SEG2105 Deliverable 4 Final Report:

Android Project: Mealer Application

Group 3

Noah do Régo - 300234846

Pradyu Vasudev - 300231804

Samuel Braun - 300238833

Maria Khalil - 300242332

Vivethen Balachandiran - 300245080

School of Electrical Engineering and Computer Science, University of Ottawa

SEG2105[A] Introduction to Software Engineering

Professor Hussein Al Osman

December 7th, 2022

Table of Contents

List of Figures

List of Tables

1.0 Introduction

This report provides an overview of the application, Mealer, that our group developed for the SEG2105 course. Mealer offers local, Ottawa-based cooks a platform to sell home-made meals to clients.

The UML diagram was updated for each deliverable as new classes and pages were created. The updated diagram demonstrates the relationships between classes and the various roles each type of user plays in the program.

Each group member contributed to the deliverables through the shared repository on GitHub and kept the rest of the group up to date with the changes made and any potential errors. Each member was able to learn how to produce aesthetic UI, utilize the real-time database, and expand on the functionality throughout the app's development. Although the app is not evaluated on its user interface, our group took the opportunity to learn how to develop aesthetic and interactive UI using different colours, layouts, and spacing.

Moreover, the report analyses the lessons learned in the planning and development phases of the application. After encountering problems regarding the first deliverable's deadline, our group employed quality documentation and learned how to improve readability of code for contributors, which saved time in later deliverables when members were updating and maintaining the code. In addition, we learned how to represent sets of occurrences in the class diagram using known patterns, and how to carefully apply patterns and justify those design decisions. Resolving difficulties regarding the timeline and the software design of our application will ultimately prepare us for future tasks in the workplace or in group projects.

