MEALER

Project Report

SEG 2105 – Introduction to Software Engineering

Fall 2022
School of Electrical Engineering and Computer Science
University of Ottawa

Course Coordinator: Hussein Al Osman

Jacob Maurice 300187393
Tess Harper 300237877
Elsa Ko 300175190
Saranki Mahendran 300226051
Katie Liu 300126321

Submission Date: Friday, December 9th 2022

Table of Contents

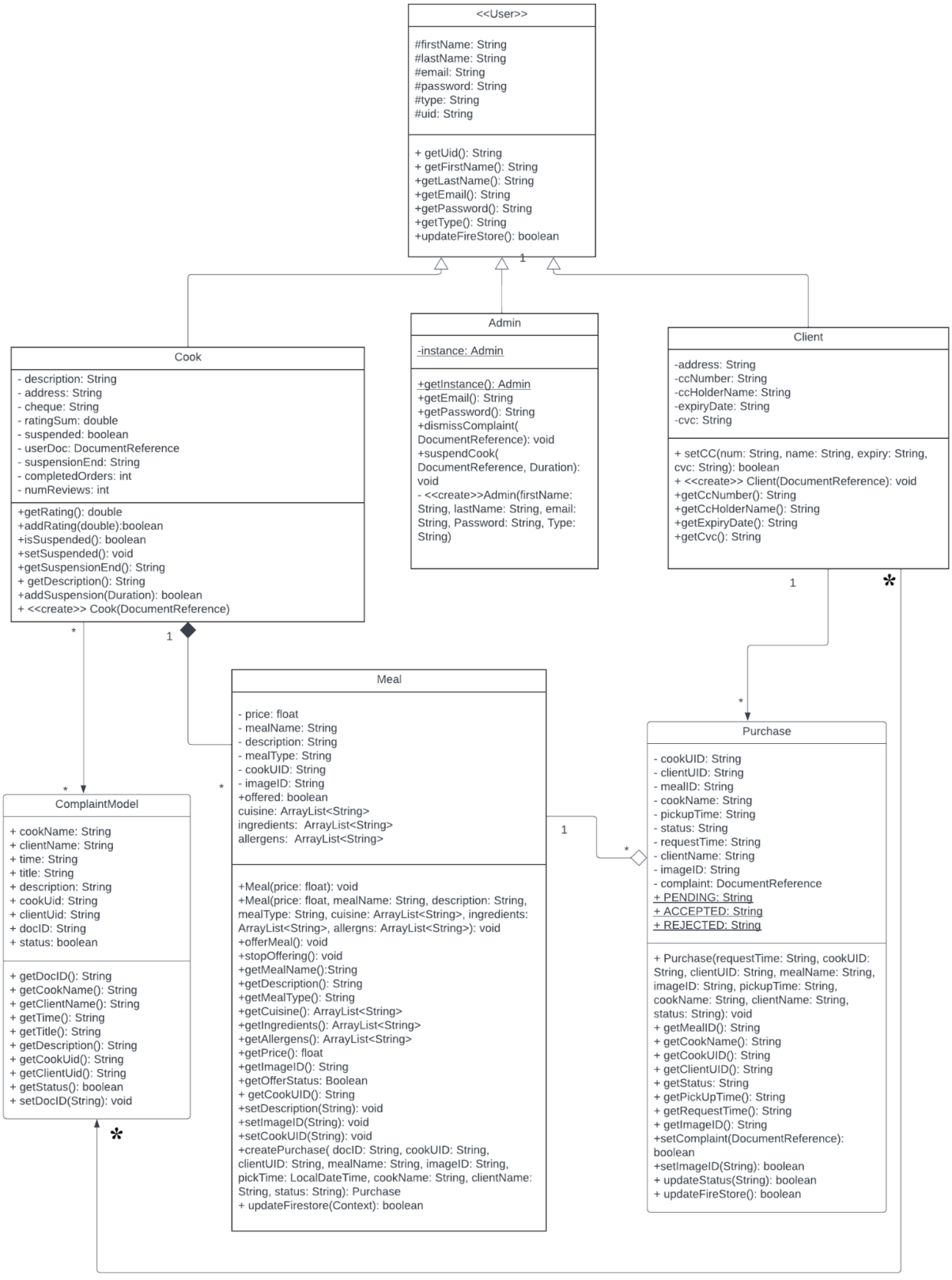
Introduction

Mealer is an Ottawa-based meal sharing application where local cooks can sell meals to clients from their home. This application supports three types of users:

(1) Cooks, a user that makes meals at home and sells them to Clients
(2) Clients, a user that buys meals from Cooks. They order the meal through the application and pick it up from the Cook's home.
(3) Administrators, a user that receives complaints about a Cook from a Client and may suspend the Cook if necessary or dismiss the complaint.

In addition, non-suspended cooks have the option to customize their menu; they can add and/or delete meals, as well as add and/or remove meals from their *offered menu* by toggling the offer status to off/on. On the other hand, a meal can only be deleted if it's not offered.
Moreover, clients can search for their desired meal and request a purchase, whereas (non-suspended) cooks can accept or reject their meal's purchase requests at specified dates and times. All (non-suspended) cooks have access to their profile page, which clients can also view from their meal search results. However, unlike cooks, clients cannot view purchase requests while on the cook profile.
Once a purchase request is accepted, the requested client is given the options to both rate and/or submit a complaint towards the purchased meal's cook. When a Cook is suspended they do not have access to the features of the app for the duration of their suspension length; when they log in, a message will appear telling them how long they are suspended for. If the Cook is suspended indefinitely then they will see a message saying so, and will not have an end time for the suspension.

UML Diagram


<<User>>
#firstName: String
#lastName: String
#email: String
#password: String
#type: String
#uid: String
+ getUid(): String
+ getFirstName(): String
+getLastName(): String
+getEmail(): String
+getPassword(): String
+getType(): String
+updateFireStore(): boolean
1
Admin
-instance: Admin
+getInstance(): Admin
+getEmail(): String
+getPassword(): String
+dismissComplaint(DocumentReference): void
+suspendCook(DocumentReference, Duration): void
- <<create>>Admin(firstName: String, lastName: String, email: String, Password: String, Type: String)
Cook
- description: String
- address: String
- cheque: String
- ratingSum: double
- suspended: boolean
- userDoc: DocumentReference
- suspensionEnd: String
- completedOrders: int
- numReviews: int
+getRating(): double
+addRating(double):boolean
+isSuspended(): boolean
+setSuspended(): void
+getSuspensionEnd(): String
+ getDescription(): String
+addSuspension(Duration): boolean
+ <<create>> Cook(DocumentReference)
Client
-address: String
-ccNumber: String
-ccHolderName: String
-expiryDate: String
-cvc: String
+ setCC(num: String, name: String, expiry: String, cvc: String): boolean
+ <<create>> Client(DocumentReference): void
+getCcNumber(): String
+getCcHolderName(): String
+getExpiryDate(): String
+getCvc(): String
Meal
- price: float
- mealName: String
- description: String
- mealType: String
- cookUID: String
- imageID: String
+offered: boolean
cuisine: ArrayList<String>
ingredients: ArrayList<String>
allergens: ArrayList<String>
+Meal(price: float): void
+Meal(price: float, mealName: String, description: String, mealType: String, cuisine: ArrayList<String>, ingredients: ArrayList<String>, allergns: ArrayList<String>): void
+offerMeal(): void
+stopOffering(): void
+getMealName():String
+getDescription(): String
+getMealType(): String
+getCuisine(): ArrayList<String>
+getIngredients(): ArrayList<String>
+getAllergens(): ArrayList<String>
+getPrice(): float
+getImageID(): String
+getOfferStatus: Boolean
+ getCookUID(): String
+setDescription(String): void
+setImageID(String): void
+setCookUID(String): void
+createPurchase(docID: String, cookUID: String, clientUID: String, mealName: String, imageID: String, pickTime: LocalDateTime, cookName: String, clientName: String, status: String): Purchase
+ updateFirestore(Context): boolean
Purchase
- cookUID: String
- clientUID: String
- mealID: String
- cookName: String
- pickupTime: String
- status: String
- requestTime: String
- clientName: String
- imageID: String
- complaint: DocumentReference
+ PENDING: String
+ ACCEPTED: String
+ REJECTED: String
+ Purchase(requestTime: String, cookUID: String, clientUID: String, mealName: String, imageID: String, pickupTime: String, cookName: String, clientName: String, status: String): void
+ getMealID(): String
+ getCookName(): String
+ getCookUID(): String
+ getClientUID(): String
+ getStatus: String
+ getPickUpTime(): String
+ getRequestTime(): String
+ getImageID(): String
+setComplaint(DocumentReference): boolean
+setImageID(String): boolean
+ updateStatus(String): boolean
+ updateFireStore(): boolean
ComplaintModel
+ cookName: String
+ clientName: String
+ time: String
+ title: String
+ description: String
+ cookUid: String
+ clientUid: String
+ docID: String
+ status: boolean
+ getDocID(): String
+ getCookName(): String
+ getClientName(): String
+ getTime(): String
+ getTitle(): String
+ getDescription(): String
+ getCookUid(): String
+ getClientUid(): String
+ getStatus(): boolean
+ setDocID(String): void
1
*

Contribution Table

	Deliverable 1: Oct 21	Deliverable 2: Nov 6	Deliverable 3: Nov 18	Deliverable 4: Dec 7
Elsa Ko	Base classes (User, Cook, Client, Meal, Menu - deprecated), UML, Register activity, Login=>Welcome, Connected login with welcome, send CC info & cook description to FB, opening phone gallery, uploading photo to cloud storage.	Fetching complaints from FB then populating complaint recycler (with intent passing). Movement between register activities.Do not allow doc creation if register activity doesn't complete. Significant debugging work.	Message for suspended cooks, utility class for photo upload (cloud storage), upload ingredients and allergens to FB from chipgroup, offer meal toggle & text, meal image sizing constraints, link welcome to correct userHome class, linked image with the appropriate meal	Querying offered meals from non-suspended cooks, made Purchase.java (trying to implement proxy pattern) which sends to FB, Debugged meal search with Tess, Link recent purchases with meal images, Debugged writing purchases to FB with Saranki, Debugged complaints not appearing for admins with Saranki
Katie Liu	UML. Assisted in setting up version control. Connected project to Firebase Authentication, Firestore, and Storage. Contributed to, debugged, and created field validations for registration activities. Contributed to User classes.	Local unit tests. Contributed to Admin class. Implemented cook suspension in Admin class and ComplaintView activity with firebase dependencies. Configured cook suspension message in Welcome class.	Implemented field validations for AddMeal activity. Implemented image download from FB storage (even though it wasn't required 🥺). Implemented recycler viewHolder button configurations. Resolved remaining del.2 bugs.	Modified both Meal and Purchase recycler to accept multiple viewHolder models by discriminating user type. Modified User classes and restructured Firestore documents for convenient user object reference, and better overall code design. Contributed to cook Profile activity, and configured validations for client purchases. Assisted in debugging Purchase class and ClientHome activity.

Tess Harper	Connected all register activities, UML	Dismiss and suspend buttons with help from teammates	Remove Meal, UML, Unit Tests, Keep meal image on modify button, Debugging Add Meal. Made only certain button available based on type. Ingredient & Allergen text boxes.	Unit Tests, Meal Search, Update Firebase on purchase request creation. Added toasts for meal search. Debugged meal search with elsa.
Jacob Maurice	• Designed all the xml files (UI) • Designed the logo & splash screen animation (deprecated) • Implemented all field validations for registration activities	• Designed all the xml files (UI) • Updated the UML Diagram • Implemented the RecyclerView for complaints appearing in adminHome and set up its adapter • Connected complaints to the ComplaintView activity on-click & updated fields for complaintView accordingly	• Designed all the xml files (UI) • Implemented the recyclerView for meals appearing on the menu and set up its adapter • Connected the meal cardviews to mealView and updated the fields accordingly • Implemented a modify button in mealView and connected it to the addMeal activity with the fields filled out in order to allow a user to modify an existing meal. • Implemented the offer status toggle and all related functionality in order to distinguish non-offered meals from offered meals	• Implemented the recyclerView for purchases requests and set up its adapter • Implemented the functionality of the approve / reject purchase buttons • Designed the profile's xml and updated all the fields in the cook's profile for both cooks and clients viewing it • implemented the recyclerView for searchedMeals and set up its adapter • Debugged the final project, by testing all final features of the application and fixing bugs accordingly

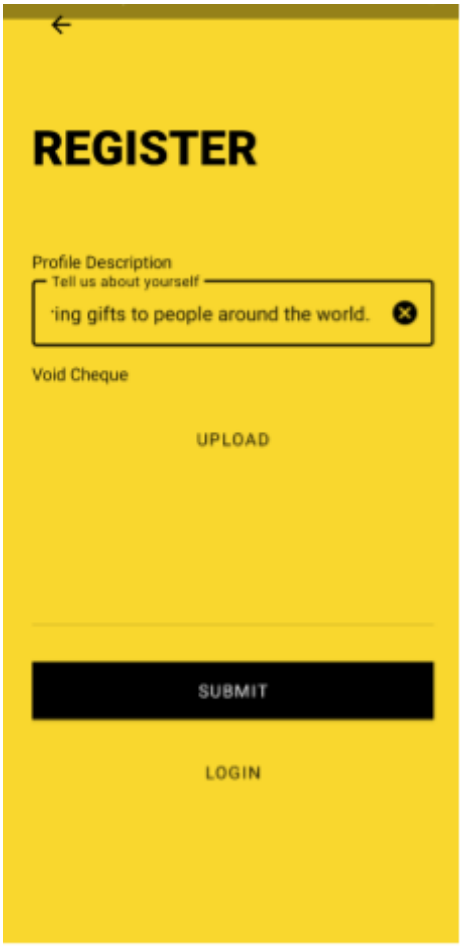
Saranki Mahendran	Admin.java, Welcome.java, UML Diagram	Dismiss and suspend buttons with help from teammates	UML Diagram, Ingredient & Allergen text boxes	ClientHome.java (Submit Complaint and Rate Cook), UML Diagram, Purchase_Recycler Adapter.java, xml pages associated with ClientHome, Debugged Purchase class and ClientHome.java with Elsa.

