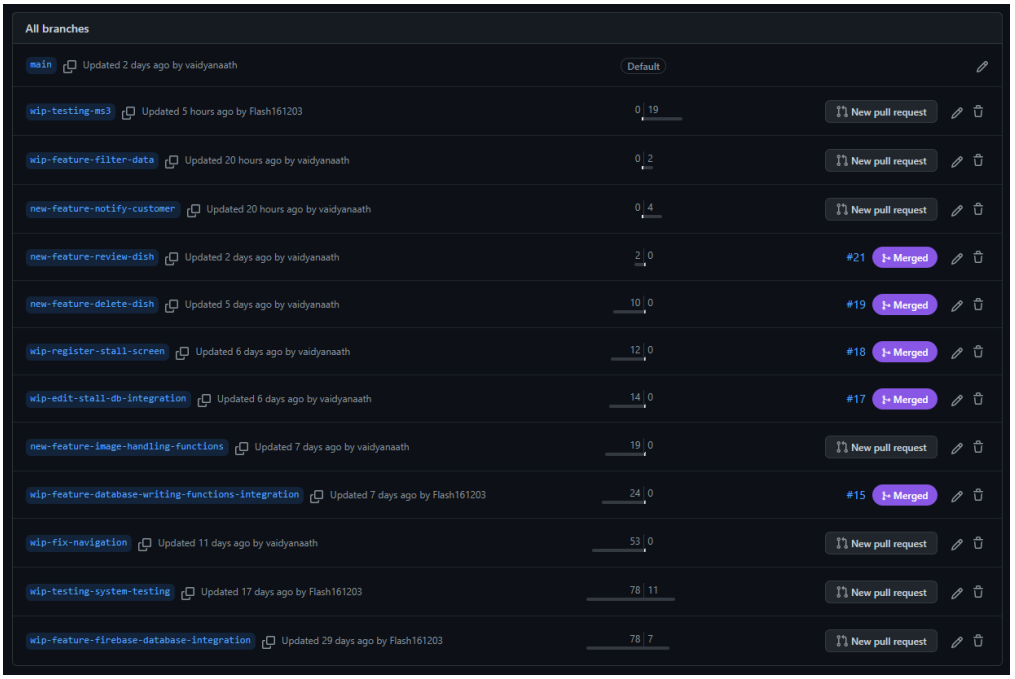
Software Engineering Practices

1. Version Control

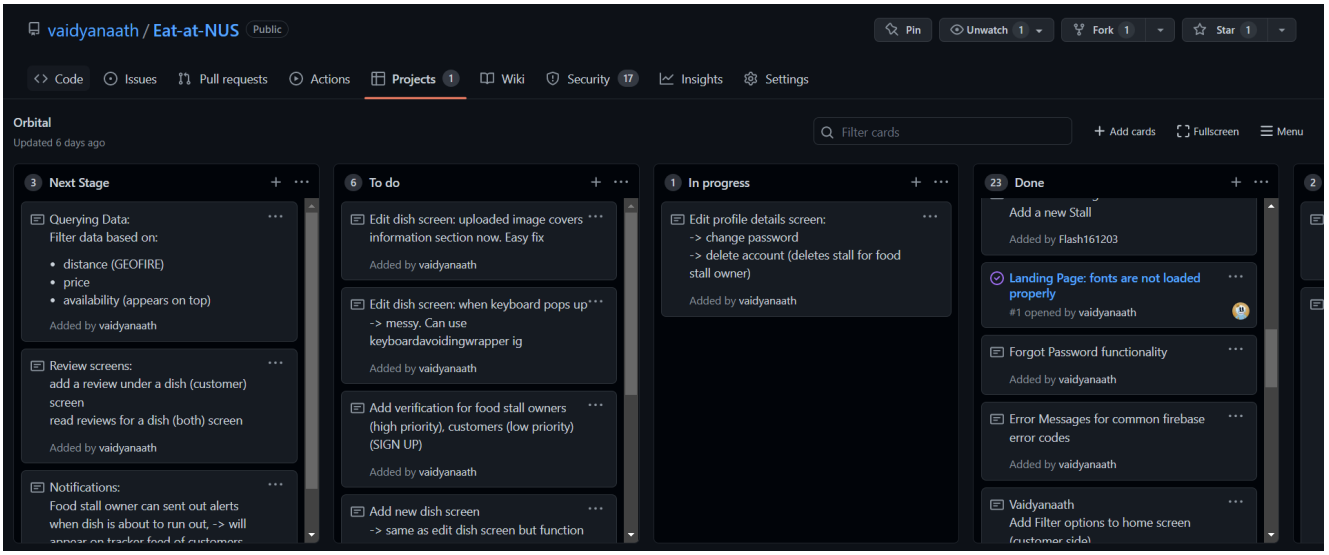
Github

- **Branching**

  Each new feature, improvement, or set of bug fixes was developed in a separate branch that followed a strict naming convention. Once the feature is complete, a pull request was created and all the members of the team were added to the reviewers list. On manually checking to see if the build works without any errors, the pull request is merged with the main branch. This was done to ensure that the main branch contains code that is always up to date and runnable.



