UNIVERSITY OF PLYMOUTH

In collaboration with Peninsula College

School of Technology & Engineering

BSc (Hons) Computer Science (Cyber Security)

(N/0613/6/0002)(04/2027)(MQA/PA15604)

MAL2017 – Software Engineering 2

(CW2)

Author:

BSCS2509254

Module Leader:

Dr. Ang Jin Sheng

Dec 19, 2025

Table of Contents

Code Repository

Github	https://github.com/Plymouth-COMP2000/design-exercises-ooiweichyeh
Google Form	https://forms.gle/5kfftohwoPvE1gtL8
Presentation	https://youtu.be/b-jYnXvSO4o

Word Count : **2350 words** (excluding Heading, Caption, Reference and Appendix)

1.0 Introduction

This coursework 2 (CW2) focuses on the design and implementation of a Restaurant Management Mobile Application developed for two distinct user groups: Staff and Guests(User). Building on the foundations established in CW1, the report wrote how the user interface (UI), structure and core controls were implemented in Android Studio following established Human–Computer Interaction (HCI) and Software Engineering principles.

Thus, the application aims to support daily operational activities such as menu management and reservation handling while providing guests with a convenient platform for browsing menus and making table bookings. A structured usability testing process involving real users was conducted to validate the design decisions and refine the interface.

2.0 UI Design

2.1 Design Philosophy

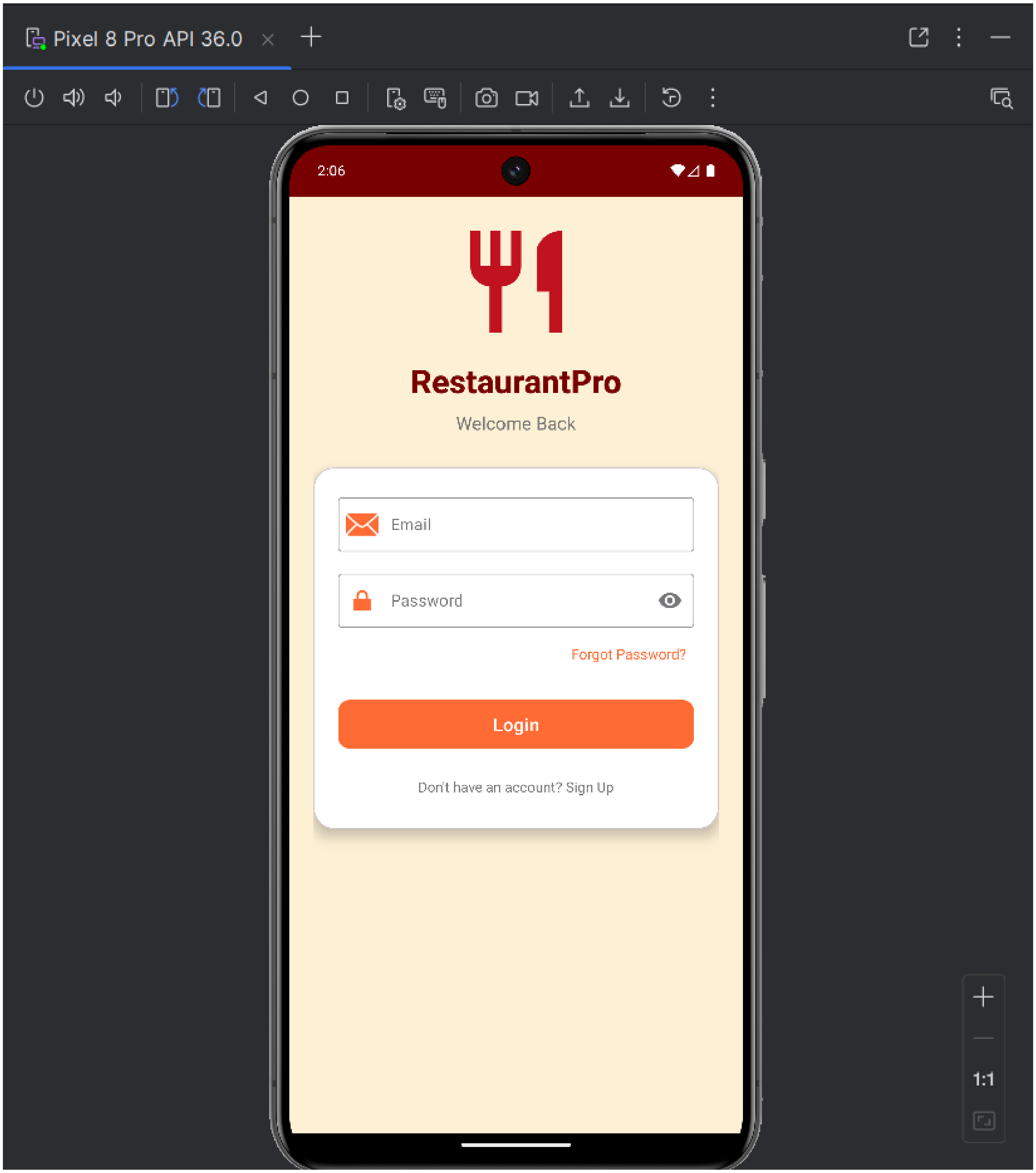
The interface design draws heavily from Material Design guidelines (Google, 2021), though not slavishly. Material Design provides a solid foundation, but I adapted elements to fit the restaurant context like warm colors that feel welcoming rather than the stark blues often seen in productivity apps.The primary palette uses:

i. Primary Orange (#FF6B35) – draws attention without being aggressive
ii. Primary Red (#C1121F) – sparks appetite (there's actually research suggesting red increases hunger)
iii. Primary Brown (#780000) – adds warmth and sophistication
iv. Background Cream (#FDF0D5) – reduces eye strain during extended use

This wasn't arbitrary. Norman (2013) discusses affordances extensively and color is one way to signal what elements do.

2.2 Navigation Architecture

Bottom navigation handles the primary user flows. This choice stems from thumb-zone research where users can reach bottom elements more easily on larger phones (Preece et al., 2015). The implementation uses Material's BottomNavigationView which handles state management and visual feedback automatically.



Pixel 8 Pro API 36.0

2:06

RestaurantPro

Welcome Back

Email

Password

Forgot Password?

Login

Don't have an account? Sign Up

Figure 1 : Login Page (applicable to both users)

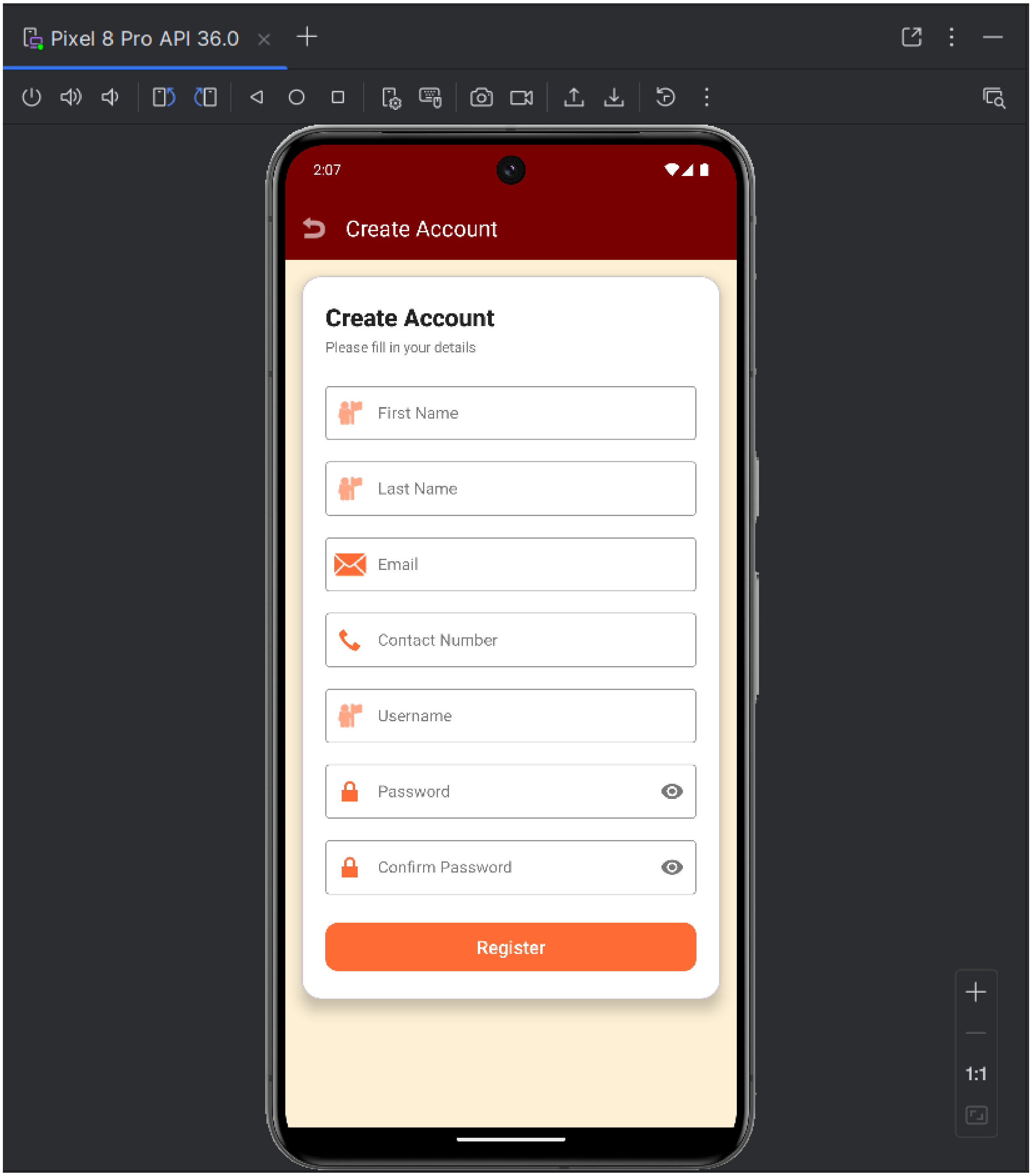
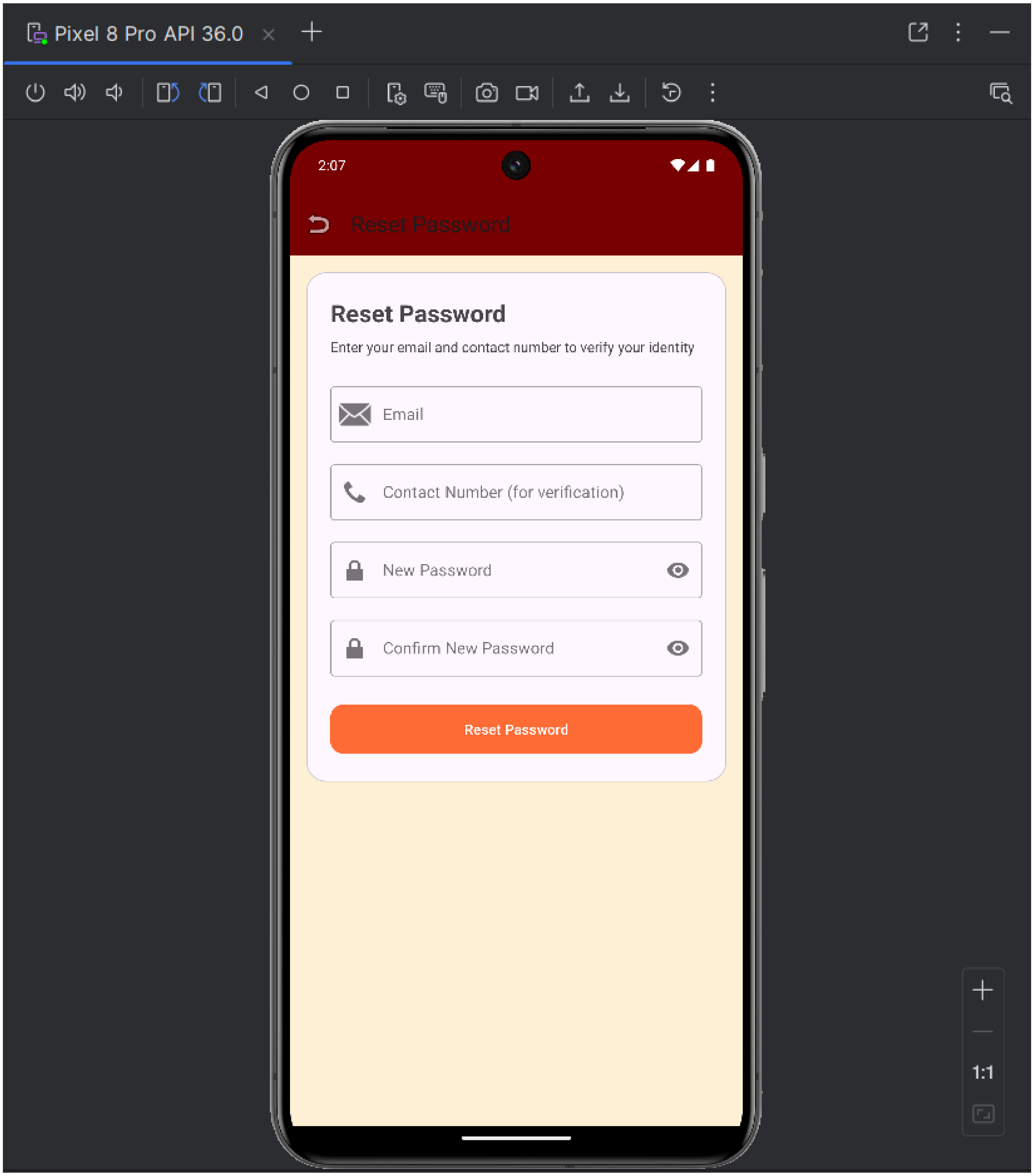


Figure 2 : Signup page (applicable to both users)



Pixel 8 Pro API 36.0

2:07

Reset Password

Reset Password

Enter your email and contact number to verify your identity

Email

Contact Number (for verification)

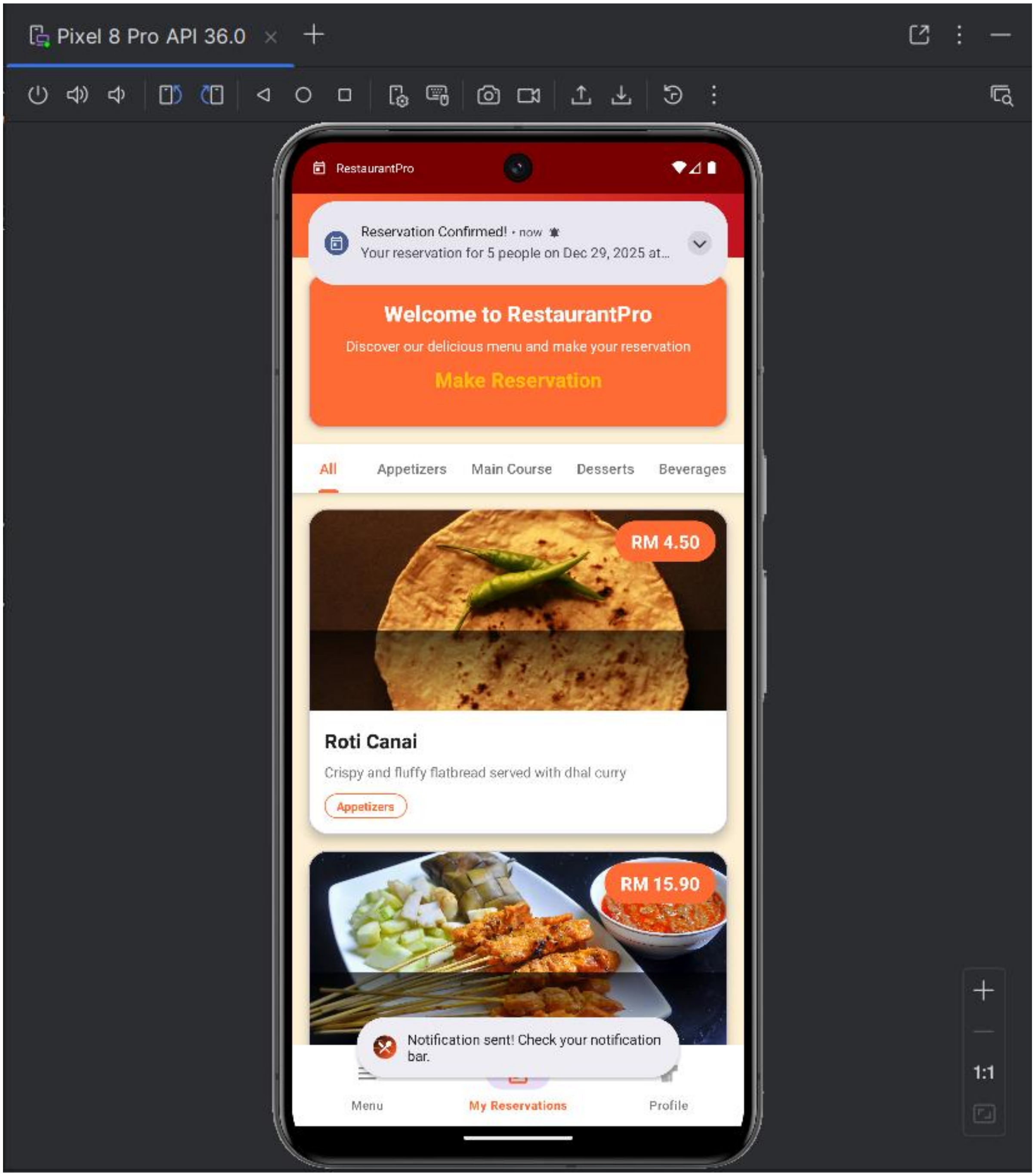
New Password

Confirm New Password

Figure 3 : Forget Password Page (applicable to both users)

For **Guests,** the navigation offers:

- Menu – browse dishes by category
- My Reservations – view booking history
- Profile – manage account details



Reservation Confirmed! • now

Your reservation for 5 people on Dec 29, 2025 at...