Screenshots

User Registration and Log-in

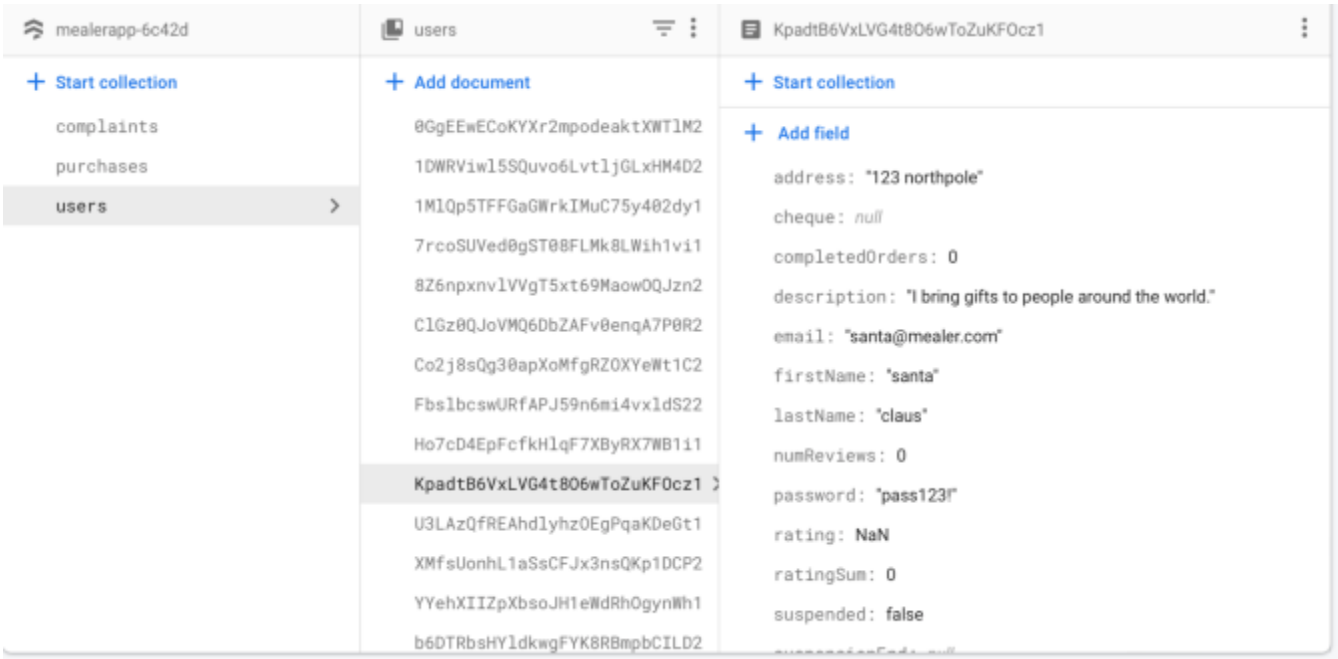
Cook sign-up and log-in:

*Landing page for **all** user types.*



Third registration step for Cooks

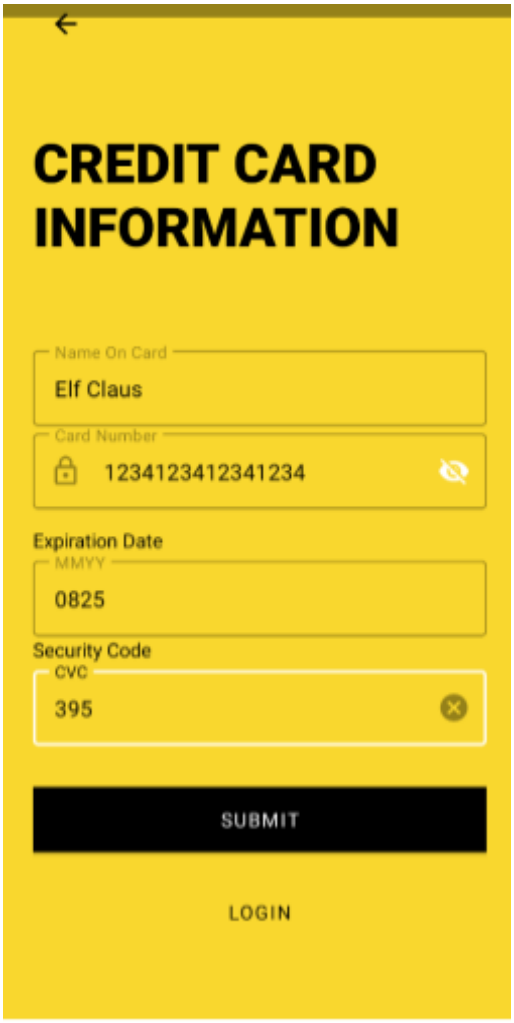
Welcome page for Cooks



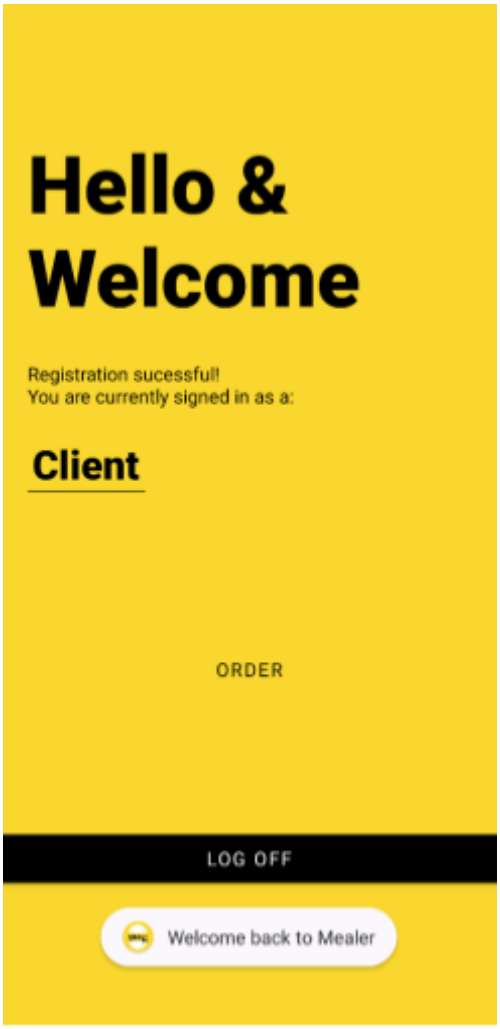
Firebase Firestore once a cook account completes registration activities.

Client sign-up and log-in:

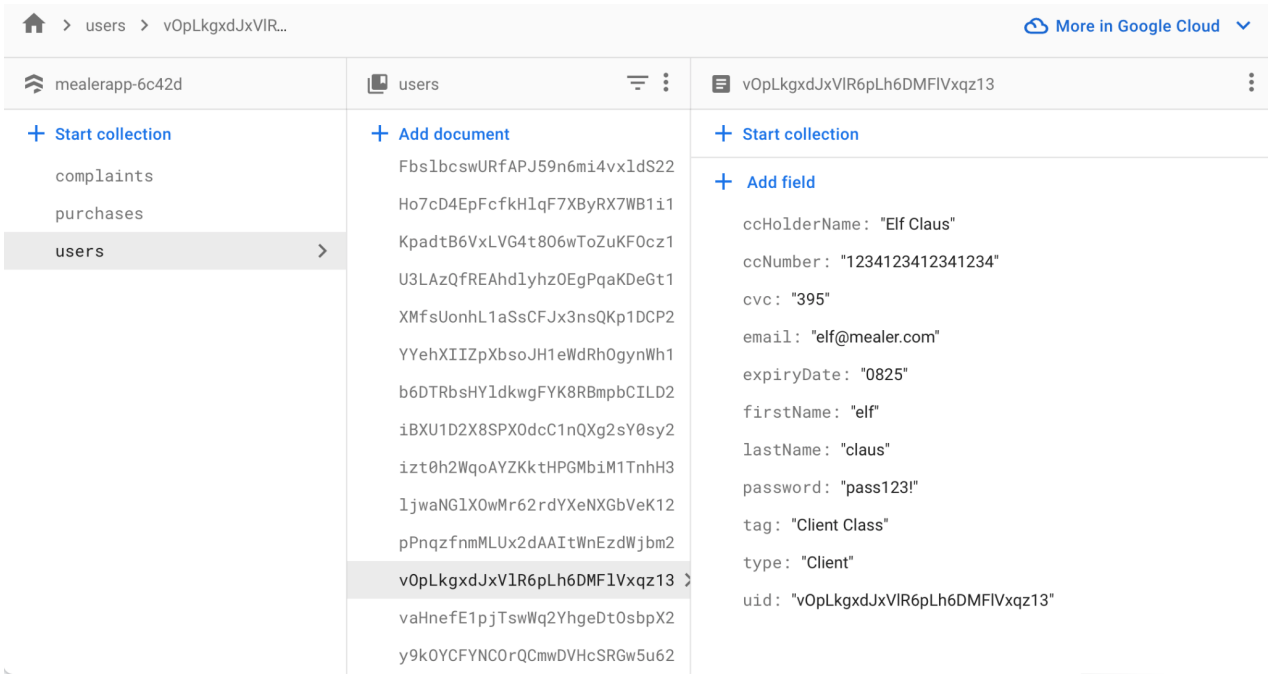
(registration step for user identification fields is not attached for the sake of space).



Third registration step for Clients



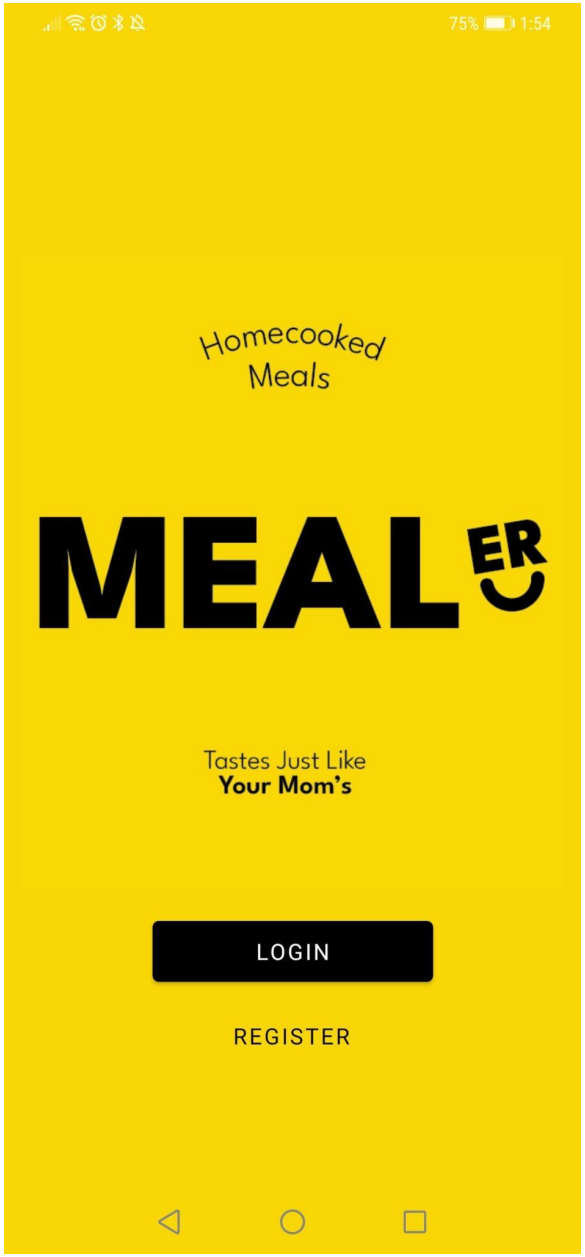
Welcome page for clients



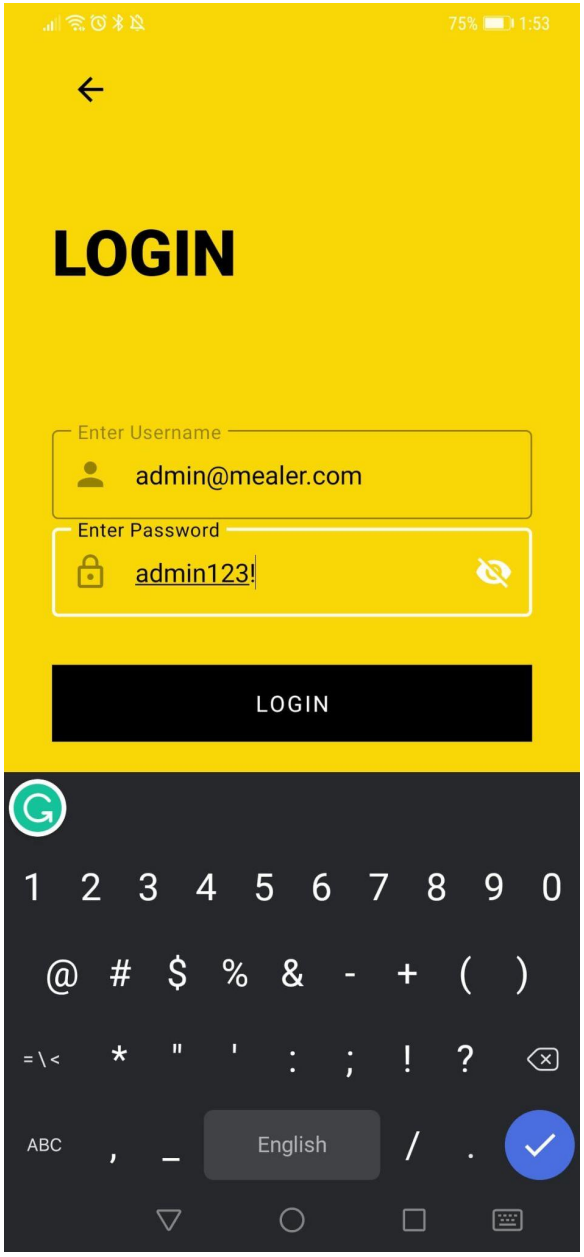
Firebase Firestore once a client account completes registration activities.

