BetterU: Productivity and Well-Being Mobile Application

By

Chia Ming Yi

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
KUALA LUMPUR

ACADEMIC YEAR
2025/26

BetterU: Productivity and Well-Being Mobile App: Goals Assistance(shared), Emotion Diary Note, Focus Timer, User, Report

By

Chia Ming Yi

Supervisor: Ms Yeoh Kar Peng

A project report submitted to the

Faculty of Computing and Information Technology

in partial fulfillment of the requirement for the

Bachelor of Information Technology (Honours)

Faculty of Computing and Information Technology

Tunku Abdul Rahman University of Management and Technology

Kuala Lumpur

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

_____

Chia Ming Yi

ID: 24WMR04090

Abstract

The BetterU mobile application is developed to address common challenges faced by students and office workers, including poor task management, cognitive overload, emotional stress, and social isolation. Existing productivity or mental well-being applications often focus on isolated functionalities, which limits their effectiveness in supporting users holistically. Therefore, this project aims to provide an integrated solution that combines productivity management with emotional well-being support to enhance users' efficiency, emotional awareness, and work–life balance.

The scope of BetterU covers five main modules: Goal Assistance (shared), Emotion Diary Note, Focus Timer, User, and Report. These modules collectively support intelligent task creation, focus session management, emotional reflection, user management, and analytical reporting. Core features include AI-assisted speech-to-task extraction, emotion-based diary analysis, focus session tracking with gamification elements, and personalised insights generated from user behaviour and emotional patterns.

An Incremental development methodology was adopted to allow continuous refinement through iterative requirement analysis, system design, implementation, and evaluation. Flutter and Dart were used for cross-platform mobile development, while Python with FastAPI supported backend services. Google Firebase was utilised for authentication, real-time database management, and cloud services. Artificial intelligence techniques, including Vosk and Whisper for speech-to-text processing, spaCy and SetFit for natural language processing, and transformer-based models for emotion classification, were integrated to enable intelligent system behaviour.

Integration testing was applied as the primary testing criterion to ensure that all system modules and internal components interact correctly under real usage conditions. The testing validated workflows such as user authentication, task creation, focus sessions, emotion analysis, and cross-module recommendations.

The results indicate that BetterU successfully meets its objectives by delivering a functional, intelligent, and user-friendly system. While limitations include dependency on network connectivity and model performance, the project demonstrates strong potential as a scalable platform that promotes productivity, emotional well-being, and sustainable work habits.

Acknowledgement

I would like to express my heartfelt gratitude to everyone who has supported, guided, and accompanied me throughout the journey of completing this project proposal. This accomplishment would not have been possible without the collective encouragement, wisdom, and kindness I received from many individuals around me.

First and foremost, I would like to extend my sincere thanks to my final year project supervisor, Ms. Yeoh Kar Peng, for her continuous guidance, encouragement, and insightful feedback. Her patience and dedication helped me stay focused and motivated, and her professional advice played a vital role in shaping the direction and success of this project.

I am especially grateful to my family, whose support has been the foundation of my strength. To my beloved mother, Chau Sook Khan, thank you for your love throughout this journey. Your support created a motivating environment that enabled me to stay resilient, even during moments of uncertainty and stress.

To my teammate, Lim Jun Wei, I would like to express my deepest appreciation for your cooperation, dedication, and commitment to our shared responsibilities. Your excellent communication, problem-solving skills, and consistent involvement contributed significantly to the overall quality and cohesiveness of our project work.

I would also like to extend my sincere thanks to my friends, Ong Yi Xin, Iris Chiam and Zenith Cheng, for always being there with words of encouragement, companionship and laughter. Your presence brought me comfort during difficult times, and your perspectives helped me find clarity when I felt overwhelmed. Thank you for listening, understanding and reminding me to stay grounded.

Last but not least, I would like to acknowledge all those who, in one way or another, offered support, shared a kind word, or simply believed in me along the way. Whether in the form of emotional support, academic feedback or quiet encouragement, each contribution has left a meaningful impact on my journey.

To all of you, thank you from the bottom of my heart.

Table of Contents

Chapter 1

Introduction

1 Introduction

This chapter introduces the BetterU mobile application, a wellness and productivity system designed to support users in managing their daily tasks while maintaining emotional balance. The chapter will provide an overview of the project background, outline its objectives, explain the problem statements that led to its development, and describe the significance and benefits of the system. It will also define the scope and limitations of the project, clarifying which features are included from the current development.

BetterU integrates productivity management with emotional well-being by offering modules such as intelligent goal assistance, emotion diary logging, focus timer sessions, personalized user settings, and progress reporting. These core functions aim to promote mental clarity, structured time management, and self-reflection. The system will support input through speech, text, and image, while also offering gamified elements and AI-powered suggestions to motivate users.

However, the current scope of the project does not include features such as real-time therapist consultation, social media integration, or cloud synchronization across multiple devices. Advanced predictive analytics for diagnosing mental health conditions are also beyond the intended scope, as the system focuses primarily on self-guided improvement and emotional tracking.

1.1 Objectives

The objectives of the proposed BetterU system are **centered around improving users' productivity and emotional well-being** through the functionalities of its core modules. These modules are designed to address the challenges of mental health, work-life balance, and productivity management. Below are the detailed objectives related to each module:

Goals Assistance Module (shared)

The Goals Assistance Module is designed to **reduce users' cognitive burden and enhance planning efficiency by providing intelligent task management**. It extracts actionable tasks from speech input, auto-generates personalized schedules, supports real-time progress tracking, and adapts to changes through dynamic rescheduling. Additionally, the module offers context-aware suggestions via the Smart Advisor to help users stay on track. As a result, users experience improved organization, reduced stress from unmanaged tasks, and better control over their daily responsibilities.

Emotion Diary Note Module

The Emotion Diary Note Module aims to **facilitate emotional awareness and personal reflection to support mental wellness**. It allows diary entries in multiple formats, including text, image, and speech, and includes mood tagging to contextualize the emotional state. By analyzing language patterns, the module detects emotional trends and stress indicators over time. Meaningful entries are saved in a momentary library for future review. This functionality helps users develop greater emotional self-awareness, track their mental state more clearly, and take proactive steps to improve their well-being.

Focus Timer Module

The Focus Timer Module is developed to **promote deep work and self-discipline** through structured and gamified focus sessions. Users can initiate distraction-free periods with app usage restrictions to prevent multitasking. The module incorporates gamification elements, such as treasure digging, to motivate consistent focus and task completion. In-app rewards, including avatar frames, visual themes, and challenge points, are provided as positive reinforcement. This encourages increased concentration, reduces mental fatigue, and enhances overall productivity.

User Module

The User Module **ensures a secure, personalized, and consistent application experience**. It supports essential account functions such as user registration, login, and profile management. User preferences, including notification settings, visual themes, and diary visibility, are stored

and applied to deliver a tailored interface. Through this module, users enjoy a seamless experience that aligns with their personal habits and minimizes unnecessary friction in daily use.

Report Module

The Report Module is responsible for **generating meaningful insights based on users' behavioral and emotional data**. It summarizes key activity metrics, including task completion rates, mood fluctuations, and focus durations. These data are presented through clear and intuitive visualizations, such as charts and graphs. The module highlights correlations between behavior and emotional well-being and offers self-assessment reports to support reflective decision-making. It empowers users to better understand themselves and maintain a balanced, informed lifestyle.

1.2 Problems

The proposed BetterU system is designed to address the following critical problems identified in the context of mental health and work-life balance in Malaysia:

1.2.1 Mental Health Crisis and Emotional Well-being

Problem Description:

- **Increasing Prevalence**
  - Mental health disorders, including depression, anxiety, and stress, are among the leading causes of disability globally, with Malaysia showing a similar upward trend. The Ministry of Health Malaysia (MOH, 2024) reports that approximately 29% of Malaysian adults suffer from mental health-related conditions, while 20.8% of individuals aged 16 to 24 experience psychological distress (NHMS, 2023). This highlights a significant increase in mental health issues across all age groups.
- **Stigma and Limited Access**
  - Mental health stigma remains deeply rooted in Malaysian society, discouraging many individuals, especially those in rural communities, from seeking treatment. The shortage of trained mental health professionals (only 0.25 psychiatrists per 100,000 people) exacerbates the problem, with uneven geographical distribution limiting access to mental health services in rural regions (Ahmad Adlan, 2022).
- **Socio-Economic Factors**
  - Financial insecurity, job instability, academic pressure, and social discrimination contribute to psychological distress. The COVID-19 pandemic further compounded these challenges, leading to widespread unemployment, income loss, and social disconnection, which in turn increased mental health-related issues (Bahmad, 2021; Abdul Yazid, 2023).

How BetterU Addresses the Problem:

- **Goals Assistance Module (shared)**
  - Reduces cognitive overload by automatically extracting tasks and generating personalized schedules. This helps users manage their responsibilities more effectively, thereby alleviating stress and promoting emotional stability.
- **Emotion Diary Note Module**
  - Provides a safe space for users to express and reflect on their emotions. Through mood tagging and emotional trend analysis, users gain better

self-awareness and can detect signs of emotional distress early, encouraging proactive mental health management.

- **Focus Timer Module**
    - Encourages users to engage in structured focus periods, reducing mental exhaustion and promoting emotional resilience through gamified rewards and positive reinforcement.
- **Report Module**
    - Offers detailed insights into users' emotional patterns and behaviors, helping them identify triggers and make informed decisions to improve their mental well-being.

1.2.2 Work-Life Imbalance and Burnout

Problem Description:

- **Increasing Work Demands**
    - Work-life imbalance is a major occupational health issue in Malaysia, with 55% of employees unable to achieve adequate balance due to rising employer expectations and technological overreach (MEF, 2023). The pervasive culture of overwork, particularly in high-pressure sectors like finance, healthcare, and education, contributes directly to burnout (HRDF, 2024).
- **Burnout Prevalence**
    - Burnout is characterized by emotional exhaustion, reduced personal accomplishment, and depersonalization. A national survey by PwC Malaysia (2024) revealed that 70% of respondents experienced high levels of stress, with many reporting feelings of detachment, fatigue, and disengagement from work. The normalization of remote work and digital communication platforms has further blurred the boundaries between work and personal life, exacerbating the problem.
- **Consequences**
    - Burnout leads to physical health problems, decreased productivity, higher absenteeism rates, and increased turnover. Dissatisfaction related to burnout is a key driver of attrition, with many workers seeking career changes or early retirement to avoid further psychological strain (PwC Malaysia, 2024).

How BetterU Addresses the Problem:

- **Goals Assistance Module (shared)**
    - Helps users organize their work and personal tasks more efficiently, preventing overcommitment and promoting better time allocation. The

module's smart advisor and real-time progress tracking ensure users maintain a balanced workflow.

- **Emotion Diary Note Module**
    - Allows users to document their emotional responses to work-related stressors, helping them identify patterns of emotional fatigue and take proactive steps to manage burnout.
- **Focus Timer Module**
    - Encourages structured work intervals and scheduled breaks, enhancing focus and preventing multitasking. The gamified reward system maintains motivation and promotes healthier work habits.
- **User Module**
    - Ensures a personalized experience, allowing users to configure settings to align with their working hours and preferences. This integration supports a seamless and less stressful user experience.
- **Report Module**
    - Provides visual and analytical overviews of users' work habits and emotional states, helping them identify signs of imbalance and adjust their routines for better work-life balance.

1.3 Advantages and Contributions

The proposed BetterU system provides multiple benefits by addressing two critical societal issues: mental health challenges and work-life imbalance. The application integrates intelligent modules that work cohesively to enhance productivity, emotional well-being, and daily structure.

1.3.1 Key Advantages

Smarter Time Management

- The Goals Assistance module extracts tasks from user input and auto-generates personalized schedules. This reduces cognitive overload and prevents burnout by promoting balanced daily planning.

Enhanced Emotional Awareness

- The Emotion Diary Note allows users to express emotions via text, image, or speech. It uses natural language processing to detect mood trends and promote reflection, helping users manage stress, anxiety, and emotional fatigue.

Improved Productivity and Focus

- The Focus Timer module encourages task completion by rewarding sustained concentration. Users experience increased efficiency through structured working sessions, reducing time wastage and supporting healthier work routines.

User Empowerment and Customization

- The User module enables users to personalize notifications, interface settings, and privacy preferences, ensuring a non-intrusive and user-friendly experience.

Insightful Feedback for Self-Improvement

- The Report module provides visual summaries of emotional patterns, focus trends, and task performance. This helps users make informed decisions to adjust habits and improve their overall well-being.

1.3.2 Contributions

BetterU offers a **holistic digital solution that bridges the gap between productivity tools and emotional support, effectively addressing both task management and mental wellness within a unified platform.** In an increasingly fast-paced and demanding world, individuals often struggle to maintain a healthy balance between personal responsibilities and professional commitments. BetterU directly responds to this challenge by promoting

structured self-management and emotional well-being, ultimately contributing to the development of a more mentally resilient and focused society.

By integrating artificial intelligence, the system enables intelligent task extraction and adaptive planning, which supports personalized and proactive goal setting tailored to individual user behaviors and preferences. This not only improves users' productivity and time management but also reduces cognitive overload and decision fatigue, which are common contributors to stress and burnout.

The application encourages regular emotional check-ins through diary entries, speech input, and mood tagging, supported by sentiment analysis. These features enable users to become more self-aware and emotionally literate, while also supporting the early detection of negative emotional patterns or signs of distress. This proactive approach to mental health empowers users to seek help early, potentially reducing the risk of long-term psychological issues.

BetterU also fosters a sustainable lifestyle by helping users organize their daily activities in a way that balances professional performance with personal well-being. It creates a safe, private, and user-centered environment that encourages consistent self-reflection and mindful living. The platform's integrated reporting features allow users to track their behavior and emotional trends over time through visual summaries, providing them with actionable insights and reinforcing positive habits.

From a social perspective, BetterU contributes to the broader goals of mental health awareness, digital well-being, and self-improvement. By providing individuals with tools to manage stress, stay motivated, and develop emotional intelligence, BetterU **supports healthier lifestyles and more resilient communities**. In the long term, such tools can ease the burden on mental health services, promote productivity in both academic and professional environments, and foster a more balanced, emotionally aware generation.

1.4 Project Plan

1.4.1 Project Scope

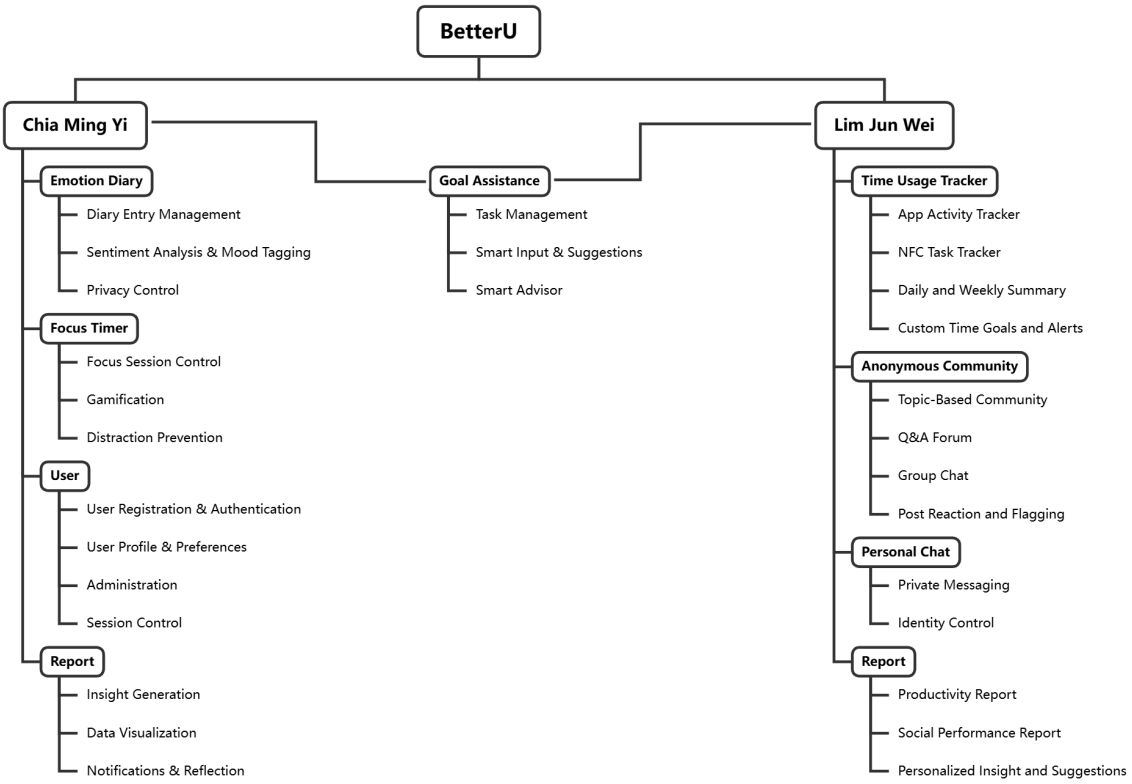


Figure 1.1: Hierarchy Chart for BetterU

Functional and Non-Functional Features

Functional Features

1.0 Emotion Diary Note Module

1.1 Diary Entry Management

1.1.1 The system shall allow users to create, edit, delete, and view diary entries.

1.1.2 The system shall support multiple entries "momentary" per day for quick reflections and future review.

1.1.3 The system shall support diary input via text, emojis, voice (speech-to-text), and image.

1.1.4 The system shall allow users to optionally attach images to diary entries.

1.1.5 The system shall automatically capture and display the creation date and time of each entry.

1.1.6 The system shall allow users to search entries by title, date, mood tags, or keyword highlights.

1.1.7 The system shall support sorting entries by creation date/time or mood.

1.1.8 The system shall support grouping entries by mood or by creation date range using a calendar view.

1.2 Sentiment Analysis & Mood Tagging

1.2.1 The system shall analyze diary content using AI-powered sentiment analysis (e.g., via Scikit-learn or NLP libraries).

1.2.2 The system shall automatically suggest mood tags based on detected emotional tone (e.g., Neutral, Happy, Sad, Anxious, Fearful, Angry, Mixed).

1.2.3 The system shall highlight keywords that indicate possible causes of stress, problems encountered, or sources of happiness.

1.2.4 The system shall allow users to manually edit mood tags and keyword highlights.

1.3 Privacy Control

1.3.1 The system shall offer optional PIN or biometric (e.g., fingerprint) protection for accessing or locking sensitive diary entries.

2.0 Focus Timer Module

2.1 Focus Session Control

2.1.1 The system shall allow users to select a task and specify a focus duration (15, 25, 45, 60 minutes, or custom).

2.1.2 The system shall support pause, cancel, resume, and reset controls during focus sessions.

2.1.3 The system shall support focus tracking tied to specific tasks for progress accumulation.

2.1.4 The system shall support phone flip detection to initiate or maintain focus mode.

2.1.5 The system shall display a visual heatmap (similar to GitHub contribution graph) to represent daily focus completion history.

2.1.6 The system shall allow users to view their session history, rewards earned, and personal progress trends.

2.2 Gamification

2.2.1 The system shall trigger start/end sound effects and visual animations during focus sessions.

2.2.2 The system shall reward users with virtual “treasures” upon completing a focus session, where both the probability of obtaining rare items and the total number of rewards shall scale with the session duration (e.g., longer sessions yield higher drop rates and multiple items).

2.2.3 The system shall allow users to collect virtual “treasures” and convert duplicate items into gems.

2.2.4 The system shall include a shop where users can exchange gems for cosmetic or premium features (e.g., limited-time avatar frames, wallpapers, feature trial days).

2.2.5 The system shall track and display daily focus streaks to encourage consistency.

2.3 Distraction Prevention

2.3.1 The system shall monitor app usage during focus sessions and pause the timer if apps outside the whitelist are accessed.

2.3.2 The system shall notify users with a warning if distraction rules are violated.

2.3.3 The system shall allow users to manage their app whitelist manually.

2.3.4 The system shall maintain a leaderboard showing top focus durations within the user's community (e.g., StudyTogetherCommunity) and globally (Overall Top 10).

3.0 User Module

3.1 User Registration & Authentication

3.1.1 The system shall allow new users to register by providing a username, password, email, gender, and occupation.

3.1.2 The system shall validate that the email is in the correct format and enforce a strong password policy.

3.1.3 The system shall send a One-Time Password (OTP) to the user's email for verification during registration.

3.1.4 The system shall allow users to log in using either their username or email together with their password.

3.1.5 The system shall provide a "Forgot Password" function that sends an OTP to the user's email for password recovery.

3.1.6 The system shall check for duplicate usernames and notify the user if the chosen username is already taken.

3.2 User Profile & Preferences

3.2.1 The system shall allow users to update their profile information, including profile image, avatar frame, username, gender, and occupation.

3.2.2 The system shall ensure that usernames remain unique when users update them.

3.2.3 The system shall assign user roles such as User, VIP, or Admin and restrict access to features based on these roles.

3.2.4 The system shall grant VIP users access to exclusive features, including unlimited AI functionalities.

3.2.5 The system shall allow users to select application themes, such as light mode or dark mode.

3.2.6 The system shall allow users to configure notification preferences, including the option to silence notifications.

3.2.7 The system shall allow users to enable or disable PIN or fingerprint protection for diary entries.

3.3 Administration

3.3.1 The system shall provide administrators with a dashboard that displays the total number of active users along with user retention metrics.

3.3.2 The system shall allow administrators to view and manage new requests for community group creation or joining.

3.3.3 The system shall enable administrators to review and moderate flagged posts, messages, and user accounts.

3.4 Session Control

3.4.1 The system shall manage secure login sessions using token-based authentication with automatic timeout and refresh mechanisms.

3.4.2 The system shall provide users with the option to log out manually and terminate all active sessions.

4.0 Report Module

4.1 Insight Generation

4.1.1 The system shall generate weekly, monthly, and yearly summaries that reflect the user's completed and pending tasks, focus sessions, and emotional records.

4.1.2 The system shall allow users to view and filter reports based on custom time ranges and selected data types, such as goals, focus duration, or mood trends.

4.1.3 The system shall provide a visual comparison between to-do tasks and completed tasks to help users evaluate their productivity.

4.2 Data Visualization

4.2.1 The system shall present data using various chart formats, including line graphs, pie charts, and bar charts.

4.2.2 The system shall visualize trends and correlations among task completion, focus time, and emotional states.

4.2.3 The system shall allow users to view progress and consistency through weekly, monthly, or yearly breakdowns in a calendar-style or timeline layout.

4.3 Notifications & Reflection

4.3.1 The system shall send users personalized summary insights via in-app notifications or email, if enabled.

4.3.2 The system shall provide motivational feedback, encouragement messages, or suggestions for habit improvement based on detected patterns and behavior gaps.

4.3.3 The system shall offer self-assessment prompts to encourage users to reflect on progress, stressors, and achievements.

4.3.4 The system shall allow users to export their reports and visual summaries as downloadable PDF files for personal tracking or external sharing.

5.0 Goal Assistance Module (shared)

5.1 Task Management

5.1.1 The system shall allow users to create, edit, delete, and mark tasks as completed.

5.1.2 The system shall support optional fields in task creation, including description, link, image attachment, and sub-tasks.

5.1.3 The system shall allow users to classify tasks into predefined or user-customized categories.

5.1.4 The system shall support task types: One-time or Repeat (with frequencies: daily, weekly, monthly, yearly).

5.1.5 The system shall allow users to set task priority levels from P1 (highest) to P4 (default, least important).

5.1.6 The system shall allow users to specify task timing: create date, occur date, and due date with preset options (Today, Tomorrow, Weekday, Weekend, Specific Date, or No Due Date).

5.1.7 The system shall allow enabling push notifications as reminders, triggered at the occurrence time or set intervals before (e.g., 10 minutes before).

5.1.8 The system shall support progress tracking based on sub-task completion (displayed as a percentage).

5.1.9 The system shall allow users to sort and filter tasks by category, priority, type, and due date.

5.1.10 The system shall support a calendar view and list view for task visualization.

5.2 Smart Input & Suggestions

5.2.1 The system shall extract task goals from natural language inputs using NLP (e.g., "Remind me to submit a project report tomorrow at 3 PM").

5.2.2 The system shall auto-suggest suitable time slots based on availability, habits, and workload.

5.2.3 The system shall recommend task categories and priority based on task keywords and past behaviors.

5.2.4 The system shall provide predefined templates for recurring task types (e.g., study, exercise, work).

5.3 Smart Advisor

5.3.1 The system shall analyze users' past task completion habits to suggest priority adjustments (e.g., start earlier, complete before usual fatigue hours).

5.3.2 The system shall detect schedule conflicts and suggest alternative time slots dynamically.

5.3.3 The system shall provide emotional and productivity-based reminders, such as taking breaks or relaxation suggestions.

5.3.4 The system shall learn long-term behavior trends to optimize future task planning (e.g., avoid late-night tasks, balance workload).

5.3.5 The system shall allow users to enable/disable Smart Advisor and choose whether to apply, ignore, or manually adjust its suggestions.

Non-Functional Features

Product

1.0 Availability

1.1 The system shall maintain 99.95% uptime, minimizing interruptions in daily use.

1.2 Scheduled maintenance shall only occur during non-peak hours and not exceed 2 hours per month.

1.3 In the event of unexpected Firebase service disruption, the app shall attempt automatic reconnection and restore functionality within 5 minutes.

2.0 Functional

2.1 The system shall ensure full feature accessibility and consistency across Android and iOS platforms.

2.2 All core modules (Goals Assistance, Emotion Diary, Focus Timer, Reports, and User Account) must function reliably on smartphones.

2.3 The system shall utilize Firebase services (e.g., Firestore, Authentication, Cloud Functions) for backend functionalities.

3.0 Usable

3.1 The app shall feature an intuitive and beginner-friendly UI, allowing users to navigate core features (e.g., setting goals, starting timers, writing emotion logs) without external help.

3.2 All UI components shall follow consistent design patterns as defined in the Figma design system.

3.3 The system shall support theme personalization for better user comfort and accessibility.

4.0 Reliable

4.1 The app shall support up to 10,000 concurrent users without noticeable performance issues using Firebase backend scalability.

4.2 The system shall prevent data loss by utilizing Firebase offline persistence and real-time synchronization to ensure local changes are synced when connectivity resumes.

4.3 Emotion logs, goal history, and focus sessions shall be backed up in real-time to cloud storage.

5.0 Flexible

5.1 The system shall support localization, including multi-language support, adjustable time/date formats, and culturally adapted content presentation.

5.2 The app shall allow easy updates of localized text through Firebase Remote Config or dedicated translation files.

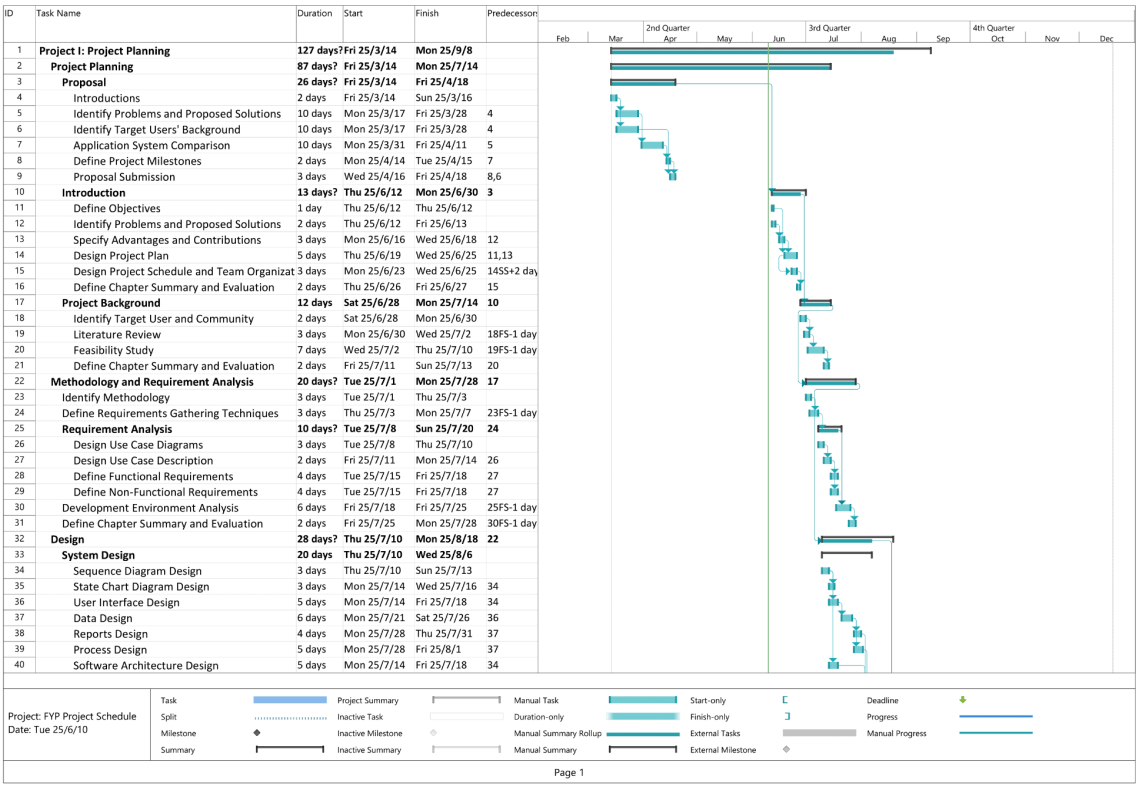
Organization

1. The system shall be developed using the Dart programming language with the Flutter framework.
2. The system shall use Firebase as the primary backend solution, including Firestore (database), Firebase Auth, and Firebase Cloud Functions.
3. Version control shall be managed via Git and hosted on GitHub, with team collaboration and issue tracking enabled.
4. The UI/UX design shall be created and iterated using Figma.
5. AI-powered features (e.g., emotion analysis, goal suggestion) shall be implemented using Scikit-learn and Pandas via backend Python microservices.

External

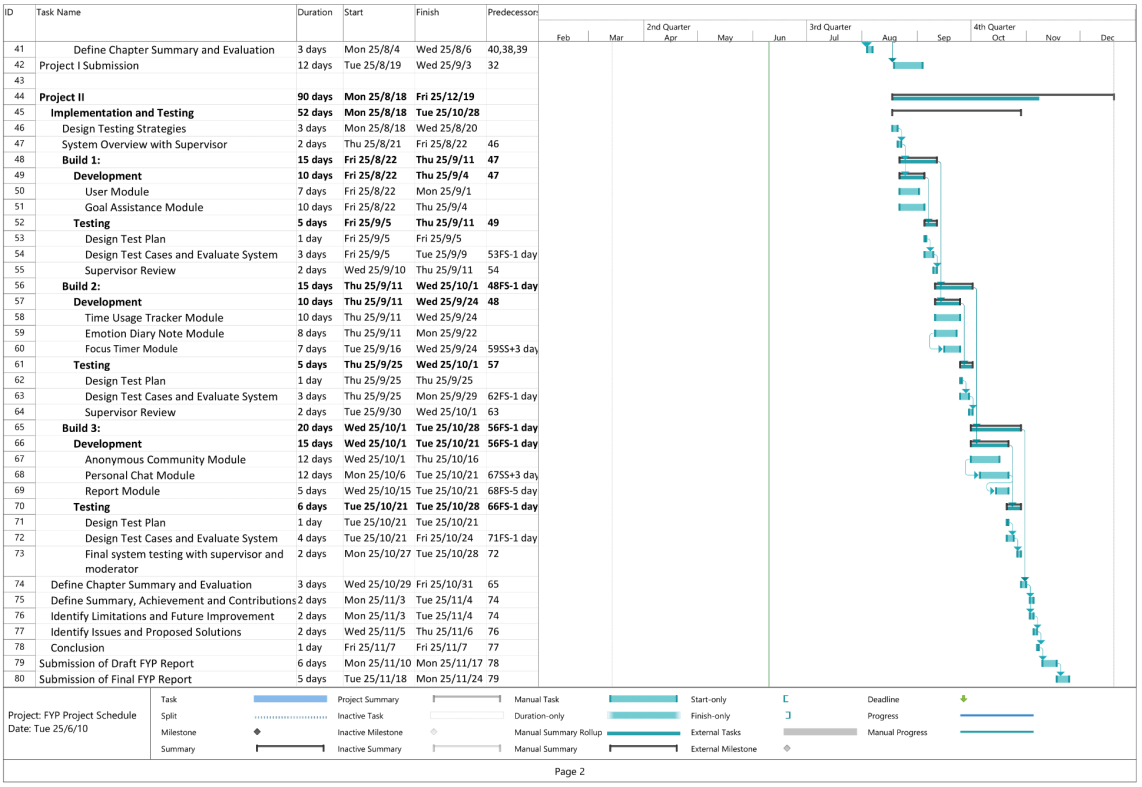
1. The app shall integrate Google or Apple Sign-In for secure user authentication.
2. If in-app purchases or donations are introduced, payment processing shall be handled via Google Play and Apple App Store's native payment APIs.
3. All user data handling must comply with relevant privacy laws (e.g., GDPR, CCPA) and Google/Firebase security best practices.

