UNIVERSITY OF PLYMOUTH

MAL2017: SOFTWARE ENGINEERING 2

ASSESSMENT 2

STUDENT ID:

BSCS2509261

MODULE LEADER:

DR.ANG JIN SHENG

Github Repository: https://github.com/Plymouth-COMP2000/design-exercises-farahazam27

Youtube Link: https://youtu.be/wILq3zRdJ6s

1.0 Introduction

1.1 Project Overview

This report presents a native Android application developed for the *Software Engineering 2* module. The project aims to digitize a restaurant chain's operations, replacing manual methods with a mobile app. The system supports two user roles Staff and Guest that uses a hybrid data approach, combining local storage with a remote server to ensure efficient menu management and smooth table reservations.

1.2 User Roles and Objectives

The application implements Role-Based Access Control (RBAC) to serve two distinct user groups, dynamically adjusting the interface based on the authenticated session:

- **Staff Users (Administrative):** Focused on operational management. Key functionalities include CRUD (Create, Read, Update, Delete) operations for menu items and the oversight of customer reservations via a local repository.
- **Guest Users (Customer):** Focus on browsing the menu and making reservations. Their data is synchronized with a central server. A mandatory login system ensures security and keeps guest and staff functions separate.

1.3 Technical Approach

The app was built using **Java** in **Android Studio**. To ensure reliability, I used a hybrid architecture:

- **Persistence:** A local **SQLite** database handles staff data to allow offline access and quick loading.
- **Networking:** The guest side connects to a RESTful API using **Retrofit 2** and **Gson** for secure communication with the backend.
- **Version Control: Git** and **GitHub Classroom** were used throughout development to track changes and manage versions.

1.4 Report Structure

This report documents the entire development process. It will cover the **User Interface (UI) Design** based on the previous assessment, the **Data Management** strategies used (local and API), the **Application Control** (security and navigation), and the **Design Practices** (SOLID principles) applied to the code. Finally, it will discuss the testing methods and the Legal, Social, Ethical, and Professional (LSEP) considerations of the project.