Admin Features

Step 1 - Admin opens the application

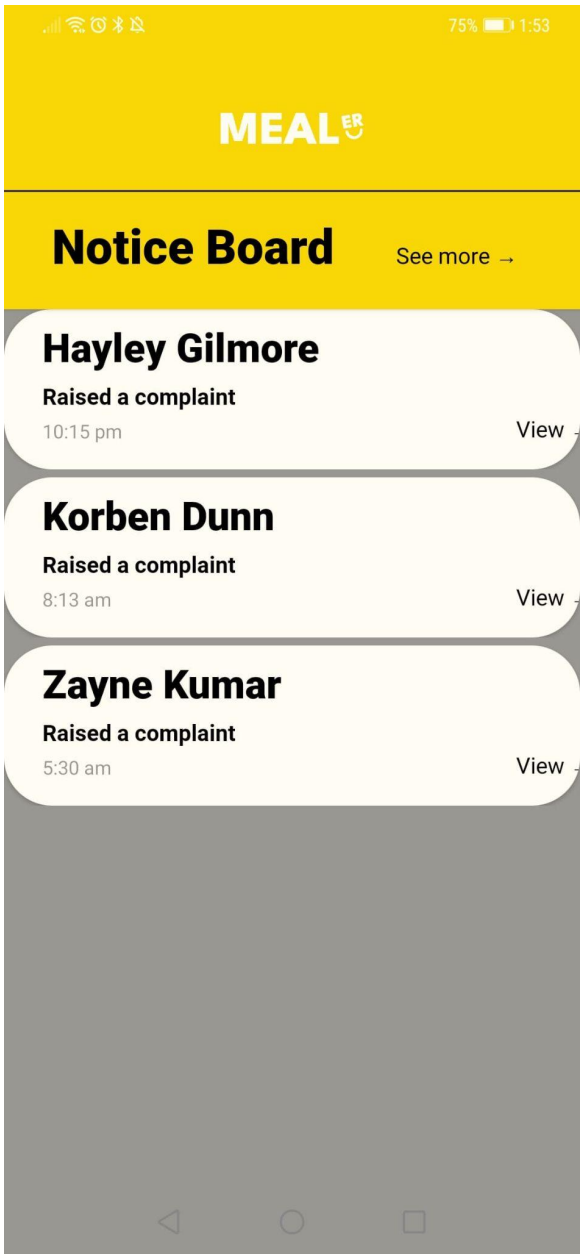


Step 2 - Admin enters their login credentials

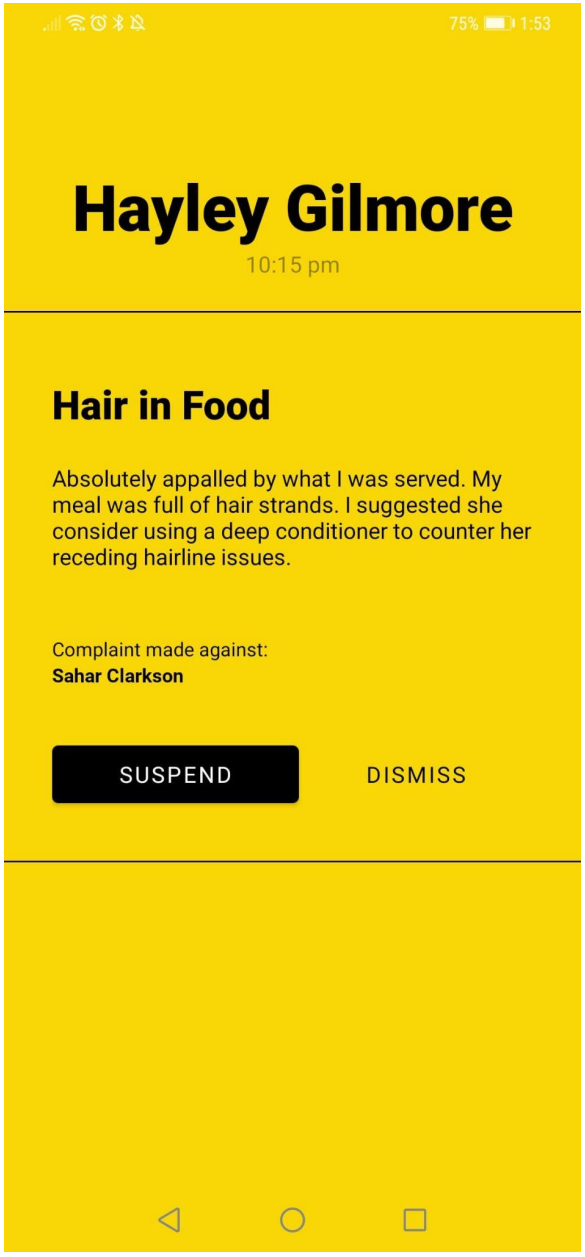


Step 3 - Admin is prompted to view the complaint board upon logging in

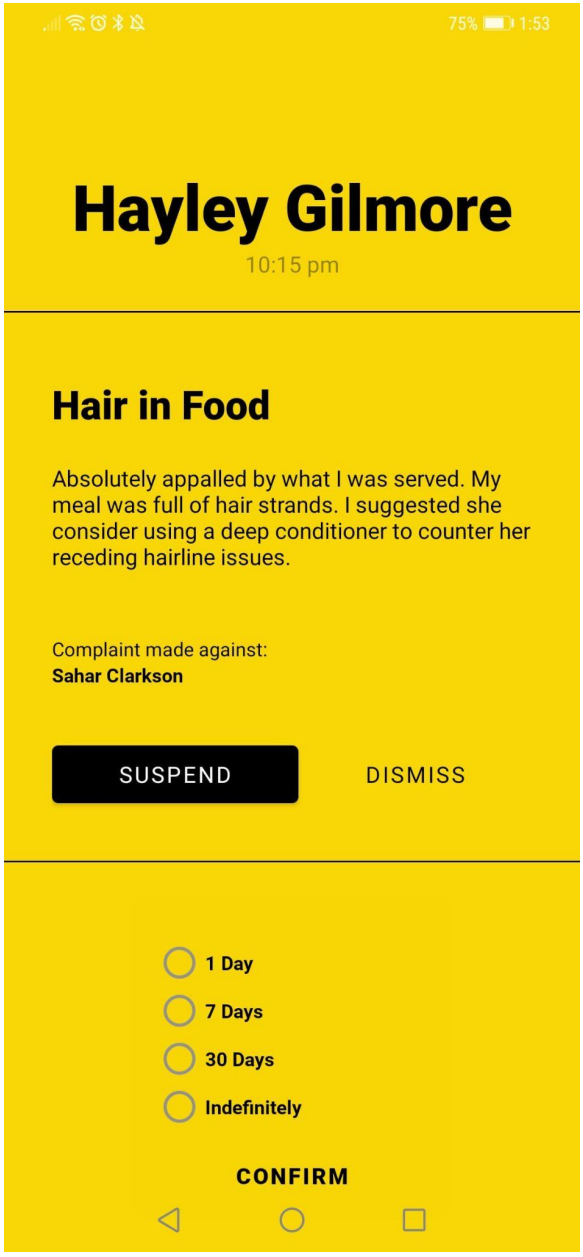
Step 4 - Clients who have raised complaints against a cook appear on the admin's notice board.



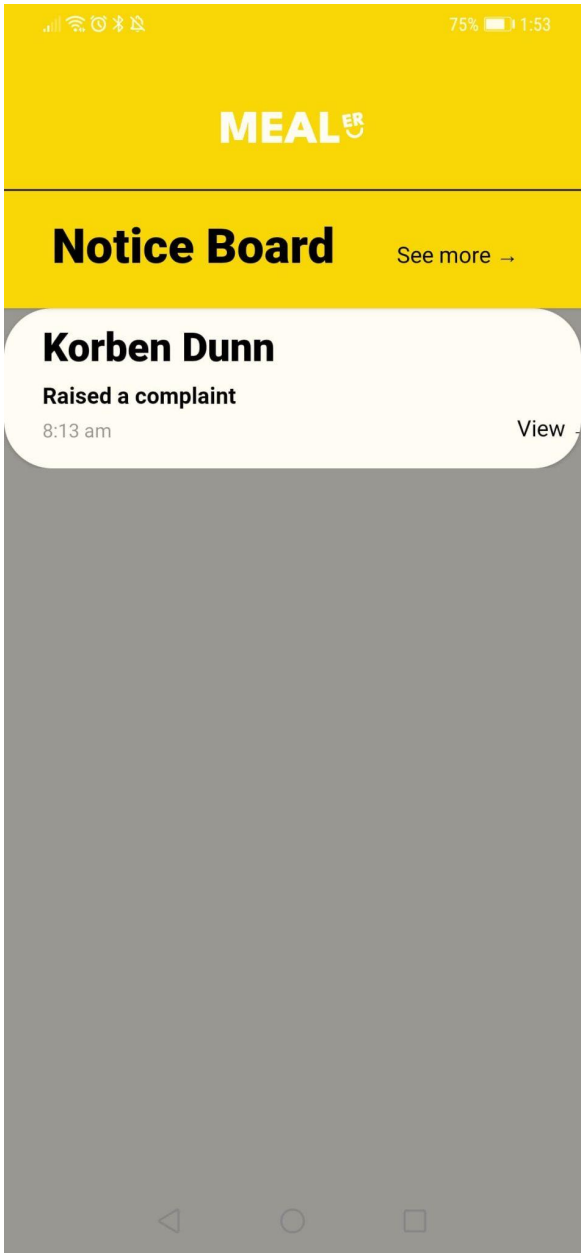
Step 5 - Admin can click on any of the complaints, which in turn displays additional relevant information.



Step 6 - A cook can either be suspended or dismissed. The suspend button prompts the admin to select a suspension period.



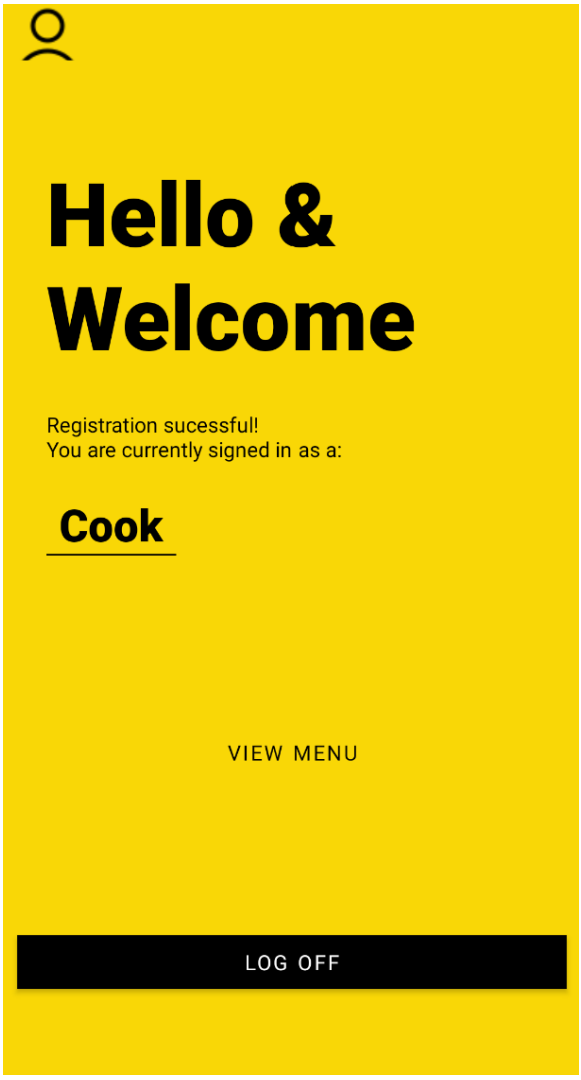
Step 7 - Once either button is clicked, the complaint disappears from the notice board.



