Table of Contents

Individual Tasks Allocation **3**
Executive Summary 9
1.0 Introduction 10
2.0. Requirement and Planning 11
2.1. Planning Decisions 11
2.2. System Requirements 12
Functional Requirements 12
Non-Functional Requirements 13
2.3 user stories 15
2.4 Feature Prioritization 16
3.0 System Design 17
3.1. Application Architecture 17
4.0 Development Process 19
4.1 Saad's Contribution 20
Tools and Development Environment 20
Architecture and Integration 21
Key Implementation – Breathing 21
Key Implementation – Journal 22
Key Implementation – Meditation 23
Integration of libraries and APIs 23
Version Control 24
4.2. Saimon's Contribution 25
Architecture and Integration 26
Key Implementation – Aura AI Chat Companion 27
AI Network Layer 28
API Communication 29
Dynamic Message Interface 31
Fallback Response System 31
Key Implementation – Patterns and Insights 32
Reusable Pattern Details Screen 34
Integration of libraries and APIs 35
Version Control 37
4.3. Juri's Contribution 38
Development Tools And Environment 38
Authentication Flow implementation 41
Profile Screen Implementation 42

Navigation and Experience Improvements 43
Version Control 44
4.4 Bader's Contribution 45
Overview of Contribution 45
Tools and Development Environment 45
Architecture and Integration 45
Key Implementation-Mood Tracker 47
Key Implementation-Sleep Logging 49
5.0. Use of AI 50
5.1. Saad's Contribution 50
5.2. Saimon's Contribution 51
5.3. Juri's Contribution 52
5.4. Bader's Contribution 53
6.0. Testing and Evaluation 54
6.1. Saad's Contribution 54
Testing methods 55
Debugging and Bug Fixing 57
Testing Evidences 58
Evaluation 60
6.2. Saimon's Contribution 61
Debugging and Bug Fixing 64
Evaluation 64
6.3. Juri's Contribution 66
Testing Approaches 66
Testing Table 67
Testing Evidence 70
Debugging and Bug Fixing 72
Input Validation 72
Authentication and Sessions 72
Navigation and Manifest Configuration 73
Profile Data Display 73
Prototype Screens' Behaviour 73
Emulator Performance 73
Evaluation 74
6.4. Bader's Contribution 75
Testing Methods 75
Testing Table 75
Debugging and Bug Fixing 76

Evaluation 76
7.0. Challenges and solution 78
7.1. Saad's Contribution 78
Fitting New Screens 78
Sourcing real, appropriate licensed assets 78
Making Prompt Rotation Work for Journal Entry 79
Managing Timer, Animation, Audio 79
Build Error 80
Android Studio Emulator Launch Crash 81
7.2. Saimon's Contribution 82
Making Network Requests without Blocking the Interface 82
API Failure and Disconnection from the Internet 82
Prevention of Multiple Requests 82
Feedback during Waiting Time 82
Handling Seven Pattern Categories 83
Designing Empty States with Insufficient Data 83
Consistency with the Existing Codebase 83
7.3. Juri's Contribution 84
Activity Navigation and Integration 84
Input Validation 85
User Interface Consistency 86
Version Control and Team Integration 86
Emulator and Laptop Performance Challenge 87
7.4. Bader's Contribution 89
Passing Data Between Mood Tracker and Mood Detail 89
Keeping Dashboard Information Updated 90
Maintaining a Consistent User Interface 90
8.0. Reflection and Future Work 91
8.1. Saad's Contribution 91
What was Learned 91
Skill Gained 92
Future Improvements 92
8.2. Saimon's Contribution 94
Reflection 94
Limitations 95
Future Work 95
8.3. Juri's Contribution 97
What was Learned 97

Skill Gained 97
Future Improvements 98
8.4. Bader's Contribution 99
Reflect on the Development Experience 99
Future Improvements 100
9.0 Conclusion 101
References 102
Appendix A 104
GitHub Link : 104
Saad's Contribution Appendix 104
Saimon Contribution Appendix 107
Juri Contribution Appendix 109
Bader Contribution Appendix 113

Executive Summary

Mental Health is a challenge in the form of Stress, poor sleep, and anxiety among university students, yet many lack easy and low barrier access to support tools that fit their daily life.

CalmU addresses this gap as an Android Mental Wellness Application, built in java using Android Studio which offers students a lightweight daily companion for tracking their mood, building small wellness habits and receiving supportive check in without needing to book a real appointment. This project was motivated by the need for an accessible,available, and judgment free place for self care with the objective of delivering a consistent and reliable experience that remains functional even without the internet.

CalmU brings five Core areas of functionality for the users: Secure user Authentication and Profile, “Today” Dashboard for mood and sleep log, AI Chat Companionion “Aura”, a “Wellness Hub” with multiple fun exercises, and a “Pattern Dashboard” for reflecting usage habits back to the user, all rounded by gamification through Achievements.

By encouraging consistent self reflection and healthy habits, CalmU contributes to Sustainable Development Goal 3 (Good Health and Well Being).

1.0 Introduction

CalmU is a native Android mental wellness application built for this project in Java using Android Studio. The app is aimed at students who want a lightweight daily companion for tracking mood, building small wellness habits, and getting a supportive check-in without needing to book a real appointment. It brings together five main areas of functionality: user authentication and profile management, a daily "Today" dashboard with mood and sleep logging, an AI chat companion named Aura, a set of guided wellness exercises (breathing, journaling, and meditation), and a pattern-insights section that reflects usage back to the user. Achievements and Challenges screens round the app out with a light layer of gamification.

This section of the report covers the requirements, system design, development process, testing approach, challenges, and reflection for the project as a whole, based on the final CalmU codebase.

2.0. Requirement and Planning

In this part, the criteria and decisions involved in designing the CalmU prototype are described. They were revised based on the original proposal of the project and in line with the time and technological constraints, as well as the architecture of the designed app.

The first and foremost purpose of the work was to build an Android prototype that provides basic mental health capabilities while remaining easy-to-navigate, consistent and having clear separation of functions performed by each of the developers.

2.1. Planning Decisions

From the start of the project, it was decided that the same technical framework would be employed to simplify the integration procedure. CalmU was programmed natively on Android using the Java programming language along with the XML format for the interfaces.

The choice was made to utilize the Activity-per-Screen approach since each major functionality needed its screen. The navigation through the application was done with Explicit Intents between the Activities including the Login, Today, Wellness, Patterns, AI Chat, and Profile screens.

It was decided not to use an additional Firebase project or backend service but only to use local storage. SQLite was applied for those pieces of information that need to be structured and saved persistently like journal entries. SharedPreferences were used for lightweight information like login status, user settings, or prototype information.

The source code was managed using GitHub. Before the completion of individual features, each team member worked separately within different branches. This approach helped minimize any chances of file conflicts. The testing of each feature was conducted before merging it into the project.

Some advanced functionalities have been intentionally left for prototype-level development. For instance, password recovery, achievements, challenges, and certain data-based functionalities show the flow without implementing the back end or any real data from users.

2.2. System Requirements

The system requirements list out the functionality and the performance level expected from the CalmU app. These requirements have been determined by considering the objectives of the project and the needs of the university students along with the functionalities planned for the prototype. The system requirements can be categorized into functional and non-functional requirements. The former is used to specify the function performed by the system whereas the latter explains the performance of the system.

Functional Requirements

ID	Functional Requirements
FR-01	The system shall enable the creation of a new user account based on the input of name, email, password, and its verification.
FR-02	The system shall enable login and subsequent access to the app for the existing user.
FR-03	The system shall enable displaying a screen with the Forgot Password function, which verifies the email inputted and shows the password recovery process.
FR-04	The system shall enable showing a Today dashboard with the user's daily information and activities related to wellness.

FR-05	The system shall enable users to log in basic mood and sleep data using the Today section.
FR-06	The system shall enable users to access wellness activities including breathing, journaling, and meditation.
FR-07	The system shall enable interaction with the Aura AI Chat Companion and provide an appropriate reply or fall-back message to the user.
FR-08	The system shall enable displaying the Patterns and Insights section with the summaries of user's activity and wellness information.
FR-09	The system shall enable accessing the user Profile section and Settings screen.
FR-10	The system shall enable displaying the Achievements and Challenges screens as the example of planned progress and gamification.
FR-11	The system should permit users to move between the main parts via buttons, hyperlinks, back button, and bottom navigation.
FR-12	The system should keep the required data for prototyping locally using SQLite or SharedPreferences.

Table 1 : Functional Requirements

Non-Functional Requirements

ID	Non-Functional Requirement
NFR-01	Usability: The system must provide clear labeling, readable text and simple navigation tailored to the needs of university students.
NFR-02	Consistency: The screens must have consistent colors, buttons, spacing, icons and navigation methods.
NFR-03	Reliability: The main screens and navigation functions must work flawlessly without unexpected crashes in regular usage.
NFR-04	Performance: The system must load screens and handle user inputs promptly and without unnecessary delay on the specified emulator.
NFR-05	Compatibility: The prototype must perform properly on the selected Android virtual devices and be adapted to common mobile phone screen sizes.
NFR-06	Maintainability: The Java Activities and XML layouts must be well-organized in order to make the understanding of assigned features easy by the team members.

NFR-07	Privacy and Security: Only test data must be used within the prototype. No real passwords, private journal data or personal user data must be inputted into the system during testing.
NFR-08	Lightweight Usage: The prototype features must work properly without using the full Firebase backend services.

Table 2 : Non-functional requirements

2.3 user stories

These user stories were formulated in order to explain the basic functionalities of the CalmU application through the eyes of its user. They served as a tool for identifying the needs of the user, defining the expected system behaviour and prioritizing the necessary functionalities for development.

Table 3: User Story—User Login

Title: User Login	**Priority: Must Have**	**Estimate: 3 Story Points**
User Story: As a university student, I want to log in using my email and password so that I can access my personal well-being dashboard and CalmU features.		
Acceptance Criteria: **Given** the user is on the Login screen. **When** valid login credentials are entered and the **Login** button is selected. **Then** the main application authentication the user and opens the main dashboard successfully.		

Table 4: User Story—Mood Tracking

Title: Mood Tracking	**Priority: Must Have**	**Estimate: 4 Story Points**
User Story: As a university student, I want to record my daily mood so that I can monitor my emotional well-being over time.		
Acceptance Criteria:		

Given the user is on the Today screen.

When a mood is selected and saved,

Then the application records and displays the latest mood successfully.

Table 5: User Story—View Profile

Title: View Profile	Priority: Should Have	Estimate: 2 Story Points
User Story: As a user, I want to view my profile information so that I can check my progress, achievements, and account details.		
Acceptance Criteria: **Given** the user selects the Profile page, **When** the page is opened, **Then** the application displays the user's profile information and related sections correctly		

The user stories helped in clearly identifying the basic functionalities of the CalmU application through the eyes of its user. They helped in defining the expected behaviour of the system and in planning the development of the CalmU prototype.

2.4 Feature Prioritization

Priority	Feature	Reason
Must Have	Login, Sign Up, Forhot Passwored, Today Dashboard, Main Navigation, Wellness Hub, and Profile	This functionality makes up the basic application flow and is necessary in order to showcase the main goal of CalmU.
Should Have	Mood check-in, sleep logging, journaling, AI Chat Compantion, Patterns and Insights, and Settings	This functionality allows users to achieve main wellness goals and provides personalisation but requires completion of the basic navigation first.

Could Have	Achievements, challenges, advanced notifications, editable profile information, and expanded progress reports	This functionality improves engagement and personalisation and is not necessary to showcase the basic prototype.
Not Included in the Current Scope	Production cloud backend, real email password recovery, full firebase authentication, iOS version, and clinical diagnosis functions	The functionality requires extra time, security measures, external services, or out-of-scope development.

Table 6 : features

3.0 System Design

3.1. Application Architecture

CalmU implements a straightforward Activity + Explicit Intent-based architecture in its entirety: All screens are independent AppCompanions with their own layout files, and one screen starts another through startActivity(new Intent(...))

Data Layer

- DatabaseHelper (SQLiteOpenHelper, calmu.db, schema v2) maintains two tables: users (id, name, email (UNIQUE, case-insensitive), SHA-256 password_hash, created_at) and journal_entries (id, user_email, mood, content, created_at). Provides typed helper functions like createUser, isValidLogin, getUserByEmail, insertJournalEntry, getJournalEntryCount, getAverageMood, and getRecentJournalEntries.
- SessionManager is a wrapper around a SharedPreferences file (calmu_session), which keeps the logged-in status and current email of the user, so the login state persists across app launches.
- TodayPrototypeStore is a wrapper around another SharedPreferences file (calmu_today_prototype), which keeps the app-specific information of the Today Dashboard and the Mood Tracker feature – last logged mood (emoji, label, category), sleep information and pending/skipped/completed status of each Today focus task.

In **AndroidManifest.xml**, there are only two Activities that have their "exported" attribute set to "true": LoginActivity (which is the launcher activity and hence has to be exported) and TodayActivity (explicitly made exportable so that it can be started directly from Android Studio for debugging purposes, as described in comments within the code).

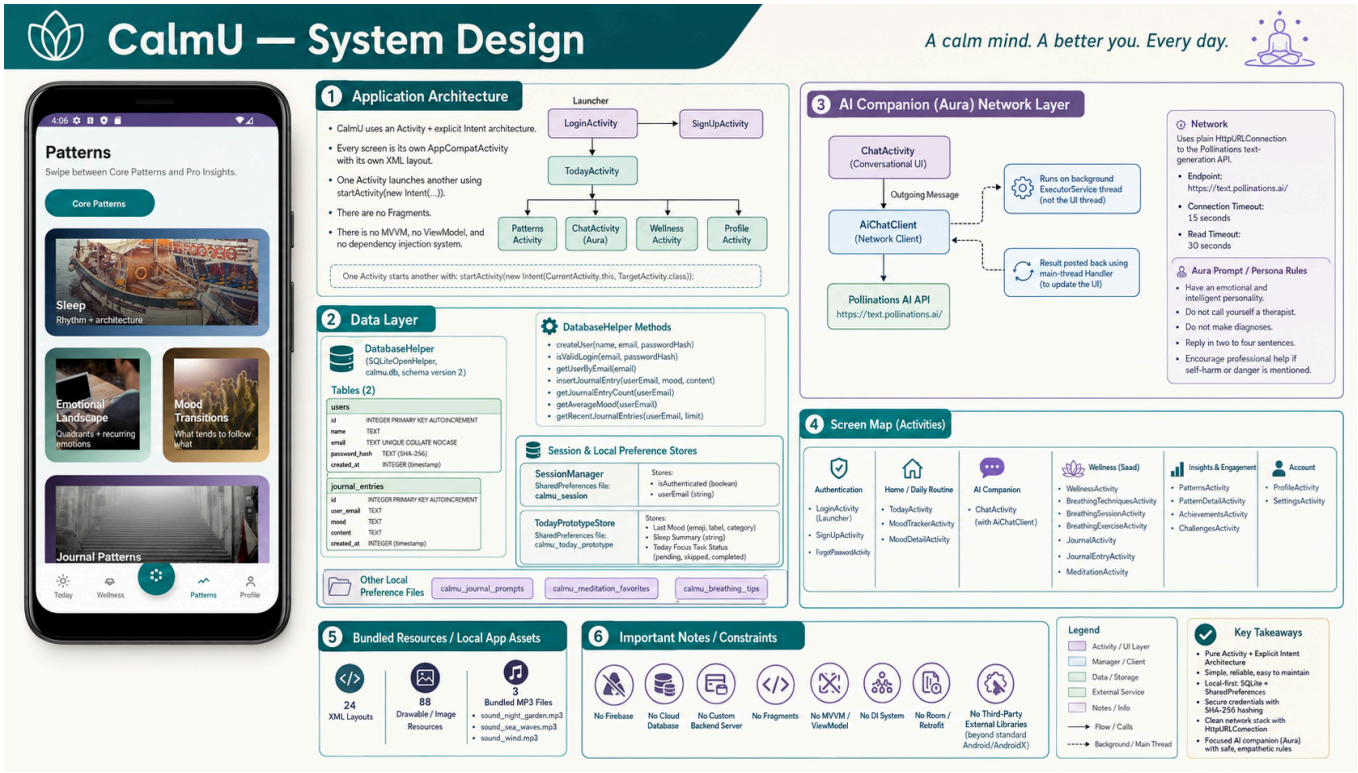


Figure 1: Architecture Diagram

4.0 Development Process

CalmU was developed collaboratively in android studio using Java, with the team splitting the application into distinct feature areas which are: Authentication and profile, AI chat companion (named AURA), wellness exercises, the recorded pattern insights, and a Today page. Each owned and implemented by a different member. All features were built on top of the same shared foundation (a common SQLite database, session handling, and screen structure) to keep the app consistent, with the work coordinated through Git and merged from individual branches into a shared repository. The following subsections detail each member's individual development process.

4.1 Saad's Contribution

My contribution to CalmU application was the Wellness feature: its a hub screen linking to three fully functional exercises which are Breathing, Journal, and Meditation. They are reachable from the app bottom navigation. This included designing and building six new screens, also extending the SQLite database to support real journal entries, and integrating androids media and animation APIs to support a guided breathing timer and ambient audio playback. The subsections below are in detail about the tools used, the architecture followed, and the key implementation work for each of the three exercises.

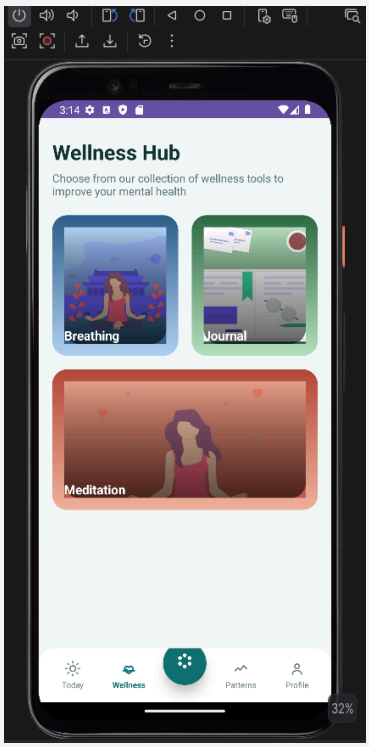


Figure 2 : Wellness Hub

Tools and Development Environment

The wellness feature was developed in android studio using java, targeting the existing projects java SDK based architecture (no kotlin, no jetpack compose – consistent with the rest of the codebase). We handled and recorded version control using Git, with all work committed to the

saimon/all-page branch on our shared github repository. The feature was built iteratively, screen by screen: each screen such as (wellness hub, then breathing, then journal, then meditation) was implemented and then it was all manually tested through a debug build (./gradlewassembleDebug) and was also tested by running on an emulator inside Android Studio before the next screen was started, so every stage was verified working before further code was added on top of it.

Architecture and Integration

This application has no fragments and no dependency injection framework; each screen is a self contained Activity with its own XML layout, and navigation between screens uses explicit Intents. The wellness feature follows that same pattern to stay consistent with the rest of the codebase, instead of introducing a different architecture (e.g. MVVM or a navigation graph) just for one feature. Seven new activities were added which are: WellnessActivity (the hub), BreathingTechniquesActivity and BreathingSessionActivity, JournalActivity and JournalEntryActivity, and MeditationActivity, plus some supporting classes were added such as (BreathingTechnique, Meditation, JournalMood, CircularProgressView).

Key Implementation – Breathing

Breathing exercise is driven by small data model BreathingTechnique.java describing three techniques which are 4-7-8 breathing, box breathing and deep belly breathing each with its own inhale/hold/exhale button. BreathingSessionActivity is the session screen that runs a real timer in the app using Handler.postDelayed, ticking once per second. it tracks which phase the user is in, and how many seconds into that phase, and also the total elapsed time against the selected session length which can be anytime between 1 to 10 minutes. Breathing circle inside exercises itself is animated with ViewPropertyAnimator, it scales up for inhaling and will go back down for exhale and will hold itself for hold instruction. There is also a first-time "tip" dialog per technique which is shown once and can be dismissed permanently via a "Don't show again" checkbox it is put through SharedPreferences. (see figure A.1 in Appendix A)

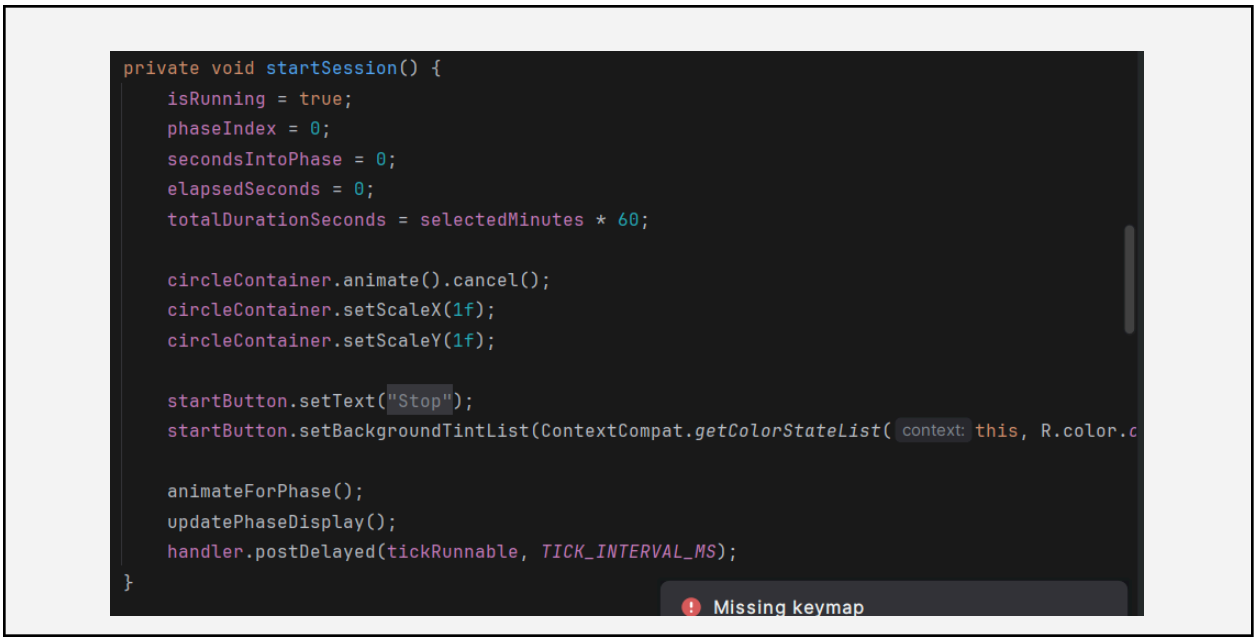
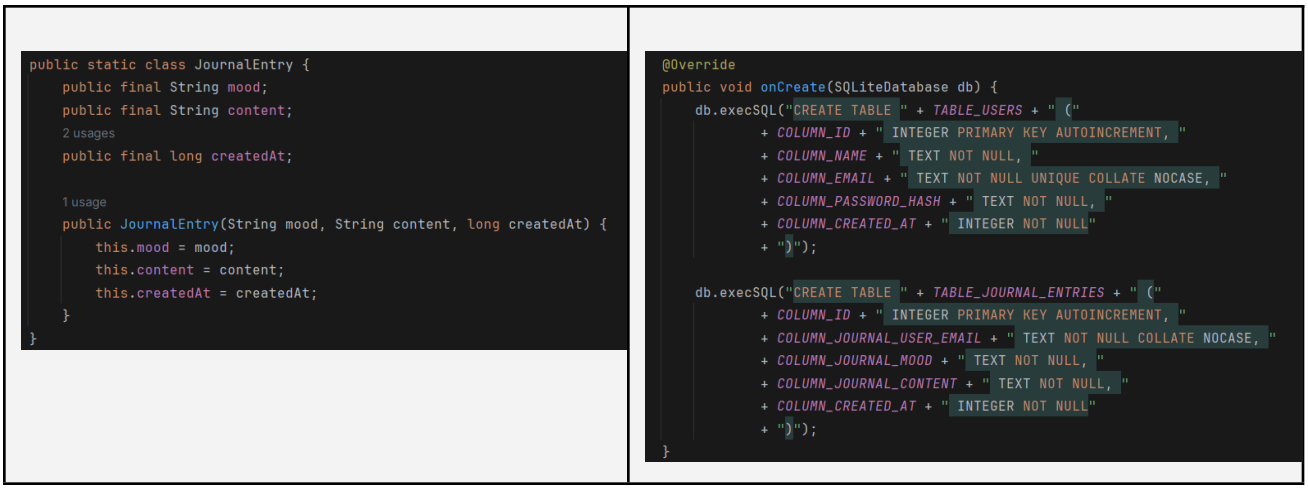


Figure 3: Breathing session Code

Key Implementation – Journal

Implementation of journal requires a real data layer, because it needs to track real journal entries instead of just displaying an entry as a mockup. I added journal_entries table to the existing SQLite database (DatabaseHelper), and database version was also incremented to trigger schema migration. Journal entries were added to the app so that the user can be able to store the mood, written content, and timestamp, and these entries are queried back to compute a running entry count, a rounded average mood, and also the most recent entries for the history list. There will be a free tier limit of total 5 entries to the user. This limit is actually imposed not just cosmetic numbers on UI. journal entry screen will rotate through five original journaling prompts which are personalized with user name and guide the user to write an entry example of a guiding prompt is *"What is one small thing you did recently that you're proud of, and how can you carry that same energy into the days ahead, Nisar?"*. Prompt will be new each time a user accesses the entry screen. This is all tracked through SharedPreferences so the rotation persists across sessions and there is also a "shuffle" button that lets the user manually cycle prompts, capped at two shuffles per visit. (see figure A.2 in Appendix A)



```
public static class JournalEntry {
    public final String mood;
    public final String content;
    2 usages
    public final long createdAt;

    1 usage
    public JournalEntry(String mood, String content, long createdAt) {
        this.mood = mood;
        this.content = content;
        this.createdAt = createdAt;
    }
}
```