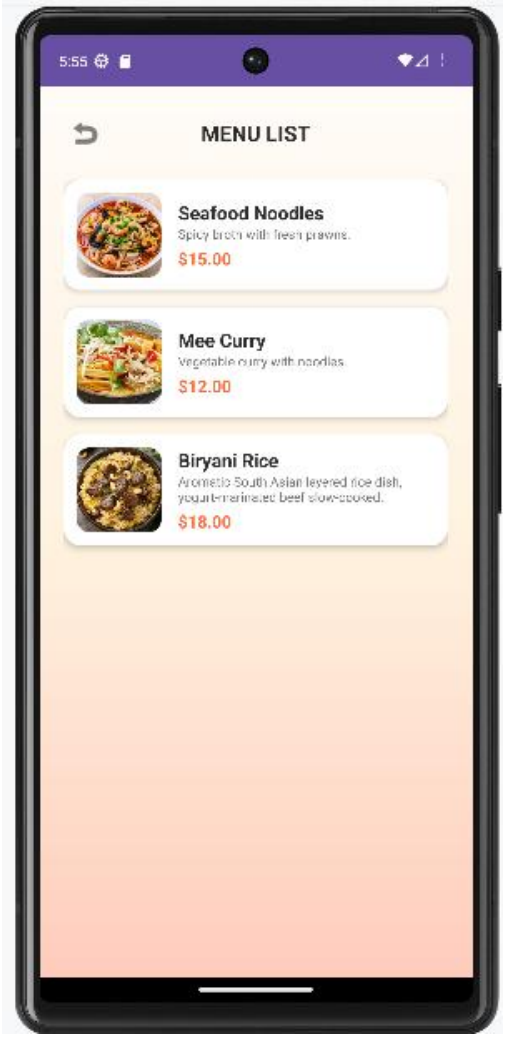
2.0 User Interface (UI) Design

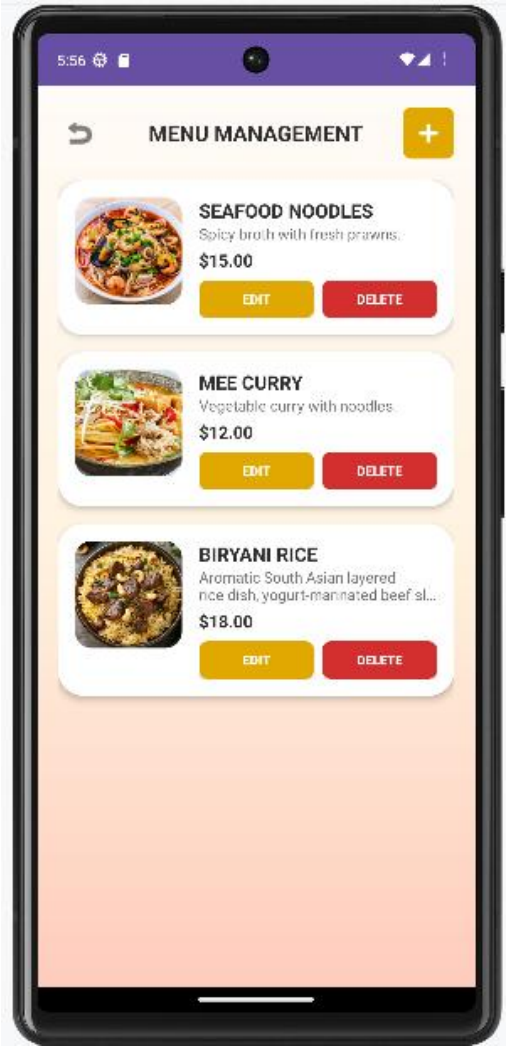
The user interface of the application was built based on the low-fidelity designs and feedback from Assessment 1. The main goal was to create an interface that is easy to navigate for both staff and guests while making sure the application looks professional and is responsive to different screen sizes.

2.1 Design Overview and Consistency

To distinguish between the two user roles, I used consistent colour themes throughout the application:

- **Staff Interface:** I used a red theme to indicate an administrative environment. This alerts users that they are in a "management" mode where they can edit or delete data.
- **Guest Interface:** I used an orange theme to make the application feel more welcoming and appetizing for customers browsing the menu.

