Mealer App Final Report

SEG2105[B] Introduction to Software Engineering

Fall 2022

School of Electrical Engineering and Computer Science

University of Ottawa

Group 16

Eric Eaton (300260795)

Brent Palmer (300193610)

Gregory Marcelin (300277019)

Victor Matyiku (300174358)

Introduction

Mealer is an app that aims to connect local cooks with local clients in the Ottawa region. Mealer has been developed by aspiring app developers from Ottawa, Eric Eaton (300260795), Brent Palmer (300193610), Gregory Marcelin (300277019), and Victor Matyiku (300174358). The core concept is similar to UberEats, wherein clients may choose from a wide variety of meals to order from the comfort of their very own home using a mobile application. The primary difference between UberEats and Mealer, is that Mealer is much more personal; instead of connecting a client with a generic restaurant, Mealer will allow for clients to connect with specific chefs in the Ottawa region. This gives personality to the meals, and gives clients the warm feeling of having a meal made that perfectly fits their needs and wants. The app also allows for communication between a client and a cook in case a client would like their meal personalized to cater to their preferences. Ultimately, Mealer brings meals straight from the kitchen of a cook's home to the dinner table of the client.

There are three main types of users that each have distinct functionality. As a cook, you have a menu of offered meals from which clients can order. Once registered you may add meals to your menu, choose which meals you offer to the public, edit the meals you have on your menu, delete meals from your menu, and view your purchase requests. You may send mail to other users and view your inbox of messages from other users. You may also view your profile, and impress your friends with your incredible ratings. Be careful not to be suspended for poor performance though. As a client, once registered you may search for offered meals based on meal name, meal type, and cuisine type. You may view the information of the meals as well as the information of the cooks that offer those meals. You may submit a request to order a meal, and then if the cook accepts it, you will be able to pick up the meal. You may also view your purchase requests and list of meals they purchased. You may also rate the meals you order, and send a complaint to a cook if there are any issues. You may also send mail to other users and view your inbox of messages from other users. Finally, as an administrator you may view your inbox, send messages to other users, and handle complaints from clients by dismissing complaints or suspending cooks either temporarily or indefinitely.

