Mealer Project

Final Report

SEG2105[A]

Professor Hussein Al Osman

Members:

Rahul Atre (300250370)

Kristen Duong (300240425)

Amy Huang (300240777)

Pranav Kural (300241227)

Anjali Mohammed (300242701)

Justin Wang (300234186)

Table of Contents

Introduction

The Mealer project exhibits a meal-sharing application built on top of the Android application framework using some of the most efficient software design patterns and the highest standards of code quality.

Development of the project happened in four phases which were divided into four deliverables wherein each deliverable was focused on achieving a certain set of major application functions.

The Mealer app has three primary user roles: Chef, Client, and Admin. Some of the actions which could be performed by each of these roles in the app are as followed:

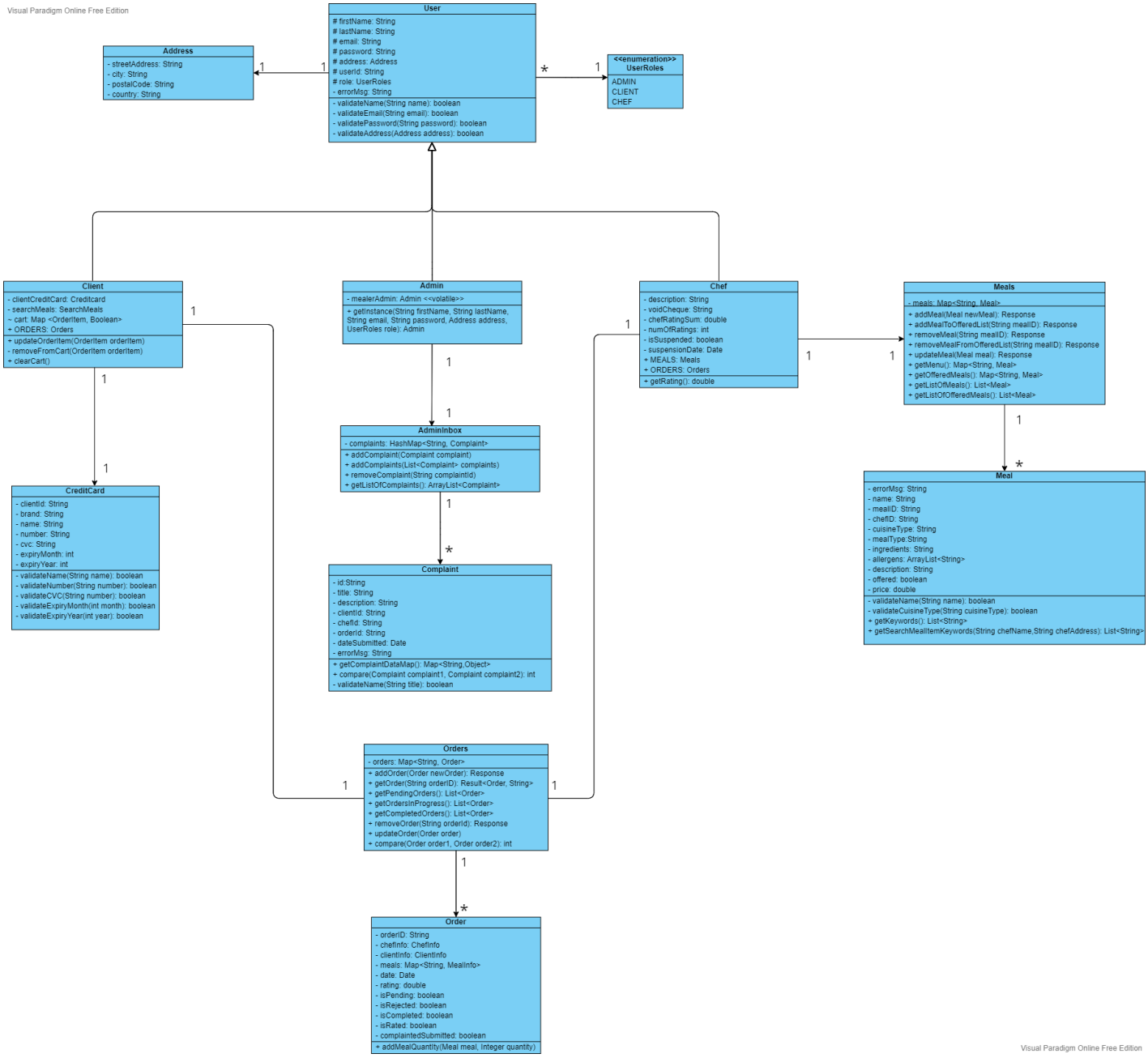
Chef: create a menu of meals; offer and sell to customers

Client: search for meals and request orders for pick-up

Admin: handle complaints filed against Chefs

Following the requirements provided for each deliverable, the necessary UI components were created and the data handling logic was implemented to build an efficient and high-quality android application. The following sections detail the breakdown of the application, the contributions of each member, and a summary of the knowledge acquired.

Final UML Diagram



Team Contributions

RAHUL ATRE

Deliverable 1

- Created model classes for Admin (implemented using singleton-pattern), Chef, and Client
- Designed XML for client additional information and assisted with sign-up screen
- Assisted in setting up Firebase authentication for login and sign-up

Deliverable 2

- Designed XML for the admin screen using an array adapter
- Implemented logic for admin's complaint screen
- Added functionality to dismiss a complaint from a client, thus removing it from the complaint screen
- Updated UML diagram to Deliverable 2 codebase
- General debugging and testing

Deliverable 3:

- Created JUnit Test cases for deliverable 3
- General debugging and testing
- Bug fixes and other UI improvements

Deliverable 4:

- Designed chef info XML file to display profile information to the chef, along with a profile button functionality

- Assisted in creating the design for the search screen on the client's side with the back button, search bar, and checkout button
- Added a rating bar UI to the Chef Info screen
- Created Firebase instances for menu items to display sample meals to the client
- Tested app from admin's perspective to ensure functionality for the demo
- General debugging and UI improvements

KRISTEN DUONG

Deliverable 1

- Designed XML for the Welcome Screen & implemented UI button logic
- Created main logo (circular version)
- Contributed towards the performing of validations for sign-up & login inputs from users
- Assisted in the creation of the UML

Deliverable 2

- Aided in the implementation and UI of banning chefs for temporary and permanent suspensions
- Tested the Admin screen and the screen for viewing each specific complaint

Deliverable 3

- Changed android studio's default icon to the group's designed logo
- Produced the official XML screen for adding a new meal and viewing a meal's

information

- Formated how a meal's information will be displayed on the screen based on Firebase information given
- Reorganized and updated UML

Deliverable 4

- Designed the patterned background for the app
- Made the app runnable in Dark Mode
- Added extra functionalities and edited colours/texts throughout the app
- Designed XML for Orders in Progress adapter and Checkout screen/adapter and contributed to its logic and viewing availability
- Created XML for ordering a meal screen (except for the addition of the chef's address/rating)
- Contributed to the chef's profile
- Formatted the email being sent to a client using HTML and added the logo to it

AMY HUANG

Deliverable 1

- Designed XML for opening screen & implemented UI button logic
- Helped implement all validation for sign-up & login forms
- General debugging & testing

Deliverable 2

- Helped implement the logic for temporary and permanent suspension of chefs
- Debugged & tested chef suspension/unsuspension
- Tested viewing complaints from the admin screen

Deliverable 3

- Implemented UI logic for adding a new meal to the chef's menu
- Implemented validation for all forms in the new meal screen
- Helped with logic for storing meals in Firebase
- General debugging & testing

Deliverable 4

- Implemented UI logic for the search meal information screen
- Helped with logic for storing & removing orders in Firebase
- Helped design XML & implement UI logic for client's checkout screen
- Designed XMLs & implemented UI logic for both clients' & chefs' pending and completed/past orders screens
- Helped implement UI logic for chef's orders-in-progress screen
- Implemented functionality for sending email notifications to clients
- Helped with logic for storing chef ratings
- Helped implement UI logic for chef ratings
- Implemented validation for all forms in the new complaint screen
- Updated past test cases
- Updated UML diagram