Welcome to RestaurantPro

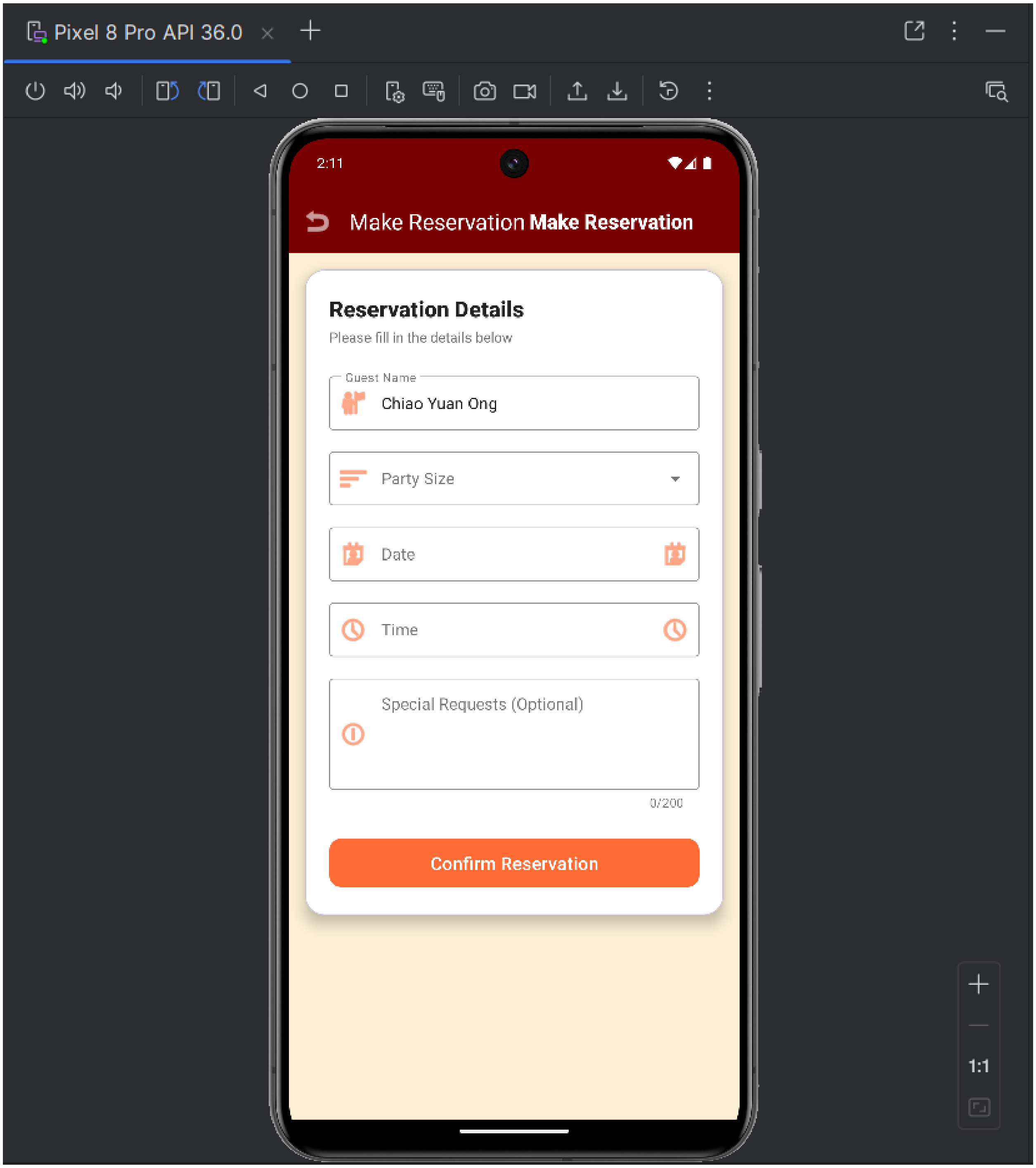
Discover our delicious menu and make your reservation

Roti Canai

Crispy and fluffy flatbread served with dhal curry

Notification sent! Check your notification bar.

Figure 4 : Guest Dashboard



Make Reservation **Make Reservation**

Reservation Details

Please fill in the details below

Guest Name: Chiao Yuan Ong

Party Size

Date

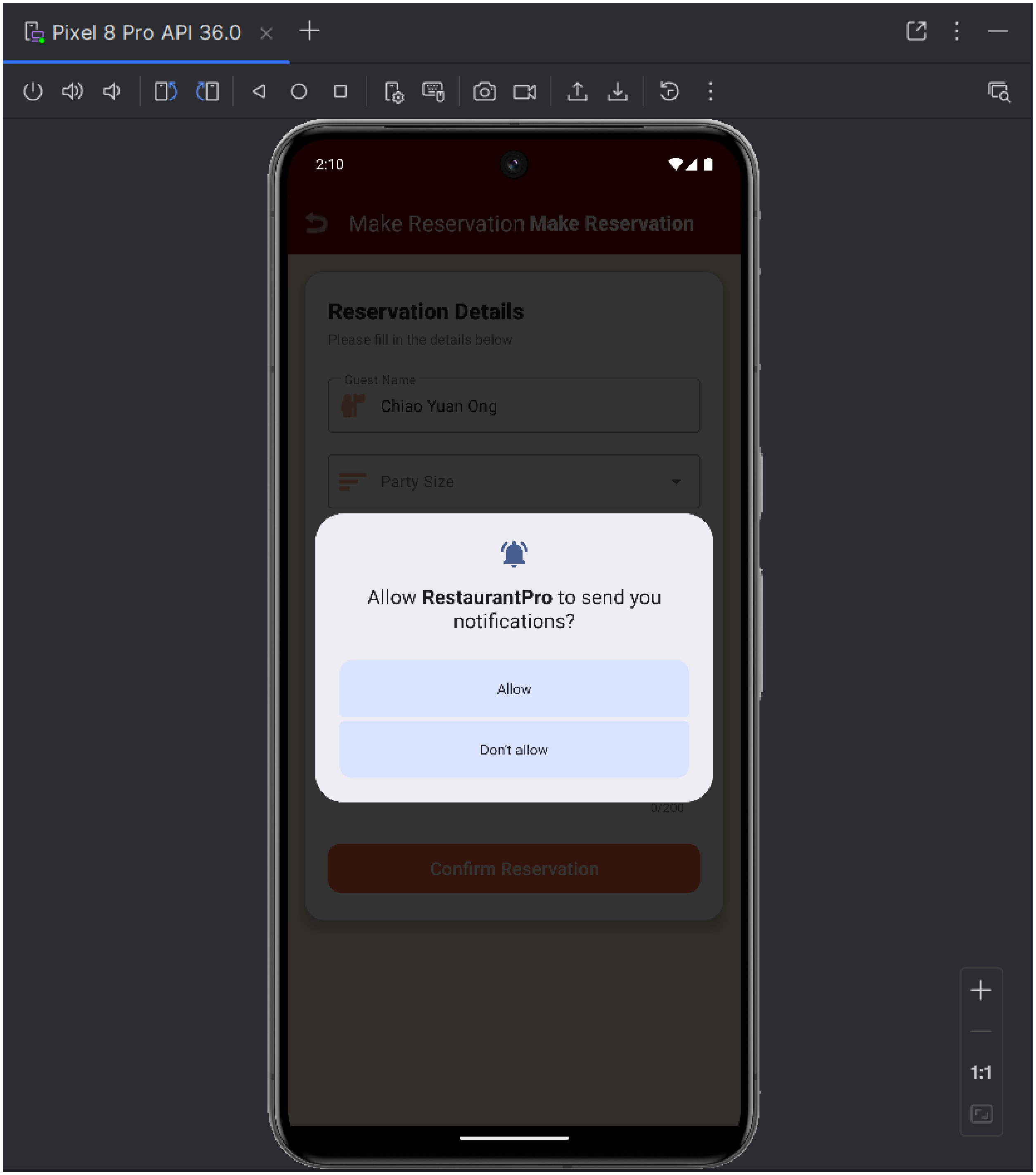
Time

Special Requests (Optional)

0/200

Confirm Reservation

Figure 5 : Guest Reservation Page



Make Reservation **Make Reservation**

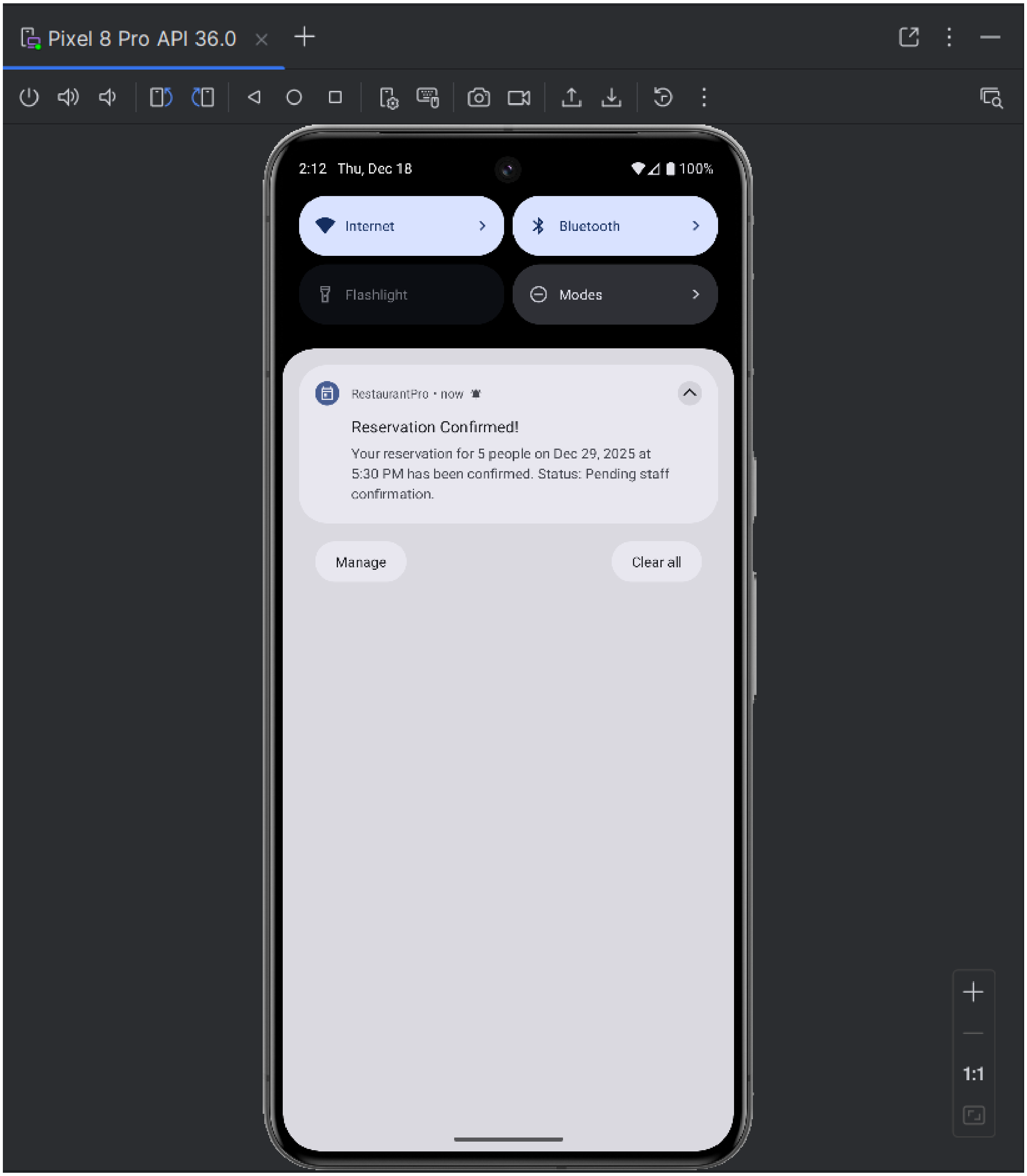
Reservation Details

Please fill in the details below

Guest Name: Chiao Yuan Ong

Allow **RestaurantPro** to send you notifications?

Figure 6 : Notification popup for Guest to confirm

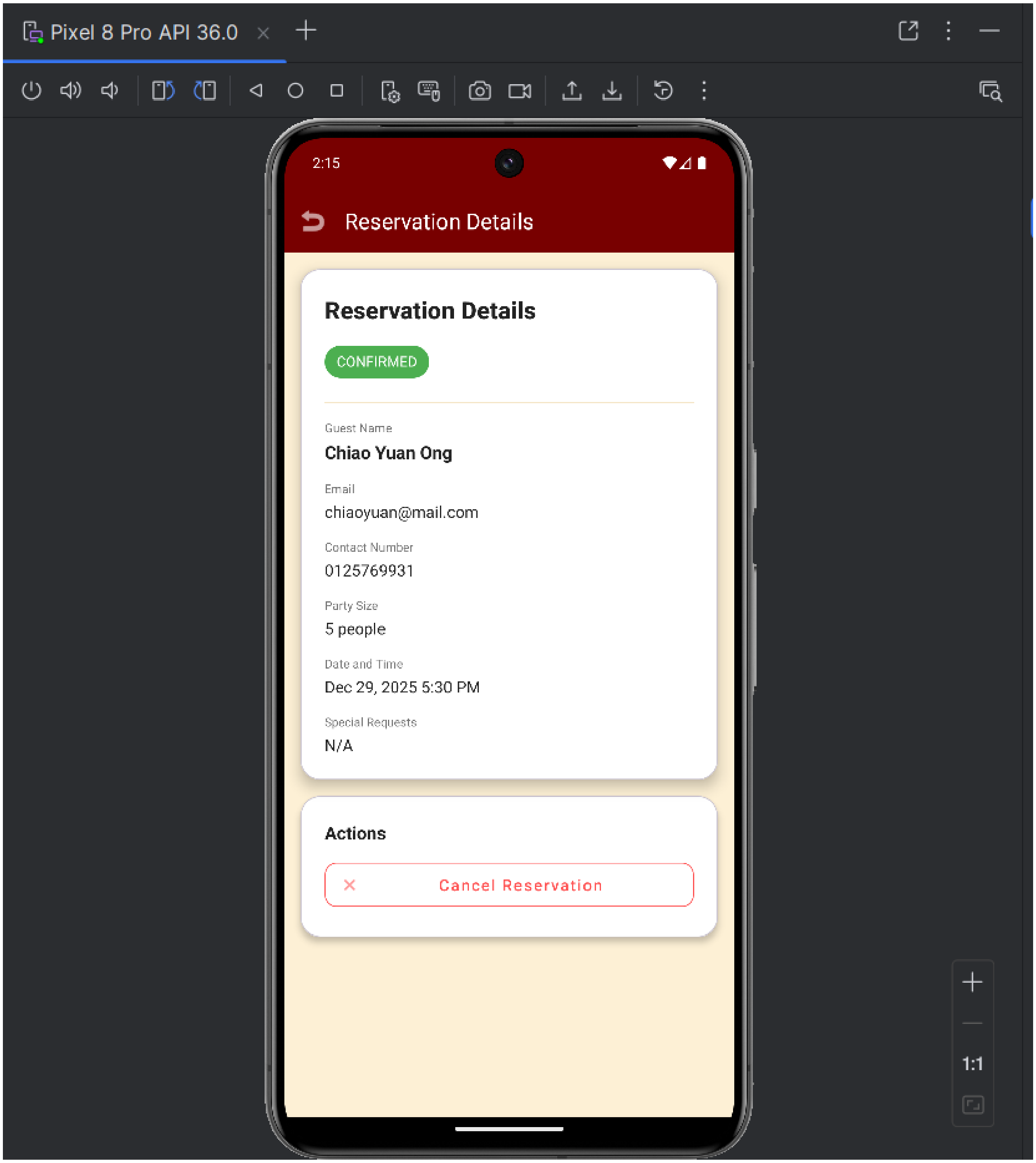


RestaurantPro • now

Reservation Confirmed!

Your reservation for 5 people on Dec 29, 2025 at 5:30 PM has been confirmed. Status: Pending staff confirmation.