Updated UML Class Diagram

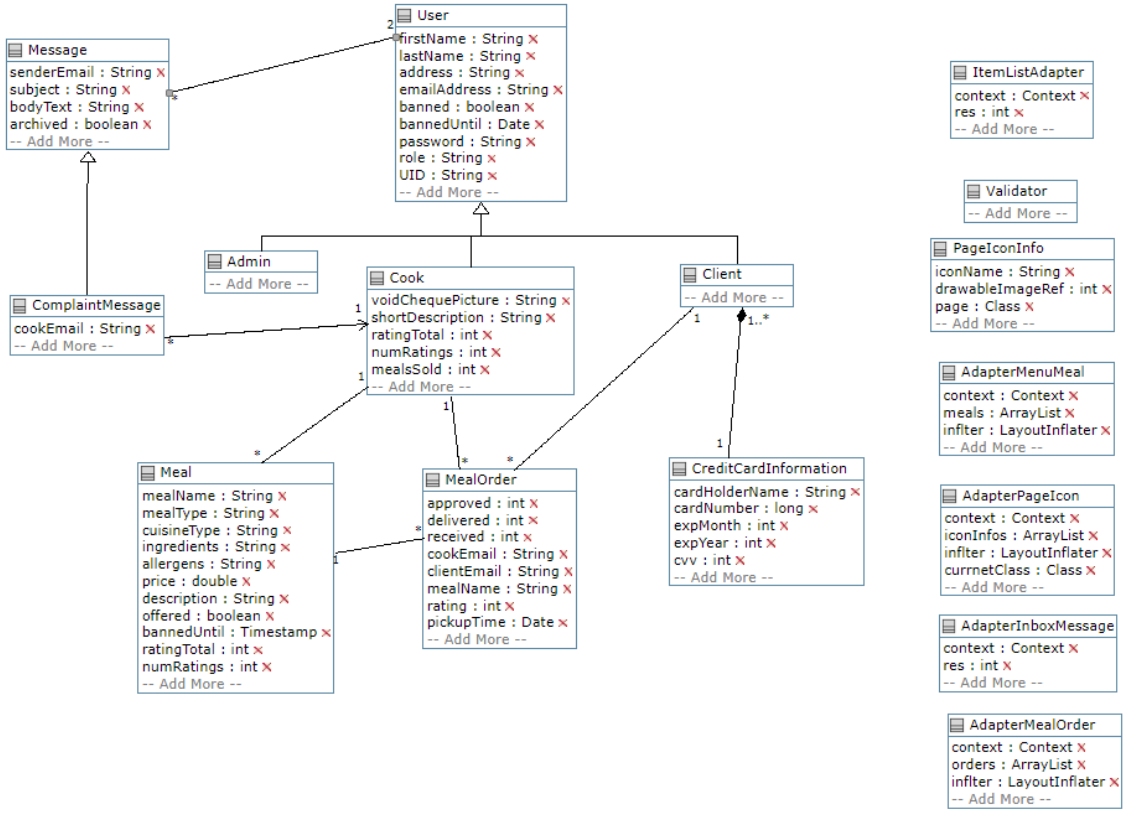


Figure 1: Updated UML Class Diagram

Contributions of Team Members for Each Deliverable

Table 1: Contributions of Each Team Member for Each Deliverable

Contributions for Deliverable 1	
Eric Eaton	-Joined GitHub -Made at least one commit to GitHub -Submitted APK -Setup Trello* -Set up UI and Android Studio backend for log off
Brent Palmer	-Joined GitHub -Made at least one commit to GitHub -UML Class Diagram -Added validation to input fields -Backend and frontend of “a user can create a client or cook account”
Gregory Marcelin	-Joined GitHub -Made at least one commit to GitHub -Backend and frontend of “a user can create a Client or Cook account” -Added basic validation for the fields
Victor Matyiku	-Joined GitHub -Made at least one commit to GitHub -Setup Firebase for the project -Setup backend in Android Studio for Firebase support -Implemented logout functionality -Implemented welcome screen -Added void cheque field
Contributions for Deliverable 2	
Eric Eaton	-Implemented Unit test cases -CircleCI automated tests -Admin can action the complaint -Made Cook suspension notice
Brent Palmer	-Created all new underlying java classes -Updated UML Class Diagram
Gregory Marcelin	-Designed the Admin complaints activity such that the admin can view the list of complaints -Made Cook suspension notice
Victor Matyiku	-Implemented storing account information in Firebase -Integrated Firebase to complaints inbox to store/access complaints in

	real-time -Admin can action the complaint -Made Cook suspension notice -Submitted APK
Contributions for Deliverable 3	
Eric Eaton	-Implemented Unit test cases Cook add meal, add meal to offered meals, edit meals, delete meals, and remove from offered meals -All fields are validated (designed validator class and updated old validations for improved modularity)
Brent Palmer	-Cook add meal, add meal to offered meals, edit meals, delete meals, and remove from offered meals -Updated login to bring to specific landing pages per user type -Updated UML Class Diagram
Gregory Marcelin	-Improved the UI for the Admin and Cook activities -Worked on Add and Remove meals activities for the cook -Submitted APK
Victor Matyiku	-Restricted banned Cook's actions to only logout -Fixed UI and functionality bugs -Wrote Firebase documentation specific to our project
Contributions for Deliverable 4	
Eric Eaton	-Client can search for a meal -Client can rate the Cook from which they purchased a meal -Client can submit meal purchase request -Client can view cook information and rating for each meal in the search result -Meal approved notifications -Field validation -Client can view status of purchase -Client can submit a complaint about a Cook to the administrator
Brent Palmer	-Updated UML Class Diagram -Submitted APK -Implemented Unit test cases -Compiled final report -Updated some outdated field validation
Gregory Marcelin	-Worked on cook profile related activities -Cook can receive the purchase request submitted by the Client -Cook can view and approve/reject purchase requests from Clients