- General debugging & testing

PRANAV KURAL

Deliverable 1

- Assisted with project planning including identification of model classes and design of UI components
- To achieve high levels of loose coupling and allow for more efficient scalability & debugging, implemented a pattern whereby all Firebase-related code was isolated from the UI components (Action classes). To achieve this, also designed a solution to handle the asynchronous nature of Firebase query handling (Handler classes) so remote and local data stay in sync at all times
- Created User, UserRole, Client, and CreditCard model classes
- Created the UI components and screen for a Chef and Client Sign Up
- Implemented the logic for handling user sign-up, including the creation of entity models for the transfer of unvalidated data to data handlers where validation logic and database data update logic were abstracted away from UI
- Implemented the core utility classes for handling method execution response which was to be used throughout the project (and deliverables) - Response & Result classes
- Updated documentation to ensure it reflected the application's current architecture information
- Assisted with the implementation of logic for data validation of the sign-up form

Deliverable 2

- Architected project's code management and data handling strategy which included finalizing the project's file structure and pattern for handling firebase-related code
- Implemented logic to ensure remote data is considered the SSOT (single-source-of-truth) to ensure no discrepancies in data seen by all live users
- Implemented logic for handling updates of complaints-related data both remotely (Firebase) and locally (app's instance)
- Refactored code across project to maintain a high code quality (for example, not using hard-coded strings where possible, implementing interfaces or abstract classes where helpful)
- Assisted with the development of UI components for displaying complaints
- Assisted with the implementation of logic for suspending Chef temporarily and permanently
- Fixed bugs in code from prior deliverables

Deliverable 3

- Identified all tasks to be performed for completion of the deliverable and created issues on GitHub to efficiently divide work & manage progress
- Implemented the Dispatch-Action-Handler (DAH) pattern for application state management (involving both local and remote data changes) and UI updates (including side effects and asynchronous updates)

- Implemented the logic for handling data and UI updates related to Chef's meals using the DAH pattern
- Created UI components and associated logic for displaying a list of meals in a way by which such components could be re-used (Menu and Offered Meals Screens)
- Assisted with the implementation of logic for passing of data between UI screens (especially the transfer of data from a meals list screen to a meal info screen)
- Refactored the whole firebase code related to meals data when a new strategy to store meals and chef-related data was adopted
- Assisted others with UI and logic-related issues
- Fixed bugs in code from prior deliverables
- Contributed to the maintenance of proper and detailed documentation of the project by updating the README file of the project on GitHub with essential info

Deliverable 4

- Implemented the logic and UI for providing Clients the ability to search for meals
    - Implemented a Trie-based prefix search for efficiency
    - Created comparator utility for sorting results by closeness to the Client's postal code
    - Created custom models to store, extract & transfer search results efficiently
- Assisted with the implementation of logic for handling orders (adding items to

the cart and order creation on checkout)

- Assisted with the UI for displaying meal and chef information when the client is looking to buy a meal
- Assisted with UI for checkout screen, specifically, displaying client's credit card and total order cost
- Assisted team members with any logic & UI related issues throughout the deliverable 4
- Fixed issues and code bugs from prior deliverables
- Ensure a high-level of code quality throughout the project

ANJALI MOHAMMED

Deliverable 1

- Designed the login screen
- Implemented Firebase Authentication to allow the verification of emails and passwords
- Coded the methods that take a local user object and store it in Firebase Firestore and that retrieve a user object from the database
- Created the UML diagram using a drawing tool

Deliverable 2

- Helped with displaying the suspension message for the chef
- Adapted the chef objects in firebase to now store suspension attributes

- Implemented the methods to update chef suspension in the database
- Updated the UML diagram
- General debugging and testing

Deliverable 3

- Implemented the class that handles the storage of meals in the database, retrieving meals from the database and updating the ‘offered’ attribute of a meal
- Updated the UML diagram
- General debugging and testing

Deliverable 4

- Code the classes that stores and updates orders to local user objects and in firebase, retrieve orders from the database
- Designed the pending and completed orders screen for a client
- Implemented the logic for rating a chef and retrieving a chef’s rating
- Helped create smaller ‘info’ classes to store in an order object
- Updated the necessary models to store orders locally and store the chef’s rating in the database
- Updated the UML diagram
- Fixed the date stored in an order object
- General debugging and testing

Deliverable 1

- Void cheque
    - Creating UI
    - Implemented camera functionality
    - Error handling and validation
    - Sending unique string to firebase database
- Connected screens together with buttons

Deliverable 2

- Helped create ListView for complaints
- Helped create UI for individual complaints and the admin screen
- Implemented date-picker for banning chefs
- Helped create complaint handling and created date format
- Created Junit tests to test methods for the client class
- Updated UML

Deliverable 3

- Created initial UI for chef screen and menu screens
- Bug fixes and other UI improvements
- Helped create Junit tests

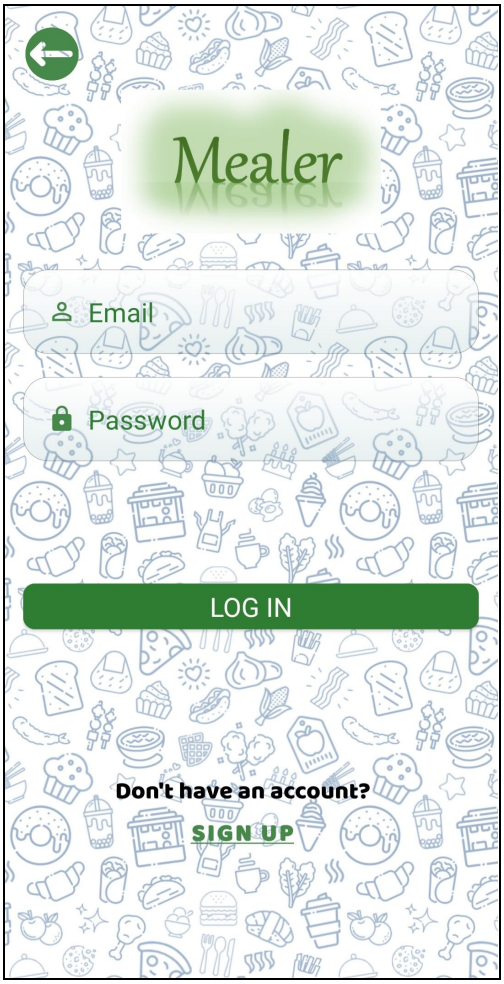
Deliverable 4

- Created back-end files for handling client orders
- Set up front-to-back connection for adding items to the cart and checking out

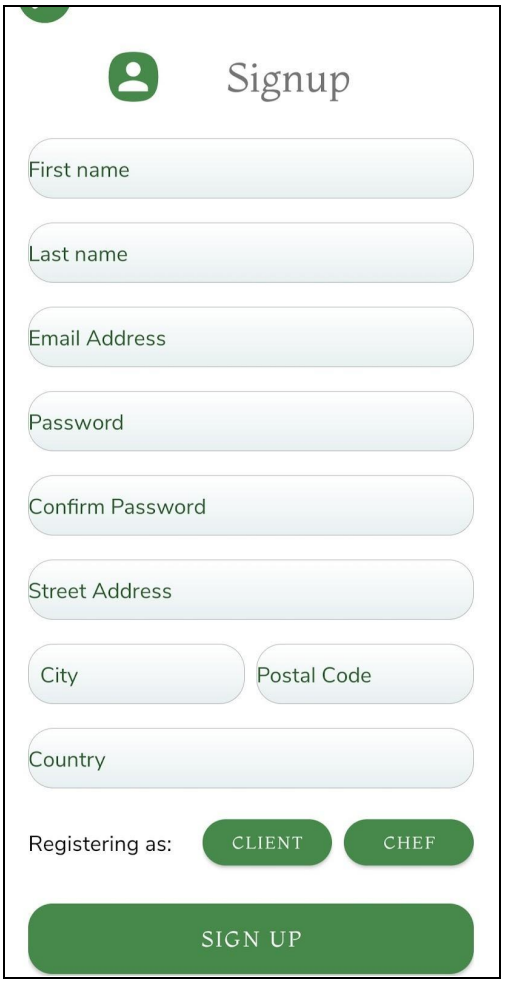
- Created backend files to handle order objects
- Created order adapter to handle displaying information on specific orders to display onto the screen
- Handled complaint logic and sending complaints to firebase
- Created UI for complaint screen
- Added validation and fixed bugs for previous deliverables such as void cheque screen