- **Project boards to organise and assign tasks**



Testing

ESLint (Static Analysis):

In order to catch common errors such as unused code and inconsistent styling, we made use of ESLint. It enforced a strict code style (Airbnb jsx style guide) and paired with Prettier, made it very easy to spot errors.

Component Testing:

Rendering (Snapshot Tests):

To check if the components render correctly, we performed snapshot testing on all our components. Here are the results for some of them:

```
PASS __tests__/screens/main/customer/Home.test.js
  <Home />
    √ Renders correctly (570 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   1 passed, 1 total
Time:        2.113 s, estimated 4 s
Ran all test suites matching /__tests__\\screens\\main\\customer\\Home.test.js/i.
```

```
PASS __tests__/screens/auth/CustomerSignIn.test.js
  <CustomerSignIn />
    √ Renders correctly (195 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   1 passed, 1 total
Time:        2.022 s
Ran all test suites matching /__tests__\\screens\\auth\\CustomerSignIn.test.js/i.
```

```
PASS __tests__/screens/auth/Landing.test.js
  <Landing />
    √ Renders correctly (84 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   1 passed, 1 total
Time:        1.571 s, estimated 2 s
Ran all test suites matching /__tests__\\screens\\auth\\Landing.test.js/i.
```

User Testing:

In order to receive feedback from users , we asked a few of our friends and family members to review the mobile application and point us towards any errors, bugs, or areas of improvement. We distributed the application along with a survey and the responses are summarised below.

First, we decided to test out the **customer side** of the application.

User	Proficiency	Target objective	Actions Performed	Remarks
Friend	Expert	Filter and search for dishes	Home → SearchBar: type “pasta” → Filter Button → Click on Western → Apply Filters	Both stalls and dishes with names including “pasta” appear. But the name of the stall/dish has to match exactly with the search string. For example searching for “past” yields no results.
Family Member	Novice	Register a new customer account + Sign out	Landing → Customer → Sign Up → Enter Details → HomePage → Profile Button	The process is mostly straightforward. The error messages for non-matching passwords, empty fields are clear. I suggest creating a profile screen or changing the sign out icon to something else.
Family Member	Expert	View details of the most popular dish	Home → First item on popular dish → Dish screen	Simple and intuitive navigation , clean UI for cards in the scrolling containers. Cannot see the restaurant a popular dish belongs to.
Friend	Expert	Write a Review for a dish	Home → Stall → Dish → Reviews → Write a Review → Submit a Review	The reviews page looks fine, the ‘write a review’ button is aptly placed (call for action). Very simple interface to write a review.
Friend	Novice	View a random dish from a random stall	Home → Stall → Dish → Dish screen	The app works as intended.

Since we were not able to distribute the application to real **food stall owners**, we asked the respondents to the previous survey to review the application from a food stall owner's perspective by giving them a testing food stall owner account.

User	Proficiency	Target objective	Actions Performed	Additional Remarks
Friend	Expert	Register a new food stall owner account	Landing → Food Stall Owner → Sign Up → Register Stall → Stall Owner Home	The error messages for wrong inputs are well handled and the UI is simple and easy to navigate.
Family Member	Novice	Delete a dish	Stall Owner Home → Dish Screen → Delete Button	The deletion is instantaneous. There is no lag or delay after deleting. Works fine.
Family Member	Expert	Edit Stall info	Stall Owner Home → Edit Stall Info	The UI is similar to the register stall screen as expected. It would be a good idea to add more cuisine types or even multiple cuisine types (eg. Vegetarian + Indian). The datetime pickers work well.
Friend	Expert	Edit Dish Info	Stall Owner Home → Dish Screen → Edit Dish Info	Simple and well-formatted. The placeholder for an empty image looks interesting (call for action too). Good image picking and display support. Dishes can possibly have tags like "Halal", etc.
Friend	Novice	Add a new dish	Stall Owner Home → Add Dish → Stall Owner Home → Dish Screen → Delete	It is good that the keyboard type changes to numeric for price and calories. Easy to use and create a new dish.
Family Member	Novice	Edit availability of dish	Stall Owner Home → Toggle Switch next to dish	Works as expected. But the switches are a bit too small in my opinion. Sometimes, it navigates to the dish screen instead of toggling availability.

Based on the feedback received, the following changes have since been implemented:

1. The sign out button with the avatar icon now navigates to a **profile screen** with dedicated 'sign out' and 'delete account' buttons.
2. **Filtering data**: search by distance and price range features have been removed due to inconsistent results. Instead, the search feature now supports **elastic string matching** to show results.
3. **Stall Images** can be separately uploaded by the food stall owner.
4. Minor UI design changes (e.g. save buttons)

Acceptance Testing:

This video shows a walkthrough of the application from the perspective of both customer and food stall owner: Video

Github Repo: https://github.com/vaidyanaath/Eat-at-NUS

(contains instructions for testing the app)