	-Worked on view meal activities -Field validation -Client can view status of purchase -Client can submit a complaint about a Cook to the administrator -Client can view cook information and rating for each meal in the search result
Victor Matyiku	-Rewrote code for better functionality with new activities -Fixed IU functionality bugs -Implemented Unit test cases

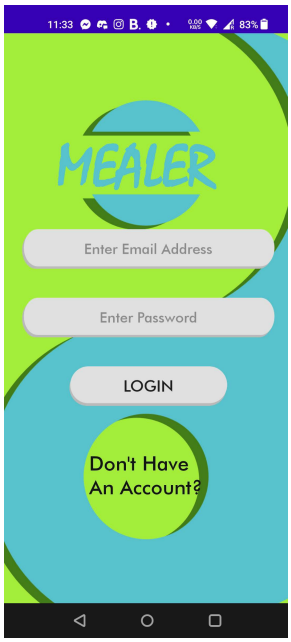
*Trello is an online resource we used to track the work of the project. We listed all the functionality necessary for each deliverable, assigned work to specific members of the group, linked necessary pages (like the UML), maintained a bug log, documented each of the classes, and documented how many of the more complicated features (like the implementation of Firebase) works. We also documented our presentation, among many other qualities that pertained to the project. To view the comprehensive documentation, you can join the Trello with this link:

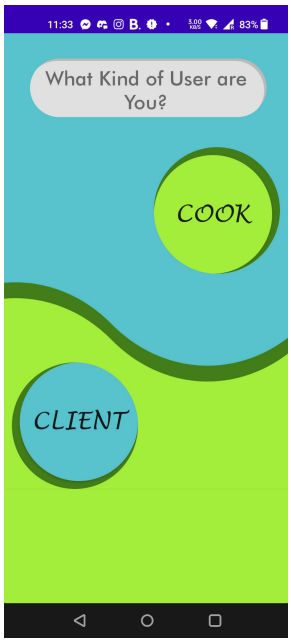
https://trello.com/invite/b/dr1O4Fqo/ATTIcc1ccb13ae86046ad1bfb38badd94dbb92927B2F/seg-project