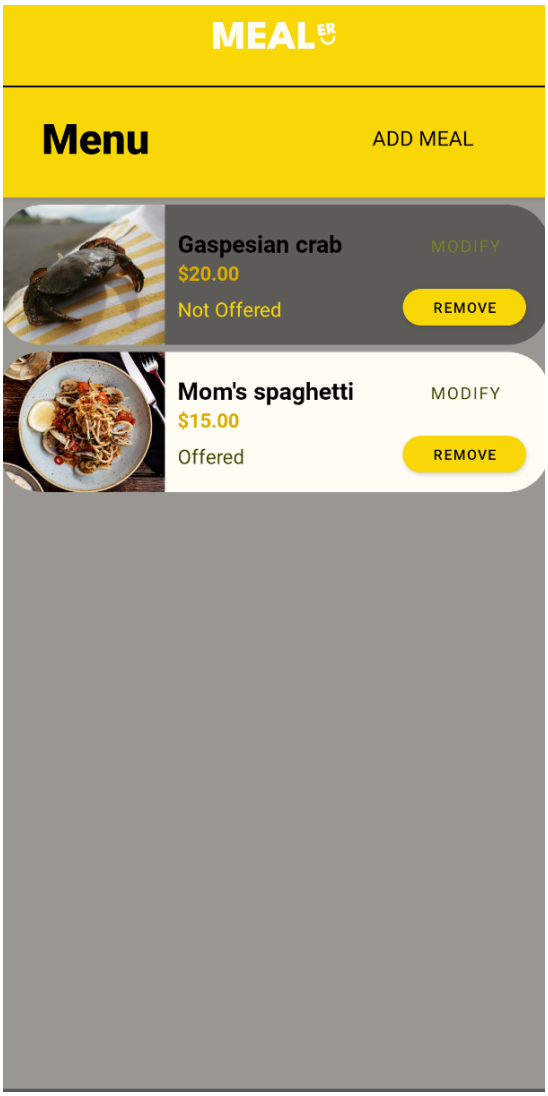
Step 8 - Upon logging-in, a suspended cook will observe a message highlighting the time until which they are suspended. During that period, the cook cannot access any additional functionality on the app.

Cook Features: Add Meal

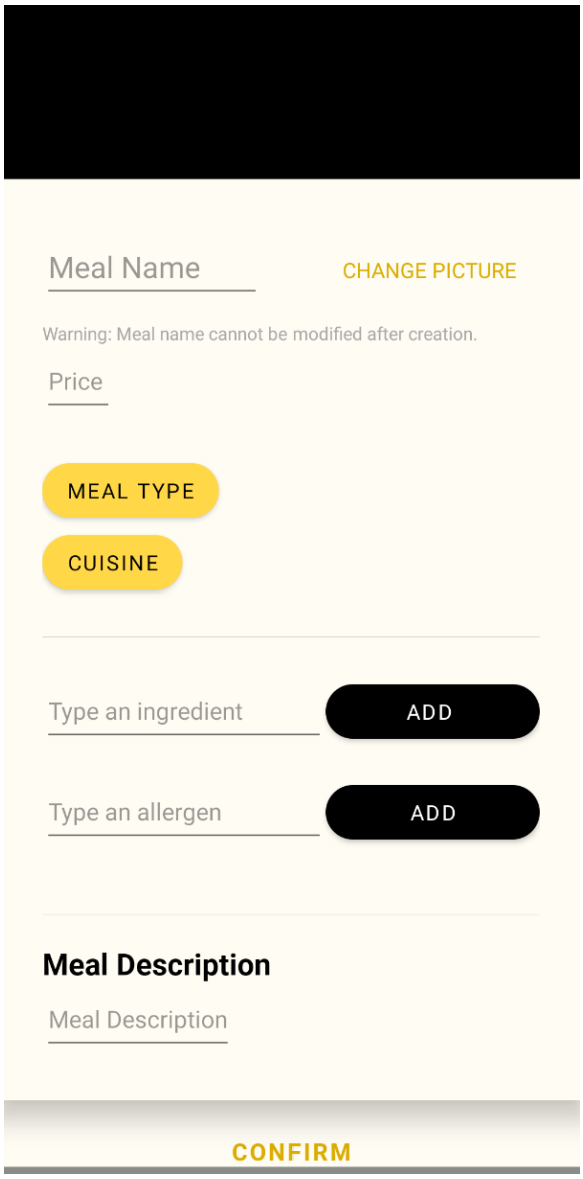
Step 1: Once logged in, cook users have the option to view their menu.



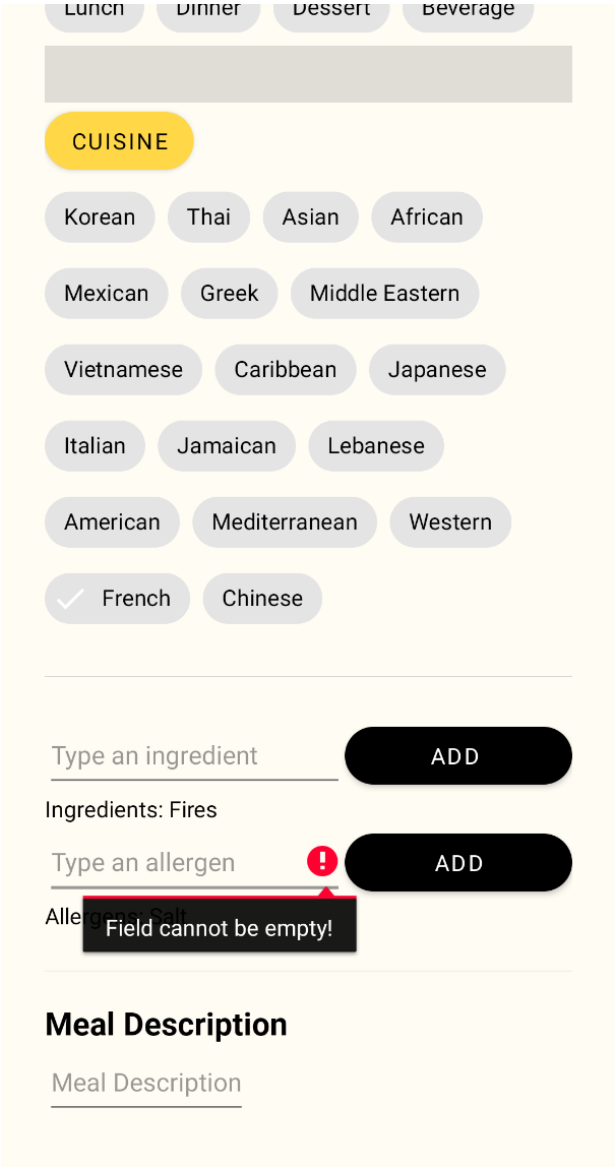
Step 2: Clicking the Menu button launches an activity displaying the respective cook's list of meals: both offered and not offered. Cook's also have the option to add a meal to their menu from this page.



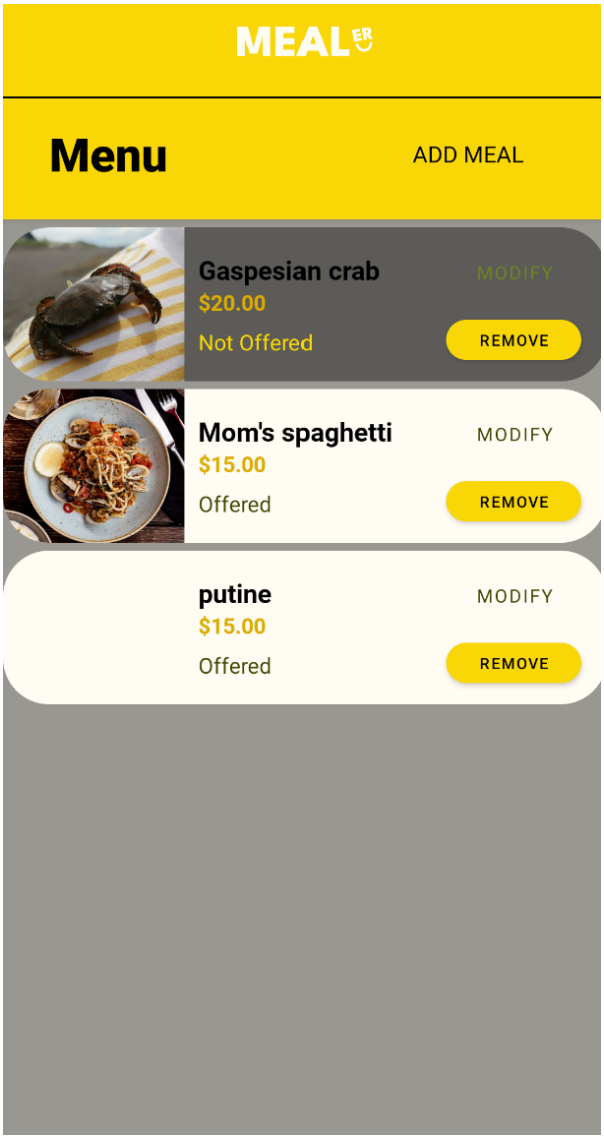
Step 3: After clicking the add meal button, cooks are directed to an activity that accepts a user's meal information; the form includes meal name, picture, price, meal type, cuisine, ingredients, allergies, and meal description.



Step 4: To complete the add meal activity, apart from allergens, every field must be filled with valid input, otherwise an error message will occur for the respective field(s).

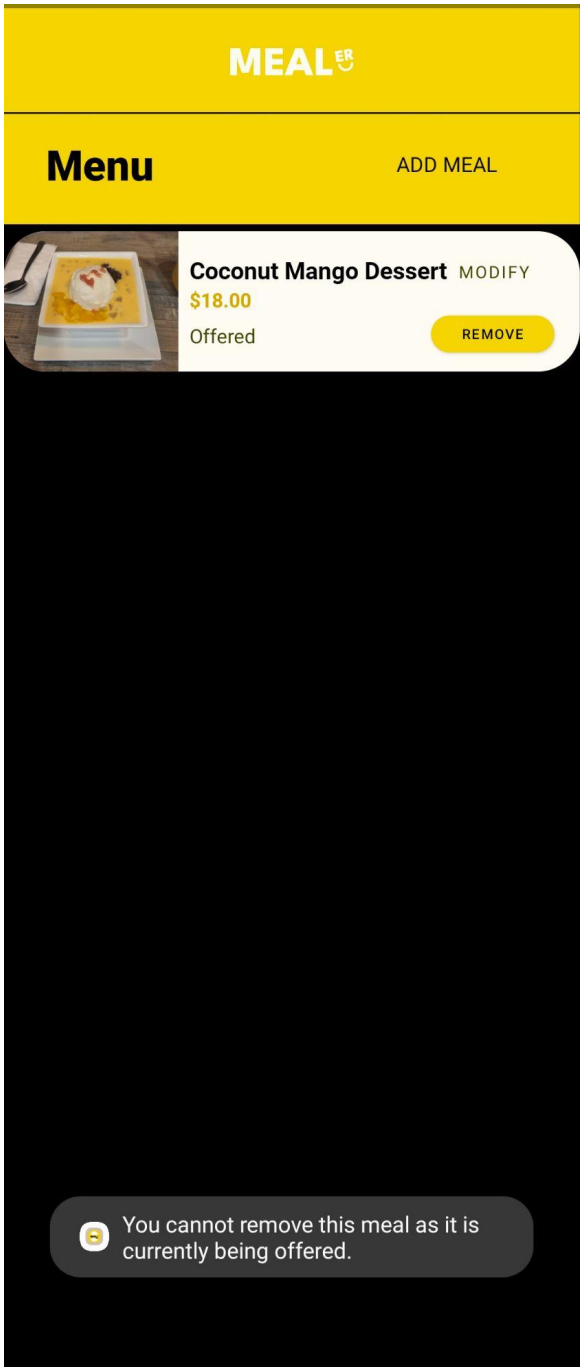


Step 6: Upon field completion, pressing the confirm button will direct cook users back to their menu page with the newly added meal.

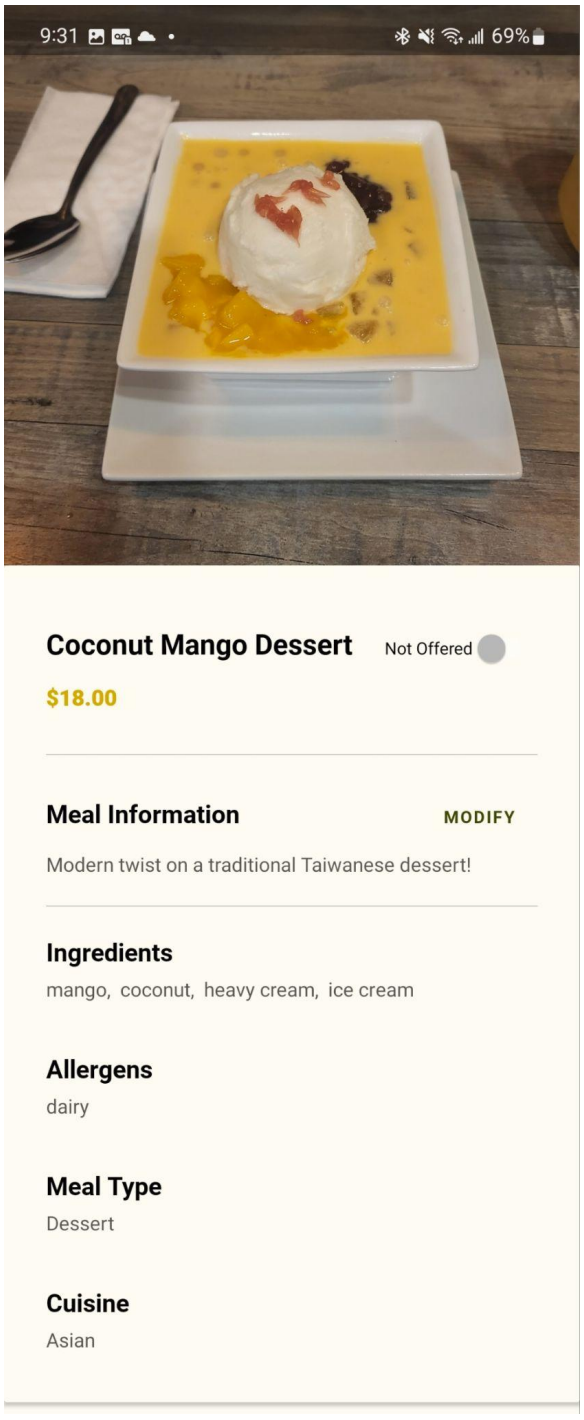


Cook Features: Modifying Menu (and *currently offered*)