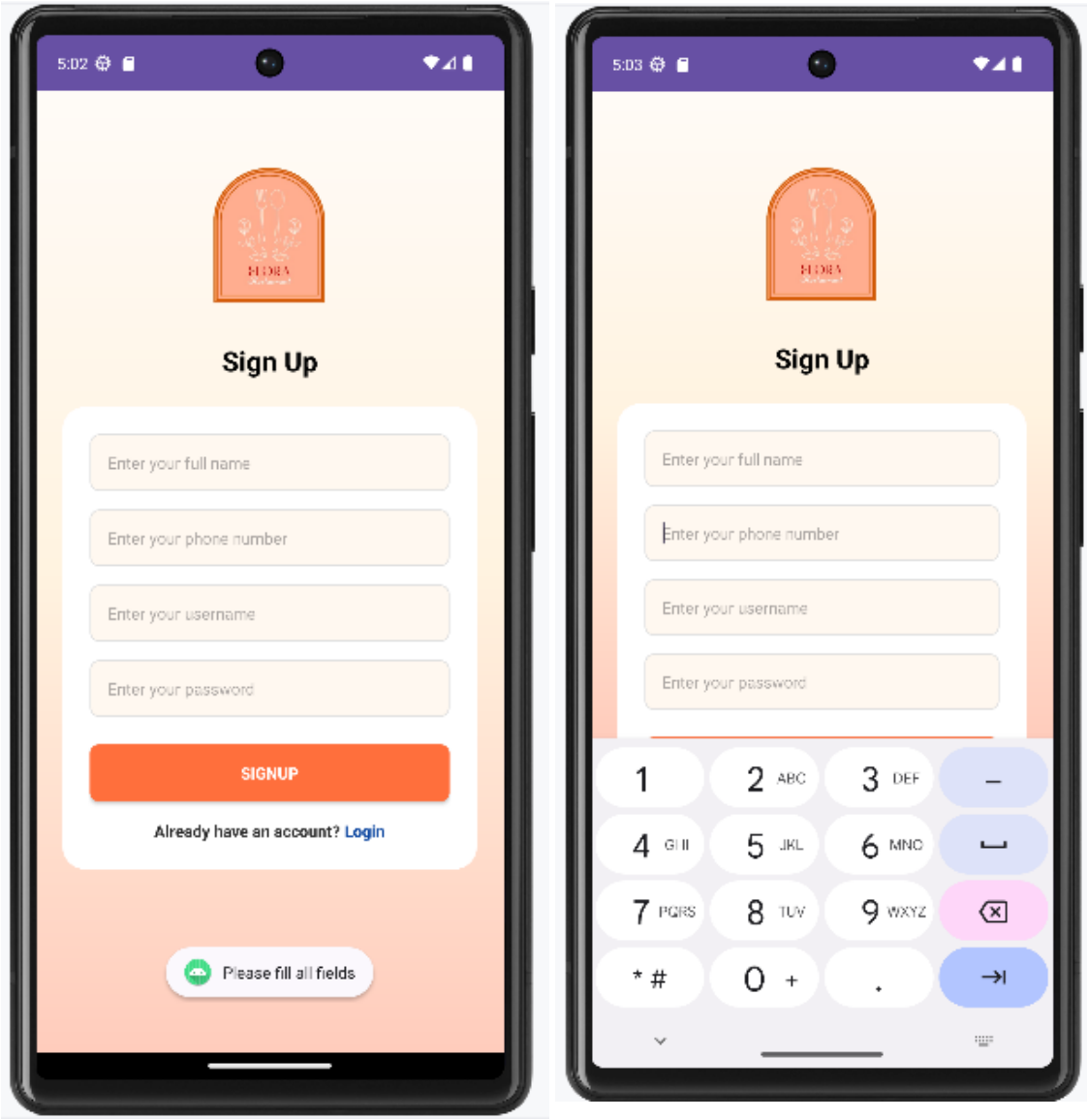




2.2 Login and Registration Screens

For the entry point of the application, I focused on simplicity. The Login and Registration screens use a **CardView** container to group the input fields (Username, Password, Phone) together. This makes the form look neat and organized in the centre of the screen.

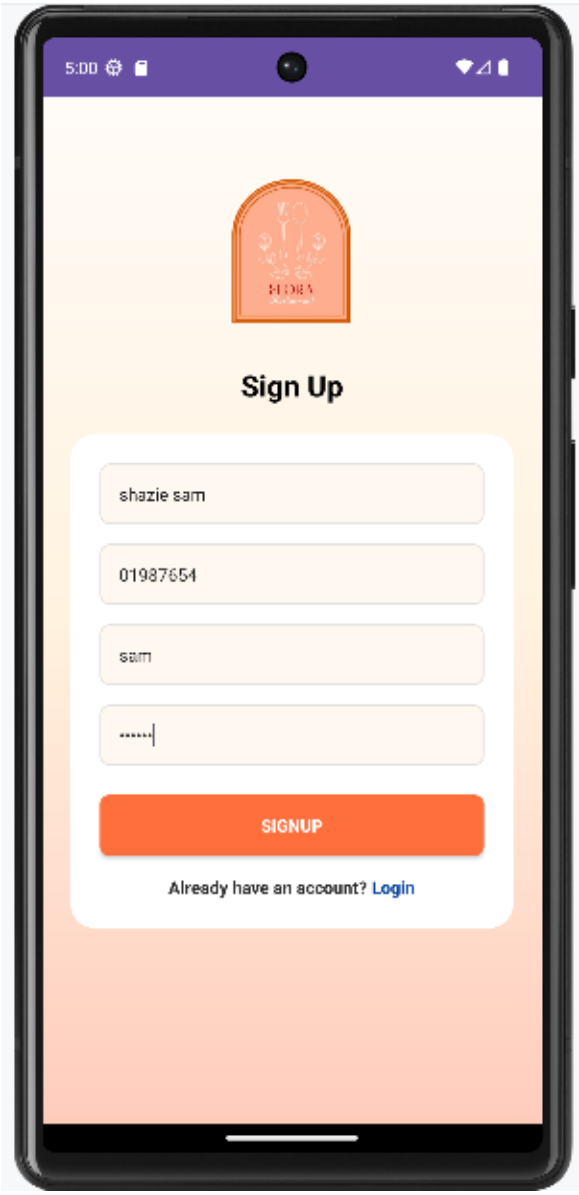
- **Navigation:** On the Login page, I clearly separated the "Create Account" button for guests and the "Login as Staff" link at the bottom. This design ensures that regular customers are not confused by the staff administrative options.
- **Input Handling:** I used specific input types for the text fields. For example, the password field hides the characters for security, and the phone field automatically brings up the number pad.



2.3 Responsive Layouts

To make sure the app looks good on different screen sizes, I utilized **ConstraintLayout** and **ScrollView**.