2.0 UML Diagram

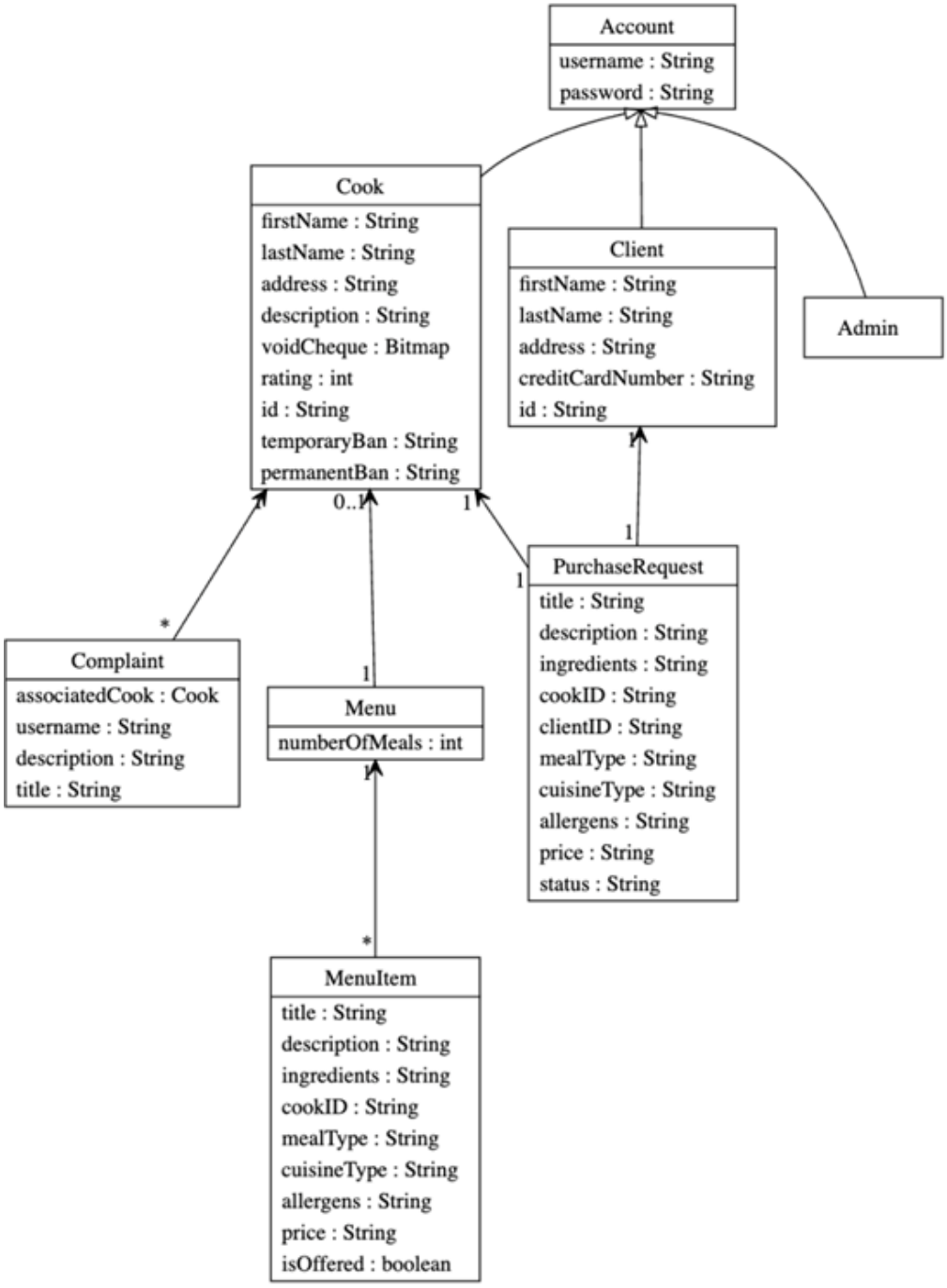

Figure 2.1. Final UML Diagram for the Mealer app (Activities removed for clarity)

3.0 Team Members' Contribution

Table 3.1.

Contributions made by each team member categorized by deliverable.

Member	**Deliverable 1**	**Deliverable 2**	**Deliverable 3**	**Deliverable 4**
Noah	- UML Class Diagram - Register Cook Page XML - Register Cook Page .java - Modified structure to follow OOP (Account Classes) - Linked Welcome XML pages - Added to README - Tested code	- UML Class Diagram - Added User-Database relation in firebase (using User ID) - Modified login to check for user - Added admin ability to action complaints - Cleaned up code - Tested code	- UML Class Diagram - Meal + Menu UI - Meal Pop-up functionality - Adding fields to meals - Cleaned + tested code	- UML Class Diagram - The Client can rate the Cook from which they have purchased a meal - Cook can view their profile and rating(s) - Refactor UI to use constraints (support different devices) - The Client can view the status of their purchase (pending, approved, or rejected)

				- The Cook can view and approve/reject purchase requests received from Clients.
Pradyu	- Register Client Page XML - Register Client Page .java - Logging off feature - Implemented original register functionality - Cleaned up code - Tested code	- Created Object-Oriented banning system - Created complaint displaying functionality in Admin inbox - Improved login functionality - Cleaned up code - Tested code	- Setting up menu UI for cook - Functionality for adding food item to menu + offered menu - Indicate if food is offered or not - Tested code	- Client search meal - Client can search results (not from suspended cooks) - Client can view meal information
Samuel	- User validation for Register Cook Page and Register Client Page - User validation for Login Cook Page and Login Client Page - User validation for General Login Page	- Created activities to be shown for a banned cook - Developed inbox for complaints - Implemented checks to see if cooks are banned - Cleaned code - Tested code	- Functionality for Deletion + Offering in Menu - Menu Layout - Cleaned + tested code	- The Cook can receive the purchase request submitted by the Client - The Client can submit a complaint about a Cook to the Administrator - Client can submit purchase request