Application Screenshots

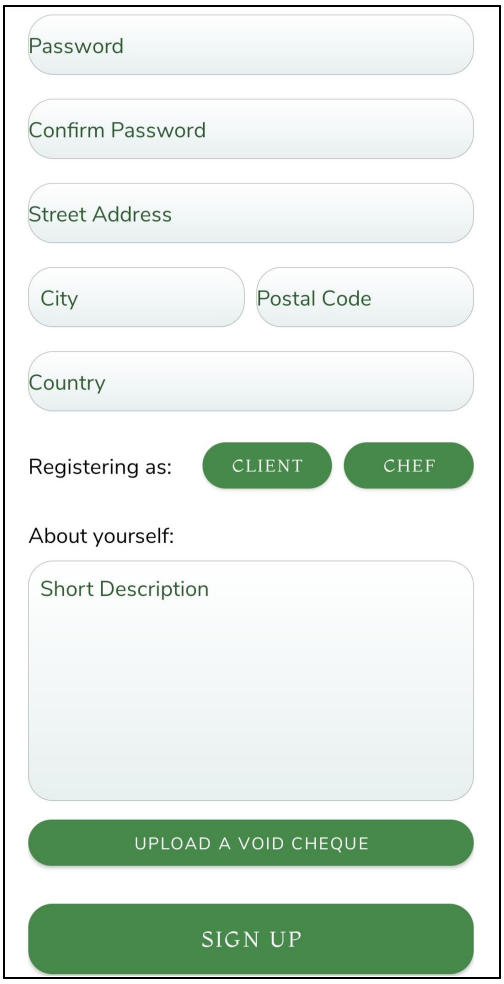
(Opening Screen)

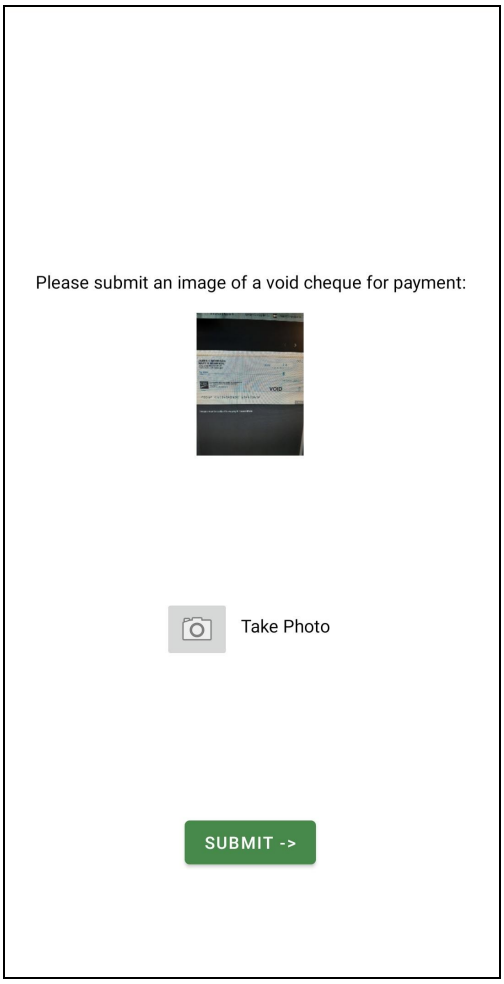


(Login Screen)

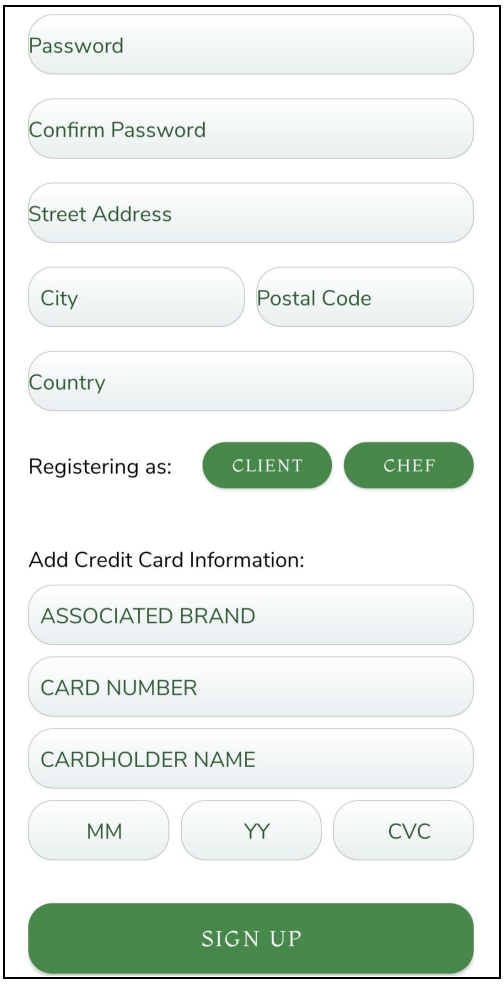


(Sign-Up Screen)