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_USERS + " ("
            + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + COLUMN_NAME + " TEXT NOT NULL, "
            + COLUMN_EMAIL + " TEXT NOT NULL UNIQUE COLLATE NOCASE, "
            + COLUMN_PASSWORD_HASH + " TEXT NOT NULL, "
            + COLUMN_CREATED_AT + " INTEGER NOT NULL"
            + ")");

    db.execSQL("CREATE TABLE " + TABLE_JOURNAL_ENTRIES + " ("
            + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + COLUMN_JOURNAL_USER_EMAIL + " TEXT NOT NULL COLLATE NOCASE, "
            + COLUMN_JOURNAL_MOOD + " TEXT NOT NULL, "
            + COLUMN_JOURNAL_CONTENT + " TEXT NOT NULL, "
            + COLUMN_CREATED_AT + " INTEGER NOT NULL"
            + ")");
}
```

Figure 4 : Code Snippet Journal Entry in Database Helper

Key Implementation – Meditation

Meditation screen inside our application presents six tracks across three categories which are: (Sleep, Anxiety, Relax), each of them is represented by a small Meditation data model. Three of the six tracks play a real ambient audio file bundled in the app's res/raw resources, using Android's MediaPlayer API; the other three run a silent countdown only. The Playback option is on the same card. Users can tap to play starts (or, for silent tracks, simulates) a one-minute countdown using the same Handler based ticking approach used in the breathing timer. Card countdown duration text will be updated live. Only one track can play at a time; starting a new one will stop whatever was already playing. playback is stopped in onPause() so audio never continues after the user leaves the screen. There is also a favorite tab implemented which is also maintained using SharedPreferences, users can choose to favorite a sound. (see figure A.3 in Appendix A)

Integration of libraries and APIs

Wellness feature only uses androids own APIs such as MediaPlayer for audio playback, Handler for timing, ViewPropertyAnimator for breathing animation, SQLiteOpenHelper for persistence, and SharedPreferences to maintain state. Material Components library like (MaterialButton) already used elsewhere in the project. There is no use of any third-party dependencies or external network APIs added for this feature; it does not call the internet at all, unlike the Chat feature elsewhere in the app.

Version Control

All of the work inside wellness hub feature was committed as a single commit on saimon/all-page-Updated-by-Saad after the full feature (Hub, breathing, journal, meditation) had been built and manually verified end to end. After that it was pushed to the shared Github repository.

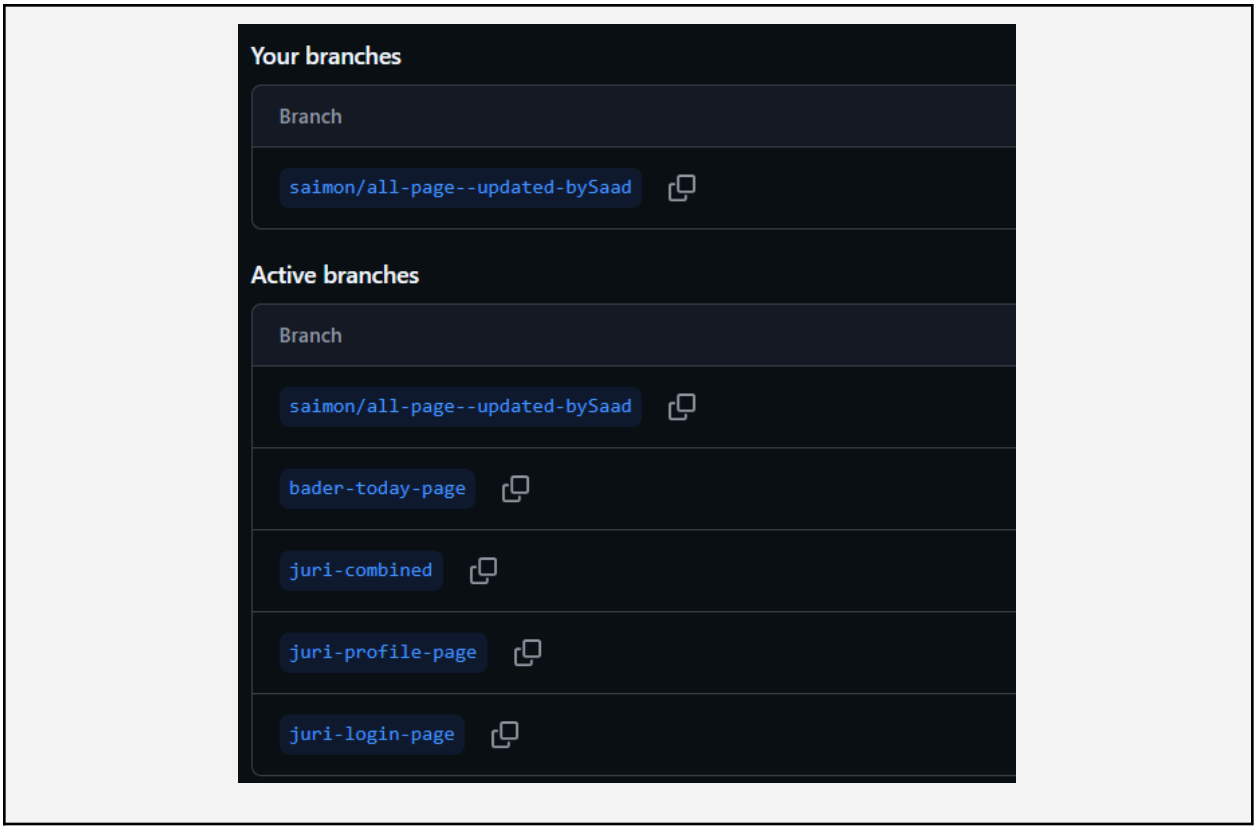


Figure 5 : Github Version Control Saad

4.2. Saimon's Contribution

The Aura AI Chat Companion and Patterns features were built on Android Studio using Java 11 programming language for application logic and XML for layout interfaces. The application had its minimum SDK version set at 24 and followed the existing AndroidX AppCompatActivity framework. The development process of the project was done in an incremental fashion. Firstly, the interface was created and connected to simple message bubble logic. When the interface worked, the network client was added and connected to Pollinations text generation API. Then error handling and fallback messages were added. As far as the Patterns feature is concerned, initially the category screen was created and one reusable details screen which displayed different content depending on category selection. The features were being tested at various stages of development using Android Studio emulator. Git version control system was used for this project and the work was done on Saimon-All-Page branch.

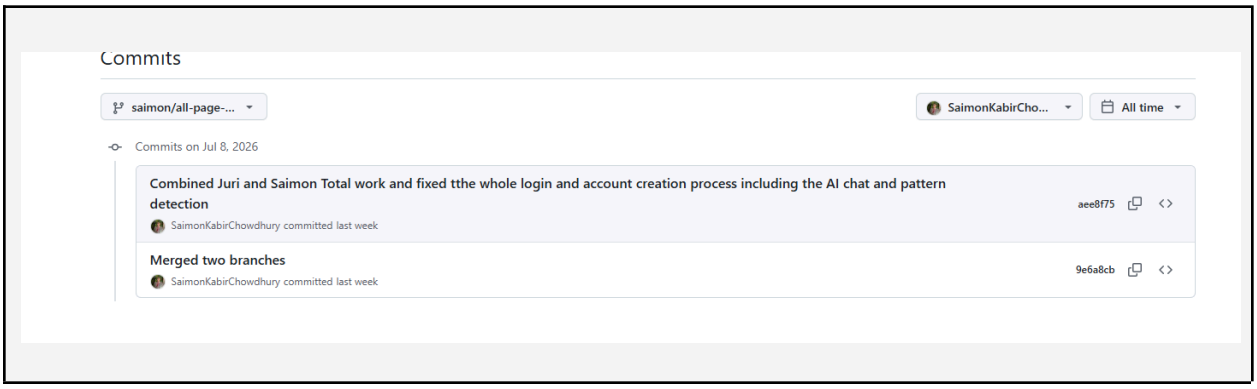


Figure 6 : Git branch and version-control evidence for Saimon's contribution

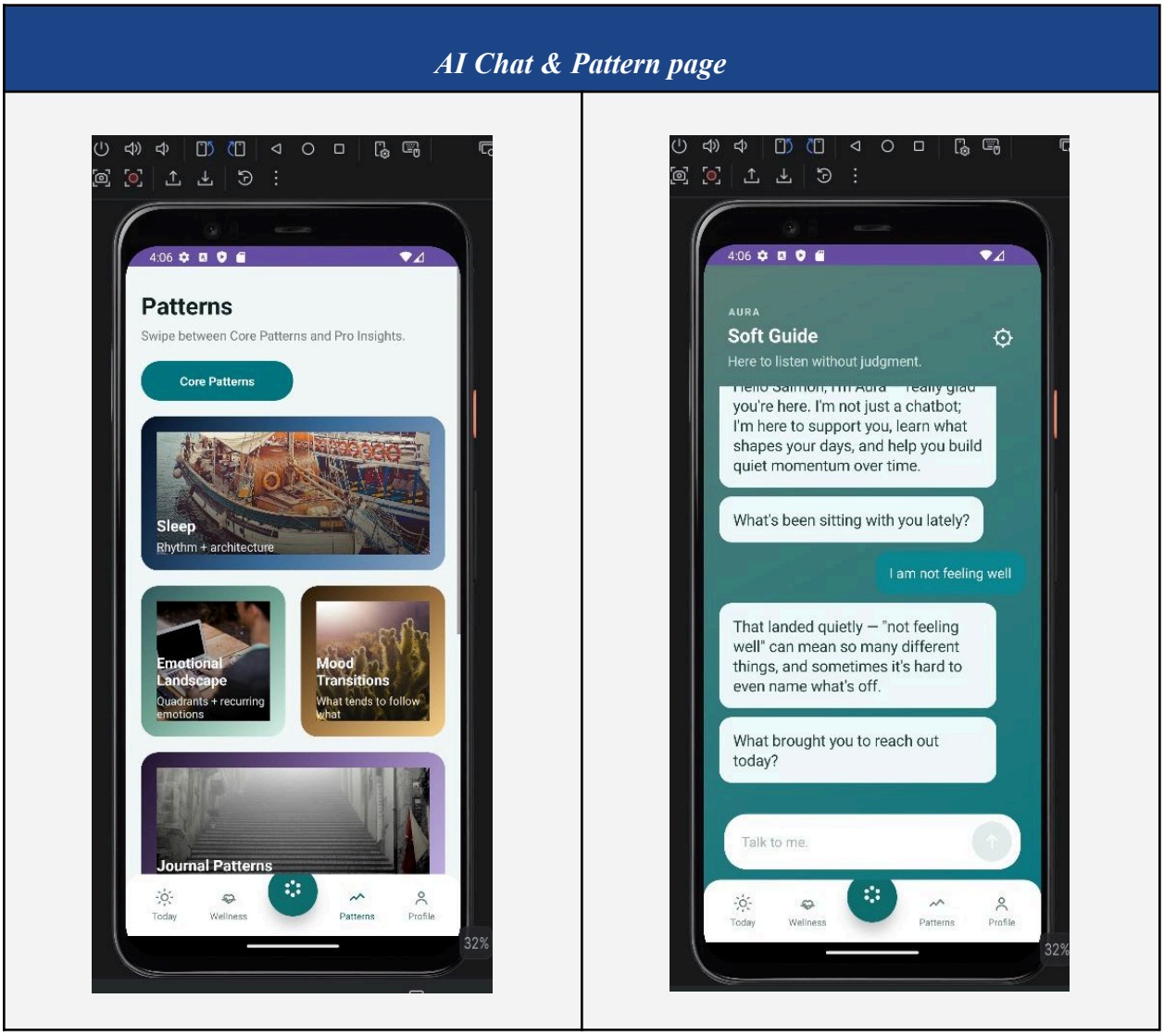


Figure 7 : Aura AI chat and Pattern Page

Architecture and Integration

This application does not utilize Fragments, a navigation graph, nor any dependency injection framework. As such, the development of my features adhered to the project architecture, where all important screens consist of separate Activities, each having its own XML layout file.

The key classes that I created/used include:

ChatActivity.java — controls the Aura conversation interface.

AiChatClient.java — performs the AI network request and handles fallback responses.

PatternsActivity.java — displays the seven available insight categories.

PatternDetailActivity.java — dynamically displays the selected insight category.

activity_chat.xml — defines the Aura chat interface.

activity_patterns.xml — defines the main Patterns screen.

activity_pattern_detail.xml — provides the container for dynamically generated insight content.

The navigation among the screen and other parts of the application takes place using explicit intents. The navigation at the bottom enables the users to navigate among the Today, Wellness, Chat, Patterns, and Profile screens with consistency in the navigation interface.

Key Implementation – Aura AI Chat Companion

Aura is a lightweight AI-based wellness chatbot screen. The main function of the screen includes a chatting interface that shows users' messages on the right and Aura's answers on the left side of the screen. Messages are created programmatically and appended to the vertical container. Upon launching ***ChatActivity***, it creates an instance of ***AiChatClient*** and fetches the logged-in user using ***SessionManager*** and ***DatabaseHelper***. Then, it holds a name that can be passed in the prompt that is sent to the AI service to make the answer more customized. There is a ***TextWatcher*** applied to the input field. Only when the input is not empty, the Send button becomes active. It ensures that the application won't process empty messages. As soon as the message is sent successfully, it is added to the chatting interface, keyboard is closed, input field is wiped and a "typing" bubble of Aura is shown. Variable ***isWaitingForReply*** blocks the possibility of sending several messages while one of the previous requests is being processed. When the answer is received, typing bubble is removed, and waiting for reply is set to false.

```
private void sendMessage() {
    String message = inputMessage.getText() != null
            ? inputMessage.getText().toString().trim()
            : "";

    if (TextUtils.isEmpty(message)) {
        return;
    }

    if (isWaitingForReply) {
        return;
    }

    addMessage(message, true);
    inputMessage.setText("");
    updateSendButton(false);
    hideKeyboard();
    showTypingBubble();
    isWaitingForReply = true;

    aiChatClient.send(userName, message, reply -> {
        removeTypingBubble();
        isWaitingForReply = false;
        addMessage(reply, false);

        updateSendButton(inputMessage.getText() != null
                && !TextUtils.isEmpty(
                inputMessage.getText().toString().trim()));
    });
}
```

Figure 8 : Sending the user's message and displaying Aura's asynchronous response

If the message is not blank and there is no other response pending, then this block modifies the interface and calls AiChatClient. After the response is received, the callback modifies the interface.

AI Network Layer

The handling of the AI request happens in its own isolated context in AiChatClient. Isolating the networking part from the ChatActivity makes it much clearer. The client utilizes a single-thread ExecutorService for making the request in a separate thread from the Android main one. The interface actions have to happen in the main thread, while the network actions may take some

seconds. Running the request directly in the main thread will either freeze the app or throw a NetworkOnMainThreadException.

Once the asynchronous request is finished, the Handler attached to Looper.getMainLooper() receives the results in the main thread. Then, the result handler can make updates to the message container in ChatActivity.

```
private final ExecutorService executor =
        Executors.newSingleThreadExecutor();

private final Handler mainHandler =
        new Handler(Looper.getMainLooper());

public void send(String userName, String userMessage,
                 Callback callback) {

    executor.execute(() -> {
        String reply;

        try {
            reply = requestPollinationsReply(
                    userName, userMessage);
        } catch (Exception e) {
            reply = createFallbackReply(userMessage);
        }

        String finalReply = reply;

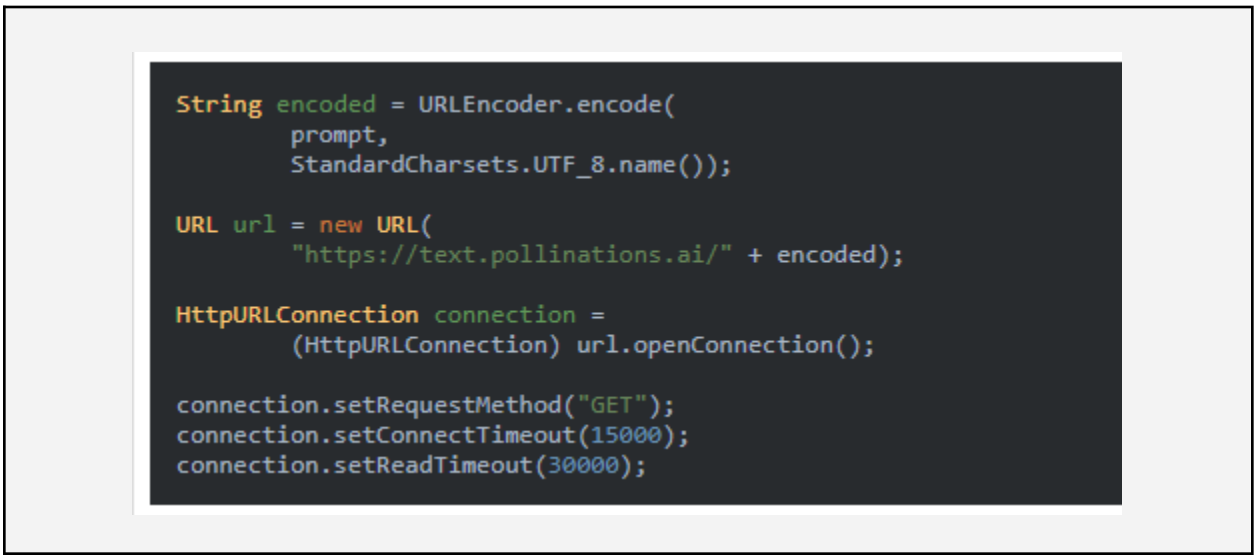
        mainHandler.post(() ->
                callback.onReply(finalReply));
    });
}
```

Figure 9 : Processing the AI request on a background thread and returning it to the UI thread

The code also includes exception handling. When the remote service cannot be reached, returns an invalid response or times out, the application generates an appropriate local fallback response instead of crashing.

API Communication

This application uses the Pollinations text generation API via Java's ***HttpURLConnection*** class. The prompt consists of the user message, user name, and Aura personality instructions. The prompt is then URLEncoded before being appended to the request URL. Timeout values of 15 seconds for the connection and 30 seconds for reading were set. This avoids a deadlock situation when there is a slow internet connection or the AI service is not available.



```
String encoded = URLEncoder.encode(
        prompt,
        StandardCharsets.UTF_8.name());

URL url = new URL(
        "https://text.pollinations.ai/" + encoded);

HttpURLConnection connection =
        (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");
connection.setConnectTimeout(15000);
connection.setReadTimeout(30000);
```

Figure 10 : Configuring the HTTP request for the Pollinations text-generation service

In this app, the Pollinations text generation API is accessed through the Java class called HttpURLConnection. The prompt in this case comprises the user message, user name, and the instructions for the aura personality. After that, the prompt is URLEncoded and added to the request URL. The timeout was set at 15 seconds for connection and 30 seconds for reading. This will prevent the deadlock scenario where there is a poor internet connection or the AI service is down. The response will be read by the use of a BufferedReader. In case there is an empty response or if there is an occurrence of an exception from the service provider, the response that would have been normally expected will be ignored, and instead, the fallback response mechanism will be applied. The prompt of Aura is intended to ensure that the responses are kept short and supportive within the context of a wellness application. The prompt also ensures that the system does not pose itself as a therapist or diagnose any medical condition.

Dynamic Message Interface

The chat messages are created programmatically through the use of the TextView widgets. There are varied background sources, text color, and maximum width that are set according to whether the sender is Aura or the User.

```
private void addMessage(String text, boolean fromUser) {
    TextView bubble = new TextView(this);

    bubble.setText(text);
    bubble.setTextSize(fromUser ? 16 : 18);
    bubble.setPadding(dp(16), dp(14),
            dp(16), dp(14));

    bubble.setBackgroundResource(fromUser
            ? R.drawable.bg_message_out
            : R.drawable.bg_message_in);

    bubble.setTextColor(fromUser
            ? 0xFFFFFFFF
            : 0xFF203B3B);

    LinearLayout.LayoutParams params =
            new LinearLayout.LayoutParams(
                    LinearLayout.LayoutParams.WRAP_CONTENT,
                    LinearLayout.LayoutParams.WRAP_CONTENT
            );

    params.gravity = fromUser
            ? Gravity.END
            : Gravity.START;

    messagesContainer.addView(bubble, params);
    scrollToBottom();
}
```

Figure 11 : Dynamically generating user and Aura message bubbles

After adding each message, the `ScrollView` automatically moves to the bottom so that the newest message remains visible.

Fallback Response System

It is true that the chat functionality is dependent on internet connection, but the application has been designed in such a way that it is not rendered non-functional if there is no remote service.

The createFallbackReply() function analyses the text typed by the user and gives an appropriate reply from a predefined list. A number of categories are created depending upon the emotional state, and some of the frequently occurring states include fear, sadness, frustration, and tiredness. A generic reply is provided if the text typed by the user is not found among any of the categories.

There are three major advantages of this system:

- The application will not crash even if the AI service is offline.
- The user gets a relevant reply.
- The chat system is still demonstrable.

This fallback system is not being suggested as a substitute to the AI service.

Key Implementation – Patterns and Insights

The Patterns section offers a graphical representation of various user behavior and wellness practice types. The following seven pattern cards have been developed:

- Sleep
- Emotional Landscape
- Mood Transitions
- Journal Patterns
- Practice Patterns
- Consistency
- Goal Momentum

Each card holds local images, overlays, title and description. Instead of loading pattern cards images during runtime, drawable resources have been used. It makes the application more reliable and lets the screen be loaded even in case if there is no Internet connection on the device.

PatternsActivity provides each card with its unique category ID. Once the user selects any pattern card, PatternsActivity transfers the ID to PatternDetailActivity via Intent extra.

```
private void openPatternOnClick(
        int viewId, String pattern) {

    findViewById(viewId).setOnClickListener(v -> {
        Intent intent = new Intent(
                PatternsActivity.this,
                PatternDetailActivity.class);

        intent.putExtra(
                PatternDetailActivity.EXTRA_PATTERN,
                pattern);

        startActivity(intent);
    });
}
```

Figure 12 : Passing the selected pattern category through an Intent

During onCreate(), the method is connected to each card:

```
openPatternOnClick(R.id.card_sleep, "sleep");
openPatternOnClick(R.id.card_emotional, "emotional");
openPatternOnClick(R.id.card_mood, "mood");
openPatternOnClick(R.id.card_journal, "journal");
openPatternOnClick(R.id.card_practice, "practice");
openPatternOnClick(R.id.card_consistency, "consistency");
openPatternOnClick(R.id.card_goal, "goal");
```

Figure 13: Each card included in the method

Reusable Pattern Details Screen

```
static PatternContent from(String pattern) {
    switch (pattern) {
        case "emotional":
            return emotional();

        case "mood":
            return mood();

        case "journal":
            return journal();

        case "practice":
            return practice();

        case "consistency":
            return consistency("Consistency");

        case "goal":
            return goalMomentum("Goal Momentum");

        default:
            return sleep();
    }
}
```

Figure 14 : Mapping a pattern identifier to the appropriate insight content

Each PatternContent object contains:

- The detail-screen title.
- Snapshot label.
- Main headline.
- Supporting description.
- Summary metrics.
- Insight sections.

- Empty-state information.
- Suggestion chips.
- Colour-tone selection.

The Activity creates the dynamic parts needed for the interface. The snapshot card appears first, then the sections of the category and feedback buttons. The sections have either metrics or an empty state message.