1.4.2 Project Schedule



ID	Task Name	Duration	Start	Finish	Predecessors
1	**Project I: Project Planning**	**127 days?**	**Fri 25/3/14**	**Mon 25/9/8**	
2	**Project Planning**	**87 days?**	**Fri 25/3/14**	**Mon 25/7/14**	
3	**Proposal**	**26 days?**	**Fri 25/3/14**	**Fri 25/4/18**	
4	Introductions	2 days	Fri 25/3/14	Sun 25/3/16	
5	Identify Problems and Proposed Solutions	10 days	Mon 25/3/17	Fri 25/3/28	4
6	Identify Target Users' Background	10 days	Mon 25/3/17	Fri 25/3/28	4
7	Application System Comparison	10 days	Mon 25/3/31	Fri 25/4/11	5
8	Define Project Milestones	2 days	Mon 25/4/14	Tue 25/4/15	7
9	Proposal Submission	3 days	Wed 25/4/16	Fri 25/4/18	8,6
10	**Introduction**	**13 days?**	**Thu 25/6/12**	**Mon 25/6/30**	**3**
11	Define Objectives	1 day	Thu 25/6/12	Thu 25/6/12	
12	Identify Problems and Proposed Solutions	2 days	Thu 25/6/12	Fri 25/6/13	
13	Specify Advantages and Contributions	3 days	Mon 25/6/16	Wed 25/6/18	12
14	Design Project Plan	5 days	Thu 25/6/19	Wed 25/6/25	11,13
15	Design Project Schedule and Team Organizat	3 days	Mon 25/6/23	Wed 25/6/25	14SS+2 day
16	Define Chapter Summary and Evaluation	2 days	Thu 25/6/26	Fri 25/6/27	15
17	**Project Background**	**12 days**	**Sat 25/6/28**	**Mon 25/7/14**	**10**
18	Identify Target User and Community	2 days	Sat 25/6/28	Mon 25/6/30	
19	Literature Review	3 days	Mon 25/6/30	Wed 25/7/2	18FS-1 day
20	Feasibility Study	7 days	Wed 25/7/2	Thu 25/7/10	19FS-1 day
21	Define Chapter Summary and Evaluation	2 days	Fri 25/7/11	Sun 25/7/13	20
22	**Methodology and Requirement Analysis**	**20 days?**	**Tue 25/7/1**	**Mon 25/7/28**	**17**
23	Identify Methodology	3 days	Tue 25/7/1	Thu 25/7/3	
24	Define Requirements Gathering Techniques	3 days	Thu 25/7/3	Mon 25/7/7	23FS-1 day
25	**Requirement Analysis**	**10 days?**	**Tue 25/7/8**	**Sun 25/7/20**	**24**
26	Design Use Case Diagrams	3 days	Tue 25/7/8	Thu 25/7/10	
27	Design Use Case Description	2 days	Fri 25/7/11	Mon 25/7/14	26
28	Define Functional Requirements	4 days	Tue 25/7/15	Fri 25/7/18	27
29	Define Non-Functional Requirements	4 days	Tue 25/7/15	Fri 25/7/18	27
30	Development Environment Analysis	6 days	Fri 25/7/18	Fri 25/7/25	25FS-1 day
31	Define Chapter Summary and Evaluation	2 days	Fri 25/7/25	Mon 25/7/28	30FS-1 day
32	**Design**	**28 days?**	**Thu 25/7/10**	**Mon 25/8/18**	**22**
33	**System Design**	**20 days**	**Thu 25/7/10**	**Wed 25/8/6**	
34	Sequence Diagram Design	3 days	Thu 25/7/10	Sun 25/7/13	
35	State Chart Diagram Design	3 days	Mon 25/7/14	Wed 25/7/16	34
36	User Interface Design	5 days	Mon 25/7/14	Fri 25/7/18	34
37	Data Design	6 days	Mon 25/7/21	Sat 25/7/26	36
38	Reports Design	4 days	Mon 25/7/28	Thu 25/7/31	37
39	Process Design	5 days	Mon 25/7/28	Fri 25/8/1	37
40	Software Architecture Design	5 days	Mon 25/7/14	Fri 25/7/18	34

Figure 2.1.1: Gantt Chart of BetterU Project Schedule - Page 1



ID	Task Name	Duration	Start	Finish	Predecessors
41	Define Chapter Summary and Evaluation	3 days	Mon 25/8/4	Wed 25/8/6	40,38,39
42	Project I Submission	12 days	Tue 25/8/19	Wed 25/9/3	32
43					
44	**Project II**	**90 days**	**Mon 25/8/18**	**Fri 25/12/19**	
45	**Implementation and Testing**	**52 days**	**Mon 25/8/18**	**Tue 25/10/28**	
46	Design Testing Strategies	3 days	Mon 25/8/18	Wed 25/8/20	
47	System Overview with Supervisor	2 days	Thu 25/8/21	Fri 25/8/22	46
48	**Build 1:**	**15 days**	**Fri 25/8/22**	**Thu 25/9/11**	**47**
49	**Development**	**10 days**	**Fri 25/8/22**	**Thu 25/9/4**	**47**
50	User Module	7 days	Fri 25/8/22	Mon 25/9/1	
51	Goal Assistance Module	10 days	Fri 25/8/22	Thu 25/9/4	
52	**Testing**	**5 days**	**Fri 25/9/5**	**Thu 25/9/11**	**49**
53	Design Test Plan	1 day	Fri 25/9/5	Fri 25/9/5	
54	Design Test Cases and Evaluate System	3 days	Fri 25/9/5	Tue 25/9/9	53FS-1 day
55	Supervisor Review	2 days	Wed 25/9/10	Thu 25/9/11	54
56	**Build 2:**	**15 days**	**Thu 25/9/11**	**Wed 25/10/1**	**48FS-1 day**
57	**Development**	**10 days**	**Thu 25/9/11**	**Wed 25/9/24**	**48**
58	Time Usage Tracker Module	10 days	Thu 25/9/11	Wed 25/9/24	
59	Emotion Diary Note Module	8 days	Thu 25/9/11	Mon 25/9/22	
60	Focus Timer Module	7 days	Tue 25/9/16	Wed 25/9/24	59SS+3 day
61	**Testing**	**5 days**	**Thu 25/9/25**	**Wed 25/10/1**	**57**
62	Design Test Plan	1 day	Thu 25/9/25	Thu 25/9/25	
63	Design Test Cases and Evaluate System	3 days	Thu 25/9/25	Mon 25/9/29	62FS-1 day
64	Supervisor Review	2 days	Tue 25/9/30	Wed 25/10/1	63
65	**Build 3:**	**20 days**	**Wed 25/10/1**	**Tue 25/10/28**	**56FS-1 day**
66	**Development**	**15 days**	**Wed 25/10/1**	**Tue 25/10/21**	**56FS-1 day**
67	Anonymous Community Module	12 days	Wed 25/10/1	Thu 25/10/16	
68	Personal Chat Module	12 days	Mon 25/10/6	Tue 25/10/21	67SS+3 day
69	Report Module	5 days	Wed 25/10/15	Tue 25/10/21	68FS-5 day
70	**Testing**	**6 days**	**Tue 25/10/21**	**Tue 25/10/28**	**66FS-1 day**
71	Design Test Plan	1 day	Tue 25/10/21	Tue 25/10/21	
72	Design Test Cases and Evaluate System	4 days	Tue 25/10/21	Fri 25/10/24	71FS-1 day
73	Final system testing with supervisor and moderator	2 days	Mon 25/10/27	Tue 25/10/28	72
74	Define Chapter Summary and Evaluation	3 days	Wed 25/10/29	Fri 25/10/31	65
75	Define Summary, Achievement and Contributions	2 days	Mon 25/11/3	Tue 25/11/4	74
76	Identify Limitations and Future Improvement	2 days	Mon 25/11/3	Tue 25/11/4	74
77	Identify Issues and Proposed Solutions	2 days	Wed 25/11/5	Thu 25/11/6	76
78	Conclusion	1 day	Fri 25/11/7	Fri 25/11/7	77
79	Submission of Draft FYP Report	6 days	Mon 25/11/10	Mon 25/11/17	78
80	Submission of Final FYP Report	5 days	Tue 25/11/18	Mon 25/11/24	79

Figure 2.1.2: Gantt Chart of BetterU Project Schedule - Page 2

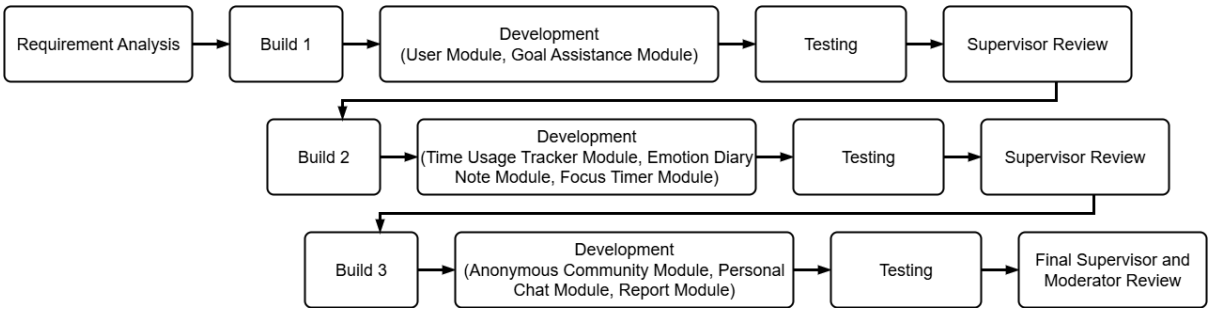


Figure 2.2: Incremental Model Diagram for BetterU

1.5 Project Team and Organization

Module in charge		Member Name
Emotion Diary Note	• Diary Entry Management • Sentiment Analysis & Mood Tagging • Privacy Control	Chia Ming Yi
Focus Timer	• Focus Session Control • Gamification • Distraction Prevention	
User	• User Registration & Authentication • User Profile & Preferences • Administration • Session Control	
Report	• Insight Generation • Data Visualization • Notifications & Reflection	
Time Usage Tracker	• App Activity Tracker • NFC Task Tracker • Daily and Weekly Summary • Custom Time Goals and Alerts	Lim Jun Wei
Anonymous Community	• Topic-Based Community • Q&A Forum • Group Chat • Post Reaction & Flagging	
Personal Chat	• Private Messaging • Identity Control	
Report	• Productivity Report • Social Performance Report • Personalized Insight and Suggestions	
Goal Assistance (Shared)	• Task Management • Smart Input & Suggestions • Smart Advisor	Chia Ming Yi Lim Jun Wei

Table 1.1: Organization Table for BetterU

1.6 Chapter Summary and Evaluation

1.1 Objectives

The BetterU system aims to enhance productivity and emotional well-being through five core modules:

- Goals Assistance Module (Shared) :
  Intelligent task management via smart scheduling, task extraction from speech, adaptive rescheduling, and real-time progress tracking.
- Emotion Diary Note Module:
  Emotional tracking through text, image, or speech entries, mood tagging, sentiment analysis, and a personal reflection library.
- Focus Timer Module:
  Gamified focus sessions with app usage restrictions, in-app rewards, and progress tracking to encourage deep work.
- User Module:
  Secure and personalized user management with account control, preference storage, and customizable settings.
- Report Module:
  Visual summaries and insights on user behavior, mood trends, and focus patterns to support self-assessment and decision-making.

1.2 Problems

Mental Health and Emotional Well-being:

- Rising mental health issues in Malaysia, stigma, limited professional access, and socio-economic pressures.
- BetterU Solutions: Smart task management, emotional tracking, gamified focus sessions, and actionable insights for early stress detection and self-care.

Work-Life Imbalance and Burnout:

- Overwork culture, high stress levels, blurred work-life boundaries, and increasing burnout cases.
- BetterU Solutions: Efficient task organization, emotional journaling, structured work intervals, and personalized user experiences to promote balance.

1.3 Advantages and Contributions

- Improves productivity and focus through gamification.
- Enhances emotional awareness via diary and sentiment analysis.
- Supports smarter time management with intelligent scheduling.
- Offers user-friendly customization for comfort and control.
- Provides insightful feedback to empower self-improvement.
- Bridges productivity tools and emotional support in a single app.
- Encourages proactive mental health management.
- Promotes a balanced and sustainable lifestyle.

1.4 Project Plan

In the project plan, BetterU solution has adopted the **Incremental Model** as the process model. After the requirement gathering and analysis, BetterU will mainly focus on development of User and Goal Assistance modules as fundamental modules during the first build.

During the second build, BetterU will integrate with Time Usage Tracker and Focus Timer as an extended integration module for improving the relevance and accuracy of planning users' task schedules, and enhancing productivity of users based on their behavior.

In the third build, BetterU will focus on Anonymous Community, Personal Chat and Report modules for providing a secure and comfortable social engagement to users, and generating various reports with suggestions for users.

In every build, BetterU will undergo a series of testing and supervisor review for collecting feedback and continuous improvement via feedback integration. Thus, BetterU can always maintain a high quality of services and functionality.

1.5 Project Team and Organization

The Table 1.1 outlines the project team and organization for BetterU, specifying the modules each team member is responsible for. Here's a summary:

Chia Ming Yi will be responsible for:

- Emotion Diary Note
- Focus Timer
- User
- Report

Lim Jun Wei will be responsible for:

- Time Usage Tracker
- Anonymous Community
- Personal Chat
- Report

Both of us will be responsible for:

- Goal Assistance

Chapter 2

Literature Review

2 Literature Review

Chapter 2 provides a review of the background, technologies and tools applied in the development of BetterU mobile application. It identifies the target users of this mobile application such as students and working professionals. Meanwhile, it also explores the challenges faced by them when managing productivity and emotional well-being. On the other hand, this chapter also highlights the core technologies used in BetterU, including programming languages, development frameworks, backend systems, AI tools and UI/UX design platforms. Eventually, it examines the feasibility of the project from technical, economic, operational and scheduling perspectives to ensure the practicality and effectiveness of the proposed solution.

2.1 Project Background

2.1.1 Target Markets

BetterU is designed to support **two primary target markets in Malaysia: students and working professionals**. Both groups face unique challenges that impact their productivity, emotional well-being, and ability to manage daily responsibilities.

1. Students (Secondary and University Level in Malaysia)

Students, particularly those at the secondary and university levels, often struggle with overwhelming academic demands, co-curricular activities, and personal obligations. These pressures can lead to high stress, disrupted routines, and emotional exhaustion. The BetterU assists students by offering structured modules that help break down complex study plans into manageable tasks, track emotional trends through a flexible diary feature, and encourage focused study sessions via a gamified timer. Students can personalize their app settings to fit their academic schedules and receive visual progress reports that enhance their self-awareness and promote healthier study habits.

Demographic Analysis:

- Age Range: 13 – 25 years old
    - This age group includes adolescents and young adults who are frequent smartphone users and are highly engaged with mobile applications. They are part of a generation that is comfortable using digital solutions for learning, task management, and mental health support.
- Education Level: Secondary school, pre-university, college, and university students

    - Students within these education levels typically experience escalating workloads, complex assignments, and exam pressures. The demand for efficient planning and task management increases as they progress through their academic journey.
- Occupation: Full-time students
    - The application is tailored for students whose primary focus is on their studies. These students often juggle multiple responsibilities such as coursework, co-curricular activities, part-time jobs, and personal commitments, making them ideal beneficiaries of a productivity-focused solution.
- Location: Primarily urban and semi-urban areas
    - Students in urban and semi-urban regions are more likely to have consistent access to stable and fast internet connections, which are essential for the optimal use of BetterU's mobile application features. These areas also typically show higher rates of mobile app adoption and usage.
- Device Access: Regular access to smartphones
    - The target students commonly own smartphones that run on widely supported operating systems such as Android and iOS. This consistent device access enables them to use BetterU anytime and anywhere, ensuring the app's practicality in their daily academic and personal routines.

Psychographic Analysis:

- Lifestyle: Students within this target market typically manage demanding schedules that include schoolwork, co-curricular activities, frequent examinations, and sometimes part-time employment. These overlapping commitments often leave students with limited personal time, increasing their stress levels and reducing opportunities for socialization. Especially in top-ranked schools and competitive universities, students are pressured to maintain high academic performance while balancing multiple responsibilities.
- Goals: Students aim to achieve their academic targets while efficiently managing their tasks and deadlines. In addition to academic performance, they seek to maintain emotional stability, build self-confidence, and preserve social interactions, which are essential for their overall mental well-being.
    - Academic success
    - Improved time management
    - Emotional balance
    - Stronger peer and social connections

- Values: The target students value continuous growth, effective time use, and becoming more organized. They are also becoming increasingly aware of the importance of maintaining their mental health, seeking solutions that can support both their academic and emotional needs.
    - Self-improvement
    - Productivity and efficiency
    - Mental health awareness
- Challenges: Many students face persistent procrastination and have difficulty structuring their tasks clearly, which often results in incomplete work and increased anxiety. Their high sensitivity to surrounding distractions makes it even more challenging to stay focused, especially when under pressure.
    - Procrastination tendencies
    - Difficulty in task organization
    - High stress from multitasking
    - Struggles with maintaining focus due to environmental distractions

Behavioral Analysis:

Modern students heavily rely on mobile devices for learning, productivity, communication, and relaxation. Their daily routines are closely intertwined with mobile applications that support both academic tasks and social engagement. BetterU is designed to meet their behavioral patterns and preferences by offering a simplified, student-centered task management experience.

- Technology Usage: Students exhibit high smartphone dependency throughout the day for various purposes:
    - Accessing online learning resources (text, images, videos)
    - Using task management applications for scheduling and organizing assignments
    - Engaging with social media to communicate with family, classmates, and online communities
    - Consuming entertainment content for relaxation
- Technology Habits: Students typically access mobile applications at specific times of the day:
    - Before school: Checking social media updates and reviewing task reminders
    - After school: Using entertainment apps, chatting with friends, and reviewing pending tasks or assignments
    - After completing tasks: Returning to social media platforms or entertainment apps to unwind and stay socially connected

- Application Preferences:
    - User-friendly: Simple navigation and intuitive interface
    - Flexible: Adaptable to changing schedules and needs
    - Personalized: Allow customization to fit individual preferences
    - Simple to use: Straightforward features without overwhelming complexity
- Pain Points:
    - Many existing task management applications are overly complex and include features that are not relevant to students' daily needs.
    - The unnecessary features in these apps make them difficult to adopt and discourage consistent usage.
    - Students may feel overwhelmed or disinterested in apps that require extra effort to learn or configure.

2. Working Professionals (Urban Office Workers in Malaysia)

Working professionals, especially urban office workers in Malaysia, frequently encounter work-life imbalance due to heavy workloads, digital overexposure, and the blurred boundaries between work and rest in hybrid work environments. This often results in chronic stress, emotional detachment, and decreased productivity. The system addresses these issues by helping professionals organize tasks, balance personal and work commitments, monitor emotional well-being, and maintain focus through structured timers. The solution is highly customizable, offering settings that align with individual routines and lifestyles, while providing regular reports to help users detect early signs of burnout and optimize their work-life balance.

Demographic Analysis:

- Age Range: 22 – 45 years old
    - This age group includes young to mid-career professionals who are actively engaged in building their careers. They are experienced with digital tools and are frequent users of productivity and communication applications.
- Occupation: Full-time office workers, administrative staff, junior to mid-level professionals
    - These individuals primarily work in office environments or remote settings, dealing with task-heavy roles that require continuous coordination, communication, and deadline management.
- Education Level: College graduates or higher

    - Most urban office workers hold at least a college degree, equipping them with the skills and knowledge to manage complex work responsibilities using digital tools.
- Location: Primarily urban and semi-urban areas with good Internet infrastructure
    - The target professionals are located in areas where fast and stable internet connectivity is accessible, which is crucial for the seamless operation of BetterU's mobile application.
- Device Access: Regular access to smartphones, laptops, and desktop computers
    - These professionals frequently use multiple devices for work and task management, including smartphones, laptops, and desktops, ensuring consistent connectivity and access to task management tools.

Psychographic Analysis:

- Lifestyle: Office workers in this segment manage packed schedules with long hours in front of screens. Their daily routines often involve extended working hours, overtime, and frequent digital communication. As a result, they face limited opportunities for personal and social interactions outside of work.
- Goals: They desire effective ways to balance professional success with personal well-being while minimizing mental fatigue. They aim to achieve:
    - Work-life balance
    - High work efficiency
    - Reduced stress
    - Continuous career growth
- Values: They seek solutions that can help them optimize task completion, reduce unnecessary workload stress, and protect their mental health.
    - Time management
    - Productivity
    - Mental well-being
- Challenges: Without efficient planning tools, they risk confusion, wasted time, and decreased productivity, often leading to frustration and burnout. Many office workers struggle with:
    - Limited time and effort to create detailed work plans
    - Losing focus and feeling overwhelmed by task overload
    - Poor task organization
    - Difficulty balancing work responsibilities with personal life

Behavioral Analysis:

Office workers frequently rely on productivity applications such as Google Calendar, Notion, and Slack to manage their tasks, track meetings, and take notes. They also use communication platforms like Zoom and Microsoft Teams for virtual meetings and real-time collaboration.

- Technology Usage:
  - Heavy use of productivity apps and task management tools
  - Dependence on communication apps for online meetings and workplace coordination
  - Multi-device access throughout the workday (smartphones, laptops, desktops)
- Habits:
  - Checking notifications and task reminders in the morning before work
  - Accessing calendars and scheduling apps during office hours to track daily tasks
  - Using communication platforms to receive assignments and update progress throughout the day
  - Reviewing remaining tasks after work to plan for the next day
- Application Preferences:
  - Minimalist design for easy navigation and quick access
  - Responsive interface with real-time updates and fast synchronization
  - Highly customizable task management features to fit individual workflows
- Pain Points:
  - Current task management apps often lack real-time tracking and smooth synchronization across devices
  - Many apps include generic, non-personalized features that do not support the unique demands of individual users
  - Complex interfaces and manual configuration requirements increase user effort instead of simplifying task management

2.1.2 Technology and Development Tools Used

Development Platform and Framework

BetterU is developed using the Flutter framework with the Dart programming language. Flutter was chosen for its ability to streamline development through a single codebase, significantly reducing complexity and development time. Its extensive library of customizable widgets and strong community support also enable rapid prototyping and flexible UI design, making it well-suited for implementing the features and user experience envisioned in the BetterU solution.

Database

Google Firebase is used by the BetterU mobile application for real-time data storage and synchronization. It offers seamless integration with Flutter and provides a range of backend services such as authentication, cloud storage, and analytics. As a free and developer-friendly platform, Firebase can be easily implemented into the BetterU app, effectively reducing both development time and costs.

AI or ML Features

Speech Recognition (Vosk, Whisper)

BetterU integrates speech recognition capabilities using Vosk and Whisper to provide real-time and high-accuracy transcription from user voice input. These Automatic Speech Recognition (ASR) models support multi-language input and automatic language detection, which enhances accessibility for diverse users. This feature allows users to input task items through speech, improving convenience and usability, especially in mobile contexts where typing may not be practical (Radford et al., 2023; Vosk API, 2020).

Task Parsing and Intent Detection (spaCy, Hugging Face)

To interpret the meaning and structure of user input, BetterU uses Natural Language Processing (NLP) tools such as spaCy and Hugging Face Transformers. These tools tokenize, parse, and extract relevant information like dates and times from both voice and text entries. This process supports automatic scheduling by identifying the most likely task deadlines and refining task descriptions to be concise, relevant, and easily readable (Explosion AI, 2023; Wolf et al., 2020).

Task Classification and Prioritization (Scikit-learn)

Scikit-learn is utilized to classify tasks into different categories (e.g., work, health, personal) and to assign priority levels based on urgency and importance. The model is trained on labeled datasets to detect common patterns, enabling it to automate task sorting and reduce the user's cognitive load. This intelligent prioritization ensures that users focus on what matters most while reducing manual overhead (Pedregosa et al., 2011).

Personalized Recommendations (LightFM, Embeddings)

The Smart Advisor module in BetterU leverages a recommendation system built with LightFM and matrix factorization techniques. By analyzing user preferences, habits, and contextual behavior, the system provides tailored suggestions for goals, tasks, or productivity strategies. Embedding techniques allow the model to learn fine-grained patterns across users and tasks, improving recommendation accuracy over time (Kula, 2015).

Time Optimization Engine (OR-Tools, RLib)

BetterU integrates time optimization algorithms using Google OR-Tools for constraint solving and reinforcement learning through RLib to adaptively plan user schedules. These techniques consider user availability, task durations, and predefined constraints to generate efficient and feasible schedules. Reinforcement learning enables the system to improve scheduling outcomes based on continuous feedback (Google OR-Tools, 2023; Liang et al., 2018).

Adaptive Rescheduling (Random Forest, LSTM)

To dynamically respond to changes in user behavior, BetterU applies machine learning models such as Random Forest and Long Short-Term Memory (LSTM) networks. These models are trained to detect behavior patterns and predict potential delays or interruptions. Based on these predictions, the app can suggest task rescheduling or notify users of potential conflicts in advance, enhancing flexibility and reliability (Hochreiter & Schmidhuber, 1997; Breiman, 2001).

UI Design Tool

Figma is utilized for designing the graphical user interface (GUI) of the BetterU mobile application. Figma is a cloud-based design tool that allows multiple team members to work simultaneously on the same interface design. This real-time collaboration enhances team productivity by enabling parallel work on different modules or screens, reducing design cycle times and minimizing version conflicts (Nelson, 2020).

Figma is accessible across various devices, including smartphones, tablets, and laptops, which ensures flexibility and convenience for remote or hybrid development teams. As an interactive prototyping tool, Figma supports the creation of dynamic components that respond to user actions, such as clicks, hovers, or transitions. This functionality allows for more realistic and engaging prototypes during the development phase, making it easier to visualize user interactions and gather feedback before actual implementation. Moreover, its freemium pricing model helps control design costs, aligning with the project's goal of cost-effective development.

Through Figma, BetterU’s development and design teams can stay synchronized throughout the design process, ensuring a cohesive and user-centered interface experience.

2.2 Literature Review

2.2.1 Programming Languages and Frameworks

Dart

Dart is a client-optimized, object-oriented programming language developed by Google, designed to build fast, scalable applications across multiple platforms (Dart, n.d.). In BetterU, Dart is primarily used to implement the core app logic and to integrate tightly with Flutter's UI framework, enabling smooth and consistent UI adaptation across different platforms.

- **Powerful Performance**
    - Dart employs Ahead-of-Time (AOT) compilation, which compiles the code into native machine code before the app runs. This significantly improves app startup time and runtime performance, resulting in a smooth user experience (Code Carbon, 2023). Additionally, during development, Dart supports Just-In-Time (JIT) compilation that allows developers to see live updates immediately as code changes are made, without restarting the app. This capability greatly enhances development efficiency by shortening the edit-test cycle (MoldStud, 2023).
- **Cross-Platform Development**
    - Dart is designed to work seamlessly with Flutter, allowing a single codebase to produce apps that run consistently on both Android and iOS devices (Java Code Geeks, 2022). This unification of development saves significant time and reduces maintenance efforts, since the same codebase can be reused for multiple platforms rather than developing separate native apps.
- **Object-Oriented and Familiar Syntax**
    - Dart's syntax resembles popular languages such as Java and C++, making it easier for developers with prior experience in these languages to transition smoothly. Its strong typing system helps catch errors early during development, improving code reliability and robustness (Java Code Geeks, 2022; MoldStud, 2023).

Python

Python is a versatile, easy-to-learn programming language widely adopted in mobile app backend development and AI-related functionalities (MobileAppDaily, n.d.; Monterail, n.d.). For BetterU, Python plays a critical role in implementing powerful backend logic and AI features that support advanced user needs.

- **Powerful AI and NLP Libraries**

- Python offers a rich ecosystem of artificial intelligence (AI) and natural language processing (NLP) libraries such as spaCy, Scikit-learn, and SetFit. These libraries enable BetterU to implement features like speech-to-text conversion, task extraction, and intelligent task categorization with relatively minimal effort (spaCy, n.d.; Scikit-learn, n.d.; Chamrah, 2023).

- **Backend Development**
  - Using FastAPI, a modern Python web framework, BetterU's backend APIs can efficiently connect the mobile frontend with AI services. FastAPI's straightforward syntax and automatic documentation generation promote rapid prototyping and high accessibility for AI models, streamlining the development of complex AI-driven functionalities (FastAPI, n.d.).
- **Clean Separation of Concerns**
  - Python handles backend processing and AI logic, while Dart combined with Flutter manages the frontend user interface and lighter logic. This clear division of responsibilities makes the system easier to develop, debug, and maintain by isolating concerns into separate layers (FastAPI, n.d.).

Flutter

Flutter is an open-source UI toolkit created by Google for building high-performance, natively compiled applications from a single codebase across multiple platforms (Flutter, n.d.). In BetterU, Flutter serves as the foundation for delivering a visually appealing and consistent user experience.

- **Cross-Platform Development Efficiency**
  - Flutter enables BetterU to be deployed not only on Android and iOS but also on web and desktop platforms with consistent look and behavior. This broad coverage significantly reduces the effort needed for development and maintenance compared to building separate apps for each platform (Monterail, n.d.).
- **Abundant and Customizable UI Components**
  - Flutter offers a rich set of customizable widgets that allow developers to create intuitive, interactive user interfaces with smooth animations and transitions. This flexibility supports the development of a seamless and engaging user experience in BetterU (Anuyat, n.d.).
- **Hot Reload for Fast Development**
  - One of Flutter's standout features is hot reload, which lets developers instantly view code changes reflected in the running app without restarting it. This capability accelerates the design, development, and testing phases,

especially when fine-tuning UI elements and behaviors (Unfolding the Advantages, n.d.).

2.2.2 Development Tools and IDE

Android Studio

Android Studio is Google's official Integrated Development Environment (IDE) tailored for mobile application development, providing comprehensive support for Flutter through dedicated plugins (Cellular Insider, n.d.). As the main development environment for the BetterU frontend, Android Studio streamlines the entire UI creation process.

- **Flutter Plugin Integration**
  - The Flutter plugin within Android Studio automates many repetitive tasks such as project setup and configuration, allowing developers to focus on coding and design. It supports Flutter's hot reload feature, which enables immediate preview of UI changes without restarting the application, significantly accelerating the development cycle and enabling rapid iteration (Cellular Insider, n.d.).
- **Built-in Emulator**
  - Android Studio includes Android Virtual Devices (AVDs), which simulate a variety of Android devices with different screen sizes, resolutions, and API levels. This virtual testing environment allows developers to debug and test the BetterU app across multiple device profiles without needing physical hardware, improving testing flexibility and reducing costs (Android Developers, n.d.).
- **Intelligent Code Editor**
  - The IDE offers an intelligent code editor with features like code completion, syntax highlighting, and real-time error detection. Advanced debugging tools such as breakpoints, step-through execution, and variable inspection help developers identify and fix issues efficiently, improving code quality and reducing development time (Cellular Insider, n.d.).

Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is the chosen lightweight yet powerful code editor for BetterU's Python backend development (Microsoft, n.d.). Its versatility and extensibility make it well suited for developing backend APIs and AI logic.

- **Integrated Terminal**

- VS Code's built-in terminal allows developers to run shell commands, manage virtual environments, and install packages without leaving the editor interface. This seamless integration promotes a smooth workflow by eliminating the need to switch between multiple tools (Microsoft, n.d.).

- **API Development Support**
  - With Python extensions, VS Code offers real-time syntax and error checking, intelligent code completion, and debugging capabilities. These features facilitate the development of FastAPI backend services, which interface efficiently with BetterU's Flutter frontend, enabling quick iteration and integration of new features (FastAPI, n.d.).

GitHub Desktop

GitHub Desktop is an open-source graphical user interface for Git version control that simplifies repository management and collaborative development (GeeksforGeeks, 2022). It plays a critical role in coordinating work among BetterU's development team.

- **Branch and Commit Management**
  - The tool streamlines essential Git operations such as staging changes, writing descriptive commit messages, creating and switching branches, and merging code. This structure helps keep feature development organized and traceable, facilitating parallel workflows and agile development (GeeksforGeeks, 2022).
- **Conflict Resolution**
  - GitHub Desktop provides intuitive interfaces for resolving merge conflicts that arise when multiple developers modify the same files. This reduces integration errors and helps maintain a stable codebase in a collaborative environment (GeeksforGeeks, 2022).
- **Improved Collaboration**
  - By enabling real-time syncing and pushing of code changes to remote repositories, GitHub Desktop minimizes the risk of conflicting edits and ensures all team members work with the latest codebase. This fosters efficient teamwork and continuous integration practices (GeeksforGeeks, 2022).

2.2.3 Backend and API Integration

FastAPI

FastAPI is a modern, high-performance Python web framework for building APIs using standard Python type hints (FastAPI, n.d.). It bridges BetterU's Flutter frontend and Python backend, enabling smooth communication for AI features like task extraction and speech recognition. FastAPI handles HTTP requests and JSON responses efficiently (DEV Community, 2023).

2.2.4 AI Tools and Algorithms

Vosk

Vosk is an offline speech recognition toolkit that supports multiple languages and enables real-time voice-to-text transcription without requiring an internet connection (Vosk API, n.d.). In BetterU, Vosk provides robust offline transcription capabilities, ensuring user privacy and functionality even in environments with limited connectivity.

Whisper

Whisper, developed by OpenAI, is employed alongside Vosk to enhance transcription accuracy, particularly for English and Mandarin voice inputs (OpenAI, n.d.). Whisper's complementary use enables BetterU to deliver precise speech recognition across multiple languages and accents, improving user experience.

spaCy

spaCy is an advanced Natural Language Processing (NLP) library designed for efficient linguistic analysis, including tokenization, part-of-speech tagging, and syntactic parsing (spaCy, n.d.). BetterU leverages spaCy to extract key task information such as descriptions, deadlines, and relevant entities from transcribed voice inputs, enabling intelligent task management.

dateparser

dateparser is a Python library specialized in interpreting natural language date and time expressions (dateparser, n.d.). Integrated with spaCy, it converts ambiguous temporal phrases such as "next Friday" or "at 3 pm" into standardized datetime objects, facilitating accurate task scheduling within BetterU.

Scikit-learn

Scikit-learn is a versatile Python machine learning library used for feature extraction, model building, and training (Scikit-learn, n.d.). BetterU employs Scikit-learn to develop predictive

models that categorize user tasks by type and assign priority levels, enabling automated task organization and prioritization.

SetFit

SetFit fine-tunes sentence transformer models to perform classification tasks efficiently on small labeled datasets (Chamrah, 2023). Within BetterU, SetFit enhances the Goal Assistance module by improving task categorization accuracy while minimizing the amount of required training data.

2.2.5 Database and Data Storage

Google Firebase

Google Firebase is a backend-as-a-service (BaaS) platform offering scalable, secure, real-time database and storage solutions for BetterU (Google, n.d.).

- **Realtime Database and Firestore**
  - Firebase's NoSQL databases store data as JSON trees, providing real-time synchronization across clients and enabling instant updates without manual refresh (Firebase Docs, n.d.; Estuary Dev, 2023). This supports BetterU's dynamic task data and user preferences.
- **Authentication and Security**
  - Firebase supports multiple authentication methods (email, Google, Facebook) and enforces security rules, ensuring data privacy and authorized access (Airbyte, n.d.).
- **Cloud Storage**
  - Firebase Cloud Storage handles large media files securely and efficiently, ideal for posts and media in BetterU's community and chat features (Firebase Docs, n.d.).

2.2.6 UI and UX Design

Figma

Figma is a web-based design tool used for UI/UX design and prototyping in the development of BetterU (Jimmy Viquez, n.d.).

- **Seamless Collaboration**
  - Figma supports real-time multi-designer collaboration, allowing team members to work simultaneously on the same design file. Features such as visible collaborator cursors and comprehensive version history facilitate safe,

efficient teamwork and minimize conflicts during the design process (Design Project, 2023).

- **Consistent and Scalable Design System**
    - By utilizing reusable components and variants, Figma enables BetterU to maintain a consistent visual language across all user interface elements. This approach allows global updates to be applied quickly and efficiently, ensuring UI scalability and reducing duplicated efforts (Design Project, 2023).
- **Rapid Prototyping**
    - Figma's interactive and clickable prototypes can be created rapidly without the need for coding. This capability supports early-stage usability testing, enabling the team to gather user feedback and make design improvements before development, ultimately minimizing costly rework (Design Project, 2023).

2.3 Feasibility Study

2.3.1 Technical Feasibility

Hardware Feasibility

The BetterU mobile application is compatible with standard mobile devices running Android and iOS operating systems since it is built using Flutter, which supports cross-platform development (Flutter, n.d.). For design and coding, a Windows laptop or desktop is sufficient for developing the Flutter frontend and Python backend. Additional GPU resources can further accelerate the training and testing of AI models efficiently (Monterail, n.d.). During the testing phase, Android or iOS physical devices will be used to validate the features within the installed BetterU application.

Software Feasibility

Several essential software and platforms are required for the BetterU development process. Android Studio is used for building the Flutter frontend using Dart and provides virtual devices to simulate physical devices for debugging and testing (Cellular Insider, n.d.). Visual Studio Code is primarily used for backend development, including Python-based API services and AI model training using FastAPI (FastAPI, n.d.). VS Code also supports integration with numerous open-source AI and NLP libraries such as spaCy, Vosk, Whisper, and SetFit, which are crucial for implementing BetterU's AI features (spaCy, n.d.; Vosk API, n.d.; OpenAI, n.d.; Chamrah, 2023). Also, Figma is employed for user interface (UI) design and prototyping. Figma's design tools allow the BetterU team to simulate the user flow and design structure efficiently before proceeding to code implementation (Design Project, 2023).

2.3.2 Economical Feasibility

The BetterU mobile application project is economically feasible as it primarily relies on free and open-source tools, which significantly minimize the financial investment required for development.

- Cost of Development Tools
    - All core development tools used in BetterU, including Flutter, Dart, Python, and FastAPI, are open-source and freely available (Flutter, n.d.; FastAPI, n.d.; Monterail, n.d.). This eliminates the need for purchasing costly proprietary software licenses, resulting in a highly cost-efficient development process.
- Cloud Services

- BetterU utilizes Google Firebase's free-tier cloud services, which provide essential features such as real-time database management, user authentication, and cloud storage (Firebase, n.d.). These free-tier offerings are sufficient to support the application's data storage, user management, and synchronization needs during initial development and deployment phases, further reducing operational expenses.

- Version Control and Collaboration
  - GitHub Desktop and GitHub repositories are freely available tools that facilitate seamless code collaboration, version control, and real-time synchronization among BetterU developers (GitHub Desktop, n.d.). These capabilities enhance team productivity and ensure effective project management without additional cost.
- AI Tools and Libraries
  - The AI-driven components of BetterU leverage open-source libraries such as Vosk for offline speech recognition, Whisper for voice transcription, and spaCy for natural language processing (Vosk API, n.d.; OpenAI, n.d.; spaCy, n.d.). Utilizing these freely accessible tools allows for sophisticated AI functionalities without incurring expenses on commercial AI services or licenses.
- Overall Project Cost
  - By employing open-source software, free cloud services, and no-cost AI libraries, the BetterU project avoids significant direct financial expenditures. This strategic choice enhances the project's sustainability, cost-effectiveness, and overall financial viability for both current development and future improvements.

2.3.3 Operational Feasibility

The BetterU mobile application is operationally feasible, supporting seamless deployment on widely used platforms and devices while enabling easy backend accessibility and web-based management.

Cross-Platform Compatibility

BetterU is developed using the Flutter framework, which facilitates cross-platform deployment (Flutter, n.d.). However, the current project scope focuses exclusively on Android devices, supporting API level 21 (Android 5.0 Lollipop) and above. By targeting this broad range of Android versions, BetterU ensures compatibility with the majority of modern

smartphones, providing smooth performance without requiring high-end hardware. The application will not be developed or deployed for iOS platforms.

Backend Accessibility

The backend services of BetterU are implemented with FastAPI, allowing efficient integration with the Flutter frontend through RESTful API calls over HTTP/HTTPS (FastAPI, n.d.). This architecture supports real-time data exchange and enables AI-powered features such as speech recognition and task categorization, ensuring responsive user experiences.

Database Integration

Google Firebase manages BetterU's real-time database, user authentication, and cloud storage functionalities (Firebase, n.d.). Firebase's cloud-based infrastructure securely stores all user data and task information, enabling instant synchronization across multiple devices. This real-time database capability guarantees that users experience consistent, up-to-date information regardless of device.

Web Accessibility for Backend Services

While BetterU is primarily designed as a mobile Android application, its backend APIs can be deployed on free cloud hosting platforms supporting Python applications, such as Heroku or PythonAnywhere. This makes the backend services accessible via web browsers like Google Chrome, simplifying testing, maintenance, and providing a potential pathway for future web-based versions or administrative interfaces.

2.3.4 Schedule Feasibility

In order to ensure that the BetterU mobile application project is able to be completed within the time range provided, it has been separated into Project 1 and 2. Project 1 is mainly focusing on identifying and analyzing the objectives, problems and solutions for the BetterU mobile application. Meanwhile, it is also used to create the plan or strategies for analyzing requirements and designing the system using various diagrams such as sequence diagram, state chart diagram, software architecture diagram and other relevant design documents. Project 1 will start working on 14/3/2025 while the deadline is assigned on 8/9/2025.

- Proposal (14/3/2025 - 18/4/2025)
- Chapter 1: Introduction (12/6/2025 - 30/6/2025)
- Chapter 2: Project Background (28/6/2025 - 14/7/2025)
- Chapter 3: Methodology and Requirement Analysis (1/7/2025 - 28/7/2025)
- Chapter 4: System Design (10/7/2025 - 18/8/2025)

When it comes to Project 2, it will mainly focus on designing the testing strategies, developing various planned modules using code and supervisor review sessions. Since the BetterU mobile application project is applying the incremental model, Project 2 will be separated into 3 builds. Within each build, there will be development, testing and review phases focusing on different modules. Thus, it can efficiently and effectively ensure the smooth progress for all modules development and testing while avoiding overburdened situations within a phase. Project 2 will start working on 18/8/2025 while the deadline is assigned on 19/12/2025.