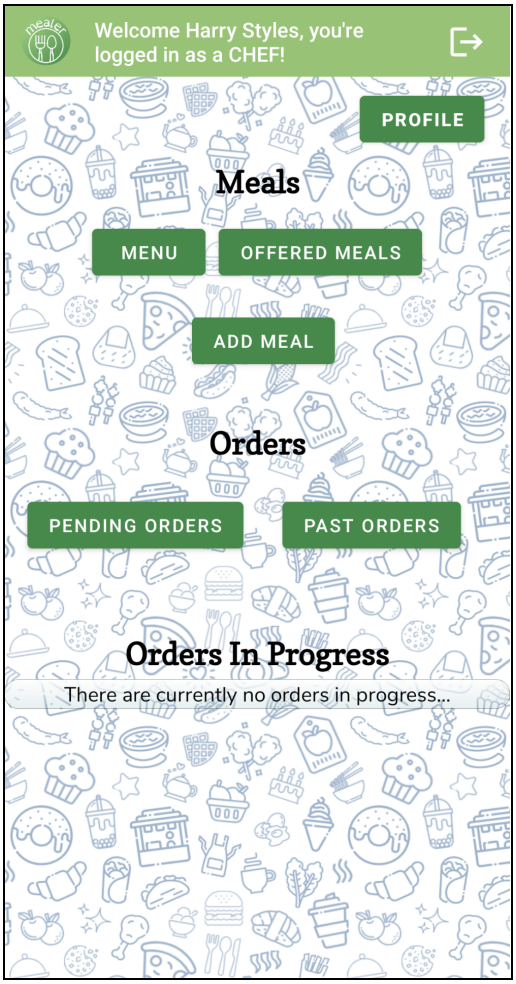


(Sign-Up Screen with Additional Chef Section) **(Void Cheque Screen)**

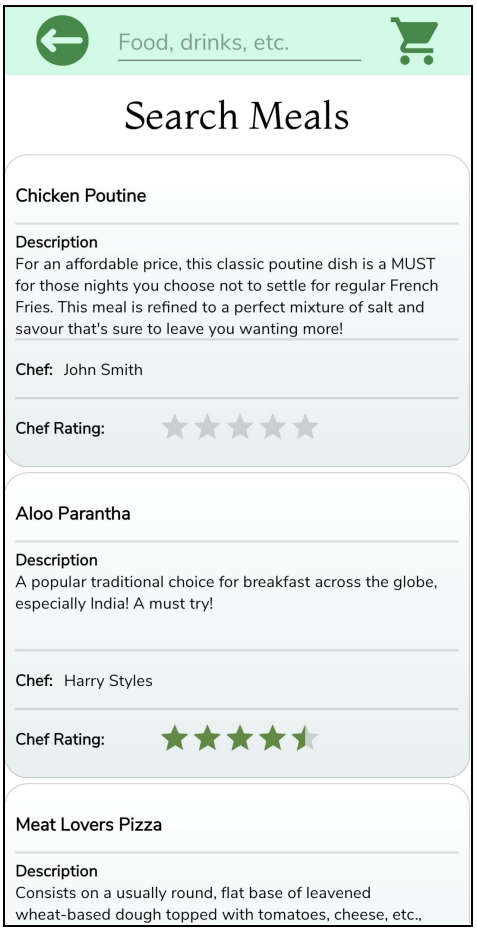


(Sign-up Screen with Additional Client Section)

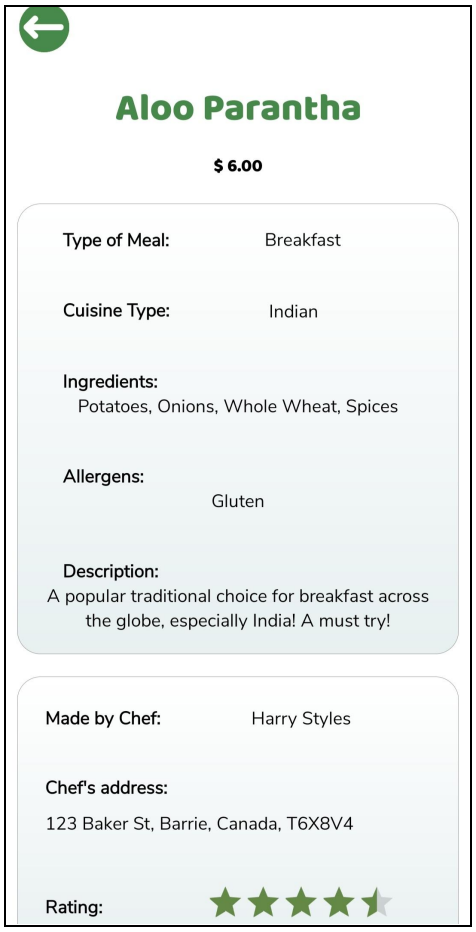
(Client Home Screen)



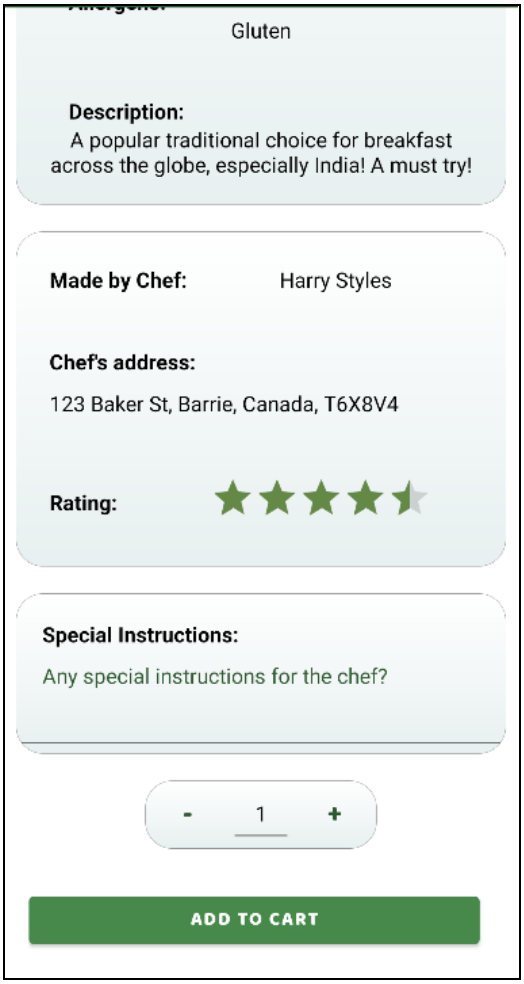
(Chef Home Screen)



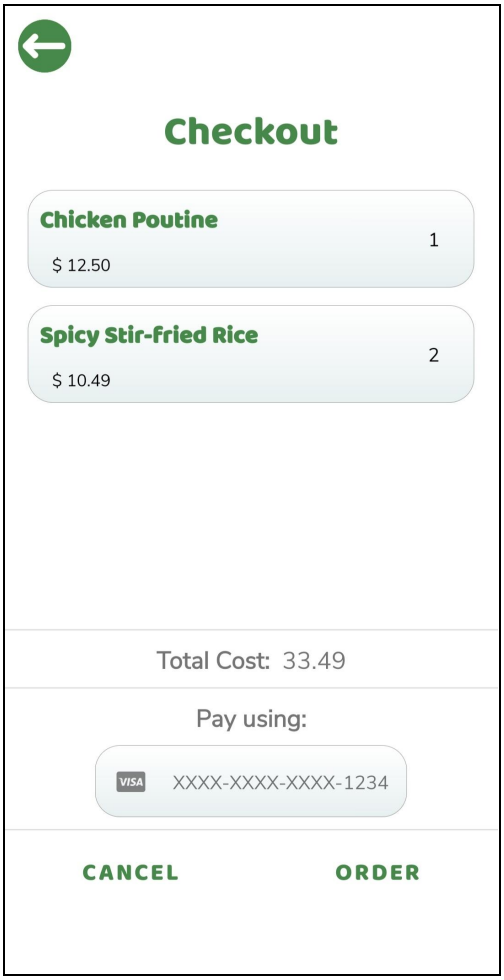
(Search Meal Screen)



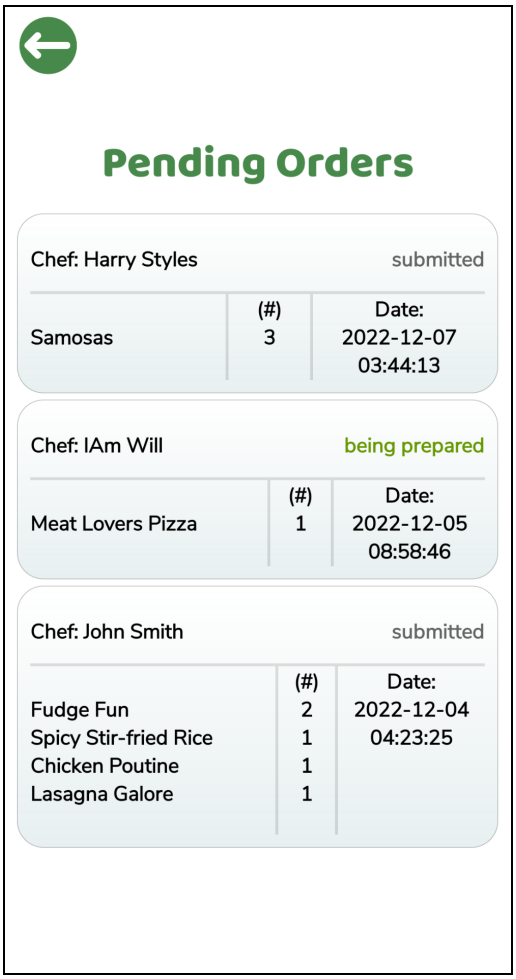
(Search Meal Information Screen)