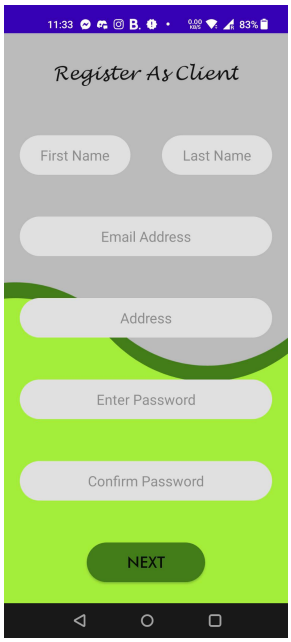
Screenshots of Your App

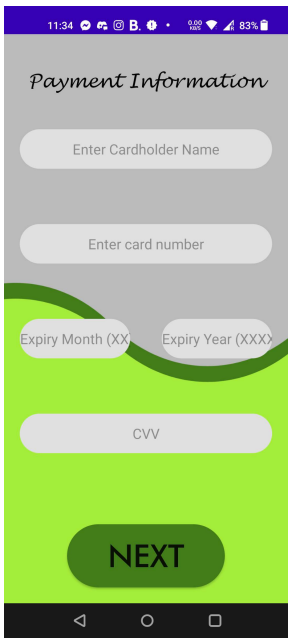
Splash Screen

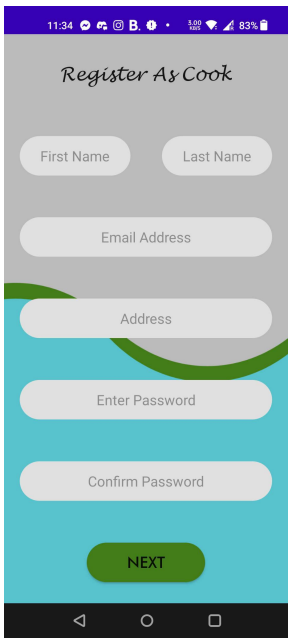
Login and Registration

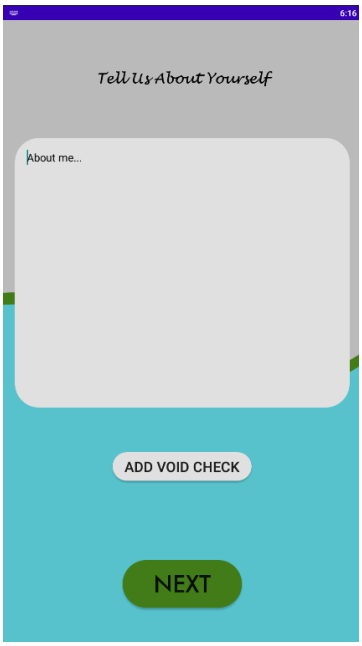




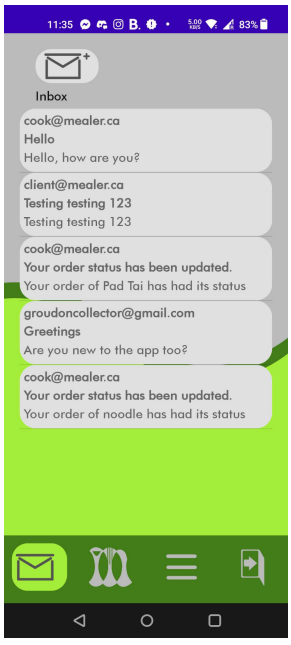


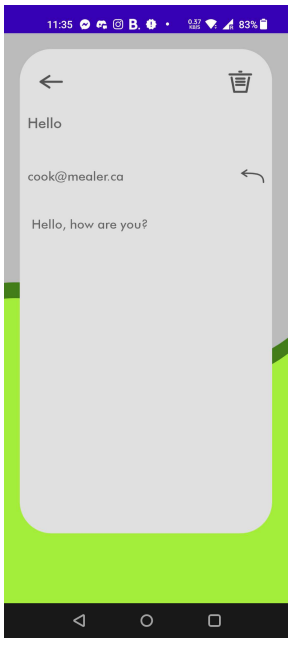




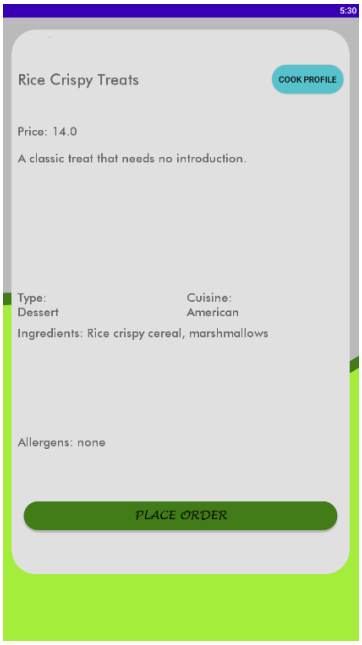


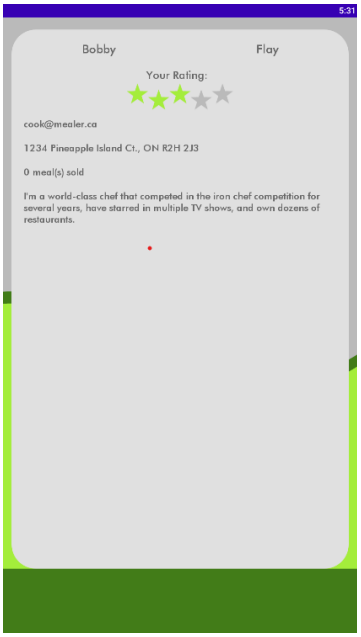
Client Inbox and Messaging

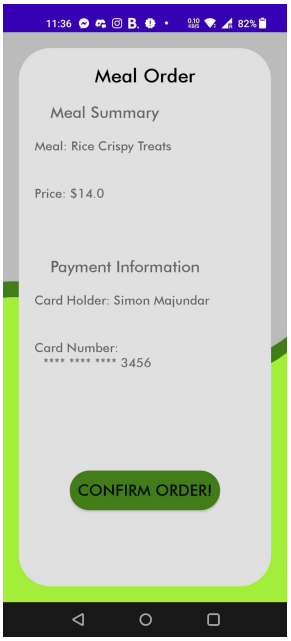




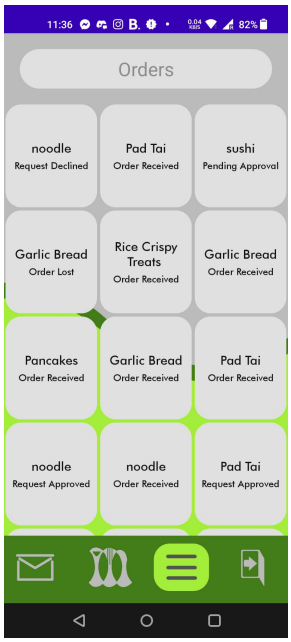
Client Meal Search

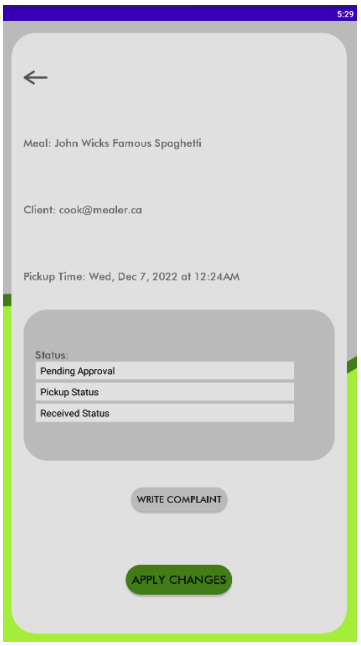


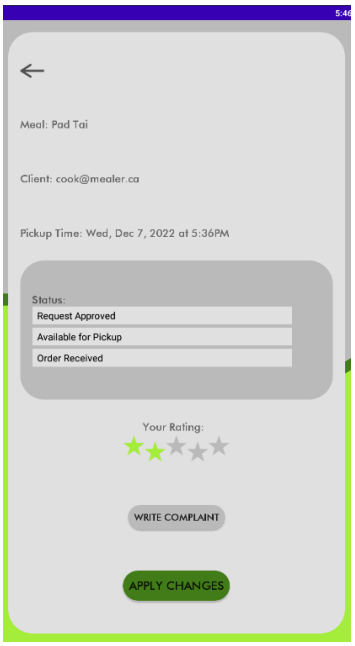


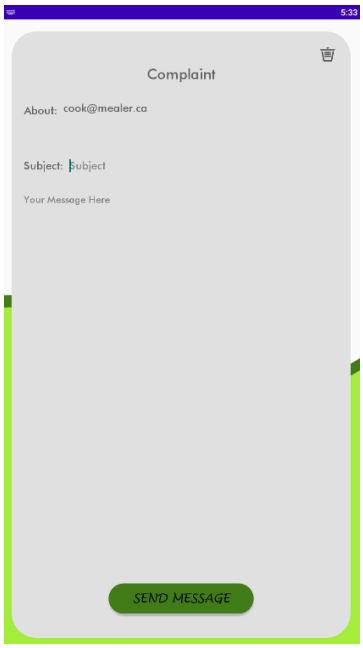


Client Meal Orders

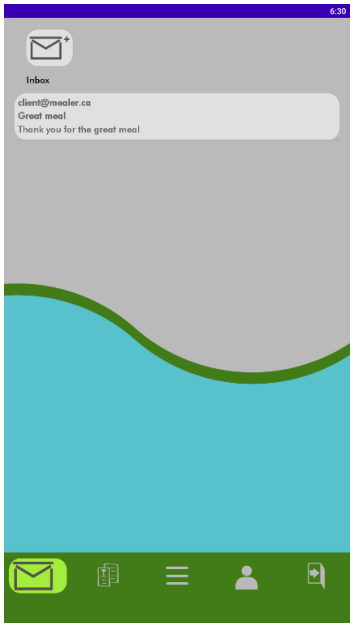


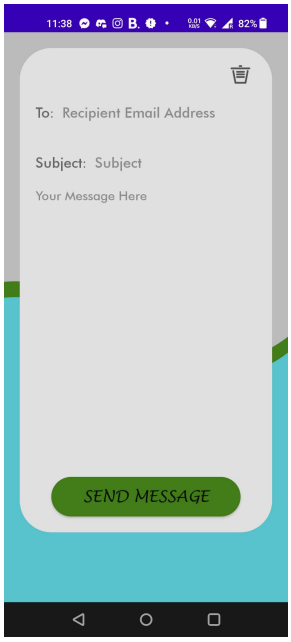


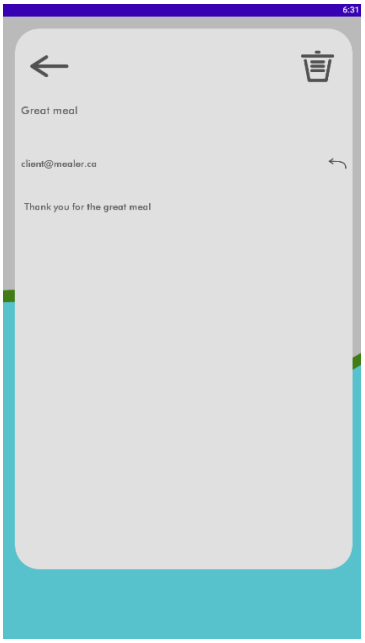




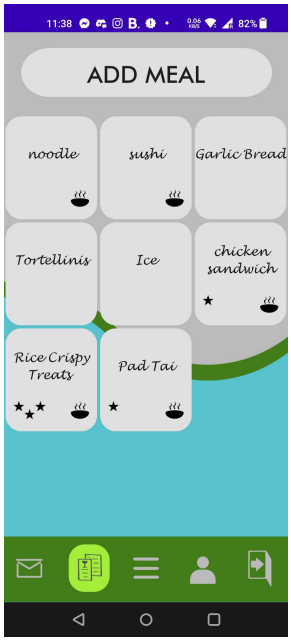
Cook Inbox and Messaging

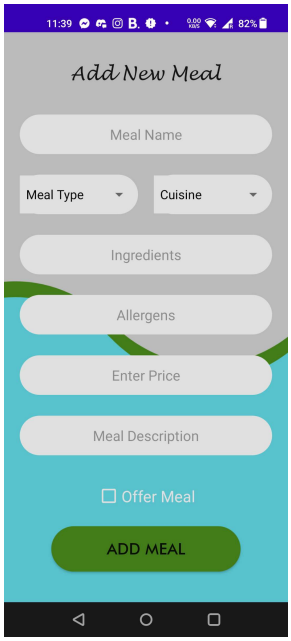


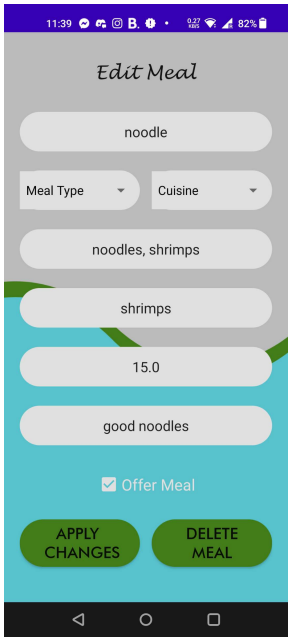




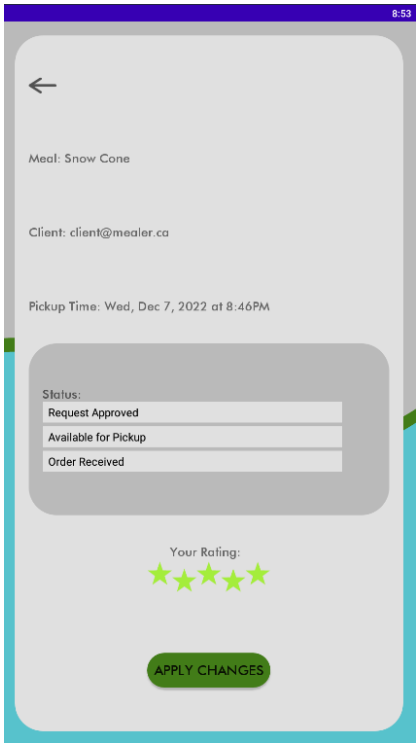
Cook Menu



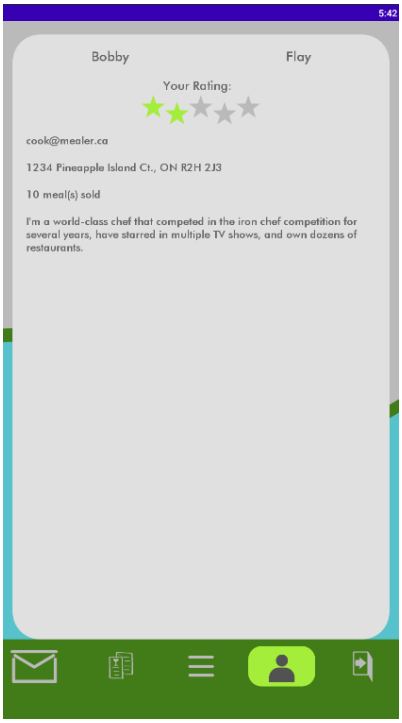