- Design Testing Strategies (18/8/2025 - 20/8/2025)
- Chapter 5: Implementation and Testing
- Build 1: User Module, Goal Assistance Module (22/8/2025 - 11/9/2025)
- Build 2: Time Usage Tracker Module, Emotion Diary Note Module, Focus Timer Module (11/9/2025 - 1/10/2025)
- Build 3: Anonymous Community Module, Personal Chat Module, Report Module (1/10/2025 - 28/10/2025)
- Chapter 6: Discussions and Conclusions (29/10/2025 - 7/11/2025)

2.4 Chapter Summary and Evaluation

The BetterU mobile application is designed to serve two primary user groups: students and office workers. These groups were chosen due to their shared challenges in maintaining productivity, managing complex schedules, coping with multitasking demands, and dealing with mental fatigue. By analyzing their demographic, psychographic, and behavioral patterns, it has been identified that BetterU addresses key issues they face by improving task management, mental clarity, and overall connection with personal goals and society.

Based on the findings from the literature review, several core technologies were selected for BetterU's system development. In terms of programming languages and frameworks, Dart and Flutter were chosen due to their strong cross-platform capabilities and developer-friendly features, including hot reload powered by Just-In-Time (JIT) compilation. These technologies allow rapid iteration and consistent deployment on Android devices. For backend and AI functionalities, Python was selected for its simplicity, versatility, and extensive library support.

The selected development tools and integrated development environments (IDEs) include Android Studio, Visual Studio Code, and GitHub Desktop, which collectively provide a robust environment for collaborative coding, version control, and cross-functional teamwork. To power BetterU's AI-driven features, a suite of open-source libraries was adopted. These include Vosk and Whisper for speech recognition, spaCy and dateparser for natural language processing and task extraction, and Scikit-learn and SetFit for intelligent task categorization and prediction. These tools enable advanced, user-centric capabilities while maintaining low implementation costs. For cloud services, Google Firebase was selected to handle real-time data storage, user authentication, and cloud synchronization. Its free-tier services are sufficient for early deployment and ensure real-time data updates across user devices. On the design side, Figma was chosen as the primary UI/UX tool due to its reusable components, collaborative editing features, and interactive prototyping support. This allows designers and developers to work in sync and iterate quickly based on feedback.

The feasibility study has concluded that BetterU is viable from technical, economic, operational, and schedule perspectives. The availability of development hardware, such as a Windows laptop with Visual Studio Code, and access to open-source tools and cloud services strongly support its technical and economic feasibility. Operational feasibility is also ensured through the integration of Flutter, FastAPI, and Firebase, enabling smooth backend–frontend communication and user data management. The project follows an incremental development model, which ensures structured progress, supports early testing and validation, and reduces implementation risks.

Chapter 3

Methodology and Requirements Analysis

3 Methodology and Requirements Analysis

This chapter outlines the foundational stages involved in the development of the BetterU mobile application, including project planning, user research, requirement analysis, and technical preparation. The aim of this chapter is to demonstrate how the project was strategically approached to ensure that the application is both practical and aligned with user needs. By adopting an appropriate development model, conducting relevant research, and selecting suitable tools and technologies, the project establishes a solid groundwork for building a responsive, intelligent, and user-centered wellness application.

3.1 Methodology

3.1.1 Selection of Development Methodology

The BetterU mobile application adopts the **Incremental Development Model** as the foundation for its project development workflow. This model was selected primarily because it supports staged development and continuous improvement, which aligns well with the modular architecture and complex feature set of the BetterU system. The application is divided into several key modules, including Goal Assistance (shared module), Emotion Diary Note, Focus Timer, User, and Report modules.

To avoid overwhelming the development process by building all modules at once, the incremental model allows the system to be **divided into multiple functional builds**, each targeting specific module functionalities. Unlike the traditional Waterfall model, which requires all requirements to be finalized before development begins, the incremental approach enables **progressive implementation**. This method allows each module to be individually developed, tested, refined, and validated before proceeding to the next stage, reducing risk and complexity across the project.

Each increment follows its own cycle of planning, development, testing, and review, focusing on selected modules such as Goal Assistance and Anonymous Community. Through continuous feedback and integration, the development team can **identify issues early and implement necessary improvements in a timely manner**. This ensures a smooth flow for enhancing existing features and integrating new ones into the system.

By breaking the application into incremental builds, the development team can prioritize modules based on importance. Core modules, which are foundational and interlinked with other features, are developed in the early stages. Supportive modules are then integrated in later stages, either to extend the capabilities of the core modules or to enhance user experience.

Planned Functional Builds:

- Functional Build 1: User, Goal Assistance modules
- Functional Build 2: Time Usage Tracker, Emotion Diary Note, Focus Timer modules
- Functional Build 3: Anonymous Community, Personal Chat, Report modules

This structured approach ensures steady progress, reduces overall project risk, and enhances the flexibility and adaptability of the BetterU mobile application throughout its development lifecycle.

3.1.2 Application of Incremental Model in BetterU Development

Requirement Analysis Phase

The requirement analysis phase serves as the foundation for the entire development process of the BetterU mobile application. At the beginning of this phase, the development team **identifies real-world problems related to task management, productivity, and mental well-being.** This is accomplished through online research and observation to gain accurate and relevant insights. Based on these findings, the team proposes effective solutions that directly address the identified challenges.

In parallel, the development team determines the primary target users of the system by analyzing their backgrounds, behaviors, and daily routines. This user profiling helps consolidate the development direction of the application. Furthermore, a **comparative evaluation is conducted between the proposed BetterU mobile application and existing platforms such as Notion, Microsoft To Do, Google Keep, and Confluence**. This analysis supports the identification of unique strengths, weaknesses, and opportunities for improvement in the proposed solution.

Following the research on user needs, real-world issues, and competing applications, the team moves on to define the system and user requirements. This step ensures that the final product is capable of delivering services that meet the users' expectations, while also adhering to quality standards and development constraints such as time and budget.

Prior to collecting requirements, the team selects appropriate fact-finding techniques to ensure efficient and accurate information gathering. Methods such as **online research and user observation are employed** to support this process. Once the requirements are collected, they are translated into technical documentation. This includes Use Case Diagrams, Use Case Descriptions, and comprehensive listings of both Functional and Non-Functional Requirements. These deliverables help convert user needs into clear technical guidelines, providing a structured reference for the development team and minimizing the risk of ambiguity during later stages of the project.

System Design Phase

The System Design Phase is essential for translating the requirements gathered during the analysis stage into a detailed and organized blueprint that guides the implementation of the BetterU mobile application. Prior to initiating module development, various system design components are developed to ensure a clear understanding of application workflows, module interactions, and technical architecture.

The key design elements include the Software Architecture Diagram, Process Design, Sequence Diagrams, State Chart Diagrams, User Interface Design, Data Design, and Reports Design. These elements collectively define the technical and functional structure of the system, providing a solid foundation for efficient development and integration.

Software Architecture Diagram

The software architecture defines the high-level structural framework of the BetterU mobile application, outlining how core components interact with one another. BetterU adopts a client-server architecture that separates the frontend interface from backend processing and data storage.

The frontend is developed using Flutter, enabling cross-platform mobile deployment. The backend is powered by FastAPI using Python, responsible for handling business logic, processing API requests, and executing AI-related tasks such as task extraction and classification. Google Firebase serves as the primary backend database, offering real-time synchronization and user authentication. The frontend communicates with the backend through API calls, allowing Python logic to access data stored in Firebase.

Process Design

Process design defines the internal workflows and logical operations for each core module in BetterU. It outlines how data and control signals flow through the application in response to user actions and system events. Each key feature is represented by an activity diagram, providing a visual map of sequential steps, decisions, and conditions.

These diagrams are created using Draw.io and assist developers in understanding and implementing features according to well-defined logic. The following features are modeled with dedicated activity diagrams:

- Goal Assistance Module
    - Task entry and intelligent task extraction
    - Smart reminder generation
    - Task status updates (e.g., in progress, completed)
- Focus Timer Module
    - Start and stop session
    - Session categorization
    - Timer completion feedback
- Emotion Diary Note Module
    - Daily mood logging

    - Sentiment tagging
    - Note editing and deletion
- User Module
    - Account creation and login
    - Profile editing and preference setting
- Report Module
    - Generation of daily, weekly, and custom reports
    - Data visualization and insights display

Sequence Diagram Design

Sequence diagram design represents the flow of interactions between different components of the BetterU mobile application from time to time. It mainly focuses on how the system parts such as frontend interface, backend server, database and external services interact with each other when being triggered by user actions. This helps for visualizing the order and timing of messages exchanged between these components, making it easier to understand the system's dynamic behavior during runtime.

Sequence diagrams illustrate the interaction between various system components over time. These diagrams detail how the frontend (Flutter), backend (FastAPI), and Firebase database collaborate to complete user requests and system processes.

Each diagram will include the key actors and lifelines for representing the system components with arrows showing what message exchanged between in chronological order. Some examples of the key scenarios can be modeled using sequence diagrams:

- Task entry and extraction
    - User submits task (via text or voice) → NLP processing → datetime extraction → task saved
- Reminder triggering
    - Scheduled reminder triggers notification at the right time
- Diary note submission
    - Entry saved → stored in Firebase → available for report
- Report generation
    - User opens report → system queries tasks logs → data visualized

State Chart Diagram Design

State chart diagrams are used to model the behavior of key objects as they transition through various states due to user interactions or system triggers. These diagrams help ensure reliable state management and consistent logic across the application.

Examples of key state transitions include:

- Task (Goal Assistance Module)
    - Pending → In Progress → Completed → Overdue
- Reminder (Goal Assistance Module)
    - Scheduled → Triggered → Snoozed
- Focus Session (Focus Timer Module)
    - Not Started → Active → Paused → Completed
- Diary Entry (Emotion Diary Note Module)
    - Draft → Created → Edited → Deleted

User Interface Design

User interface design is a critical aspect of BetterU, as it directly affects usability and user satisfaction. The design process begins in Figma, where all screens are prototyped before development in Flutter. Interfaces are crafted to be clean, responsive, and accessible, ensuring seamless navigation for students and working professionals.

Each module features consistent design elements and screen layouts, including:

- Goal Assistance Module: Task creation screen, smart suggestions, task list, and task editing interface
- Focus Timer Module: Timer dashboard, session summaries, category filters
- Emotion Diary Note Module: Mood selection interface, diary editor, history view
- User Module: Login screen, profile settings, onboarding steps
- Report Module: Graphical dashboards, productivity summaries, suggestion panels

Data Design

Data design is applied in BetterU development to define how the information is structured, stored and accessed within the mobile application. BetterU requires efficient and scalable data storage planning since it will be handling different types of object data such as user preferences, tasks items, reminders, posts, messages and other crucial information. Thus, a

Class Diagram, Entity Relationship Diagram (ERD) and data dictionary will be designed in advance before development. BetterU mobile application uses Google Firebase as the primary backend database storage. The characteristics of flexible and NoSQL document-based structure allows the mobile application to store various types of data when supporting real-time synchronization across devices. The data will be organized into collections while each collection will represent a category of data. Examples of key collections and their document structures include:

- **Tasks**
  - Attributes: task_id (PK), task_cat_id (FK), member_id (FK), title, description, priority, planned_start_time, planned_end_time, status
  - Associated with the TaskCategory and Member by task_cat_id and member_id
- **Emotion Diary Notes**
  - Attributes: diary_id (PK), member_id (FK), emotion_tag_id (FK), title, content, mood_level, context_tags, sentiment_score, created_date, is_momentary, is_protected, status
  - Associated with the Member and EmotionTag by member_id and emotion_tag_id
- **Focus Sessions**
  - Attributes: focus_session_id (PK), member_id (FK), task_id (FK), subtask_id (FK), nfc_tag_id (FK), start_time, end_time, duration, nfc_triggered, interruption_count, status, reward_earned
  - Associated with Member, Task, Subtask and NFCTag by member_id, task_id, subtask_id and nfc_tag_id
- **Person**
  - Attributes: person_id (PK), username, email, password, registered_date, status
  - Inherited by Member (member_id (PK), person_id (FK), anonymous_name, occupation, theme, busy_start_time and others) and Admin entity (admin_id (PK), person_id (FK), last_login_datetime, created_by_id)
- **Rewards**
  - Attributes: reward_id (PK), name, type, description, icon_url, points_required, created_datetime, status

Reports Design

The reports design is centered on generating visual summaries of user behavior and progress to facilitate self-reflection and promote well-being within the BetterU mobile application. Reports are generated using data collected from Google Firebase, aggregated from core modules including the Goal Assistance Module, Emotion Diary Note, and Focus Timer.

To ensure consistency and clarity across all reports, each report will include the following elements:

- A clear and descriptive title
- The report period or issue date
- Filter options that allow users to view reports by month, custom date range, or category
- A summary section presenting key statistics such as total tasks completed and time spent on activities
- Detailed records comprising task lists (categorized by completed, overdue, or in progress) and a breakdown of time usage
- Graphical representations such as bar charts, line graphs, and pie charts to illustrate trends and correlations

Additionally, three distinct types of reports will be produced:

- **Summary Reports:** These provide a concise overview of essential metrics, filtered by date or month, offering users a quick snapshot of their performance
- **Detailed Reports:** These contain comprehensive listings of entries, including individual tasks and time usage histories, enabling in-depth review
- **Exception Reports:** These highlight significant insights derived from user data, such as missed deadlines, disproportionate time allocation to specific tasks or activities, and indicators of low social engagement

This structured reporting approach empowers users to better understand their productivity patterns and emotional well-being, supporting continuous improvement and informed decision-making.

Functional Build 1: User & Goal Assistance

At the starting point of functional build 1, the testing strategy will be firstly outlined to ensure that the early builds can meet the quality standards. Various types of testing will be conducted such as unit testing and integration testing. Meanwhile, an appropriate test case design will also need to be prepared in advance. After that, a system overview session with the supervisor will be conducted for validating the design and clarifying the expectations for Build 1.

During the development phase of Build 1, the project will focus on the user module and goal assistance module. The main reason is that both modules are the core modules in BetterU mobile application. Since users must log in before accessing the useful functionalities in the mobile application, the user module was developed first to handle user registration, login, session management and account preferences. On the other hand, the goal assistance module also takes the same precedence and importance for allowing users to implement task input via text or voice, task extraction using NLP, task categorization and other basic task management features such as task editing, deleting and status updating.

After the development, the test plan and test cases will be created to evaluate the functionality and accuracy of existing functionalities such as task extraction and user registration. This can ensure that all the existing features developed can work properly and logically without any significant issues. Once the completion of development and testing, the build will be reviewed by the supervisor to assess functionality, adherence to requirements and areas for improvement. The feedback will be collected and act as the adjustment guidance during the next build. Via this build 1, the basic system and core task management can be delivered successfully. This is also a foundation for time usage tracker, focus timer and report generation features in future builds.

Functional Build 2: Time Usage Tracker, Emotion Diary Note, Focus Timer

In functional build 2, the development will focus on 3 key modules which are Time Usage Tracker, Emotion Diary Note and Focus Timer modules. The Time Usage Tracker Module monitors how users spend their time by tracking their app usage. Meanwhile, it also offers the NFC tag interactions for task clock-in and clock-in and clock-out features by integrating with the goal assistance module. It can also integrate with the focus timer module to achieve starting or stopping focus timer via NFC connectivity. Apart from that, the Emotion Diary Note module will also be developed during this build, it enables users to log their daily emotions, write reflective notes and receive the mood scores using the built-in NLP sentiment analysis. This helps for tracking the emotional patterns over time for self-awareness and well-being. Moreover, the Focus Timer module supports the work sessions by providing a gamified countdown timer with start, pause and stop functionality. It also includes the reward mechanisms or progress feedback to encourage productive behaviors. After the development of modules, all the modules will be undergoing a series of test plans and test cases to ensure the functionality and integration between modules. Not only that, the build will also be presented to the supervisor for evaluation for collecting feedback regarding system performance, UI behavior and overall integration.

Functional Build 3: Anonymous Community, Personal Chat, Report

Functional build 3 is the final functional build which focuses on implementing the social communication and reporting features in BetterU mobile application. The Anonymous Community, Personal Chat and Report modules will be involved in this build. The Anonymous Community module will allow users to post and respond anonymously within a community and forum. Users can submit the posts, view others' posts and comment while maintaining their privacy. For the Personal Chat module, it allows the user to conduct a secure one-to-one messaging with his or her selected trusted contacts. For the report module, it will generate the regular summaries based on user activity, task progress, time usage and social performance weekly and monthly. The reports will include the visual elements such as charts, completion rates and personalized suggestions to encourage the improvement of users.

3.2 Requirements Gathering Techniques

3.2.1 Observation

In order to better understand and explore real challenges faced by potential users, informal observations were conducted within my daily environment. These observations focused on individuals such as close friends, cousins, relatives' children, and myself. By doing so, relevant functional and non-functional requirements were identified and incorporated into the BetterU mobile application. This approach helps ensure that the application is capable of providing practical and effective solutions to the problems faced by its target users.

Poor Task Arrangement and Time Mismanagement

Through personal observation, I noticed that many of my friends often **struggle to arrange their tasks effectively**. This frequently leads to missed deadlines or a decline in the quality of their work. Even when they recognize the need for proper planning, their time management tends to be weak. Tasks that require minimal effort are sometimes given excessive attention, while more demanding tasks are rushed or postponed. This imbalance often leads to procrastination and overwhelming pressure near deadlines.

Trouble with Task Recording Habits

Some of my cousins and relatives' children often **lack consistent habits in recording their tasks**, whether academic or personal. They perceive the process of organizing and managing tasks as tedious. Instead of using structured tools or applications, they tend to rely on random paper notes or disorganized note-taking apps on their phones. Over time, these notes are either misplaced or become unreadable, **resulting in confusion and forgotten responsibilities**.

Distraction from Mobile Apps

Another issue I frequently observe among my friends is their tendency to **get distracted by entertainment apps while trying to focus on important tasks**. Without external reminders or supervision, they easily drift into endless scrolling on social media or mobile games. This significantly hampers their productivity and wastes valuable time. **As deadlines approach, the accumulated delays contribute to increased stress levels**.

Lack of Time for Social Life

One of my cousins, who works full-time, struggles to maintain a healthy work-life balance. He often leaves early in the morning and returns home late at night. After dinner, he usually spends a short time on social media before going to bed. He has shared with me how **difficult** it is **to keep in touch with his old friends or to find time for new meaningful social**

interactions. This ongoing lack of connection can eventually have a negative impact on mental well-being.

Fear of Reaching Out for Help

I also observed that some of my friends and cousins tend to avoid seeking help even when they encounter problems, especially in academic or work environments. Being more reserved, they **feel uncomfortable reaching out in group discussions or asking colleagues and peers for assistance**. They are often **worried about being judged or misunderstood**. This reluctance causes delays in resolving issues and leads to increased emotional pressure.

Emotional Instability

As for myself, balancing academic responsibilities, part-time work, and personal life often results in emotional instability. There are times when I feel **overwhelmed by the tight schedules and find it difficult to manage my time efficiently**. The lack of structured planning and emotional support can make it **challenging to stay focused, motivated, and mentally well**. This personal struggle further emphasizes the importance of having a mobile application like BetterU to support emotional management and task organization.

3.2.2 Online Research

To comprehensively explore the pain points and expectations of users when engaging with task management tools, a detailed online research study was carried out. This research involved gathering insights from user-generated reviews, blog discussions, productivity forums, and academic articles. Through this method, BetterU aims to uncover the real-world frustrations, preferences, and needs of potential users in diverse environments such as student life, freelancing, and personal productivity. The findings derived from this research help inform the system design by offering a user-centric foundation for functional and non-functional requirement development.

1. Most Task Management Apps Are Designed for Teams, Not Individuals

One prominent observation from online reviews and comparative studies is that most widely used task management tools, including Trello, Asana, and Notion, are primarily created with team collaboration in mind rather than for individual users. These platforms often offer a wide range of functionalities such as Gantt charts, collaborative boards, task assignment features, and shared team spaces. While these capabilities are beneficial in corporate or group project contexts, they may be excessive and even counterproductive for individuals who simply want to manage their own daily routines.

According to feedback from solo users and students, these tools tend to be cluttered with irrelevant functions such as team timelines, permissions, and workflow approvals, which complicate the user experience. Users have voiced that for personal use, they often find it difficult to navigate through these features or customize the app to a more simplified, private workspace (Amplework, 2024; Bateman, 2024; Lindy.ai, 2024). As a result, many users expressed a desire for tools that are streamlined, intuitive, and better suited for individual task planning.

2. Excessive Manual Effort Reduces Efficiency and Motivation

Another recurring theme identified from the research is the extensive amount of manual input required by many existing task management applications. Most apps require users to create tasks manually, assign due dates, update progress, set reminders, categorize items, and periodically reorganize their boards or lists. This constant administrative overhead not only demands time but also interrupts the user's cognitive flow, leading to decreased motivation and, in some cases, app abandonment.

Empirical studies and reports suggest that a significant portion of professionals' weekly working hours is consumed by repetitive manual tasks that could be automated using intelligent systems. According to Smartsheet (2023) and ProcessMaker (2024), employees spend up to 25 percent of their week on manual digital work that lacks automation. Furthermore, the inability of apps to learn user behavior or recommend intelligent actions compounds the issue, as users must continually perform the same repetitive steps. This inefficiency contradicts the core purpose of productivity tools and highlights a gap where AI-enhanced automation could add substantial value (TeamDynamix, 2024).

3. Difficulty in Task Prioritization and Cognitive Overload

A third critical issue users face relates to decision-making fatigue and poor task prioritization support. Although many task management applications allow users to enter tasks and deadlines, they generally do not provide intelligent guidance on how to prioritize activities or manage workloads based on urgency, time constraints, or personal energy levels. As a result, users are left to make complex decisions without system support, which can quickly become overwhelming.

Research by Nielsen Norman Group (2024) and IJSRD (2024) points out that users frequently encounter decision fatigue when required to manually assign importance to every task without contextual suggestions. This is especially problematic for students or busy professionals managing multiple responsibilities. On platforms like Reddit (2024), users reported that while their tools provided notification systems, they lacked mechanisms for real scheduling

assistance or time allocation planning. Thus, instead of functioning as productivity assistants, these tools often end up being static task repositories.

This issue is further compounded by the psychological burden associated with too many simultaneous choices, a phenomenon supported by cognitive psychology literature. Without adequate visual or algorithmic support to break down large to-do lists, users experience mental overload, reduced clarity, and often disengage from the tools altogether.

4. Emotional Discomfort and Lack of Anonymous Support Spaces

Beyond task execution features, emotional well-being emerged as an often-overlooked but essential aspect of digital productivity tool usage. During the online research, it became evident that users frequently deal with emotional challenges such as stress, guilt from unfinished tasks, or burnout. However, existing platforms generally lack built-in support mechanisms to address these issues in a private and safe manner.

Many users are hesitant to share their struggles or seek advice in online communities due to the lack of anonymity. Platforms like Reddit or productivity forums usually require user accounts and track digital footprints, which may discourage users from opening up about their psychological difficulties. According to studies conducted by Pew Research Center (2013) and Kiesler et al. (2013), individuals feel more secure expressing vulnerability when anonymity is guaranteed.

Concerns about data privacy in mental health and productivity-related apps further restrict users from using available emotional support features. The Journal of Medical Internet Research (2025) highlights that several mental health and wellness apps have weak data protection measures, leading users to mistrust their platforms. This highlights a vital requirement for BetterU to integrate anonymous communication spaces or journaling features that offer privacy and psychological safety.

3.2.3 Summary of Fact Gathering Results

The fact-finding phase of the BetterU project included two primary research methods: informal personal observation and structured online research. Together, these approaches provided a multifaceted understanding of the real-world challenges faced by potential users, especially students, young professionals, and individuals managing multiple responsibilities in their daily lives. The findings reveal a consistent pattern of difficulties across several key areas of task management and personal well-being.

1. Users Lack Effective Task Planning and Prioritization Strategies

Observational data revealed that many individuals, particularly students and young adults, struggle to plan their tasks efficiently. They often misjudge the time and effort required for different activities, resulting in procrastination, last-minute stress, and reduced output quality. This issue is further exacerbated by users' inability to prioritize tasks meaningfully based on urgency, complexity, or personal energy levels. Existing task management tools do not provide intelligent support for task sorting or workload balancing, thereby contributing to decision fatigue and cognitive overload.

2. Manual Workload and Lack of Automation Discourage Long-Term Use

Both personal observations and online reviews pointed to the burden of manual input as a major barrier to user engagement. Many users expressed frustration with having to constantly create, update, and reorganize their task entries. This administrative effort consumes time and interrupts the natural flow of work. Moreover, most apps do not adapt to user behavior or offer automated suggestions, which leads to repetitive actions and lower motivation over time. The absence of AI-assisted automation in existing systems is a significant gap that BetterU aims to address.

3. Inconsistent Task Logging Habits and Tool Aversion

Several individuals, especially among younger users such as students and teenagers, exhibit inconsistent or disorganized methods of recording their tasks. Rather than using structured platforms, they rely on fragmented tools like paper notes or default mobile apps, which are often lost or forgotten. Many users perceive task logging as tedious and unnecessary, reflecting a broader aversion to conventional productivity systems. This behavior suggests a need for more engaging, user-friendly interfaces that reduce friction in task entry and tracking.

4. Distraction and Lack of Focus Management Tools

A recurring issue observed among users is their susceptibility to digital distractions, particularly from mobile entertainment apps and social media. Without built-in accountability features, users frequently drift away from important tasks, leading to time mismanagement and elevated stress near deadlines. Current tools generally do not include mechanisms for managing distractions, such as focus timers, usage blockers, or motivational nudges. This missing functionality limits their effectiveness in supporting sustained attention and discipline.

5. Limited Support for Work-Life Balance and Social Interaction

Many users, especially working professionals, struggle to maintain a healthy work-life balance due to long work hours and tightly packed schedules. There is often little time left for

meaningful social interaction or personal reflection, which contributes to emotional fatigue and social isolation. Most task management tools focus narrowly on productivity metrics and fail to incorporate features that support overall well-being, such as time-blocking for personal activities or reminders to reconnect with friends.

6. Emotional Challenges and Barriers to Seeking Help

Users commonly face emotional discomfort related to workload pressure, failure to complete tasks, or fear of judgment when seeking help. Personal observations indicated that many individuals avoid asking for assistance, especially in academic or team settings, due to anxiety or social inhibition. Online research echoed this sentiment, with users expressing concern about lack of privacy in forums and mental health apps. The absence of anonymous and empathetic support channels further discourages users from addressing emotional difficulties openly.

Together, these findings underscore the importance of designing a mobile application that not only organizes tasks efficiently but also supports users holistically. BetterU should prioritize features that offer intelligent task automation, reduce manual workload, encourage consistent task logging, manage distractions, promote social well-being, and provide private emotional support. The insights gathered through observation and research form the foundation for defining the functional and non-functional requirements of the application, ensuring alignment with user needs and expectations.

3.3 Requirements Analysis

3.3.1 Use Case (UC) Diagram and Description

Use Case Diagram

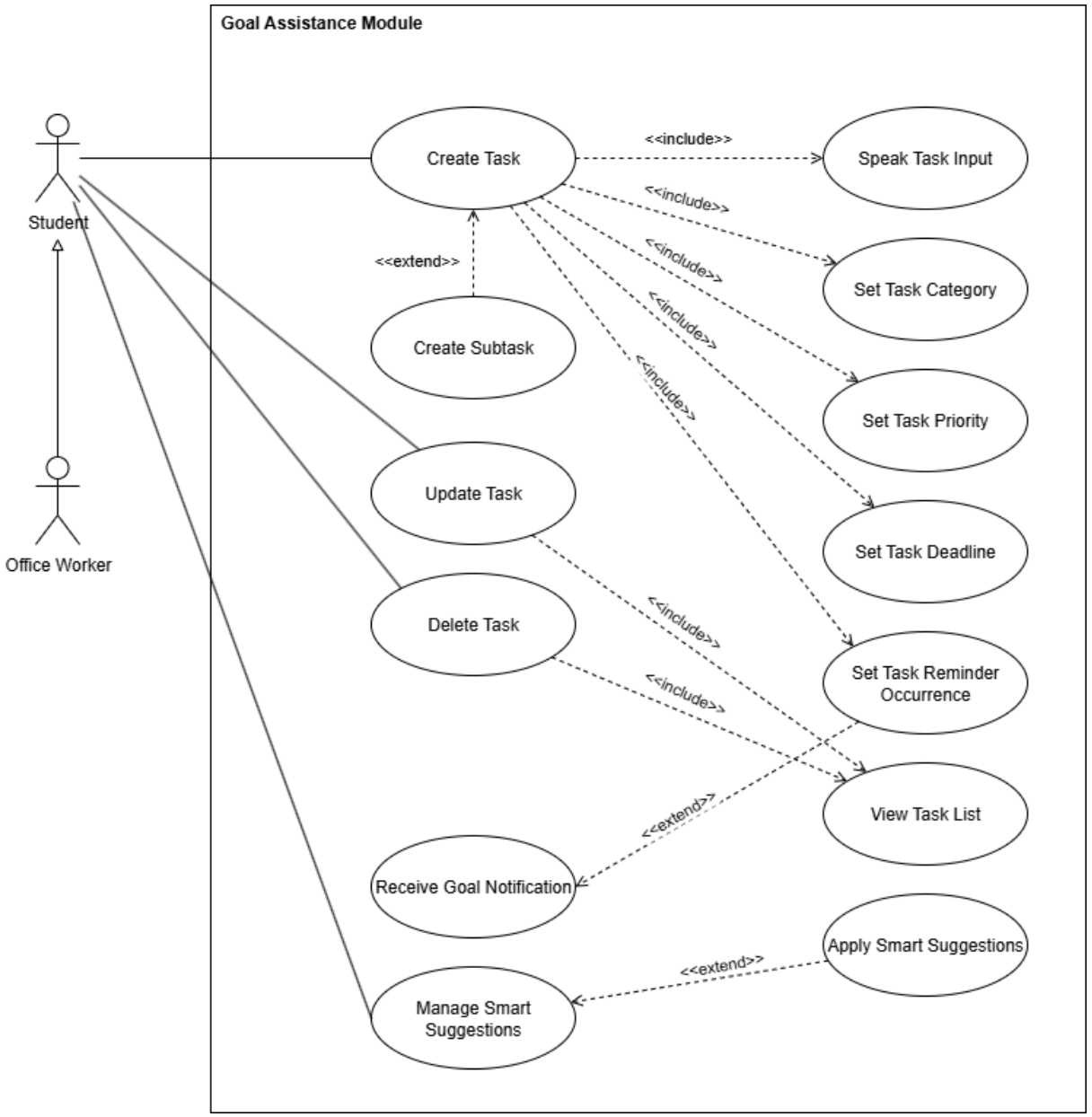


Figure 3.1.1: Use Case Diagram - Goal Assistant Module (shared)