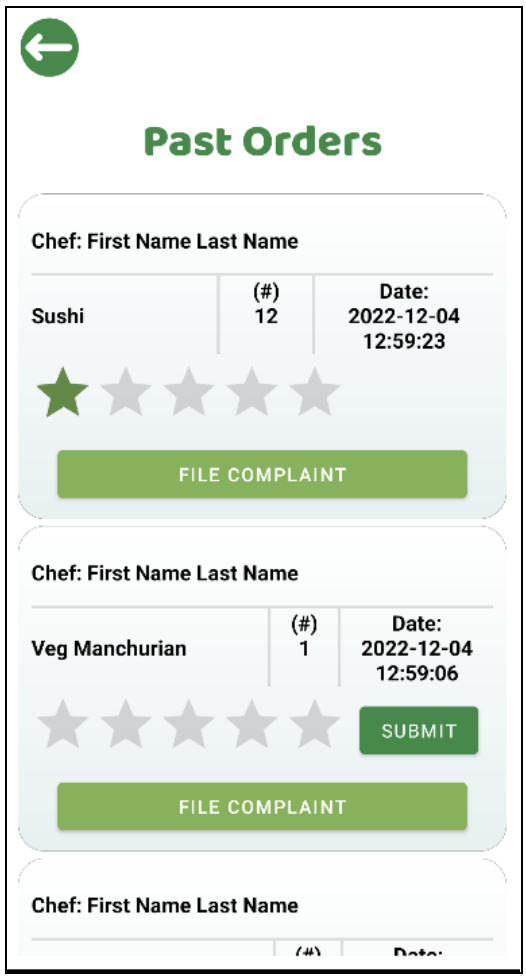
(Search Meal Information Screen Pt. 2)



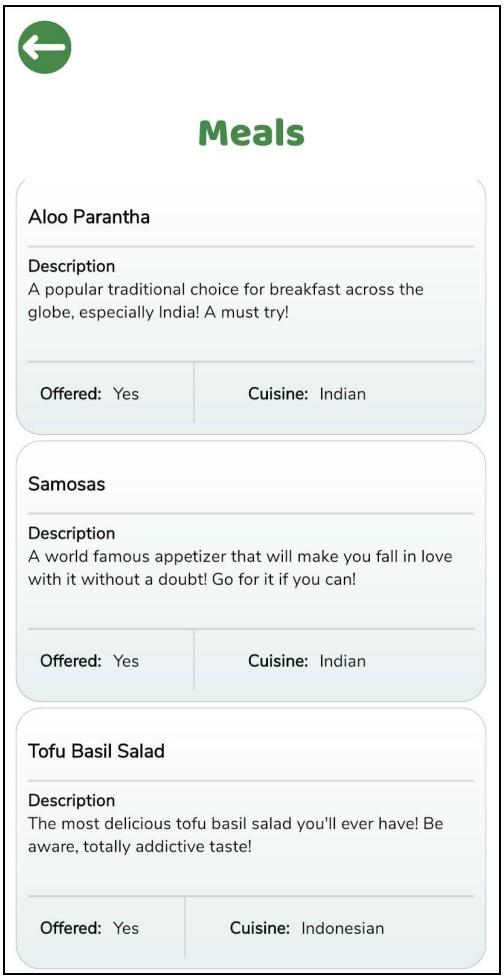
(Checkout Screen)



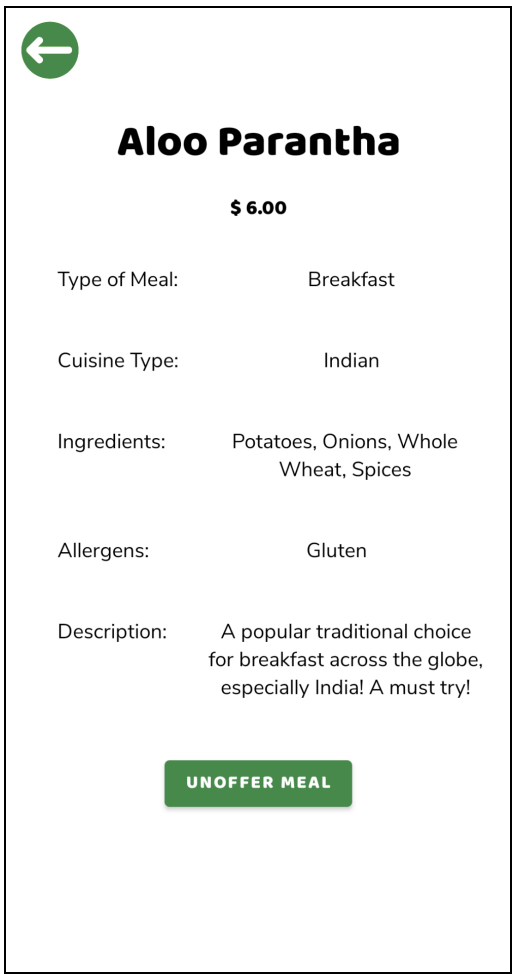
(Client Pending Orders Screen)



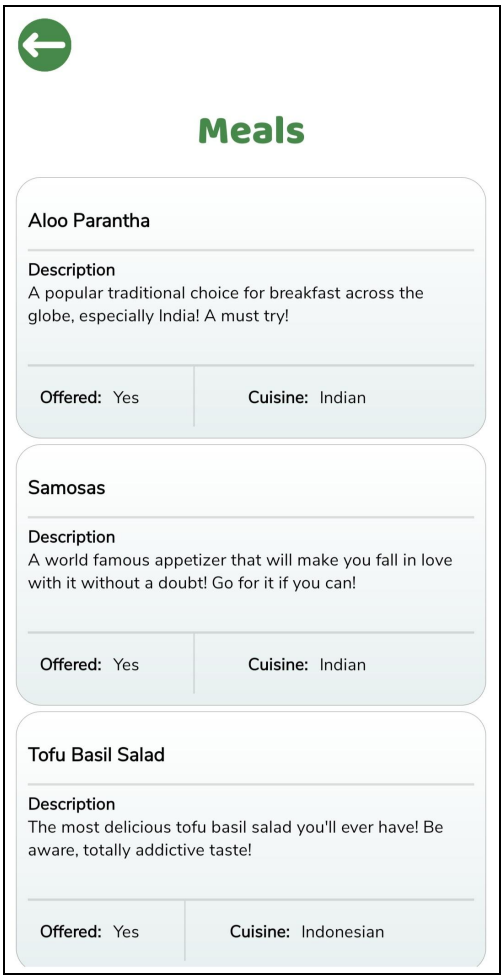
(Client Past Orders Screen)



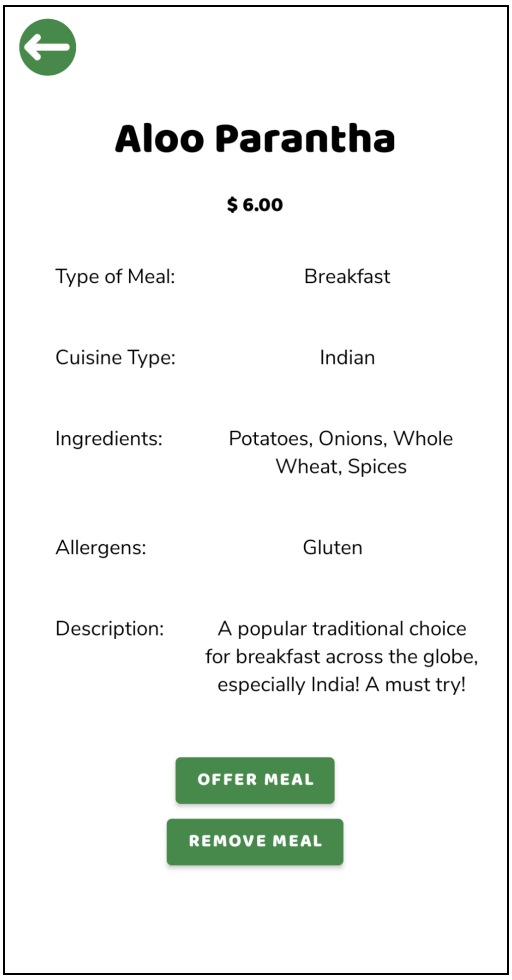
(Chef Menu Screen)