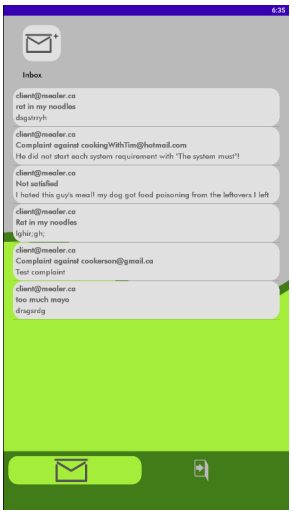
Cook View Orders

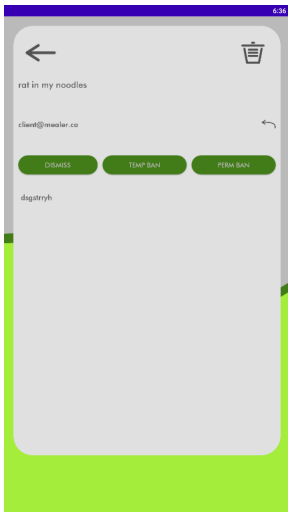


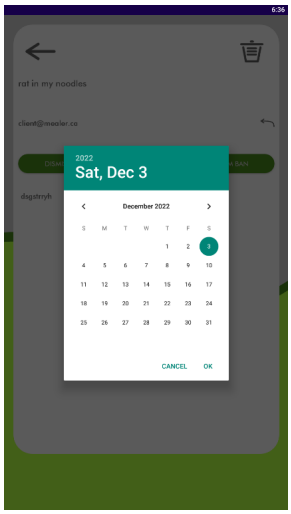
Cook Profile



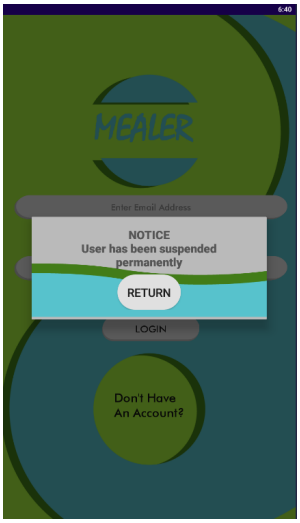
Admin Inbox

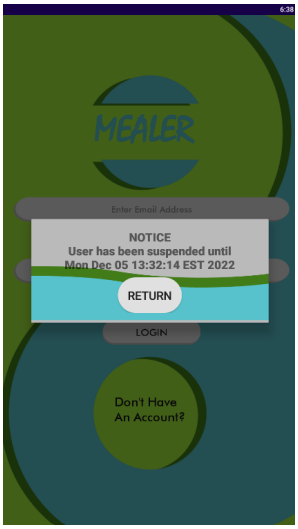






Bans and Suspensions





Lessons Learned

Throughout the development of the Mealer app, many lessons were learned by our group. Firstly, the importance of using a version control system. We used GIT as our version control system. Many of us had never used a system like this before, and we were very quick to realize the power it offers. The first major lesson was the effectiveness of branches and to not work directly in main. Using branches facilitates the modularity of work assignment; it made it much easier to work on individual components of the project by working in different branches. We also learned the importance of relevant commit messages and branches. Having relevant names makes it much easier to find specific versions and branches. We also learned the usefulness of being able to save previous versions. In some cases work was preemptively pushed to main, and it was very nice being able to revert changes that were not complete.