```
PatternContent content =
        PatternContent.from(pattern);

((TextView) findViewById(
        R.id.text_detail_title))
        .setText(content.title);

container = findViewById(
        R.id.detail_container);

addSnapshot(content);

for (DetailSection section : content.sections) {
    addSection(section);
}

addHelpfulButtons();
```

Figure 15: Dynamically constructing the selected pattern-details interface

This reutilized pattern decreased redundancy in the code and increased efficiency while adding or changing the insight categories. Future category will be created by defining another PatternContent method and adding it to the mapping process..

Integration of libraries and APIs

In my contribution, no third party networking library like Retrofit, Volley, or OkHttp was included. Aura Network requests were carried out by using the normal HttpURLConnection class. Java concurrency classes and the Handler/Looper in Android were employed in threading.

The major technologies used include:

Technology	Purpose
Java	Application and feature logic
XML	Screen layouts and drawable resources
HttpURLConnection	Sending requests to the AI service
Pollinations text-generation service	Producing Aura responses
ExecutorService	Performing network operations in the background
Handler and Looper	Returning results to the main UI thread
Explicit Intents	Navigation and transferring pattern identifiers
DatabaseHelper	Retrieving the current user's information
SessionManager	Identifying the logged-in account
AndroidX AppCompat	Activity compatibility and application structure
Material Components	Existing project interface components

Table 7 :technologies

The application's manifest includes the Internet permission required by Aura:

```
<uses-permission
    android:name="android.permission.INTERNET" />
```

Figure 16: Android Permission for accessing the internet

The Patterns feature itself does not require an internet connection because its layouts, images and prototype content are stored locally.

Version Control

My implementation was done independently in the Saimon-All-Page branch. This made it possible to implement the chat feature and Patterns separately from the shared project. Once the screens and navigation had been reviewed, the implementation was merged into the shared project. Git was used to track changes in the files and avoid loss of working code while developing.

The files in my implementation include:

```
AiChatClient.java
ChatActivity.java
PatternsActivity.java
PatternDetailActivity.java
activity_chat.xml
activity_patterns.xml
activity_pattern_detail.xml
Chat and Patterns drawable resources
```

Figure 17: All Java Files included in my task

4.3. Juri’s Contribution

The two main parts I wrote, for the CalmU application, were authentication flow and profile-related. These parts are crucial as they the user's initial interaction with the application, and they give the user access to their progress, settings, achievements, and challenges. Implementation was done by following the Activity based structure employed in the project with the aid of Android studio, Java and XML.

Development Tools And Environment

Development was done in the Android Studio development environment. The Activity logic was written in Java and the user screen UI designs were made in XML. The screens are tested: Appearance, navigation flow and basic input validation of screens are tested with the help of the android emulator. Version control was also employed using GitHub where separate branches were created for the work and eventually pushed to the remote repository.

Android Studio project structure for Juri's implemented screens.

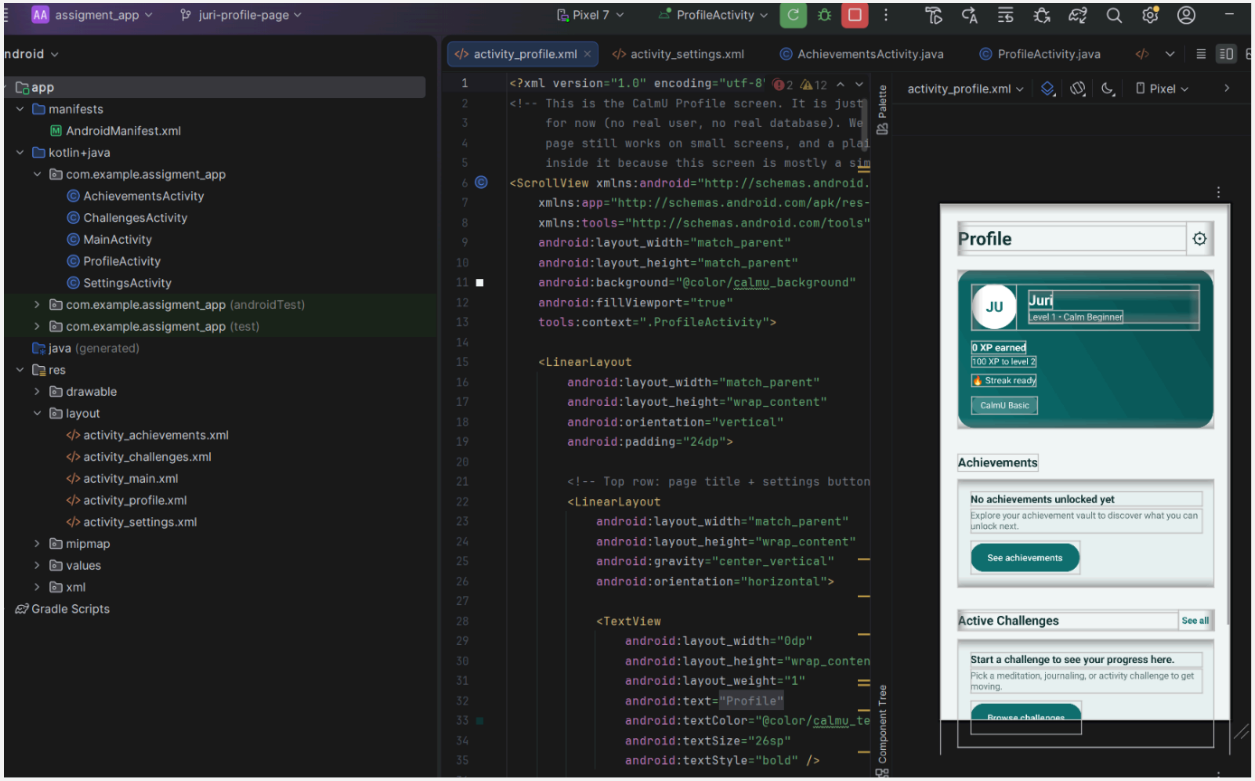


Figure 18: Juri implemented the screen for the profile page

Android Studio project structure for Juri's implemented screens.

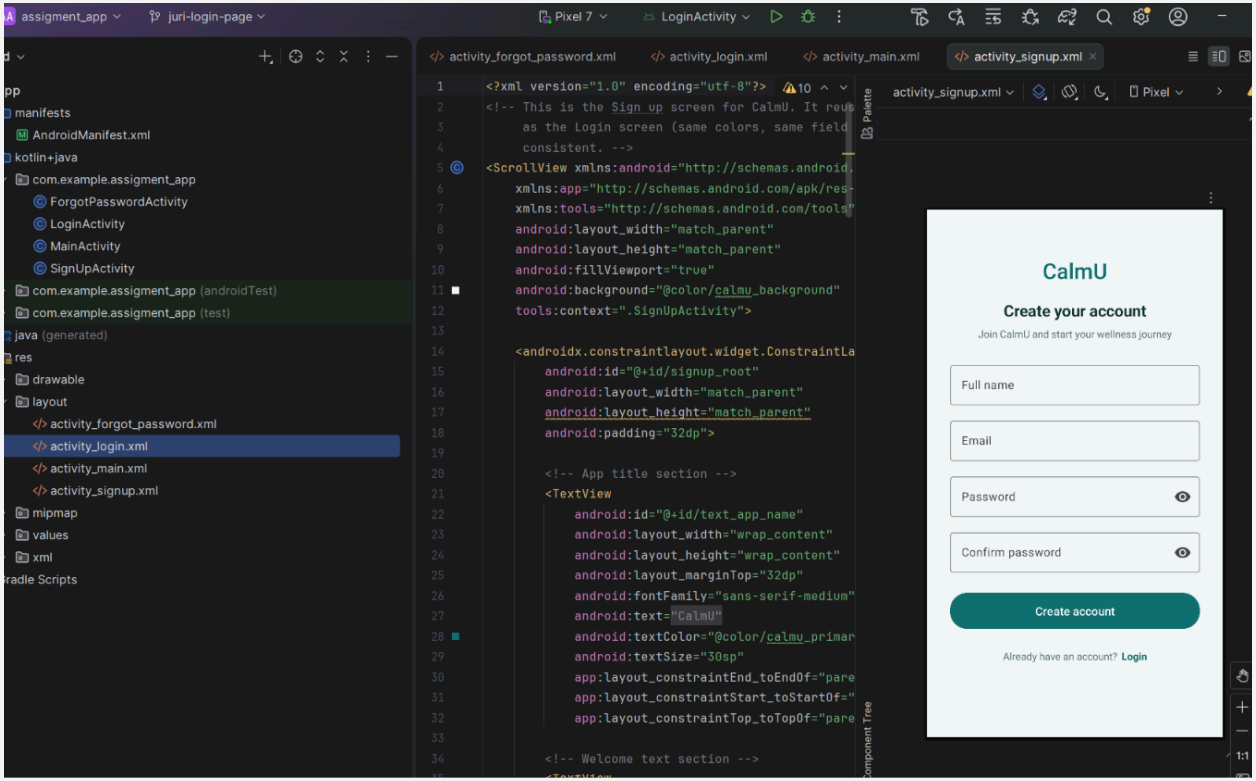


Figure 19: Juri implemented screen for the Login page

Authentication Flow implementation

The initial part of my work was the authentication flow! This comprised the Login, Sign Up and password forgot screen. On the Log in screen, the email and password are so that the user can log in, but on the Sign up screen, the full name, email, password and confirm password are so that the new user can register for the app. Forgot password screen added, so the users can ask for password recovery via their e-mail.

Each screen was developed as an Activity with its XML layout design file. This led to implementing the process in a more organised manner as each screen had a specific objective. Explicit Intents were used to link the Login screen to the Sign Up / Forgot Password screen to enable users to easily progress between the authentication screens.

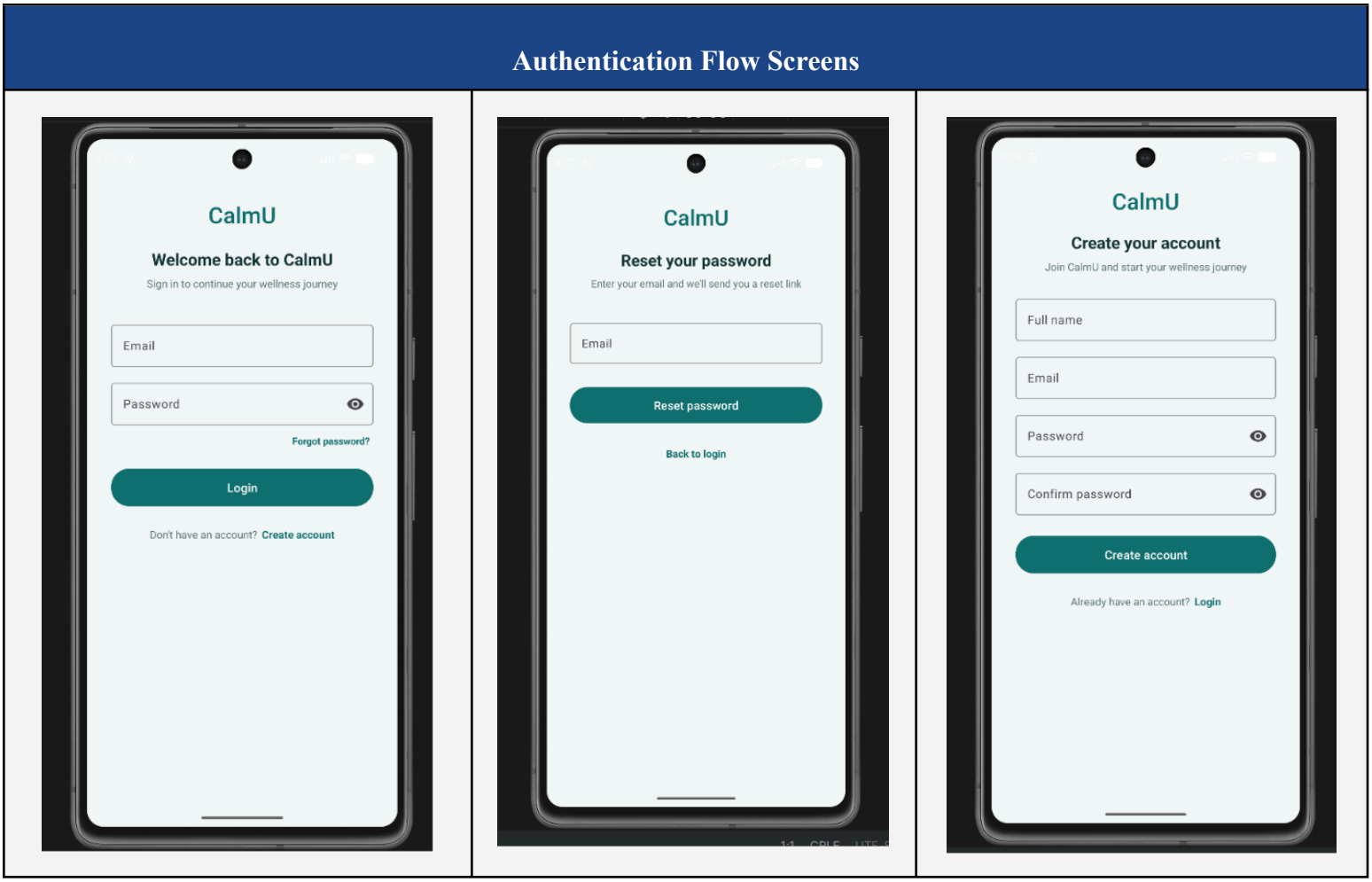


Figure 20 : Authentication Flow Screens

Basic validation was added just to make things easier for the user. The Login screen, for instance, validates the empty email and password fields and the validity of the email format. The Sign Up screen also validates that all required fields are filled out, and that the password and confirm password fields are the same. These validation messages guide users and prevent user entries errors.

Profile Screen Implementation

The second part of my work was the Profile section. The Profile screen was made to show all the user's basics including his/her level, XP, streaks, and the type of an account. There are achievements and challenges sections to help users see their achievements and further explore the app. The Profile page design is characterized by an easy and soothing interface reflecting the CalmU theme. The use of rounded cards, soothing colors, and legible texts made the page easy and user-friendly for the students. The Java Activity is responsible for handling the navigation from the Profile page to the other related pages including Settings, Achievements, and Challenges.

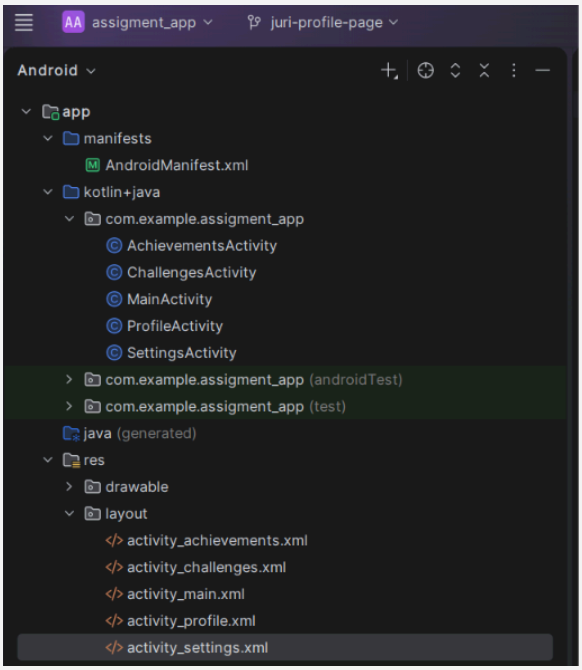
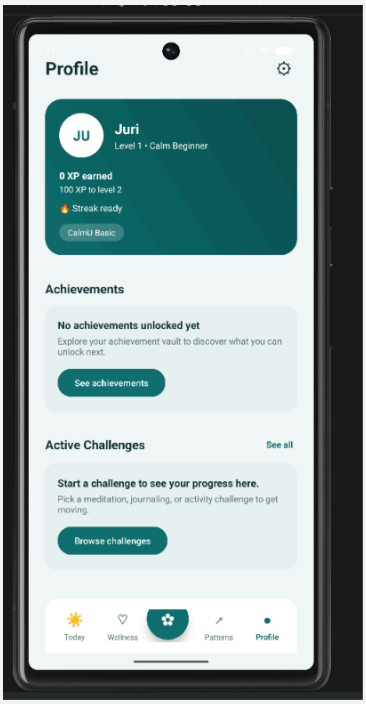
Profile Screen Implementation	
	

Figure 21: Profile Screen Implementation

Apart from the development of Profile screen, I have developed Settings, Achievements, and Challenges screens. The Settings screen includes different groups of options such as app settings, account settings, and help and support. The Achievements screen contains a message about the empty state in case there are no achievements unlocked at the moment.

The same goes for the Challenges screen since it has an empty state too. These sections contribute to completing the Profile section. Even though the functionality is on a prototype level, the pages give us an idea of the end user experience in the application.

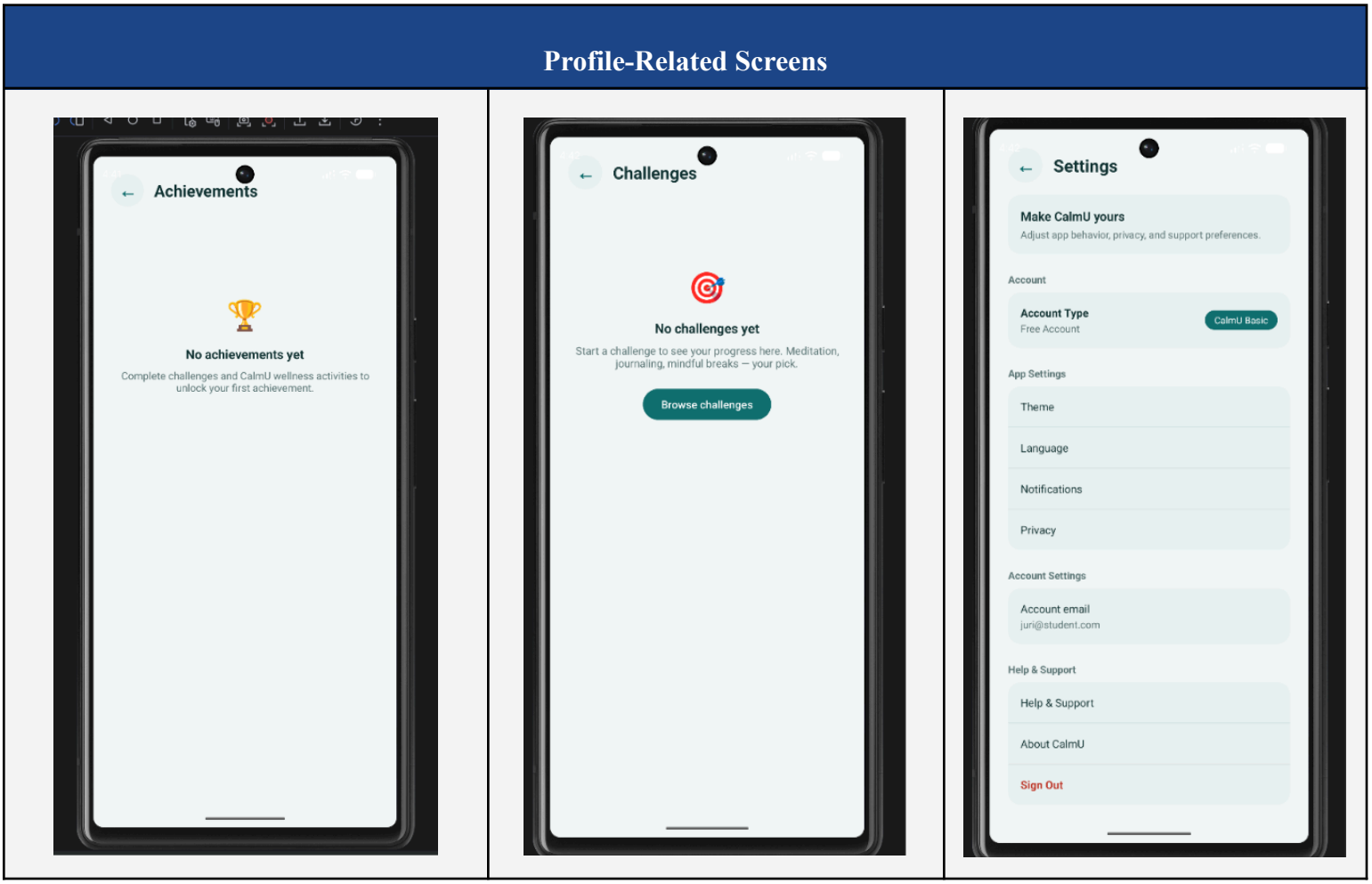


Figure 22: Profile-Related Screen Implementation

Navigation and Experience Improvements

Navigation is one of the key aspects of my project. In the authentication process, a user has the option of navigating from the Login screen to the Sign Up or Forget password screens. Within the Profile module, a user may navigate to the Settings, Achievements, and Challenges screens.

I have also made a number of improvements to user interface features for improved navigation. Back navigation buttons have been enhanced so that they can be easily seen and used on the Profile pages which are related. Bottom navigation area has also been updated to enhance visual consistency with the rest of the app design.

Version Control

The work was done through GitHub. The tasks relating to authentication and profile were done on different branches in order to ensure that the work is structured and not conflicted with the work done by other team members. After testing the code, the work was committed and pushed to the repository.

Version Control using GitHub

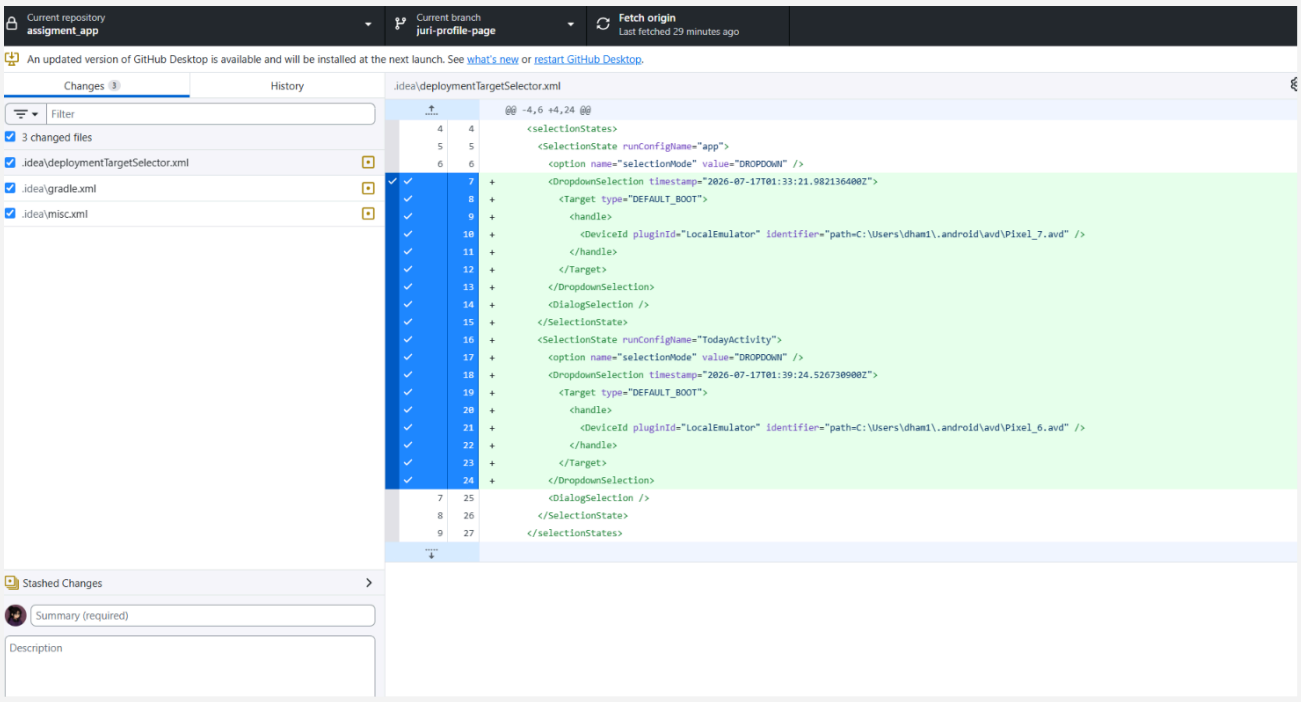


Figure 23: Version Control & branch management using GitHub

In general, my contribution was aimed at forming the access flow for users as well as the profile functionality of the CalmU app. The authentication screens provide a way for users to access the application, create accounts, and regain access to the app when necessary. The Profile screen allows users to see their achievements and problems. This way, all these components form the basic architecture of the CalmU app.

4.4 Bader’s Contribution

Overview of Contribution

My part of the CalmU application was creating the Today Dashboard and related daily wellness tools. I designed and implemented the Today screen, Mood Tracker, Sleep Logging interface, Today Focus activities, activity sequence, and mood history display using Java and XML in Android Studio. These functions enable users to track their mood at the moment, their sleeping data, propose the activities they are advised to do on a daily basis and access their last wellness data from the dashboard.

Tools and Development Environment

The features were written in Java for the application logic and in XML for user-interface layouts, and built in Android Studio. The implementation followed the existing Activity-based architecture of CalmU. Each main screen has been developed as an Android activity and explicit intents are used when a navigation between different screens is required. SharedPreferences were used through the existing TodayPrototypeStore to store the latest mood, sleep information, and the completion status of Today Focus activities. This enabled the information to be retained through the user's screen-change or app re-open. The existing report confirms that the Today Dashboard data is managed through TodayPrototypeStore, which stores the latest mood, sleep data, and task status.

Architecture and Integration

My contribution was integrated with the shared architecture used by the whole application. The Today Dashboard is the main interface that will be displayed for repeat customers, offering entry to the Mood Tracker, Sleep Logging, and day by day wellness actions. The dashboard fetches the information the user has stored and updates the mood, sleep status and activity sequence shown.

Today Focus section lists daily wellness activities and shows if these activities are upcoming, completed or skipped. The activity cards are linked to their respective screens and users could

start an activity directly from the dashboard. The implementation was also based on the same bottom navigation pattern seen across CalmU, both in Today and Wellness, Chat and Patterns, as well as Profile, to ensure the consistency of navigating between the content areas.

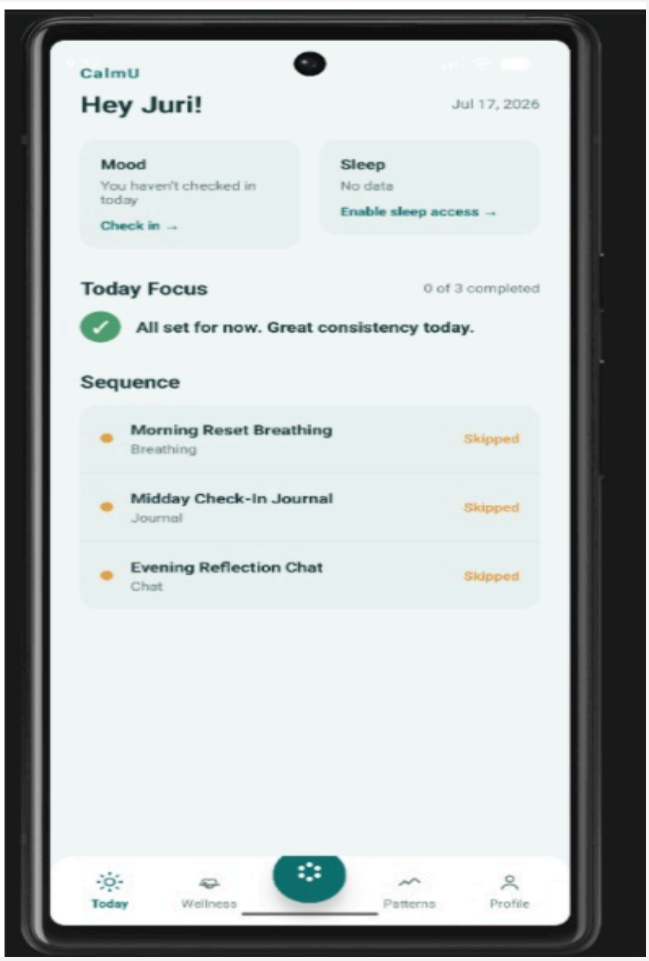


Figure 24: CalmU Today Dashboard and daily wellness activities

The Today Dashboard integrates the user's mood, sleep, Today Focus progress and daily activity sequence into one interface as illustrated in Figure 1. It shows the user their most recent status from the day, and gives immediate access to the main wellness features.

Key Implementation-Mood Tracker

Mood Tracker is designed to enable users to capture their current emotional state in a simple and organized manner. Emotions are classified into four groups on the interface: Vibrant, Settled, Agitated and Drained. A few feelings exist in each category and the user has the option to choose one of them to save the mood entry. Once the user writes down a mood, the latest mood is saved and shown to the user in the application, so that they can see their current mood.

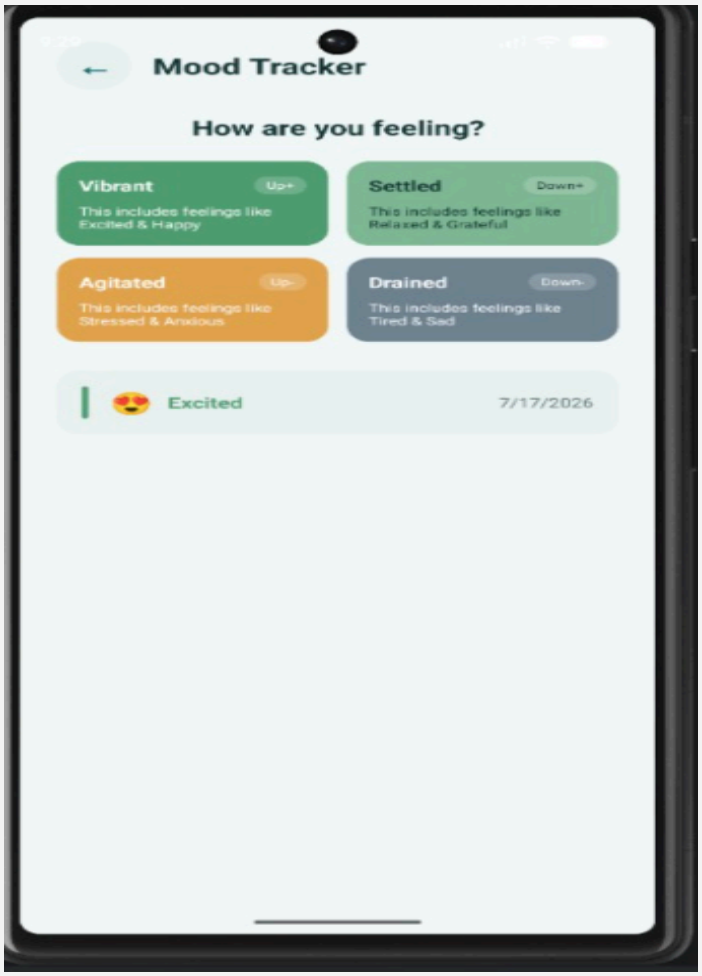


Figure 25: Mood Tracker interface showing the four mood categories and the latest recorded mood.

As shown in Figure 2, emotions are organised into four categories to make mood selection easier and clearer. A brief description is provided for each category and the most recent recorded mood and date are shown at the bottom of the screen for easy reference.

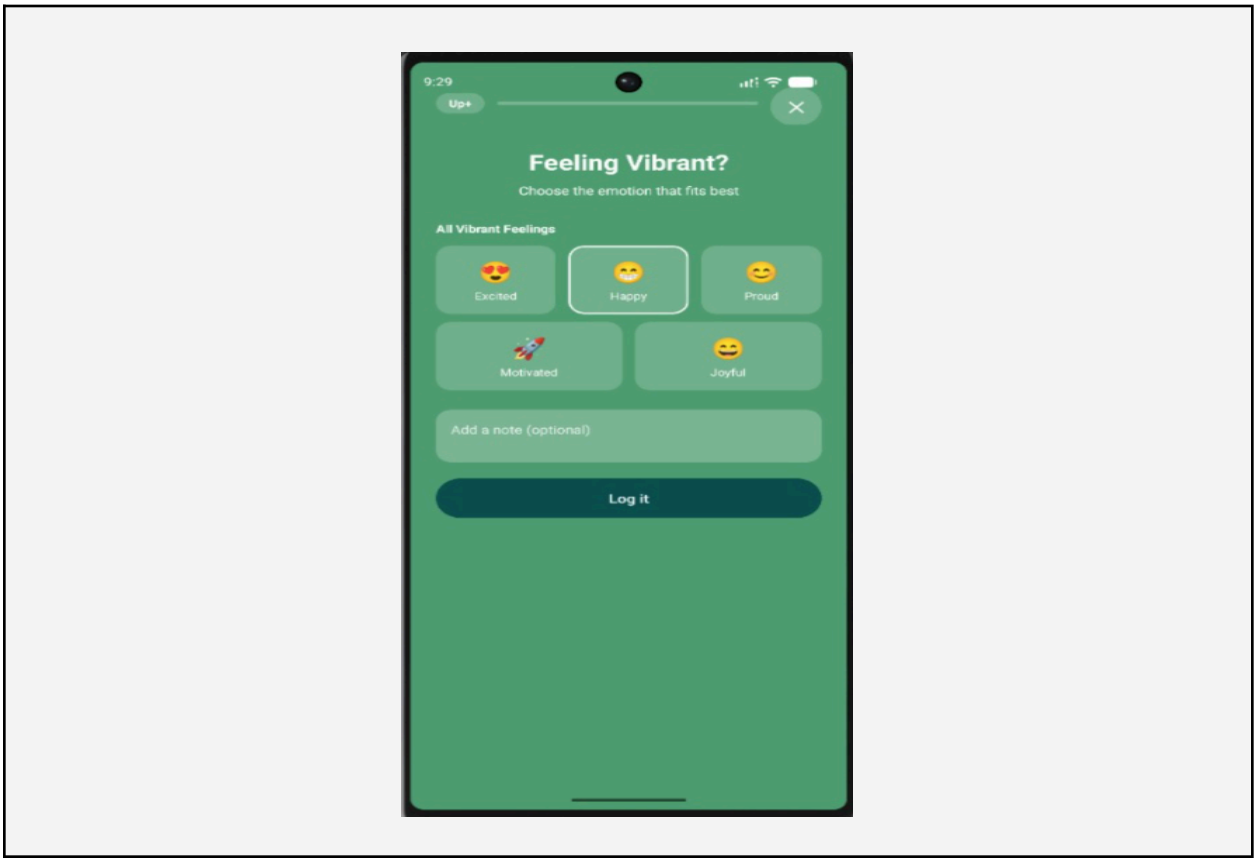


Figure 26: Mood Detail interface for selecting and saving a specific feeling

The user will be directed to the Mood Detail screen once they have chosen the mood category. This screen shows a set of very specific feelings corresponding to the category selected. The user can select one feeling, enter an optional note and press the Log it button to save the mood entry.