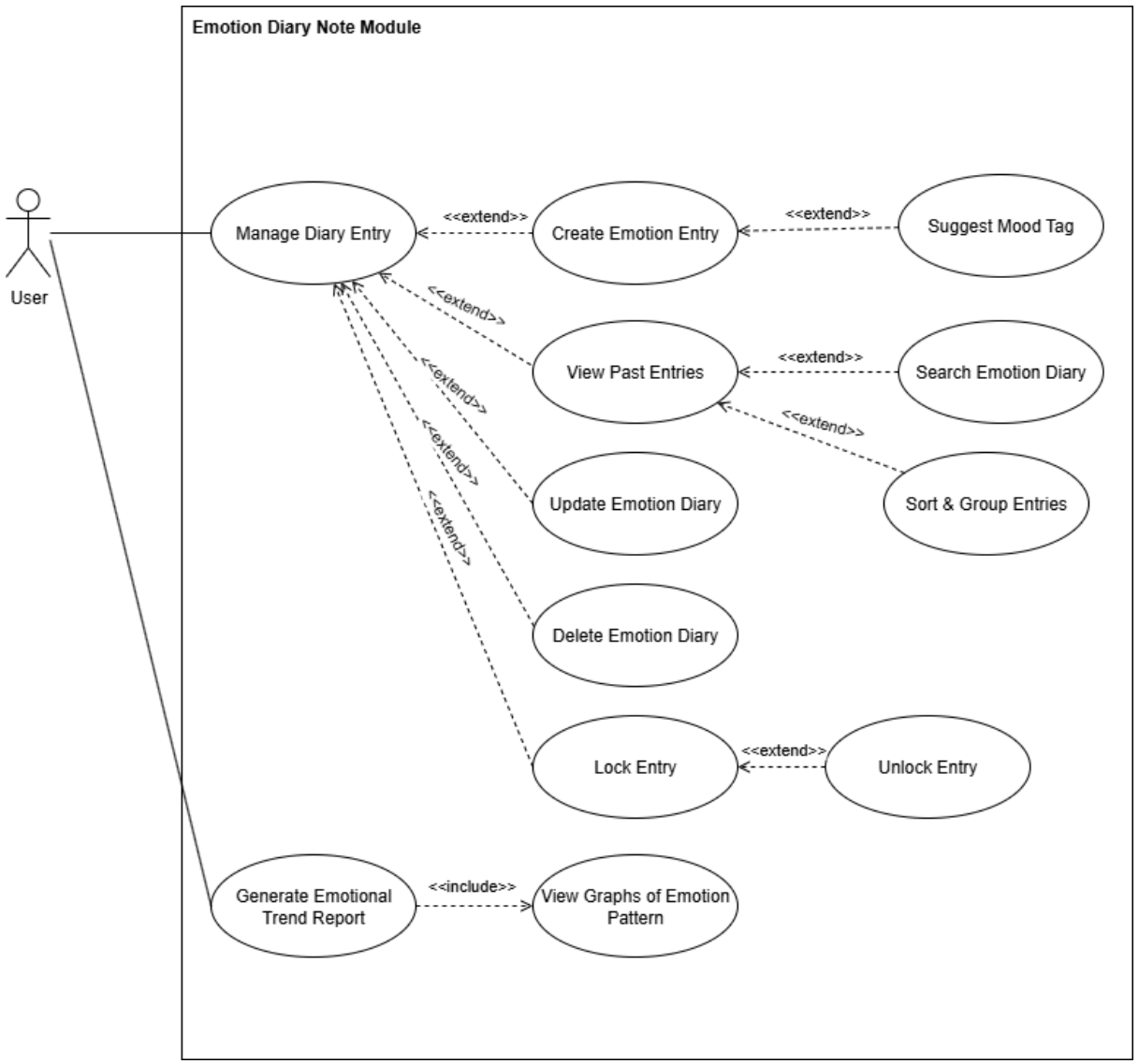


Figure 3.1.2: Use Case Diagram - Emotion Diary Note Module

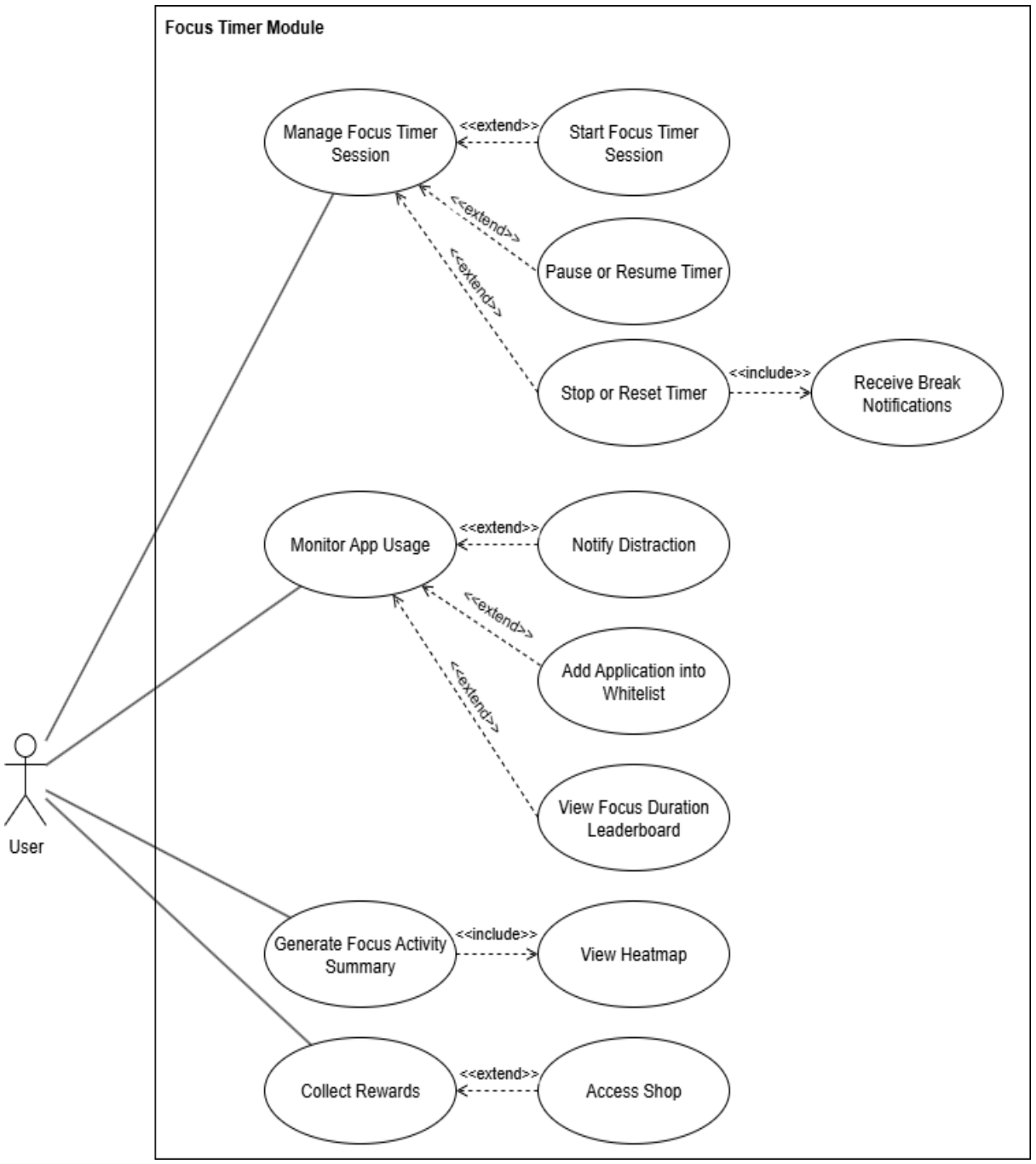


Figure 3.1.3: Use Case Diagram - Focus Timer Module

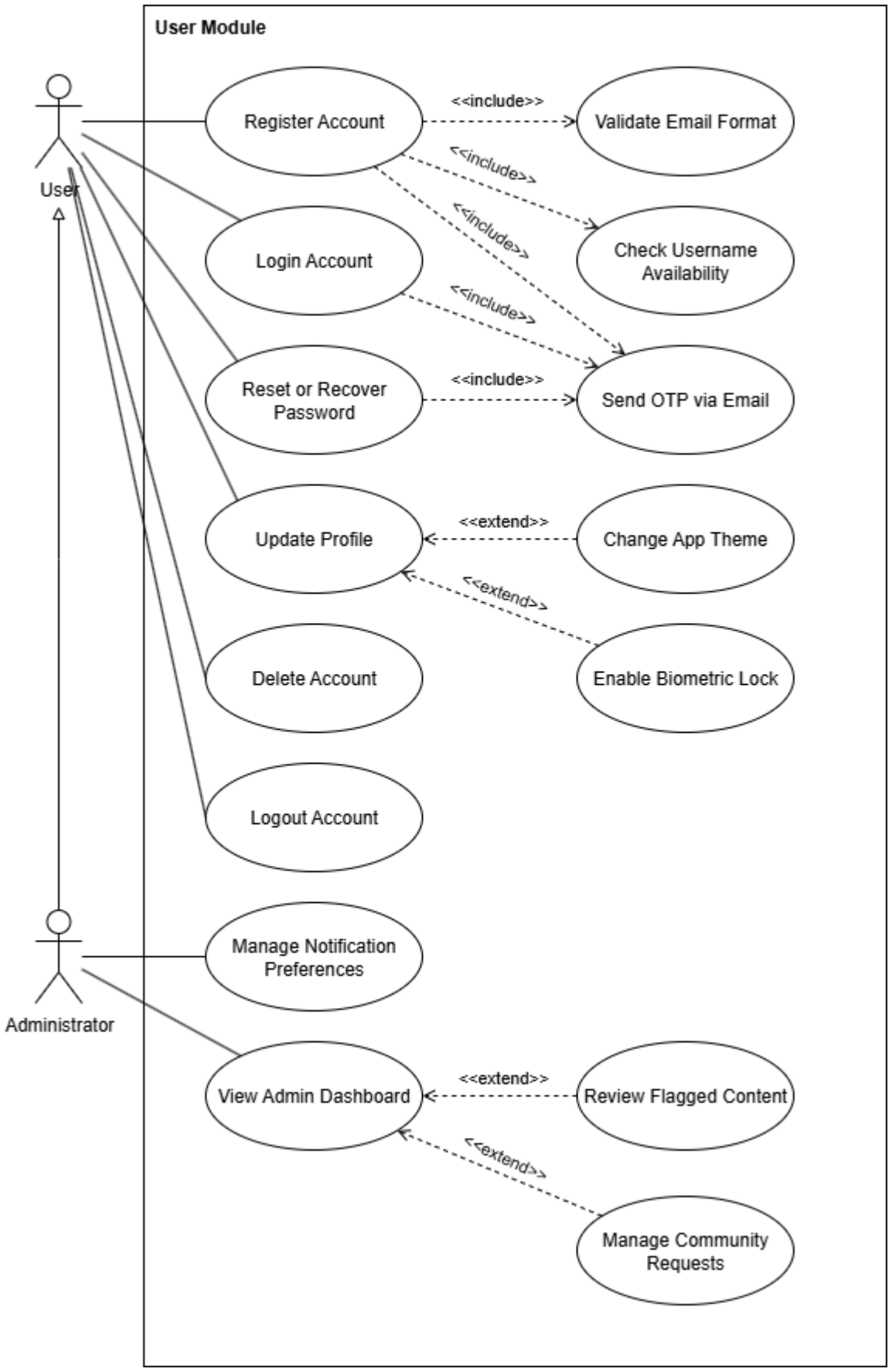


Figure 3.1.4: Use Case Diagram - User Module

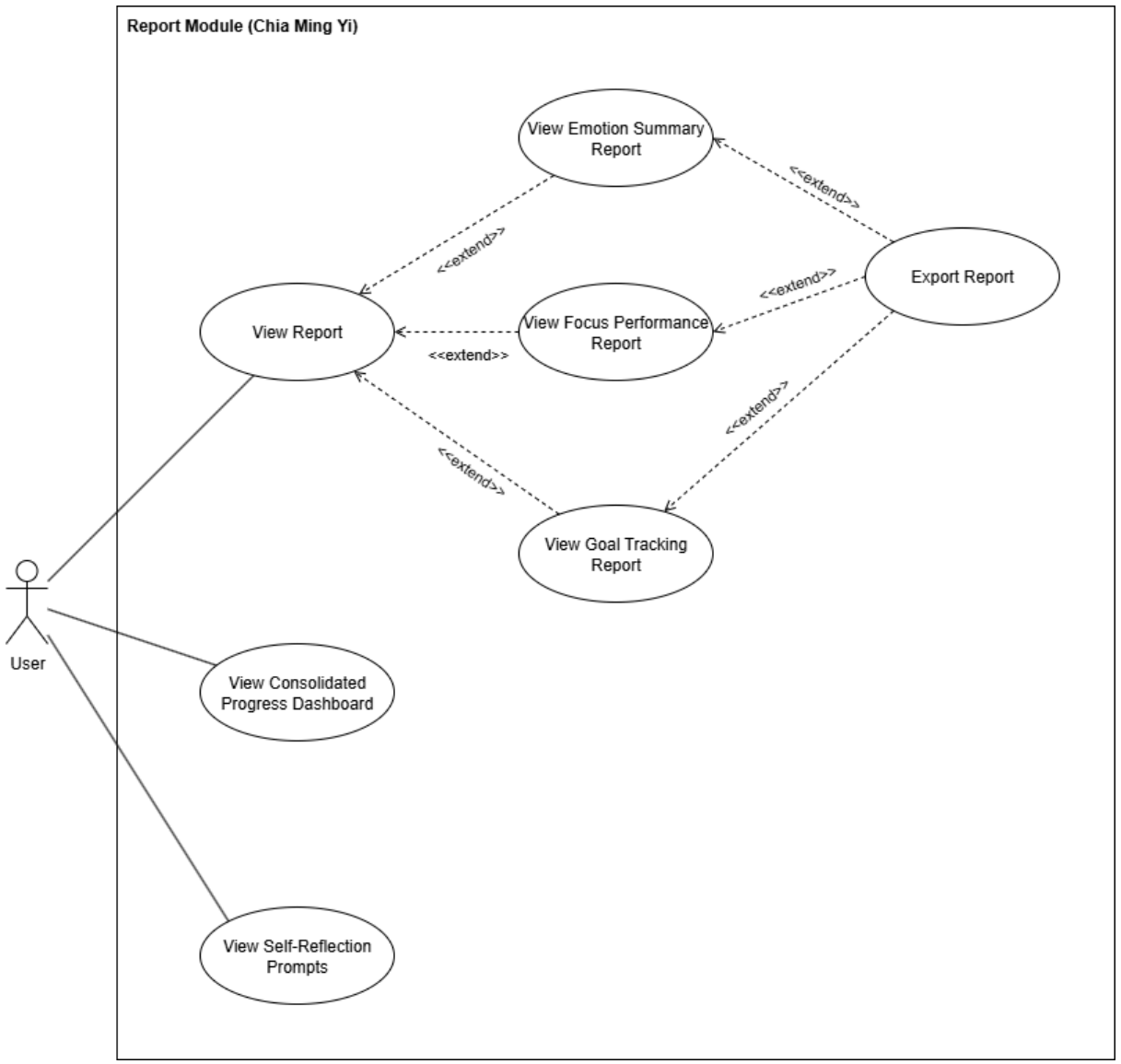


Figure 3.1.5: Use Case Diagram - Report Module

Use Case Description

Goal Assistance Module (shared) :

<table>
<tr><td colspan="2">Name of Use Case: Create Task</td></tr>
<tr><td colspan="2">Brief Description: The user can create a new task with subtasks by entering task details such as title, description, deadline, category and priority level. The system validates the input, stores the task and displays it in the task list.</td></tr>
<tr><td colspan="2">Actor: Student, office worker</td></tr>
<tr><td colspan="2">Precondition: The user must be logged into the system and the role is member.</td></tr>
<tr><th>Actor Action</th><th>System Response</th></tr>
<tr><td>1. The user selects the "Task" tab.</td><td>2. The system displays the task list view.</td></tr>
<tr><td>3. The user selects the "Create Task" button from the task menu.</td><td>4. The system displays the task creation form.</td></tr>
<tr><td>5. The user enters task title, description, deadline, priority level, categories and task reminder occurrence via text input and voice input.</td><td>6. The system validates the input fields.</td></tr>
<tr><td>7. The user adds one or more subtasks with individual titles and due dates.</td><td>8. The system adds each subtask to the main task structure.</td></tr>
<tr><td>9. The user clicks the "Create" button.</td><td>10. The system saves the task along with any linked subtasks to the database.</td></tr>
<tr><td></td><td>11. The system displays a confirmation message and shows the task in the task list.</td></tr>
<tr><td colspan="2">Alternative Flow:

A1. Step 6. If the user has left the task title field empty and attempts to save the task, the system will highlight the missing fields and display an error message "You must fill in the task title" which indicates that the required field must be filled. If the user has chosen an invalid date or deadline in the past, the system will display an error message "The date should not be before today's date." to prompt the user to correct the deadline.

A3. Step 7. If the user chooses not to add any subtasks, the system will proceed to save the task without any subtasks.</td></tr>
<tr><td colspan="2">Postcondition: A new task is successfully created and added to the user's task list. The user can directly view the new added task in the task list view.</td></tr>
</table>

Table 3.1.1: Use Case Description - Create Task of Goal Assistance Module

Name of Use Case: Update Task	
Brief Description: The user can modify the details of an existing task such as its title, description, deadline, category, priority levels and subtasks.	
Actor: Student, office worker	
Precondition: The user must be logged into the system and the role is member. The user must have created at least one task in the task list.	
Actor Action	**System Response**
1. The user selects the "Task" tab.	2. The system displays the task list view.
3. The user selects the existing task item from the task list.	4. The system displays the task details in editable fields.
5. The user modifies the task details such as title, description, due date and subtasks.	6. The system validates the modified input.
7. The user presses the "Save" button.	8. The system saves the updated task to the database.
	9. The system displays the updated task in the task list.
Alternative Flow: A1. Step 6. If the user has left the task title field empty and attempts to save the task, the system will highlight the missing fields and display an error message "You must fill in the task title" which indicates that the required field must be filled. If the user has chosen an invalid date or deadline in the past, the system will display an error message "The date should not be before today's date." to prompt the user to correct the deadline.	
Postcondition: A task is successfully updated and displayed to the user's task list. The user can directly view the updated task in the task list view.	

Table 3.1.2: Use Case Description - Update Task of Goal Assistance Module

Name of Use Case: Delete Task	
Brief Description: The user can remove an existing task and its associated subtasks if any from the task list.	
Actor: Student, office worker	
Precondition: The user must be logged into the system and the role is member. The user must have created at least one task in the task list.	
Actor Action	**System Response**

1. The user selects the "Task" tab.	2. The system displays the task list view.
3. The user long presses on the existing task item from the task list.	4. The system pops out an options bar.
5. The user selects the "Delete" option from the options bar.	6. The system prompts the user with a confirmation dialog "Are you sure to delete this task? The task will be removed along with its associated subtasks.".
7. The user selects the "Confirm" button in the confirmation dialog.	8. The system removes the task and associated data, then updates the task list view.
Alternative Flow: A1. Step 7. If the user selects on the "Cancel" button in the confirmation dialog, the system will retain the task and return back to the task list view.	
Postcondition: The selected task and its associated data are permanently deleted from the system.	

Table 3.1.3: Use Case Description - Delete Task of Goal Assistance Module

Name of Use Case: Receive Goal Notifications	
Brief Description: The system sends reminder notifications to the user when a task's scheduled occurrence time is reached.	
Actor: Student, office worker	
Precondition: The user must be logged into the system and the role is member. The user has already created a task and has set a reminder occurrence time.	
Actor Action	**System Response**
	1. The system periodically checks tasks for upcoming reminder times.
	2. The system detects a task with a scheduled reminder.
	3. The system pushes notification to the user about the task.
4. The user taps the task reminder notification.	5. The system navigates the user to the task detail interface.
Alternative Flow: A1. Step 4. If the user ignores the task reminder notification, the system will not navigate the user to the task detail interface.	

Postcondition: The user receives a task reminder and can access the task's detail view through the notification.

Table 3.1.4: Use Case Description - Receive Goal Notifications of Goal Assistance Module

Name of Use Case: Manage Smart Suggestion	
Brief Description: When a user creates or updates a task, the system will evaluate it. If needed, the system will provide a smart suggestion to improve scheduling or productivity. The user may choose to apply or ignore the suggestion.	
Actor: Student, office worker	
Precondition: The user must be logged into the system and the role is member. The user has created or updated a task and enabled the smart suggestion analysis.	
Actor Action	**System Response**
1. The user creates or updates a task.	2. The system displays successful operation dialog once the task creation or updation is successful.
	3. The system analyzes the task for optimization needs.
	4. The system displays a pop-up at the bottom of the task list view.
5. The user taps on "Apply" on the suggestion pop-out.	6. The system performs the modification and collapses the suggestions.
	7. The system updates the task list view.
Alternative Flow: A1. Step 4. If there is no optimization needed, the system will not display the suggestion pop-up at the bottom of the task list view. A2. Step 5. If the user taps on the "Ignore" button on the suggestion pop-out, the system will only collapse the suggestions. The users can apply the suggestion in the future via expanding the collapsed suggestions at the bottom of the task list view if needed.	
Postcondition: The smart suggestion is either applied or ignored, its status is collapsed and hidden below the task list. The task is updated accordingly if applied.	

Table 3.1.5: Use Case Description - Manage Smart Suggestion of Goal Assistance Module

Emotion Diary Note Module:

Name of Use Case: Manage Diary Entry	
Brief Description: The user can manage their emotion diary by creating, viewing, updating, deleting, and locking/unlocking entries. The system validates inputs and updates the diary records accordingly.	
Actor: User	
Precondition: The user must be logged into the system and the role is member.	
Actor Action	**System Response**
1. The user selects the “Emotion Diary” module.	2. The system displays the Emotion Diary dashboard.
3. The user chooses to create a new entry.	4. The system opens the diary input form.
5. The user writes thoughts and selects a mood tag.	
6. The user submits the diary entry.	7. The system validates and saves the entry.
8. The user selects a specific date of the past diary in the calendar view.	9. The system displays the specific past diary overview.
10. The user uses search or filter options.	11. The system filters entries accordingly.
12. The user selects an entry to update or delete.	13. The system opens the entry for modification or deletion.
14. The user locks or unlocks an entry.	15. The system updates the lock status and requests biometric if enabled.
Alternative Flow: A1. Step 5: If the user submits the form without entering content, the system displays: "Diary content cannot be empty." A3. Step 15: If the user attempts to unlock a locked entry without biometric authentication, the system shows: "Biometric verification failed. Try again."	
Postcondition: The diary entry is created, updated, deleted, or locked/unlocked as requested and stored in the database.	

Table 3.1.6: Use Case Description - Manage Diary Entry of Emotion Diary Note Module

Name of Use Case: Generate Emotional Trend Report	
Brief Description: The user views emotional patterns through visual graphs generated from existing diary data.	
Actor: User	
Precondition: The user must have at least one existing diary entry.	
Actor Action	**System Response**
1. The user selects “Generate Emotional Trend Report”.	2. The system displays emotion heatmaps, line charts, and emotion frequencies.
Alternative Flow: A1. Step 2: If the system finds no entries, it displays: "You need at least one diary entry to generate a report." A2. Step 2: If the graph fails to load due to connectivity issues, the system displays: "Unable to load graph. Please check your connection."	
Postcondition: A visual trend report is successfully displayed to the user.	

Table 3.1.7: Use Case Description - Generate Emotional Trend Report of Emotion Diary Note Module

Focus Timer Module:

Name of Use Case: Manage Focus Timer Session	
Brief Description: The user can manage a focus timer session by starting, pausing, resuming, stopping, or resetting it. The system tracks the session time, notifies about breaks, and saves focus duration data.	
Actor: User	
Precondition: The user must be logged into the system and the role is member.	
Actor Action	**System Response**
1. The user navigates to the “Focus Timer” module.	2. The system displays the focus timer interface.
3. The user clicks the “Start” button.	4. The system starts the timer countdown and logs the session.
5. The user clicks “Pause” during the session.	6. The system pauses the countdown.
7. The user clicks “Resume”.	8. The system resumes the timer from the paused time.

9. The user clicks “Stop” or “Reset”.	10. The system stops the session, clears or reset the timer.
11. The user finished its focus timer session.	12. The system records the session duration and updates the session history.
Alternative Flow: A1. Step 4: If the session duration set is less than 5 minutes, the system displays: “Minimum session must be 5 minutes.” A2. Step 5: If the user attempts to pause a timer that hasn't started, the system displays: “You must start a session before pausing.” A4. Step 11: If the network is unavailable, the system stores session data locally and syncs it once reconnected.	
Postcondition: The focus timer session is successfully recorded, and break notifications are handled. The session data is available for reporting.	

Table 3.1.8: Use Case Description - Manage Focus Timer Session of Focus Timer Module

Name of Use Case: Monitor App Usage	
Brief Description: The user can receive distraction alerts during focus time, manage a whitelist of allowed apps, and view leaderboard stats related to focus performance.	
Actor: User	
Precondition: The user must be logged in and have a session in progress.	
Actor Action	**System Response**
1. The user begins a focus session.	2. The system monitors app usage in the background.
3. The user opens a distracting app.	4. The system displays a notification: “You are getting distracted.”
5. The user adds an app to the whitelist.	6. The system adds the selected app to the allowed list.
7. The user selects “Leaderboard”.	8. The system shows focus duration rankings among users.
Alternative Flow: A1. Step 4: If the user disables notification permission, the system cannot alert and logs: “Distraction alert blocked by device settings.” A2. Step 5: If the app cannot be added to the whitelist (e.g., invalid app ID), the system shows: “This app cannot be whitelisted.”	

A3. Step 8: If leaderboard data fails to load, the system displays: “Unable to retrieve leaderboard. Please check your connection.”
Postcondition: App usage is monitored and alerts/leaderboard/whitelist functions operate as expected.

Table 3.1.9: Use Case Description - Monitor App Usage of Focus Timer Module

Name of Use Case: Generate Focus Activity Summary	
Brief Description: The user can view a heatmap of their focus activity, including session frequency and duration by time of day.	
Actor: User	
Precondition: The user must be logged into the system and the role is member.	
Actor Action	**System Response**
1. The user selects “Focus Activity Summary”.	2. The system processes focus data and generates a heatmap with highlights high and low productivity periods using color intensity.
Alternative Flow: A1. Step 2: If the user has no focus session data, the system shows: “No focus activity found. Start a session to generate a summary.” A2. Step 2: If the heatmap fails to load due to data corruption or rendering error, the system displays: “Unable to display heatmap. Please try again later.”	
Postcondition: The user receives a visual summary of their productivity patterns.	

Table 3.1.10: Use Case Description - Generate Focus Activity Summary of Focus Timer Module

Name of Use Case: Collect Rewards	
Brief Description: After completing focus sessions, the user may earn points and access the shop to redeem rewards.	
Actor: User	
Precondition: The user must have completed at least one focus session.	
Actor Action	**System Response**
1. The user selects “Collect Rewards”.	2. The system checks earned focus points.

3. The user accesses the reward shop.	4. The system displays available items for redemption.
5. The user selects a reward and confirms.	6. The system deducts points and completes redemption.
Alternative Flow: A1. Step 4: If the user has no points, the system displays: “You don’t have enough points to redeem rewards.”	
Postcondition: Rewards are redeemed successfully, and points are updated.	

Table 3.1.11: Use Case Description - Collect Rewards of Focus Timer Module

User Module:

Name of Use Case: Register Account	
Brief Description: The user registers an account by providing personal credentials. The system validates the inputs and creates a new account.	
Actor: User	
Precondition: The user must not be logged in or already registered.	
Actor Action	**System Response**
1. The user selects “Register”.	2. The system displays the registration form.
3. The user enters email, username, and password.	4. The system validates the email format and username availability.
5. The user submits the form.	6. The system creates a new account and redirects to the login screen.
Alternative Flow: A1. Step 4: If the email format is invalid, the system shows: “Invalid email format.” A2. Step 4: If the username is already taken, the system displays: “Username not available.” A3. Step 6: If account creation fails due to a network issue, the system displays: “Registration failed. Please try again later.”	
Postcondition: A new user account is successfully created and ready for login.	

Table 3.1.12: Use Case Description - Register Account of User Module

Name of Use Case: Login Account

Brief Description: The user logs into the system using valid credentials.	
Actor: User	
Precondition: The user must have a registered account.	
Actor Action	**System Response**
1. The user selects “Login”.	2. The system displays the login form.
3. The user enters the email and password.	4. The system validates the credentials.
5. The user submits the form..	6. The system navigates the user to the home page.
Alternative Flow: A1. Step 4: If credentials are incorrect, the system shows: “Invalid email or password.” A2. Step 6: If the system cannot connect to the server, it shows: “Login failed. Please check your internet connection.”	
Postcondition: The user is logged in and redirected to the main interface.	

Table 3.1.13: Use Case Description - Login Account of User Module

Name of Use Case: Reset or Recover Password	
Brief Description: The user requests to reset a forgotten password. A verification code is sent via email.	
Actor: User	
Precondition: The user must provide a valid registered email address.	
Actor Action	**System Response**
1. The user selects “Forgot Password”.	2. The system prompts for email input
3. The user enters their email.	4. The system sends an OTP to the user’s email.
5. The user enters the OTP and sets a new password.	6. The system validates and updates the password.
Alternative Flow: A1. Step 3: If the email is not registered, the system shows: “No account found with this email.” A2. Step 4: If email sending fails, the system shows: “Failed to send OTP. Try again.” A3. Step 5: If the OTP is incorrect, the system shows: “Invalid verification code.”	

Postcondition: The user's password is successfully updated.

Table 3.1.13: Use Case Description - Reset or Recover Password of User Module

Name of Use Case: Update Profile	
Brief Description: The user must be logged in.	
Actor: User	
Precondition: The user must not be logged in or already registered.	
Actor Action	**System Response**
1. The user navigates to Profile settings.	2. The system displays the current profile information.
3. The user selects Edit Profile.	4. The system displays editable fields for profile details.
5. The user updates the desired information (e.g., name, email, phone, password, profile picture).	6. The system validates the input formats and checks for conflicts (e.g., duplicate email/username).
7. The user saves the changes.	8. The system updates the profile information in the database.
	9. The system confirms the update with a success message and displays the updated profile.
Alternative Flow: A1. Step 3: If the email/phone format is invalid, the system shows: "Invalid format." A2. Step 3: If the new password is weak, the system shows: "Password does not meet complexity requirements." A3. Step 3: If the uploaded profile picture is in an unsupported format, the system shows: "Invalid file format." A4. Step 4: If the update fails due to a system error, the system shows: "Update failed. Please try again."	
Postcondition: The user's profile is updated.	

Table 3.1.14: Use Case Description - Update Profile of User Module

Name of Use Case: Delete Account
Brief Description: The user can permanently delete their account after confirmation.

Actor: User	
Precondition: The user must be logged in and confirm account deletion.	
Actor Action	**System Response**
1. The user selects “Delete Account” from settings.	2. The system prompts a confirmation dialog.
3. The user confirms the deletion.	4. The system deletes the account and logs the user out.
Alternative Flow: A1. Step 3: If the user cancels, the system returns to the settings page without deleting the account. A2. Step 4: If deletion fails due to server or connection issues, the system shows: “Unable to delete account. Try again later.”	
Postcondition: The account is permanently removed, and the user is redirected to the welcome screen.	

Table 3.1.15: Use Case Description - Delete Account of User Module

Name of Use Case: Manage Notification Preferences	
Brief Description: The user can manage push notifications related to reminders, updates, and goals.	
Actor: User	
Precondition: The user must not be logged in or already registered.	
Actor Action	**System Response**
1. The user selects “Notification Preferences” from settings.	2. The system displays available notification categories.
3. The user toggles specific notification types.	4. The system updates preferences and confirms changes.
Alternative Flow: A1. Step 4: If saving preferences fails, the system shows: “Unable to update notification settings, Try again.”	
Postcondition: Notification preferences are successfully updated and saved.	

Table 3.1.16: Use Case Description - Manage Notification Preferences of User Module

Name of Use Case: Logout Account	
Brief Description: The user logs out of the system to end their current session.	
Actor: User	
Precondition: The user must be logged into the system.	
Actor Action	**System Response**
1. The user selects the “Log Out” option from the settings or main menu.	2. The system prompts for confirmation.
3. The user confirms log out.	4. The system ends the session, clears local session data, and redirects to the login or welcome screen.
Alternative Flow: A1. Step 2: If the user cancels the confirmation, the system returns to the previous screen and does not log out. A2. Step 4: If the log out process fails due to system error, the system shows: “Log out failed. Please try again.”	
Postcondition: The user is securely logged out, and all session data is cleared from the device.	

Table 3.1.17: Use Case Description - Logout Account of User Module

Report Module:

Name of Use Case: View Report	
Brief Description: The user can view three types of reports: Emotion Summary Report, Focus Performance Report, and Goal Tracking Report. Each report includes visual insights and can be exported.	
Actor: User	
Precondition: The user must be logged in and have relevant data (diary entries, focus sessions, or goals) available.	
Actor Action	**System Response**
1. The user selects the “Reports” option from the main menu.	2. The system displays available report types: Emotion, Focus, and Goal.
3. The user selects “Emotion Summary Report”.	4. The system generates emotion statistics and graphs.
5. The user clicks “Export Report”.	6. The system creates a downloadable

	file (PDF/CSV/Image).
7. The user selects “Focus Performance Report”.	8. The system shows charts summarizing total focus hours, session distribution, and improvement trends.
9. The user clicks “Export Report”.	10. The system exports the focus performance report.
11. The user selects “Goal Tracking Report”.	12. The system displays visual data on goal completion, progress, and categories.
13. The user clicks “Export Report”.	14. The system generates the downloadable goal report file.
Alternative Flow: A1. Step 4: If no emotion data exists, the system shows: “No emotion data available. Please write diary entries to view this report.” A2. Step 6: If export fails due to a system or network error, the system shows: “Failed to export report. Please try again later.” A3. Step 8: If no focus sessions have been recorded, the system displays: “No focus session data available.” A4. Step 10: If export fails due to file permission issues, the system displays: “Unable to save file. Please allow device storage permissions.” A5. Step 12: If the user hasn’t tracked any goals, the system shows: “No goal tracking data available.” A6. Step 14: If export succeeds but the user cancels download, the system logs: “Export canceled by user.”	
Postcondition: The user successfully views and/or exports reports related to their emotional state, focus performance, and goal achievement based on existing app data.	

Table 3.1.18: Use Case Description - View Report of Report Module

Name of Use Case: View Consolidated Progress Dashboard	
Brief Description: The user can view a unified dashboard showing a summary of emotion trends, focus patterns, and goal progress in one place.	
Actor: User	
Precondition: The user must be logged in and have at least one type of data (diary, focus, or goal) available.	
Actor Action	**System Response**

1. The user selects “Dashboard” from the report or home screen.	2. The system loads and displays a consolidated view of emotion, focus, and goal data.
3. The user scrolls or navigates through the dashboard sections.	4. The system presents charts, stats, and highlights for each module (e.g., top moods, total focus time, goals completed).
Alternative Flow: A1. Step 2: If the system fails to load data due to connectivity issues, it displays: “Failed to load dashboard. Please check your internet connection.” A2. Step 2: If no relevant data is found in all modules, the system displays: “No data available to generate dashboard. Start using the app to see progress here.” A3. Step 4: If data from one module (e.g., focus) is missing, that section will be grayed out with a message: “No focus data yet.”	
Postcondition: The user views an integrated overview of their personal progress across the app modules.	

Table 3.1.19: Use Case Description - View Report of Report Module

Name of Use Case: View Self-Reflection Prompts	
Brief Description: The user is presented with AI-generated or pre-written self-reflection prompts based on recent emotional patterns and app usage.	
Actor: User	
Precondition: The user must be logged into the system and have at least one recent diary entry or emotion record.	
Actor Action	**System Response**
1. The user selects “Self-Reflection Prompts” from the report or diary module.	2. The system analyzes recent emotion data and generates tailored reflection prompts.
3. The user reads through the prompts.	4. The system offers an option to respond or write a diary entry based on the prompt.
Alternative Flow: A1. Step 2: If no emotional data is found in the past week, the system shows: “No recent emotional patterns found. Write a diary entry to receive reflections.”	
Postcondition: The user receives meaningful self-reflection prompts and may optionally respond through a diary note.	

Table 3.1.20: Use Case Description - View Self-Reflection Prompts of Report Module

3.3.2 Functional Requirements

1.0 Emotion Diary Note Module

1.1 Diary Entry Management

1.1.1 The system shall allow users to create, edit, delete, and view diary entries.

1.1.2 The system shall support multiple entries "momentary" per day for quick reflections and future review.

1.1.3 The system shall support diary input via text, emojis, voice (speech-to-text), and image.

1.1.4 The system shall allow users to optionally attach images to diary entries.

1.1.5 The system shall automatically capture and display the creation date and time of each entry.

1.1.6 The system shall allow users to search entries by title, date, mood tags, or keyword highlights.

1.1.7 The system shall support sorting entries by creation date/time or mood.

1.1.8 The system shall support grouping entries by mood or by creation date range using a calendar view.

1.2 Sentiment Analysis & Mood Tagging

1.2.1 The system shall analyze diary content using AI-powered sentiment analysis (e.g., via Scikit-learn or NLP libraries).

1.2.2 The system shall automatically suggest mood tags based on detected emotional tone (e.g., Neutral, Happy, Sad, Anxious, Fearful, Angry, Mixed).

1.2.3 The system shall highlight keywords that indicate possible causes of stress, problems encountered, or sources of happiness.

1.2.4 The system shall allow users to manually edit mood tags and keyword highlights.

1.3 Privacy Control

1.3.1 The system shall offer optional PIN or biometric (e.g., fingerprint) protection for accessing or locking sensitive diary entries.

2.0 Focus Timer Module

2.1 Focus Session Control

2.1.1 The system shall allow users to select a task and specify a focus duration (15, 25, 45, 60 minutes, or custom).

2.1.2 The system shall support pause, cancel, resume, and reset controls during focus sessions.

2.1.3 The system shall support focus tracking tied to specific tasks for progress accumulation.

2.1.4 The system shall support phone flip detection to initiate or maintain focus mode.

2.1.5 The system shall display a visual heatmap (similar to GitHub contribution graph) to represent daily focus completion history.

2.1.6 The system shall allow users to view their session history, rewards earned, and personal progress trends.

2.2 Gamification

2.2.1 The system shall trigger start/end sound effects and visual animations during focus sessions.

2.2.2 The system shall reward users with virtual “treasures” upon completing a focus session, where both the probability of obtaining rare items and the total number of rewards shall scale with the session duration (e.g., longer sessions yield higher drop rates and multiple items).

2.2.3 The system shall allow users to collect virtual “treasures” and convert duplicate items into gems.

2.2.4 The system shall include a shop where users can exchange gems for cosmetic or premium features (e.g., limited-time avatar frames, wallpapers, feature trial days).

2.2.5 The system shall track and display daily focus streaks to encourage consistency.

2.3 Distraction Prevention

2.3.1 The system shall monitor app usage during focus sessions and pause the timer if apps outside the whitelist are accessed.

2.3.2 The system shall notify users with a warning if distraction rules are violated.

2.3.3 The system shall allow users to manage their app whitelist manually.

2.3.4 The system shall maintain a leaderboard showing top focus durations within the user's community (e.g., StudyTogetherCommunity) and globally (Overall Top 10).

3.0 User Module

3.1 User Registration & Authentication

3.1.1 The system shall allow new users to register by providing a username, password, email, gender, and occupation.

3.1.2 The system shall validate that the email is in the correct format and enforce a strong password policy.

3.1.3 The system shall send a One-Time Password (OTP) to the user’s email for verification during registration.

3.1.4 The system shall allow users to log in using either their username or email together with their password.

3.1.5 The system shall provide a “Forgot Password” function that sends an OTP to the user’s email for password recovery.

3.1.6 The system shall check for duplicate usernames and notify the user if the chosen username is already taken.

3.2 User Profile & Preferences

3.2.1 The system shall allow users to update their profile information, including profile image, avatar frame, username, gender, and occupation.

3.2.2 The system shall ensure that usernames remain unique when users update them.

3.2.3 The system shall assign user roles such as User, VIP, or Admin and restrict access to features based on these roles.

3.2.4 The system shall grant VIP users access to exclusive features, including unlimited AI functionalities.

3.2.5 The system shall allow users to select application themes, such as light mode or dark mode.

3.2.6 The system shall allow users to configure notification preferences, including the option to silence notifications.

3.2.7 The system shall allow users to enable or disable PIN or fingerprint protection for diary entries.

3.3 Administration

3.3.1 The system shall provide administrators with a dashboard that displays the total number of active users along with user retention metrics.

3.3.2 The system shall allow administrators to view and manage new requests for community group creation or joining.

3.3.3 The system shall enable administrators to review and moderate flagged posts, messages, and user accounts.

3.4 Session Control

3.4.1 The system shall manage secure login sessions using token-based authentication with automatic timeout and refresh mechanisms.

3.4.2 The system shall provide users with the option to log out manually and terminate all active sessions.

4.0 Report Module

4.1 Insight Generation

4.1.1 The system shall generate weekly, monthly, and yearly summaries that reflect the user's completed and pending tasks, focus sessions, and emotional records.

4.1.2 The system shall allow users to view and filter reports based on custom time ranges and selected data types, such as goals, focus duration, or mood trends.

4.1.3 The system shall provide a visual comparison between to-do tasks and completed tasks to help users evaluate their productivity.

4.2 Data Visualization

4.2.1 The system shall present data using various chart formats, including line graphs, pie charts, and bar charts.

4.2.2 The system shall visualize trends and correlations among task completion, focus time, and emotional states.

4.2.3 The system shall allow users to view progress and consistency through weekly, monthly, or yearly breakdowns in a calendar-style or timeline layout.

4.3 Notifications & Reflection

4.3.1 The system shall send users personalized summary insights via in-app notifications or email, if enabled.

4.3.2 The system shall provide motivational feedback, encouragement messages, or suggestions for habit improvement based on detected patterns and behavior gaps.

4.3.3 The system shall offer self-assessment prompts to encourage users to reflect on progress, stressors, and achievements.

4.3.4 The system shall allow users to export their reports and visual summaries as downloadable PDF files for personal tracking or external sharing.

5.0 Goal Assistance Module (shared)

5.1 Task Management

5.1.1 The system shall allow users to create, edit, delete, and mark tasks as completed.

5.1.2 The system shall support optional fields in task creation, including description, link, image attachment, and sub-tasks.

5.1.3 The system shall allow users to classify tasks into predefined or user-customized categories.

5.1.4 The system shall support task types: One-time or Repeat (with frequencies: daily, weekly, monthly, yearly).

5.1.5 The system shall allow users to set task priority levels from P1 (highest) to P4 (default, least important).

5.1.6 The system shall allow users to specify task timing: create date, occur date, and due date with preset options (Today, Tomorrow, Weekday, Weekend, Specific Date, or No Due Date).

5.1.7 The system shall allow enabling push notifications as reminders, triggered at the occurrence time or set intervals before (e.g., 10 minutes before).

5.1.8 The system shall support progress tracking based on sub-task completion (displayed as a percentage).

5.1.9 The system shall allow users to sort and filter tasks by category, priority, type, and due date.

5.1.10 The system shall support a calendar view and list view for task visualization.

5.2 Smart Input & Suggestions

5.2.1 The system shall extract task goals from natural language inputs using NLP (e.g., "Remind me to submit a project report tomorrow at 3 PM").