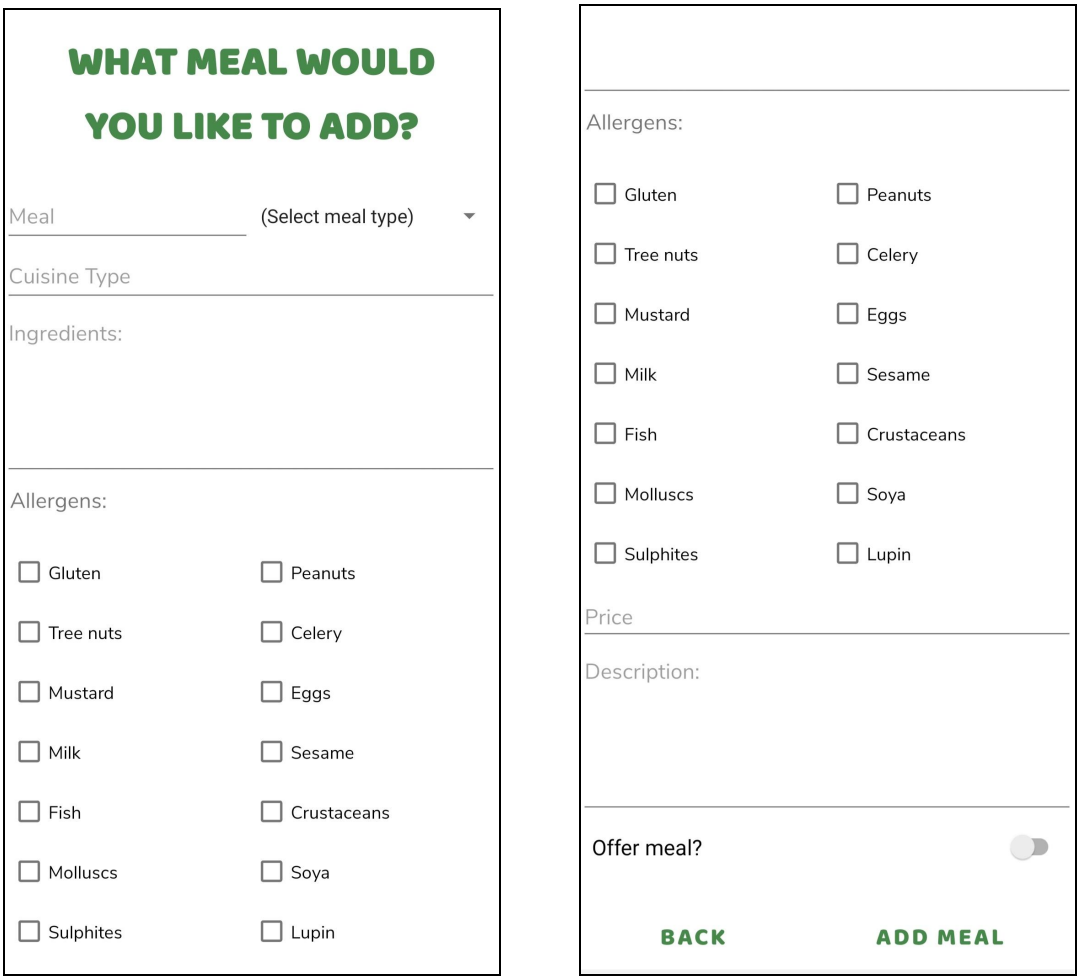
(Chef Meal Information Screen)



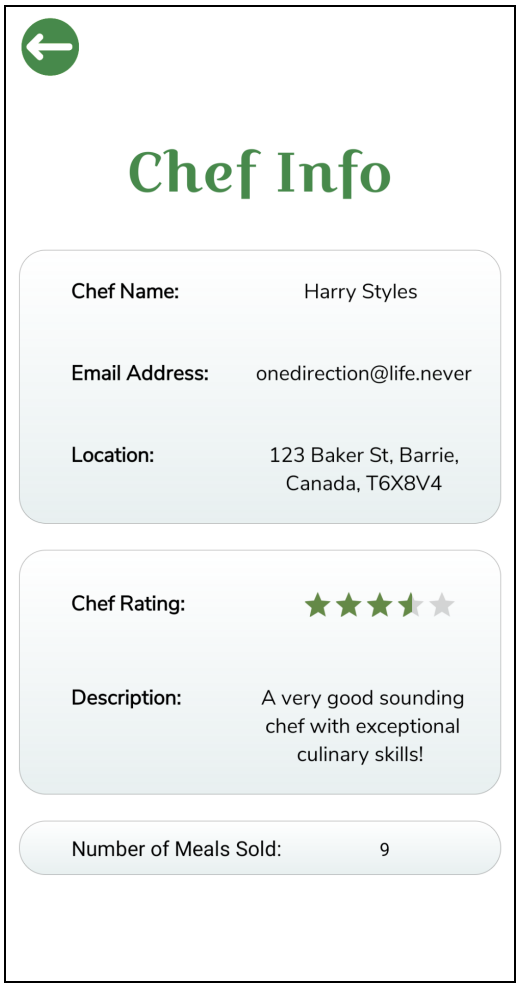
(Offered Meals Screen)



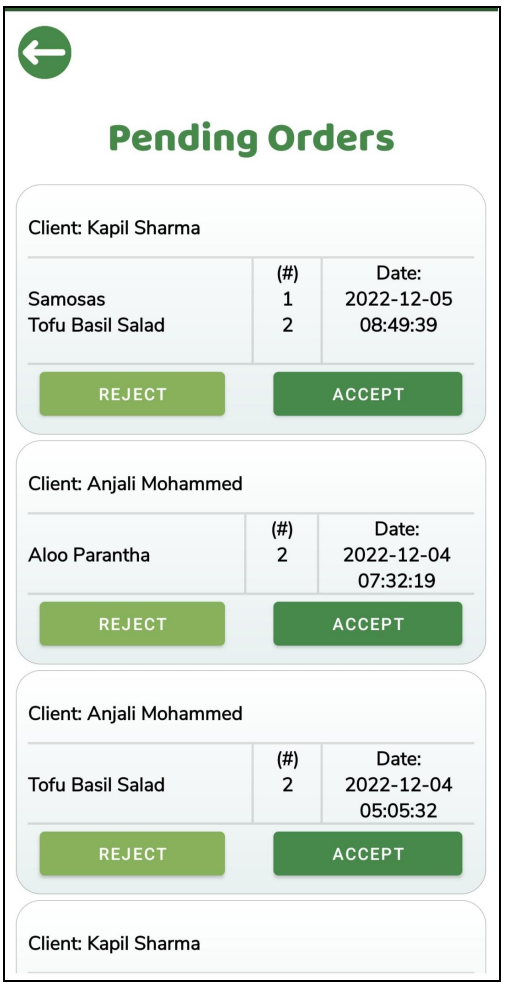
(Unoffered Menu Item Screen)



(Add New Meal Screen)