	- Tested code			
Maria	- Created and designed Home Page XML (+ Logo) - Designed Login Client Page XML - Designed Login Cook Page XML - Tested code	- Created Admin inbox XML - Added dropdown system for user type on login page - Field validation for Register Page - Tested code	- Fix complaint inbox UI - Field validation for suspension date in AdminInboxActivity Page - Tested code	- Final Report (putting together all contributions, introduction, lessons learned) - User validation for Complaints page
Vivethen	- Designed Cook Welcome XML - Designed Client Welcome XML - Designed Admin Welcome XML - Tested code	- Started work on complaint handling - Implemented test cases - Tested code	- Add log off button for banned cooks - Fix banned cook UI - 4 JUnit Test Cases	- JUnit Test cases

4.0 Screenshots of App

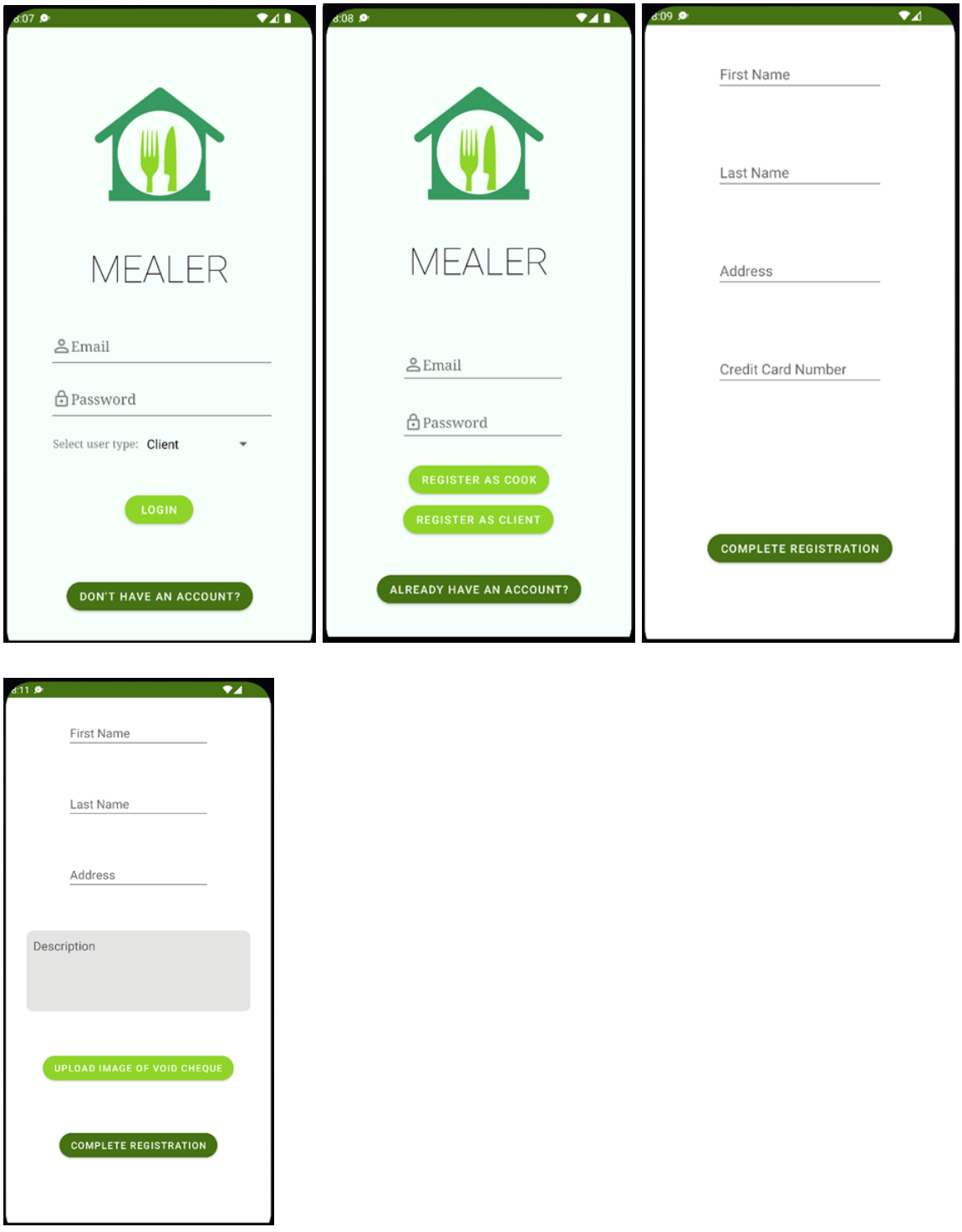

Figure 4.1. Cook and Client Login / Register Pages

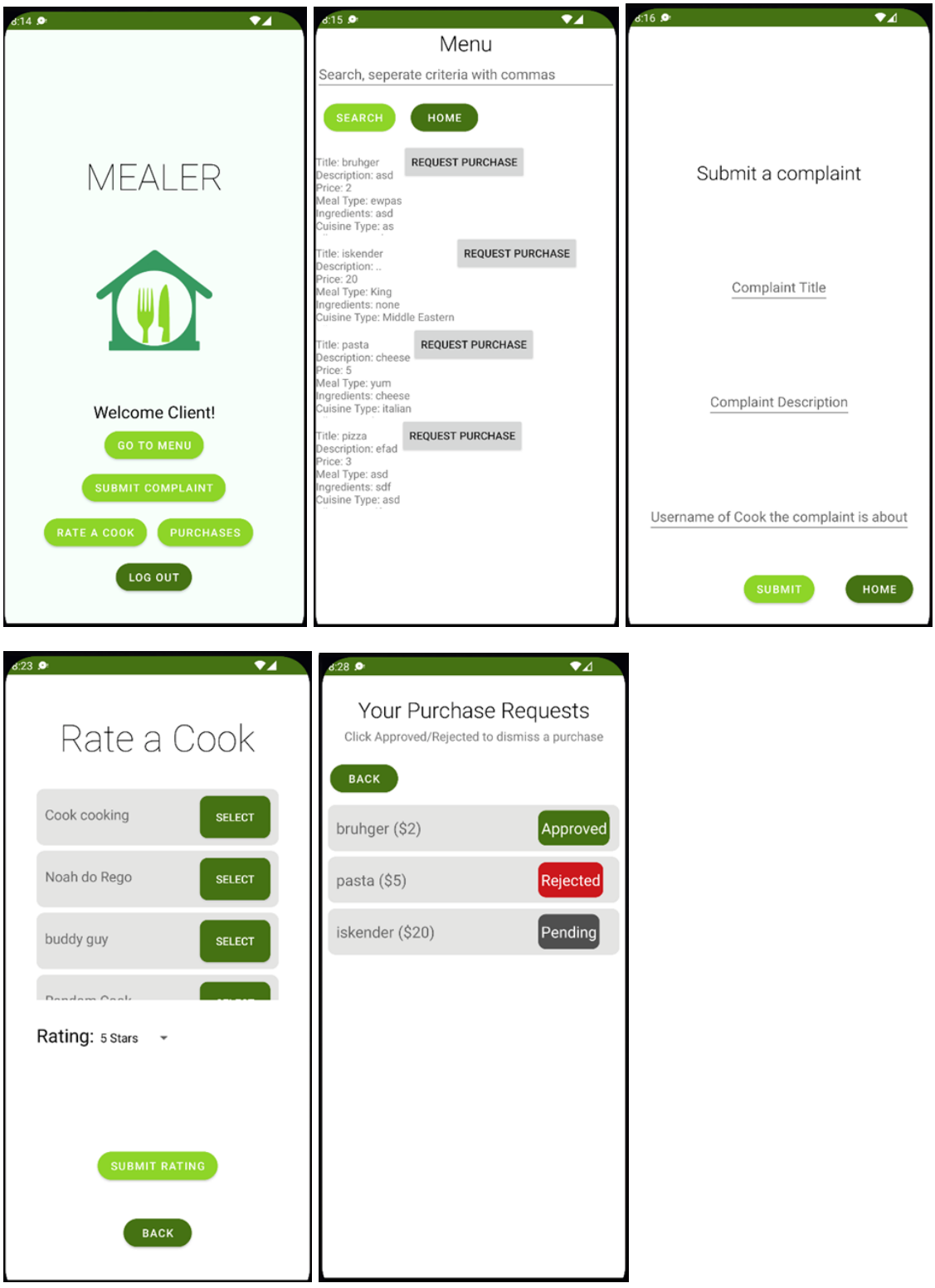

Figure 4.2. Client Pages

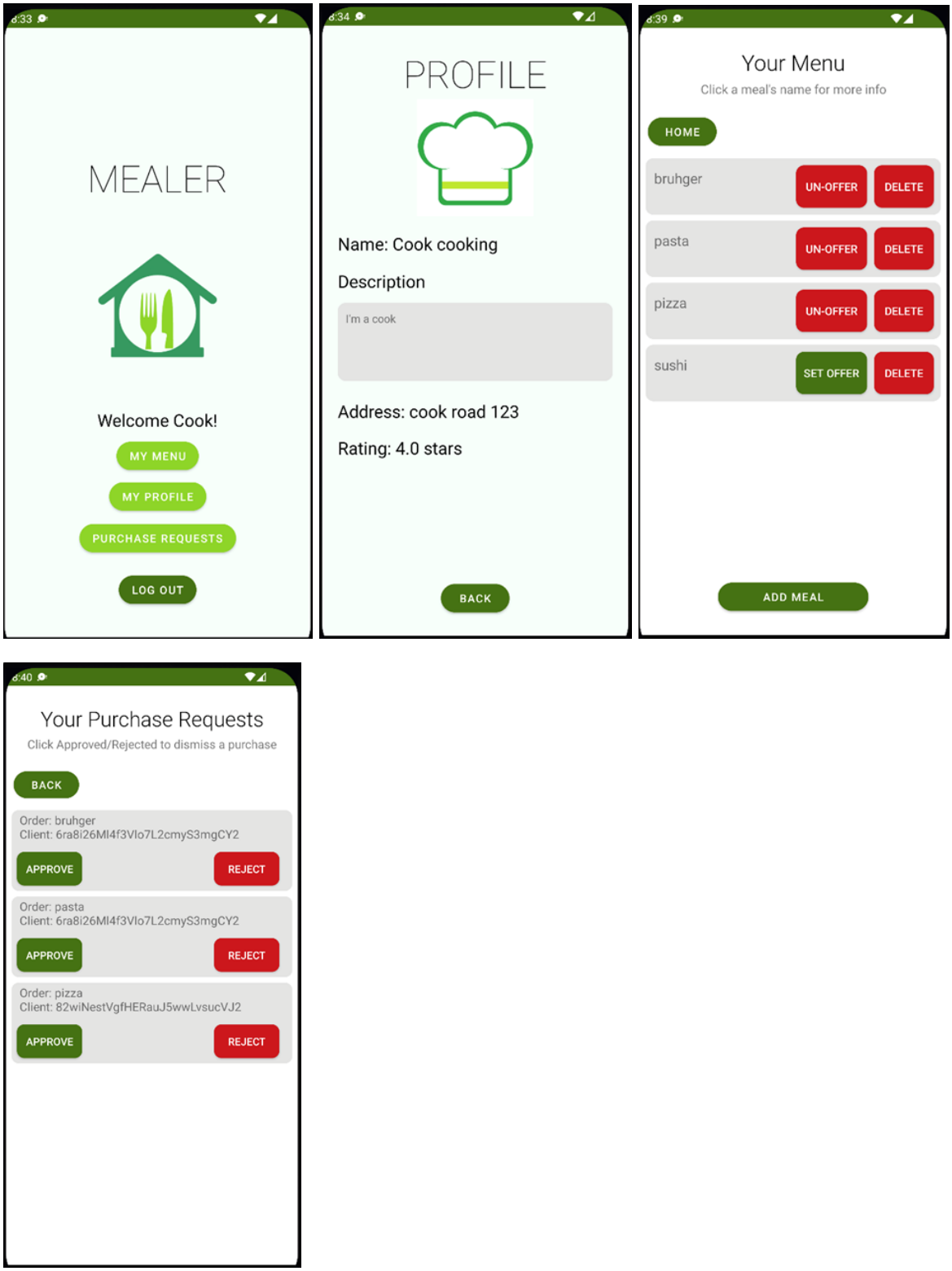

Figure 4.3. Cook Pages

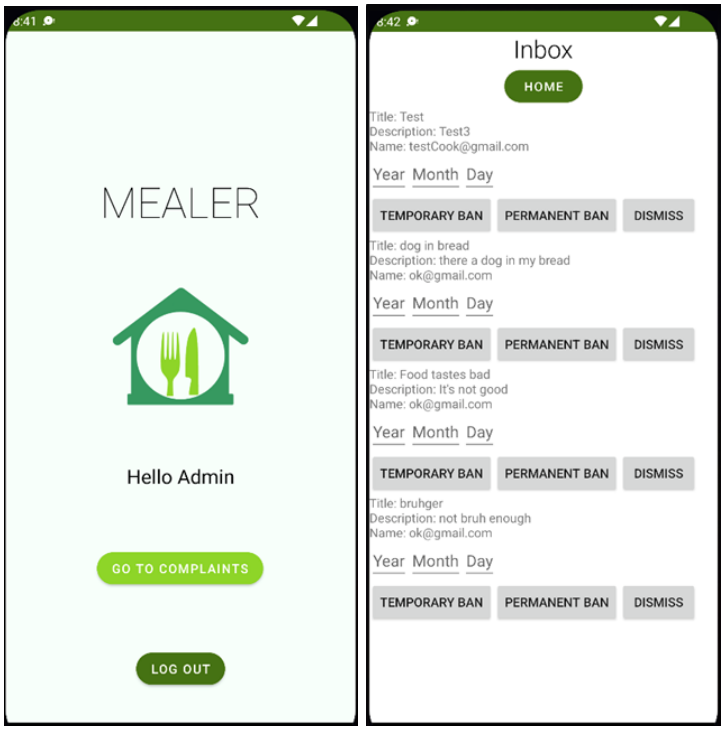

Figure 4.4. Admin Pages

5.0 Lessons Learned

The semester-long project gave our group the opportunity to learn how to collaborate with and update code written by other members using coding conventions taught in class, and we learned how to classify and apply patterns to UML diagrams.

Our team focused on producing well-written, documented code, which not only allowed us to apply coding conventions and programming practices learned in class, but also provided the contributors with a better understanding of the logic behind the code that was written. Time was the most notable factor in members not thoroughly documenting their code since there were deadlines to adhere to throughout the project's timeline. However, documentation ensures software structural quality, ameliorates the readability of code, and encourages maintenance on the entire system. Taking the time to apply these practices ultimately preserved time in later deliverables as team members revisited the code and did not need to depend on the member who developed the code to understand it. As the deliverables progressed, group members were able to update classes and activities, when necessary, through the use of detailed comments and naming conventions.

Furthermore, employing patterns for a solid code foundation facilitated the development process of our application since members did not need to reinvent patterns to solve a problem. Existing patterns provide the groundwork of the solution and encourage software reusability. In the earlier stages of development, our group created the Client and Cook account management component using an abstraction-occurrence pattern. In our UML diagram, there is a set of occurrences of account types (Admin, Cook, and Client), that share common information, which are the username and password fields. The group proposed to represent the sets of occurrences in the UML diagram using the abstraction-occurrence pattern. This pattern avoids duplicating data when representing the members of each set of occurrences, and the use of a known pattern is an effective method of solving a commonly occurring problem.

These programming practices will inevitably be applied in subsequent courses and roles in the workplace.