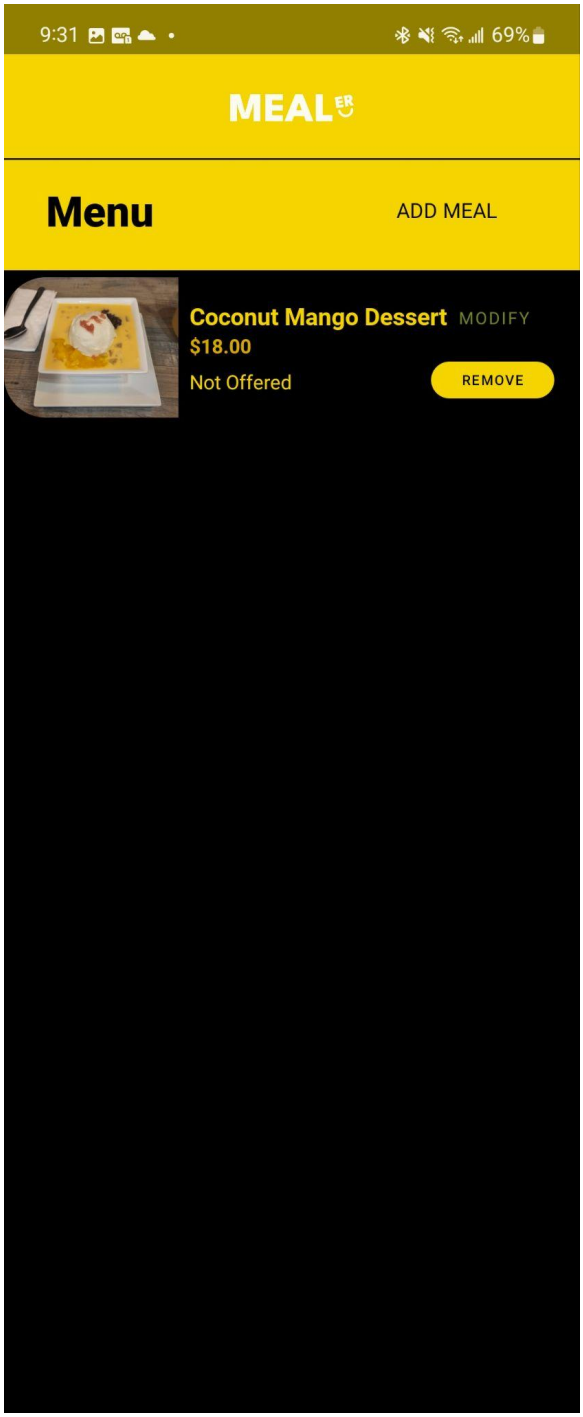
Step 1: A cook has an offered meal they want to make modifications to in their menu. Upon selecting the remove button, they are informed that the meal cannot be removed since it is currently being offered.



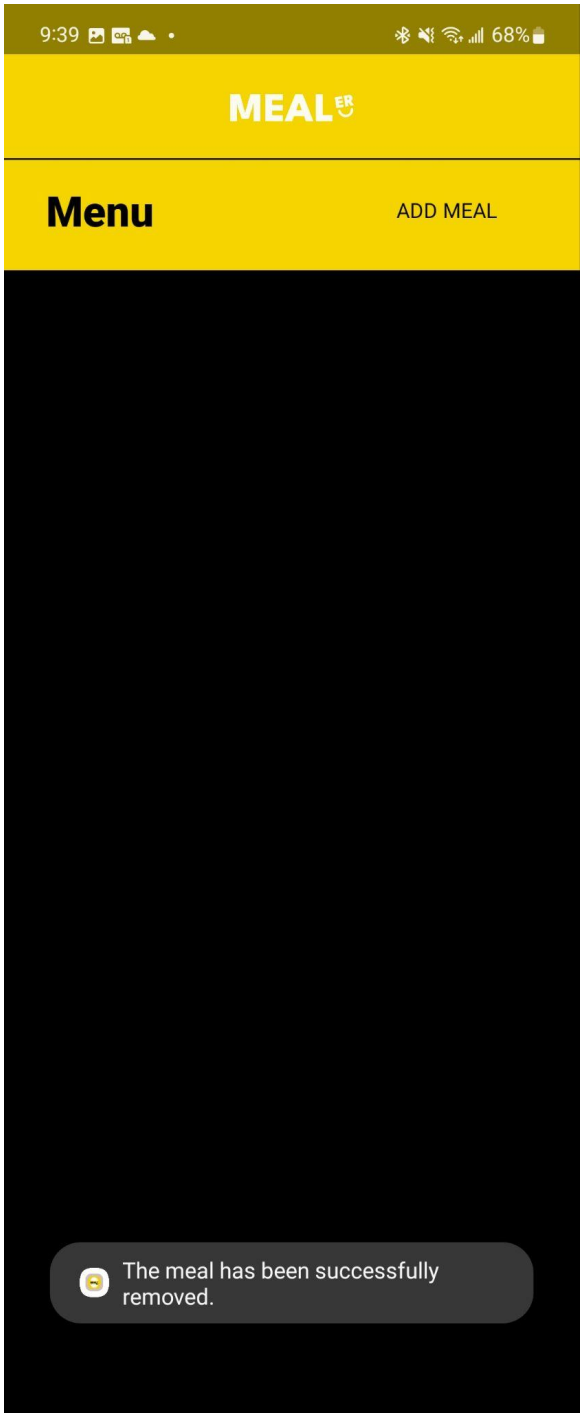
Step 2: To remove a meal from their *currently offered* menu, a cook must select the meal to launch the full meal view. When the *offer* button is toggled off, it moves left and changes to display “Not Offered”.



Step 3: After actioning back to the menu view, the respective meal is displayed as *not offered* in the cook's menu and the background color changes to reflect this.

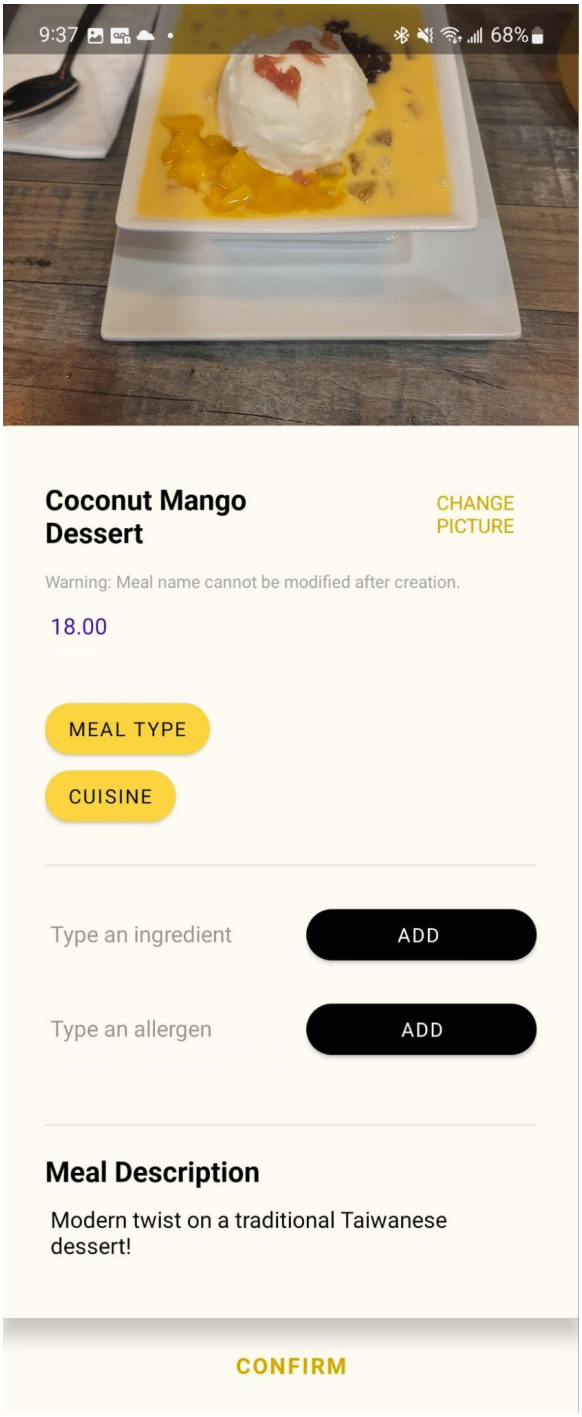


Step 4: With the meal status set to not *offered*, a cook can proceed with deleting the meal entirely. All related data is immediately removed from both their menu and the Firestore database.



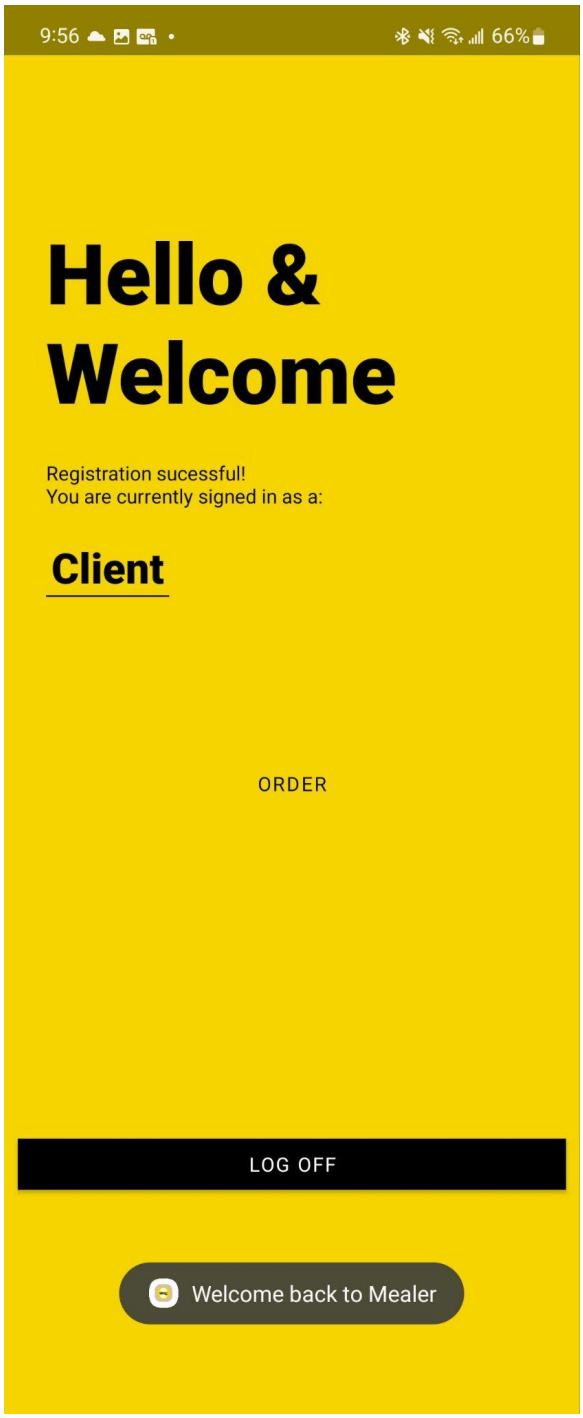
Cook Features: Modifying Meal

Step 1: To modify their meal information, a cook must click modify from their selected meal's expanded view. This button will launch the same activity used for adding a meal, with the meal image pre-loaded. However, users will not have the option to rename their meal.