Figure 7 : Notification shown in Phone



Reservation Details

Reservation Details

CONFIRMED

Guest Name
Chiao Yuan Ong

Email
chiaoyuan@mail.com

Contact Number
0125769931

Party Size
5 people

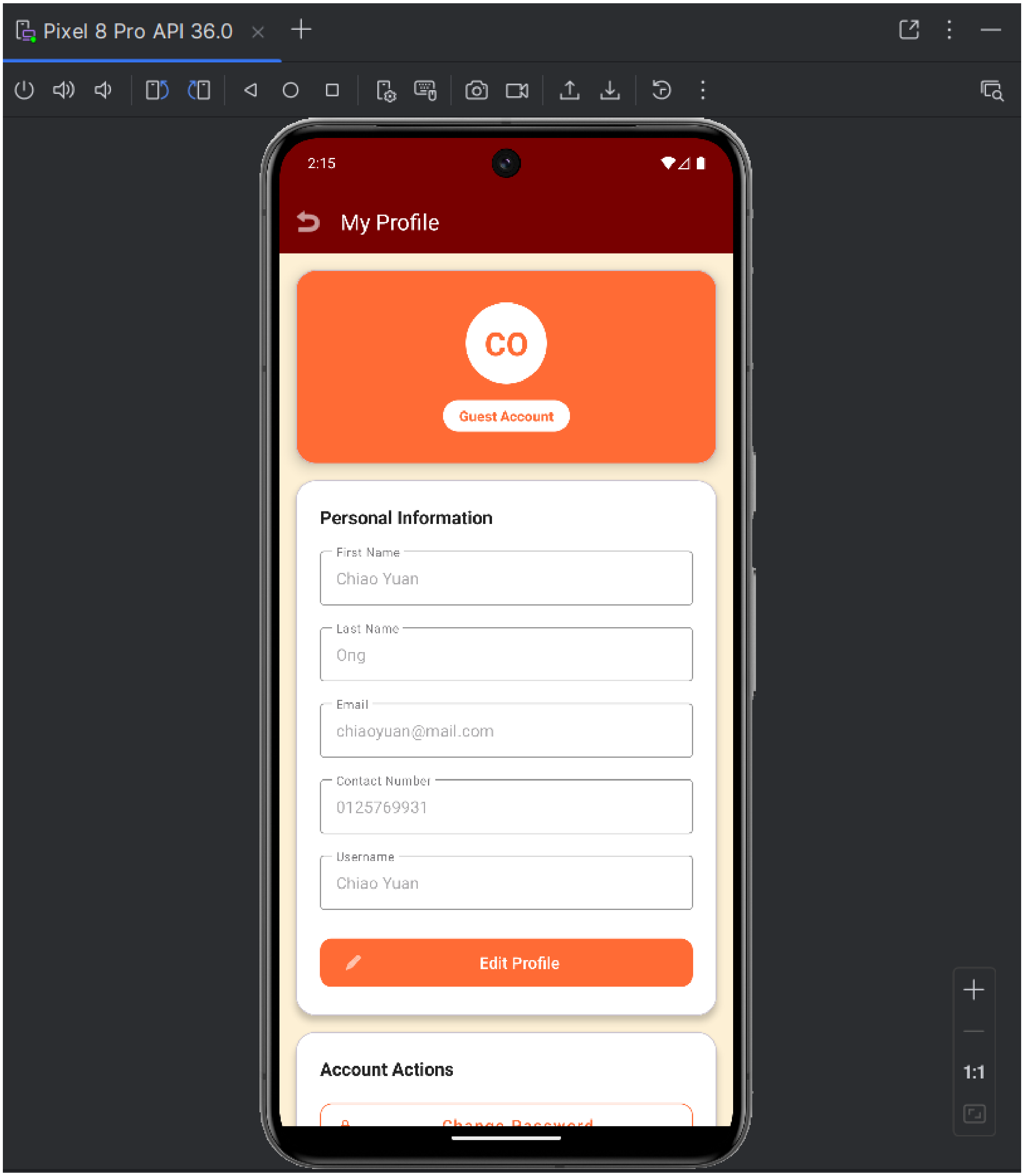
Date and Time
Dec 29, 2025 5:30 PM

Special Requests
N/A

Actions

✕ Cancel Reservation

Figure 8 : Reservation Page made by Guest



First Name: Chiao Yuan

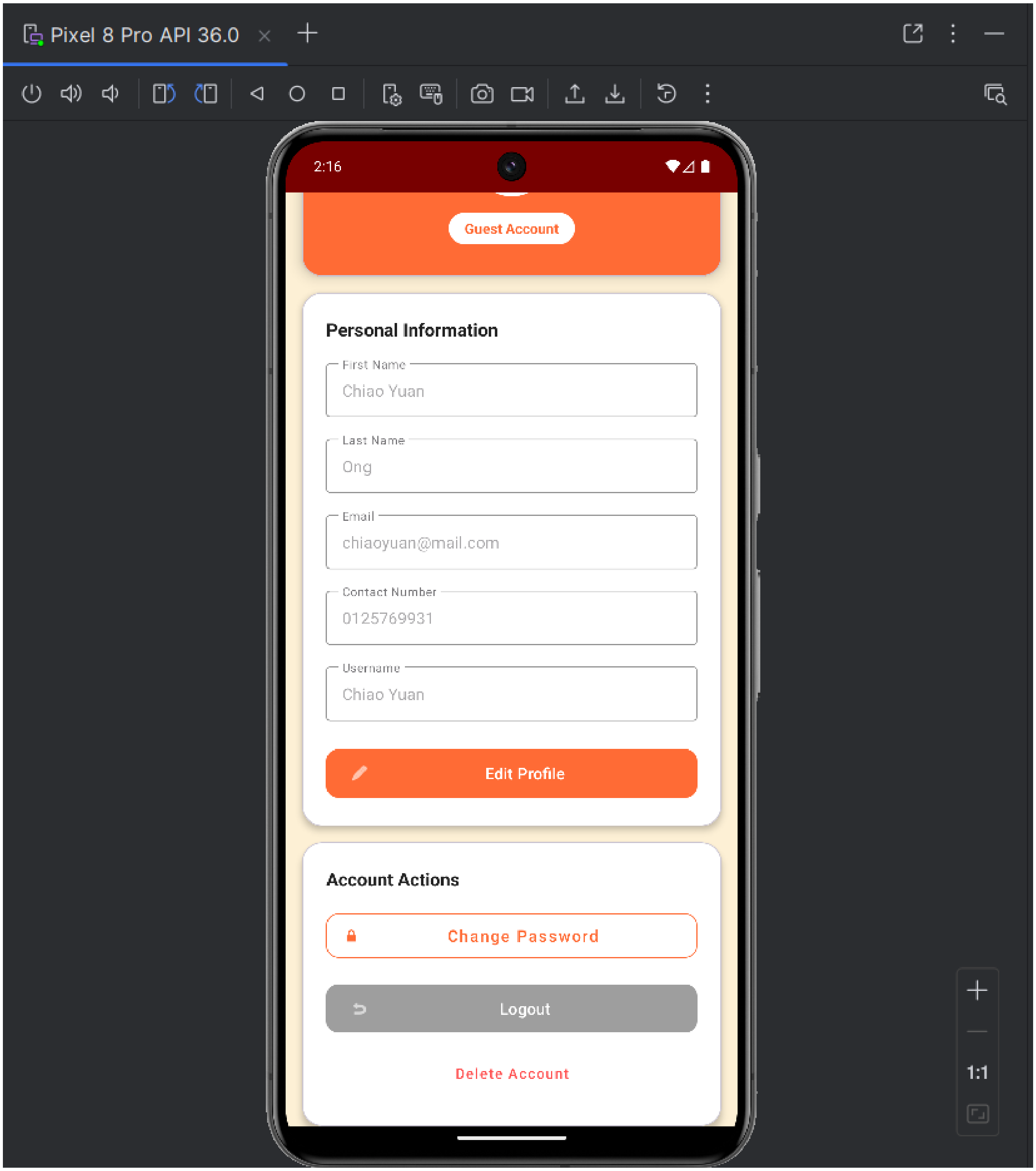
Last Name: Ong

Email: chiaoyuan@mail.com

Contact Number: 0125769931

Username: Chiao Yuan

Figure 9 : Guest Profile Page 1



Personal Information

First Name: Chiao Yuan

Last Name: Ong

Email: chiaoyuan@mail.com

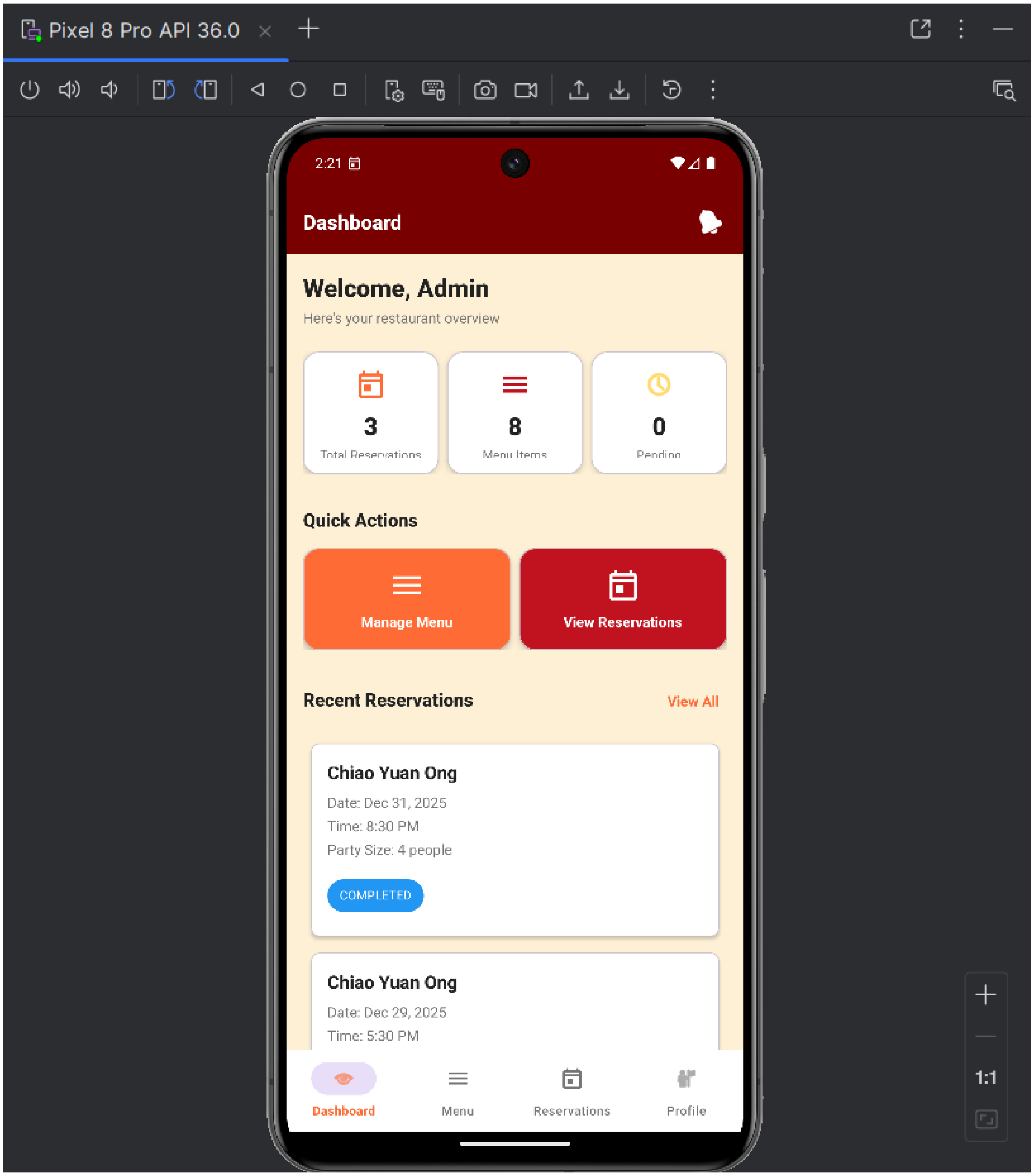
Contact Number: 0125769931

Username: Chiao Yuan

Figure 10 : Guest Profile Page 2

Staff see slightly different options:

- Dashboard – overview of daily operations
- Menu Management – add, edit, remove dishes
- Reservations – handle booking requests
- Profile – account settings



Welcome, Admin

Here's your restaurant overview

3 Total Reservations

8 Menu Items

0 Pending

Chiao Yuan Ong

Date: Dec 31, 2025

Time: 8:30 PM

Party Size: 4 people

Chiao Yuan Ong

Date: Dec 29, 2025

Time: 5:30 PM

Figure 11 : Staff Dashboard

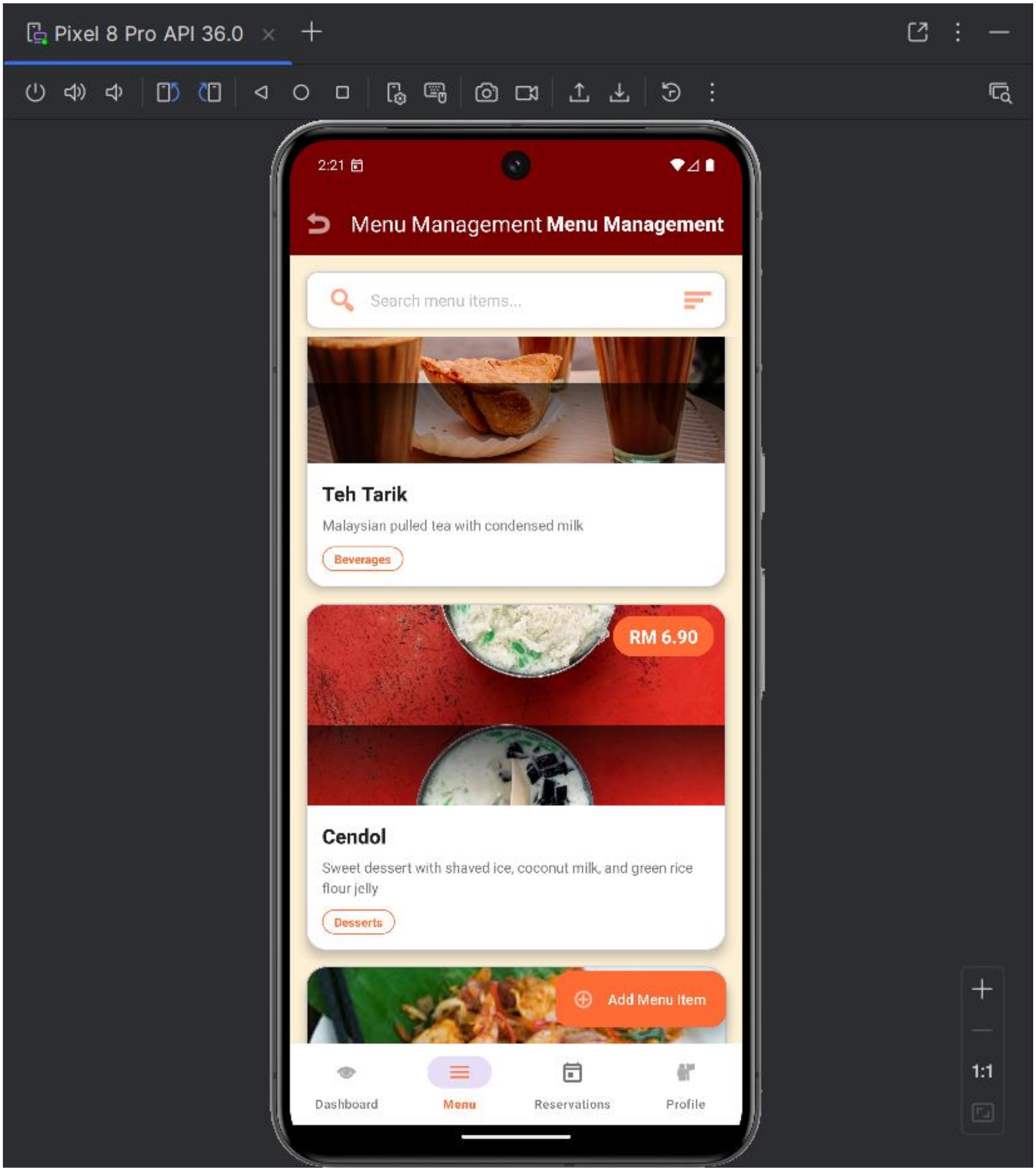
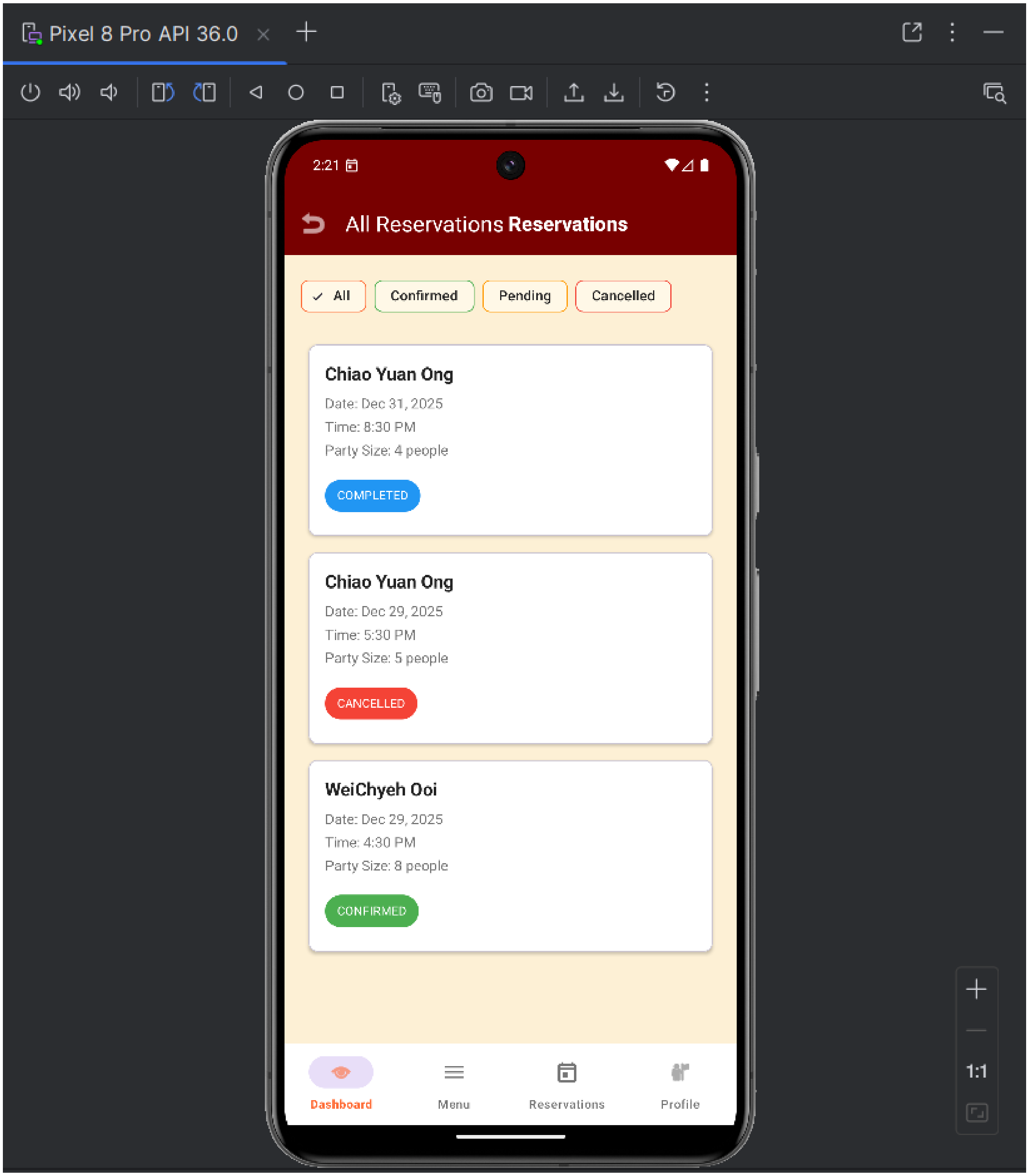

Pixel 8 Pro API 36.0
2:21
Menu Management Menu Management
Search menu items...
Teh Tarik
Malaysian pulled tea with condensed milk
Beverages
RM 6.90
Cendol
Sweet dessert with shaved ice, coconut milk, and green rice flour jelly
Desserts
Add Menu Item
Dashboard
Menu
Reservations
Profile
1:1