```
// Tapping any mood category card opens MoodDetailActivity, and
// tells it which category was tapped through an Intent extra.
openMoodDetailOnClick(R.id.card_vibrant, MoodDetailActivity.CATEGORY_VIBRANT);
openMoodDetailOnClick(R.id.card_settled, MoodDetailActivity.CATEGORY_SETTLED);
openMoodDetailOnClick(R.id.card_agitated, MoodDetailActivity.CATEGORY_AGITATED);
openMoodDetailOnClick(R.id.card_drained, MoodDetailActivity.CATEGORY_DRAINED);
```

Figure 27: Navigation from the four mood categories to the Mood Detail Screen using explicit

The four mood category cards were connected to MoodDetailActivity using explicit Intents. The user clicks on Vibrant, Settled, Agitated, Drained, the selected category is sent to the next screen. This allows the Mood Detail screen to display the specific feelings related to the selected mood category.

Key Implementation-Sleep Logging

The Sleep Logging feature was developed to allow users to record their bedtime and wake-up time directly from the Today Dashboard. A simple dialog interface is used to reduce the number of steps required to enter sleep information. Once the user saves the entry, the application works out how long they slept and updates the latest sleep data that is shown on the dashboard.



Figure 28 :Sleep Logging dialog for recording bedtime and wake-time.

As shown in Figure ***28*** , the Sleep Logging feature uses a dialog that allows users to enter their bedtime and wake-up time without leaving the Today Dashboard. This design makes the process faster and easier because the user can record the information within the same screen. Once the entry is saved, the most recent sleep duration will be shown on the dashboard.

5.0. Use of AI

5.1. Saad’s Contribution

For my part which is Wellness Hub in which i developed (Breathing, Journal and Meditation) I used Claude in a pretty limited way mostly to get unstuck. Most of the screens, activities and logic I built myself in Android Studio. I mainly used claude when i hit build errors and i didnt understand right away. For example, when I got the resource compilation error on activity_breathing_session.xml i wasnt sure at first what was even going wrong, Claude was then used to understand the error. Once It was understood it was just a missing namespace declaration, I fixed it. And it also pointed out this could be a repeated mistake, so I went and checked the rest of my layout files for the same issue.

Claude was also used to talk through how to structure the timer for the breathing sessions, the circle animation with ViewPropertyAnimator, since I hadn't built anything like this before it wasn't easy for me. I also used claude to understand the logic behind extending the SQLite database to add a journal_entries table.

Everything else was also tested side by side with claude and (./gradlew assembleDebug) and on the emulator, one screen at a time, before starting the next. So that problem must be solved before moving on. So, as a learning and productivity tool Ai was generally used overall for clarification, understanding the problems better, driving their solutions and general questions on Android studio Logic.

5.2. Saimon's Contribution

Generative AI was used in a limited supporting role during the development of my assigned features, which were the Aura AI Chat Companion and the Patterns and Insights section. The main Generative AI tool used was ChatGPT. It was treated as a reference and learning aid rather than being used to produce the complete application. ChatGPT was occasionally used to clarify Android development concepts and interpret technical error messages encountered during development. For example, it helped explain background-thread processing, callbacks, Android activity navigation and possible causes of Java or XML errors. It was also used to review small sections of interface wording, such as pattern descriptions, empty-state messages and short labels displayed to users. Some general ideas for organising the different Patterns categories were discussed before the final categories and screen structure were selected manually.

The main application logic was developed and integrated manually in Android Studio. This included creating the Activities and XML layouts, connecting the navigation, implementing the message interface, integrating the AI network request, handling failed connections and creating the reusable Pattern Detail screen. Generative AI was not relied upon to make final implementation decisions. All the advice provided by ChatGPT was reviewed prior to implementation. It was compared with the current structure of the project and modified if needed so that it would be consistent with the architecture of the application based on Java and Activities. All the advice that required additional libraries and did not correspond to the requirements of the project was disregarded. Modification of the code was performed after compilation of the project, manual test of the application in the emulator in Android Studio and the corresponding screen/function. The use of Generative AI was performed responsibly through all the stages of the development of the application. No real users' credentials, passwords, contents of the private journals and conversation logs were provided to the tool. The advice provided by the AI was not copied directly but was properly reviewed and implemented after adaptation of the suggested solution to the current code of the project.

5.3. Juri’s Contribution

The generative AI was not utilized to create the Java code, the XML layouts, application screens, test results, and implementation of the assigned features. It was done following the requirements of the course in order for me to ensure that the components of authentication and Profile will be my work. The assigned features are Login, Sign Up, Forgot Password, Profile, Settings, Achievements, and Challenges screens. They were implemented manually using Android Studio. The process of implementation comprised the creation of the Activities and XML layouts, configuration of navigation via explicit intents, registration of necessary Activities in the AndroidManifest.xml file, and testing the screens via Android Emulator.

In case there were some issues related to the implementation of the program, I mainly relied on the lecturer's help, tutorial videos, error messages of Android Studio, and testing of the screens manually. In case the emulator showed black screen or did not respond at all, I used the method presented by the lecturer and switched the virtual devices one to another. GitHub actions (creating branches, commiting, pushing the files) were performed manually as well.

None of the AI-generated text was copied in the application. The source text was checked during the development process, and the application was developed and tested after every major update. Messages validation, screen navigation, layout display, and back button functionality were tested on the emulator.

Ethics played an important role in the development of the project. None of the user credentials, passwords, journal entries, and project data was sent to any external AI tool. The submitted solution stayed consistent with academic standards, and all the implementation decisions were made depending on the project architecture, lecturer's instructions, and the results of the manual test.

In general, the project provided an opportunity to develop my programming and problem-solving skills and create an Android application independently of Generative AI, as well as to realize the importance of academic integrity and tool use.

5.4. Bader's Contribution

During my contribution to the CalmU application, Generative AI was used as a supporting tool to improve productivity, enhance understanding, and assist with problem-solving without replacing my own work. It was primarily used to explain Android development concepts, provide guidance on Java and XML implementation, suggest possible solutions for debugging, improve user interface implementation ideas, and refine project documentation. The generated suggestions helped me better understand different implementation approaches and reduced development time while working on the assigned application features. AI-generated content was still never accepted without verification, however. Every suggestion for coding, technical explanation and documentation was thoroughly read, improved as required and tested in Android Studio to ensure it fulfills the project requirements and adheres to the best practices in Android development. The last implementation, debugging, testing and documentation was left to me. AI was used, therefore, not to replace technical knowledge or as a substitute for individual work and contributions, but as a learning and productivity tool, to ensure responsible use, academic integrity and confidence in the quality and correctness of the final application.

6.0. Testing and Evaluation

Testing and Evaluation for CalmU app was carried out manually and with help of AI, with each member verifying their own screens against real debug builds (./gradlew assembleDebug), by running on the Android Studio emulator and by manually testing code. Each screen was tested individually as it was built; before the next screen was started, so that any issue could be traced back to a small, recent change rather than searched for across the whole feature. The following subsections detail each member's individual testing process, and evaluation.

6.1. Saad’s Contribution

Testing for my part of the project which is Wellness Feature includes End to End from The hub, Breathing, Journal, and meditation screens. All testing was carried out manually against real debug builds with each screen verified before the next was started. The following covers the testing methods used and the test cases run. Two build failures were encountered and how they were resolved along the way.

Testing methods

No automated or unit tests were written but all the testing was manual which was carried out directly on a debug build (./gradlew assembleDebug) run in Android Studio's emulator. Testing was done one by one rather than at the end. While each screen was built, then fully tested before the next screen was even started, this was done so a failure at any stage could be traced to a small, instead of searched for across the whole feature. Each of the four screens (Wellness Hub, Breathing, Journal, Meditation) had its own separate test pass.

Table 8: Testing table

#	Feature	Test Case	Steps Performed	Expected Result	Actual Result
1	Wellness Hub	Navigation entry points	Open Wellness tab from Chat, Patterns, and Profile screens	Hub opens each time; Wellness tab highlighted teal	Pass
2	Wellness Hub	Card routing	Tap Breathing, Journal, Meditation cards	Each opens its correct screen	Pass (see figure A.1)
3	Breathing	Technique list rendering	Open technique list	3 cards show correct name, description, duration bar chart, tip line	Pass (see figure A.1)
4	Breathing	First-time tip dialog	Open a technique for the first time	Tip dialog auto-appears once; reopening the screen after checking "Don't show again" suppresses it	Pass
5	Breathing	Duration selection	Tap 1/2/3/5/10 min chips	Selected chip highlights; others reset	Pass

6	Breathing	Guided session timer	Tap Start Exercise, let a full session run	Circle animates in sync with Inhale/Hold/Exhale/Hold; phase label and count update every second; "Session complete" toast on finish	Pass (see figure 29)
8	Journal	Hub zero-state	Open Journal Hub with no entries	Shows "5 left", "0 Total entries", default mood, empty history card	Pass (see Figure A.2)
9	Journal	Save button gating	Type text without selecting a mood, then select a mood	Save stays disabled until both text and mood are present	Pass (see figure 31)
10	Journal	Prompt rotation	Save an entry, reopen Journal Entry	A different prompt is shown than the previous visit	Pass
11	Journal	Shuffle limit	Tap shuffle 3 times in one visit	Prompt changes on first 2 taps; button disables after 2	Pass
12	Journal	Entry limit enforcement	Save entries until 5 are reached	6th attempt is blocked with a limit-reached message	Pass (see figure 31)
13	Meditation	Audio + silent tracks	Play a track with sound (e.g. Seashore Calm) and a silent track (e.g. Evening Meadow)	Sound track plays audio + countdown; silent track counts down with no audio	Pass
14	Meditation	Favourites	Favourite two tracks, switch to Favourites tab, unfavourite one	Tab shows only favourited tracks; list updates immediately on unfavourite	Pass (see Figure A.3)

15	Meditation	Audio lifecycle	Start playback, then press Back	Audio stops; no playback continues in the background	Pass

Debugging and Bug Fixing

There were Two Build failures which were encountered during development, both had the same root. The first one was while building the Breathing session screen. The error message was:

```
Resource compilation failed (Failed to compile resource file:
...activity_breathing_session.xml. Cause:
javax.xml.stream.XMLStreamException: ParseError at [row,col]:[266,43]
```

The layout used a Material Components attribute, `app:cornerRadius`, on a button, but the root `<LinearLayout>` element only declared the `xmlns:android` and `xmlns:tools` XML namespaces — not `xmlns:app`. Without that declaration, the `app:` prefix was undefined, which the resource compiler rejected. The fix was adding `xmlns:app="http://schemas.android.com/apk/res-auto"` to the root element.

This is the same mistake that appeared again in another layout file which is (activity_Journal.xml). The error message was:

```
Message:
http://www.w3.org/TR/1999/REC-xml-names-19990114#AttributePrefixUnbound?co
m.google.android.material.button.MaterialButton&app:cornerRadius&app
```

Second problem was fixed by a preventive check. To do that every layout file was scanned for (app:) attributes used without a corresponding (xmlns:app) declaration, to rule out the bug. No further instances were found. This also changed the verification habit going forward instead of treating a successful partial build as sufficient, a full ./gradlew assembleDebug was also run after every new layout file to catch this class of error immediately rather than at final submission.

Testing Evidences

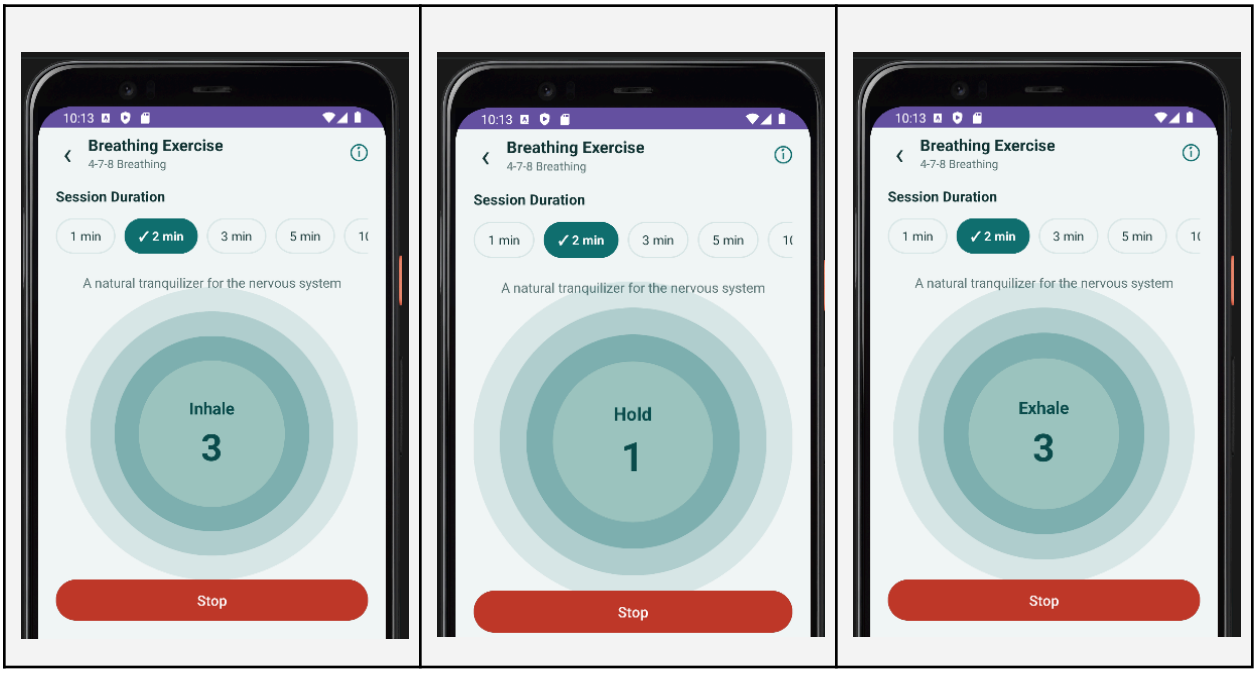


Figure 29 : Breathing Exercise Test

It is shown in the evidence (Figure 29) that the breathing exercise test is successful. It starts with Inhale – > Hold – > Exhale.

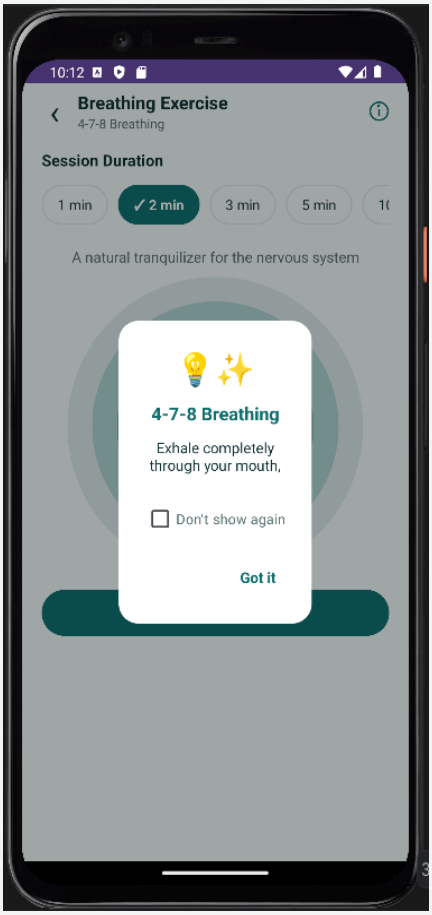


Figure 30 : Tip Dialogue Test for Breathing Exercise

This test confirms that when Breathing Exercises are accessed by the user, these exercise show a first time Tip Box with a suggestion. This tip box can be turned off for lifetime by selecting "Dont show again".

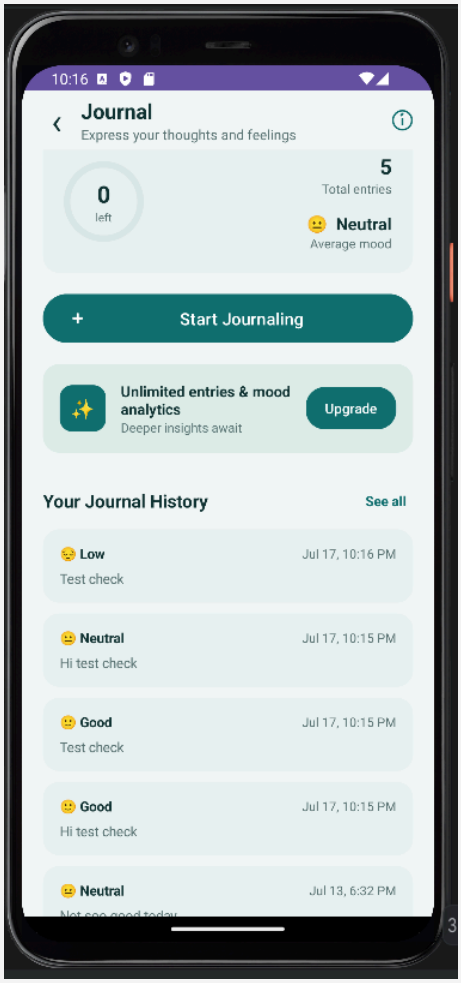


Figure 31: Journal Entry Limit Test

Journal entry test successfully confirms the limit on entries for a free account is working. It is also shown in (figure 31) that all 5 entries are successfully stored in Journal History.

Evaluation

In the end of the testing process all three exercises worked end to end on a real build: Breathing completes a full guided session with correct timer and animation, also Journal persists real entries and its entry limit was also working; Meditation plays real audio correctly by category and its favourites tab was also correctly working. The main limitation of this testing approach is that it was entirely manual and single-device and there was no automated regression tests done,

so future changes to shared files (e.g. DatabaseHelper) could silently break Wellness feature behaviour without a test suite to catch it.

6.2. Saimon's Contribution

My contribution was tested in both Aura AI Chat Companion and Patterns and Insights. The type of test conducted on both contributions involved manual functional testing through the use of an Android emulator. This means that once each element of the interface had been implemented, it was tested in order to identify any issues prior to integrating it with the rest of the application.

No automated unit tests were written for both the Aura AI Chat Companion and Patterns and Insights. The existing Android default testing files could not be utilized to test the entire chat process and patterns workflow.

Testing Methods

The following methods were used:

- Manual interface testing in the Android Studio emulator.
- Valid and invalid input testing.
- Online and offline chat testing.
- Navigation testing between Activities.
- Repeated-interaction testing.
- Testing of dynamic content for all seven Pattern categories.
- Back-button and bottom-navigation testing.
- Debug-build verification.

#	Feature	Test Case	Steps Performed	Expected Result	Actual Result
1	Aura	Empty message validation	Open Chat and leave the input field empty	Send button remains disabled and no message is added	Pass
2	Aura	Send button state	Type text and then remove it	Button enables when text exists and disables when empty	Pass
3	Aura	Valid message submission	Enter a normal message and press Send	User message appears on the right side	Pass
4	Aura	Typing indicator	Send a valid message	“Aura is typing...” appears while waiting	Pass
5	Aura	Successful AI response	Send a message with an active internet connection	Typing bubble disappears and Aura’s response appears	Pass
6	Aura	Duplicate request prevention	Press Send repeatedly before the response arrives	Only one request is accepted while waiting	Pass
7	Aura	Automatic scrolling	Send enough messages to fill the screen	Chat automatically scrolls to the latest message	Pass

8	Aura	Offline fallback	Disable the internet and send a message	A local fallback response appears and the app remains usable	Pass
9	Aura	Settings navigation	Press the settings button from Chat	Settings screen opens	Pass
10	Patterns	Main card rendering	Open the Patterns screen	All seven category cards and images appear	Pass
11	Patterns	Sleep navigation	Select the Sleep card	Sleep Patterns content appears	Pass
12	Patterns	Emotional navigation	Select Emotional Landscape	Emotional-pattern content appears	Pass
13	Patterns	Mood navigation	Select Mood Transitions	Mood-transition content appears	Pass
14	Patterns	Journal navigation	Select Journal Patterns	Journal-pattern content appears	Pass
15	Patterns	Other categories	Open Practice, Consistency and Goal cards	Correct content appears for each card	Pass
16	Patterns	Missing Intent fallback	Open the details Activity without a category value	Consistency content is used as the safe default	Pass
17	Patterns	Back navigation	Open a category and press Back	User returns to the Patterns screen	Pass

18	Patterns	Feedback control	Press Helpful or Not Helpful	Confirmation Toast appears	Pass
19	Navigation	Bottom navigation	Move between Today, Wellness, Chat, Patterns and Profile	Selected destination Activity opens	Pass
20	Build	Debug build	Build the application using the debug configuration	Application compiles and a debug APK is produced	Pass

Table 9: Testing table

Debugging and Bug Fixing

One critical thing about implementation was making sure the AI request does not execute on the interface thread. In the first design, there was too much integration between the interface and the networking part. There would be chances of the Chat screen freezing because of waiting for a reply. The approach was shifting the network code to AiChatClient and executing it using a single threaded ExecutorService. After that, the reply would be delivered to the Activity using a Handler from the main-thread. There were also problems related to the clicking of Send button multiple times. As long as there was no waiting state, there would be possibilities of multiple requests being sent. The isWaitingForReply boolean was introduced in order to reject all further submissions until the current callback was complete. Patterns also presented some problems of maintainability. There would have been too many activities, one for each of the seven categories. The problem was fixed with the introduction of PatternDetailActivity. It takes care of the dynamic creation of title, snapshot, metrics, sections and suggestions based on the Intent value.

Evaluation

At the completion of the test cases, the Aura function was able to accept user inputs, display bubbles for outgoing and incoming messages, detect typing status, and provide a response from either the online AI or the offline fallback AI response. It also managed to filter out empty messages and duplicate message submissions. The Patterns screen was able to show seven categories that were correctly directed to the details screen. The details Activity was able to pick up the right category contents and display the various snapshot values, sections, empty states, and suggestion chips. The biggest weakness in the testing process was that it relied heavily on manual and emulator testing. Results from the network may vary based on the availability and speed of the external AI services. The project also lacks automated regression tests, and any changes in the future may affect the functionalities.

6.3. Juri's Contribution

My contribution testing included authentication process testing and testing of the Profile-related screens. Authentication screens that have been tested include Login, Sign Up and Forgot Password. Profile screens include Profile, Settings, Achievements and Challenges. The purpose of testing included checking whether validation was performed properly, whether registered users can access the application, whether session is kept and whether each screen navigates properly to the intended destination. Also the Profile screen was tested for showing the information about the logged in user and Settings, Achievements and Challenges screens were tested for their proper functionality.

Testing Approaches

Manual testing of the application has been done using a debug version of the application on the Android Studio emulator. No automatic unit tests or interface testing has been developed for this feature. Each individual screen was tested following its implementation and then again after its integration with the whole CalmU application.

The following testing methods were used:

- Manual functional testing using the Android Studio emulator.
- Valid and invalid input testing.
- Authentication and session testing.
- Activity navigation testing.
- Back-button and bottom-navigation testing.
- User-data display testing.
- Empty-state interface testing.
- Debug-build verification.

The authentication screens were validated with blank fields, wrong email address formats, wrong passwords, and correct information. The Profile screens were validated through opening the screen from the Profile screen and validating its content, controls, and functionality.

Testing Table

No.	Feature	Test Case	Steps Performed	Expected Result	Actual Result
1	Login	Empty-field validation	Leave the email and password fields empty and press Login	Required-field messages appear and login does not continue	Pass
2	Login	Invalid email format	Enter an incorrectly formatted email and press Login	An invalid-email message appears	Pass
3	Login	Missing password	Enter a valid email but leave the password empty	A password-required message appears	Pass
4	Login	Invalid credentials	Enter an unregistered email or incorrect password	An invalid-credentials message appears and access is denied	Pass
5	Login	Successful login	Enter the email and password of an existing account	A success message appears, the session is saved and the Chat screen opens	Pass
6	Session	Existing login session	Close and reopen the app after logging in	The user is recognised as logged in and is not required to log in again	Pass
7	Login	Create-account navigation	Tap the Create Account link	The Sign Up screen opens	Pass
8	Login	Forgot-password navigation	Tap the Forgot Password link	The Forgot Password screen opens	Pass

9	Sign Up	Empty-field validation	Leave all fields empty and press Create Account	Required-field messages appear for the missing information	Pass
10	Sign Up	Invalid email	Enter an invalid email address with otherwise valid details	An invalid-email message appears	Pass
11	Sign Up	Short password	Enter a password containing fewer than six characters	A password-length validation message appears	Pass
12	Sign Up	Password mismatch	Enter different values in Password and Confirm Password	A password-mismatch message appears	Pass
13	Sign Up	Existing email	Attempt to register using an email already stored in the application	An email-already-exists message appears	Pass
14	Sign Up	Successful account creation	Enter a name, valid email and matching valid passwords	The account is created, the session is saved and the Chat screen opens	Pass
15	Forgot Password	Empty email	Leave the email field empty and press Reset Password	An email-required message appears	Pass
16	Forgot Password	Invalid email	Enter an incorrectly formatted email	An invalid-email message appears	Pass
17	Forgot Password	Valid prototype request	Enter a valid email and press Reset Password	A confirmation Toast appears and the user returns to Login	Pass

18	Profile	User information display	Log in and open the Profile screen	The account name, initials and email are displayed correctly	Pass
19	Profile	Settings navigation	Tap the Settings icon	The Settings screen opens	Pass
20	Profile	Achievements navigation	Tap See Achievements	The Achievements screen opens and displays its empty state	Pass
21	Profile	Challenges navigation	Tap See All or Browse Challenges	The Challenges screen opens and displays its empty state	Pass
22	Settings	Account information	Open Settings after logging in	The current email and prototype account type are displayed	Pass
23	Settings	Prototype options	Tap Theme, Language, Notifications, Privacy, Help or About	A suitable “coming soon” Toast appears	Pass
24	Settings	Sign-out function	Tap Sign Out	The session is cleared and the Login screen opens	Pass
25	Settings	Sign-out back-stack protection	Sign out and press the device Back button	The previously authenticated screen does not reopen	Pass
26	Challenges	Browse button	Tap Browse Challenges	A “coming soon” message appears without crashing	Pass
27	Navigation	Back-button navigation	Open Settings, Achievements or	The user returns to the Profile screen	Pass

			Challenges and press the back arrow		
28	Bottom Navigation	Main screen navigation	From Profile, select Today, Wellness, Chat and Patterns	Each selected Activity opens correctly	Pass
29	Interface	Layout consistency	Open all authentication and Profile-related screens	Text, buttons, cards and navigation remain readable and visually consistent	Pass
30	Build	Debug-build verification	Run the application using the debug configuration	The project builds successfully and launches in the emulator	Pass

Table 10 : Test

Testing Evidence

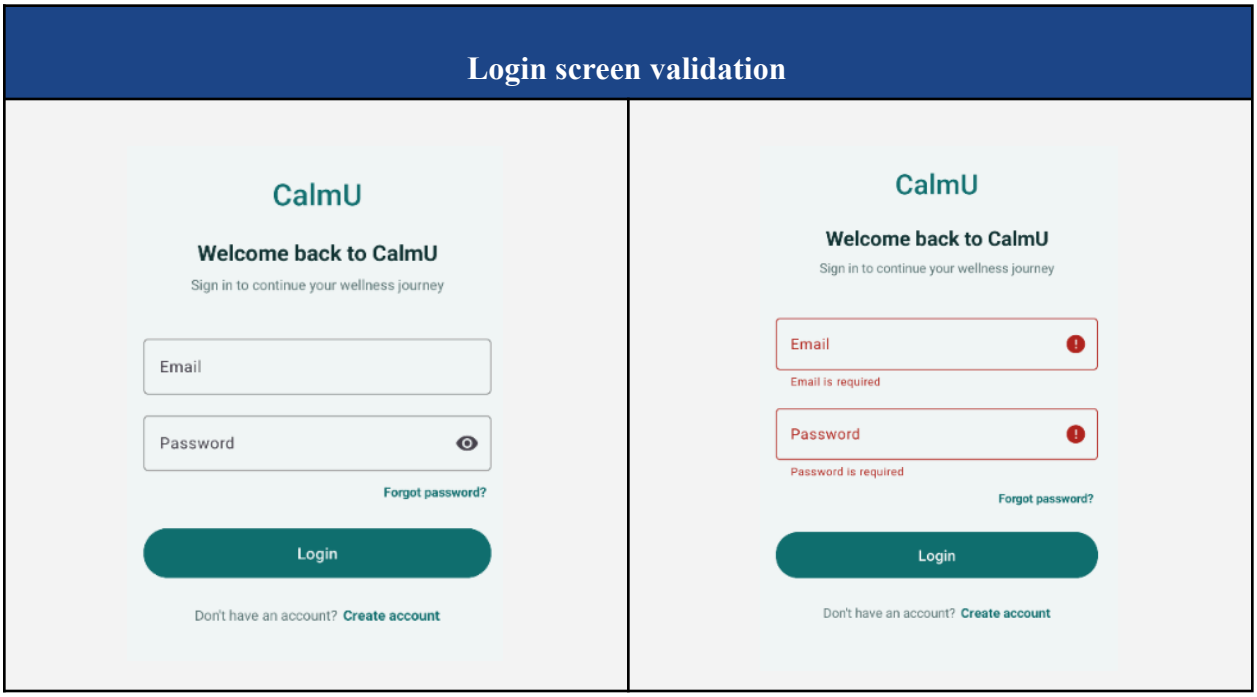


Figure 32: *Login screen validation for empty user input*

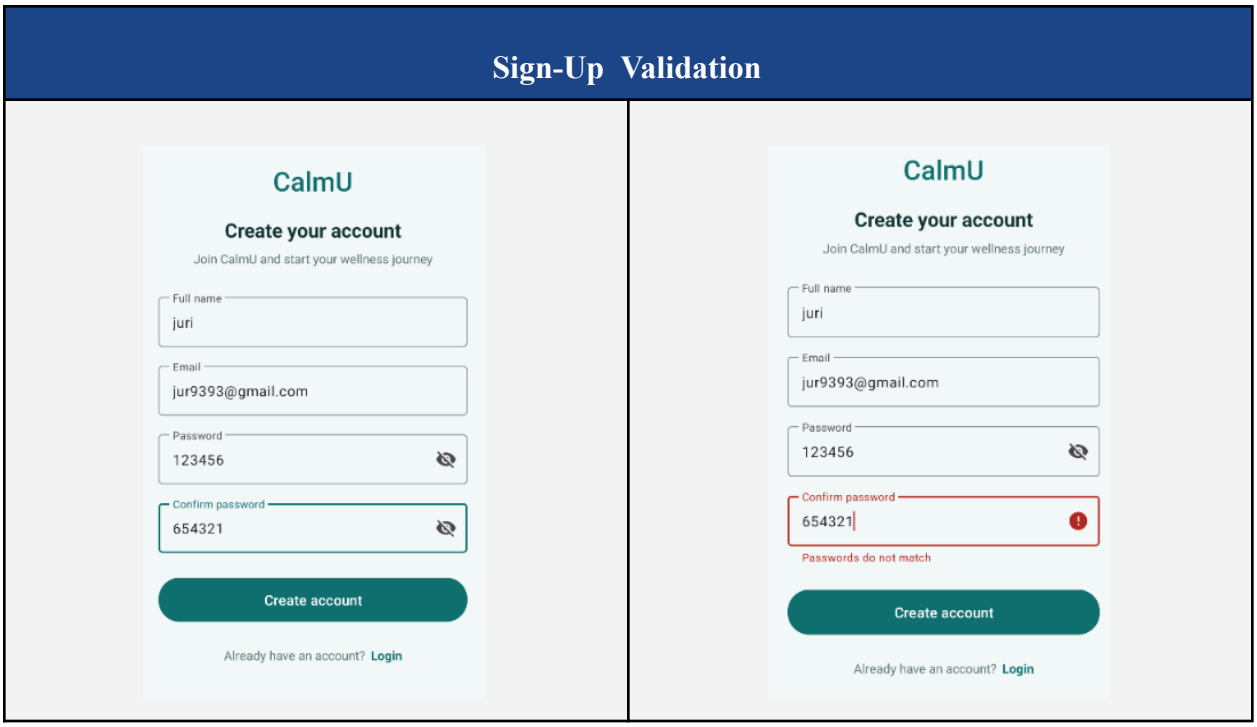


Figure 33: Sign-up validation for mismatched passwords



Figure 34: Profile screen displaying the logged-in user's account information

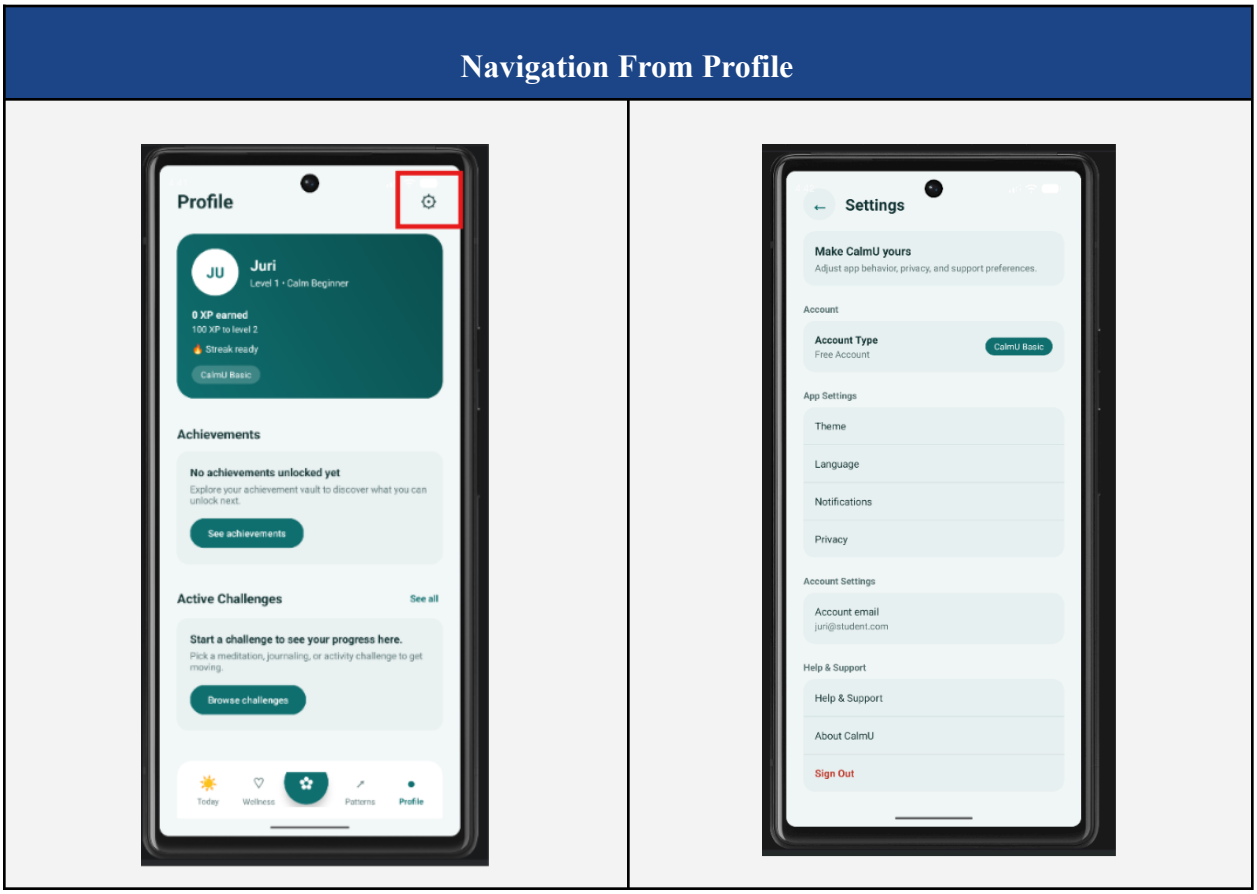


Figure 35: Successful navigation from Profile to the related screen

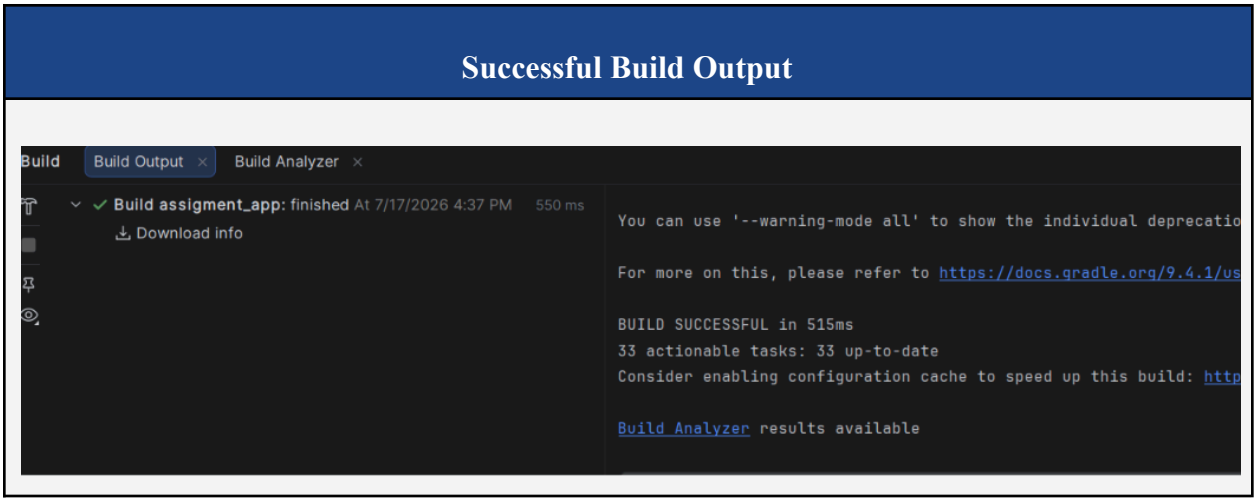


Figure 36: Successful debug build and emulator testing

Debugging and Bug Fixing

When implementing and testing the features of authentication and Profile, there were a number of areas which required further checking and tweaking.

Input Validation

It was necessary to ensure that in the authentication pages, the users cannot submit incomplete or wrong information. Without input validation, a user would be able to try to log in or sign up with empty fields, wrong email format, short password or non-matching passwords.

Input validation was solved through checking every single field before proceeding. In the case of Login, the input validation checks if email and password have been entered as well as checks the right email format. As for Sign Up, it checks user's name, minimum password length, matching passwords and if the email has not yet been registered. Previous error messages were erased before input validation.When input validation was implemented, each of the invalid input scenarios was checked on the emulator individually.

Authentication and Sessions

It was necessary to test whether a successful login or signup would create a session and that logging out would delete the session. The successful authentication could be determined by checking whether the user was redirected to the Chat screen and still stayed logged in on reopening the application. The Sign Out functionality was tested in order to determine that it would delete the session stored by the application. The Login Activity is opened with flags that clears up the existing Activity stack. This prevents users who have signed out from pressing the Back button and accessing a restricted screen.

Navigation and Manifest Configuration

Each authentication page as well as each Profile related page had been designed as an individual Activity. All necessary Activities, thus, had to be declared in AndroidManifest.xml. The explicit Intents between Login, Sign Up, Forgot Password, Profile, Settings, Achievements and Challenges activities were tested. This had to be redone after the integration of the app within the whole group application to make sure that there was no error in Activity opening or closing.

Profile Data Display

The initial requirement for the Profile screen was the connection to user data which gets saved at the registration stage. In the final version, the Profile screen connects the currently logged-in email via SessionManager and gets the user's record via DatabaseHelper. It was checked that the name of the registered user, user's initials and email were shown in the Profile screen. The Settings screen was also checked to display the email of the currently used account.

Prototype Screens' Behaviour

The functions of Achievements and Challenges screens are still prototype ones without any data about the progress of user. It was checked that these screens show corresponding empty states' messages instead of blank screens. The options which are not yet fully developed, like some settings and Browse Challenges, were linked to "coming soon" Toast messages in order to give feedback and prevent user from thinking that this button does not work.

Emulator Performance

In certain instances, there were some issues with the speed of emulation due to limited computer hardware capabilities. Testing continued with the use of an appropriate Android virtual device. Whenever necessary, the application was built anew and the process started again. Each feature was tested separately to help with debugging and to differentiate emulator-related issues from application bugs. Every time a change was made, the corresponding screen was tested once more. It was also important to test related functionality to find out whether there were no other bugs involved.

Evaluation

Testing the application in various scenarios demonstrated that the authentication module and the Profile screen functionality operated correctly. The Login function prevented incorrect, improperly formatted, or invalid user credentials from being used. The Sign Up operation validated all the mandatory parameters, password length, passwords match, and email uniqueness. Successful registration and login created a user session and gave access to the application. Persistent session allowed logging in and reopening the application without reentering user credentials. Signing out operation successfully cleared the session and returned the user to the Login screen. The Forgot Password screen validated the user's email input and gave the proper confirmation for a valid input but it is only a prototype functionality as there is no implementation of an email backend and Firebase in the application, hence no actual password recovery email is currently sent.

The Profile screen displayed the information related to the currently logged-in account. Switching from the Profile screen to Settings, Achievements and Challenges screens went properly. Bottom navigation menu allowed accessing Today, Wellness, Chat, and Patterns. While the testing process was adequate to test the key functional requirements and interface issues prior to the final integration, it was conducted only via manual testing using an emulator. Unit testing, Espresso interface testing, and testing on multiple physical devices running Android were not done. In the future, testing will have to be done using automated validation testing, database testing, session testing, and testing on devices running different versions and screen sizes of

Android. On the whole, the authentication module and the Profile module were able to provide a stable and usable interface for the CalmU prototype.

6.4. Bader's Contribution

My features that were tested were the Today Dashboard, Mood Tracker, Mood Detail and Sleep Logging. The goal of testing was to confirm that each feature worked as designed, that users' actions were being performed as described, that navigating between screens was consistent, and that the stored information was being stored accurately. Features that were implemented were tested individually after development and retesting after code changes to see if there were additional issues that arose due to the changes before completion of the final application.

Testing Methods

All testing was done manually, using a built version of the app (./gradlew assembleDebug). The functionality of each feature was validated by testing it individually before integration with the other features. The testing process included screen navigation, validation of user input, local data storage and user action-driven interface changes. Testing was done throughout the development process, not after implementation, allowing issues to be identified, isolated and solved at a much earlier stage. Once each issue was addressed, the impacted feature was retested to ensure that the desired actions had been restored without impacting other previously completed actions.

Testing Table

Feature	Tast Case	Steps Performed	Expected Result	Actual Result
1. Today Dashboard	Dashboard loading	Log in and open the Today Dashboard	Quickly loads the dashboard with the most recent mood, sleep summary, and Today Focus section	**Pass**
2. Today Dashboard	Navigation to Mood Tracker	Tap the Mood card from the Today Dashboard	Mood Tracker screen opens successfully	**Pass**

3. Mood Tracker	Mood category selection	Select any mood category (Vibrant, Settled, Agitated, or Drained)	The selected category appears on the Mood Detail screen.	**Pass**
4. Mood Detail	Mood logging	Select a feeling and tap Log it	Mood entry is saved successfully and returned to the Today Dashboard	**Pass**
5. Today Dashboard	Mood information update	Return to the Today Dashboard after saving a mood	The most recent mood that has been recorded is shown properly on the dashboard	**Pass**
6. Sleep Logging	Sleep data entry	Enter Sleep Logging under Open Sleep Logging, the bedtime and wake-up time, then tap Save sleep.	The duration of sleep is calculated and stored correctly	**Pass**
7. Today Dashboard	Sleep information update	Return to the Today Dashboard after saving sleep information	The latest sleep information is displayed correctly on the dashboard	**Pass**
8. Navigation	Screen navigation consistency	Navigate between Today Dashboard, Mood Tracker, Mood Detail, and Sleep Logging	All screens open as expected and there are no navigation errors.	**Pass**

Table 11: Test

Debugging and Bug Fixing

While testing the features implemented, some minor issues were found during the development process. Such problems were primarily related to navigation on the screen, the user interface's working and updating of information displayed after a user action. The related Java implementation and XML layouts were reviewed for each of the issues and then tested the functionality that was changed within Android Studio. Once the issues were addressed, the impacted features were retested to ensure the issues were fixed and new issues were not created with the changes. Through this iterative debugging process the stability and reliability of the features implemented were enhanced prior to the final submission.

Evaluation

Testing revealed that the features that were implemented performed as expected in normal usage conditions. The Today Dashboard was able to show the most recent data, the Mood Tracker and Mood Detail screens were able to capture mood entries by the user, and Sleep Logging was able to store and update sleep data as expected. Manual testing further validated the consistency of navigation from the implemented screens and the consistency of the results from user interactions. The testing methodology used was manual but it was enough to ensure that the features implemented were able to achieve the functional requirements and gave a stable and usable user experience.

7.0. Challenges and solution

Development of CalmU as a four people team on a shared codebase came with its own set of technical and practical difficulties. Each member faced challenges specific to their assigned feature area, and resolved them through a mix of manual debugging, testing on the emulator, and use of Generative AI for clarification. The following subsections detail each member's individual challenges and how they were addressed.

7.1. Saad’s Contribution

While building the wellness feature hub there were genuine technical and design challenges that were faced beyond simple screen building, which were mainly around fitting new functionality into an existing codebase’s conventions, also like sourcing the real assets that project did not have, and getting real time states such as (timers, animations, and audio) to work correctly across the android life cycle. All of these are detailed below, along with how each was resolved.

Fitting New Screens

This is the first challenge that was faced, the rest of the app doesn't use fragments or any shared navigation setup and every screen is its own Activity with its own Layout too. And the bottom navigation bar is also copied into each screen's XML rather then built as a reusable component. When its about adding six new screens for Wellness Hub, the choice was either to follow that same pattern or build something clean just for this feature. So for Wellness Hub the existing pattern was kept, even though it meant repeating the bottom navigation bar in each new layout, so the codebase didn't end up with two different screen structures that someone would later have to reconcile.

Sourcing real, appropriate licensed assets

Project also needed photos, illustrations, audio for the wellness hub exercises tab and none of them could be created from scratch. So using placeholder or unlicensed content wasn't an option for a submitted piece of coursework. Free properly licensed sources were used instead of illustrations from unDraw for the Wellness Hub cards, and ambient sound effects were taken from Mixkit for the three Meditation tracks that play audio with each source's licence terms

checked before the use. So that its confirm that they permitted use in an app like this without restriction.

Making Prompt Rotation Work for Journal Entry

The journal entry feature needed a real 5 entry limit and a prompt also which could rotate instead of repeating itself. For this challenge the solution of extending the existing SQLite database with a new table to track real entries per user was derived, and using SharedPreferences to track prompt rotation and shuffle counts across separate app sessions, so that the behavior would look natural instead of only looking correct on first visit.