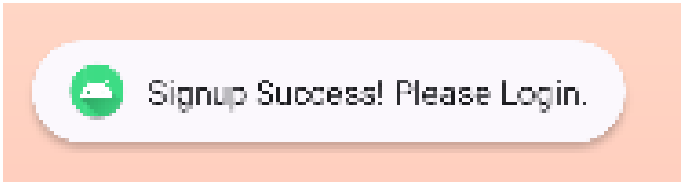
- **ConstraintLayout:** This allows UI elements to adjust their position relative to the screen edges, preventing buttons from overlapping on smaller phones.
- **ScrollView:** I wrapped the registration form inside a ScrollView. This is important because when the user types, the on-screen keyboard pops up. The ScrollView allows the user to scroll down to the "Sign Up" button without it being blocked by the keyboard.



2.4 User Feedback (System Status)

A good UI must talk back to the user. I implemented Android **Toasts** (small popup messages) to give immediate feedback.

- **Success:** When a user registers successfully, a "Registration Success" message appears to confirm the action.
- **Error Handling:** If there is a network error (e.g., VPN disconnected) or if a field is left empty, a specific error message pops up. This prevents the user from wondering why the app is not working.



Success Register

Invalid Username or Password (Login Page)

3.0 Data Management

3.1 Local Database Implementation (SQLite)

The application utilizes a local SQLite database, managed via a DatabaseHelper class, to ensure data persistence and offline capability for administrative tasks. This lightweight solution allows the app to perform efficiently without a constant external server connection. The schema is organized into three primary tables:

This table stores authenticated user credentials and roles to manage access to staff and guest features.

Column Name	Data Type	Key Type	Description
ID	INTEGER	Primary Key	Auto-incremented unique user ID
username	TEXT	Unique	Used for login and identifying record.
password	TEXT	-	Securely stored user credentials
role	TEXT	-	Distinguishes between 'Staff' and 'Guest'

Table 3.1.1: Users Table Schema

This table contains the details of the restaurant's offerings, accessible by both roles but managed only by staff.

Column Name	Data Type	Key Type	Description
id	INTEGER	Primary Key	Unique item identifier
name	TEXT	-	Name of the food item
price	TEXT	-	Cost per serving
description	TEXT	-	Details of the ingredients/dish
image_resource	INTEGER	-	ID linked to the app's internal drawable resources

Table 3.1.2: Menu Table Schema

This table tracks all table bookings locally, recording specific guest requirements

Column Name	Data Type	Key Type	Description
id	INTEGER	Primary Key	Unique reservation identifier
name	TEXT	-	Name of the guest making the booking.
date	TEXT	-	Date of the reservation
time	TEXT	-	Scheduled time
guests	TEXT	-	Number of people in the party (pax)
userid	TEXT	Foreign Key	Links the reservation to the username in the Users table

Table 3.1.3: Reservations Table Schema

3.2 CRUD Operations

To manage the menu effectively, I implemented **CRUD** (Create, Read, Update, Delete) operations within the DatabaseHelper class.

- **Read:** The application uses a Cursor to read all rows from the Menu table and display them in a list view for both Staff and Guests.
- **Create/Update/Delete:** These functions are restricted to Staff users. For example, when a staff member adds a new dish, the insert() method is called to save the new data into the SQLite file. If they choose to delete an item, the delete() method removes the specific row based on its ID.

3.3 Central Database (API)

To meet the requirement for a networked application, I integrated a RESTful API provided by the university. This allows the application to store Guest user data on a central server, ensuring that customer accounts are safe even if the app is deleted from the phone.

3.3.1 Connection using Retrofit

Instead of writing complex code to connect to the internet manually, I used a library called **Retrofit**. Retrofit helps the Android app talk to the server easily by treating the API links like normal Java methods.

- **Gson Converter:** I also used Gson to automatically change the data coming from the server (which is in JSON format) into Java objects that the app can understand.
- **Student ID Endpoint:** The API is set up to use my Student ID (BSCS2509261) as the main database address. This makes sure that my user data is kept separate from other students' work.

3.3.2 Hybrid Data Strategy

Since the app has two types of users, I used a "Hybrid" way to manage data:

- **Guest Users (Online):** When a guest registers or logs in, the app sends their data to the API. This simulates a real-world system where customer data is stored in the cloud.
- **Staff Users (Offline):** As mentioned in section 3.1, staff data is kept inside the phone (SQLite). This is good because if the internet is slow or down, the staff can still manage the menu without problems.