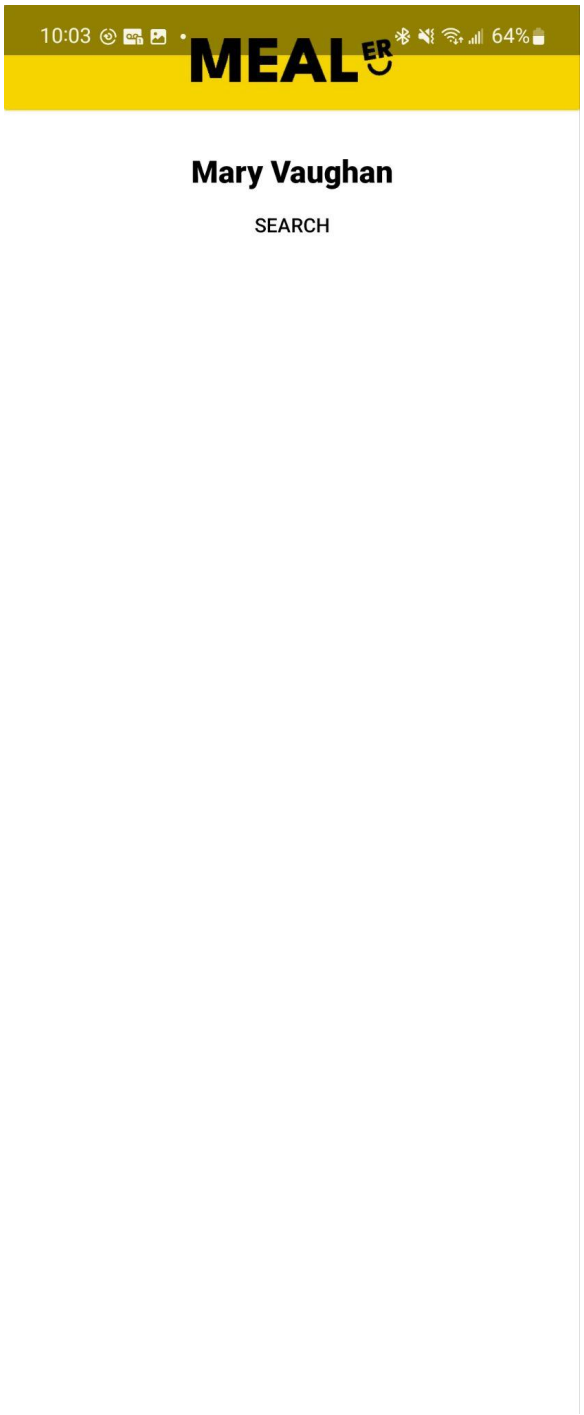


Client Features: Purchasing a Meal

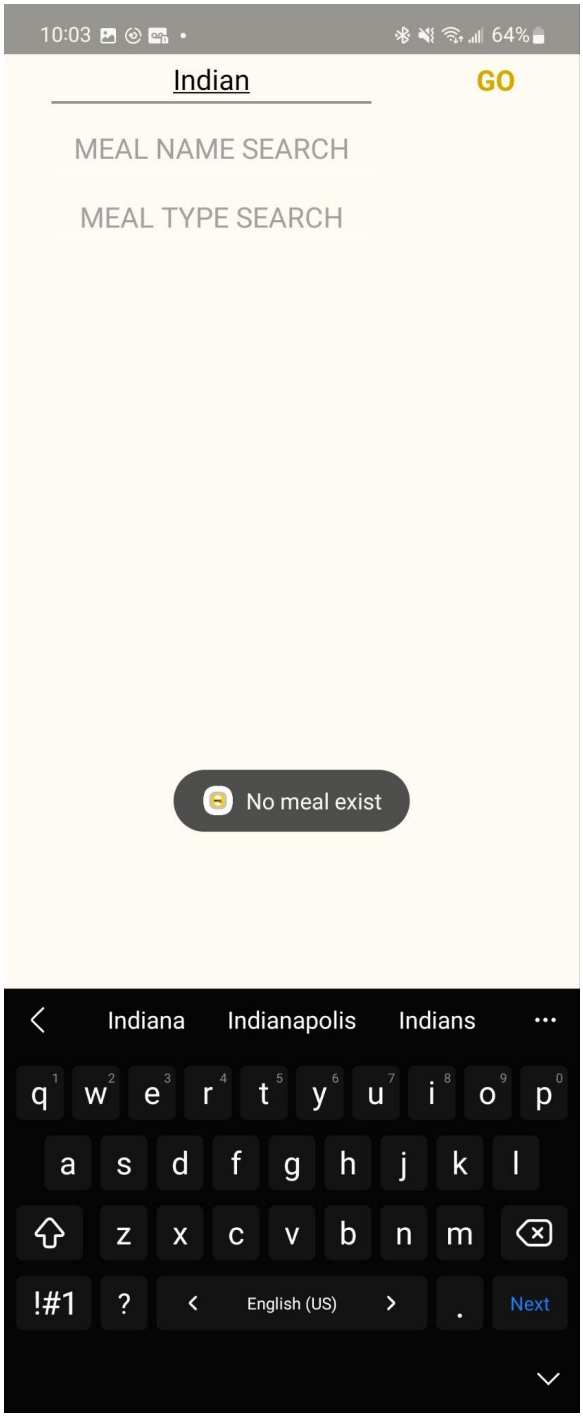
Step 1: From their welcome page, a client can both access their recently purchased meals by clicking the “Order” button.



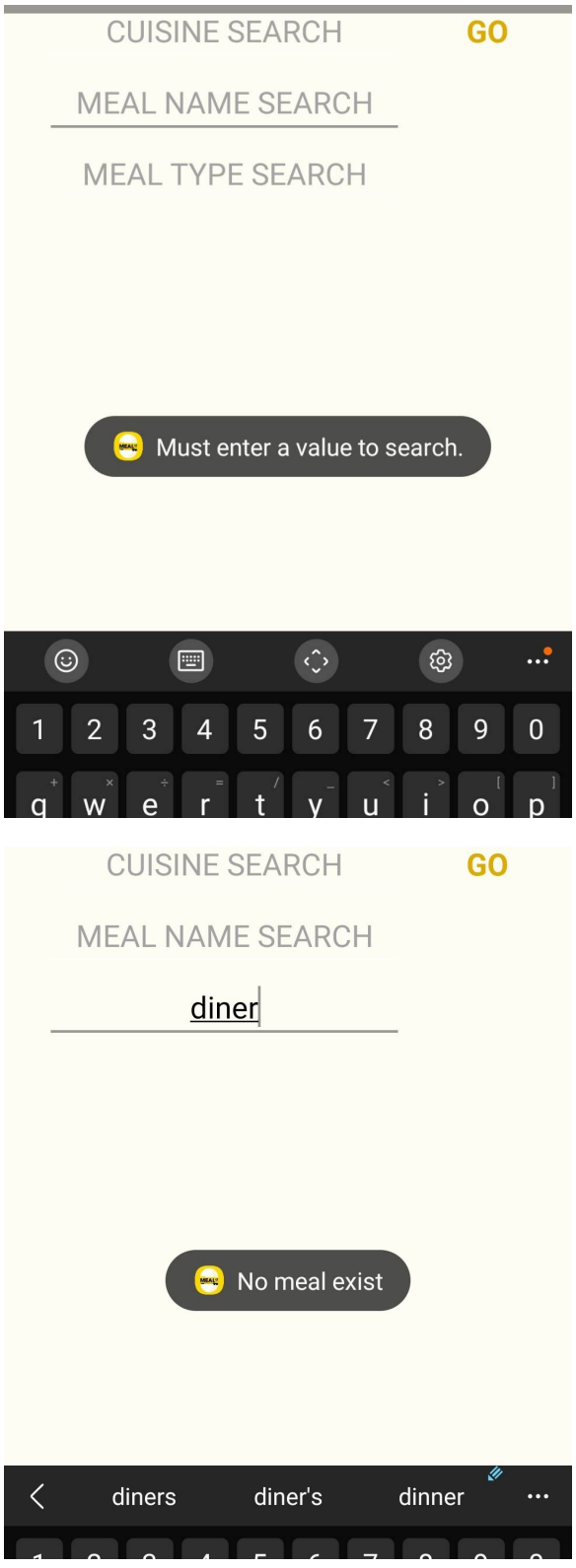
Step 2: If a client has not requested any purchases yet, their screen will display an empty activity. To browse for new meals and request to purchase a meal, a client can click the *search* button.



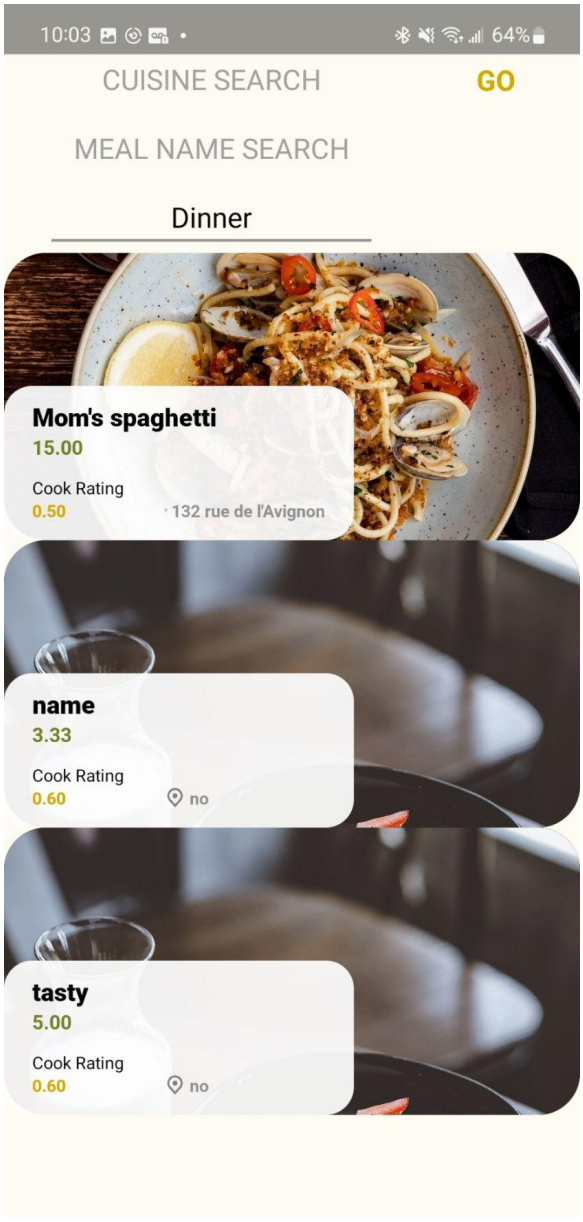
Step 3: Clicking the *search* button will launch the search activity. Clients can search for meals offered exclusively by non-suspended cooks by cuisine, meal name, and meal type.



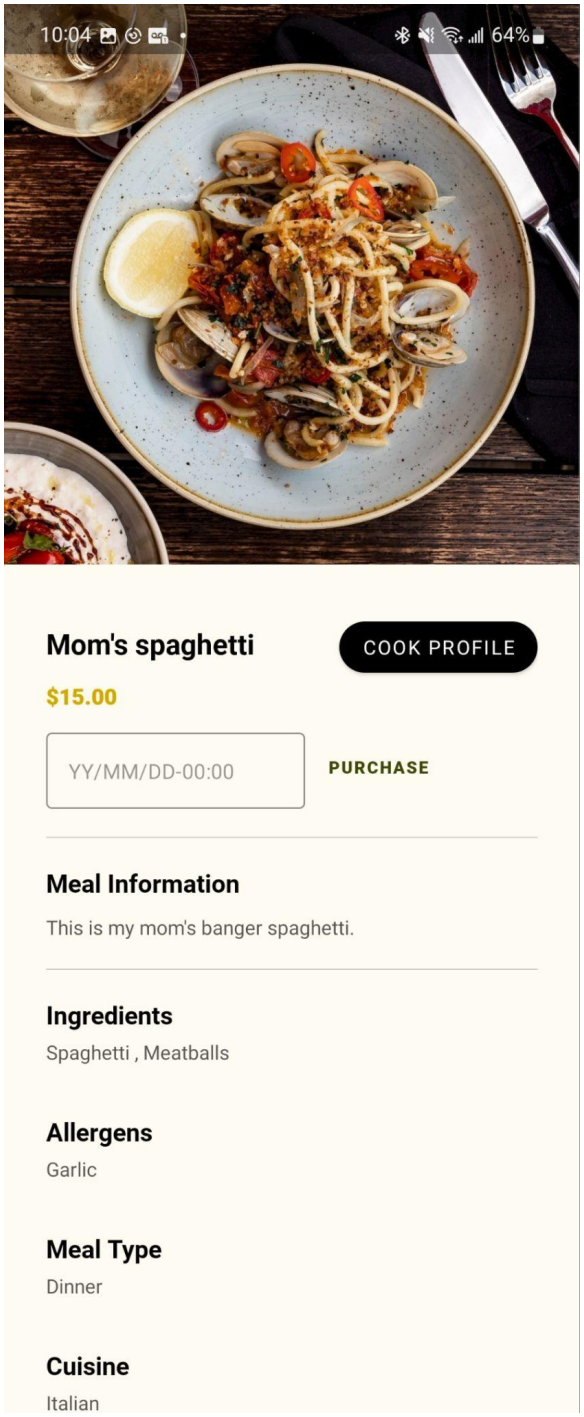
Step 3.5: For best querying, users are advised to type at least one field or have the correct formatting.



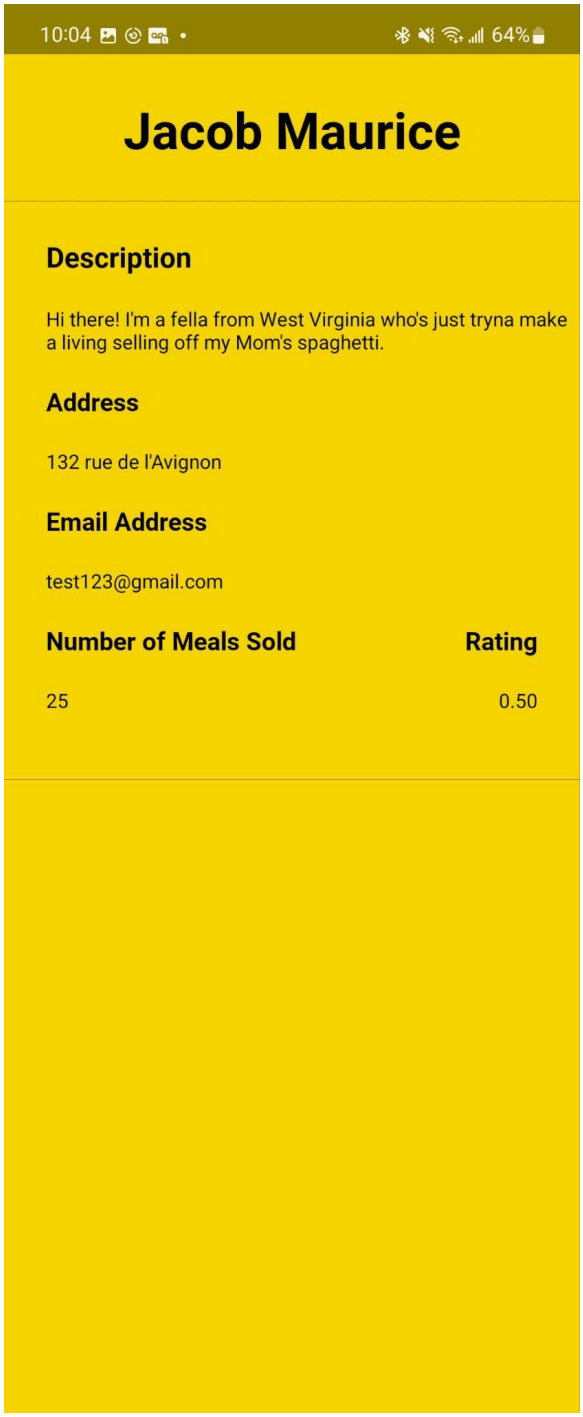
Step 4: Upon a valid search query, a client can view meal results offered by non-suspended cooks.