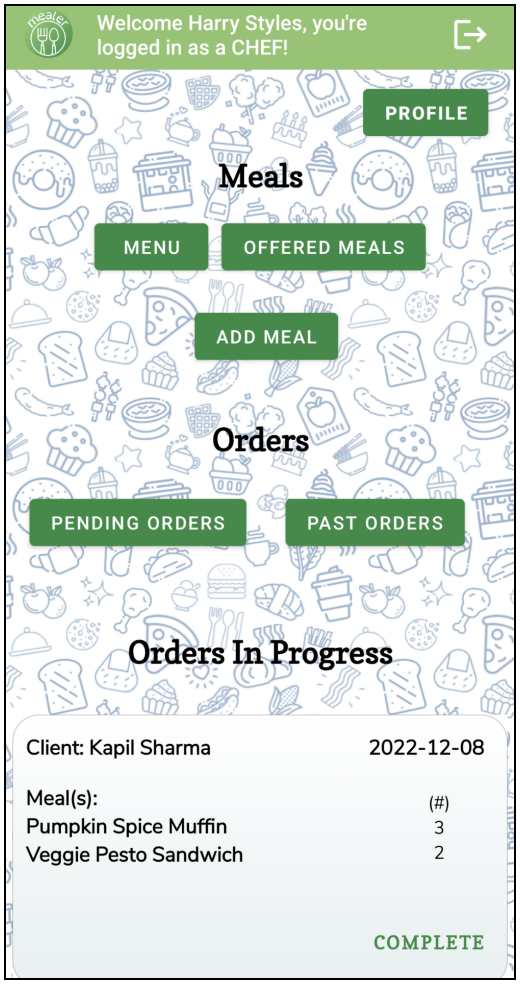


(Chef Profile Screen)



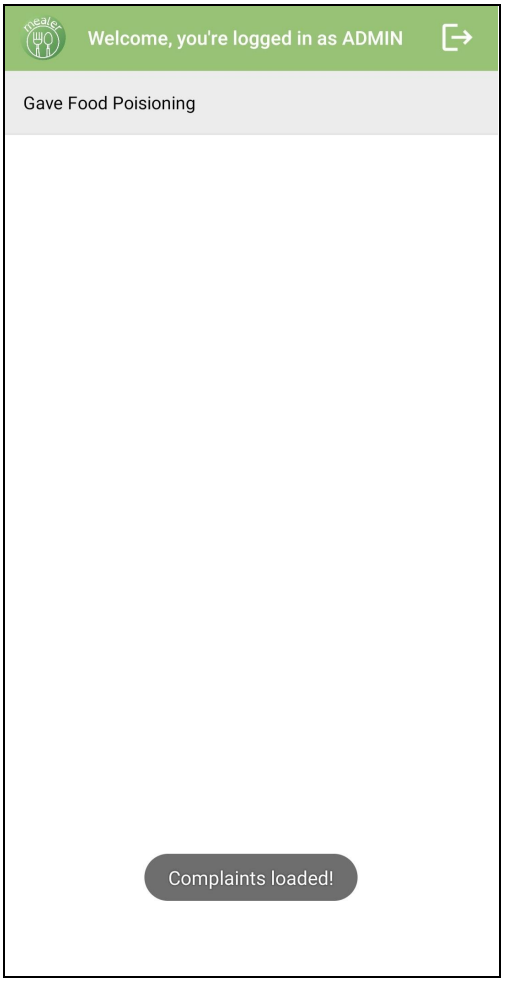
(Chef Pending Orders Screen)

(Chef Past Orders Screen)

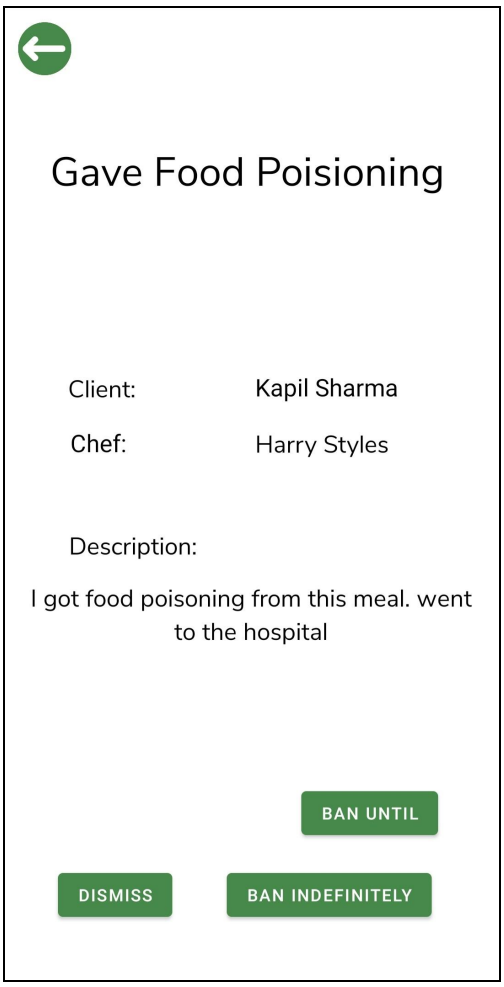


(Chef Orders-in-Progress Screen)

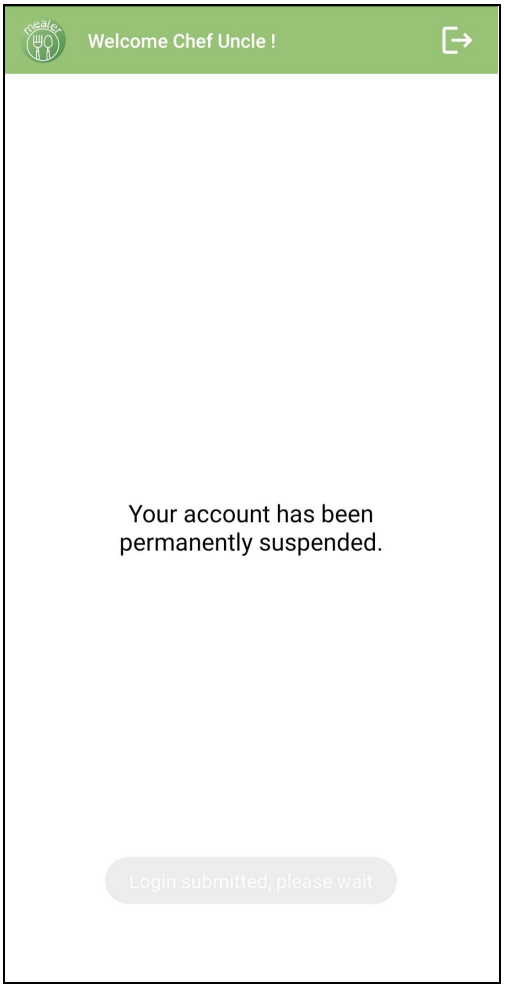
(File New Complaint Screen)



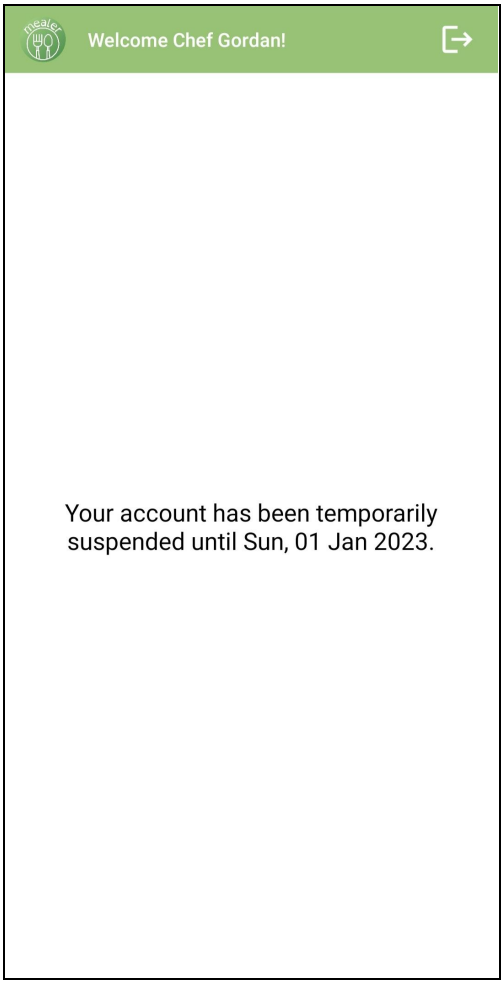
(Admin Screen)