Figure 12 : Staff Menu Management Page



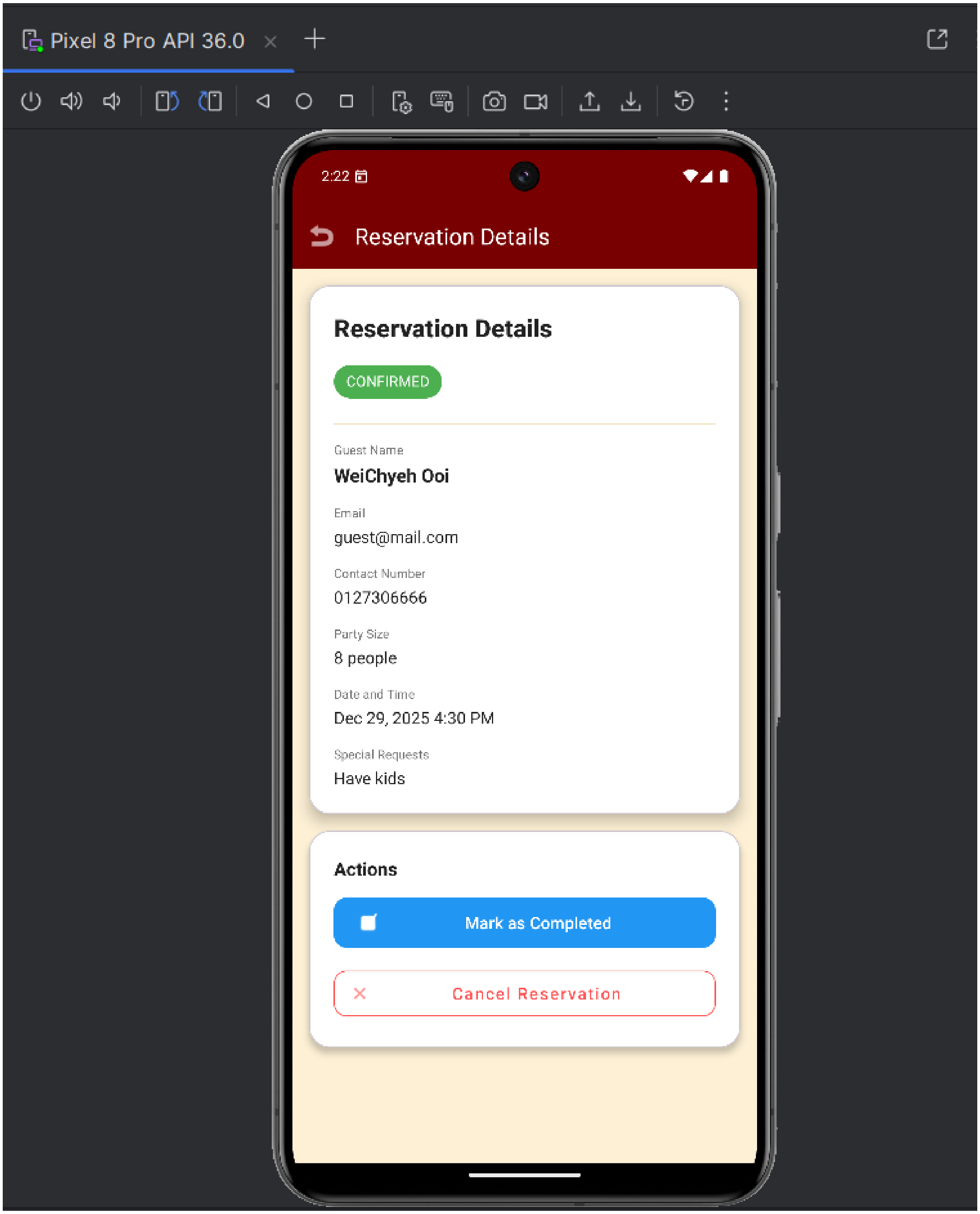
All Reservations Reservations

Chiao Yuan Ong
Date: Dec 31, 2025
Time: 8:30 PM
Party Size: 4 people
COMPLETED

Chiao Yuan Ong
Date: Dec 29, 2025
Time: 5:30 PM
Party Size: 5 people
CANCELLED

WeiChyeh Ooi
Date: Dec 29, 2025
Time: 4:30 PM
Party Size: 8 people
CONFIRMED

Figure 13 : Staff Reservation Page made by Guest



Reservation Details

Reservation Details

CONFIRMED

Guest Name
WeiChyeh Ooi

Email
guest@mail.com

Contact Number
0127306666

Party Size
8 people

Date and Time
Dec 29, 2025 4:30 PM

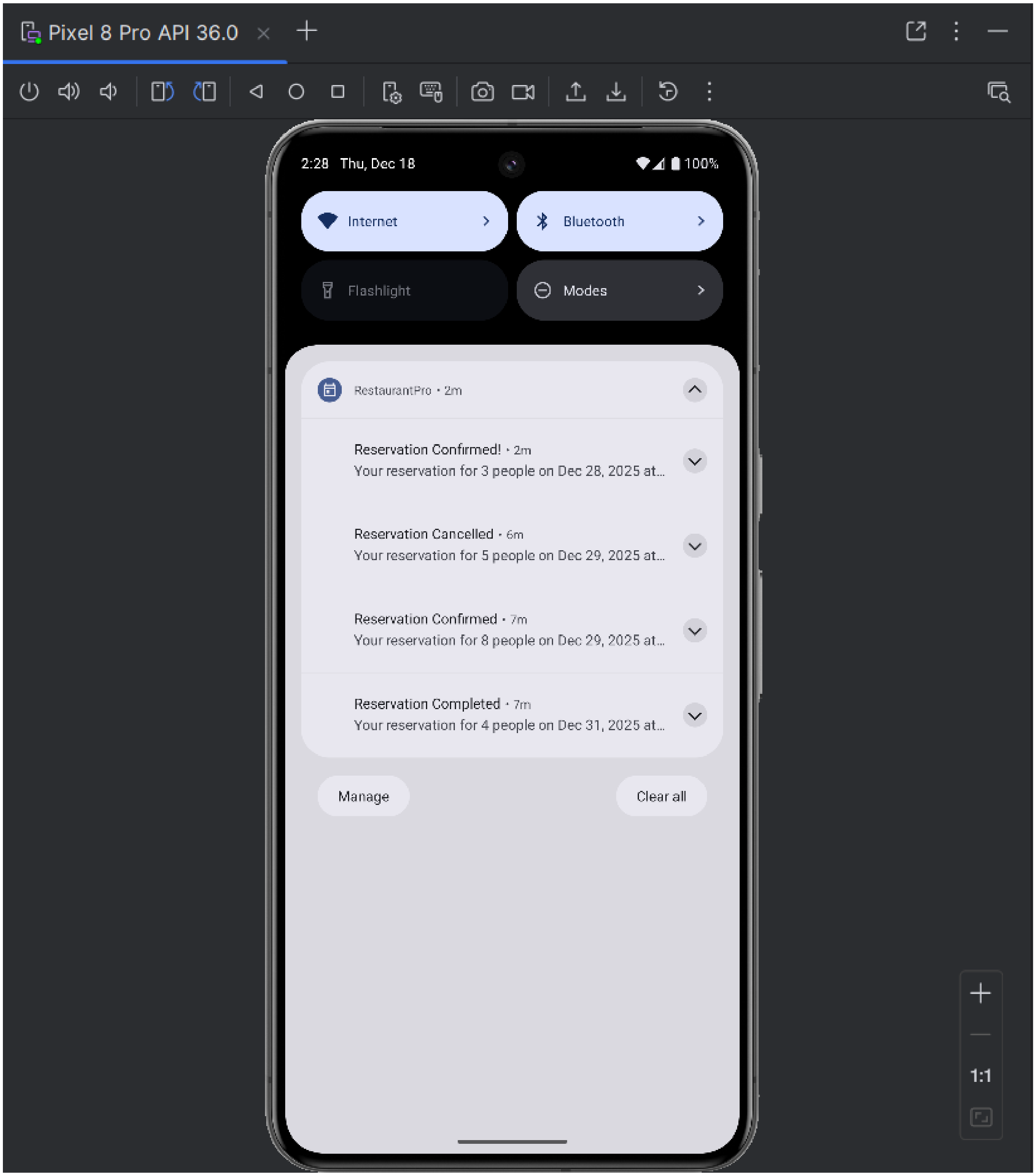
Special Requests
Have kids

Actions

Mark as Completed

× Cancel Reservation

Figure 14 : Staff Reservation Detail Page



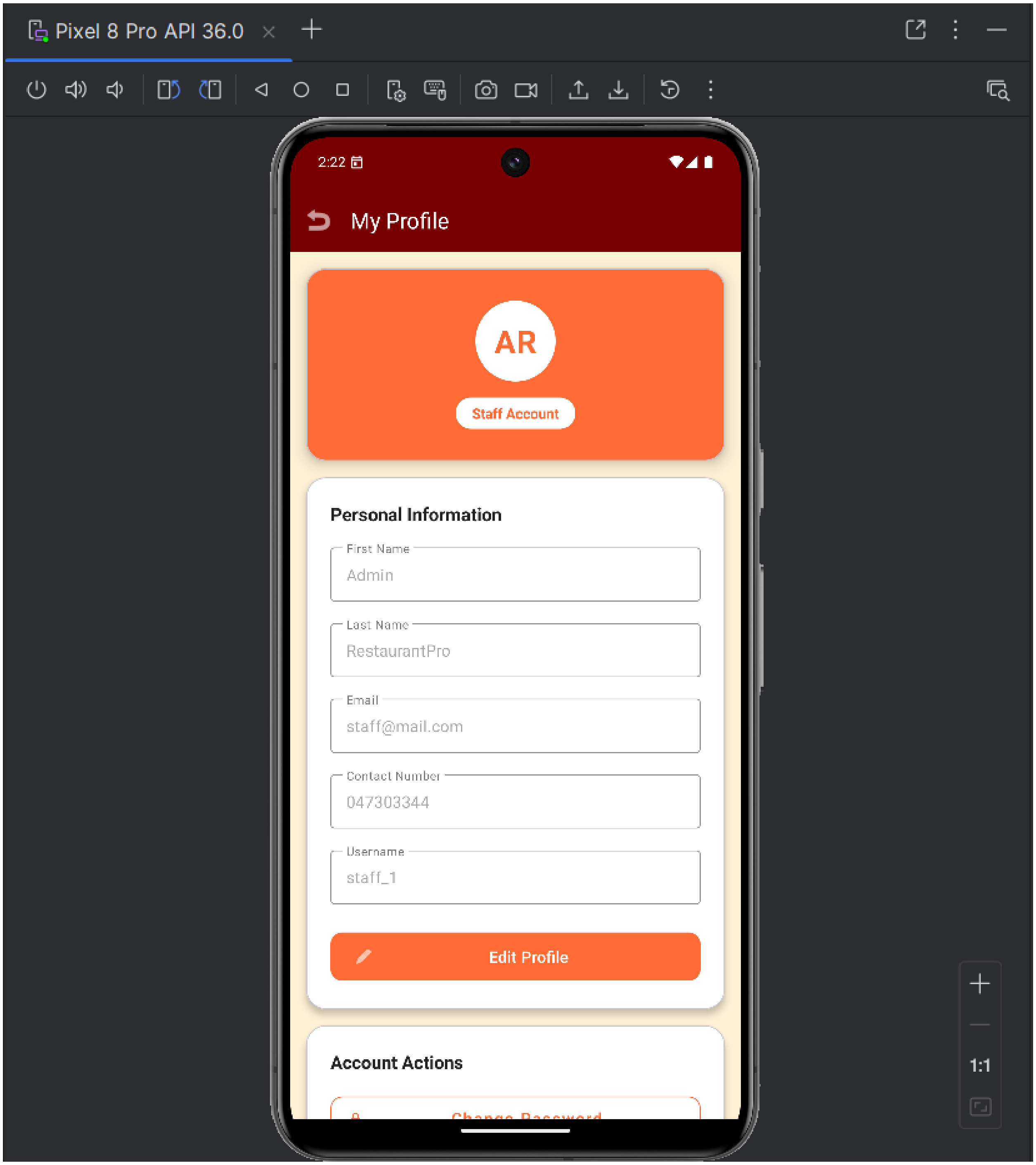
Reservation Confirmed! • 2m
Your reservation for 3 people on Dec 28, 2025 at...

Reservation Cancelled • 6m
Your reservation for 5 people on Dec 29, 2025 at...

Reservation Confirmed • 7m
Your reservation for 8 people on Dec 29, 2025 at...

Reservation Completed • 7m
Your reservation for 4 people on Dec 31, 2025 at...

Figure 15 : Notification Received by Staff when made action of Guests Booking



My Profile

AR

Staff Account

Personal Information

First Name: Admin

Last Name: RestaurantPro

Email: staff@mail.com

Contact Number: 047303344

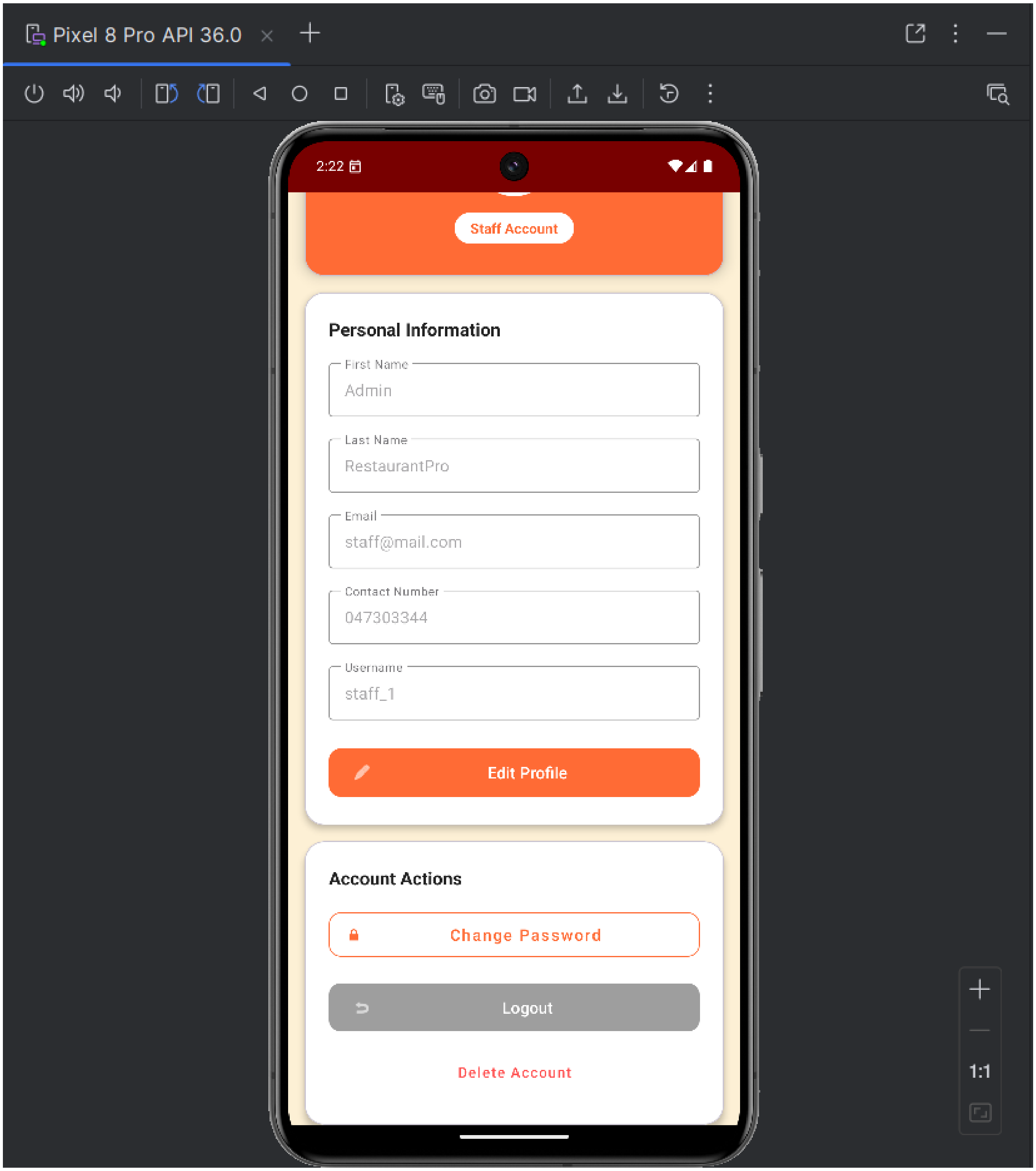
Username: staff_1

Edit Profile

Account Actions

Change Password

Figure 16 : Staff Profile Page 1



Personal Information

First Name: Admin

Last Name: RestaurantPro

Email: staff@mail.com

Contact Number: 047303344

Username: staff_1

Account Actions

Figure 17 : Staff Profile Page 2

The decision to use different navigation structures for different user types follows Occam's Razor rule (the simplest solution that serves each group's needs). Staff need quick access to management functions, while guests prioritize browsing and booking.

2.3 Typography and Readability

Text hierarchy uses system fonts (sans-serif) at carefully chosen sizes:

- Headers: 24sp, bold
- Body text: 16sp, regular
- Supporting text: 14sp, regular
- Captions: 12sp, medium

These aren't random numbers. Android's accessibility guidelines recommend minimum 16sp for body text to ensure readability across age groups. The 1.5 ratio between header and body text provides clear visual hierarchy without feeling jarring.

2.4 Component Selection

2.4.1 Input Fields

TextInputLayout from Material Components wraps all input fields, providing:

- Floating label animation
- Error message display
- Character counters (for notes fields)
- Password visibility toggles

When an email field contains invalid input, the error message appears below the field in red and the outline changes color such as multiple visual cues that something needs attention.

```
<com.google.android.material.textfield.TextInputLay
out

style="@style/Widget.MaterialComponents.TextInputLa
yout.OutlinedBox"
    android:hint="Email"
    app:boxStrokeColor="@color/primary_orange"

app:startIconDrawable="@android:drawable/ic_dialog_
email">


<com.google.android.material.textfield.TextInputEdi
tText
        android:inputType="textEmailAddress"
        android:padding="16dp"/>
</com.google.android.material.textfield.TextInputLa
yout>
```

The padding (16dp) provides comfortable tap targets. Android's minimum recommended touch target is 48dp which these fields exceed when accounting for the entire input area.

2.4.2 Cards and Elevation

Menu items and reservations use MaterialCardView with 8dp elevation. Cards appear to float above the background, suggesting they're interactive. The corner radius (20dp for major cards, 16dp for list items) softens the interface without looking childish.

```
<com.google.android.material.card.MaterialCardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:cardElevation="8dp"
    app:cardCornerRadius="20dp">
    <!-- Content -->
</com.google.android.material.card.MaterialCardView
>
```

Elevation values follow Material's recommendations, though I reduced them slightly. The default 8dp often felt too prominent, so cards use 4dp in list contexts and 8dp for standalone cards that need emphasis.

2.4.3 Buttons and Actions

Primary actions use filled buttons with 56dp height—large enough to tap comfortably but not so large they dominate the screen. The MaterialButton component handles touch feedback automatically, showing a ripple effect that confirms user interaction.

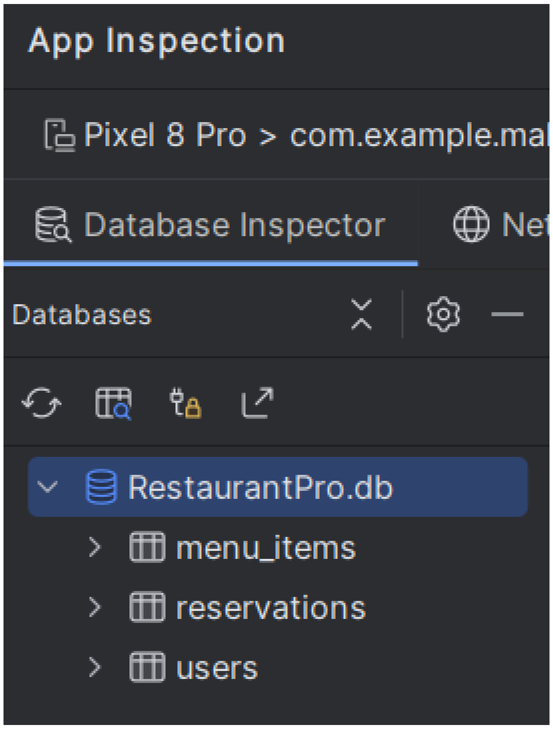
```
<com.google.android.material.button.MaterialButton
    android:layout_width="match_parent"
    android:layout_height="56dp"
    android:text="Confirm Reservation"
    app:backgroundTint="@color/primary_orange"
    app:cornerRadius="12dp"/>
```

Secondary actions use text buttons or outlined buttons, visually de-emphasized compared to primary actions that would helps users identify the main action on each screen.