```
private static final String COLUMN_JOURNAL_CONTENT = "content";

3 usages
public static final int FREE_JOURNAL_ENTRY_LIMIT = 5;

4 usages
private static final String[] MOOD_ORDER = {"Low", "Neutral", "Good", "Happy"};

7 usages
public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME,  factory: null, DATABASE_VERSION);
}
```

Figure 37 : Journal Entry Limit

Managing Timer, Animation, Audio

For Breathing Session (phase by phase countdown was synced to a scaling animation) to look real and intimidating and the Meditation screen (a countdown paired with real audio, only one track playing at a time) they both depend on the state that keeps changing every second while the user is on the screen. The main risk faced was that timer or audio playback would keep running after the user left the screen for example: the sound of a Meditation track continuing to play in the background, or a countdown crashing the app. So to solve this challenge Each screen's timer and MediaPlayer instance were stopped and released in the Activity's onPause(), so that nothing would continue once the screen is no longer visible.



```
    private void stopPlayback(boolean resetToIdle) {
        handler.removeCallbacks(tickRunnable);

        if (mediaPlayer != null) {
            mediaPlayer.stop();
            mediaPlayer.release();
            mediaPlayer = null;
        }

        if (resetToIdle && playingId != null) {
            Meditation meditation = findMeditation(playingId);
            List<CardViewSet> sets = viewsByMeditationId.get(playingId);
            if (sets != null) {
                String idleText = "⏱ {meditation.durationSeconds / 60} min";
                for (CardViewSet set : sets) {
                    set.durationView.setText(idleText);
                    set.playButton.setText("▶");
```

```
@Override
protected void onPause() {
    super.onPause();
    stopPlayback( resetToIdle: true);
}

1 usage
```

Figure 38 : Managing Timer code Snippet

Build Error

This problem is also covered in the Testing and evaluation section in detail. It's the same layout challenge that was faced (a missing XML namespace declaration) which broke the build twice on two separate screens. To solve this challenge every layout file in the project was checked for the same issue. The main lesson from this was procedural rather than technical which is that verifying with a full build after every new screen, rather than assuming a screen was finished once it visually looked right this would have caught the second occurrence early.

Android Studio Emulator Launch Crash

While testing CalmU the emulator was launched and relaunched multiple times within short intervals of 5 to 10 minutes. This repeated relaunching slowly increased Android Studio's memory usage, which led to Android Studio crashing on multiple occasions and caused repeated halts in development progress. After investigating the issue, it was identified that Windows Real-Time Protection was blocking access to Android Studio and its associated libraries, which was the root cause of the crashes. To resolve this, Android Studio was added to the exclusion list in Windows Security, after which the crashing issue did not occur again.

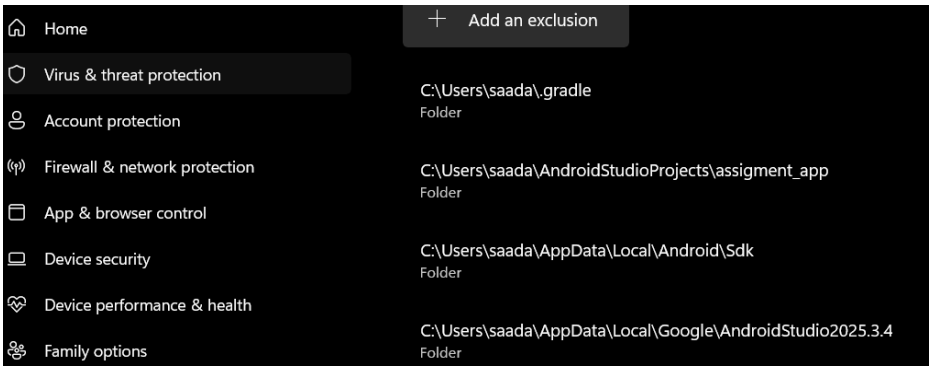


Figure 39 : Android Studio Exclusion from Windows Security

7.2. Saimon’s Contribution

Making Network Requests without Blocking the Interface

First of all, Aura needed a network request. Any operation in the Android user interface works on the main thread. Performing a network request on the same thread could freeze the interface or cause any other type of application errors. It was solved by making an ExecutorService in AiChatClient. The request to AI runs on the background thread of this executor. Once the result arrives, a Handler associated with the main Looper dispatches a callback on the UI thread. It made the user interface work while awaiting the arrival of the service

API Failure and Disconnection from the Internet

An internet-connected functionality should not be certain of an outside system's proper functioning. The call may not succeed due to the absence of internet connection, timeout, empty response, and service disconnection. This was dealt with by implementing exception handling and fallback-response in case of failure of the call from the internet. In case of unsuccessful online calls, Aura generates a predefined response according to key words from the call. Consequently, the functionality will continue working without crashing. Timeouts for connecting and reading have been implemented as well so as not to wait forever for the response.

Prevention of Multiple Requests

The user could press the Send button multiple times while Aura is forming its answer. This could lead to duplicates, extra requests, and confusion about conversation flow. The issue was addressed via the isWaitingForReply Boolean flag. It goes True right after the message is sent and goes back to False once the callback completes. The Send button also relies on it to determine if it should be active.

Feedback during Waiting Time

It takes several seconds to get an answer from the AI. Without any visual feedback, the user might think that the app does not work anymore. The temporary "Aura is typing..." bubble was

introduced before the network request. It disappears once the callback is completed. This provides instant feedback that the user's message is accepted and processed.

Handling Seven Pattern Categories

The Patterns section included seven categories, each needed a title, a snapshot, metrics, descriptions, and suggestions. Creation of seven separate Activities would have resulted in lots of code duplication. The problem was solved via the reusable PatternDetailActivity. All the categories have their own PatternContent objects. The Activity gets a category ID and creates its interface based on the chosen object.This reduced duplication and improved maintainability.

Designing Empty States with Insufficient Data

While most of the insight categories should be based on historical data like mood history, sleep history, journaling patterns and wellness activity, the current project does not have sufficient integrated historical data for calculating each of those insights. Rather than providing inaccurate figures, the screen uses screenshots from a prototype and empty-state text to explain why there is no information. The Journal Patterns insight, for instance, lets the user know that themes will become available as he/she keeps adding journal entries. Possible suggestions are shown to show how the feature will become helpful with more data.

Consistency with the Existing Codebase

Since the project already had separate Activities and bottom-navigation rather than using Fragments or navigation components, implementing a whole new architecture for my features would make the process harder.I thus decided to stick with the existing code base which consisted of an Activity and explicit Intent pattern even though implementing a better architecture would help reduce redundancy within a bigger application.

7.3. Juri's Contribution

When creating the authentication and Profile features, there were many difficulties that arose. These difficulties mainly centered on the navigation of Activities, validation of user inputs, UI consistencies, and incorporating my code into the team's repository. The problems have been solved manually, repeatedly running the application on the Android Emulator and little help from Generative AI.

Activity Navigation and Integration

The biggest technical difficulty was making connections between these views: Login, Sign Up, Forgot Password, Profile, Settings, Achievements, and Challenges. While working on the first release of the application, it happened more than once that some buttons were non-functional due to the unfinished connections between Activities or the missing activity altogether.

The problem was solved through manual inspection of the explicit Intents used for the navigation and AndroidManifest.xml file. Then, the application was built anew, and each navigation process was tested once more on the Android Emulator. In addition, the generative AI was employed to determine possible reasons behind the problem and specify which files need to be checked.

Activity Registration in AndroidManifest.xml

```
24          <!-- Profile section screens. ProfileActivity is exported so it can
25               be launched directly from Android Studio for testing; the
26               other three are only reached by navigating from Profile. -->
27          <activity
28              android:name=".ProfileActivity"
29              android:exported="true" />
30          <activity
31              android:name=".AchievementsActivity"
32              android:exported="false" />
33          <activity
34              android:name=".ChallengesActivity"
35              android:exported="false" />
36          <activity
37              android:name=".SettingsActivity"
38              android:exported="false" />
39      </application>
40
41  </manifest>
```

Figure 40 : Profile Activities in AndroidManifest.xml

Input Validation

A difficulty that occurred was related to providing appropriate validation of the authentication forms. The Login and Sign Up forms required detection of empty fields, incorrect email format, and password discrepancy. Without appropriate validation, users could continue filling the form even if the entered data was incomplete and/or inappropriate.

The problem was solved with the implementation of validation conditions in Java. Each condition was checked separately to make sure that the appropriate error message was provided. For instance, the system is checking if the fields for email and password are empty, if the email has the correct format, and if the values for password and confirm password are equal.

Generative AI was applied to give an example of simple validation. But the code was further checked and corrected in order to fit the existing CalmU project. The validation was checked manually in the Android Emulator.

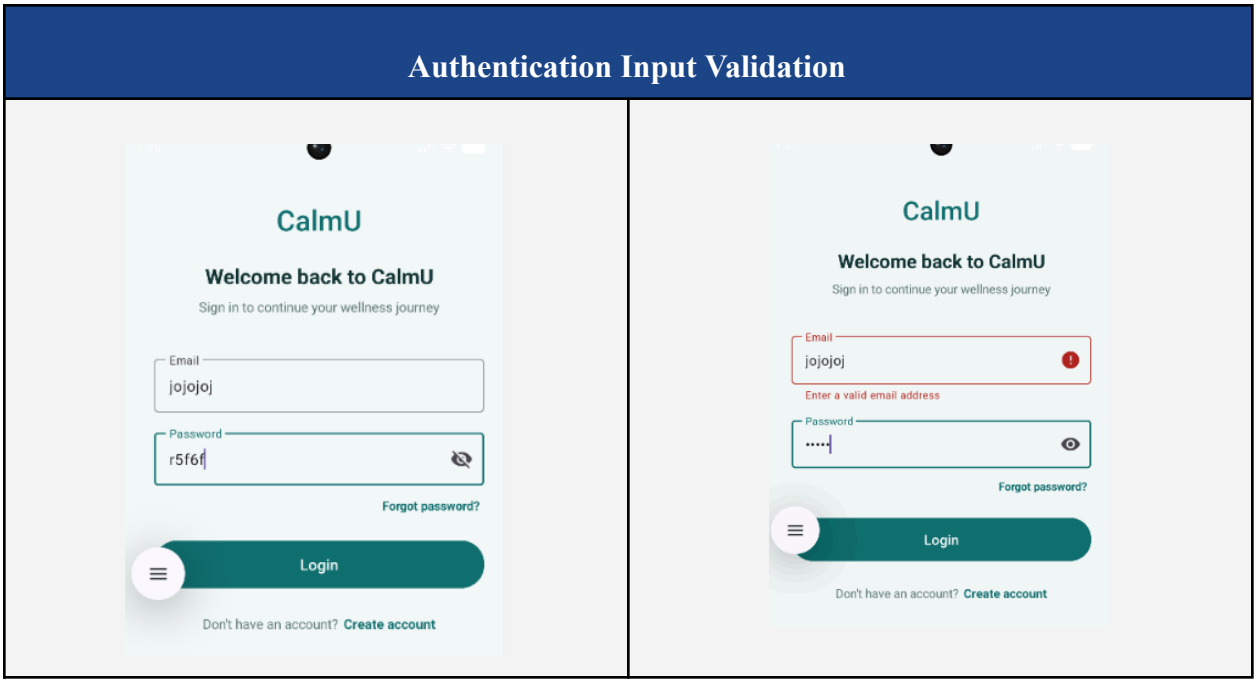


Figure 41 : Invalid email Validation test

User Interface Consistency

In addition, it was difficult to ensure that the application design is consistent on different screens. Some XML layouts seemed acceptable in the Android Studio preview window but showed some discrepancies once run on the emulator. These discrepancies included uneven spacing, overcrowded screen elements, confusing back buttons, and improper alignment of the components.

All the design problems have been solved through manual tuning of margin, padding, size of the component and alignment values of the XML layout. Also, the back buttons on the Profile-related screens have been optimized for clarity and ease of click-through. All the changes made have been tested on the emulator to see whether they have ensured that the layout is consistent with the CalmU design.

Generative AI technology has been utilized to provide suggestions for possible XML alterations and layout improvements. The generated suggestions were not accepted as is and were rather chosen on the basis of the way the app looks like in the emulator.

Version Control and Team Integration

Another difficulty was performing the authentication and Profile development without interfering with work of other team members. As multiple members were involved into development of an Android application at the same time, it could result in potential problems due to adding of changes to a single main branch, which would result either in conflicts or file replacement.

In order to avoid possible issues, the authentication and Profile development took place separately from the main GitHub branch. Before committing changes, all files under consideration were inspected, and those that were unnecessary for Android Studio were removed. Testing was performed before pushing changes to the remote repository.

As the team created a new branch that included the integrated project, the related work was reviewed to make sure that navigation and layouts were working properly. In this way, the repository could be organized and conflicts among team members could be minimized.

Emulator and Laptop Performance Challenge

The greatest difficulty that I had while developing the application was that of the performance of my computer with regards to Android Studio and Android Emulator. Right from the moment that Android Studio and the Android Emulator were launched, my computer began to heat up and emit noise from the fan. At times, the Emulator used to show a black screen and not respond at all.

To find the cause of this problem, I watched a number of tutorial videos and consulted with Generative AI on the causes of such a situation and solutions. I also sought assistance from the lecturer and followed the procedure that he showed us in the class. Following the lecturer's steps, I used the emulator for a longer time before it would show the same problem.

However, switching the virtual device would allow me to use the application again. For instance, in case when one of the Pixel devices showed a black screen, I would switch to another available device and start the application. Though this solution was temporary, it allowed me to keep developing and doing testing.

It seemed that the issue had to do with the large number of computer resources required for Android Studio and the Android Emulator. As I was not able to fix the limitation in laptop performance, I made sure to decrease the number of unnecessary applications and use the method provided by the lecturer and switched the emulator device whenever the black-screen issue appeared



Figure 42 : Switching Emulator Devices To Fix The Black-Screen Issues

In conclusion, all issues were solved through trial and error. Manual debugging was the major technique, while Generative AI offered some explanations and suggestions as well. The combination of those allowed solving the problems with navigation, validation, interface, and integration of the authentication and Profile features.

7.4. Bader's Contribution

While implementing the Today Dashboard, Mood Tracker, Mood Detail, and Sleep Logging features, several technical challenges were encountered beyond simply creating the user interfaces. The main difficulties involved ensuring that information was correctly transferred between screens, keeping the dashboard updated after user actions, and maintaining a consistent interface across multiple related screens. The challenges had to be thoroughly tested and modified during the development process. The major problems noticed and the solutions adopted are described below.

Passing Data Between Mood Tracker and Mood Detail

One of the first challenges was ensuring that the selected mood category from the Mood Tracker screen was correctly transferred to the Mood Detail screen. Each category showed different emotions and information, and therefore, when the user chose one of the mood cards, the correct category had to be passed. This challenge was solved by using Android Intents to pass the selected mood category between activities. These mood cards were associated with the corresponding categories and therefore the app was able to automatically open the proper Mood Detail screen based on the selected mood card. The implementation was tested repeatedly to ensure that every selected category displayed the correct information without any errors.