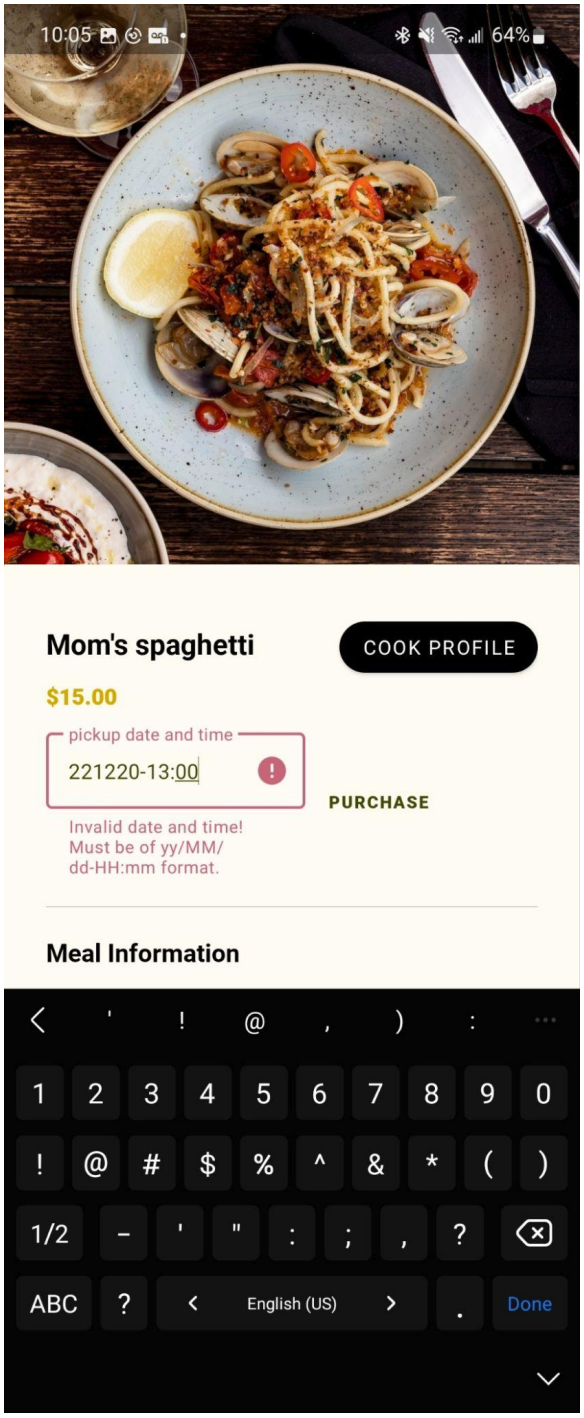
Step 5: Selecting a meal from their search results will launch the expanded meal view with additional meal information. Clients have the option to view the meal's cook profile before requesting to purchase.



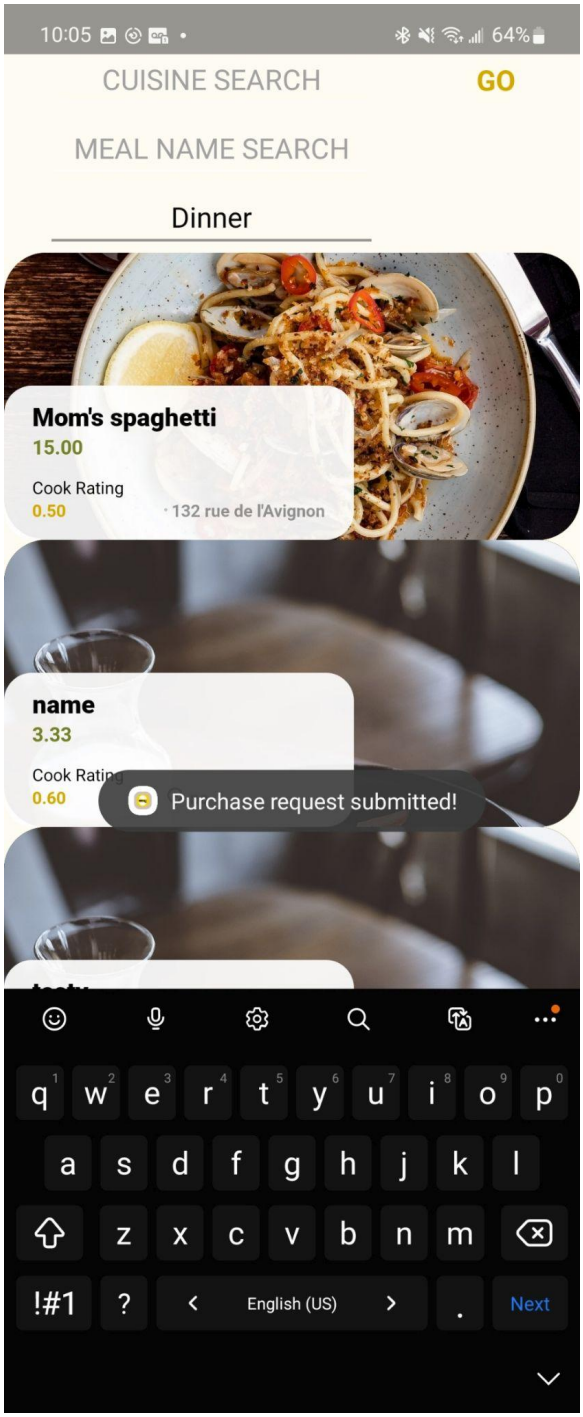
Step 6: Clicking the *cook profile* button will direct clients to view the cook of interest's information, including their average rating.



Step 7: To request to purchase a meal, clients are required to type in a valid date and time when they would prefer to pick-up their meal, otherwise they cannot submit their request.



Step 8: Upon submitting a valid pick-up time, clients will be directed back to their search activity with a toast notifying their request has been received.

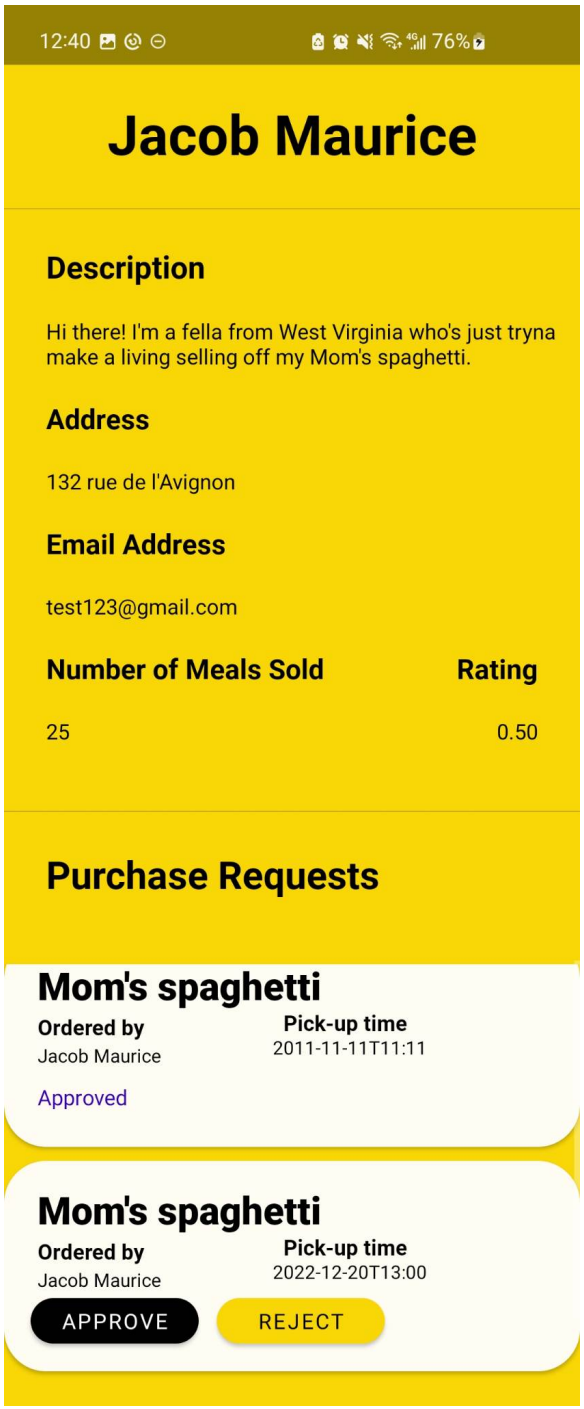


Step 9: On their homepage, clients can view their purchase request(s) and check their status(es).

Cook Features: Access Profile

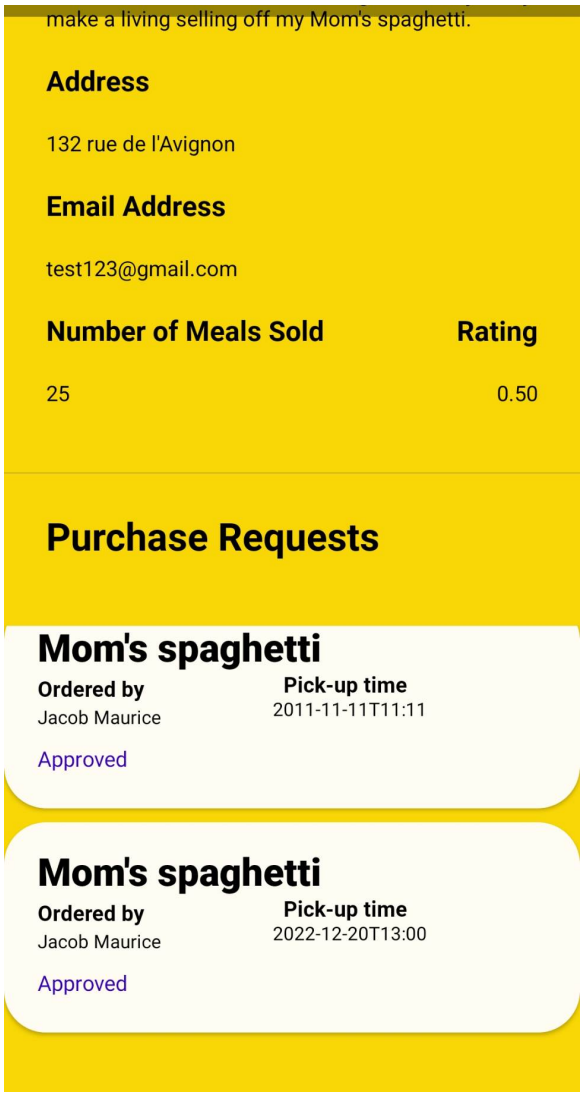
Step 1: An unsuspended cook can view a profile button at the top left corner.

Step 2: Clicking it opens their own profile view with latest purchase requests and other personal info



Cook Features: Action Purchase Requests

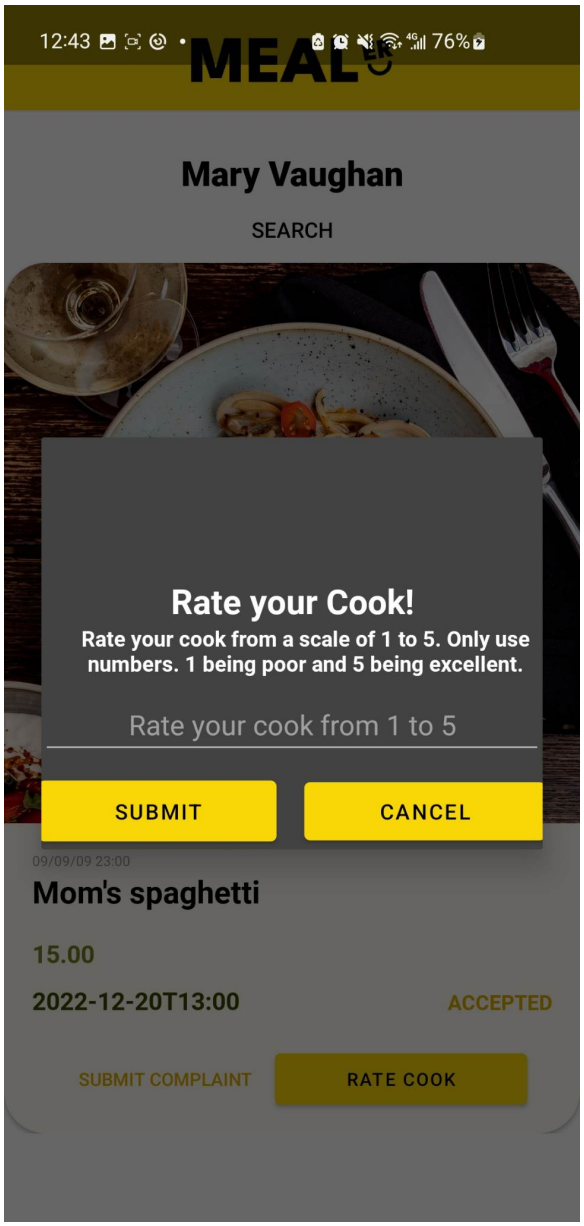
Step 3: A pending purchase request can be actioned by pressing accept or reject.