3.0 Data Management

3.1 Database Architecture

SQLite handles local data persistence. The choice between SQLite and Room (Android's database wrapper) wasn't obvious. Room provides compile-time SQL verification and reduces boilerplate, but I stuck with SQLite directly to maintain transparency in how queries work. The database schema includes three main tables:

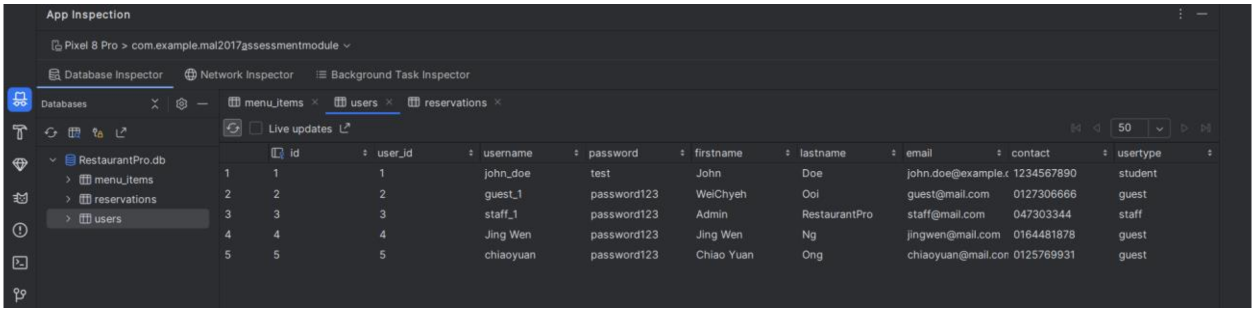


RestaurantPro.db
- menu_items
- reservations
- users

Figure 18 : SQLite Database Inspector with tables

3.1.1 Users Table

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER UNIQUE,
    username TEXT UNIQUE,
    password TEXT,
    firstname TEXT,
    lastname TEXT,
    email TEXT UNIQUE,
    contact TEXT,
    usertype TEXT
)
```



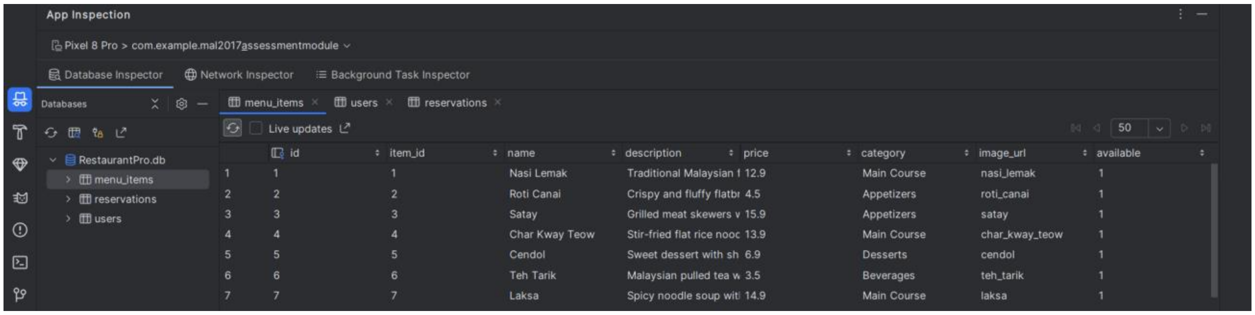
	id	user_id	username	password	firstname	lastname	email	contact	usertype
1	1	1	john_doe	test	John	Doe	john.doe@example.c	1234567890	student
2	2	2	guest_1	password123	WeiChyeh	Ooi	guest@mail.com	0127306666	guest
3	3	3	staff_1	password123	Admin	RestaurantPro	staff@mail.com	047303344	staff
4	4	4	Jing Wen	password123	Jing Wen	Ng	jingwen@mail.com	0164481878	guest
5	5	5	chiaoyuan	password123	Chiao Yuan	Ong	chiaoyuan@mail.con	0125769931	guest

Figure 19 : SQLite database table for Users

Email and username have UNIQUE constraints, preventing duplicate accounts. Passwords are stored in plain text locally but acceptable for a prototype. In a real system, even local storage would use encryption, possibly with Android's EncryptedSharedPreferences.

3.1.2 Menu Items Table

```
CREATE TABLE menu_items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    item_id INTEGER UNIQUE,
    name TEXT,
    description TEXT,
    price REAL,
    category TEXT,
    image_url TEXT,
    available INTEGER
)
```



	id	item_id	name	description	price	category	image_url	available
1	1	1	Nasi Lemak	Traditional Malaysian t	12.9	Main Course	nasi_lemak	1
2	2	2	Roti Canai	Crispy and fluffy flatbı	4.5	Appetizers	roti_canai	1
3	3	3	Satay	Grilled meat skewers v	15.9	Appetizers	satay	1
4	4	4	Char Kway Teow	Stir-fried flat rice nooc	13.9	Main Course	char_kway_teow	1
5	5	5	Cendol	Sweet dessert with sh	6.9	Desserts	cendol	1
6	6	6	Teh Tarik	Malaysian pulled tea w	3.5	Beverages	teh_tarik	1
7	7	7	Laksa	Spicy noodle soup wit	14.9	Main Course	laksa	1

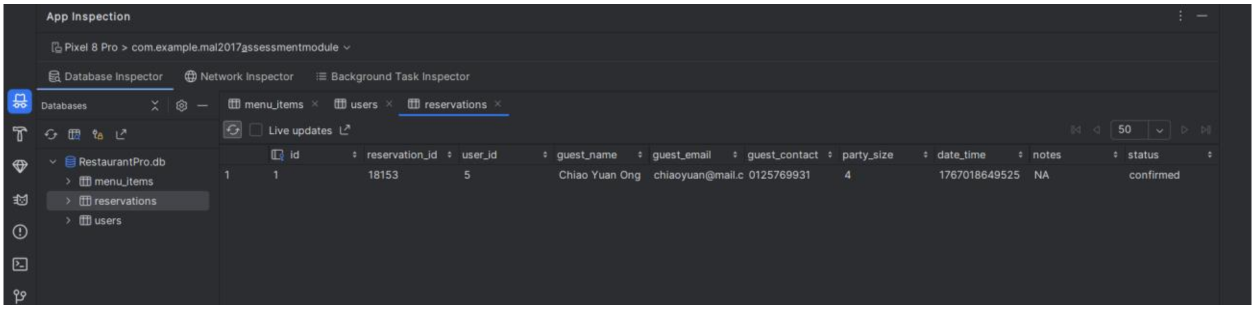
Figure 20 : SQLite database table for Menu

The available field uses INTEGER (0/1) rather than BOOLEAN because SQLite doesn't have a native boolean type. This is a SQLite quirk that trips up developers coming from other databases.

Categories aren't foreign keys to a separate categories table somehow denormalization simplifies queries and makes sense given the limited, stable set of categories (Appetizers, Main Course, Desserts, Beverages). If categories needed complex attributes or frequently changed, a separate table would be warranted.

3.1.3 Reservation Table

```
CREATE TABLE reservations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    reservation_id INTEGER UNIQUE,
    user_id INTEGER,
    guest_name TEXT,
    guest_email TEXT,
    guest_contact TEXT,
    party_size INTEGER,
    date_time INTEGER,
    notes TEXT,
    status TEXT
)
```



	id	reservation_id	user_id	guest_name	guest_email	guest_contact	party_size	date_time	notes	status
1	1	18153	5	Chiao Yuan Ong	chiaoyuan@mail.c	0125769931	4	1767018649525	NA	confirmed

Figure 21 : SQLite database table for Reservation

The date_time field stores Unix timestamps (milliseconds since epoch) as INTEGER. Java's Calendar and SimpleDateFormat classes work seamlessly with this format, and it avoids SQLite's somewhat awkward date handling.

Status values ("pending", "confirmed", "cancelled", "completed") are also plain text. An enum would be cleaner in the Java code, but SQLite doesn't support enums natively, so I enforce constraints in the application layer.

3.2 DatabaseHelper Implementation

The DatabaseHelper class extends SQLiteOpenHelper, implementing the singleton pattern:

```
private static DatabaseHelper instance;

public static synchronized DatabaseHelper
getInstance(Context context) {
    if (instance == null) {
        instance = new
DatabaseHelper(context.getApplicationContext());
    }
    return instance;
}
```

This prevents multiple database instances which could cause threading issues or resource conflicts. However in practice, the app initializes the database on the main thread before any background operations occur.

3.3 API Integration

API communication uses Volley rather than Retrofit which might seem unconventional. Retrofit is generally favored for its cleaner syntax and RxJava integration, but Volley is lighter and simpler for straightforward REST calls. Given the API's basic structure (standard HTTP methods with JSON payloads), Volley's request queue and built-in caching were sufficient.

3.3.1 ApiClient Singleton

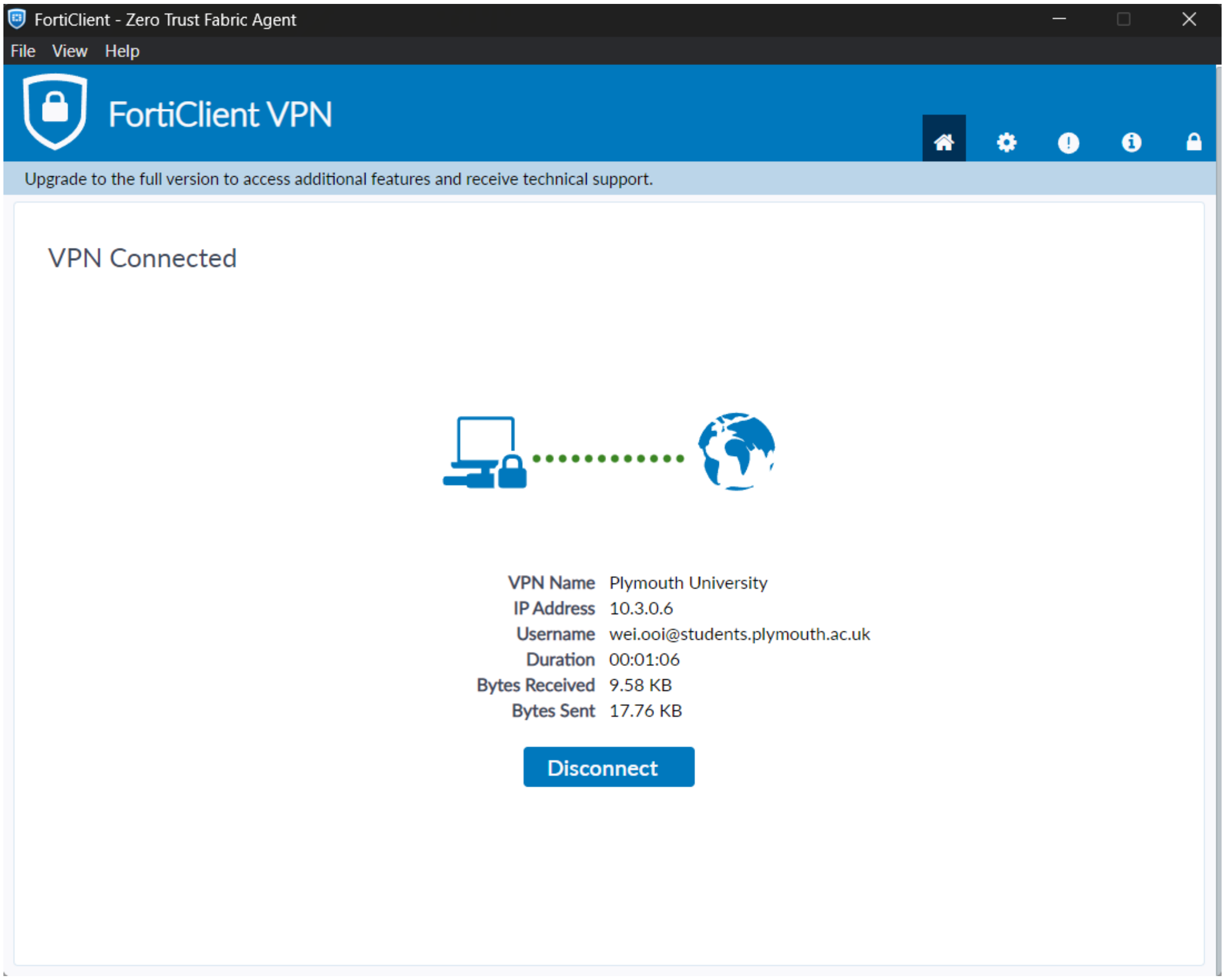
```
public class ApiClient {
    private static final String BASE_URL =
"http://10.240.72.69/comp2000/coursework/";
    private static ApiClient instance;
    private RequestQueue requestQueue;

    public static synchronized ApiClient
getInstance(Context context) {
        if (instance == null) {
            instance = new ApiClient(context);
        }
        return instance;
    }

    public RequestQueue getRequestQueue() {
        if (requestQueue == null) {
            requestQueue =
Volley.newRequestQueue(context);
        }
        return requestQueue;
    }
}
```

The BASE_URL points to Plymouth University's API server, accessed via FortiClient VPN.

3.3.2 API Service Connection

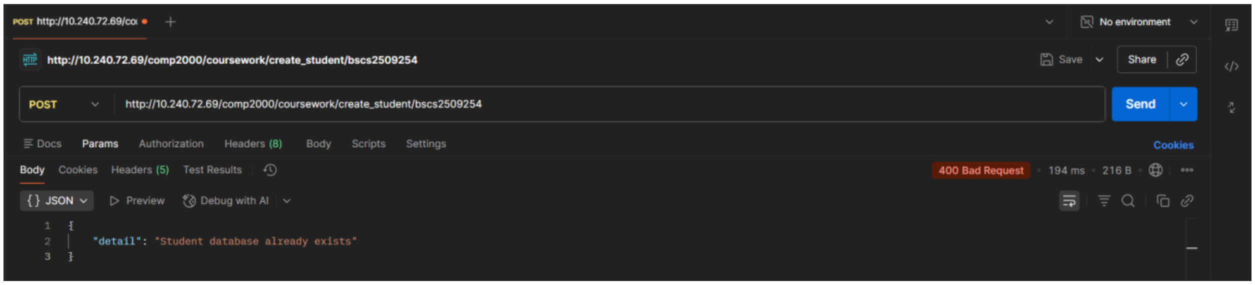


Upgrade to the full version to access additional features and receive technical support.

VPN Connected

VPN Name Plymouth University
IP Address 10.3.0.6
Username wei.ooi@students.plymouth.ac.uk
Duration 00:01:06
Bytes Received 9.58 KB
Bytes Sent 17.76 KB

Figure 22 : FortiClient VPN Connection



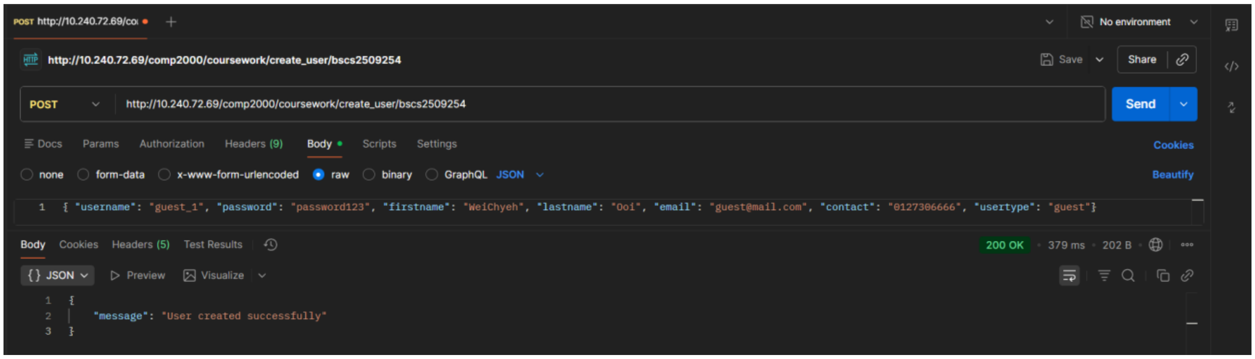
http://10.240.72.69/comp2000/coursework/create_student/bscs2509254

POST http://10.240.72.69/comp2000/coursework/create_student/bscs2509254

400 Bad Request · 194 ms · 216 B

```
{
    "detail": "Student database already exists"
}
```

Figure 23 : Postman API Test Case 1 – create_student for Database



http://10.240.72.69/comp2000/coursework/create_user/bscs2509254

POST http://10.240.72.69/comp2000/coursework/create_user/bscs2509254

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

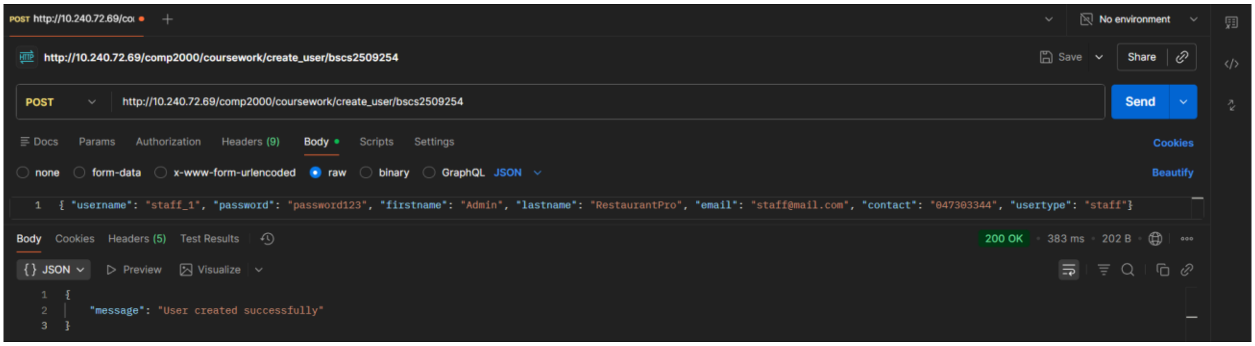
```
{ "username": "guest_1", "password": "password123", "firstname": "WeiChyeh", "lastname": "Ooi", "email": "guest@mail.com", "contact": "0127306666", "usertype": "guest"}
```

Body 200 OK 379 ms 202 B

{} JSON

```
{
    "message": "User created successfully"
}
```

Figure 24 : Postman API Test Case 2 – create_user for Guest



http://10.240.72.69/comp2000/coursework/create_user/bscs2509254

POST http://10.240.72.69/comp2000/coursework/create_user/bscs2509254

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

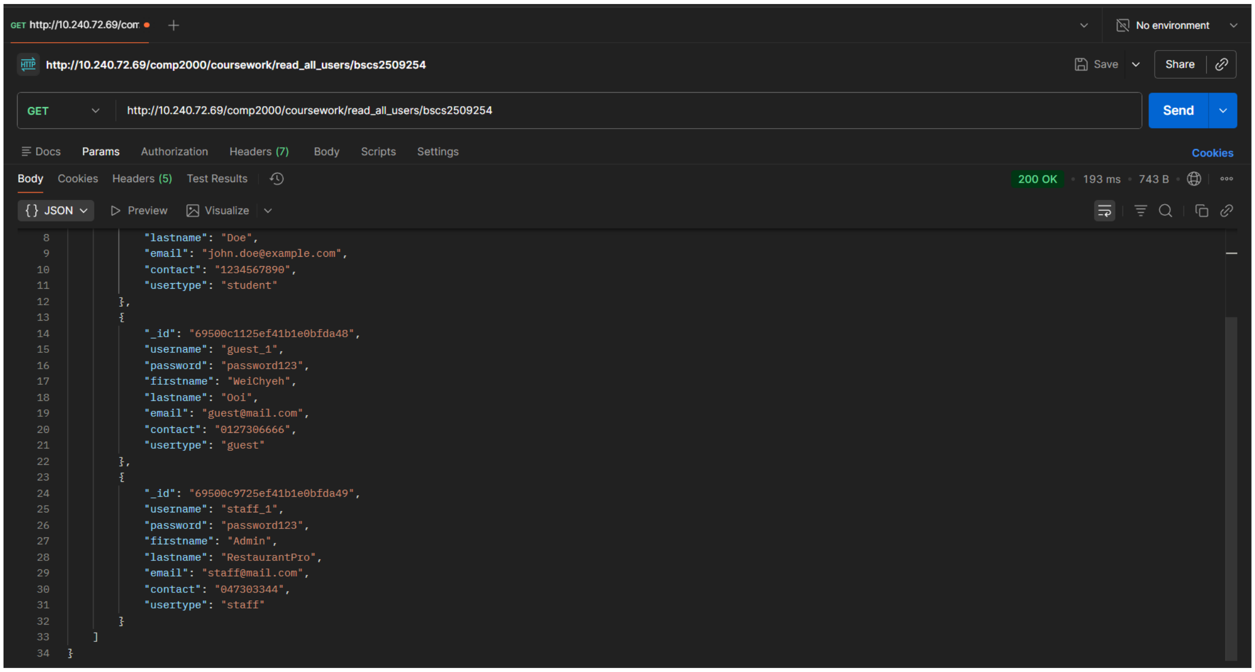
```
{ "username": "staff_1", "password": "password123", "firstname": "Admin", "lastname": "RestaurantPro", "email": "staff@mail.com", "contact": "047303344", "usertype": "staff"}
```

Body 200 OK 383 ms 202 B

{} JSON