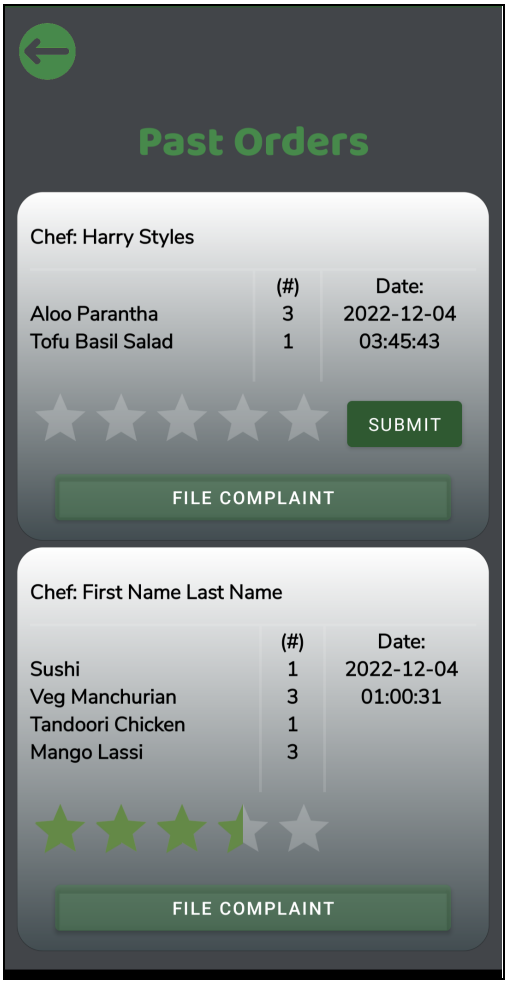
(Full Complaint Screen)



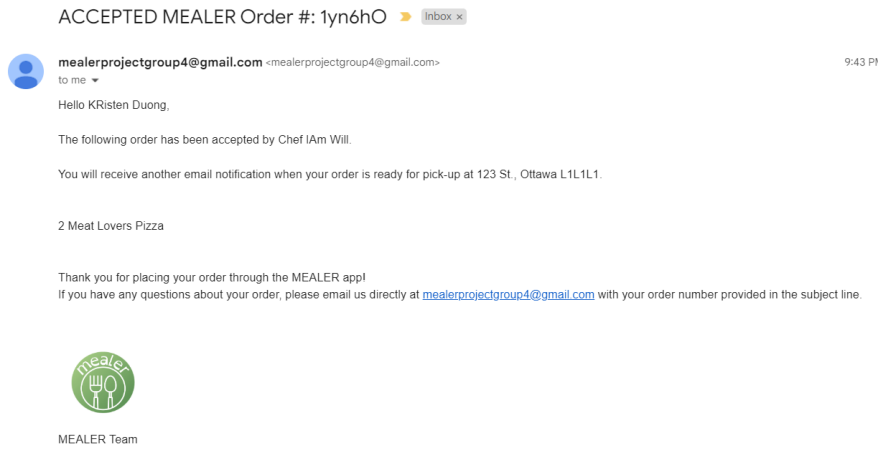
(Permanently Suspended Chef Screen)



(Temporarily Suspended Chef Screen)



(Example of a Screen in Dark Mode)



(Accepted Order Client Email Notification)

REJECTED MEALER Order #: r1lMBY Inbox ×

mealerprojectgroup4@gmail.com <mealerprojectgroup4@gmail.com>
to me

Hello KRisten Duong,

We regret to inform you that the following order has been rejected by Chef IAm Will.

3 Meat Lovers Pizza

Thank you for understanding.

MEALER Team

(Rejected Order Client Email Notification)

READY-FOR-PICK-UP MEALER Order #: 1yn6hO Inbox ×

mealerprojectgroup4@gmail.com <mealerprojectgroup4@gmail.com>
to me

Hello KRisten Duong,

The following order is ready for pick-up at 123 St., Ottawa L1L1L1.

2 Meat Lovers Pizza

If you have any questions about pick-up, please contact us at mealerprojectgroup4@gmail.com with your order number provided in the subject line.

MEALER Team

(Completed Order Client Email Notification)

Lessons Learned

The True Meaning of Teamwork

Throughout this project, it was evident how necessary it is to cooperate with all the team members. ‘Teamwork’ by definition means working together towards a common goal. However, it was discovered that teamwork in practice means putting together each member's strengths to negate any weaknesses. Particularly, one member’s weakness may be another member’s strength. With this ideology, the coding process becomes much easier and the final product is of the highest quality.

The Importance of Pulls and Merges

One problem encountered multiple times during the work process was the failure to avoid conflicts between everyone’s work. This general issue was rectified by reminding members to work on separate branches and merge work together afterward. Nevertheless, errors still occasionally arose, where a member's code was overwritten, or the whole application would no longer run. Eventually, the group found fewer issues as the project progressed, and the team remembered to be considerate of each other’s work. These situations ultimately emphasized the importance of having pulled from the master branch and debugging all errors before merging and pushing code to the repository.

No Such Thing As a Perfect Solution

An unusual realization but nevertheless an important one that was learned while working on this project is that there is no such thing as a “perfect” solution. Many times

we came across technical obstacles, like when implementing data-handling logic, where a multitude of solutions utilizing different design patterns could be implemented. The biggest challenge, however, was identifying the *best* solution that doesn't only solve the problem but also doesn't cost efficiency or code quality. What we learned is that there may not exist a perfect solution and oftentimes you have to make a choice by considering both advantages and disadvantages that a solution provides, for example, implementing a certain pattern for handling data updates might help increase efficiency drastically but may still lead to an indirect increase in the tight coupling of code overall. In such scenarios, it's vital to take a well-balanced and far-sighted decision where sacrificing one small aspect of your project doesn't hinder you from growing the project in the right direction.

Communication is the Key to Success (& Failure)

An axiomatic principle across various fields, it unquestionably applies to the development of any software application in a group setting. We learned early on that improper communication led to obstacles in the division of work, lack of flow of ideas, conflicts between group members, inefficient decision-making, and an increase in unproductive avoidable work like refactoring the same code multiple times with no clear objective. We worked diligently to ensure everyone on the team communicated well from the start of working on any deliverable. We also used tools like an online messaging app to enable a free flow of ideas and feedback. In-person meetings were given a high priority by all team members, as they allowed for a better environment for

the exchange of information between team members and hence enhanced everyone's ability to communicate and participate.

Code Maintainability Should Be Prioritized From the Start

An almost obvious proposition yet very easily ignorable when starting to work on a new project is to prioritize code maintainability from the very start. When we started the project, our focus was streamlined on creating a highly efficient and performant application, but we quickly realized as the app (and the features) grew that debugging the code was getting increasingly harder. Improper use of methods that were highly efficient but not completely understood, and a non-existent project file structure strategy, were leading us into a situation where we were spending more time debugging issues than adding new features. To resolve the problem, we implemented a proper project file structure strategy and modularized and abstracted the code as much as possible. We prioritized code maintainability in all deliverables thereafter, which resulted in a huge decrease in the need for debugging as well as time spent on fixing bugs and logic issues.

Everyone's Opinion Matters

Despite having a team leader, all members were treated equally and fairly. Having no one believe that they were above another or act as if they were, allowed everyone to work in harmony. With each member having their own voice and opinion on any decision being made, everyone had the freedom to share their thoughts without the fear of being degraded as all ideas were valued and appreciated as a group. That being

said, if an inclusive work environment was not created at the start, or respected by all members, the team would not have been able to cooperate or accomplish any feat. Being able to build on others' ideas often leads to new innovations, but that can only happen if everyone shares their thoughts. In the end, the lesson to take away from this group project is that work and efficiency can only thrive when everyone is given an equal opportunity to voice their opinions.

Conclusion

Overall, we as a group learned quite a lot from our time working together and we believe that these lessons will be carried over to future group tasks during school and in the workplace.