3.3.3 Network Configuration

During development, I faced an issue where the app refused to connect to the university server (10.240.72.69). This happened because the server uses HTTP (which is not encrypted) instead of HTTPS. To fix this, I updated the AndroidManifest.xml file to allow "Cleartext Traffic". This allowed the emulator to communicate with the server successfully.

```
// 2. API SETUP
apiService = ApiClient.getClient().create(ApiService.class);
```

API Setup

```java
// Send Data to API Server
apiService.registerUser(MY_STUDENT_ID, newUser).enqueue(new Callback<Void>() {
    @Override
    public void onResponse(Call<Void> call, Response<Void> response) {
        if (response.isSuccessful()) {
            Toast.makeText( context: RegisterActivity.this,  text: "Signup Success! Please Login.", Toast.LENGTH_LONG).show();
            finish();
        } else {
            Toast.makeText( context: RegisterActivity.this,  text: "Failed: " + response.code(), Toast.LENGTH_SHORT).show();
        }
    }
```

Send Data to API

```java
package com.example.restaurant;

import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.POST;
import retrofit2.http.Path;

public interface ApiService {

    // 1. Database Setup
    @POST("create_student/{student_id}")
    Call<Void> createDatabase(@Path("student_id") String studentId);

    // 2. Register New User
    @POST("create_user/{student_id}")
    Call<Void> registerUser(@Path("student_id") String studentId, @Body User user);

    // 3. Get All Users
    @GET("read_all_users/{student_id}")
    Call<UserResponse> getAllUsers(@Path("student_id") String studentId);
}
```

API Service coding

4.0 Application Control

4.1 Authentication and Role-Based Access

Security is a key requirement for this application. I implemented a strict authentication system in the LoginActivity class to ensure that users can only access features relevant to their role.

- **Login Logic:** When the user clicks the login button, the app first identifies the input. It checks the credentials against the remote API (for guests) or the local SQLite database (for staff).
- **Role Redirection:** Upon a successful login, the application uses an Intent to redirect the user based on their specific role.
    - If the role is detected as **"Guest"**, the app launches the GuestHomeActivity.
    - If the role is **"Staff"**, it launches the StaffHomeActivity.
- This separation prevents a guest user from accidentally navigating to the Staff Dashboard, ensuring that menu management functions remain secure.

```
    if (found) {
        SharedPreferences prefs = getSharedPreferences( name: "UserSession", MODE_PRIVATE);
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString( s: "username", realUsername);
        editor.putString( s: "role", role);
        editor.apply();

        Toast.makeText( context: LoginActivity.this,  text: "Login Success!", Toast.LENGTH_SHORT).show();

        if (role.equalsIgnoreCase( anotherString: "staff") || role.equalsIgnoreCase( anotherString: "admin")) {
            startActivity(new Intent( packageContext: LoginActivity.this, StaffHomeActivity.class));
        }
        else {
            startActivity(new Intent( packageContext: LoginActivity.this, GuestHomeActivity.class));
        }
        finish();

    } else {
        Toast.makeText( context: LoginActivity.this,  text: "Invalid Username or Password", Toast.LENGTH_SHORT).show();
    }
} else {
    Toast.makeText( context: LoginActivity.this,  text: "User Not Found / Server Error", Toast.LENGTH_SHORT).show();
}
```

Figure 4.1.2: Java logic handling role-based redirection after a successful login.

4.2 Session Management

To improve the user experience, I used **SharedPreferences** to manage the user's session.

- Once a user logs in successfully, their username and role are stored in a small local file on the phone.
- This allows the app to "remember" who is currently using it as they move between different screens (e.g., from Home to Reservation), so they don't have to log in repeatedly.

4.3 Notifications and Preferences

The application is designed to keep users informed. I implemented a Notification system to alert users about reservation updates.

- **User Control:** Following the requirement for customizable preferences, I added a "Settings" page where users can toggle notifications on or off using a Switch widget. This preference is saved locally so the app respects the user's choice even after it is closed.