A lesson that was more important for some group members than others is the importance of self-learning. This was the first major project in university for us that required us to go beyond our knowledge from within the course when developing certain functionalities. Lots of information regarding app development in Android Studio requires supplementary self-teaching, and learning how to teach oneself about how certain Android Studio functionality works without relying on course material was an important lesson for some.

An additional major lesson is the importance of modularity. There are various different ways we implemented modularity which facilitated the design process, and also some areas in which we did not which emphasized its importance. A great example of modularity is simply the division of work (using Trello) and the use of branches in git to divide the workspaces. This approach made it easy for each group member to know exactly what they were required to

implement, and it made conflicts non-existent because we all worked in our own branches. Evidently there were still merge conflicts with main to handle, but the system remained the same for any one developer when working on a specific component in a specific branch.

Specific examples of modularity for the backend development is also insightful. In deliverable 3, we added the Validator class which had generic validation methods for the main validation checks. For example, checking if a field is alphabetic or validating an email. These methods could be reused and made the subsequent validations much easier. In addition, each of our activities were designed with plenty of methods that increased modularity within each activity. This further facilitated the design process. We also made some activities themselves modular, like the account creation page. The textboxes and themes were designed to display slightly different information based on the user (like a green theme for a client and a blue theme for a cook) which made code reuse easy. One issue we faced in terms of modularity reflected the listeners that relate to our Firebase. To get the documents from Firebase, we copy-pasted some listeners from our previous code which meant that if there was an error it needed to be changed in multiple places. This did not happen often, but it did demonstrate that modularity is important. We actually plan on continuing the development of this app beyond this course, and we are considering creating a Facade class to acquire documents to make the app more modular. An excellent final example of modularity is the navbar. We designed it in a way such that the same code for the navbar could be used in multiple activities, which once more facilitated code reuse.

Assumptions

- Contactless pick-up meaning we include both pick-up status and client-received status for ordered meals
- The “lost” state is for meals that were available for pick-up but were not received (ex: stolen meal from front porch)
- A meal is considered sold once the client has reported that the meal has been received