```
{
    "message": "User created successfully"
}
```

Figure 25 : Postman API Test Case 3 - create_user for Staff



http://10.240.72.69/comp2000/coursework/read_all_users/bscs2509254

GET http://10.240.72.69/comp2000/coursework/read_all_users/bscs2509254

Body 200 OK 193 ms 743 B

{} JSON

```
            "lastname": "Doe",
            "email": "john.doe@example.com",
            "contact": "1234567890",
            "usertype": "student"
        },
        {
            "_id": "69500c1125ef41b1e0bfda48",
            "username": "guest_1",
            "password": "password123",
            "firstname": "WeiChyeh",
            "lastname": "Ooi",
            "email": "guest@mail.com",
            "contact": "0127306666",
            "usertype": "guest"
        },
        {
            "_id": "69500c9725ef41b1e0bfda49",
            "username": "staff_1",
            "password": "password123",
            "firstname": "Admin",
            "lastname": "RestaurantPro",
            "email": "staff@mail.com",
            "contact": "047303344",
            "usertype": "staff"
        }
    ]
}
```

Figure 26 : Postman Test Case 2 – read_all_user

3.4 Data Synchronization Strategy

The app uses a "local-first" approach: data is read from SQLite immediately, then synced with the API in the background. This keeps the UI responsive even with slow network connections.

Login demonstrates this pattern:

```
private void handleLogin() {
    String email =
etEmail.getText().toString().trim();
    String password =
etPassword.getText().toString().trim();

    showLoading(true);

    // Try API first
    authenticateWithApi(email, password);
}

private void authenticateWithApi(String email,
String password) {
    apiService.getAllUsers(Constants.STUDENT_ID,
        response -> {
            List<User> users =
response.get("users");
            User authenticatedUser =
findUserByCredentials(users, email, password);

            if (authenticatedUser != null) {

onApiAuthenticationSuccess(authenticatedUser);
            } else {

authenticateWithLocalDatabase(email, password);
            }
        },
        error -> {
            // API failed, fall back to local
database
            authenticateWithLocalDatabase(email,
password);
        });
}
```

If the API call succeeds, the user is synced to the local database. If it fails (no network, server down etc.), authentication falls back to local data. This dual approach ensures the app remains functional offline.

3.5 Data Validation

Validation occurs at multiple layers:

I. **UI Layer** – Input fields show errors immediately:

```
if (TextUtils.isEmpty(email)) {
tilEmail.setError("Email is required");
return false;
}
```

II. **Business Logic Layer** – Models validate their state:

```
public boolean isValid() {
return partySize >= Constants.MIN_PARTY_SIZE
&& partySize <= Constants.MAX_PARTY_SIZE;
}
```

III. **Database Layer** – SQL constraints prevent invalid data:

```
email TEXT UNIQUE  -- Prevents duplicate emails
```

This layered approach catches errors early and would reduce invalid API requests and improving the user experience.

4.0 Application Control

4.1 Activity Lifecycle Management

Android's activity lifecycle requires careful state management. Activities can be destroyed and recreated at any time such as screen rotation, low memory, backgrounding etc. The app handles this through several mechanisms.

4.1.1 Saving State

Critical UI state is saved in onSaveInstanceState():

```
@Override
protected void onSaveInstanceState(Bundle outState)
{
    super.onSaveInstanceState(outState);
    outState.putString("current_category",
currentCategory);
    outState.putInt("scroll_position",
layoutManager.findFirstVisibleItemPosition());
}
```

If someone is browsing "Main Course" items and rotates their device, they'll still be viewing Main Course items afterward rather than being reset to "All."

4.1.2 Session Management

The SessionManager class persists login state using SharedPreferences:

```
public class SessionManager {
    private static final String KEY_IS_LOGGED_IN =
"is_logged_in";
    private static final String KEY_USER_JSON =
"user_json";

    public void createLoginSession(User user) {
        String userJson = gson.toJson(user);
        editor.putBoolean(KEY_IS_LOGGED_IN, true);
        editor.putString(KEY_USER_JSON, userJson);
        editor.commit();
    }

    public boolean isLoggedIn() {
        return prefs.getBoolean(KEY_IS_LOGGED_IN,
false);
    }
}
```

SharedPreferences persists across app restarts, so users remain logged in until they explicitly log out. The user object is serialized to JSON rather than storing individual fields separately.

4.2 Navigation Flow

User flow through the app follows standard Android patterns:

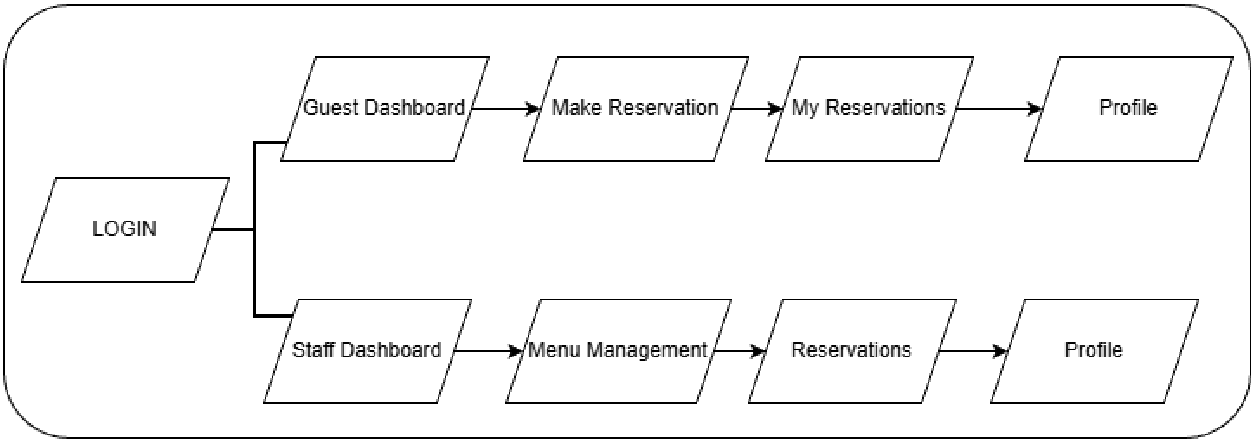


Figure 27 : Flow Diagram of RestaurantPro

The LoginActivity checks session state in onCreate():

```
if (sessionManager.isLoggedIn()) {
    navigateToHome();
    return;
}
```

If a valid session exists, the user bypasses the login screen entirely. This is standard practice but worth noting by forcing users to log in every time they open the app would be incredibly annoying.

4.3 Error Handling

The app distinguishes between different error types:

Validation Errors – Shown immediately in the UI:

```
tilEmail.setError("Please enter a valid email
address");
```

Network Errors – Fall back to local data:

```
error -> {
    Toast.makeText(this, "API unavailable, using
local data",
        Toast.LENGTH_SHORT).show();
    authenticateWithLocalDatabase(email, password);
}
```

Database Errors – Logged and shown to user:

```
catch (Exception e) {
    Log.e(TAG, "Database error: " + e.getMessage(),
e);
    Toast.makeText(this, "Error saving data",
Toast.LENGTH_SHORT).show();
}
```

This differentiation helps users understand what went wrong. "Invalid email" is actionable; "Network error" explains why they might see old data; "Database error" at least acknowledges something failed rather than silently ignoring it.

4.4 Asynchronous Operations

Network and database operations run on background threads to avoid blocking the UI:

```
new Thread(() -> {
    try {
        User user =
dbHelper.getUserByCredentials(email, password);

        runOnUiThread(() -> {
            showLoading(false);
            if (user != null) {
                onLoginSuccess(user);
            } else {
                showError("Invalid credentials");
            }
        });
    } catch (Exception e) {
        // Handle error
    }
}).start();
```

The pattern is straightforward: start a background thread for the operation, then use runOnUiThread() to update UI elements with the results. Modern Android development would use Kotlin coroutines or RxJava here, but plain Java threads are more transparent about what's happening.

4.5 Input Handling

Date and time inputs use Android's built-in picker dialogs:

```
private void showDatePicker() {
    DatePickerDialog dialog = new DatePickerDialog(
        this,
        (view, year, month, dayOfMonth) -> {
            selectedDate.set(year, month,
dayOfMonth);

etDate.setText(formatDate(selectedDate));
        },
        selectedDate.get(Calendar.YEAR),
        selectedDate.get(Calendar.MONTH),
        selectedDate.get(Calendar.DAY_OF_MONTH)
    );


dialog.getDatePicker().setMinDate(System.currentTim
eMillis());
    dialog.show();
}
```

The date picker prevents selecting past dates (setMinDate), and the time picker validates restaurant hours (10 AM - 10 PM). These constraints are enforced in the UI rather than letting users submit invalid data and showing an error afterward—prevents frustration.

5.0 Design Practices and Choices

5.1 Applying Occam's Razor

The login system demonstrates this principle well. Rather than implementing separate login flows for guests and staff, both use the same screen and credentials system—differentiation happens after authentication based on the user's type to reduces redundancy and UI complexity.

Testing showed this confused user who didn't realize they were functionally identical. Merging them simplified the interface and eliminated unnecessary branching logic.

5.2 Material Design Application

Material Design provided a framework, but not every guideline was followed strictly. For instance, Material recommends FABs (Floating Action Buttons) for primary actions. The menu management screen uses a FAB for "Add Menu Item," which works well:

```
<com.google.android.material.floatingactionbutton.E
xtendedFloatingActionButton
    android:id="@+id/fab_add_menu_item"
    android:text="Add Menu Item"
    android:layout_margin="16dp"
    app:icon="@android:drawable/ic_menu_add"/>
```

However, the guest dashboard doesn't use a FAB for "Make Reservation" because it's a prominent button within the welcome card instead. The decision came down to context: staff frequently add menu items while viewing the list, so a FAB makes sense. Guests might browse for a while before deciding to book, so the action doesn't need to be constantly floating above content.

5.3 Error Prevention vs. Error Handling

Where possible, the UI prevents errors rather than just handling them gracefully. Examples:

- Date Picker– Can't select past dates
- Time Picker – Only shows restaurant operating hours
- Party Size– Dropdown limits selections to 1-8 people
- Password Confirmation– Compared in real-time, error shown immediately

If a user can't submit invalid data in the first place, they don't experience the annoyance of filling out a form, tapping submit and saw an error message.

5.4 Feedback and Affordances

Every interactive element provides immediate feedback. Buttons ripple when pressed, navigation items highlight when selected, cards show pressed states. This seems obvious, but I've used apps where tapping something produces no response, leaving me unsure whether my touch registered.

5.5 Consistency

The app maintains consistency in several ways:

- Color usage– Orange always indicates primary actions, brown for navigation bars, cream for backgrounds
- Spacing– 16dp standard margin, 8dp between related elements, 24dp between unrelated sections
- Typography– Same size/weight hierarchy throughout
- Interaction patterns– Buttons work the same way everywhere, navigation behaves predictably

Users learn patterns once and apply them throughout the app, rather than having to figure out how each screen works independently (Norman, 2013).

5.6 Scalability Considerations

That said, I avoided over-engineering. The database doesn't have a complex normalization structure because the data volume doesn't justify it. The API layer is simple because the API itself is simple. Premature optimization would have wasted time on problems that don't exist yet.

6.0 Usability Testing

A structured summative usability test was conducted after the high-fidelity implementation, following the CW1 formative evaluation.

6.1 Method

Participants interacted with the high-fidelity prototype before completing a Google Form questionnaire. Due to the large number of participants, informed consent was obtained digitally via an introductory consent statement within Google Form. An online consent statement requiring explicit agreement ensured ethical compliance and anonymity.

6.2 Participants

A total of 8 participants from both staff (3 people) and guest(5 people) profiles responded:

Staff users: The Ship Campus Windjammer Café staff familiar with handling reservations

Guest users: Campus Students representing typical diners

6.3 Summary of Findings

I. Learnability
   a. Participants rated the clarity of the interface highly, particularly acknowledging the intuitive login process and straightforward navigation structure.

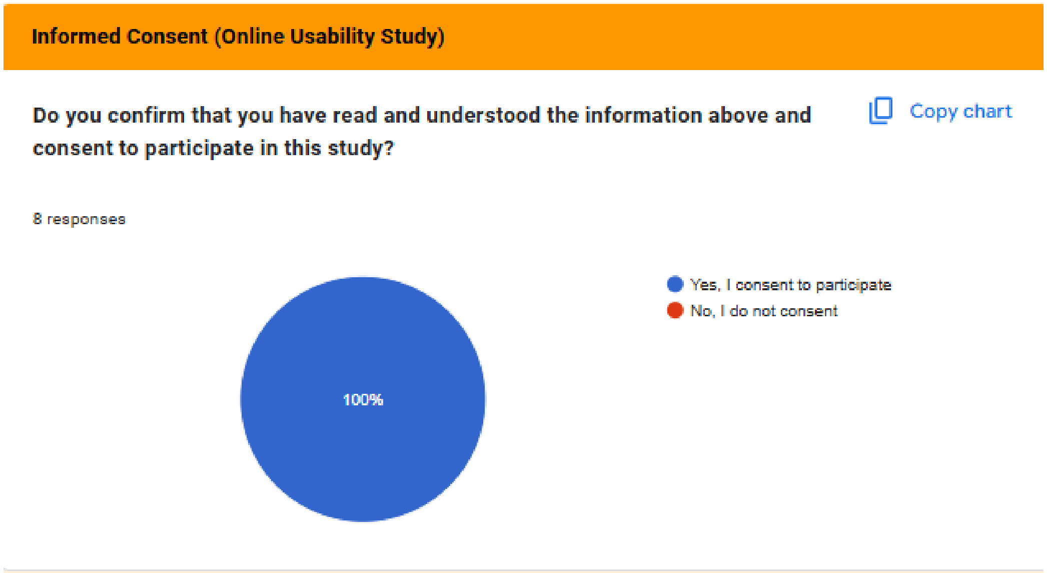
II. Navigation
   a. Most users found the bottom navigation and card-based layout easy to interpret. Feedback highlighted that the icons and labels were helpful in maintaining orientation.

III. Task Efficiency
   a. Staff participants appreciated the colour-coded reservation list, noting that it "speeded up recognition during busy periods." Guests found the booking process simple, though some commented that button spacing could be further refined.

IV. Satisfaction
   a. Overall satisfaction scores indicated positive reception, confirming that the final high-fidelity design addressed issues observed during the formative evaluation.

6.4 Questionnaire Result

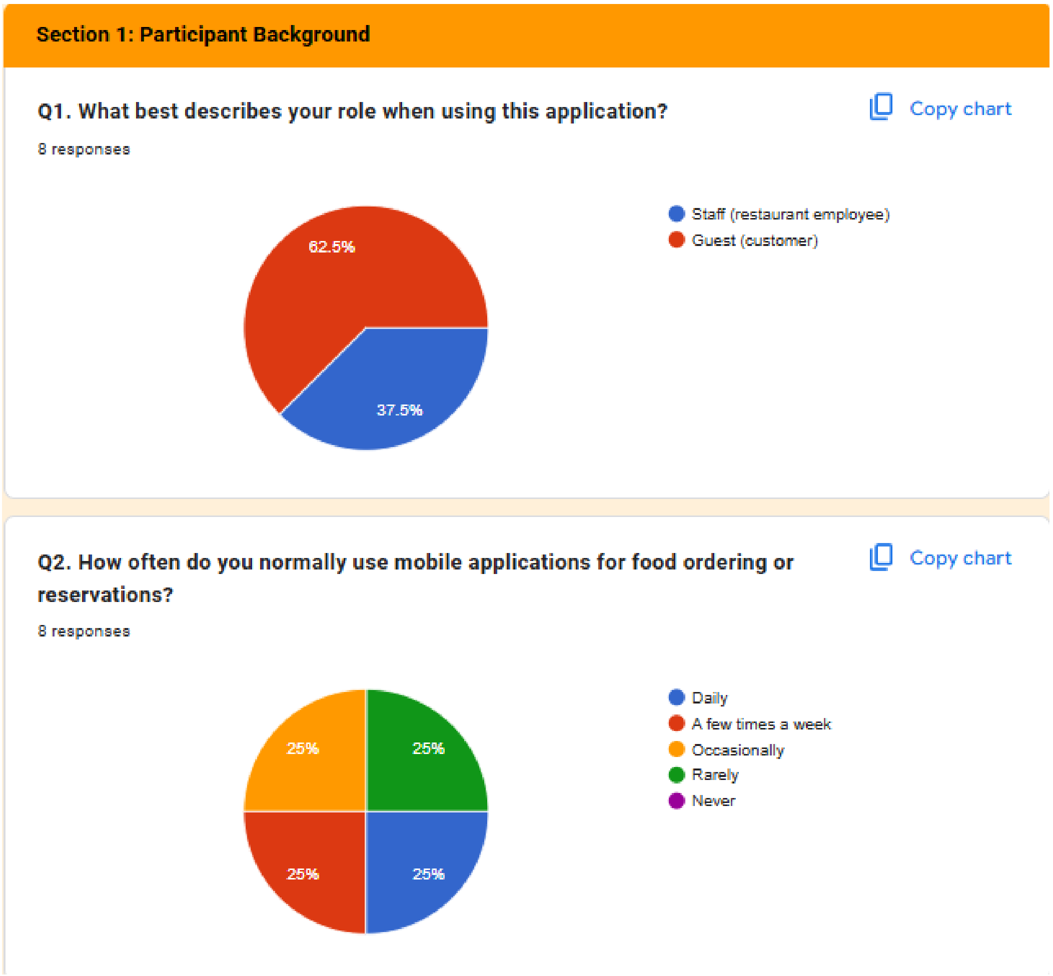
Included a set of questionnaire from Section 1 till Section 6.



Informed Consent (Online Usability Study)

Do you confirm that you have read and understood the information above and consent to participate in this study?

8 responses



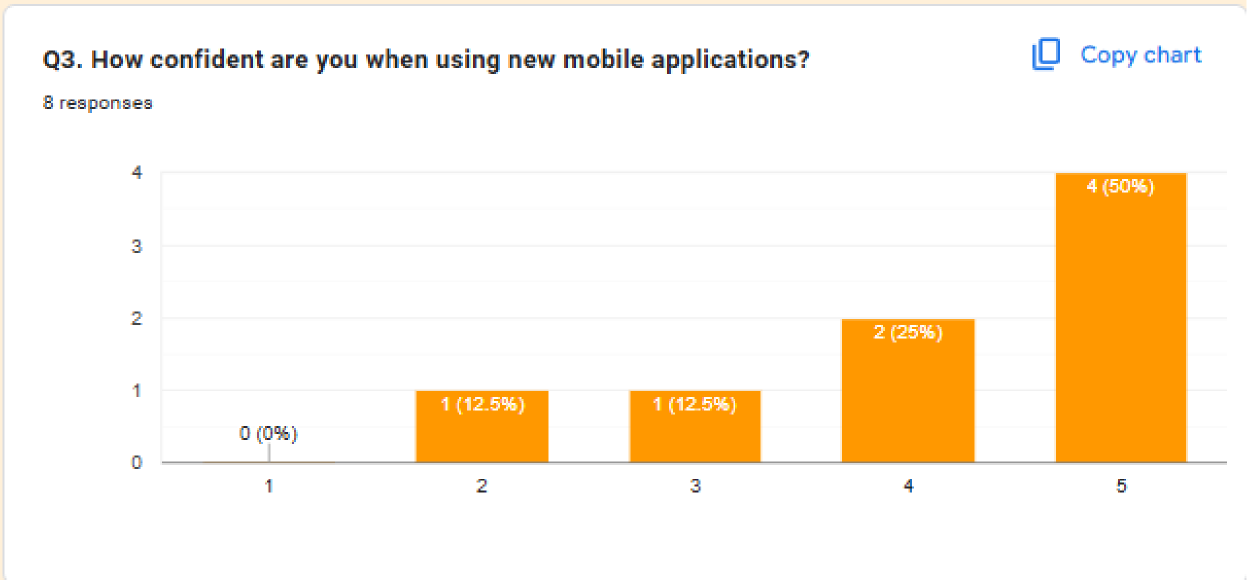
Section 1: Participant Background

Q1. What best describes your role when using this application?

8 responses

Q2. How often do you normally use mobile applications for food ordering or reservations?

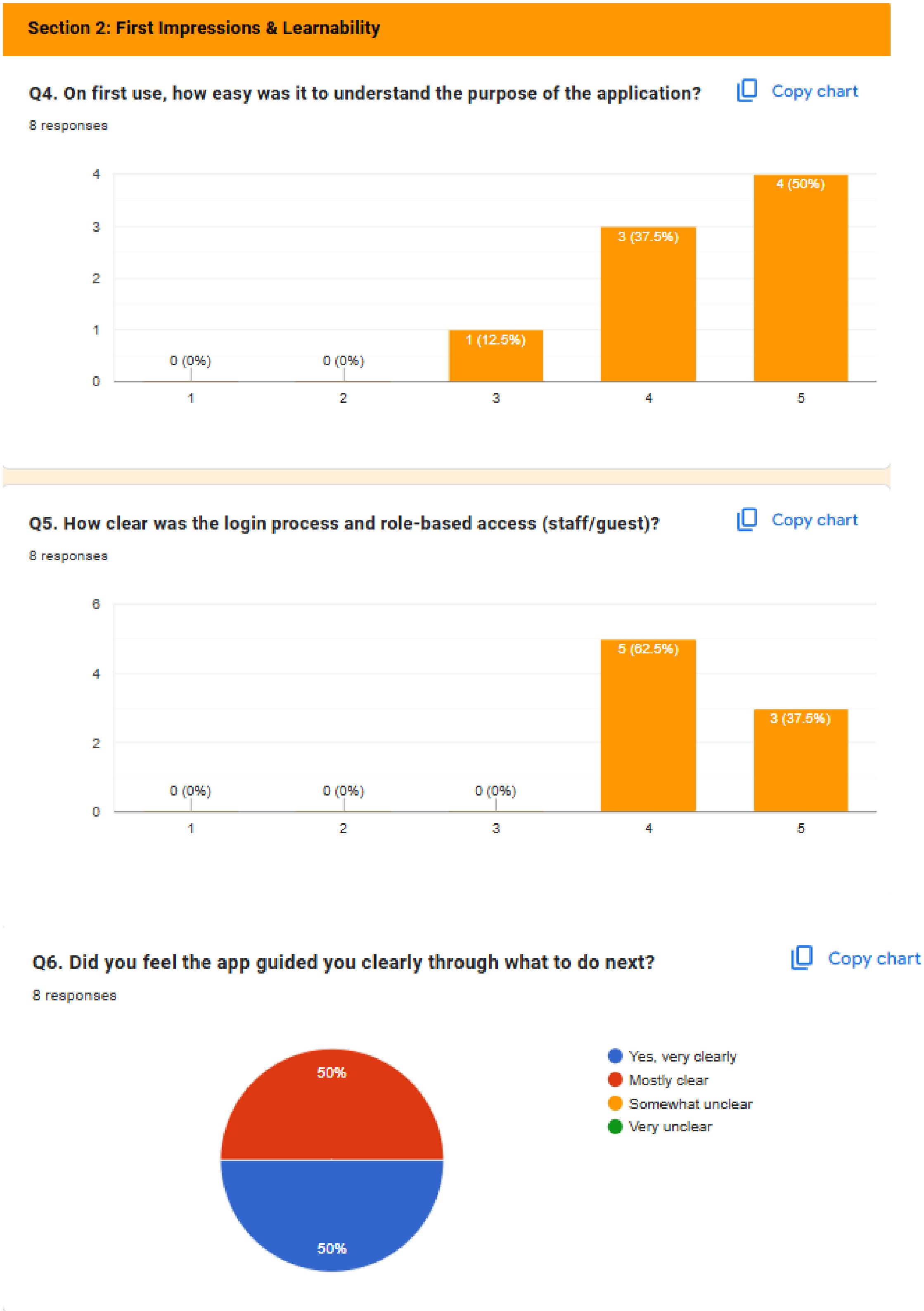
8 responses



Q3. How confident are you when using new mobile applications?

8 responses

Figure 28 : Section 1 of Usability Questionnaire



Section 2: First Impressions & Learnability

Q4. On first use, how easy was it to understand the purpose of the application?

8 responses

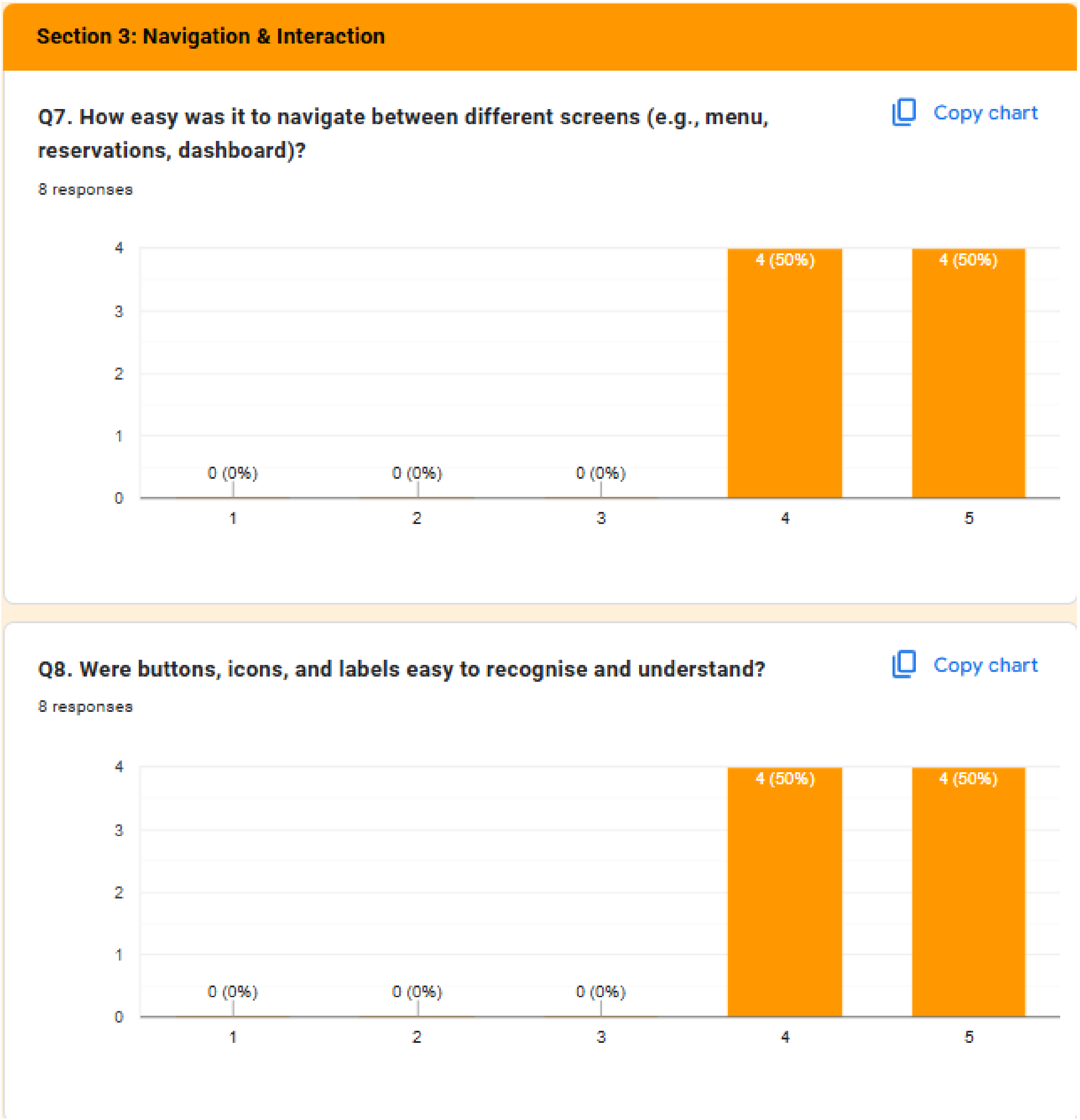
Q5. How clear was the login process and role-based access (staff/guest)?

8 responses