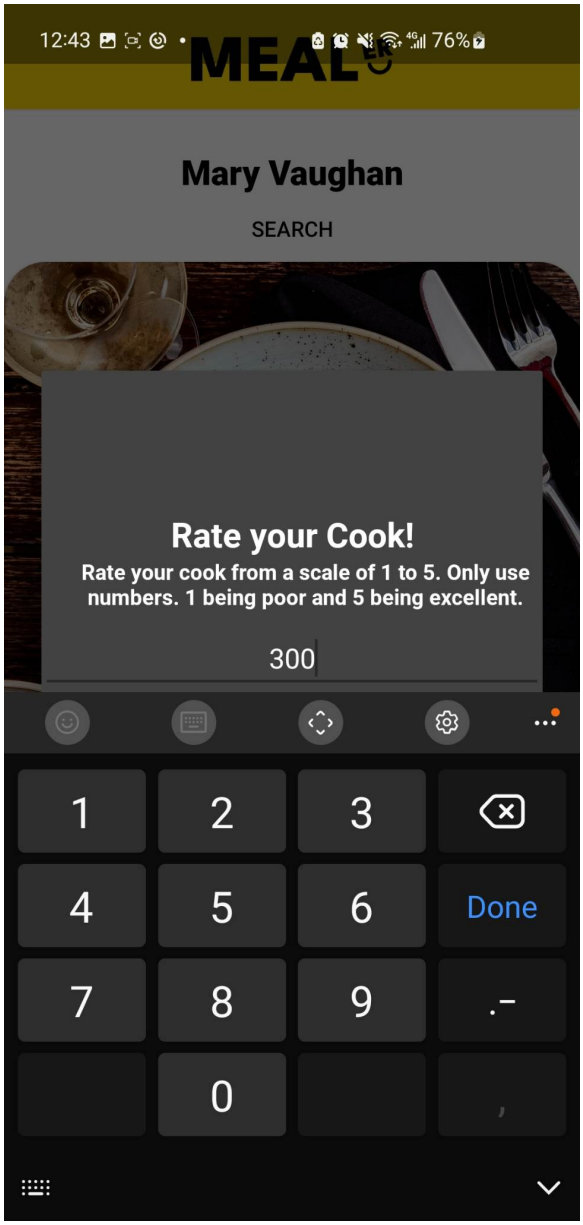
Client Features: Status change & Rate Cook

Step 1: A client's accepted meal purchase request shows as accepted rather than pending

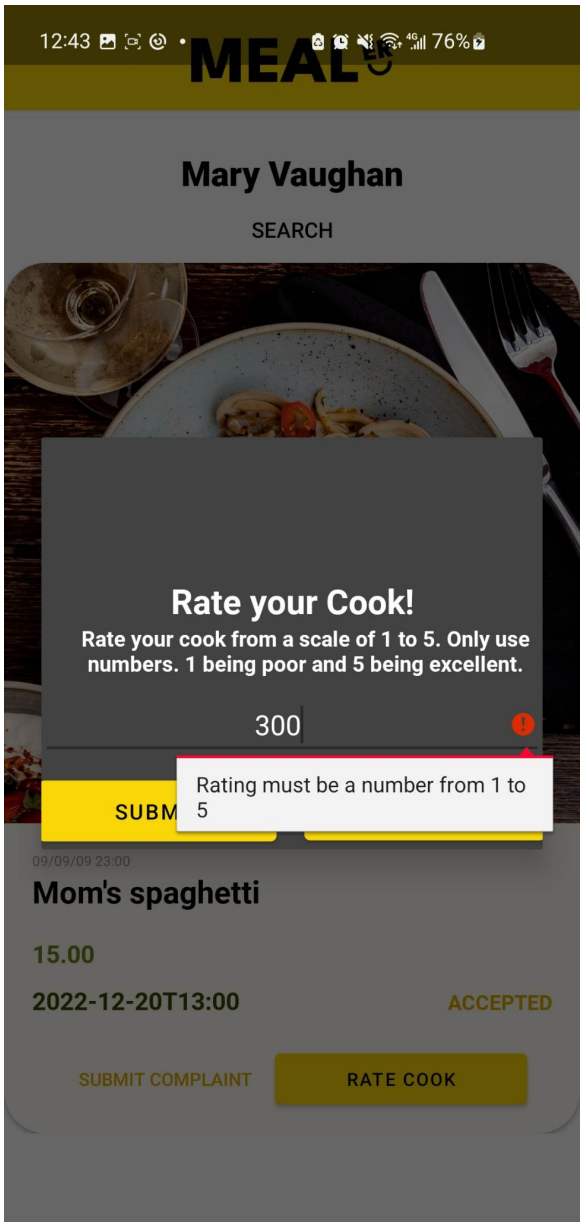
Step 2: The client can rate the cook from a pop-up dialog.



Step 3: When clients enter a rating, a number keyboard appears to restrict alphabetical and other non-related characters to decimals.



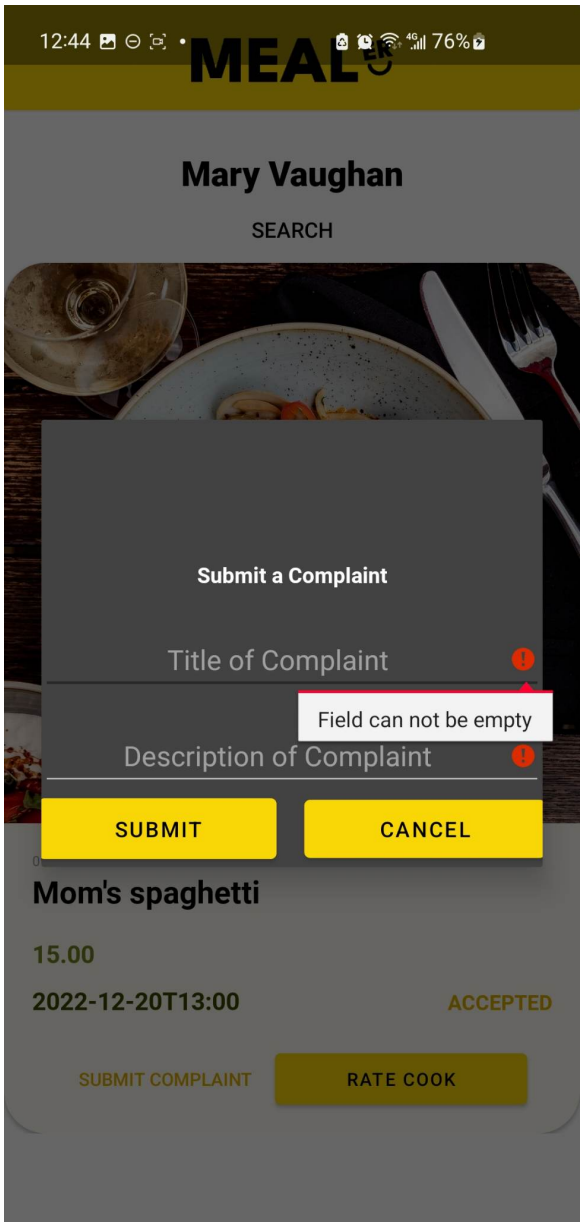
Step 4: Illegal entries will not be allowed due to field validations.



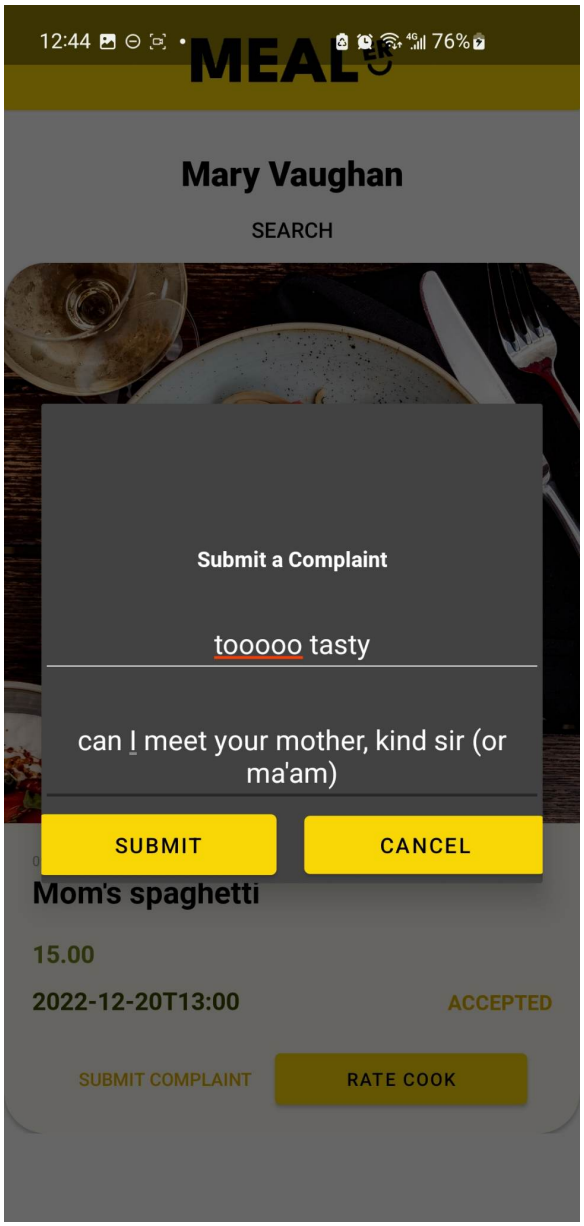
Step 5: Once the field validations pass, the rating is sent to firestore.

Client Features: Submit Complaint

Step 1: A complaint can be created on an accepted purchase. As usual, these fields are verified first.

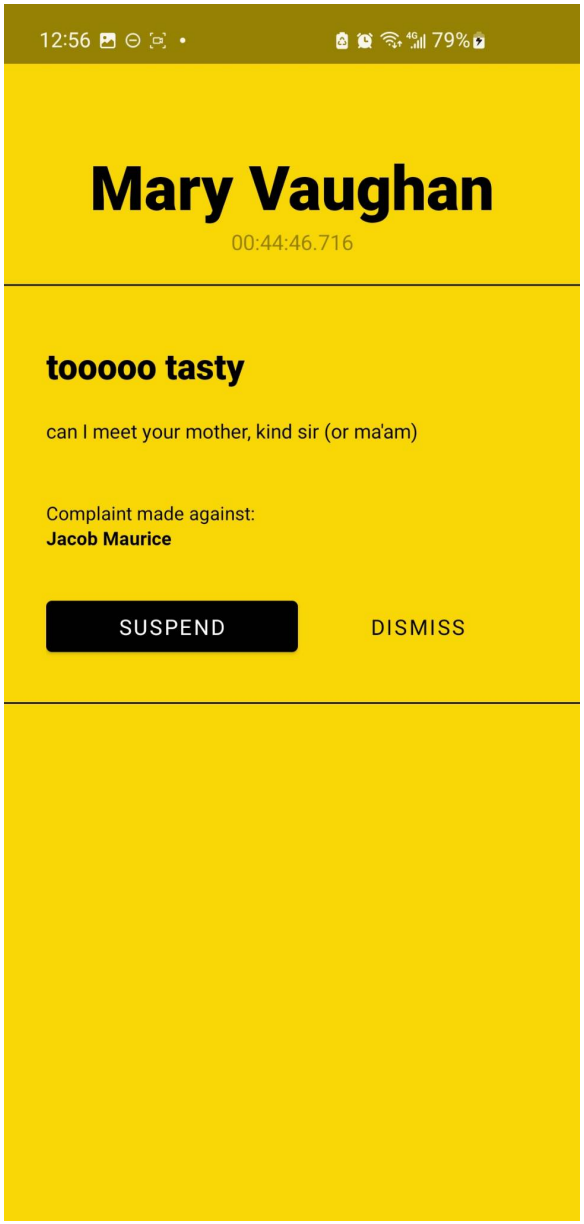


Step 2: The title and description fields will alter how the admin sees the complaint.



Step 4: When the complaint is submitted to firestore, the client sees a toast.

Admin View: The newly filed complaint will appear in the admin homepage where further action can be taken.



Lessons Learned

Throughout the completion of this project's deliverables, we have encountered and overcome countless adversities while also gaining new skills. The most notable skills being, but not limited to, the ability to learn new software, coordinate to meet deadlines, and independently study.

We are grateful for this opportunity to gain such a considerable breadth of experience in databases and Android SDK for application development. For instance, important knowledge, like debugging code with a built-in IDE debugging tool, adding and referencing Firebase data, and reading and understanding code written by others, were some of many concepts we have realized and improved on throughout the duration of this semester. Additionally, learning how to write maintainable, easy to read code and effectively using GitHub as version control was crucial, to say the least; they were the practices that played a key role in the overall increase in productivity by keeping our team on the same page and supporting simultaneous contributions with minimal conflicts.

Moreover, working in a team environment was not an easy feat, to say the least. It was essential for us to quickly grasp the following skills: delegation of responsibilities, clear and timely communication, and coordination of team time management. Fortunately, these tasks became much easier with time. Communication and appropriate task delegation to individuals was critical as each member had different experiences and expertise. This was especially helpful with balancing the number of commits between members. Similarly, the realization that splitting off into smaller sub-groups instead of attempting to account for five schedules greatly improved our efficiency.

Finally, our most significant lesson taken from this course project is independent learning and time management. After the second deliverable, independent learning became much more important as the amount of code involved with the project increased considerably. To elaborate, prioritizing time to independently study code committed by peers and recognize the program's dependence with Firebase became increasingly crucial in our group's productivity. This was especially true as the deadlines for deliverables became shorter with each increment.

To conclude, there were many lessons learned over the last four months, but as a group the most prevalent ones were learning new software, working timely in a group setting, and learning independently.