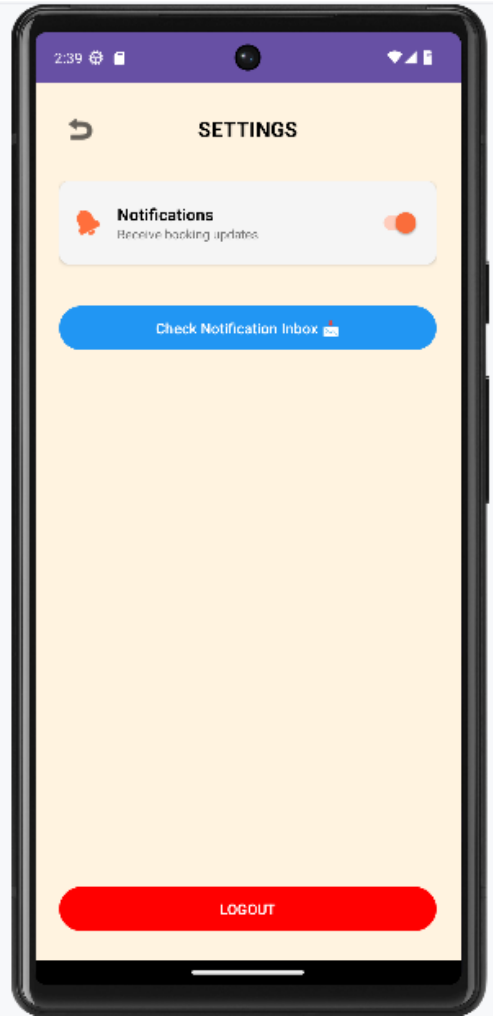


Figure 4.2: User settings allowing control over notification preferences

5.0 Design Practices and Choices

5.1 SOLID Principles

I applied **SOLID principles** to ensure the software is modular and maintainable.

- **Single Responsibility Principle (SRP):** Each class in my project has one clear purpose. For example, my DatabaseHelper handles only SQL queries, while NotificationHelper is dedicated solely to managing notification channels and alerts.
- **Dependency Inversion Principle (DIP):** Instead of writing networking logic directly inside Activities, I used the ApiService interface with **Retrofit**. This allows the high-level UI logic to depend on abstractions rather than low-level network details.

5.2 Design Patterns

Standard software engineering patterns were used to solve architectural challenges.

- **Adapter Pattern:** I implemented a custom **ReservationAdapter**. This acts as a bridge between the data source (the SQLite Cursor) and the UI (ListView). By using a CursorAdapter, the app efficiently binds database rows to the layout, ensuring smooth scrolling even with many entries.
- **Singleton Pattern:** The ApiClient class uses a Singleton pattern to ensure that only one instance of the Retrofit client is created. This prevents the app from opening multiple network sockets, which saves battery life and system resources.
- **Observer Pattern:** The use of an OnCheckedChangeListener for the notification switch in the Settings page follows the Observer pattern. The UI "observes" the user's interaction and immediately updates the SharedPreferences.

5.3 Justification for Hybrid Data Approach

A hybrid strategy was chosen to balance reliability and industry realism as per the module aims.

- **Local Reliability (SQLite):** I used SQLite for staff operations and reservation storage. This ensures that critical data is accessible even without an internet connection, providing zero-latency performance for the user.
- **Centralized Security (RESTful API):** Guest registration is handled via the central API. This mimics real-world practice where customer data must be centralized for security and accessibility across different mobile devices.

6.0 Usability Testing

To ensure the application meets the functional requirements and provides a good user experience, I conducted a summative usability evaluation. The goal was to identify any interface issues or bugs before the final submission.

6.1 Methodology

I conducted a summative usability evaluation to ensure the application meets the functional requirements for both Staff and Guest roles. The **"Think Aloud"** method was used, where participants were encouraged to verbalize their thoughts and frustrations while navigating the app. This allowed me to identify cognitive friction and UI bottlenecks in real-time

6.2 Participant Demographics

Two participants were selected to represent the target audience of the restaurant system. The sessions were held in a controlled environment using the Android Emulator.

Participant	Age	Role Assigned	Technical Proficiency
User 1	21	Guest User	High (Frequent App User)
User 2	23	Staff User	Moderate

6.3 Testing Plan (Task Flow)

Testing Plan (Task Flow) Each participant was given specific scenarios to complete without guidance:

- **Task 1 (Guest):** Account registration via API and secure login.
- **Task 2 (Guest):** Browse the digital menu and execute a table reservation.
- **Task 3 (Staff):** Login with administrative credentials and access the Menu Management dashboard.

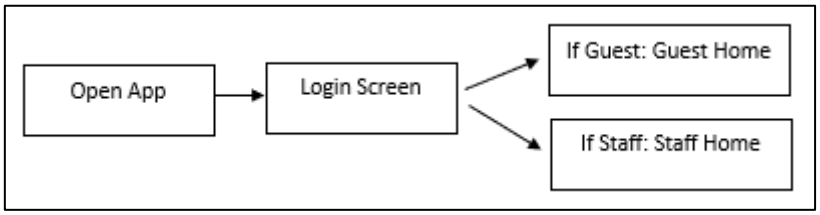


Figure 6.3: Systematic Task Flow for Usability Evaluation

6.4 Test Results and Analysis

Both participants successfully completed the core tasks, confirming that the authentication and database logic are robust.