```
// Tapping any mood category card opens MoodDetailActivity, and
// tells it which category was tapped through an Intent extra.
openMoodDetailOnClick(R.id.card_vibrant, MoodDetailActivity.CATEGORY_VIBRANT);
openMoodDetailOnClick(R.id.card_settled, MoodDetailActivity.CATEGORY_SETTLED);
openMoodDetailOnClick(R.id.card_agitated, MoodDetailActivity.CATEGORY_AGITATED);
openMoodDetailOnClick(R.id.card_drained, MoodDetailActivity.CATEGORY_DRAINED);
```

Figure 43: Intent extra used to pass selected mood category from Mood Tracker to Mood Detail.

Keeping Dashboard Information Updated

Another challenge was ensuring that the Today Dashboard always displayed the latest mood and sleep information after the user completed an action. Users might have been able to read stale information without having to refresh the displayed data when they reopen the application. To address this, the dashboard was updated immediately after successfully saving the mood or sleep data. Multiple manual tests were performed to verify that the latest information appeared correctly after every update, providing users with accurate and up-to-date feedback.

Maintaining a Consistent User Interface

One of the key challenges during development was ensuring the same user interface on several screens. The Today Dashboard, Mood Tracker, Mood Detail, and Sleep Logging features were closely related and required a consistent look and feel to prevent user confusion. Consistent layout, colour, typography, spacing and navigation behaviour were applied throughout all implemented screens as a remedy to this challenge. Regular interface testing and improvements were conducted to guarantee a seamless and intuitive user experience.

8.0. Reflection and Future Work

Development of CalmU gave each team member hands-on experience with Android development beyond what was covered in earlier coursework, from working within an existing shared codebase to picking up new APIs and tools specific to their assigned feature. This section reflects on what was learned, the skills gained, and the limitations encountered during development. The following subsections detail each member's individual reflection and proposed future work.

8.1. Saad's Contribution

My main contribution to CALMU was building its Wellness Feature Hub from start till the end. Which was done from the main hub down to its three fully working exercises which were (Breathing, journaling, Meditation). This taught me several concrete lessons about development in Android Studio for example: working within an existing codebase, the gap between a feature looking finished and actually done completely. These are reflected below along with suggestions for where the feature could go next.

What was Learned

- Most important lesson learned was that a feature is not done if its UI looks right. But, its done only when its state is managed correctly across the Android Lifecycle. The breathing timer and the meditation audio playback both were dependent on a state that changes every second while the screen is open. and both of them needed clear cleanup when the screen was left or they would keep running or (in the audio case keep making sound). after the user had already navigated away it should be stopped. This wasn't obvious until it was actually tested by leaving each screen in the middle of a session.

- A second lesson was around database migration. Adding the (journal_entries) table required increasing the existing databases version number, and also the project's existing migration strategy. Dropping and creating all tables again on a version change would mean that step wiped all existing local data, including test accounts. It was useful because a lesson was learned about why real applications need a proper migration rather than a destructive one. Which would have cost us our time.

- Third lesson was about working by following my team convention rather then my own. Which is about the existing codebase architecture which is one Activity per screen, no shared navigation component isn't how I would have designed a new project, but consistency with the existing three screens mattered more than doing it "correctly" from scratch. And following that pattern was the more disciplined choice for a shared codebase four people are contributing to.

Skill Gained

Building Wellness Hub required me to work in areas of Android Studio which I never worked before with in depth such as: SQLiteOpenHelper and querying a new table, SharedPreferences for storing states such as (prompt rotation, shuffle counts, favourites), timers. And ViewPropertyAnimator for the breathing circle animation, MediaPlayer for real audio playback, and building a small custom View (CircularProgressView) by overriding onDraw(). I also gained practical experience in sourcing and verifying the licence terms of third party assets such as (illustrations and audio) before including them in a submitted project, which isn't something covered directly in earlier courseworks.

Future Improvements

Few genuine future improvements I would like to do with Wellness Feature are:

1. Wiring up real Wellness usage such as (sessions completed and streaks) with a pattern screen inside the App for Mindful Patterns recorded for the user. This would connect two separate parts of the app. This would increase the engagement of the user in CalmU.

2. Setting up a premium version of the app in which user can buy a membership and this would enable Journals “Upgrade” button (see figure A.2 inside appendix A) and several settings such as rows which are currently non functional placeholders. This would let users enter unlimited entries inside the journal. Which will again increase the engagement and interest of the user with the application. And would allow the development team a potential revenue stream.

3. More techniques for breathing can be added inside the Breathing exercise to increase the interest of the user for a longer time.
4. Favorite option is only inside meditation for sound tracks; it could also be used inside breathing techniques.
5. Increasing number of soothing Sound tracks inside meditation hub.
6. All wellness hub data is Local only right now. Like (SQLite and SharedPreferences); it doesn't survive a reinstall or sync across devices, which will be a potential development for the future.

8.2. Saimon's Contribution

Reflection

Implementing Aura AI Chat Companion has helped me understand how to write asynchronous code for Android better. Prior to adding the functionality, everything revolved around sending a request and showing the response. However, during the development process, I realized that the aspects like network connection, threads, user experience, and error handling are equally important. Now, I know how to use the ExecutorService to shift long operations into another thread and how to use a Handler to return results to the interface safely. Additionally, my knowledge of callback programming and managing interface state like the typing state and disabled Send button has been enhanced. The implementation of the fallback response feature showed how important it is to handle graceful failures. The feature should not depend on the external service being always available. Although the fallback response option is not as flexible as generated responses, it makes sure that the feature does not stop working during network issues. Working on the Patterns made me understand more about reusability in interfaces. In fact, the various categories of patterns could all have been developed as individual screens. But it was better that way because you get to see how data objects, Intent extras, and dynamic views can be used together to show different information without duplicating everything in terms of classes and layouts. Apart from developing my knowledge about patterns and interfaces, the project taught me more about Java, XML Layout Integration, Explicit Navigation, Debugging and Git collaboration. This is because I had to work on the project in an existing codebase by an existing team.

Limitations

The current version of the Aura feature does not preserve the full conversation history. The messages are added in the UI while the current Activity session but are not stored persistently in SQLite. As a result, opening the screen will bring back only the initial conversation that was pre-defined. The AI request includes only the current message and the name of the user. Previously sent messages are not included as the context of the conversation, and thus the Aura cannot recall the full conversation through several requests.The Patterns feature provides the prototype structure of the insight. Some values have zero or empty state since the feature is not integrated yet with long-term data from the Today, Mood, Journal, Wellness, Achievements and Challenges features.Manual testing has been mostly performed. Unit, interface and network-mocking tests have not been implemented.

Future Work

For a future version, the storage of Aura conversations in SQLite can happen using a special messages table, which will hold information about the user email, sender, the text itself and the time stamp. It would help to restore the conversation if the user comes back to the screen again.

Also, multi-turn context can be achieved by passing a limited history of recently exchanged messages along with each request, so Aura would be able to reply more consistent to follow-ups. The conversation length should be controlled to keep the request size reasonable.

Improvements to the network layer can include:

Better connection-status detection.

- A visible retry button.
- More detailed error messages.
- Request cancellation when the Activity closes.
- Secure backend-based request handling.
- Structured response validation.
- Improved lifecycle handling.

Patterns feature can be integrated with the real application data. The application can collect data about the sleep records from the Today screen, moods, journaling, meditations, breathings, achievements and challenges using a common data layer, and based on the data, the application could calculate such metrics as:

- Average sleep duration.
- Most frequently recorded mood.
- Mood changes over time.
- Journaling frequency.
- Mindfulness minutes.
- Current activity streak.
- Weekly completion rate.
- Goal progress.

Charts could have been included for weekly and monthly progress analysis. The user could choose a period of interest and view a comparison of the current results against past periods. In the future, the architecture could be enhanced using MVVM, ViewModel, LiveData or StateFlow, Room database and shared navigation component. This way the separation of interface logic from data and testing the application would become more convenient.

Tests could be also written automatically. Unit tests could check the selection of patterns and fallback responses, and interface tests using Espresso could check the functionality of navigation, validation and messages. A mocked network service could be used to check successful requests, timeouts and failures without relying on an external service.

Overall, the prototype represents an initial version of an application that combines AI-based conversation and wellness analytics. In the future the prototype data could be converted into a personalized and evidence-based summary of user wellness while maintaining a clear line between a wellness companion and healthcare services.

8.3. Juri's Contribution

What was Learned

This project provided me with the opportunity to use Android Studio and GitHub for the very first time. Initially, the whole development process seemed foreign to me. Nevertheless, while developing the CalmU app, I managed to learn about the organization of an Android app structure and the function of its components.

I have been able to find out how Java activities control the behavior of the screen and how the XML files are used to design the interface. I was able to learn how to link the screens using Intent and how to implement basic validation of the inputs from the users.

Another crucial aspect of the process was how to utilize GitHub for a team project. Here I had an opportunity to practice such actions as branching, switching between branches, reviewing modified files, committing my work and pushing it to the repository. The process of branching gave me the opportunity to realize how developers could work on various features without interfering in other developer's work directly.

This project was very interesting for me, since it was possible to witness how an application is developing from scratch and evolving to its prototype. Moreover, this experience raised my interest in mobile application development.

Skill Gained

The most important skill which I developed is the creation of an Android application using Android Studio and other languages like Java and XML. During my internship, I got accustomed to developing activities and designing mobile screens and connecting different pages together.

Problem-solving skills were sharpened as well. Whenever there was a problem such as failure to open a particular screen or when the emulator failed, I would learn how to look at the code and try out various solutions by running the application again. Besides, I was able to ask for help through the aid of tutorial videos and Generative AI.

Another major skill developed during this project is GitHub. Before this project, I did not use GitHub for developing programs. At the end of the project, I felt more comfortable branching, committing and pushing changes to my code.

Finally, this project increased my software design knowledge. I realized that apart from performing correctly, a mobile application must be navigable, have readable content and layout and validation messages for users.