5.2.2 The system shall auto-suggest suitable time slots based on availability, habits, and workload.

5.2.3 The system shall recommend task categories and priority based on task keywords and past behaviors.

5.2.4 The system shall provide predefined templates for recurring task types (e.g., study, exercise, work).

5.3 Smart Advisor

5.3.1 The system shall analyze users' past task completion habits to suggest priority adjustments (e.g., start earlier, complete before usual fatigue hours).

5.3.2 The system shall detect schedule conflicts and suggest alternative time slots dynamically.

5.3.3 The system shall provide emotional and productivity-based reminders, such as taking breaks or relaxation suggestions.

5.3.4 The system shall learn long-term behavior trends to optimize future task planning (e.g., avoid late-night tasks, balance workload).

5.3.5 The system shall allow users to enable/disable Smart Advisor and choose whether to apply, ignore, or manually adjust its suggestions.

3.3.3 Non-Functional Requirements

Product

1.0 Availability

1.1 The system shall maintain 99.95% uptime, minimizing interruptions in daily use.

1.2 Scheduled maintenance shall only occur during non-peak hours and not exceed 2 hours per month.

1.3 In the event of unexpected Firebase service disruption, the app shall attempt automatic reconnection and restore functionality within 5 minutes.

2.0 Functional

2.1 The system shall ensure full feature accessibility and consistency across Android and iOS platforms.

2.2 All core modules (Goals Assistance, Emotion Diary, Focus Timer, Reports, and User Account) must function reliably on smartphones.

2.3 The system shall utilize Firebase services (e.g., Firestore, Authentication, Cloud Functions) for backend functionalities.

3.0 Usable

3.1 The app shall feature an intuitive and beginner-friendly UI, allowing users to navigate core features (e.g., setting goals, starting timers, writing emotion logs) without external help.

3.2 All UI components shall follow consistent design patterns as defined in the Figma design system.

3.3 The system shall support theme personalization for better user comfort and accessibility.

4.0 Reliable

4.1 The app shall support up to 10,000 concurrent users without noticeable performance issues using Firebase backend scalability.

4.2 The system shall prevent data loss by utilizing Firebase offline persistence and real-time synchronization to ensure local changes are synced when connectivity resumes.

4.3 Emotion logs, goal history, and focus sessions shall be backed up in real-time to cloud storage.

5.0 Flexible

5.1 The system shall support localization, including multi-language support, adjustable time/date formats, and culturally adapted content presentation.

5.2 The app shall allow easy updates of localized text through Firebase Remote Config or dedicated translation files.

Organization

6. The system shall be developed using the Dart programming language with the Flutter framework.
7. The system shall use Firebase as the primary backend solution, including Firestore (database), Firebase Auth, and Firebase Cloud Functions.
8. Version control shall be managed via Git and hosted on GitHub, with team collaboration and issue tracking enabled.
9. The UI/UX design shall be created and iterated using Figma.
10. AI-powered features (e.g., emotion analysis, goal suggestion) shall be implemented using Scikit-learn and Pandas via backend Python microservices.

External

4. The app shall integrate Google or Apple Sign-In for secure user authentication.
5. If in-app purchases or donations are introduced, payment processing shall be handled via Google Play and Apple App Store's native payment APIs.
6. All user data handling must comply with relevant privacy laws (e.g., GDPR, CCPA) and Google/Firebase security best practices.

3.4 Development Environment

3.4.1 Hardware and Device Specification

Development Machine

The development tasks, including frontend (Flutter), backend (Python microservices), AI feature development, and UI/UX design, are performed on a high-spec gaming laptop. This device is chosen for its robust CPU and GPU performance, which is essential for compiling, running emulators, debugging complex logic, and training lightweight machine learning models. It also supports virtual device creation for multi-device simulation and efficient parallel processing.

Aspect	Specification
Device Model	ROG Strix G614JV_G614JV
Processor	13th Gen Intel(R) Core(TM) i7-13650HX, 2.60 GHz (14-core CPU)
RAM	32 GB
Storage	1 TB
Graphics	(GPU 0): Intel(R) UHD Graphics (GPU 1): NVIDIA GeForce RTX 4060 Laptop GPU
Operating System	Windows 11 Home (64-bit)

Table 3.2.1: Specification of Development Machine

This machine is particularly effective for handling the resource-intensive tasks involved in cross-platform mobile app development, such as rendering complex animations in Flutter or executing AI model inference using TensorFlow or PyTorch in Python. It ensures development remains efficient even under multitasking or heavy computational workloads.

Testing Machine

To validate app functionality under real-world conditions, a physical Android device is used for testing. This enables comprehensive evaluation of performance, UI responsiveness, compatibility with device hardware (e.g., camera, NFC), and actual user experience. Real-device testing is crucial to ensuring the BetterU app meets quality expectations, especially in areas such as focus timer alerts, emotion input journaling via touch interface, and NFC-enabled interactions.

Aspect	Specification

Device Model	POCO F5
Processor	Qualcomm Snapdragon 7+ Gen 2
RAM	12 GB
Storage	256 GB
Operating System	Xiaomi HyperOS 1.0.18.0.UMRMIXM (Android 14)
NFC Support	Yes

Table 3.2.2: Specification of Testing Machine

This testing phone supports all critical mobile hardware features such as high-refresh-rate display (for UI smoothness), NFC (for smart tag interactions), and advanced multitasking, making it ideal for real-environment testing during the development lifecycle.

Virtual Machine

When there is no real physical mobile device for testing, the Android Studio can still offer a Virtual Studio Emulator for creating an Android virtual machine. Thus, all the Flutter code can be directly tested within the virtual machine without physical access to any smartphone. It is especially useful for quick debugging and UI adjustments when providing the flexibility when real hardware is not available.

Aspect	**Specification**
Device Name	Medium Phone
API Level	35
Resolution (px)	1080 x 2400
Density	420 dpi
ABI List	x86_64
Size on Disk	6.4 GB

Table 3.2.3: Specification of Virtual Machine

NFC Tags

During the development of NFC functionalities in BetterU mobile application, NFC tags are also required for acting as a medium to store the data such as users' task for clocking in and out. Thus, the app functions can be directly implemented on the existing physical NFC tags such as NFC-based task tracking features. These NFC tags will have a wide compatibility with the Android devices and BetterU mobile application to detect and update task progress.

Aspect	Specification
Tag Type	NTAG213
Frequency	13.56 MHz
Memory	144 bytes of usable NDEF memory
Features	Read and write capability with supported password encryption
Form Factor	Adhesive sticker tags
Compatibility	Fully compatible with Android devices that support NFC.

Table 3.2.4: Specification of NFC Tags

3.4.2 Software Tools and Platforms

Android Studio

Android Studio acts as the primary integrated development environment (IDE) for building and testing the BetterU mobile application. The main strength of this IDE is it can provide a robust environment with tools such as Android Emulator, layout editor and APK analyzer. (Android Developers, n.d.) It also supports different versions of APIs with different Android devices with various layouts. Plus, it has high compatibility and performance when perfectly integrating with Flutter plugins. All the Flutter code can be directly compiled and running on its virtual Android devices for displaying the smooth user interface.

Aspect	Specification
Version Used	Android Studio Meerkat Feature Drop | 2024.3.2 Patch 1
System Requirement	● Operating System: Latest 64-bit version of Windows ● RAM: 8 GB ● CPU: Virtualization support Required (Intel VT-x or AMD-V) ● Disk space: 32 GB
Installed Plugins	● Dart (version 243.27824.5) ● Flutter (version 86.0.1) ● Git and GitHub

Table 3.2.5: Software Tools Specification of Android Studio

Visual Studio Code

Visual Studio Code is also utilized alongside Android Studio for backend development using Python and FastAPI. It provides a lightweight performance and extensive extension marketplace. In BetterU development, it mainly focuses on developing the Python backend code, AI features and integration with Flutter code using FastAPI. On the other hand, it also supports syntax highlighting, code linting, Git integration and terminal access in a unified interface. (Microsoft, n.d.) So, all the operations needed on different platforms can be easily streamlined within one application.

Aspect	**Specification**
Version Used	1.102
System Requirement	● Processor: 1.6 GHz ● RAM: 1 GB ● Operating System: Windows 10 and 11 (64-bit) ● Disk: 500 MB minimum
Key Features	● Python extension support ● Git integration ● Integrated terminal
Use Case	● Backend API development ● Python environment management

Table 3.2.6: Software Tools Specification of Visual Studio Code

Github Desktop

Github Desktop is one of the most popular and important tools when it comes to the code collaboration within a development team in BetterU mobile application. It helps developers to easily manage the version control process when providing the graphical user interface for Git operations. For example, commit, merge, push and pull requests. It enables developers to keep track of the code progress from time to time. When there is any update from any team member, all other team members can directly synchronize their local code with the updated code in the GitHub repository. Meanwhile, it also acts as a free cloud-based code backup for BetterU mobile application developers. Thus, all the code can still be retrieved properly even when there are unexpected issues happening on any team members' development machine. (GitHub, n.d.)

Aspect	**Specification**

Version Used	3.5.2
System Requirement	● OS: Windows 10 or 11 ● RAM: 2 GB minimum ● Disk: 200 MB
Key Features	● Visual Git interface ● Branch comparison and merging ● Integrated GitHub sync
Use Case	● Version control ● Issue tracking ● Collaborative codebase management

Table 3.2.7: Software Tools Specification of Github Desktop

3.4.3 Programming Languages and Frameworks

Dart with Flutter

During the development of BetterU mobile application, the Dart programming language and Flutter framework were primarily used for the front-end mobile application development. Dart is a general-purpose language developed by Google which is optimized for building the user interfaces on various platforms such as Android and Windows. It is well-suited for Flutter due to its just-in-time (JIT) and ahead-of-time (AOT) compilation. This can effectively improve both development speed and runtime performance. On the other hand, the Flutter framework which is also developed by Google which supports flexible UI designs and ensures the compatibility across Android and Windows platforms with a single codebase. (Flutter, 2024)

Python with FastAPI

On the server side, the BetterU mobile application will use Python as the backend programming language. This is because it provides a high simplicity, wide community support and high availability of machine learning and natural language processing (NLP) libraries for developers for free. For example, spaCy, scikit-learn and setFit are applied within the BetterU mobile application. Meanwhile, the FastAPI framework will also be used to implement the backend REST API service. It provides a high-performance framework for building the APIs with Python type hints. So, the Flutter code can easily integrate with the Python backend code to perform all the backend services. (Tiangolo, 2024)

Criteria	Dart with Flutter	Python with FastAPI
Primary Use	Front-end mobile app development (UI and UX)	Back-end RESTful API development
Programming Language	Dart	Python
Framework	Flutter	FastAPI
Compilation	JIT for development and AOT for production	Interpreted and supports async execution
Community and Ecosystem	Strong and growing Flutter community	Growing FastAPI ecosystem, large Python community
Platform Support	Cross-platform (Android, iOS, Web and Desktop)	Server-side which is deployable on any OS with Python support

Table 3.2.8: Specification of Dart with Flutter and Python with FastAPI

3.4.4 Database and Storage Services

Google Firebase

Google Firebase serves as the primary cloud-based and non-relational database used in BetterU mobile applications. Meanwhile, it also supports Firebase Authentication, Firestore and Firebase Cloud Messaging for allowing BetterU mobile application to seamlessly store, retrieve and update all the real-time data. Firebase Realtime Database and Cloud Firestore can provide secure storage and retrieval of structured data such as the task records, summaries, NFC tag associations and user settings. Via Google Firebase, the security of the database data can also be ensured via user authentication and cross-device synchronization features provided. (Firebase, n.d.) It also has a high compatibility to fit with the Flutter framework and security features to provide the services for mobile-first development including BetterU mobile application.

Aspect	Specification
Platform Type	Cloud-based Backend-as-a-Service (BaaS)
Key Features	• Firebase Authentication

	• Cloud Firestore • Firebase Cloud Messaging • Firebase Hosting
Plan	Spark Plan (Free tier limits): • 50K document reads per day • 1 GB storage • 10K monthly cloud functions invocations • 5 GB hosting bandwidth
Use Case	Cloud storage User authentication Real-time syncing

Table 3.2.9: Specification of Google Firebase

3.4.5 Application Architecture and Deployment Environment

BetterU mobile application is designed to use a **client-server architecture** where the frontend is developed using Dart with Flutter and the backend is powered by Python with FastAPI. This architecture can help for separating the user interface and the server-side logic. Thus, both layers can be developed and maintained independently.

The **Flutter-based mobile frontend** mainly focuses on delivering interactive and responsive user interfaces across different screen sizes and platforms. In order to communicate with the Python backend code, it will use **RESTful APIs supported by FastAPI** to implement the backend features in BetterU mobile app. On the server side, FastAPI will also handle data processing, user authentication logic and manage the integration with cloud services such as **Google Firebase**.

After the development phase for each increment, the BetterU mobile app will be **deployed and tested on physical Android devices with API level 21 and above**. The device will also support the NFC tag scanning since it is part of the mobile app features. Via this modular deployment approach, all the **modules can be developed one by one via continuous incremental development**. Meanwhile, the mobile app can also be debugged efficiently for future improvements and maintenance.

3.4.6 UI/UX Design Tools

When it comes to user interface design, Figma will be primarily used as the UI/UX design tool for BetterU mobile application before the development stage. It allows the mobile app

designers to create the wireframes, mockups and interactive prototypes. Varieties of reusable and interactive components also help designers to create responsive and user-friendly interfaces. (Figma, 2024) In BetterU mobile application, it is typically useful for providing a seamless sharing, version control and feedback throughout the design process. Thus, all the team members can collaboratively design the user interface of the BetterU mobile app when synchronizing the latest updates simultaneously.

Aspect	**Specification**
Platform Type	Web-based (Cloud) with desktop app support (Windows)
Key Features	● UI/UX design ● Prototyping ● Wireframing ● Collaboration
System Requirement	● Operating System: Windows 8.1 or later ● Graphics: Windows (Nvidia or AMD) ● Minimum Browser Version: ○ Chrome 99 or later ○ Microsoft Edge 121 or later
Pricing Tier Used	Free (Education Plan)

Table 3.2.10: Specification of Figma

3.4.7 External Libraries, APIs and Plugins

Speech Recognition and Transcription

Vosk API and OpenAI Whisper are used for real-time and offline speech-to-text conversion tasks in BetterU mobile application goal assistance module. The Vosk supports a lightweight, on-device speech recognition which is compatible with Android and Python environments. (Vosk, 2024) At the same time, Whisper is also used for generating highly accurate transcripts of longer audio recordings in multiple languages. (OpenAI, 2024) Via the combination of both libraries, the accuracy of the speech transcription can be improved effectively to extract the task input from the users.

Natural Language Processing

In order to extract the task insights, the libraries like spaCy and dateparser will be used in BetterU mobile application. spaCy supports tokenization, entity recognition and dependency parsing. (Explosion AI, 2024) It can help BetterU mobile app to understand and extract the key points from users' task input such as task item. Meanwhile, the dateparser library is also used for extracting and normalizing human-readable dates from free-text inputs. (Scrapinghub, 2024) It acts as an assisting tool for extracting the specific date and time from users' task input.

Machine Learning and Text Classification

Scikit-learn library is used for implementing traditional ML models to detect the task categories in the goal assistance module. (Pedregosa et al., 2011) It can effectively detect and categorize a user's task input without the user manually selecting the task categories. Moreover, the SetFit library is also integrated for few-shot text classification using Sentence Transformers. (Zimmer, 2023) Without requiring large datasets, it can still help BetterU to fine-tune the task classification model to accurately classify the task categories and priority level.

Backend API Interface

FastAPI framework acts as a primary RESTful API backend for the BetterU mobile application. It helps for offering the scalable communication between the mobile frontend and the backend services in the mobile app. Besides, it also offers the automatic data validation, interactive Swagger documentation and asynchronous support for concurrent requests. It is highly suitable for prototyping and deploying the machine learning and NLP models used in BetterU mobile applications since it has a high performance and developer-friendly syntax. (Tiangolo, 2024)

3.5 Chapter Summary and Evaluation

This chapter presented the planning, analysis, and technical preparation stages undertaken for the development of the *BetterU* mobile application. The **Incremental Development Model** was adopted as the software process model for this project. This model was chosen because it aligns well with the nature of *BetterU*, where individual features can be developed, tested, and refined incrementally. This approach supports development flexibility, enables early feedback collection from supervisors, and allows for continuous improvement throughout the development lifecycle.

To ensure the application addresses real user needs, two fact-finding techniques were applied: **observation** and **online research**. The observation method involved monitoring the daily behaviors and habits of people within the developer's surroundings, such as friends and relatives, to gain insights into their challenges in task planning and emotional self-management. In parallel, online research was conducted by reviewing credible digital sources to explore common issues related to personal productivity and emotional well-being. The primary objective of these techniques was to gather relevant information to ensure that *BetterU* is tailored to meet the practical needs of its target users, mainly students and working professionals.

In addition, the chapter outlined the **requirement analysis process**, which included the creation of a use case diagram, detailed use case descriptions, and comprehensive listings of functional and non-functional requirements. These elements serve as a foundation for defining the system's expected behavior and guiding future development in a systematic and traceable manner.

Finally, this chapter described the **development environment and tools** used in the project. The mobile application was developed using the Flutter framework and Dart language for front-end implementation, while the backend was supported by FastAPI. The project also utilized various supportive technologies, including Figma for UI/UX design, Firebase for cloud services, and SQLite for local data storage. Additionally, external libraries and APIs were integrated to enable advanced features such as **speech recognition**, **natural language processing (NLP)**, and **machine learning (ML)**. These tools collectively enhance the app's ability to understand user input and deliver intelligent, context-aware suggestions to support users in achieving personal growth.

Chapter 4

System Design

4 System Design

This chapter presents the detailed system design of the BetterU mobile application, specifying both the structural and behavioral aspects of the system. It outlines various design models and specifications that illustrate how the system operates and interacts with its users, including students, office workers, and administrators. Sequence diagrams, state charts, and activity diagrams will be used to describe the dynamic processes and user interactions. The user interface design will provide an overview of the application's visual layout, while the data design will include class diagrams, entity-relationship diagrams, and a data dictionary to define the logical structure and flow of information. The report design will describe the required components, data elements, and levels of detail for each report, such as the productivity report and the social performance report. In addition, the process design and software architecture design will demonstrate how the system's components are organized and deployed. Finally, the chapter will highlight the AI algorithms applied within the application, supported by sample training datasets, to enable intelligent features such as task categorization and rescheduling prediction.

4.1 Sequence Diagram

Goal Assistance Module (shared)