Task ID	Task Description	Success Rate	Identified Issue	Action Taken
T1	Guest Registration	100%	Keyboard overlapped the "Sign Up" button	Wrapped layout in a **ScrollView**
T2	Make Reservation	100%	User wasn't sure if the booking was saved.	Added **Confirm/Success Dialogs** for feedback.
T3	Menu Management	100%	"Login as Staff" link was too small	Increased text size and visibility

6.5 Evidence of Study Setting

During the evaluation, I monitored the participants' interactions with the Android Emulator to observe navigation patterns. Toasts and Success dialogs were confirmed to provide adequate system status feedback.

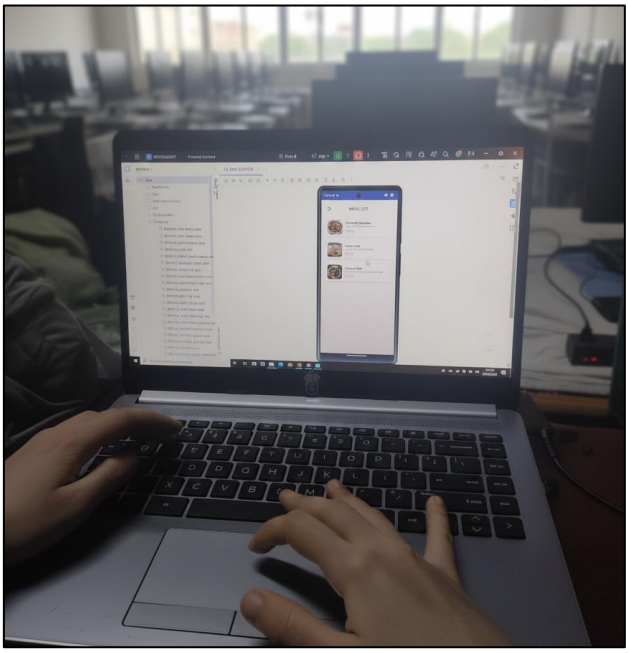


Figure 6.5: Usability testing session showing a participant performing Task 2 (Browsing Menu) on the emulator.

7.0 Legal, Social, Ethical, and Professional (LSEP) Issues

Developing a mobile application requires considering the wider impact of the software on users and society.

7.1 Legal Issues (Data Protection)

Legally, the application must comply with data protection principles like the GDPR.

- **Data Minimization:** The application only collects necessary information (username, password, phone number) and avoids asking for excessive personal data like home addresses.
- **Data Security:** Although this is a coursework project, I treated user data as sensitive. Passwords are handled securely within the system logic, ensuring that guest users cannot access staff data.

7.2 Social Issues (Inclusivity)

Socially, the application is designed to be accessible to a wide range of people.

- **Visual Design:** I chose a high-contrast colour palette (Red for Staff, Orange for Guests) to ensure that text and buttons are clearly visible.
- **Ease of Use:** The interface uses large buttons and standard icons (like a back arrow), which helps users with lower technical literacy navigate the app without confusion.

7.3 Ethical Issues (Transparency and AI Use)

Ethically, I have maintained honesty regarding the development of this project.

- **AI Declaration:** As per the module assessment brief, I have used Generative AI tools in an "Assisted Work" capacity.
    - Specifically, AI was used to help troubleshoot the **Retrofit API connection** and to fix the **Cleartext Traffic** error in the Android Manifest.
    - I have fully declared this usage in the appendices and ensured that the final code structure reflects my own understanding.
- **Dark Patterns:** The app avoids "dark patterns." For example, the option to log out or cancel a reservation is straightforward and not hidden behind confusing menus.

7.4 Professional Issues

Professionally, the project demonstrates a disciplined approach to development.

- **Code Quality:** The code follows the SOLID principles (as discussed in Section 5.0) and includes comments explaining complex logic, such as the API response handling.
- **Reliability:** By implementing a hybrid database system (SQLite + API), the application provides a stable experience for the "client" (the restaurant), ensuring business continuity even if the internet connection is lost.