Q6. Did you feel the app guided you clearly through what to do next?

8 responses

Figure 29 : Section 2 of Usability Questionnaire



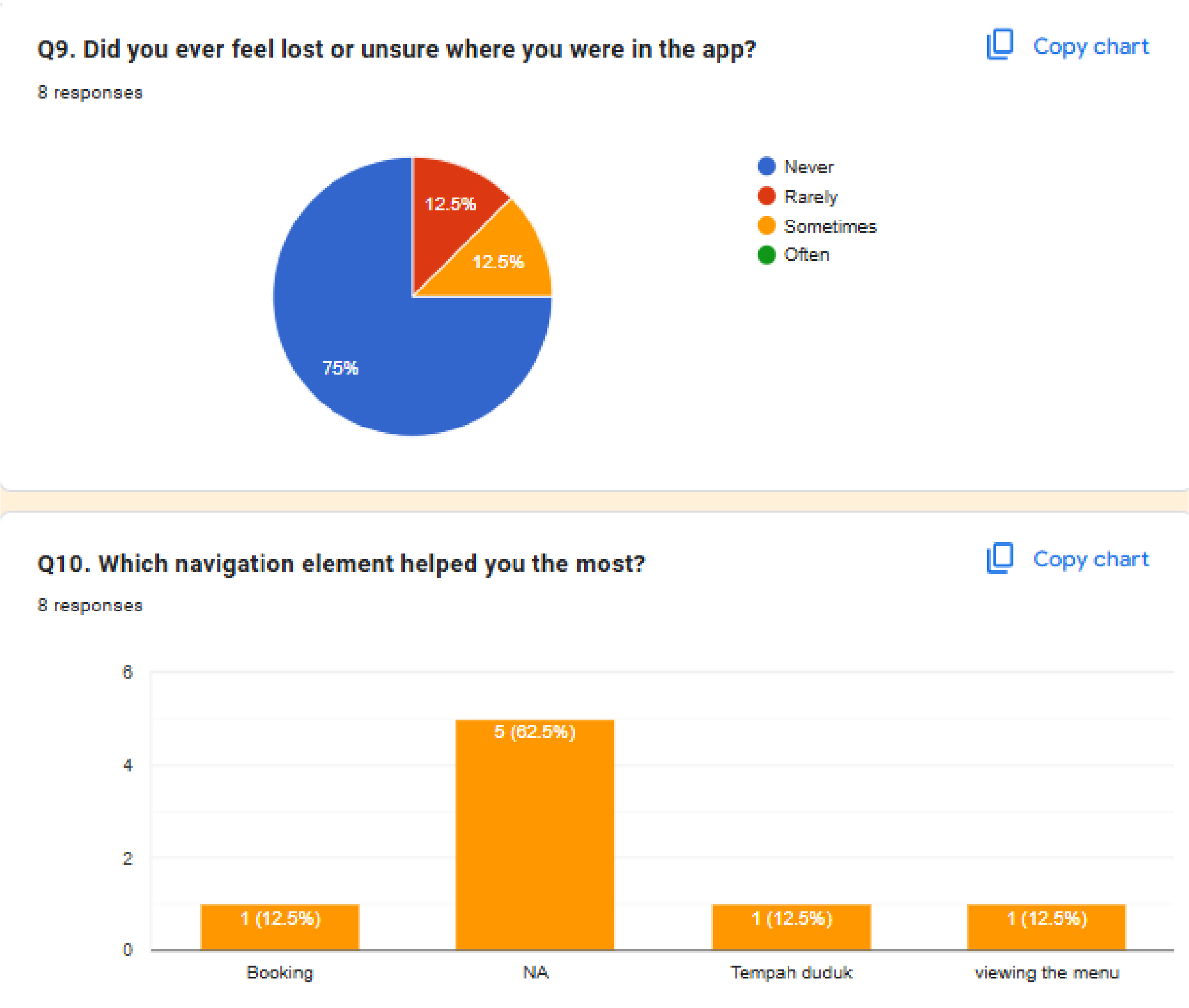
Section 3: Navigation & Interaction

Q7. How easy was it to navigate between different screens (e.g., menu, reservations, dashboard)?

8 responses

Q8. Were buttons, icons, and labels easy to recognise and understand?

8 responses



Q9. Did you ever feel lost or unsure where you were in the app?

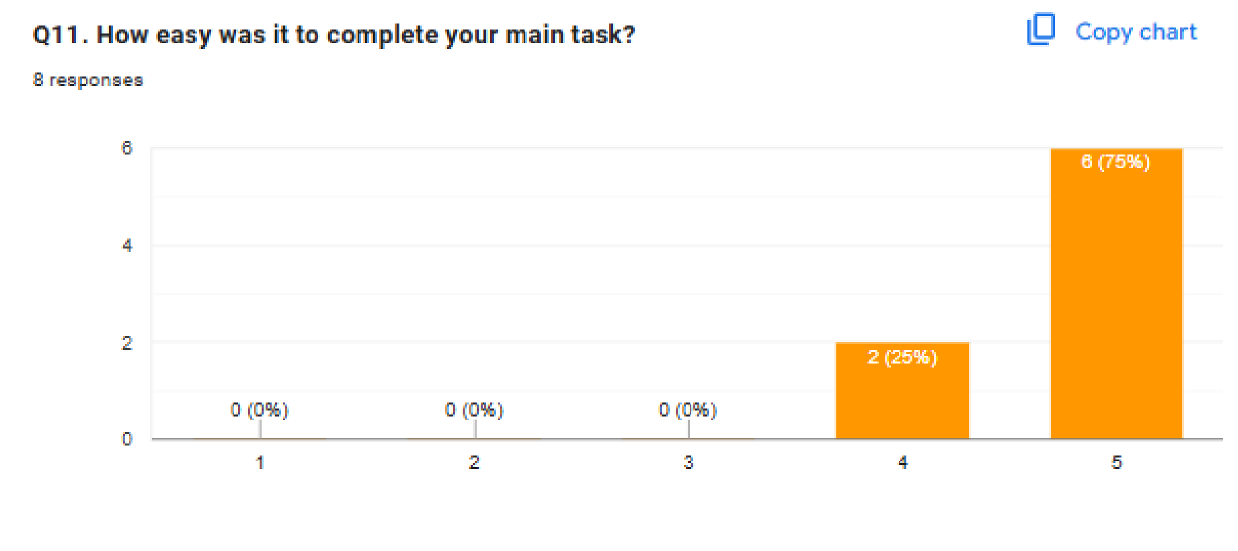
8 responses

Q10. Which navigation element helped you the most?

8 responses

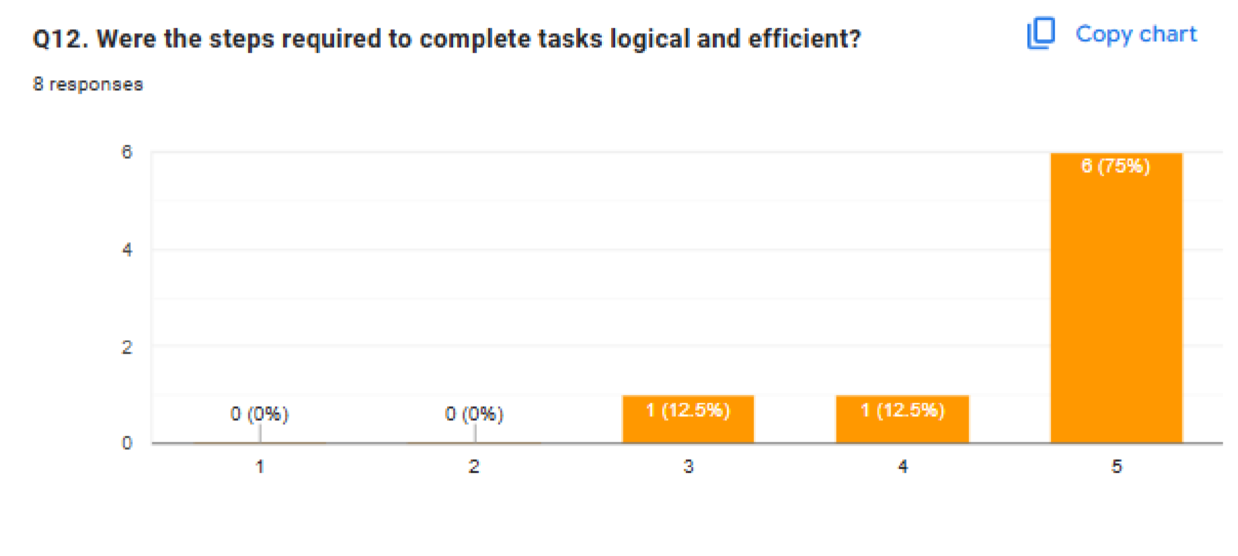
Figure 30 : Section 3 of Usability Questionnaire

Section 4: Task Effectiveness (Role-Specific)



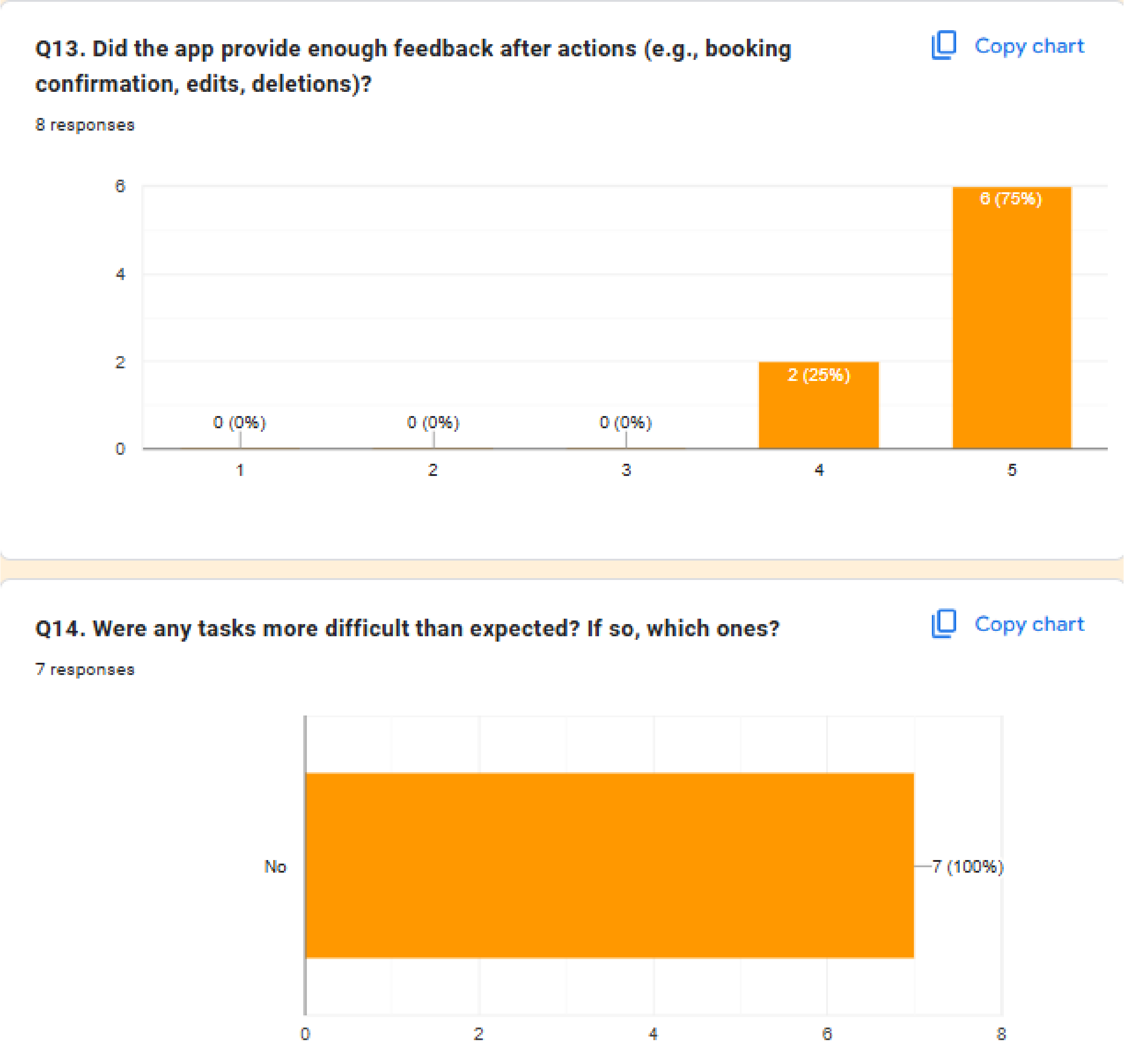
Q11. How easy was it to complete your main task?

8 responses



Q12. Were the steps required to complete tasks logical and efficient?

8 responses



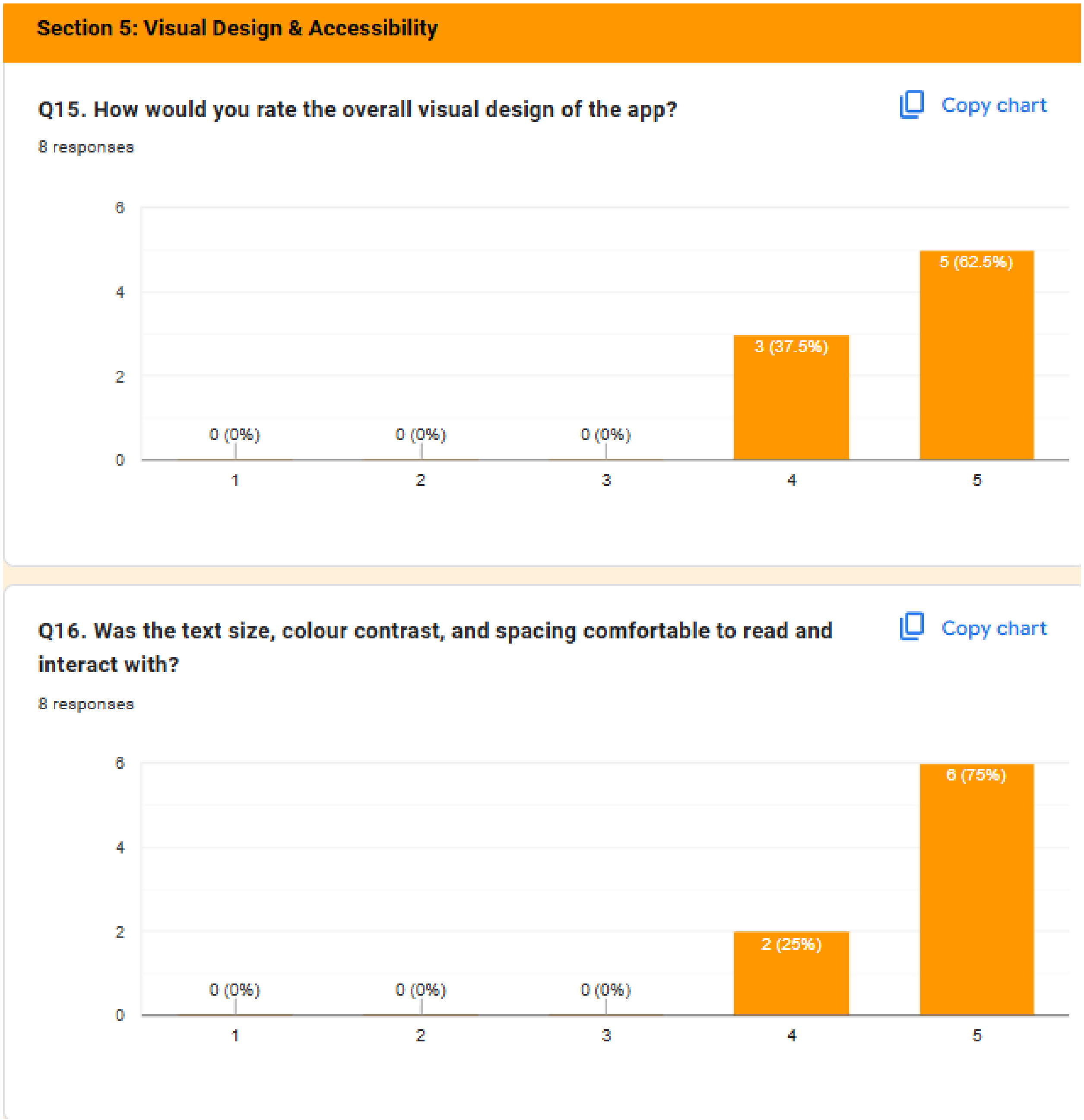
Q13. Did the app provide enough feedback after actions (e.g., booking confirmation, edits, deletions)?

8 responses

Q14. Were any tasks more difficult than expected? If so, which ones?

7 responses

Figure 31 : Section 4 of Usability Questionnaire



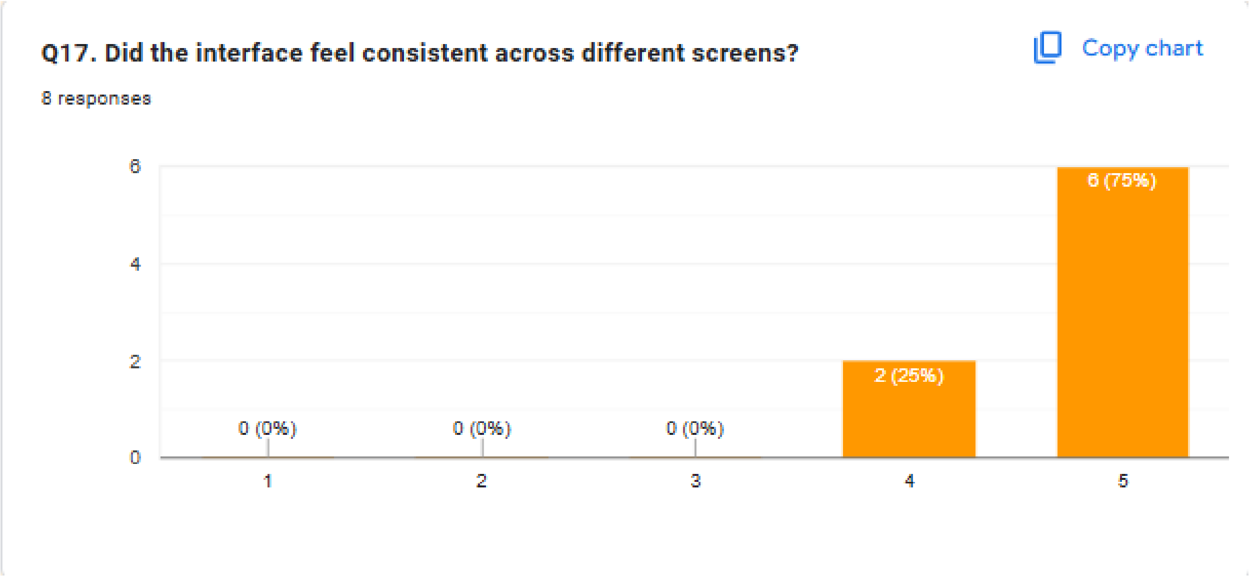
Section 5: Visual Design & Accessibility

Q15. How would you rate the overall visual design of the app?

8 responses

Q16. Was the text size, colour contrast, and spacing comfortable to read and interact with?

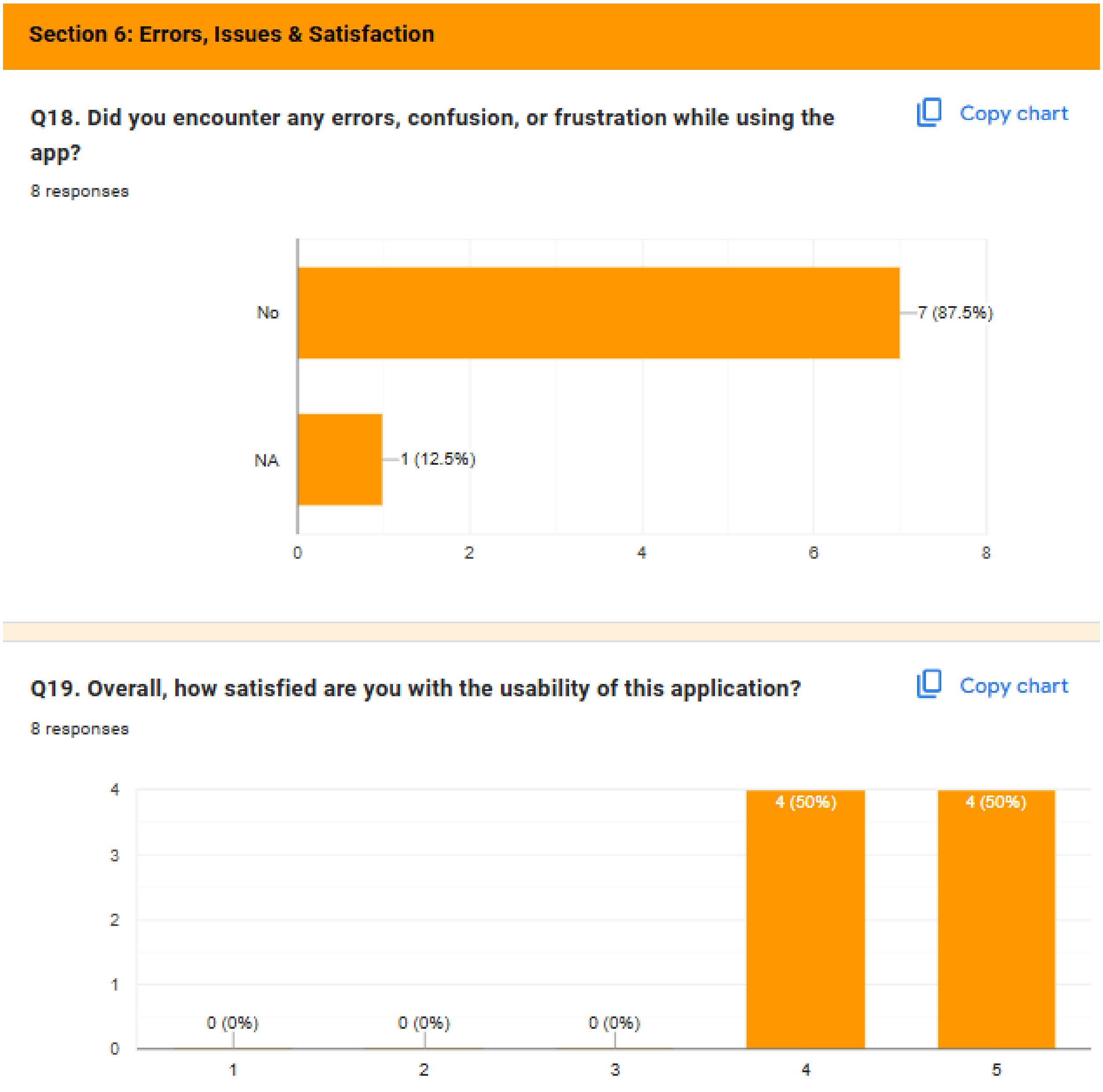
8 responses



Q17. Did the interface feel consistent across different screens?

8 responses

Figure 32 : Section 5 of Usability Questionnaire



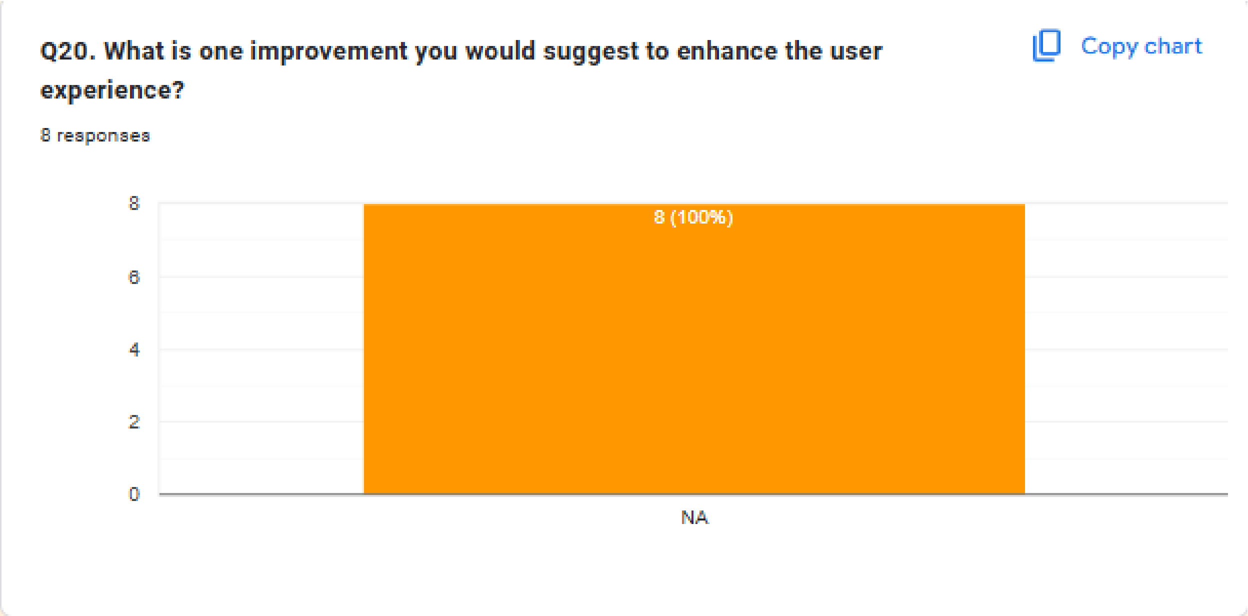
Section 6: Errors, Issues & Satisfaction

Q18. Did you encounter any errors, confusion, or frustration while using the app?

8 responses

Q19. Overall, how satisfied are you with the usability of this application?

8 responses



Q20. What is one improvement you would suggest to enhance the user experience?

8 responses

Figure 33 : Section 6 of Usability Questionnaire

6.5 Key Issues Identified

The main concerns reported were:

i. Desire for clearer success animations or stronger confirmation cues

ii. Occasional perception of tight spacing on smaller screen devices

iii. Suggestion for sorting filters on reservation lists

7.0 Legal, Social, Ethical, and Professional Considerations

The project adheres to core responsibilities expected from software professionals.

7.1 Legal Compliance

No personal identifiers were collected. Data collection and storage follow principles aligned with the Malaysian Personal Data Protection Act (PDPA) to ensure user anonymity and informed consent.

7.2 Ethical Considerations

Participants were informed about the study objectives, their rights, data usage and the voluntary nature of participation. They could withdraw at any point by exiting the form. No deceptive methods or invasive data collection were used.

7.3 Social Responsibility

The app is designed to support accessibility through readable fonts, adequate contrast ratios and simplified interactions. Inclusive design improves usability for diverse users also with those less familiar with technology.

7.4 Professional Practice

The development process applies standard coding structures, version control via GitHub and Android development best practices. Design decisions were documented clearly to support maintainability and collaborative development.

8.0 Conclusion

This CW2 demonstrates a complete development of a high-fidelity restaurant management mobile application guided by HCI principles and validated through usability testing. The interface evolved significantly from CW1 sketches to a polished Android UI by inform real user feedback and grounded in evidence-based design practices.

Nonetheless, the final system provides an efficient workflow for staff and a user-friendly booking process for guests. While certain refinements such as enhanced feedback animations and additional filtering tools could further improve the experience, the implemented design forms a strong foundation for future functional integration.

9.0 References

Android Developers (2021) Android developer documentation. Available at: https://developer.android.com (Accessed: 18 December 2025).

Google (2021) Material Design Guidelines. Available at: https://material.io/design (Accessed: 18 December 2025).

Norman, D.A. (2013) The Design of Everyday Things. Revised and expanded edition. New York: Basic Books.

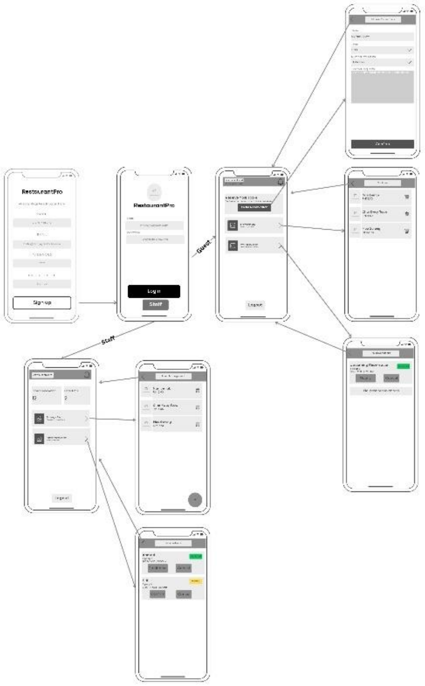
Preece, J., Rogers, Y. and Sharp, H. (2015) Interaction Design: Beyond Human–Computer Interaction. 4th edn. Chichester: Wiley.

Pressman, R.S. and Maxim, B.R. (2020) Software Engineering: A Practitioner's Approach. 9th edn. New York: McGraw-Hill.

Sommerville, I. (2016) Software Engineering. 10th edn. Harlow: Pearson Education.

10.0 Appendices

Appendix A : Low-Fidelity Wireframe



Picture 1 : Low fidelity of RestaurantPro