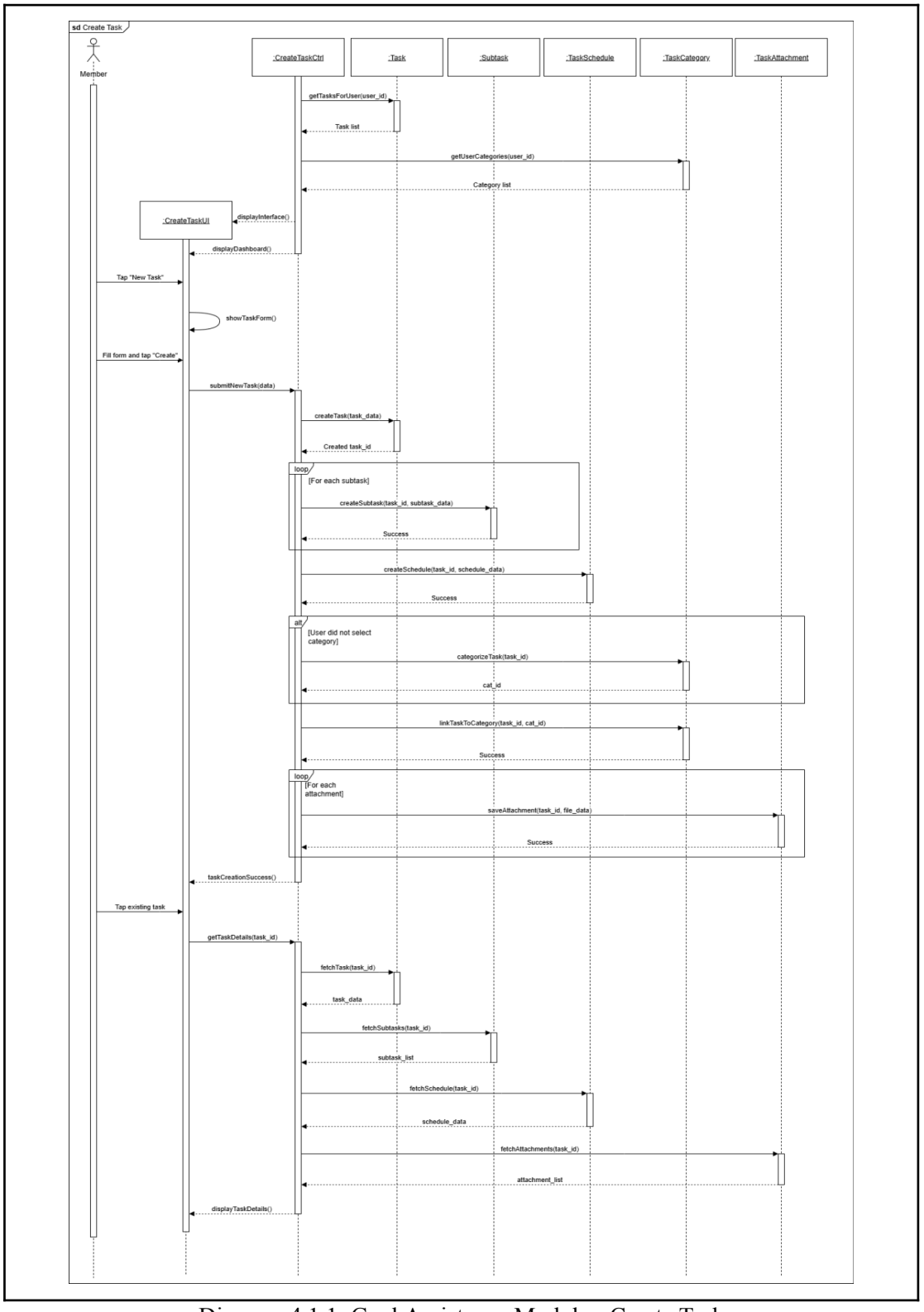


Diagram 4.1.1: Goal Assistance Module - Create Task

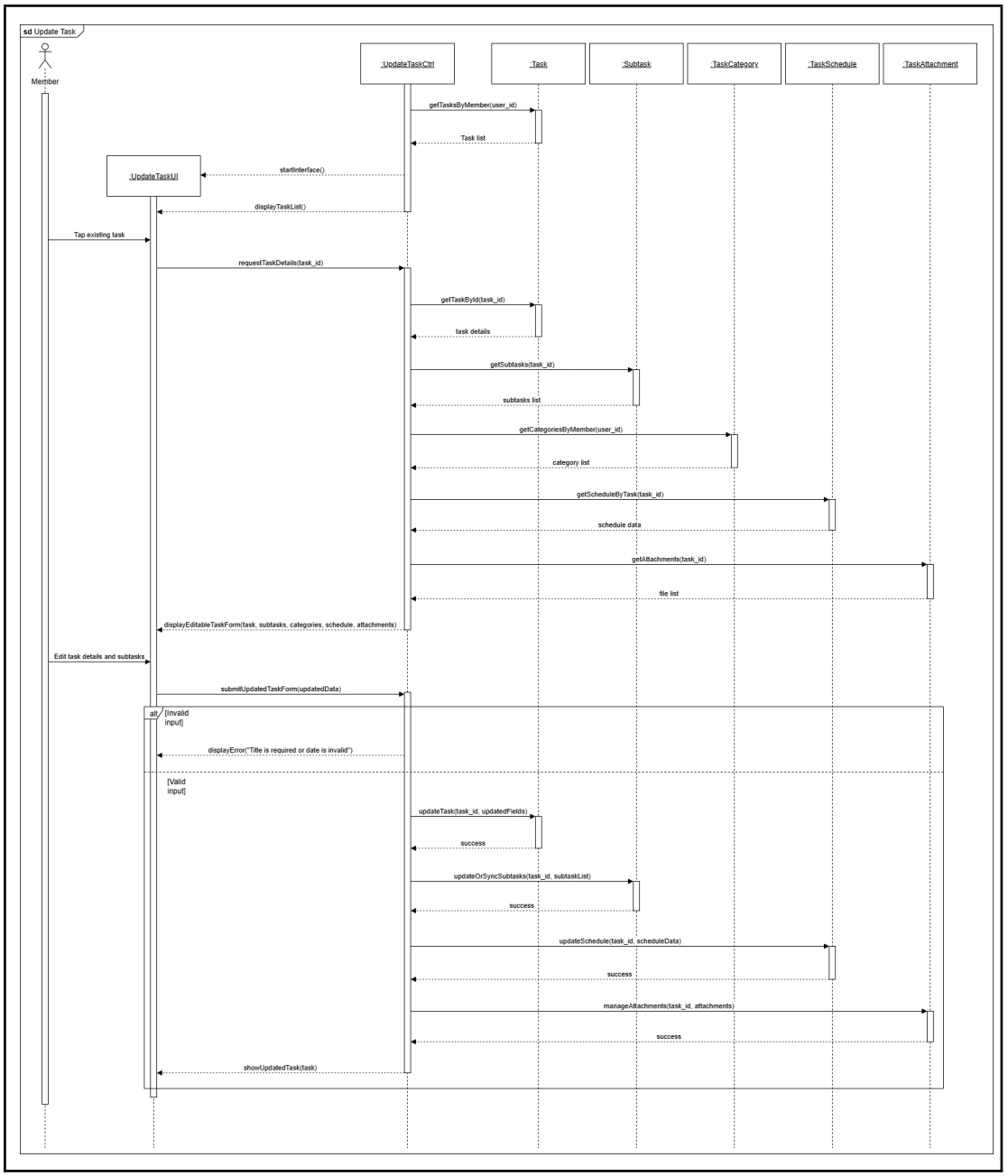


Diagram 4.1.2: Goal Assistance Module - Update Task

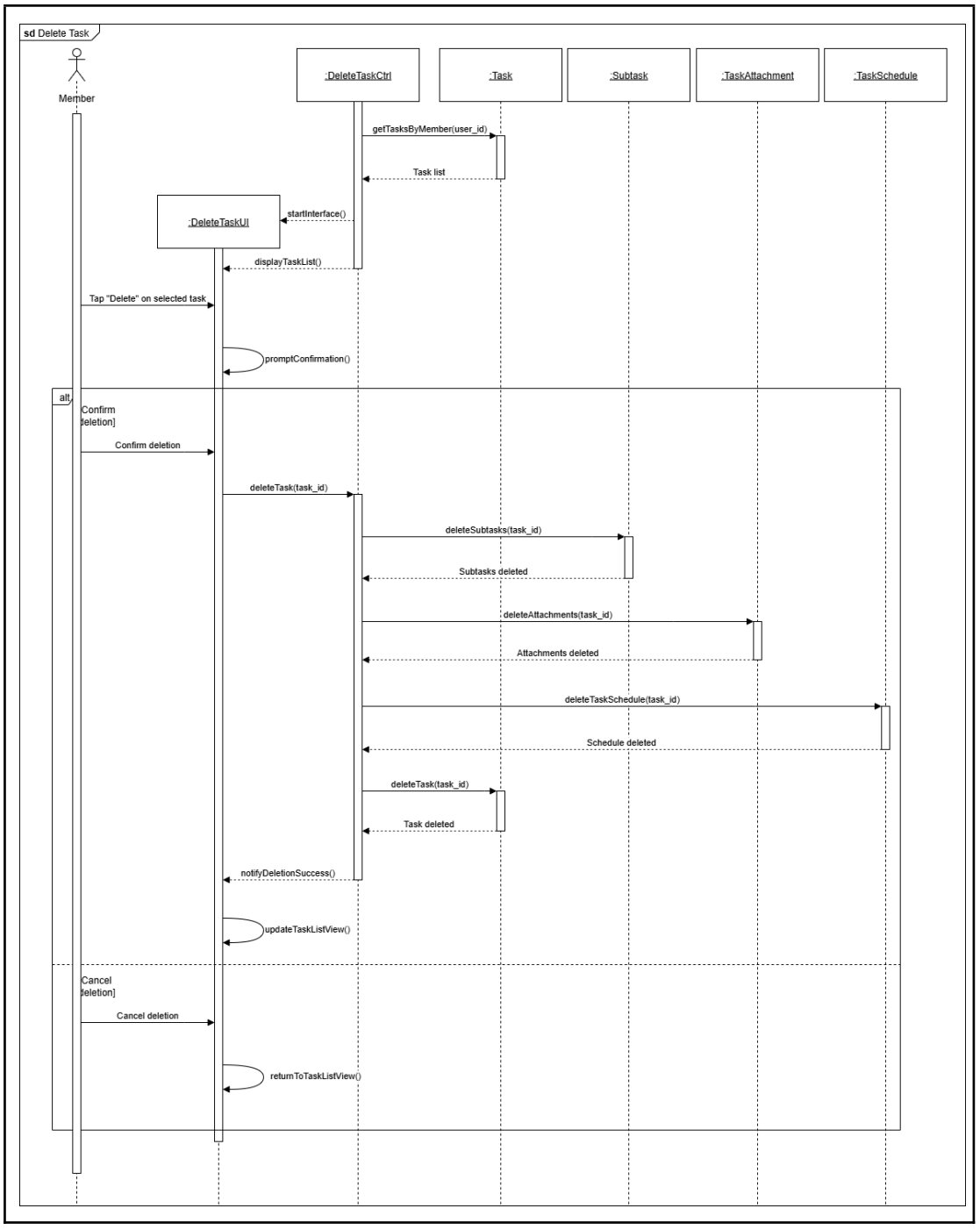


Diagram 4.1.3: Goal Assistance Module - Delete Task

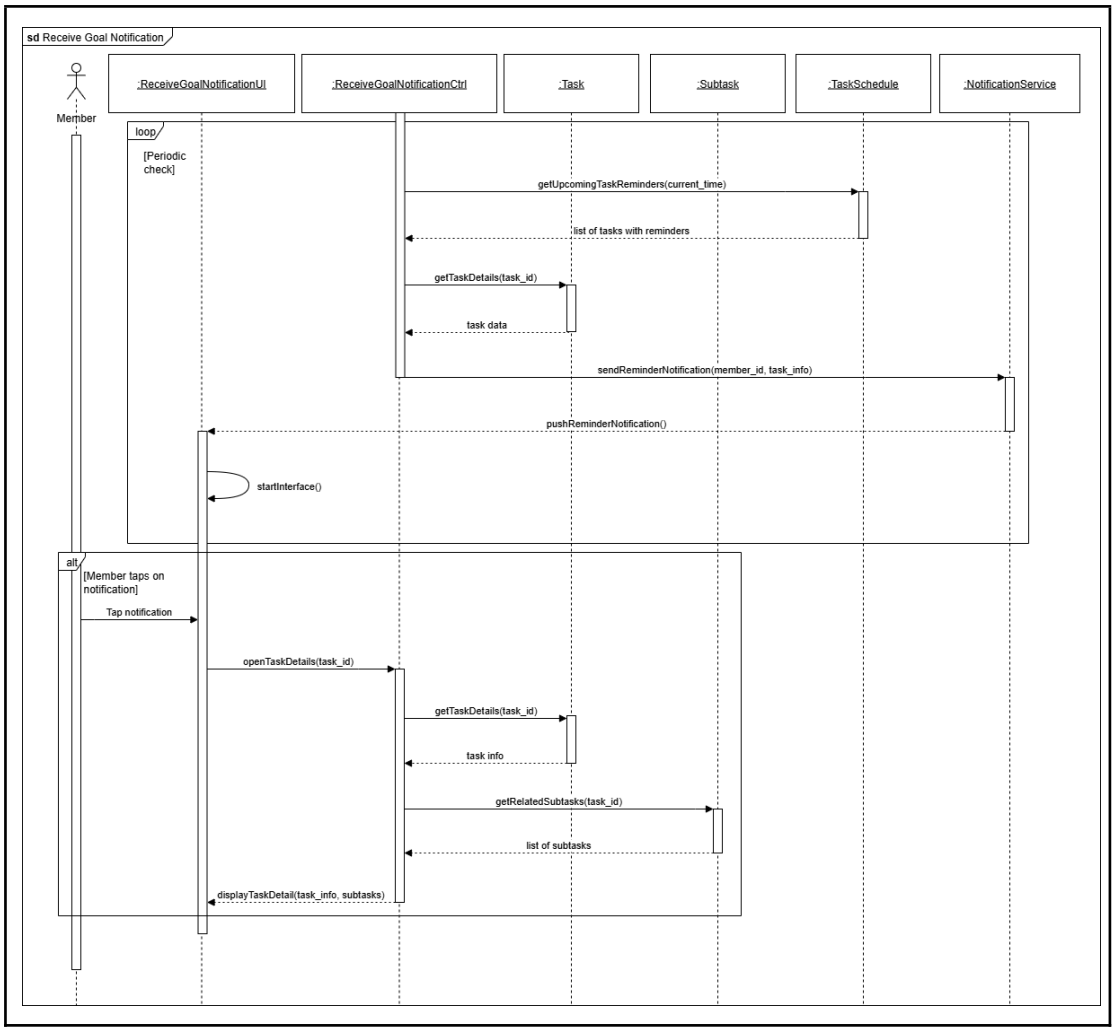


Diagram 4.1.4: Goal Assistance Module - Receive Goal Notification

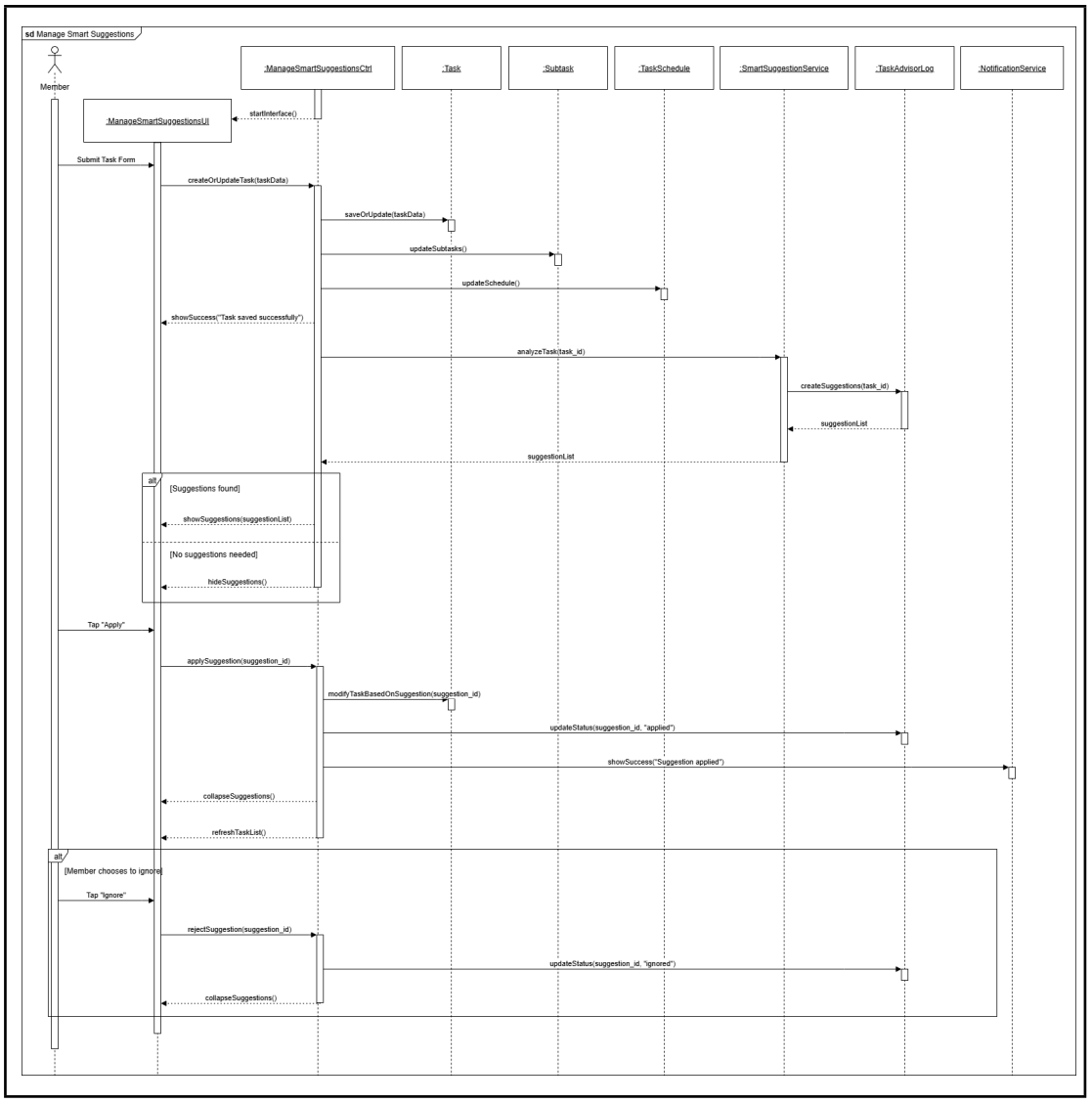


Diagram 4.1.5: Goal Assistance Module - Manage Smart Suggestions

Emotion Diary Note Module

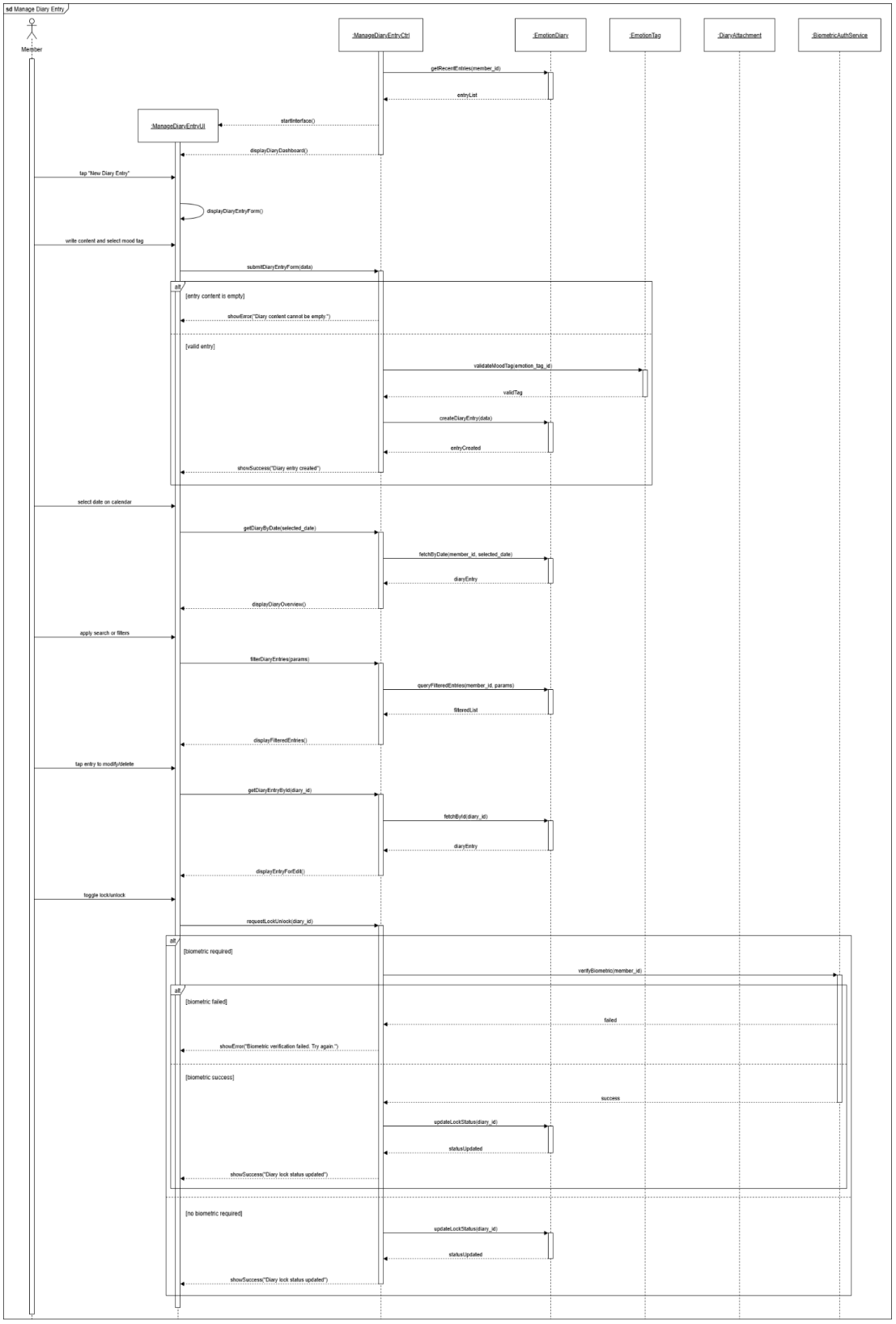


Diagram 4.1.6: Emotion Diary Note Module - Manage Diary Entry

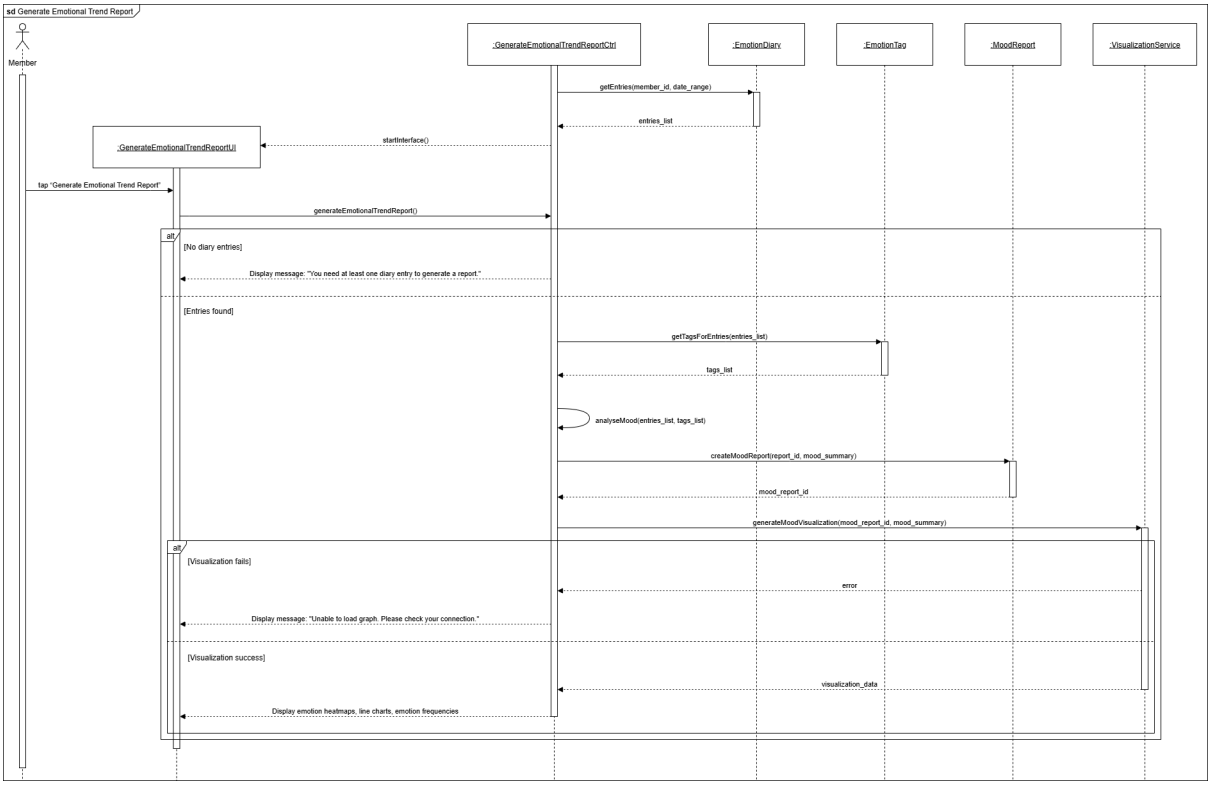


Diagram 4.1.7: Emotion Diary Note Module - Generate Emotional Trend Report

Focus Timer Module

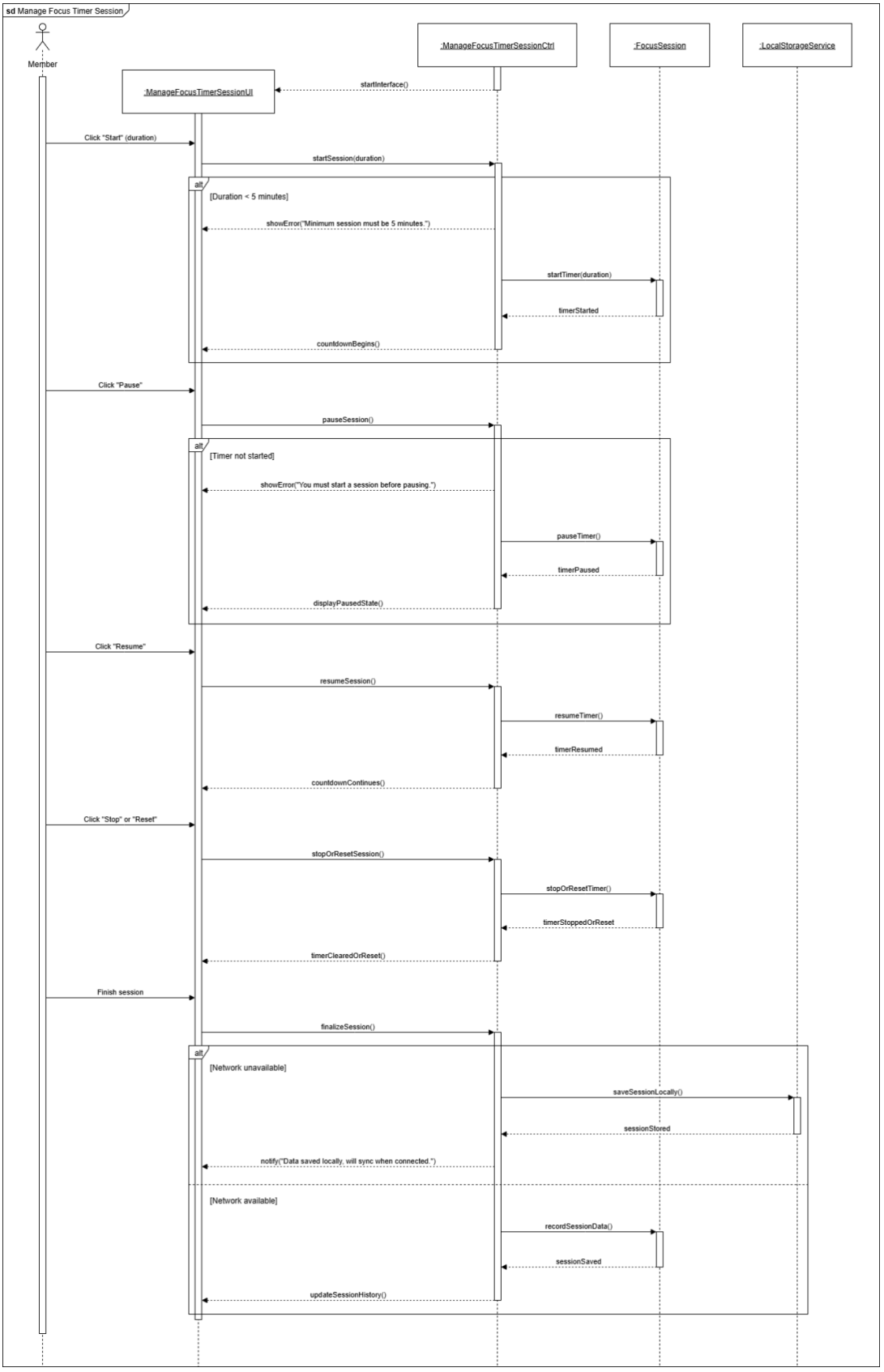


Diagram 4.1.8: Focus Timer Module - Manage Focus Timer Session

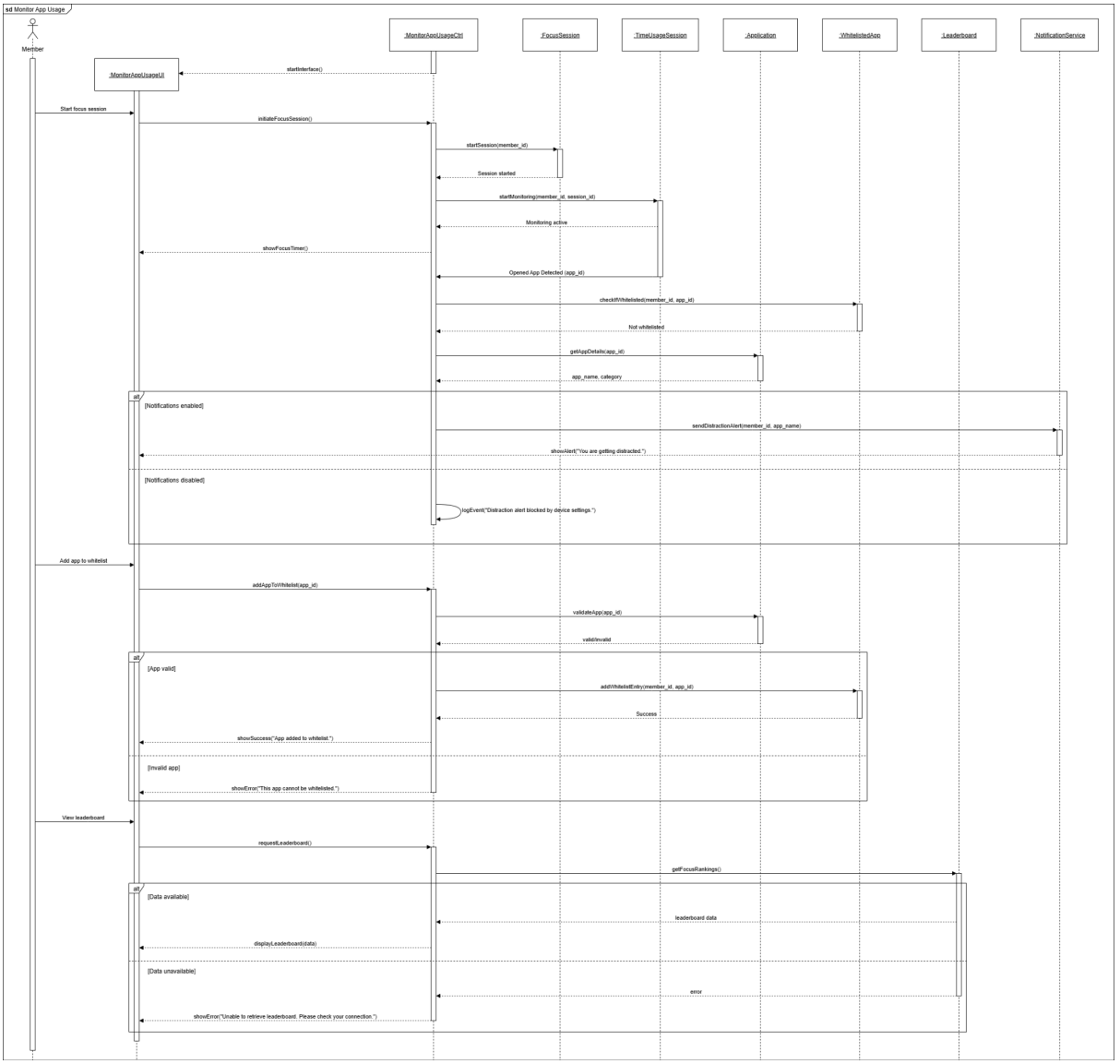


Diagram 4.1.9: Focus Timer Module - Monitor App Usage

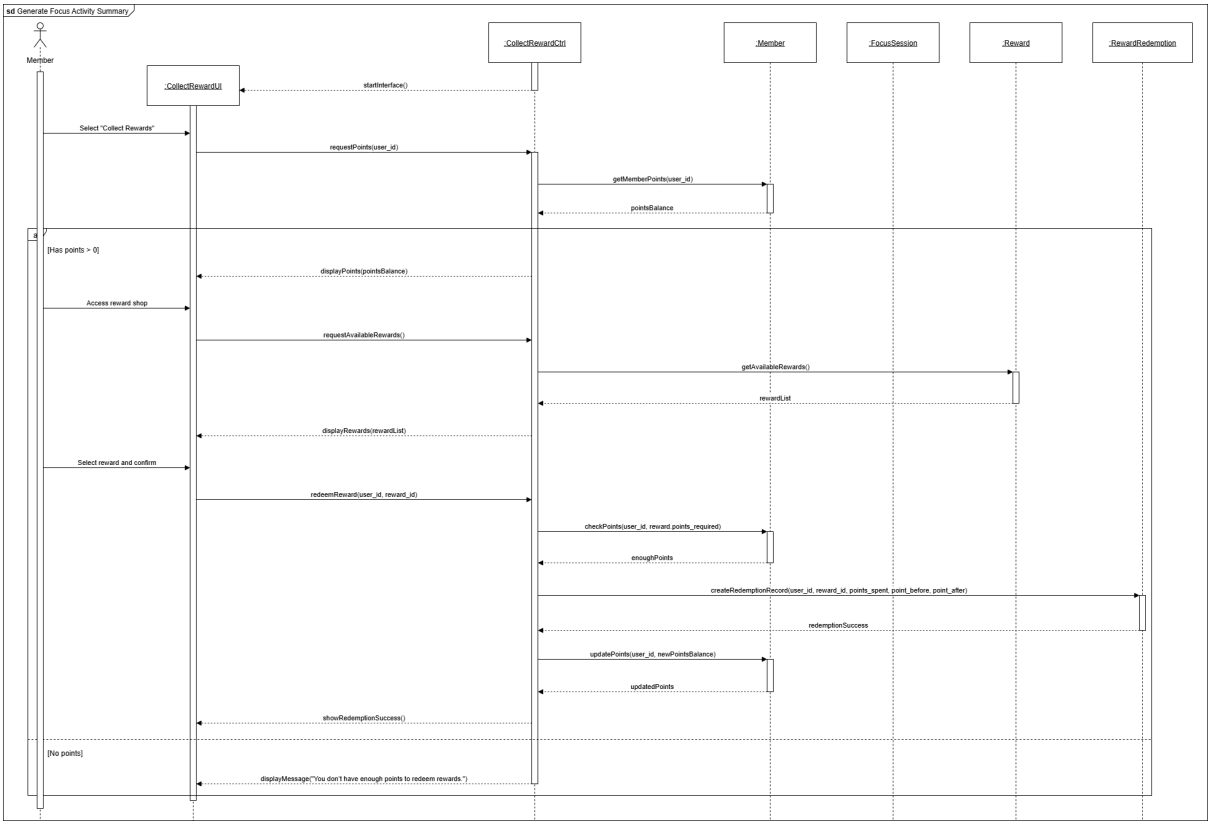


Diagram 4.1.10: Focus Timer Module - Collect Reward

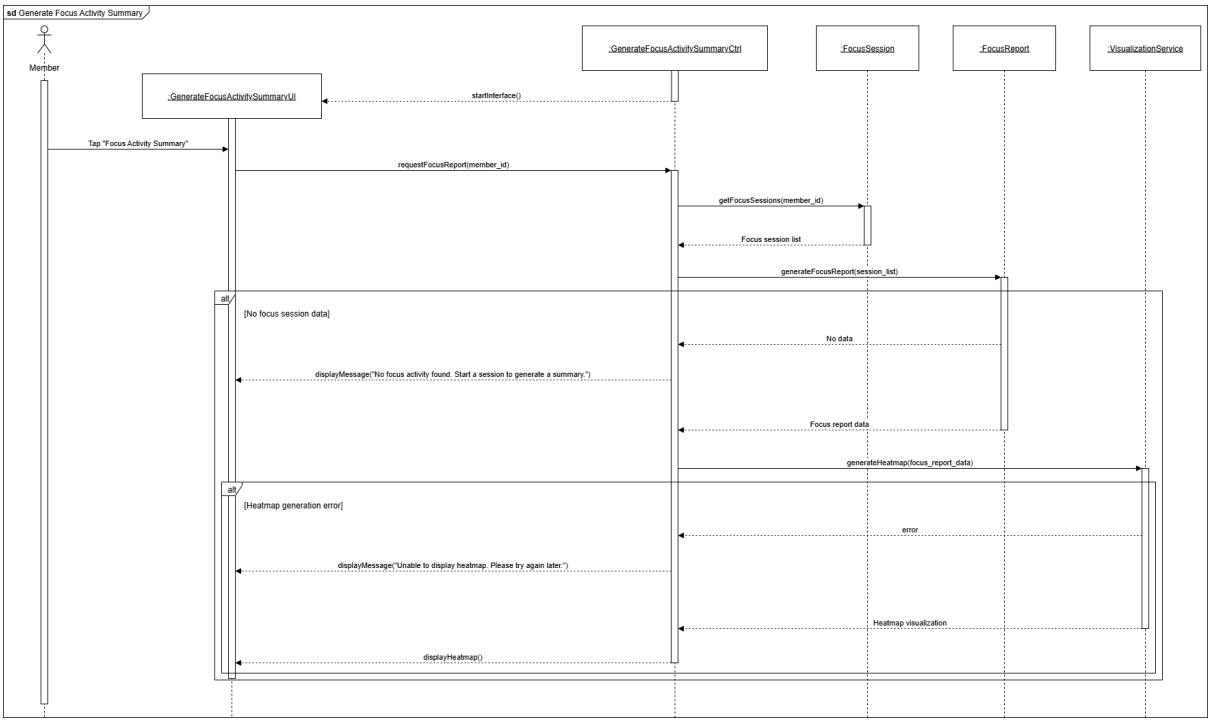


Diagram 4.1.11: Focus Timer Module - Generate Focus Activity Summary

User Module

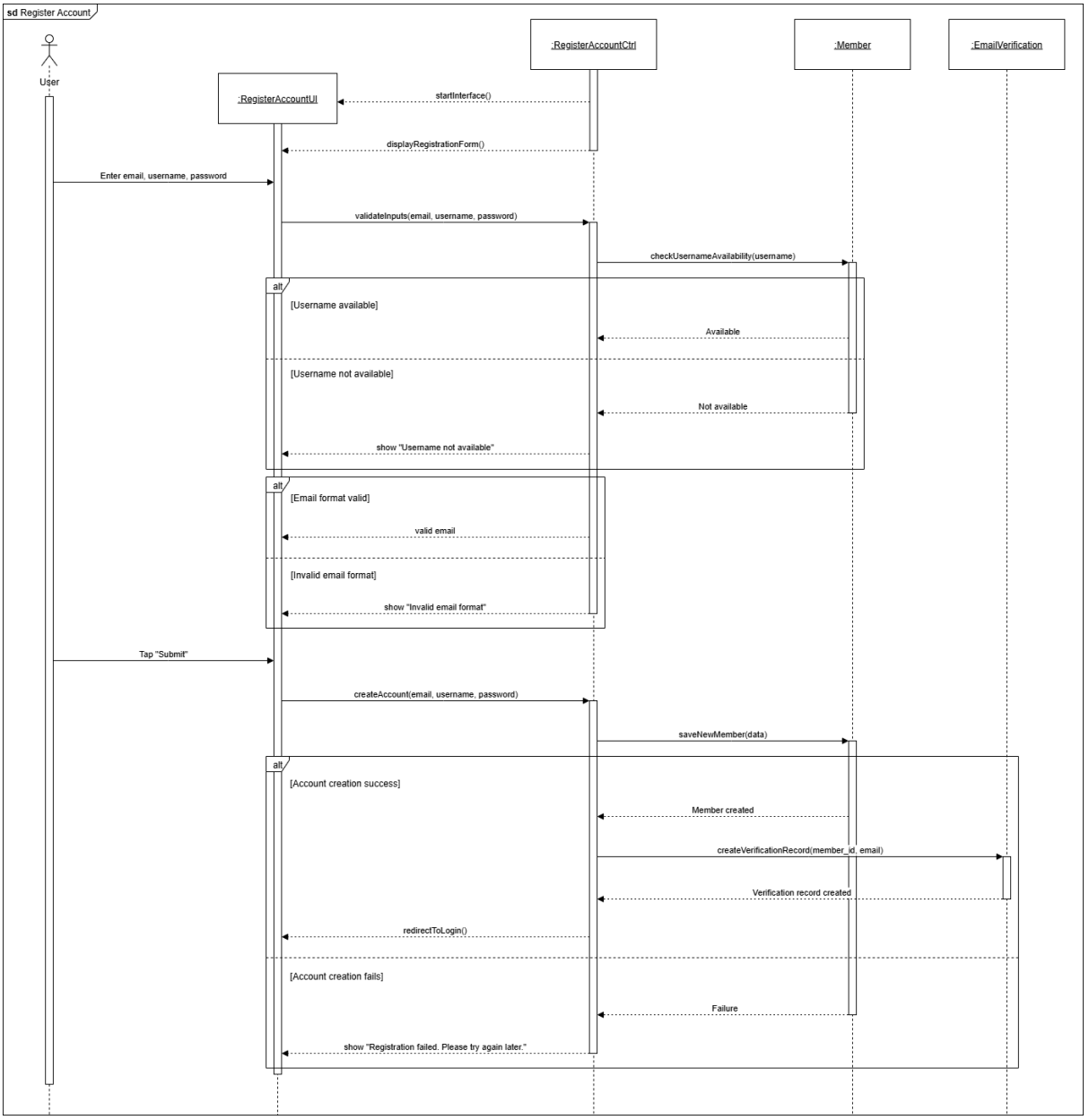


Diagram 4.1.12: User Module - Register Account

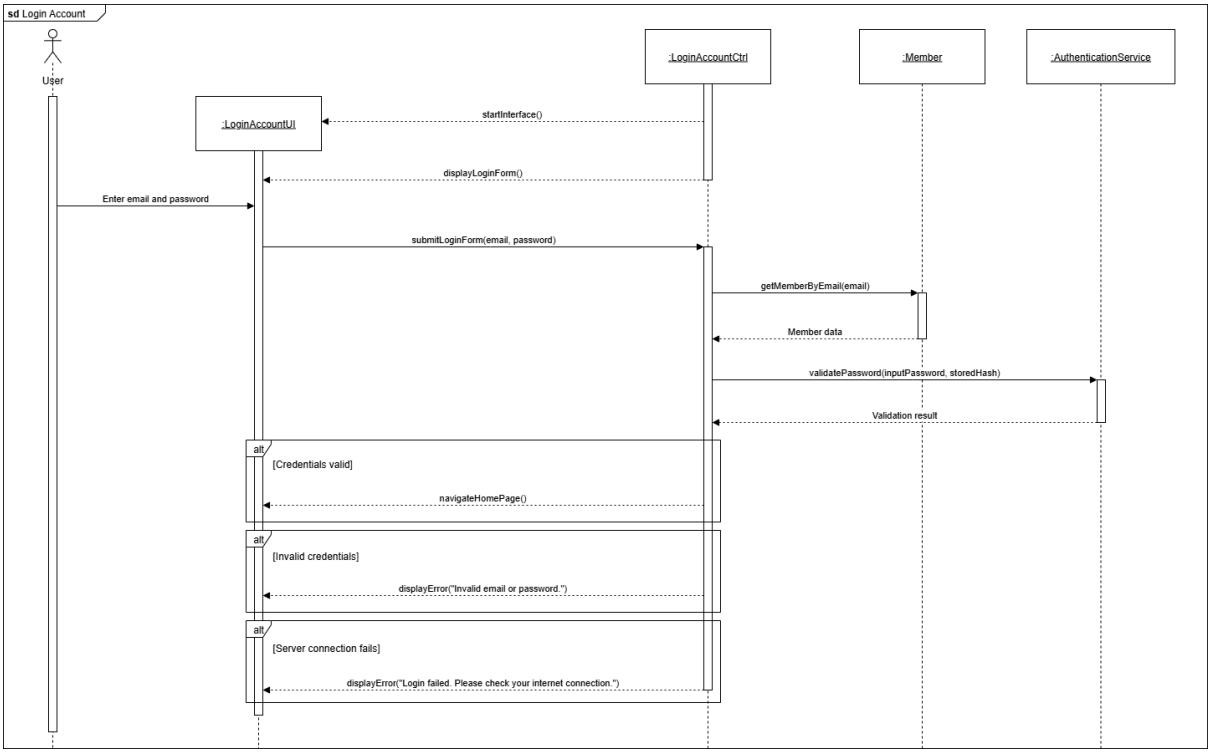


Diagram 4.1.13: User Module - Login Account

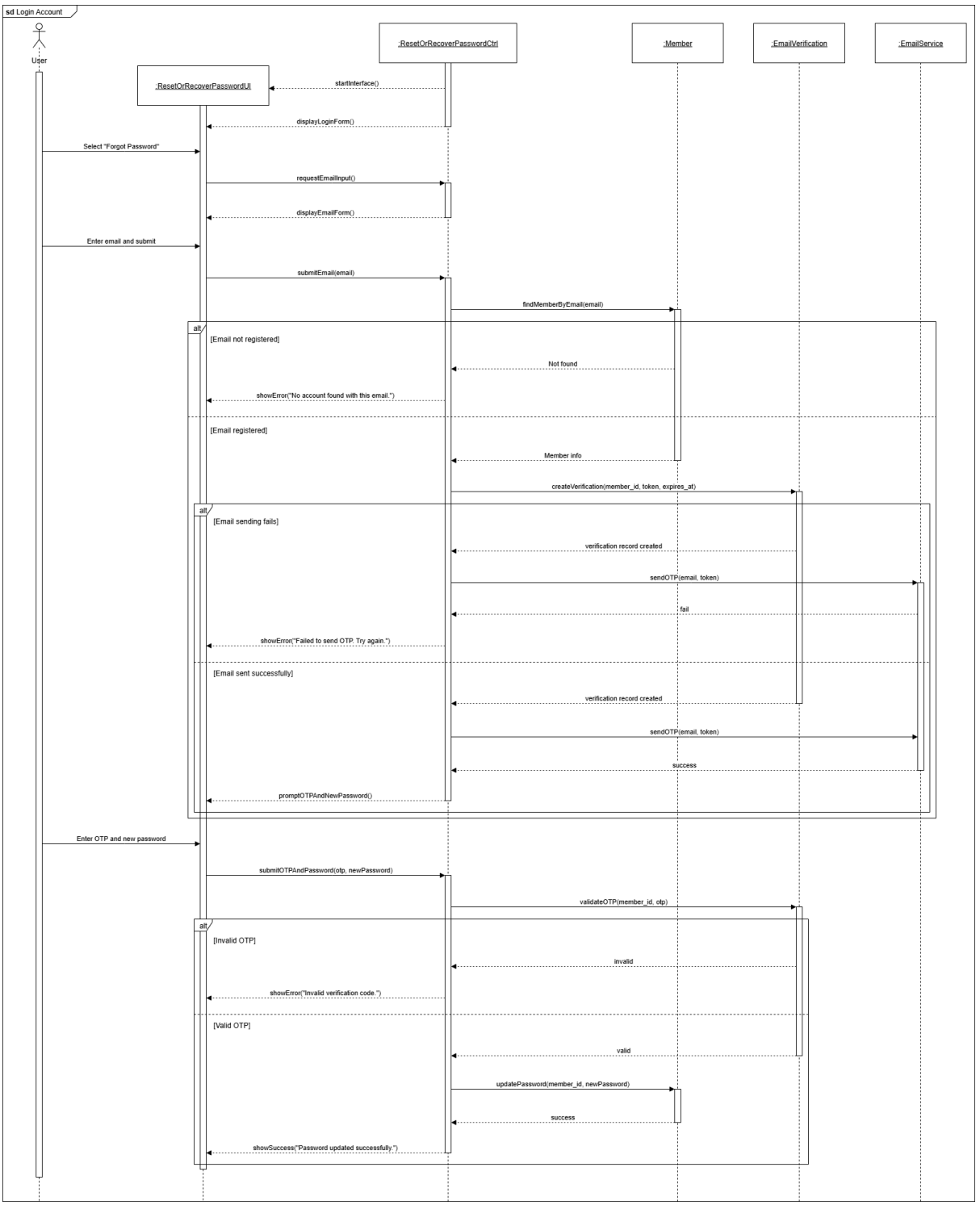


Diagram 4.1.14: User Module - Reset or Recover Password

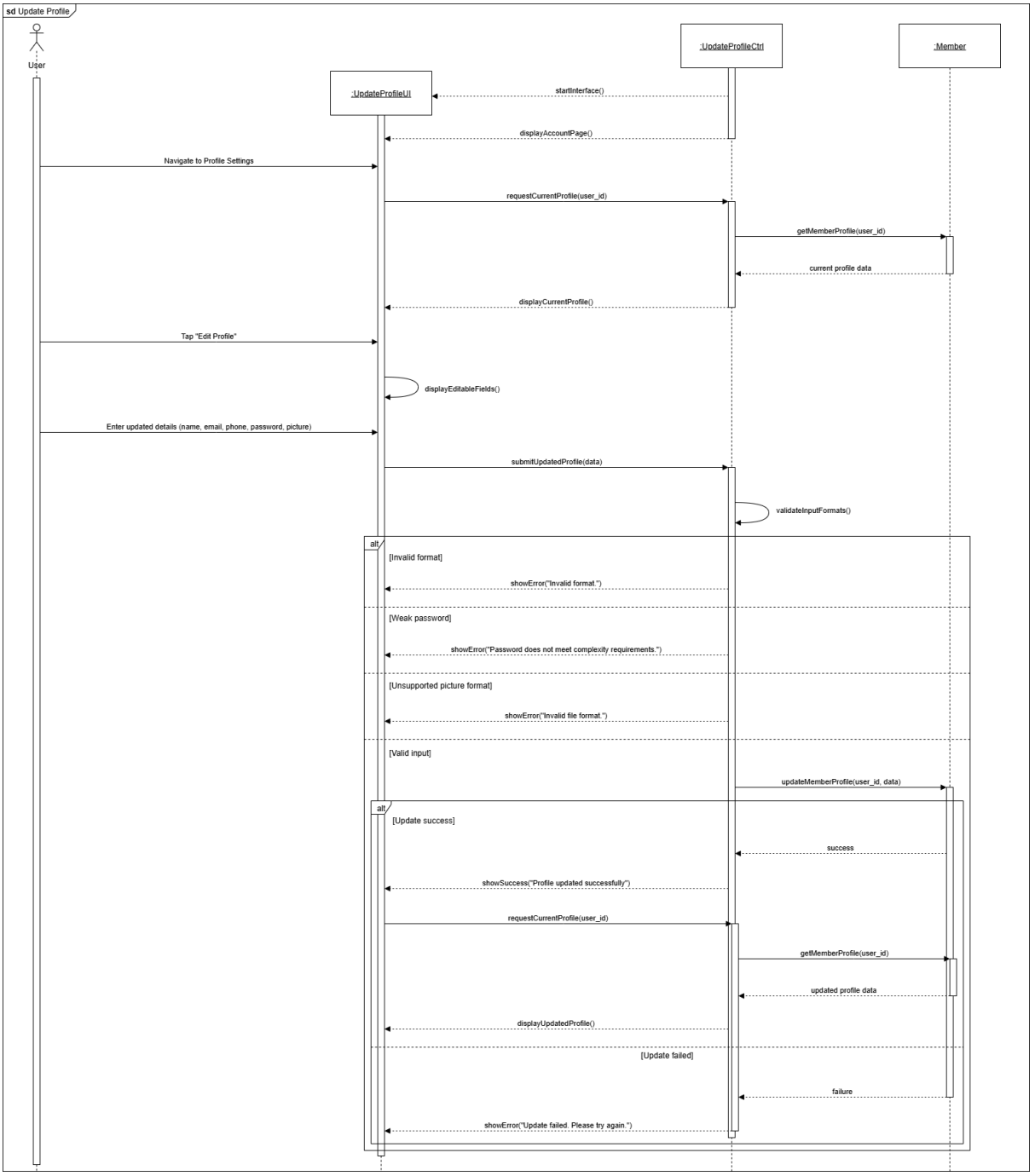


Diagram 4.1.15: User Module - Update Profile

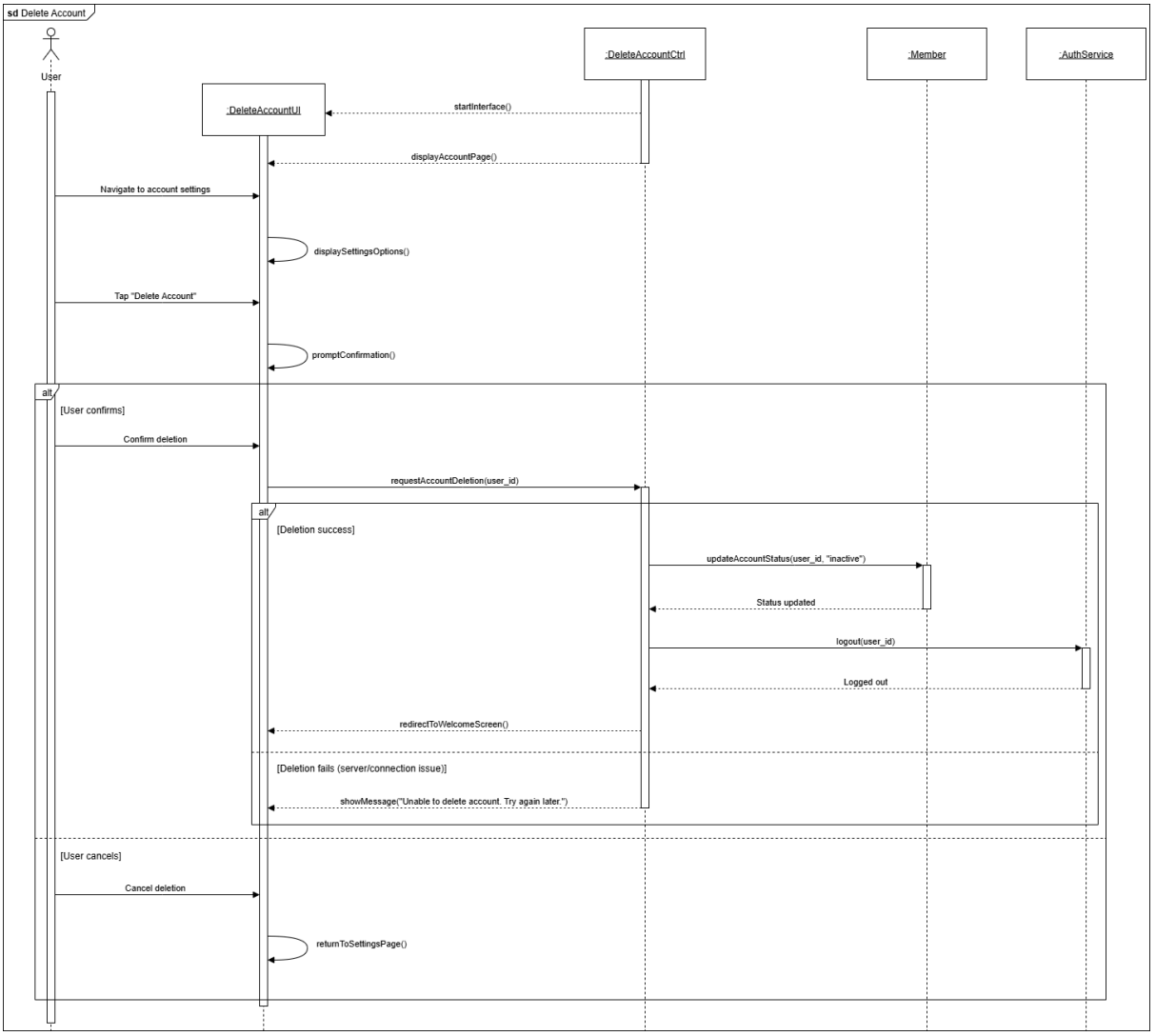