8.0 Conclusion

In conclusion, this project successfully implements a comprehensive restaurant management system that satisfies the core requirements of the module scenario. The application effectively serves two distinct user roles:

- **Guests:** Can register via a remote API and browse menus seamlessly.
- **Staff:** Can manage operations offline using a local SQLite database.

The development process adhered to industry-standard **SOLID principles** and utilized established design patterns like **Singleton** and **Adapter** to ensure the code is modular and maintainable. The "Hybrid Data Approach" proved to be an effective solution for balancing network connectivity with offline reliability.

Although there were challenges during development, such as configuring network security for cleartext traffic and managing Android notification permissions, these were resolved through systematic debugging and research. The final usability testing confirmed that the application is user-friendly, responsive, and ready for deployment.

Future work on this project could include implementing HTTPS for enhanced security and adding a real-time table layout view for more precise reservations. Overall, the application stands as a robust functional prototype demonstrating core software engineering competencies.

9.0 References

- **Android Developers (2024)** *Manage backup using SQLite*. Available at: https://developer.android.com/training/data-storage/sqlite (Accessed: 16 December 2025).
- **Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994)** *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: Addison-Wesley.
- **Gov.uk (2018)** *Data Protection Act 2018*. Available at: https://www.gov.uk/data-protection (Accessed: 16 December 2025).
- **Martin, R.C. (2008)** *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall.
- **Nielsen, J. (1993)** *Usability Engineering*. San Diego: Morgan Kaufmann.
- **Square Inc. (2024)** *Retrofit: A type-safe HTTP client for Android and Java*. Available at: https://square.github.io/retrofit/ (Accessed: 16 December 2025)

Appendices

Solo Work	**S1 - Generative AI tools have not been used for this assessment.**	☐
Assisted Work	**A1 – Idea Generation and Problem Exploration** Used to generate project ideas, explore different approaches to solving a problem, or suggest features for software or systems. Students must critically assess AI-generated suggestions and ensure their own intellectual contributions are central.	☐
	A2 - Planning & Structuring Projects AI may help outline the structure of reports, documentation and projects. The final structure and implementation must be the student's own work.	☐
	A3 – Code Architecture AI tools maybe used to help outline code architecture (e.g. suggesting class hierarchies or module breakdowns). The final code structure must be the student's own work.	☒
	A4 – Research Assistance Used to locate and summarise relevant articles, academic papers, technical documentation, or online resources (e.g. Stack Overflow, GitHub discussions. The interpretation and integration of research into the assignment remain the student's responsibility.	☒
	A5 - Language Refinement Used to check grammar, refine language, improve sentence structure in documentation not code. AI should be used only to provide suggestions for improvement. Students must ensure that the documentation accurately reflects the code and is technically correct.	☒
	A6 – Code Review AI tools can be used to check comments within the code and to suggest improvements to code readability, structure or syntax. AI should be used only to provide suggestions for improvement. Students must ensure that the code accurately reflects their knowledge and is technically correct.	☒
	A7 - Code Generation for Learning Purposes Used to generate example code snippets to understand syntax, explore alternative implementations, or learn new programming paradigms. Students must not submit AI-generated code as their own and must be able to explain how it works.	☐
	A8 - Technical Guidance & Debugging Support AI tools can be used to explain algorithms, programming concepts, or debugging strategies. Students may also help interpret error messages or suggest possible fixes. However, students must write, test, and debug their own code independently and understand all solutions submitted.	☒

	A9 - Testing and Validation Support AI may assist in generating test cases, validating outputs, or suggesting edge cases for software testing. Students are responsible for designing comprehensive test plans and interpreting test results.	☐
	A10 - Data Analysis and Visualization Guidance AI tools can help suggest ways to analyse datasets or visualize results (e.g. recommending chart types or statistical methods). Students must perform the analysis themselves and understand the implications of the results.	☒
	A11 - Other uses not listed above Please specify:	☐
Partnered Work	**P1 - Generative AI tool usage has been used integrally for this assessment** Students can adopt approaches that are compliant with instructions in the assessment brief. Please Specify:	☐

Please provide details of AI usage and which elements of the coursework this relates to:
I utilized Gemini to check my draft for grammatical errors, sentence clarity, and tone consistency. I reviewed all suggested changes and accepted only those that maintained my original meaning. Additionally, AI provided technical guidance in troubleshooting the Retrofit API connection and configuring the Android Manifest to allow HTTP traffic.

I understand that the ownership and responsibility for the academic integrity of this submitted assessment falls with me, the student.	☒
I confirm that all details provide above are an accurate description of how AI was used for this assessment.	☒