Future Improvements

The future improvement of this application can include the connection of this system to a secure database or cloud storage so that users can create accounts, log in securely and restore their password. The current screens are mostly used to illustrate the expected workflow of the application.

It would be possible to improve the functionality of Profile by adding the ability to edit the information, add a profile picture, and save changes. Achievements and Challenges can work properly by integrating them with user's actions in the application.

Further testing could be done by using physical Android phones and different screen sizes. Some automated unit tests and UI tests can be implemented to decrease the number of repeated manual tests in the future and update this application easier.

It would be also possible to make this application better by improving its performance and accessibility. Such features as dark theme, multiple languages support, large texts and accessibility labels can be useful to make CalmU more accessible to different users.

In all, this project gave me an excellent start in developing for the Android platform and in using version control. While there were many challenges, completing my portion has given me confidence and motivation to learn further.

8.4. Bader’s Contribution

Developing the Today Dashboard, Mood Tracker, Mood Detail, and Sleep Logging features provided valuable practical experience in Android application development. It wasn't just about creating individual screens; the features had to work together as a cohesive part of the CalmU application. The development, testing and refinement process helped me gain a better understanding of interface implementation, how to navigate activities, how to transfer data between activities, debugging, and user-centred mobile application design. In this section I reflect on what I have learned, what skills I have developed and what practical changes could be made to incorporate in future development.

Reflect on the Development Experience

A major take-home message from this development project was the need to link up the features of application in a correct manner, rather than building each screen independently. The Today Dashboard was required to deliver a clear picture for the user, whereas the Mood Tracker, Mood Detail and Sleep Logging had to ensure seamless user experience and user flow. Using these interconnected screens helped me understand the basic working of activities in Android and the use of Android Intents, which helps in passing data from one activity to another.

It also helped me improve my hands-on experience in java, XML and Android Studio. The interfaces were structured in XML and the behaviour, navigation and interaction with the screens controlled by Java. After testing the application, I realised that even if an interface looks right, the functionality of each button, the information being passed through it and the way the application works needs to be tested.

One of the other key messages was that it is important to be consistent in the application. The colours, spacing, typography, structure and navigation behaviour of the implemented features had to stay consistent, as they all were a part of the same user journey. This helped me realize that building a mobile app is not solely about having a working app, but it's also about having a predictable, clear experience for the user.

Learning also flowed from debugging and performing manual testing repeatedly. By going through the Java and XML implementation, finding out why certain parts of it were wrong and retesting after changes I think my problem solving skills got better. Overall, it gave me greater confidence in developing and integrating various features of Android applications.

Future Improvements

There are a few changes that may be included in later versions of the included features. The Mood Tracker/Sleep Logging feature could be expanded with a secure archive that enables users to view their mood and sleep history over time. Simplifying the historical information using basic charts or summaries to make the changes in a user's day-to-day well-being easier to understand as well.

A Today Dashboard that shows more personalized information, such as mood and sleep data, recorded recently by the user could be improved. For instance, information that the user enters voluntarily could be used to show the dashboard relevant wellbeing reminders and/or appropriate activities. These recommendations are to be supportive, however, and not a medical diagnosis.

For future development, it would be nice to add an optional reminder for the user to regularly record their mood and sleep data. Users need to be able to switch on, off or adjust these reminders to suit their needs without them becoming disruptive.

Another potential enhancement would be to link the features to secure cloud storage. This might enable users to view their information on supported devices and ensure privacy and security of their information with the right authentication and access controls. Enhanced accessibility may be achieved through the use of larger text sizes, more descriptive labels, screen-reader compatibility and adequate contrast in the interfaces.

These are to be considered future development opportunities, and are not to be implemented in the current implementation. It will take more development, testing, privacy analysis and evaluation before it can be added to a full version of the CalmU application.

9.0 Conclusion

The CalmU project was truly successful in creating a viable Android prototype for the mental health of university students. The application includes many helpful features, such as user verification, a daily wellness home page, mood and sleep tracking, breathing exercises, journaling, meditation, a virtual assistant, patterns and insights, and a user Profile containing user settings, achievements, and challenges.

The key technical targets of the project were also met by implementing the application with Java and XML on Android Studio, by utilizing activities and intents to switch between different screens as well as by storing the prototype data locally and utilizing GitHub to handle the team contribution. Main screens, validation messages, navigation links, and layout were manually and emulator tested to ensure that they functioned properly.

CalmU was created based on usability by focusing on its navigation, data readability, colour tones and consistency. These design decisions enable students to use most of the wellness tools without getting lost in the process. The application is particularly suitable for use as a prototype because it offers mental wellness support in a simple and non-clinical manner, which is geared towards students.

A completed application, with functions still at prototype level, but for which a secure backend would be needed, along with real user accounts, further testing on varied device and further development, has a strong foundation for further development. In summary, CalmU exemplifies the potential of a mobile app to help bring simple mental wellness to university students, organizing and making it more convenient.

References

1. Android Developers. (2025, February 17). *About MediaPlayer*. https://developer.android.com/media/platform/mediaplayer
2. Android Developers. (n.d.-a). *Handler*. Retrieved July 17, 2026, from https://developer.android.com/reference/android/os/Handler
3. Android Developers. (n.d.-b). *Intents and intent filters*. Retrieved July 17, 2026, from https://developer.android.com/guide/components/intents-filters
4. Android Developers. (2026a, May 28). *Material Design for Android*. https://developer.android.com/develop/ui/views/theming/look-and-feel
5. Android Developers. (2026b, March 5). *Save simple data with SharedPreferences*. https://developer.android.com/training/data-storage/shared-preferences
6. Android Developers. (2026c, February 13). *SQLiteOpenHelper*. https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper
7. Android Developers. (2026d, March 5). *What to test in Android*. https://developer.android.com/training/testing/fundamentals/what-to-test
8. GitHub. (n.d.). *About Git*. Retrieved July 17, 2026, from https://docs.github.com/en/get-started/using-git/about-git
9. Miao, F., & Holmes, W. (2023). *Guidance for generative AI in education and research*. UNESCO. https://www.unesco.org/en/articles/guidance-generative-ai-education-and-research
10. Mixkit. (n.d.). *Mixkit license*. Retrieved July 17, 2026, from https://mixkit.co/license/

11. OpenAI. (2026, July 10). *Getting started with ChatGPT*. https://openai.com/academy/getting-started/
12. Oracle. (n.d.-a). *HttpURLConnection: Java SE 11 & JDK 11*. Retrieved July 17, 2026, from https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/net/HttpURLConnection.html
13. Oracle. (n.d.-b). *java.util.concurrent: Java SE 11 & JDK 11*. Retrieved July 17, 2026, from https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/package-summary.html
14. Pollinations.AI. (n.d.). *Pollinations API documentation*. Retrieved July 17, 2026, from https://gen.pollinations.ai/docs
15. United Nations Department of Economic and Social Affairs. (n.d.). *Goal 3: Ensure healthy lives and promote well-being for all at all ages*. Retrieved July 17, 2026, from https://sdgs.un.org/goals/goal3
16. unDraw. (n.d.). *License*. Retrieved July 17, 2026, from https://undraw.co/license
17. World Health Organization. (2022). *World mental health report: Transforming mental health for all*. https://www.who.int/publications/b/64274
18. World Health Organization. (2026, March 30). *Stress*. https://www.who.int/news-room/questions-and-answers/item/stress

Appendix A

GitHub Link :

https://github.com/saadarain11/assigment_app

Saad's Contribution Appendix

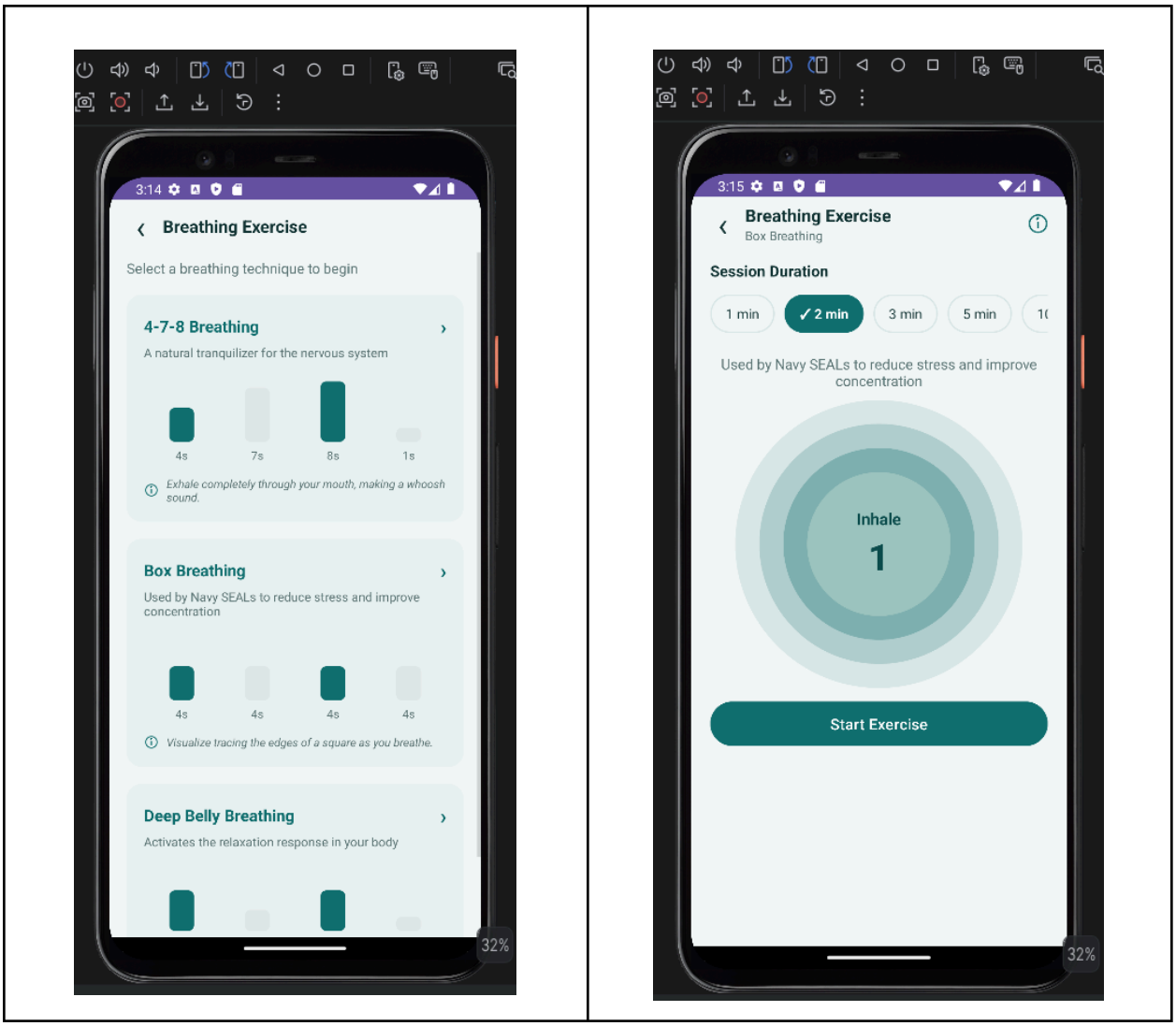


Figure A.1 : Breathing Exercise

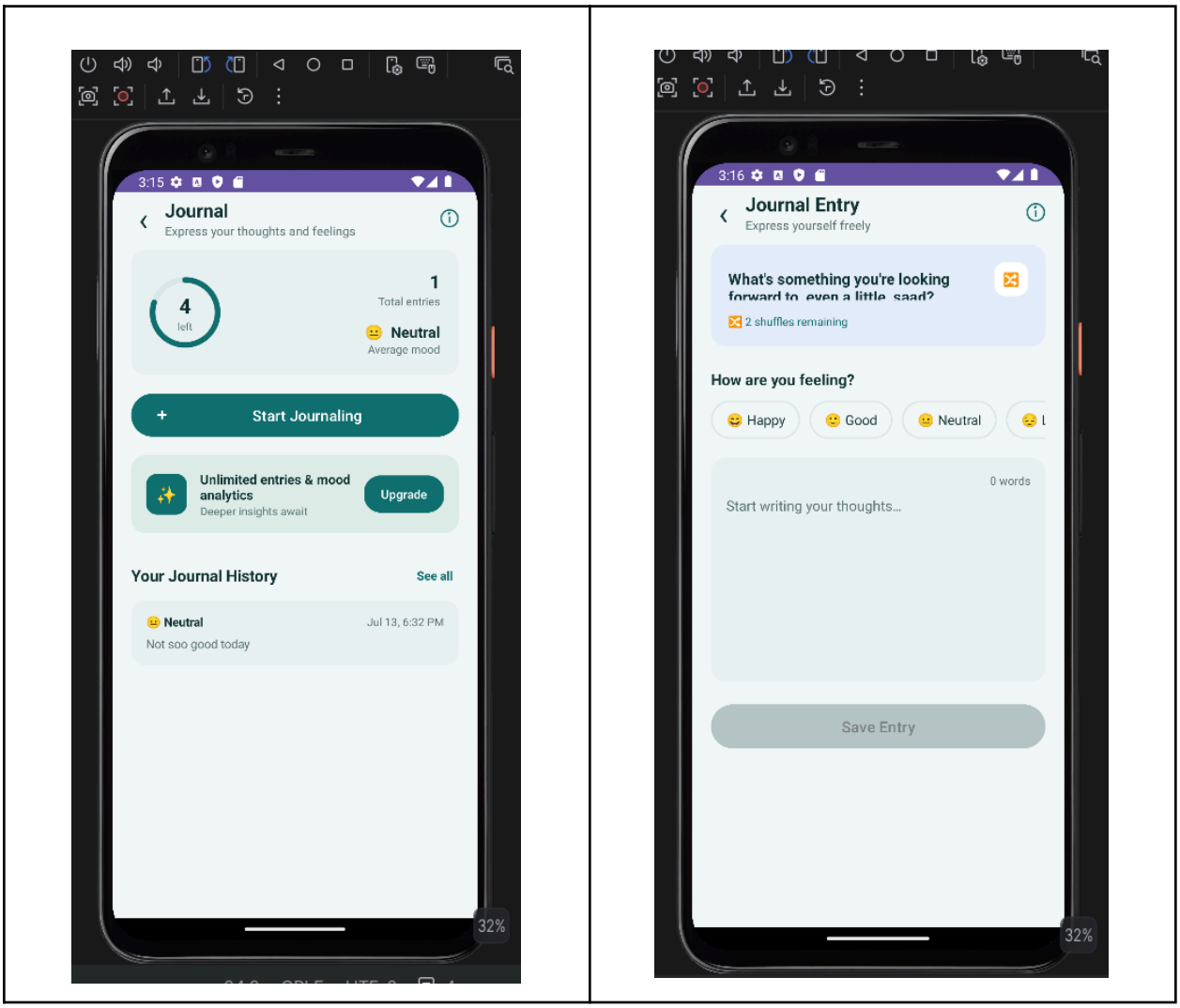


Figure A.2 : Journaling



Figure A.3 : Meditation

Saimon Contribution Appendix

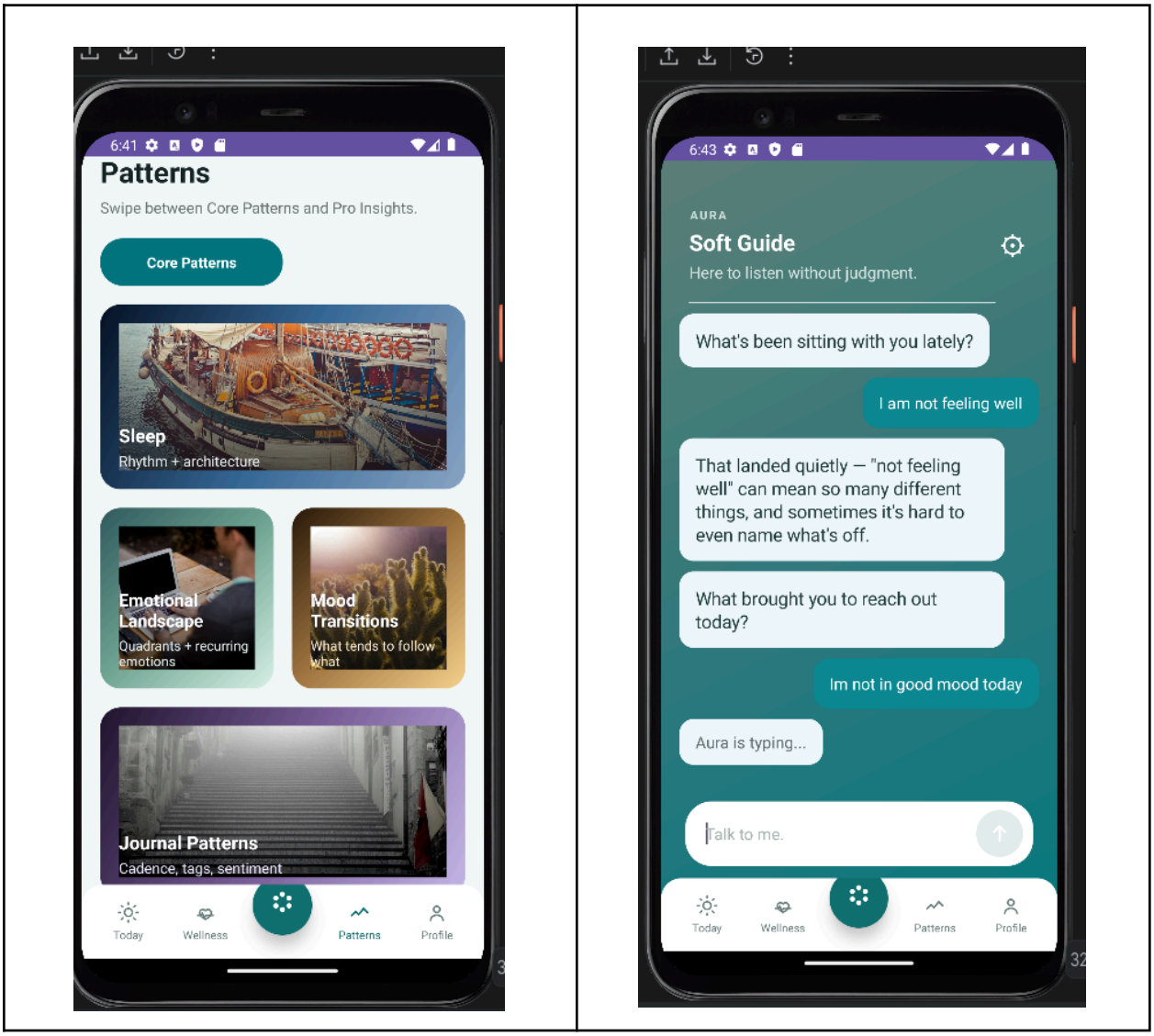


Figure A.4 AI Chat and Pattern Screen

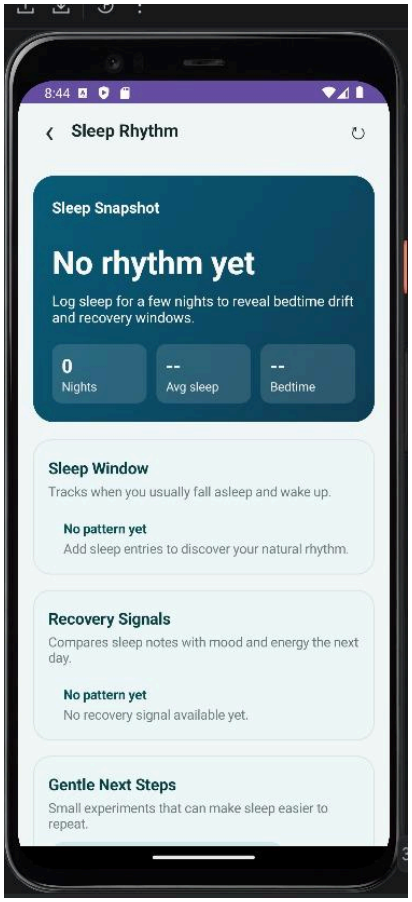


Figure A.5 One of the subsection of pattern page

Juri Contribution Appendix

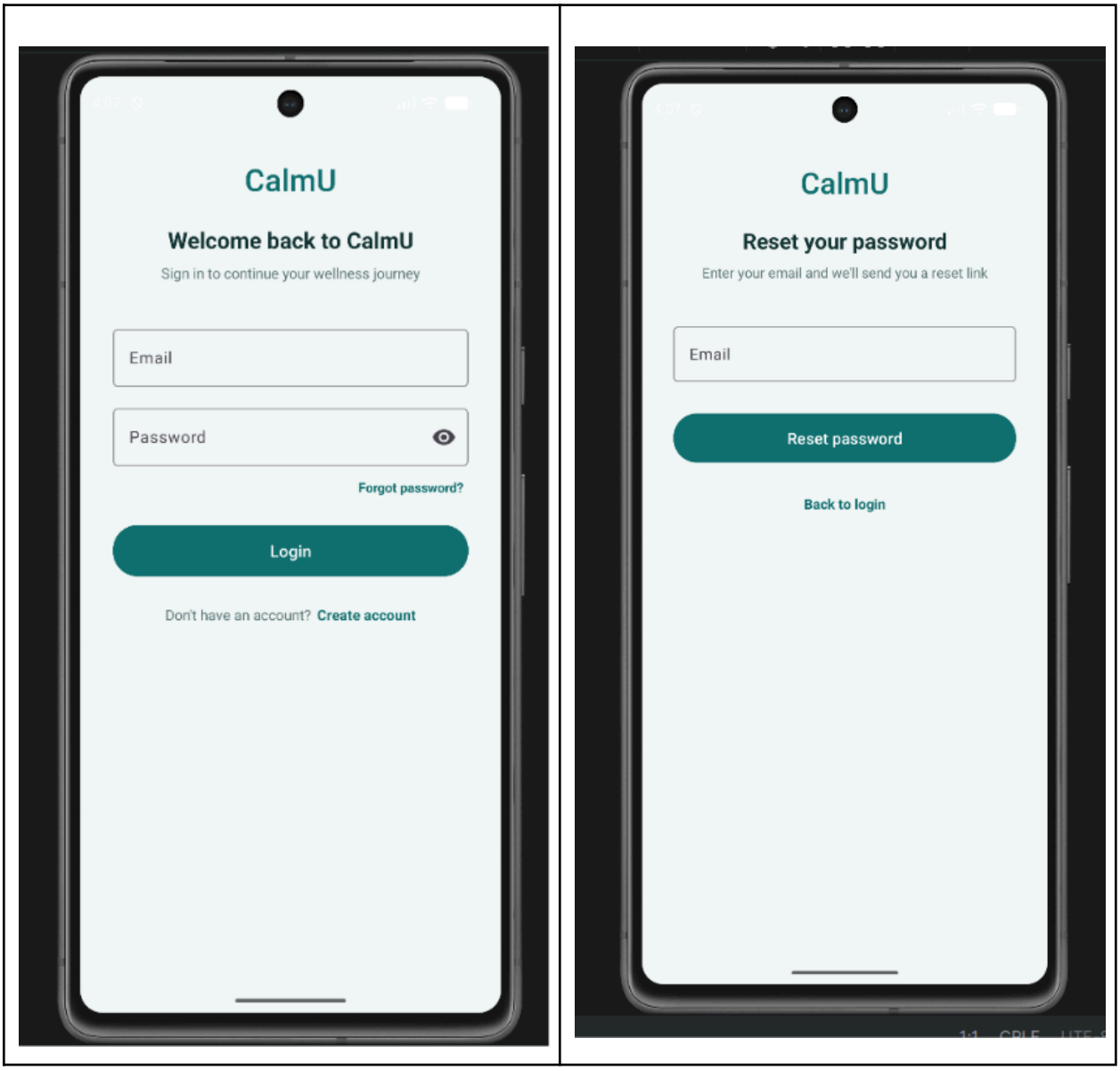


Figure A.6

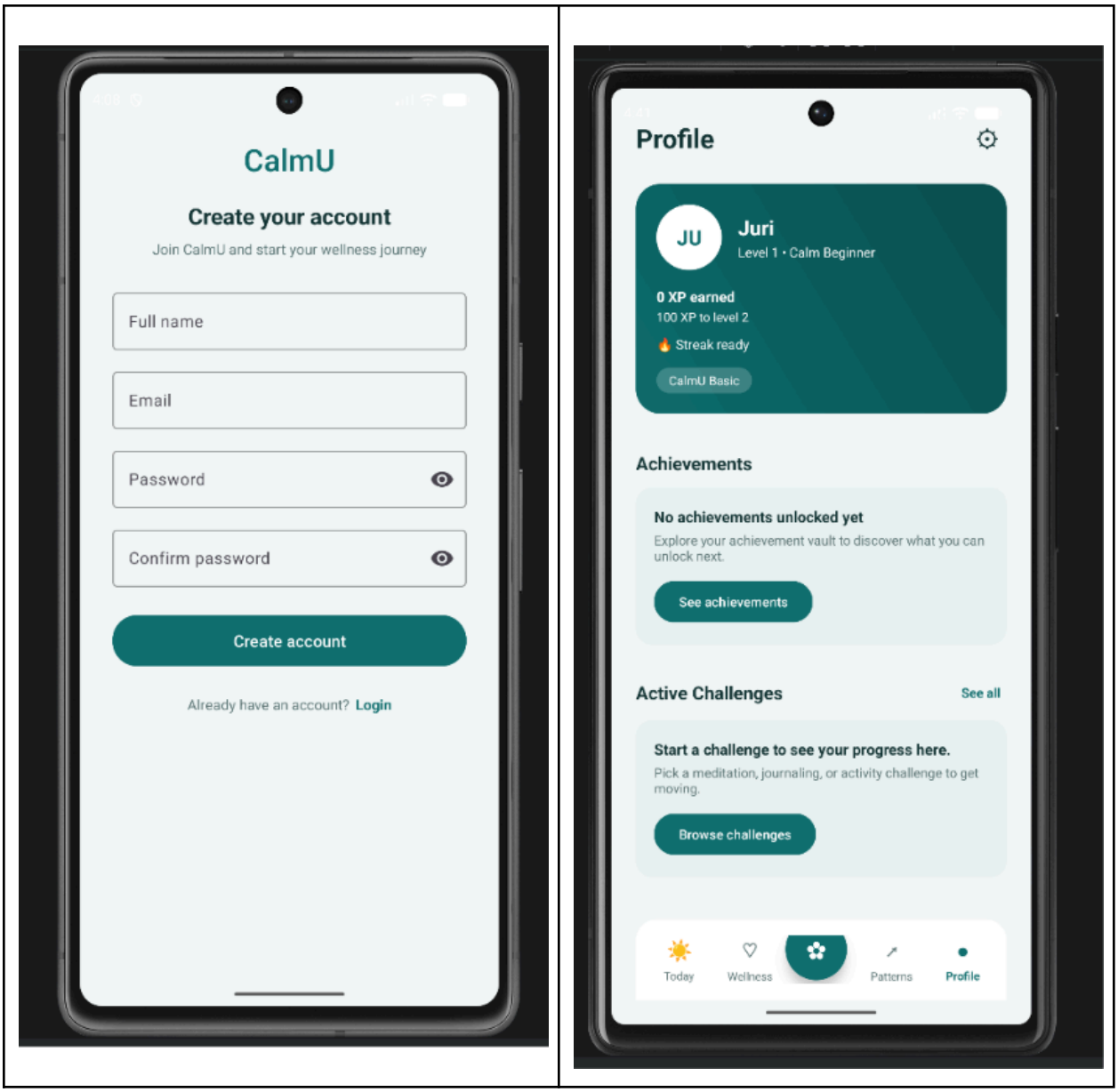

CalmU
Create your account
Join CalmU and start your wellness journey
Full name
Email
Password
Confirm password
Create account
Already have an account? Login
Profile
JU
Juri
Level 1 • Calm Beginner
0 XP earned
100 XP to level 2
Streak ready
CalmU Basic
Achievements
No achievements unlocked yet
Explore your achievement vault to discover what you can unlock next.
See achievements
Active Challenges
See all
Start a challenge to see your progress here.
Pick a meditation, journaling, or activity challenge to get moving.
Browse challenges
Today
Wellness
Patterns
Profile

Figure A.7

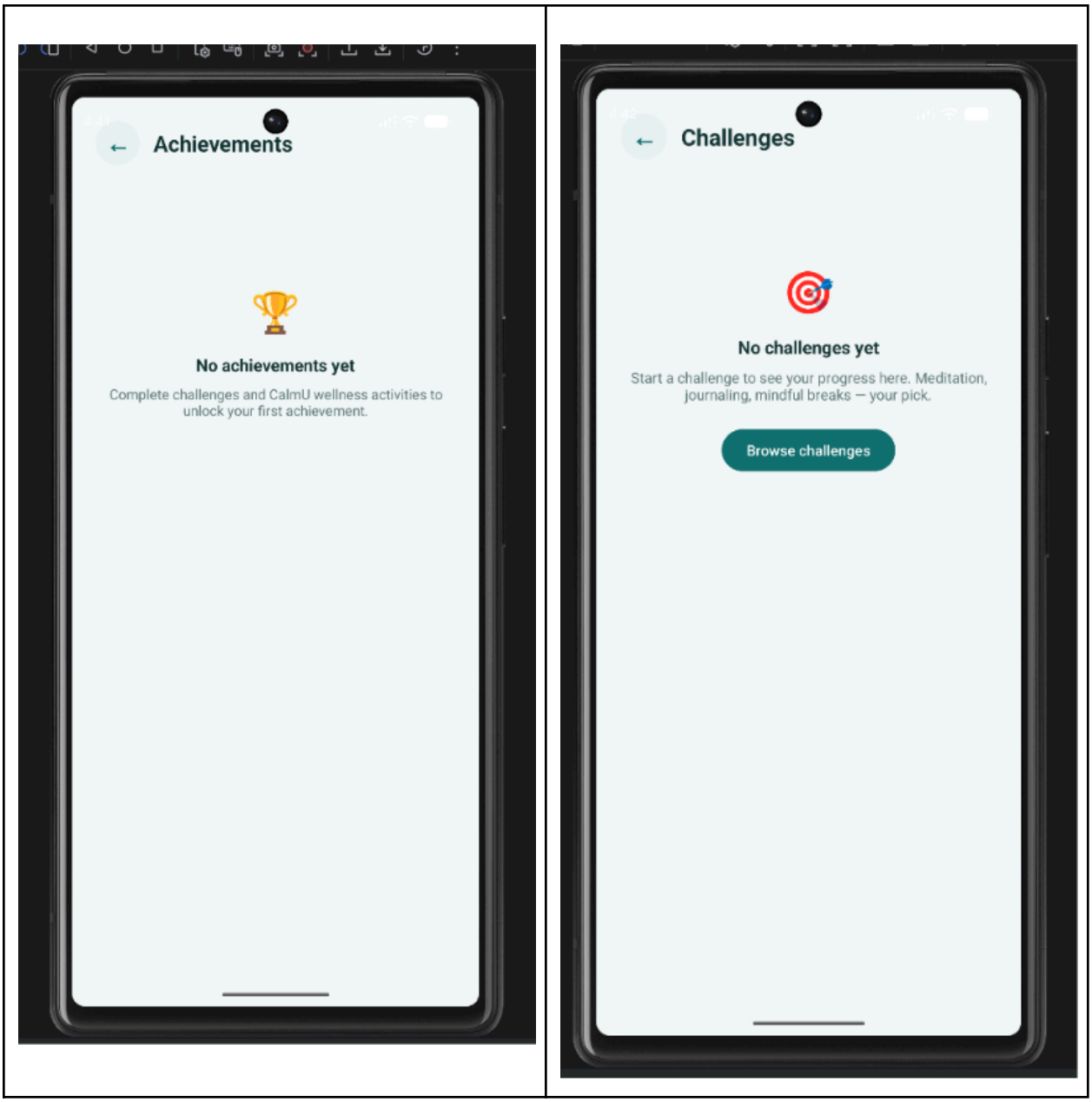

Achievements
No achievements yet
Complete challenges and CalmU wellness activities to unlock your first achievement.
Challenges
No challenges yet
Start a challenge to see your progress here. Meditation, journaling, mindful breaks — your pick.
Browse challenges

Figure A.8

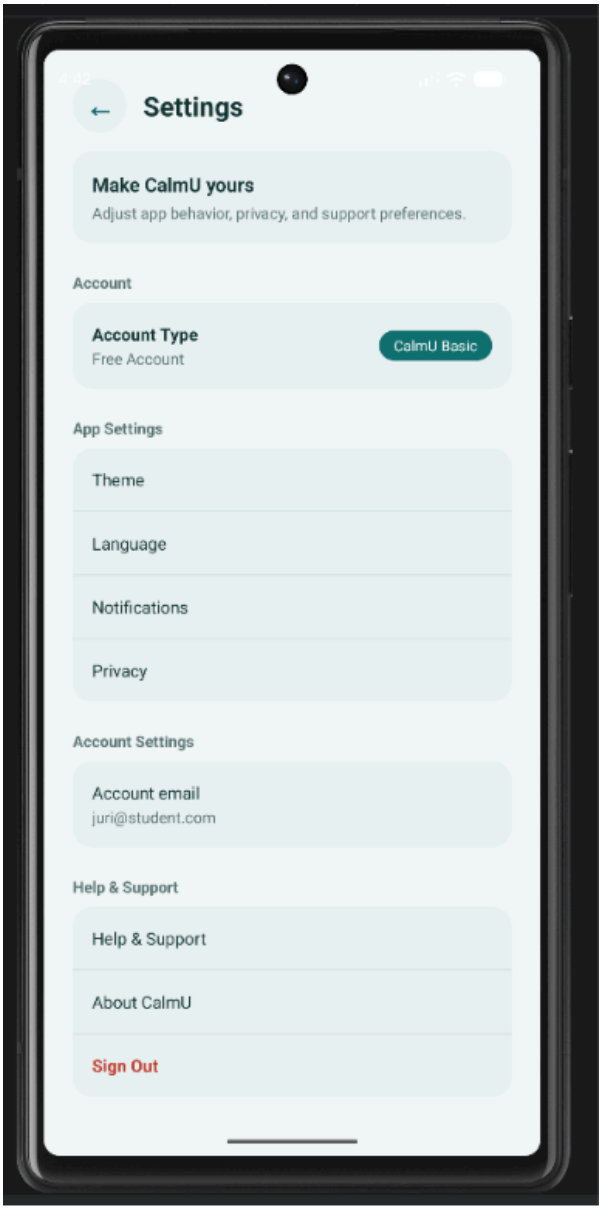

Settings
Make CalmU yours
Adjust app behavior, privacy, and support preferences.
Account
Account Type
Free Account
CalmU Basic
App Settings
Theme
Language
Notifications
Privacy
Account Settings
Account email
juri@student.com
Help & Support
Help & Support
About CalmU
Sign Out

Figure A.9

Bader Contribution Appendix

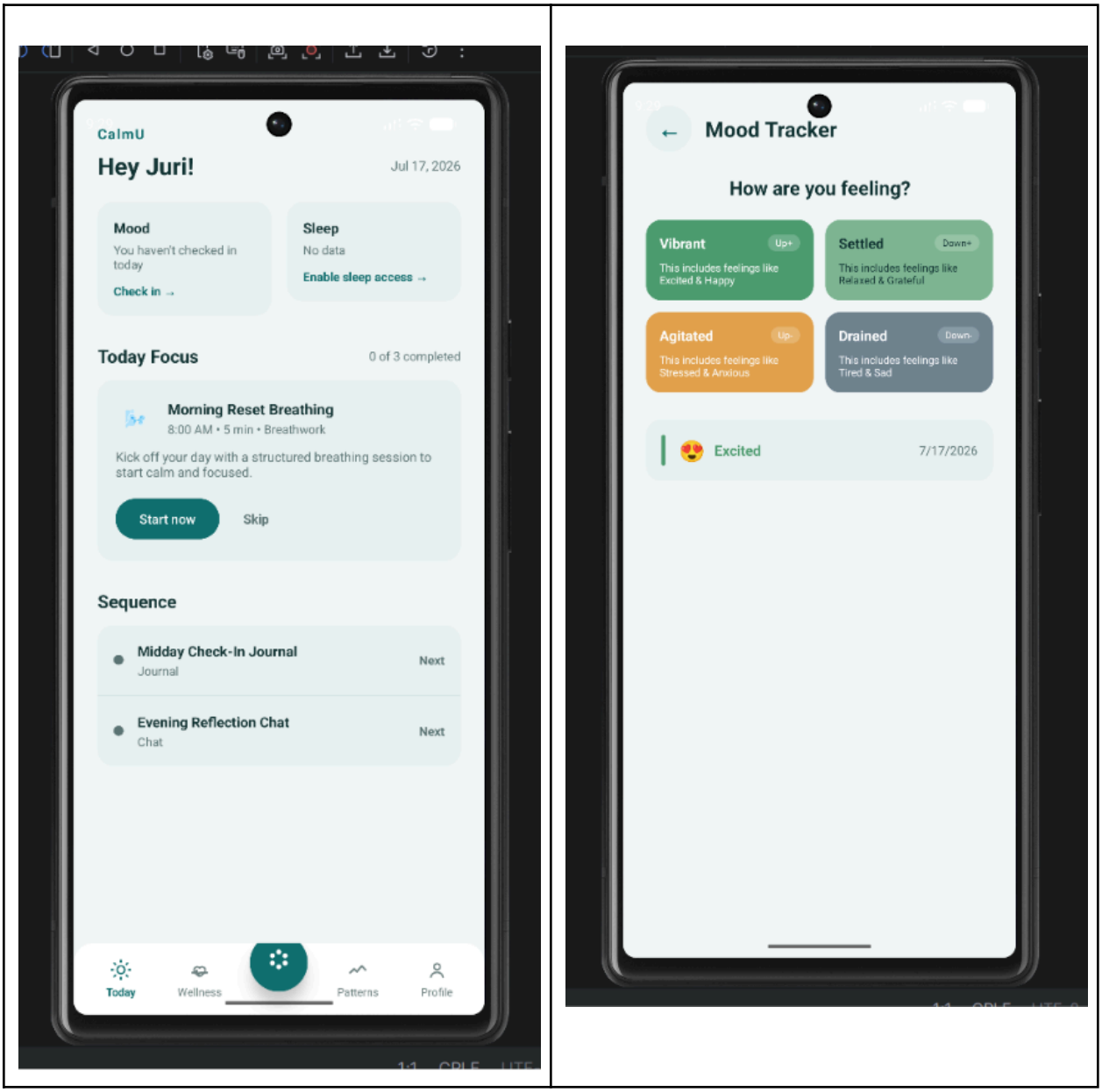


Figure A.10

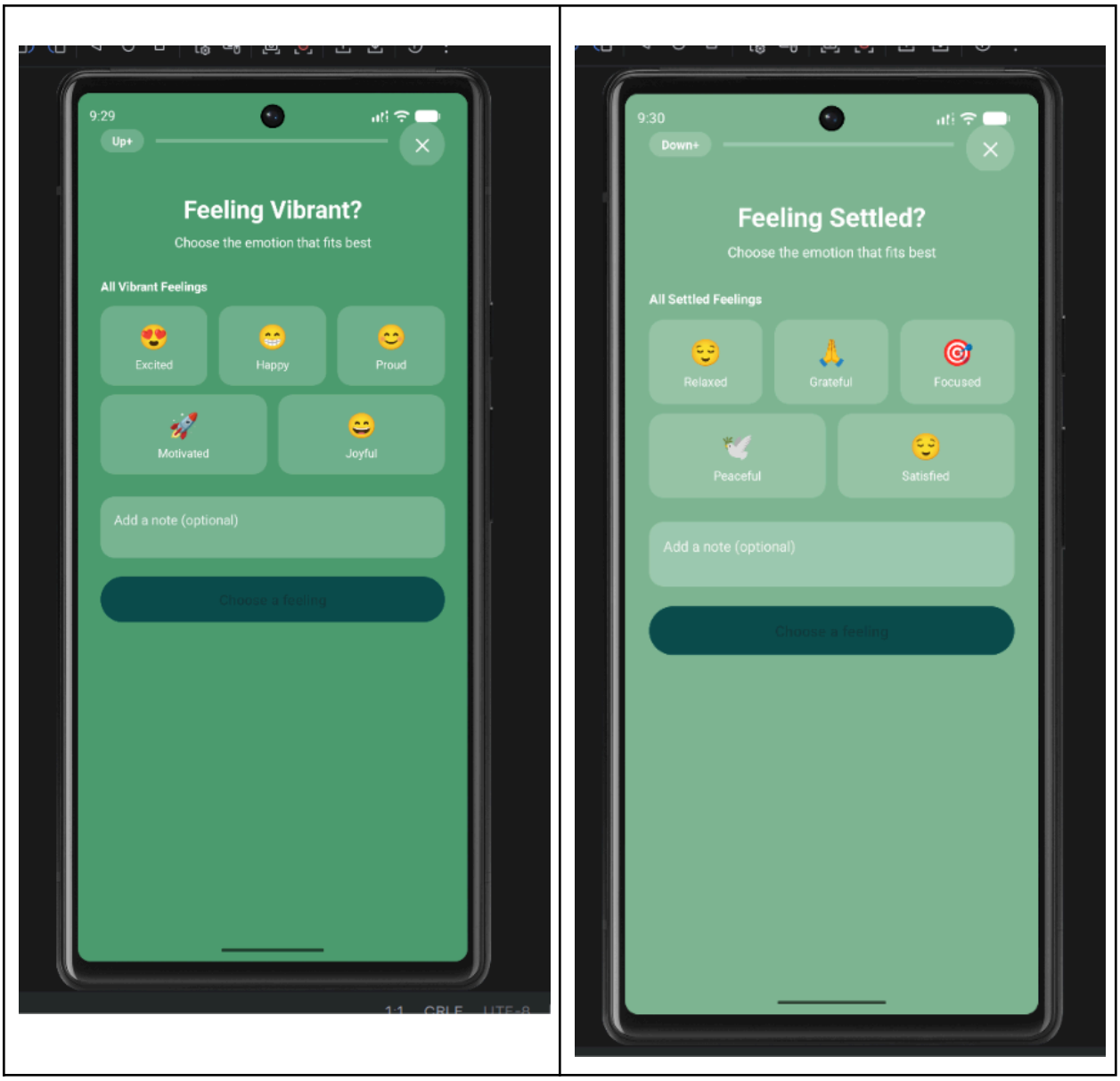

9:29
Up+
Feeling Vibrant?
Choose the emotion that fits best
All Vibrant Feelings
Excited
Happy
Proud
Motivated
Joyful
Add a note (optional)
Choose a feeling
9:30
Down+
Feeling Settled?
Choose the emotion that fits best
All Settled Feelings
Relaxed
Grateful
Focused
Peaceful
Satisfied
Add a note (optional)
Choose a feeling

Figure A.11

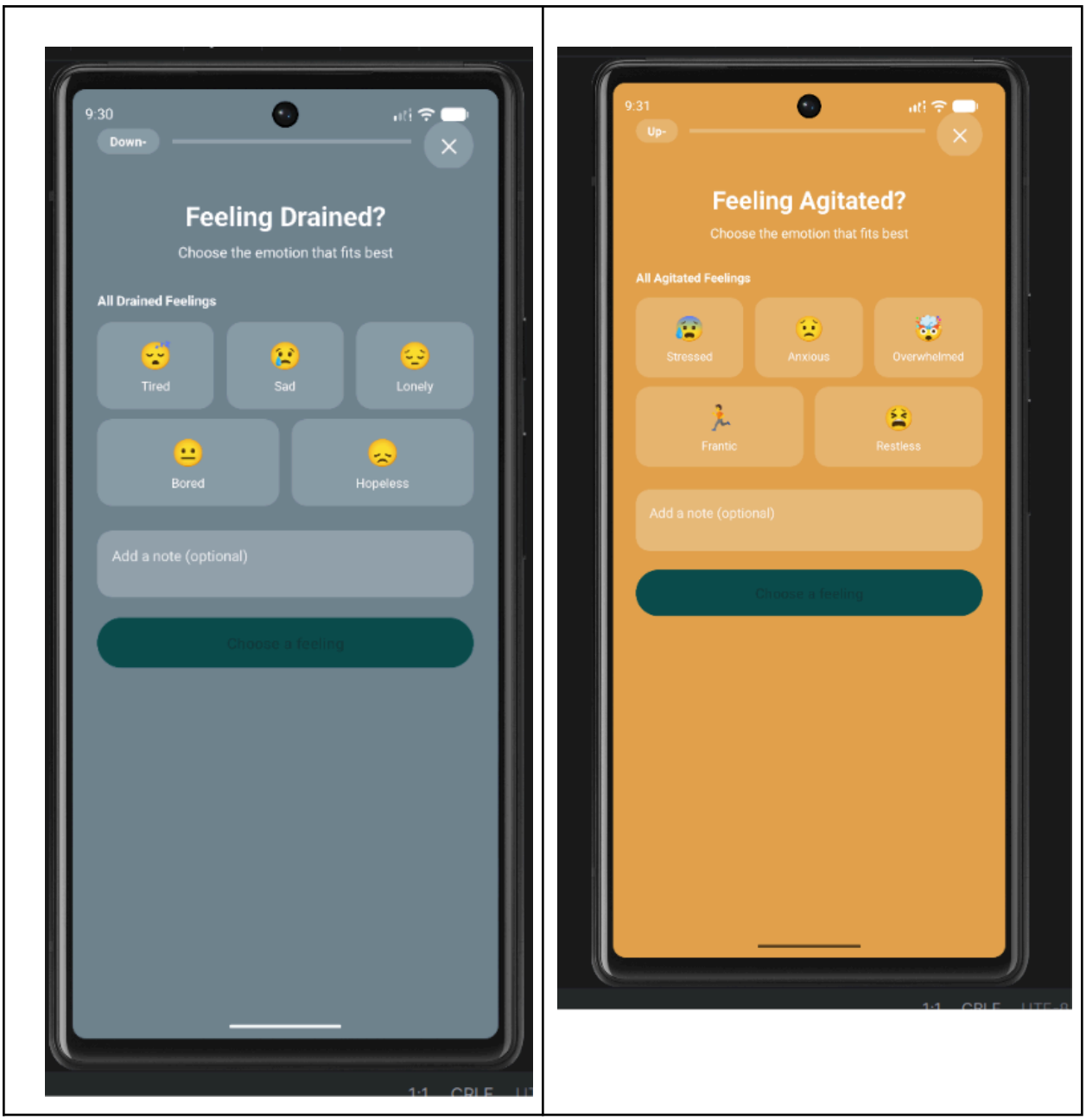

9:30
Down-
Feeling Drained?
Choose the emotion that fits best
All Drained Feelings
Tired
Sad
Lonely
Bored
Hopeless
Add a note (optional)
Choose a feeling
9:31
Up-
Feeling Agitated?
Choose the emotion that fits best
All Agitated Feelings
Stressed
Anxious
Overwhelmed
Frantic
Restless
Add a note (optional)
Choose a feeling

Figure A.12

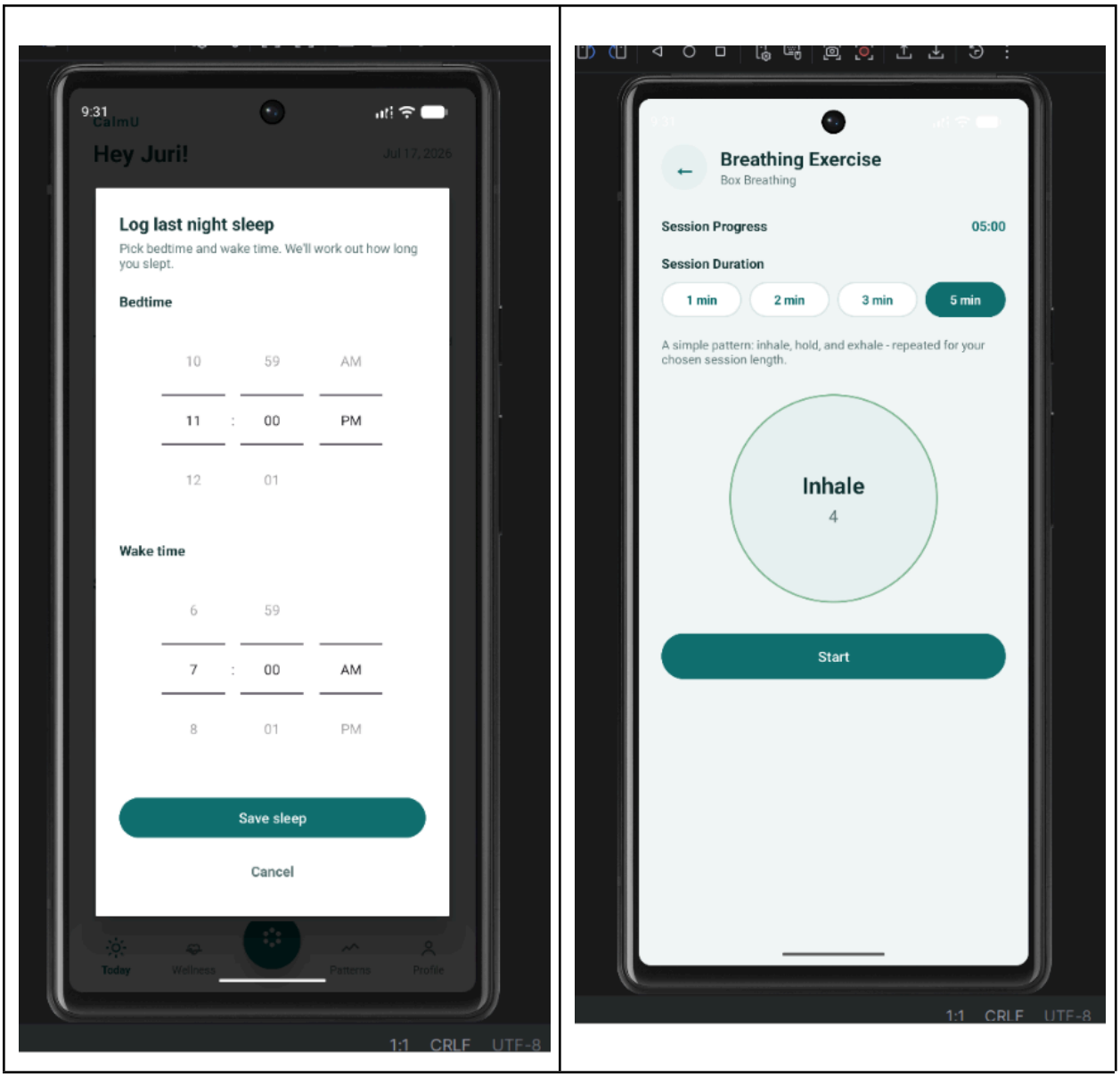

9:31
CalmU
Hey Juri!
Jul 17, 2026
Log last night sleep
Pick bedtime and wake time. We'll work out how long you slept.
Bedtime
10 59 AM
11 : 00 PM
12 01
Wake time
6 59
7 : 00 AM
8 01 PM
Save sleep
Cancel
Today
Wellness
Patterns
Profile
1:1 CRLF UTF-8
Breathing Exercise
Box Breathing
Session Progress
05:00
Session Duration
1 min
2 min
3 min
5 min
A simple pattern: inhale, hold, and exhale - repeated for your chosen session length.
Inhale
4
Start
1:1 CRLF UTF-8

Figure A.13

1

ORIGINALITY REPORT

3%	2%	0%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Taylor's Education Group Student Paper	1%
2	arxiv.org Internet Source	<1%
3	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1%
4	doczz.com.br Internet Source	<1%
5	libros.ubu.es Internet Source	<1%
6	github.com Internet Source	<1%
7	Submitted to King's Own Institute Student Paper	<1%
8	ouci.dntb.gov.ua Internet Source	<1%
9	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1%
10	Submitted to Oregon State University Student Paper	<1%
11	Submitted to Purdue University Student Paper	<1%
12	Submitted to Islington College,Nepal Student Paper	<1%
13	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
14	phamaplus.com.au Internet Source	<1%