Diagram 4.1.16: User Module - Delete Account

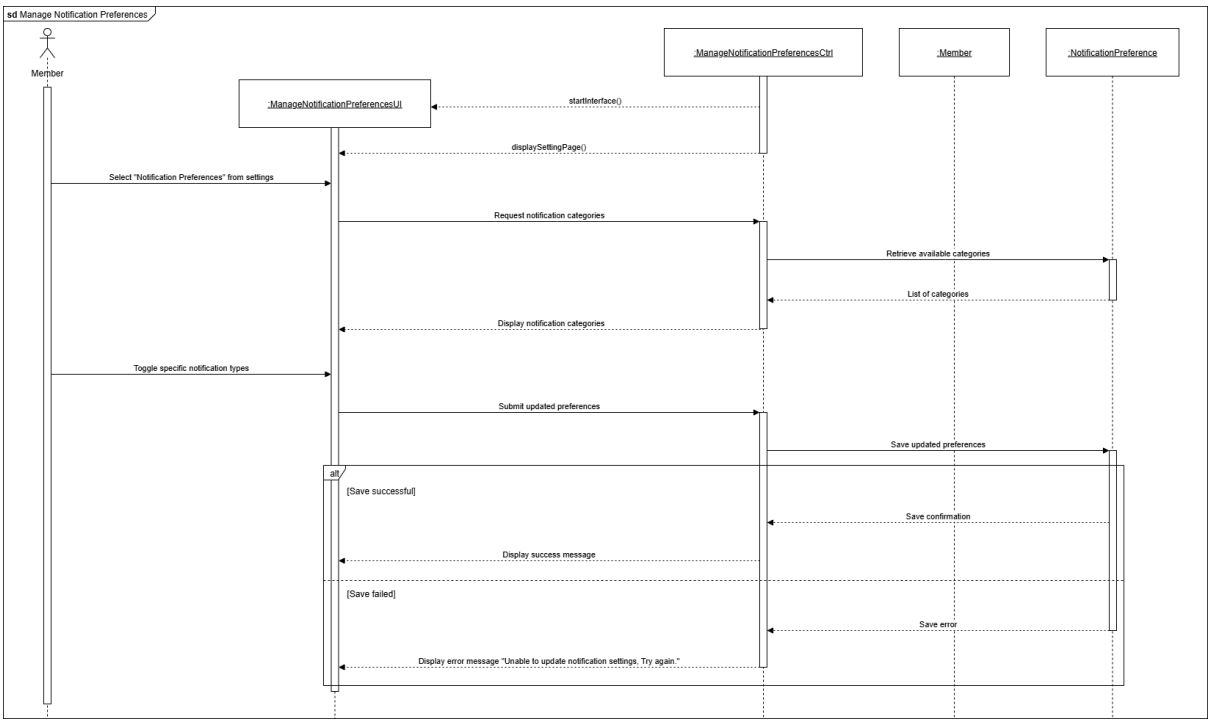


Diagram 4.1.17: User Module - Manage Notification Preferences

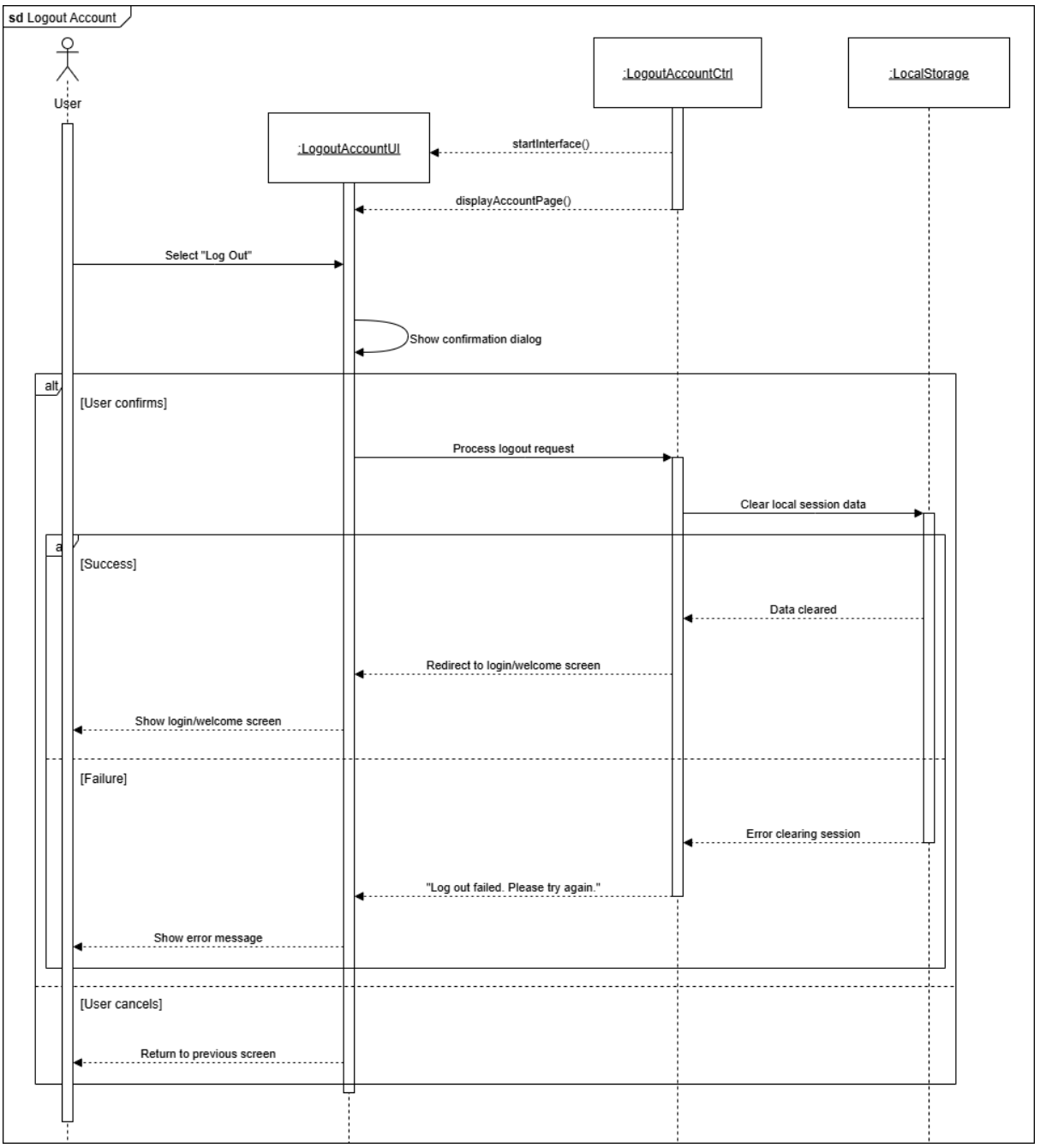


Diagram 4.1.18: User Module - Logout Account

Report Module

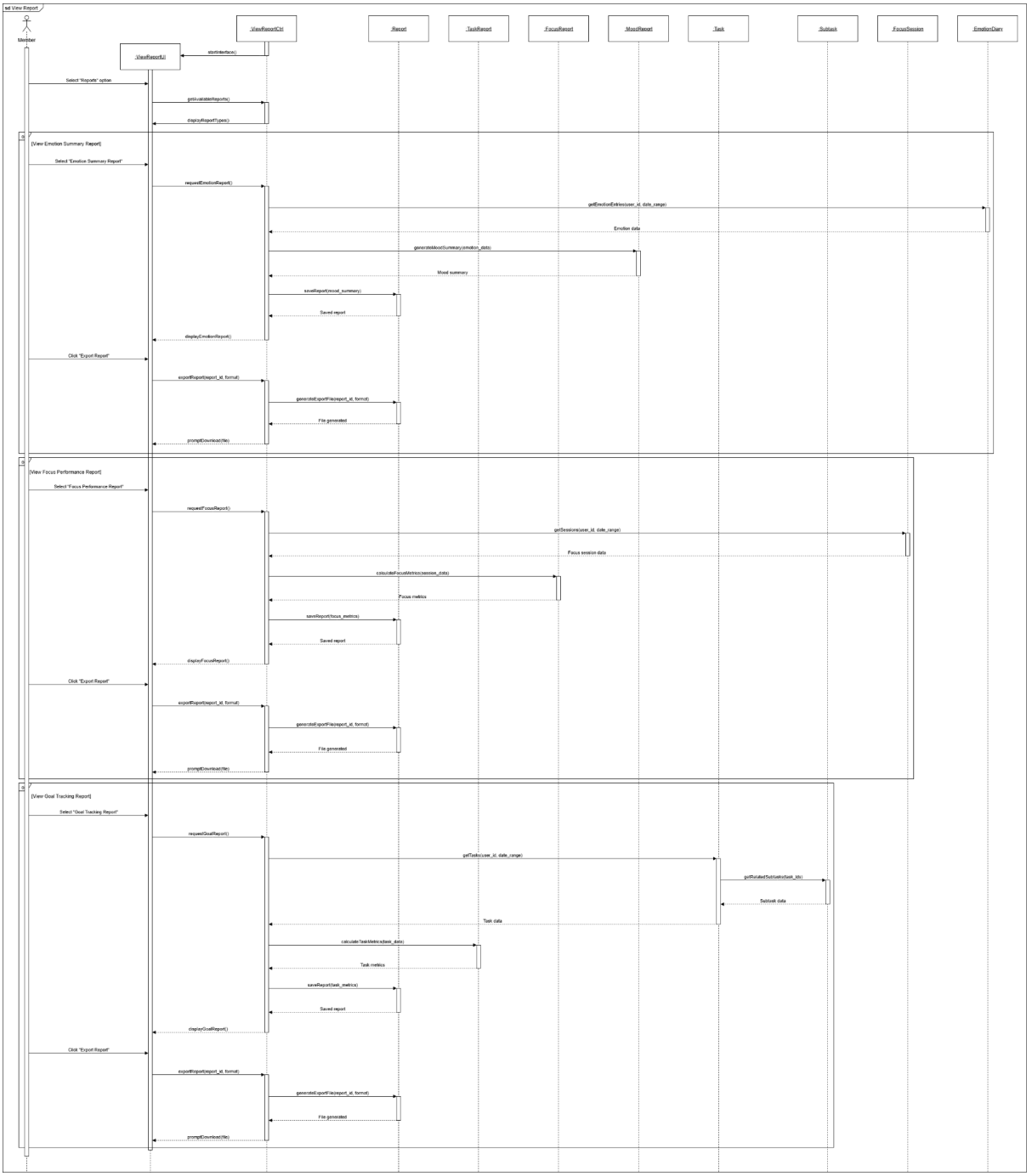


Diagram 4.1.19: Report Module - View Report

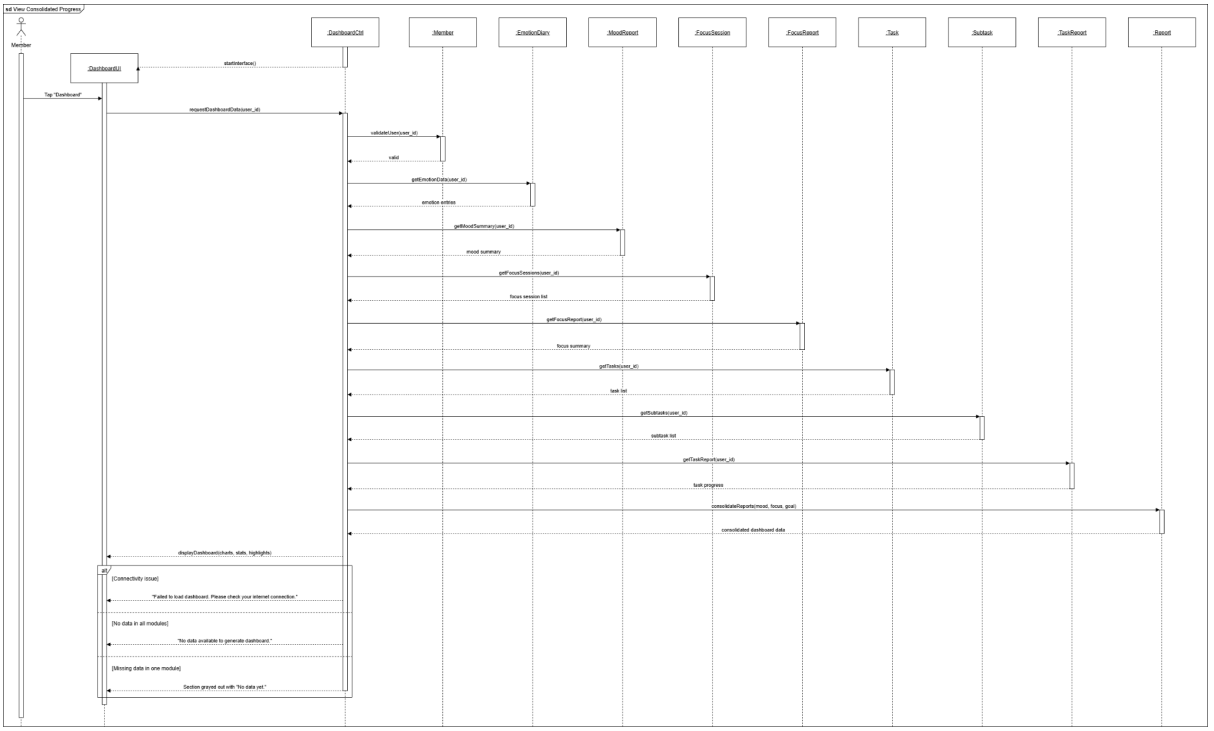


Diagram 4.1.20: Report Module - View Consolidated Progress Dashboard

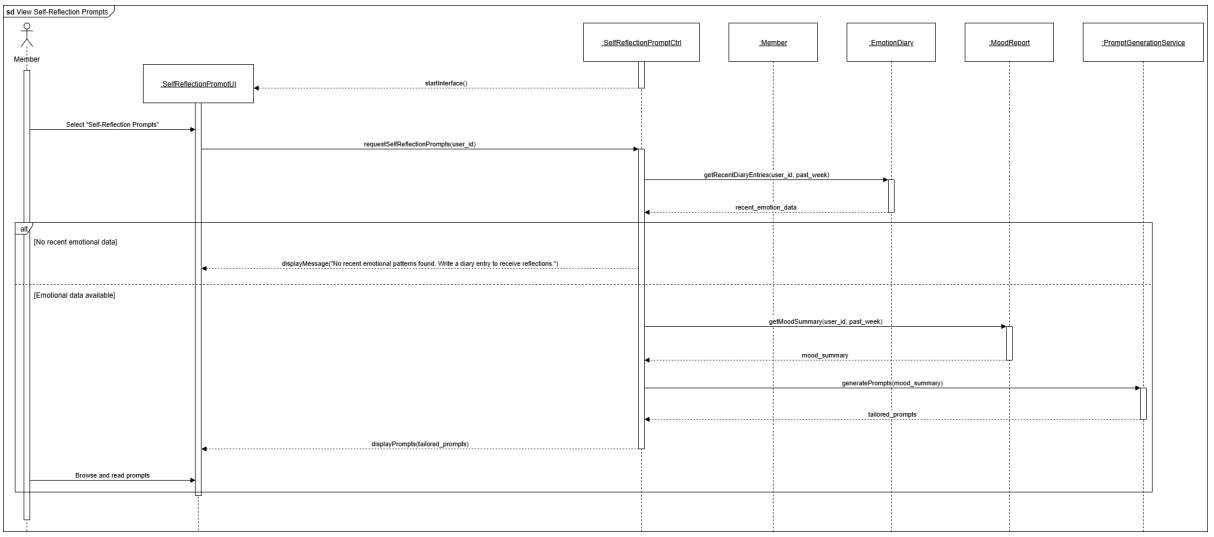


Diagram 4.1.21: Report Module - View Self-Reflection Prompts

4.2 State Chart Diagram

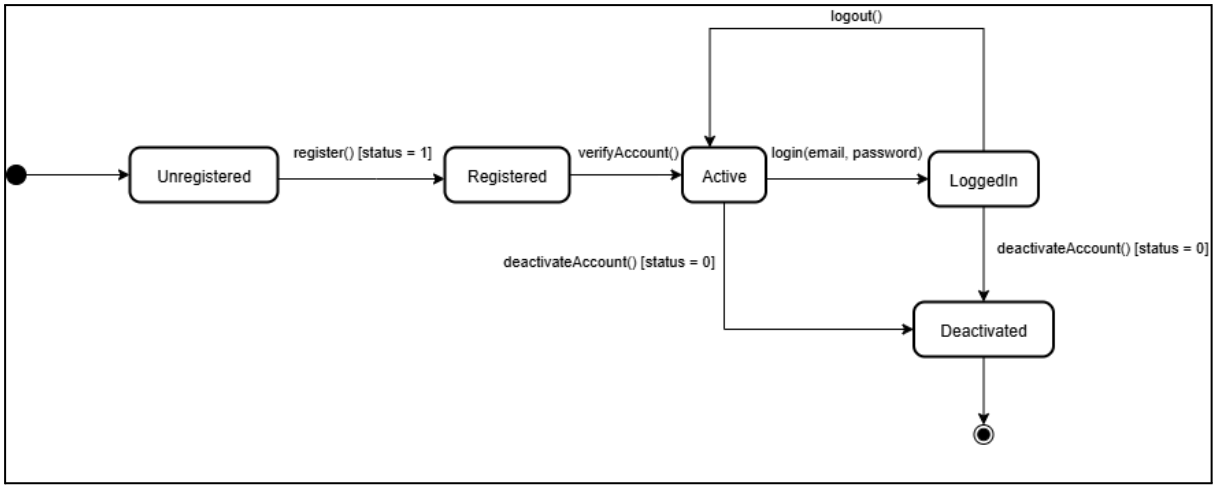


Diagram 4.2.1: State Chart Diagram - Person

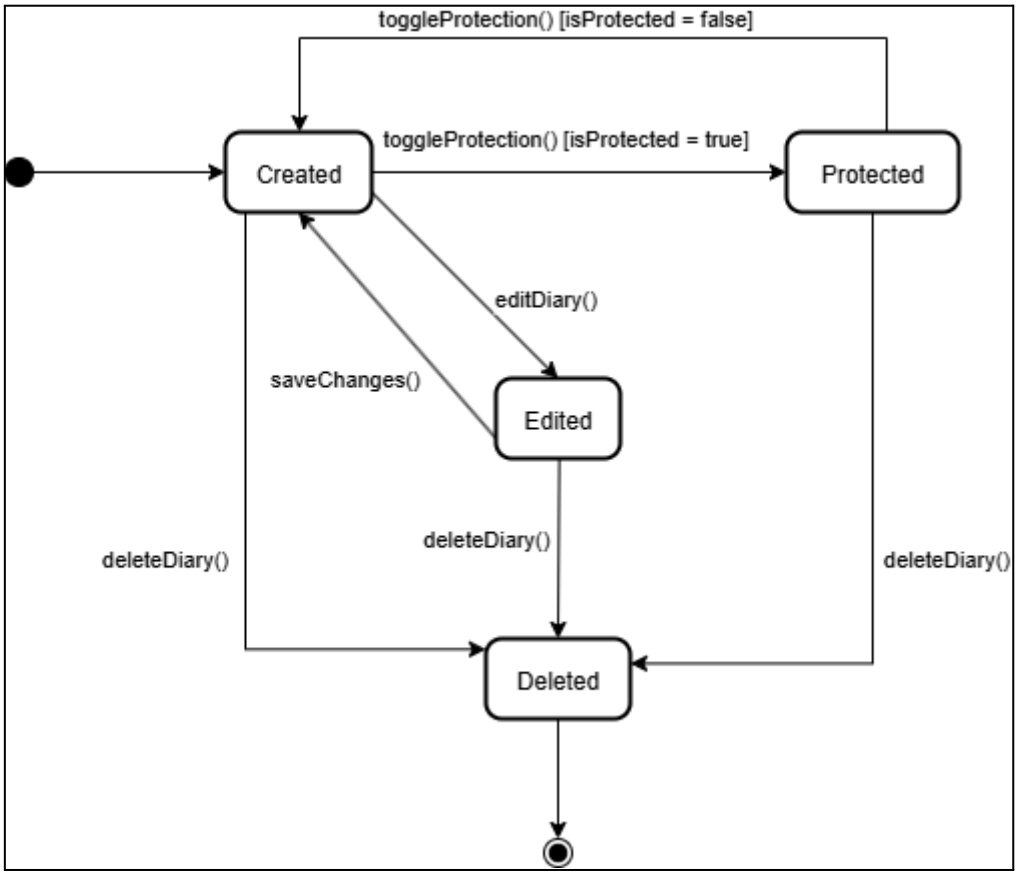


Diagram 4.2.2: State Chart Diagram - EmotionDiary

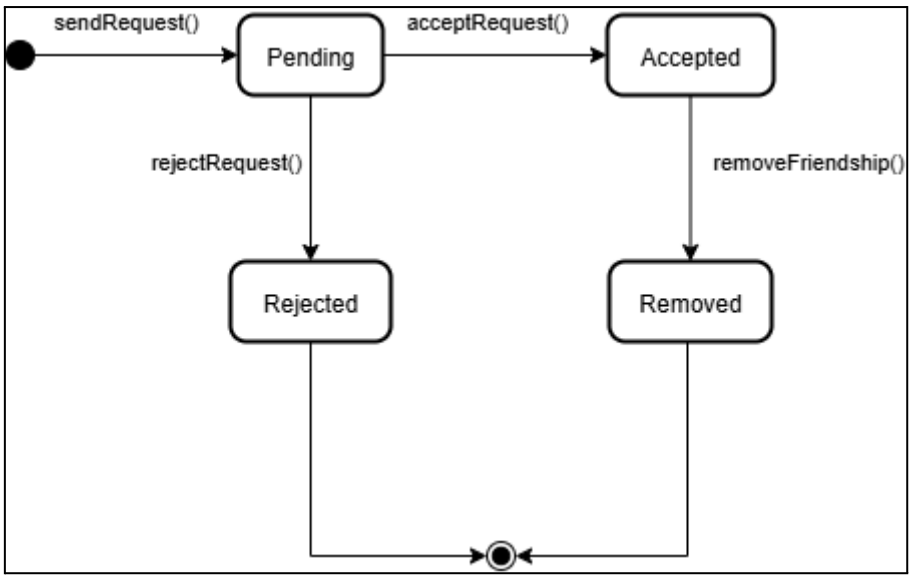


Diagram 4.2.3: State Chart Diagram - Friendship

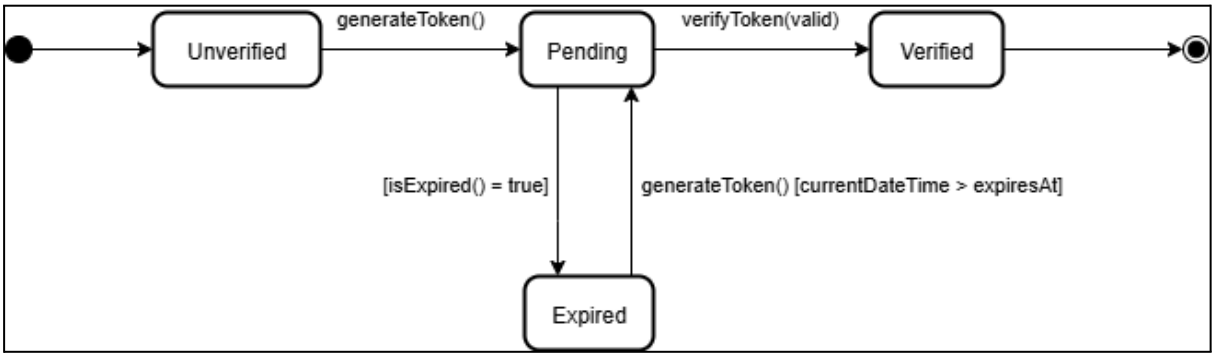


Diagram 4.2.4: State Chart Diagram - EmailVerification

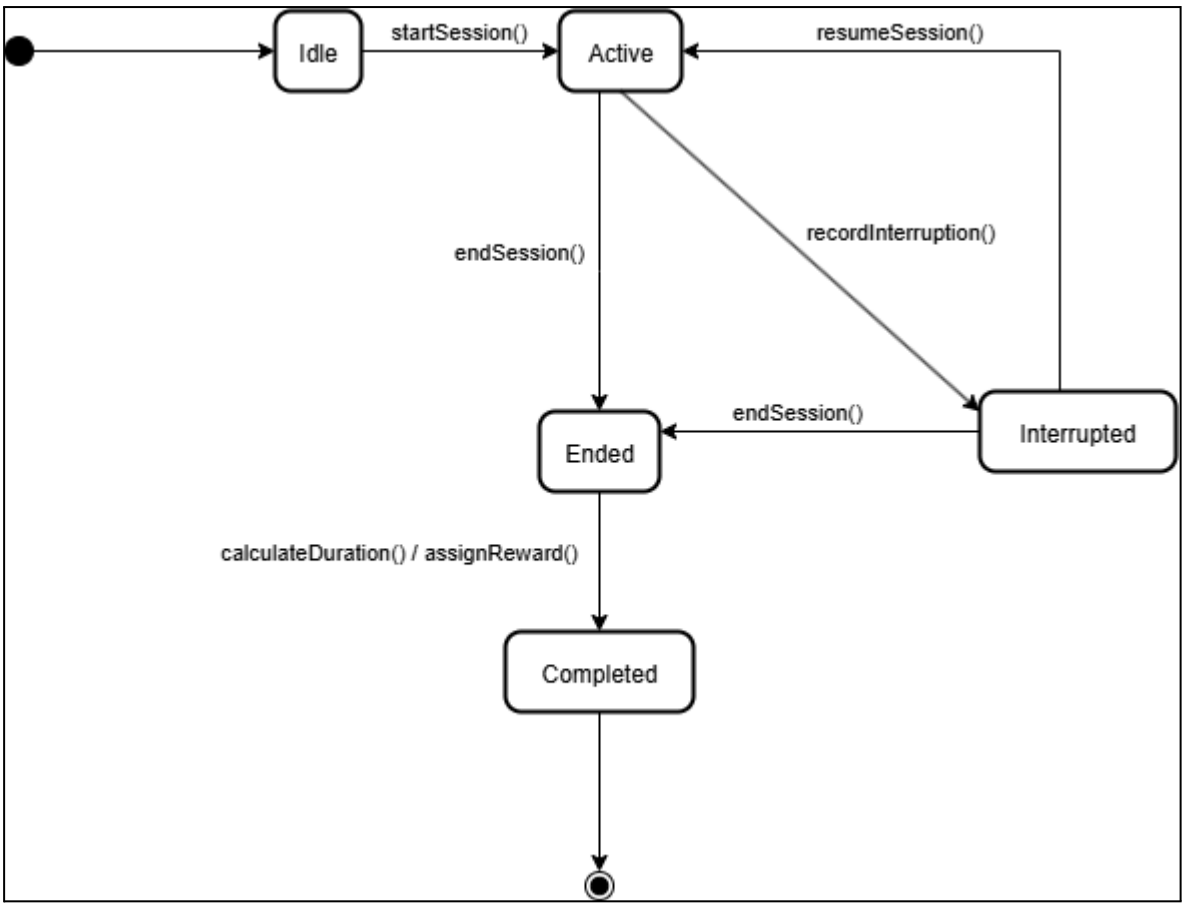


Diagram 4.2.5: State Chart Diagram - FocusSession

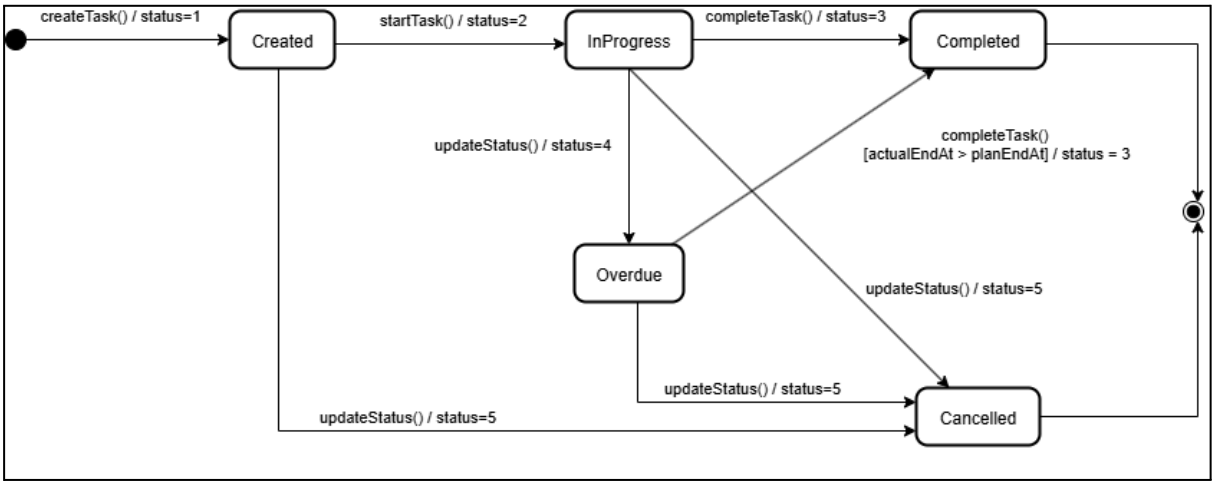


Diagram 4.2.6: State Chart Diagram - Task

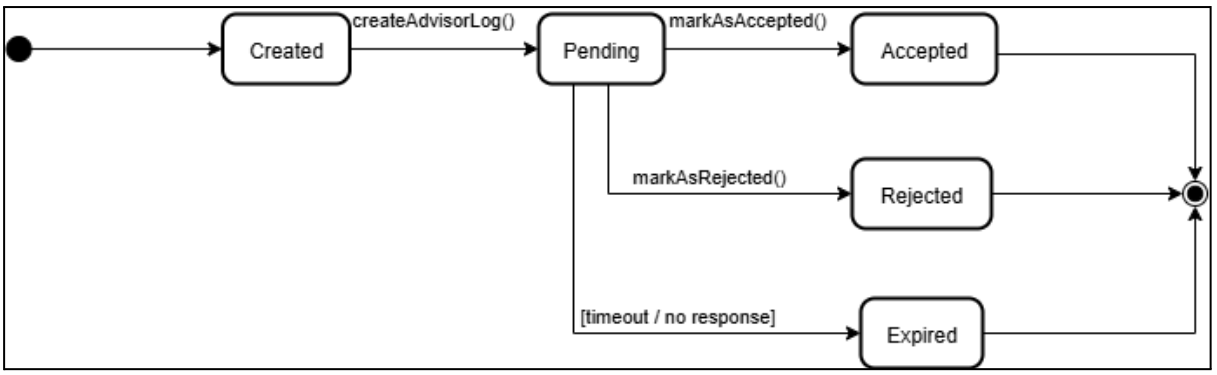
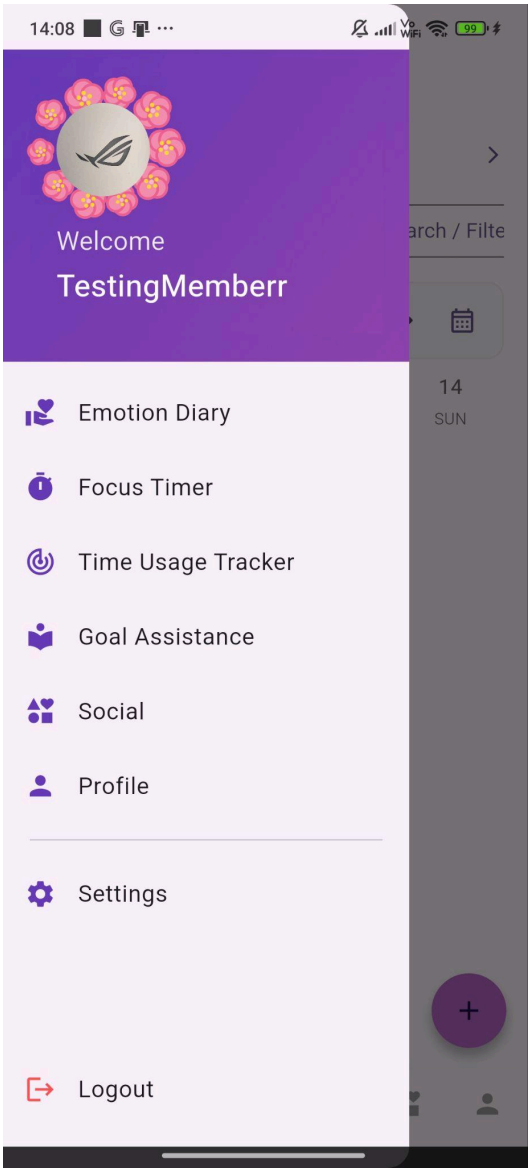


Diagram 4.2.7: State Chart Diagram - TaskAdvisorLog

4.3 User Interface Design

Side navigation drawer



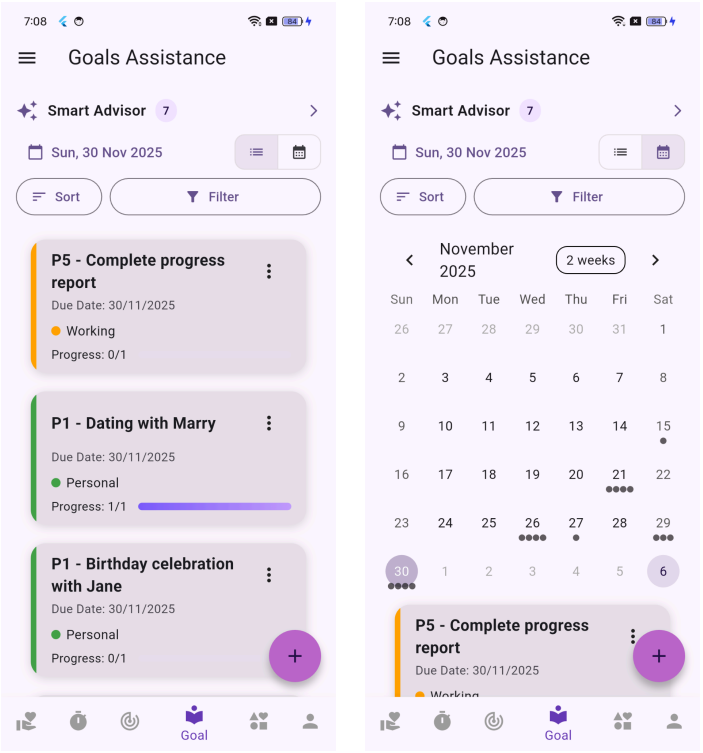
Side navigation drawer: the menu that appears when the user opens the BetterU's sidebar.

There is a profile section with a profile image with profile border redeemed and a welcome message with the username.

Below that, the menu lists the main features of the app, each with an icon.

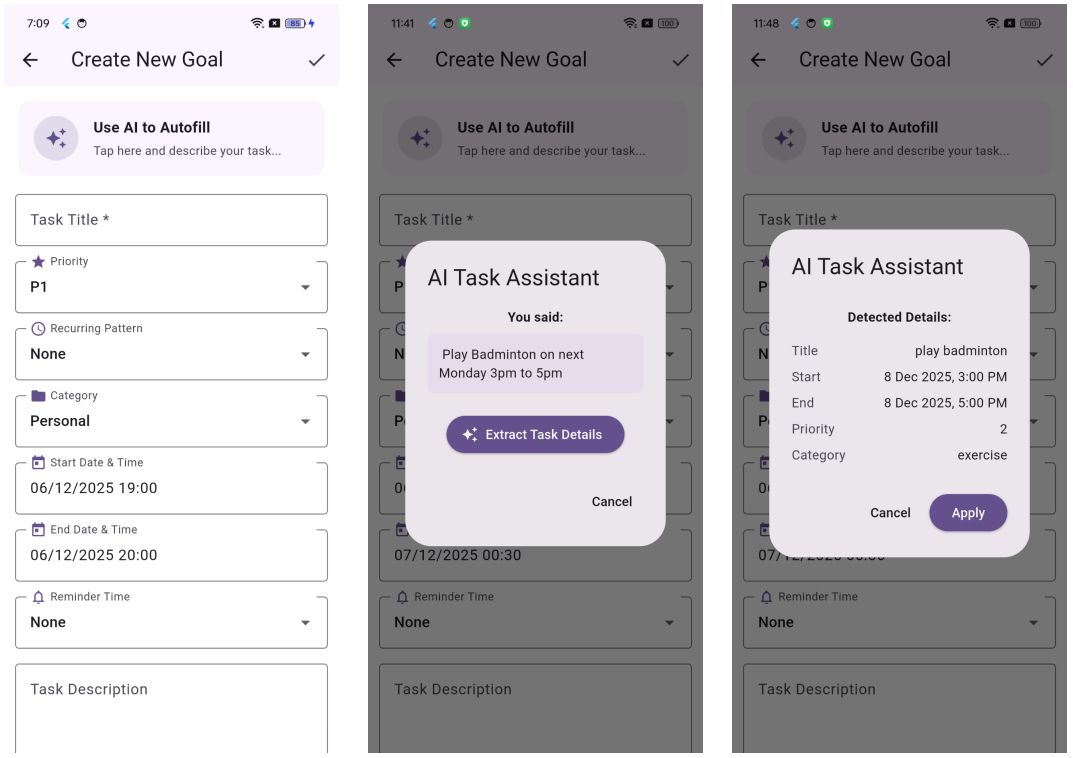
There is a Logout button at the bottom.

Goal Assistance Module (shared)



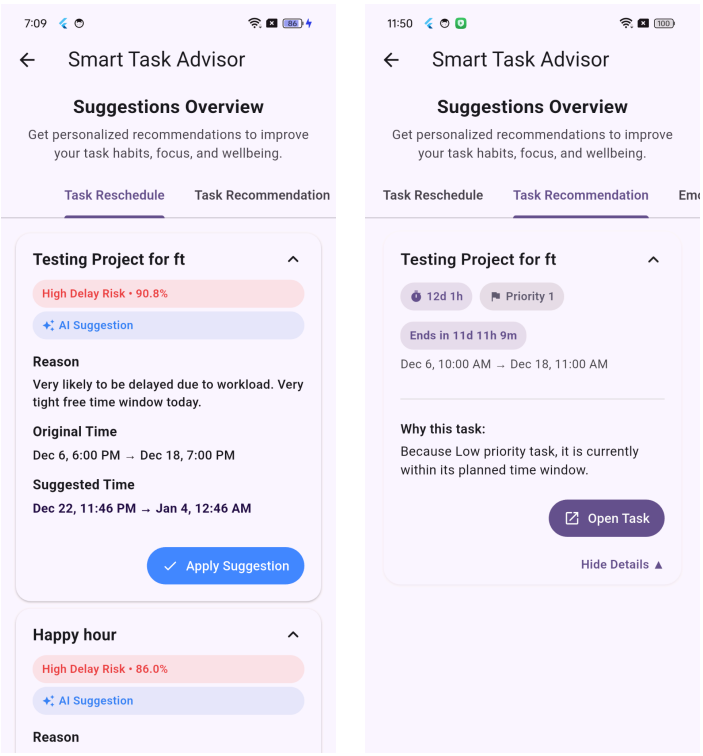
Goal Listing

- The Card View shows each goal as a clean with its title, category, due date and progress. This helps the users to quickly scan and manage their daily tasks.
- The Calendar View shows all goals based on their dates. This allows the users to check when the tasks are due and plan their schedule more easily.



Goal Creation View

- The first screen is the Goal Creation Form where users can manually fill in details such as title, priority, category, start and end date time, subtasks and other relevant info.
- The second and third screen show the AI Task Assistant which enables users to speak their task, automatically transcribe it into text, and then extract important details like title, start datetime, end date time, priority and category. Thus, users can apply these extracted details directly to the form immediately.

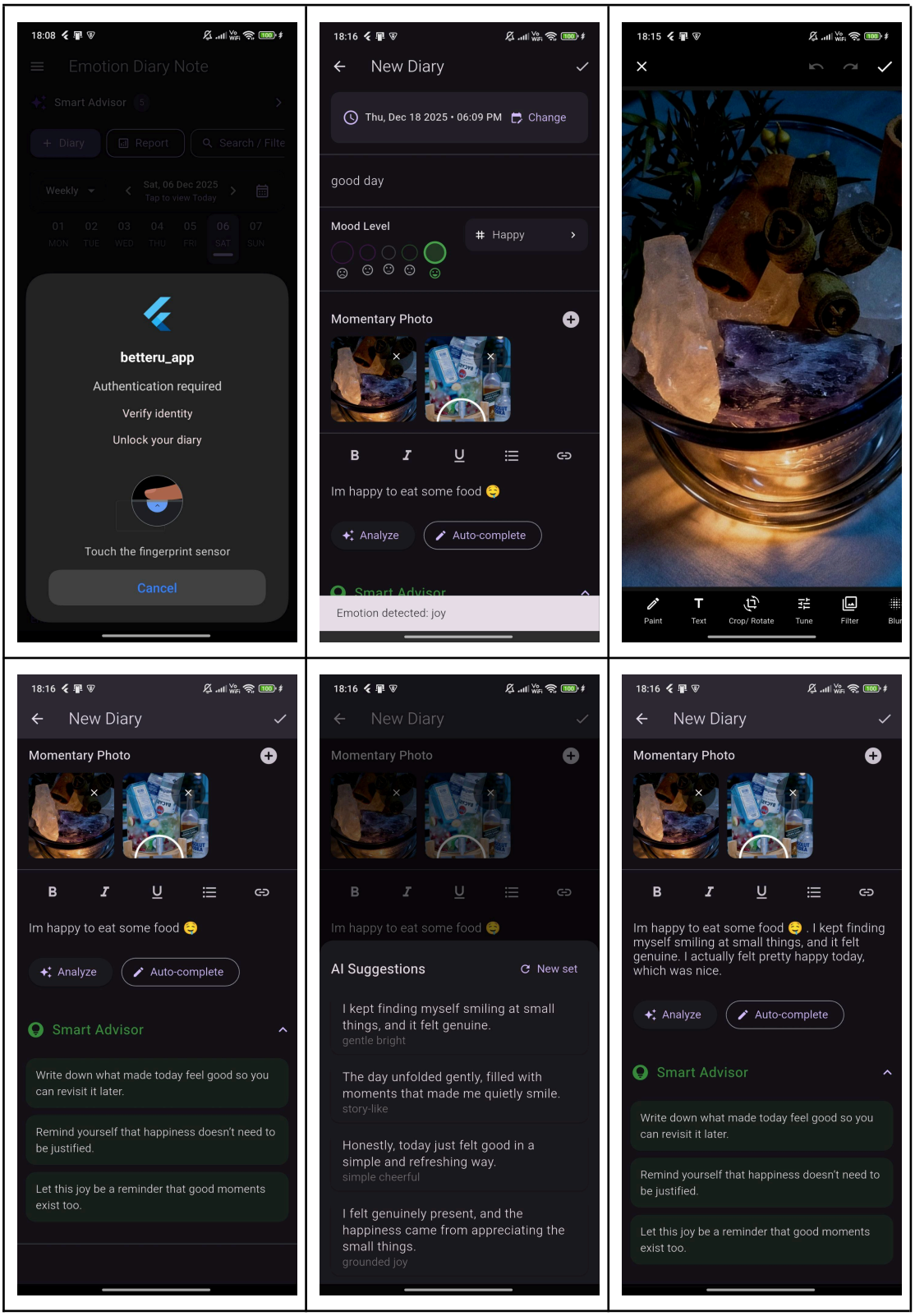


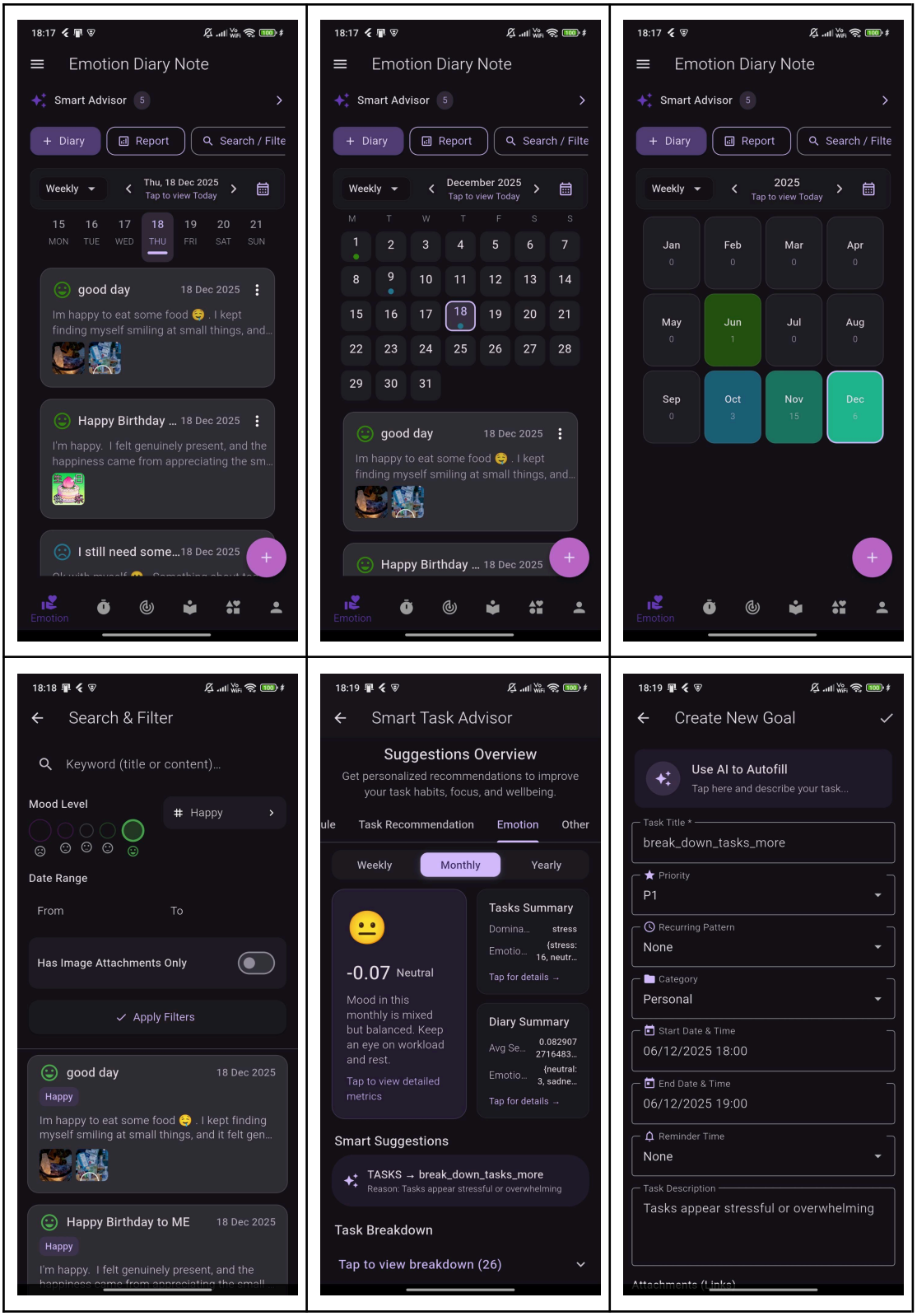
Task Reschedule & Recommendation

- The Task Reschedule view analyzes each task and identifies those that are at risk of being delayed. It explains the reason for the delay risk, shows the original schedule and recommends a new date and time. Users can choose whether to apply the suggested schedule.
- The Task Recommendation view recommends which tasks the user should handle first. The system highlights the priority level, time window and gives a clear reason for why the task is recommended at that moment.

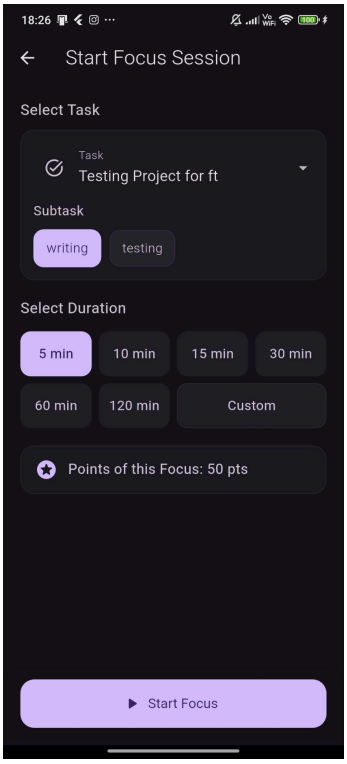
Table 4.3.1: User Interface Design - Goal Assistance Module

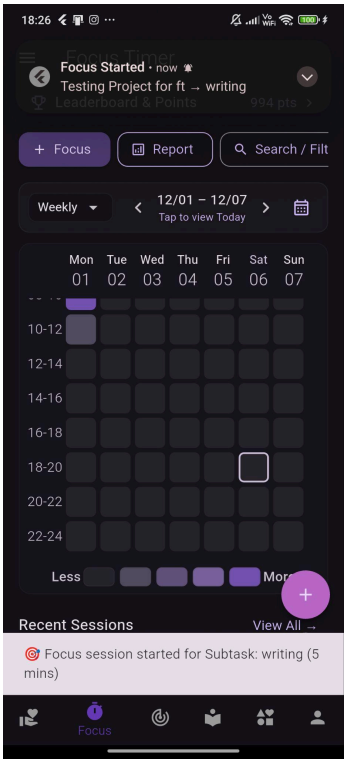
Emotion Diary Module

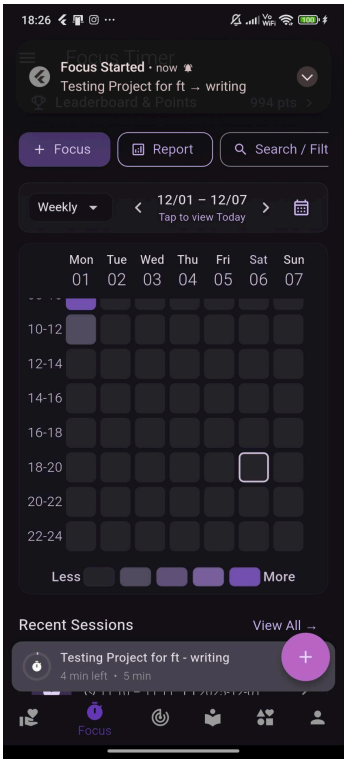


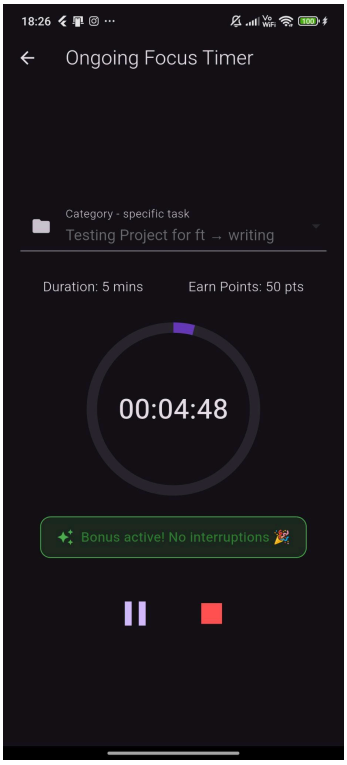

Emotion Diary Note
Smart Advisor
Diary
Report
Search / Filte
Weekly
Thu, 18 Dec 2025
Tap to view Today
good day
18 Dec 2025
Im happy to eat some food . I kept finding myself smiling at small things, and...
Happy Birthday ... 18 Dec 2025
I'm happy. I felt genuinely present, and the happiness came from appreciating the sm...
I still need some... 18 Dec 2025
Emotion
December 2025
2025
Search & Filter
Keyword (title or content)...
Mood Level
Happy
Date Range
From
To
Has Image Attachments Only
Apply Filters
Happy Birthday to ME
Smart Task Advisor
Suggestions Overview
Get personalized recommendations to improve your task habits, focus, and wellbeing.
Task Recommendation
Emotion
Other
Weekly
Monthly
Yearly
-0.07 Neutral
Mood in this monthly is mixed but balanced. Keep an eye on workload and rest.
Tap to view detailed metrics
Tasks Summary
Tap for details
Diary Summary
Smart Suggestions
TASKS → break_down_tasks_more
Reason: Tasks appear stressful or overwhelming
Task Breakdown
Tap to view breakdown (26)
Create New Goal
Use AI to Autofill
Tap here and describe your task...
Task Title *
break_down_tasks_more
Priority
P1
Recurring Pattern
None
Category
Personal
Start Date & Time
06/12/2025 18:00
End Date & Time
06/12/2025 19:00
Reminder Time
None
Task Description
Tasks appear stressful or overwhelming

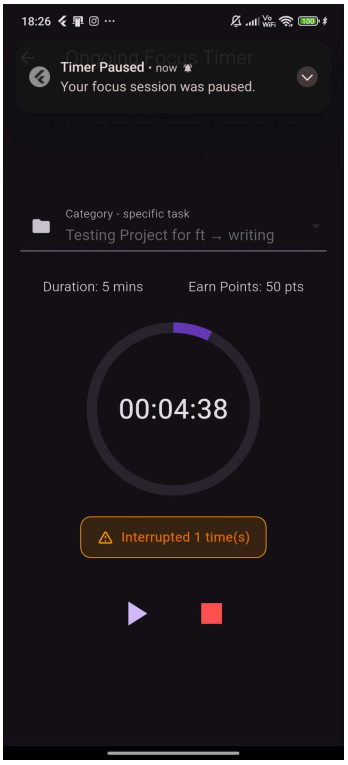
Focus Timer Module

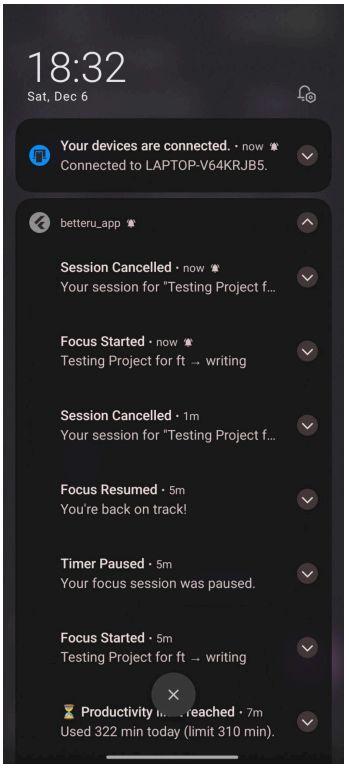




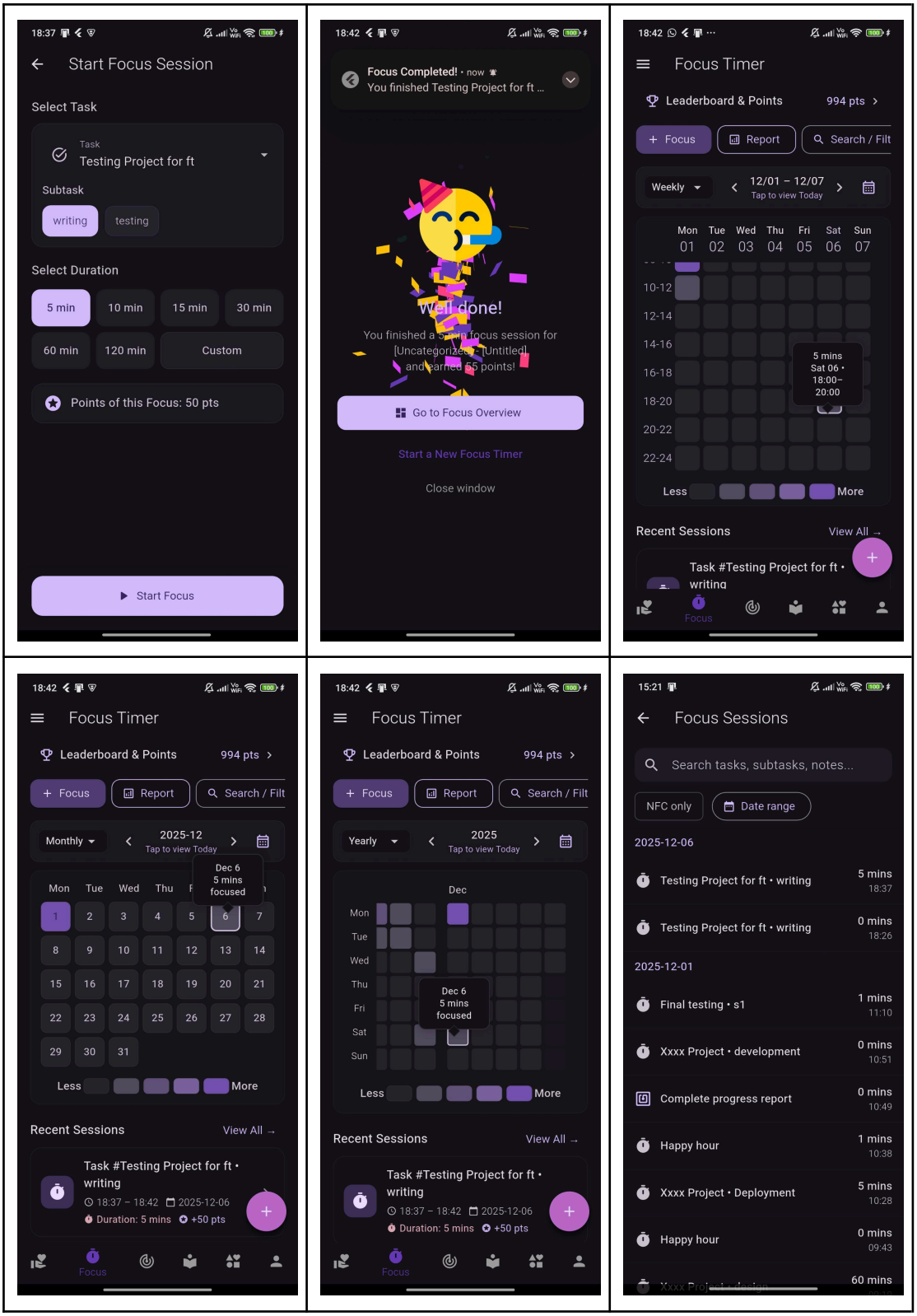


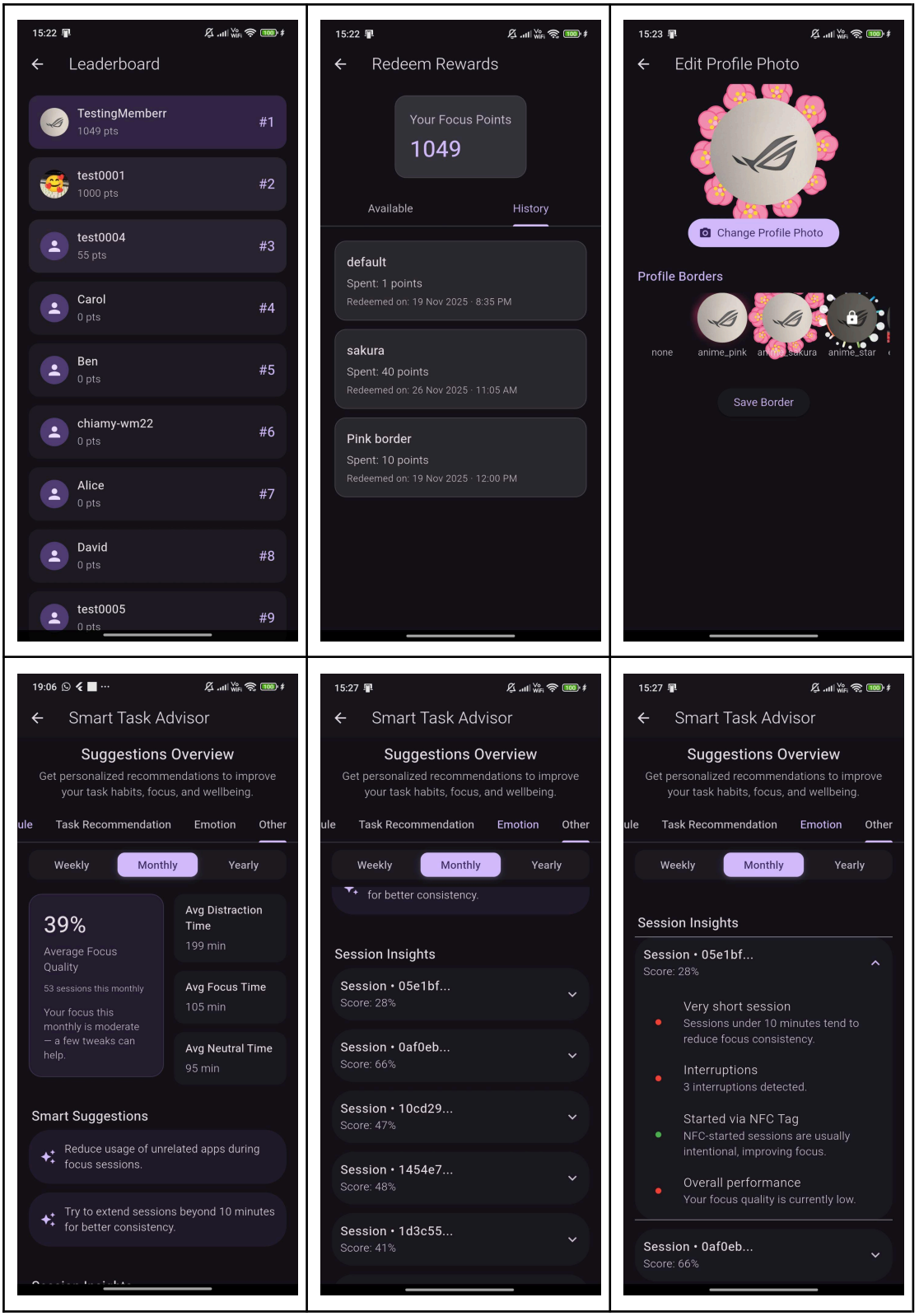




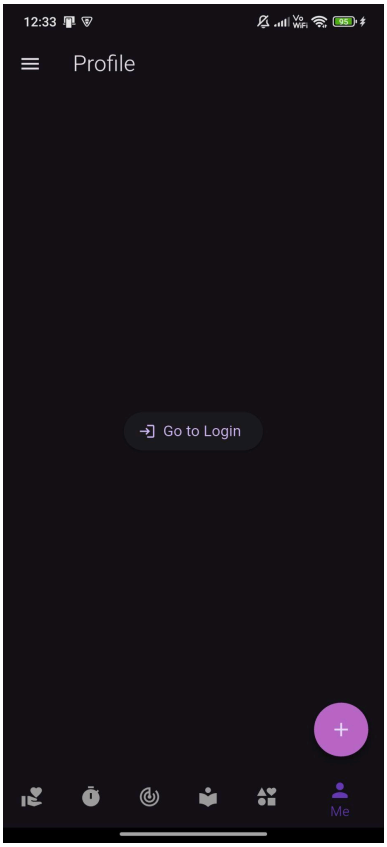


If it is detected that the user is not focusing on the BetterU app, the session will be canceled and a notification will be sent.

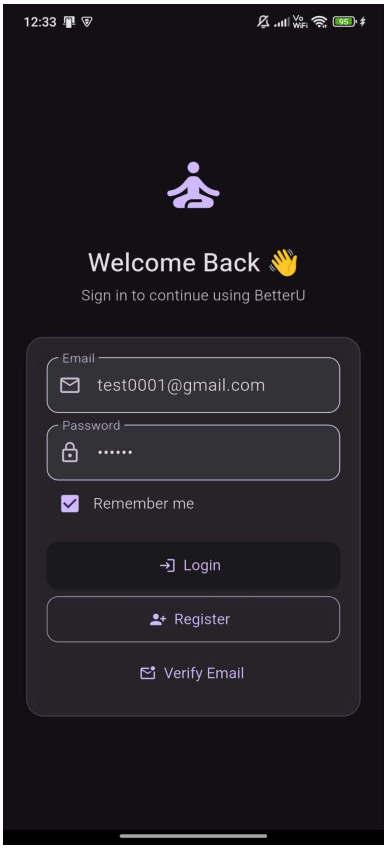

Start Focus Session
Select Task
Task
Testing Project for ft
Subtask
writing
testing
Select Duration
5 min
10 min
15 min
30 min
60 min
120 min
Custom
Points of this Focus: 50 pts
Start Focus
Focus Completed! • now
You finished Testing Project for ft ...
Well done!
You finished a 5-min focus session for
[Uncategorized] - [Untitled]
and earned 55 points!
Go to Focus Overview
Start a New Focus Timer
Close window
Focus Timer
Leaderboard & Points
994 pts
Focus
Report
Search / Filt
Weekly
12/01 – 12/07
Tap to view Today
Mon Tue Wed Thu Fri Sat Sun
01 02 03 04 05 06 07
10-12
12-14
14-16
16-18
18-20
20-22
22-24
5 mins
Sat 06 • 18:00–20:00
Less
More
Recent Sessions
View All →
Task #Testing Project for ft • writing
Focus
Monthly
2025-12
Dec 6
5 mins
focused
Yearly
2025
Dec
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Task #Testing Project for ft • writing
18:37 – 18:42 2025-12-06
Duration: 5 mins +50 pts
Focus Sessions
Search tasks, subtasks, notes...
NFC only
Date range
2025-12-06
Testing Project for ft • writing
5 mins
18:37
Testing Project for ft • writing
0 mins
18:26
2025-12-01
Final testing • s1
1 mins
11:10
Xxxx Project • development
0 mins
10:51
Complete progress report
0 mins
10:49
Happy hour
1 mins
10:38
Xxxx Project • Deployment
5 mins
10:28
Happy hour
0 mins
09:43
60 mins


15:22
Leaderboard
TestingMemberr
1049 pts
#1
test0001
1000 pts
#2
test0004
55 pts
#3
Carol
0 pts
#4
Ben
0 pts
#5
chiamy-wm22
0 pts
#6
Alice
0 pts
#7
David
0 pts
#8
test0005
0 pts
#9
15:22
Redeem Rewards
Your Focus Points
1049
Available
History
default
Spent: 1 points
Redeemed on: 19 Nov 2025 · 8:35 PM
sakura
Spent: 40 points
Redeemed on: 26 Nov 2025 · 11:05 AM
Pink border
Spent: 10 points
Redeemed on: 19 Nov 2025 · 12:00 PM
15:23
Edit Profile Photo
Change Profile Photo
Profile Borders
none
anime_pink
anime_sakura
anime_star
Save Border
19:06
Smart Task Advisor
Suggestions Overview
Get personalized recommendations to improve your task habits, focus, and wellbeing.
Task Recommendation
Emotion
Other
Weekly
Monthly
Yearly
39%
Average Focus Quality
53 sessions this monthly
Your focus this monthly is moderate — a few tweaks can help.
Avg Distraction Time
199 min
Avg Focus Time
105 min
Avg Neutral Time
95 min
Smart Suggestions
Reduce usage of unrelated apps during focus sessions.
Try to extend sessions beyond 10 minutes for better consistency.
15:27
Smart Task Advisor
Suggestions Overview
Get personalized recommendations to improve your task habits, focus, and wellbeing.
Task Recommendation
Emotion
Other
Weekly
Monthly
Yearly
for better consistency.
Session Insights
Session • 05e1bf...
Score: 28%
Session • 0af0eb...
Score: 66%
Session • 10cd29...
Score: 47%
Session • 1454e7...
Score: 48%
Session • 1d3c55...
Score: 41%
15:27
Smart Task Advisor
Suggestions Overview
Get personalized recommendations to improve your task habits, focus, and wellbeing.
Task Recommendation
Emotion
Other
Weekly
Monthly
Yearly
Session Insights
Session • 05e1bf...
Score: 28%
Very short session
Sessions under 10 minutes tend to reduce focus consistency.
Interruptions
3 interruptions detected.
Started via NFC Tag
NFC-started sessions are usually intentional, improving focus.
Overall performance
Your focus quality is currently low.
Session • 0af0eb...
Score: 66%

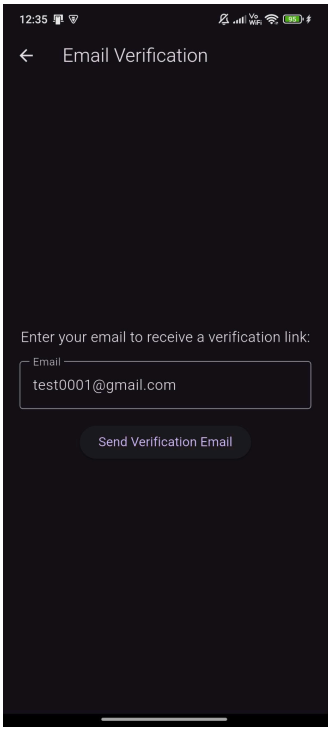
User Module



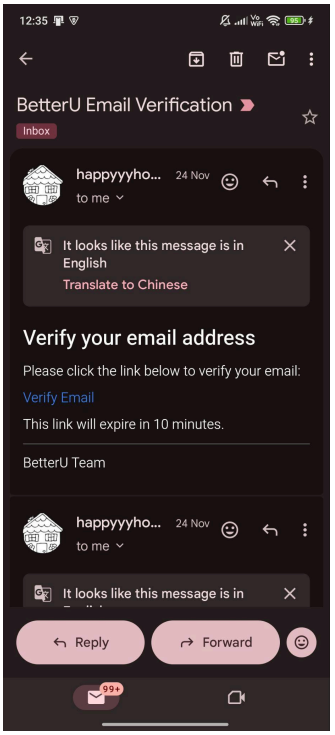
Page before login



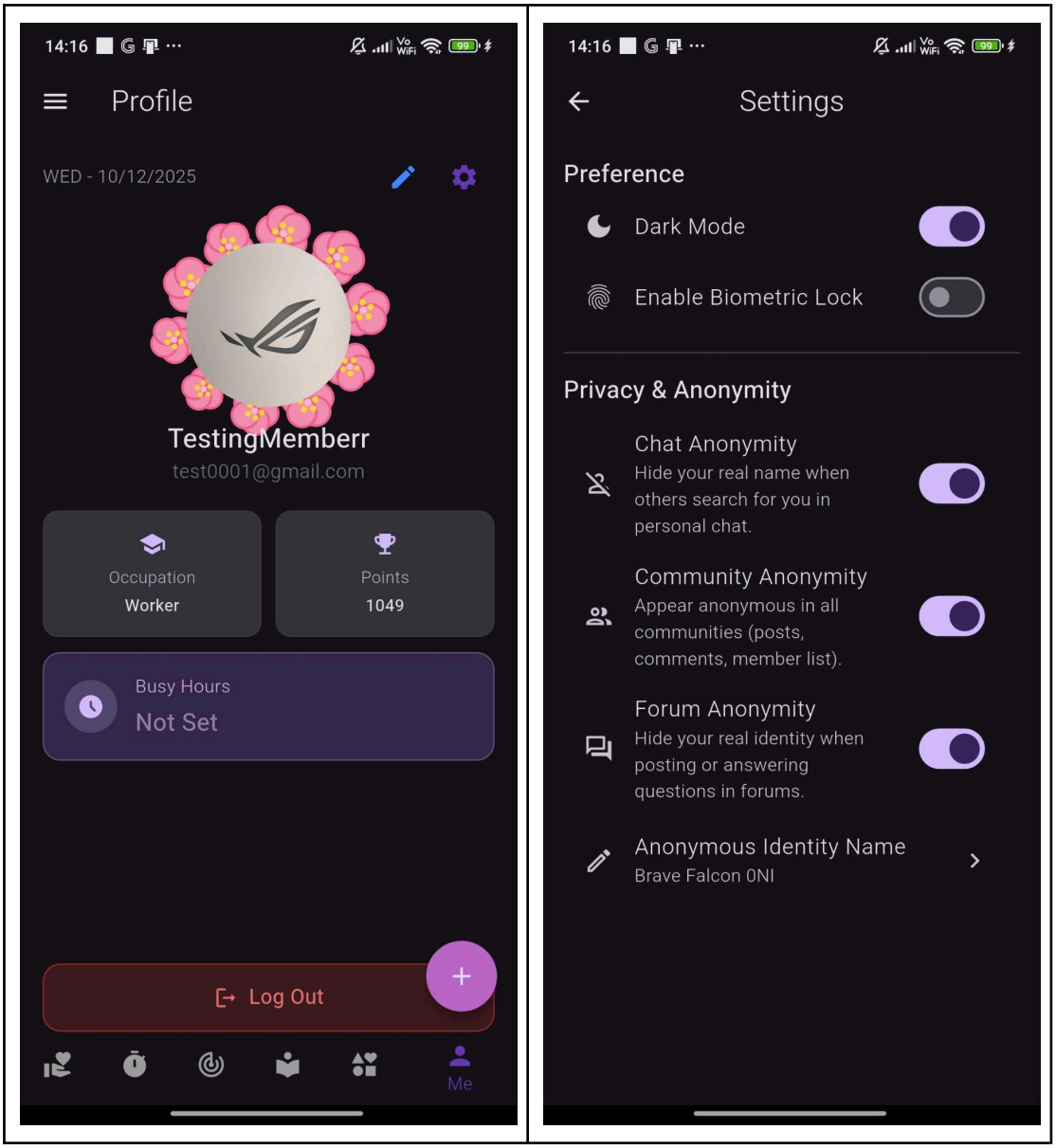
Login and Register Page



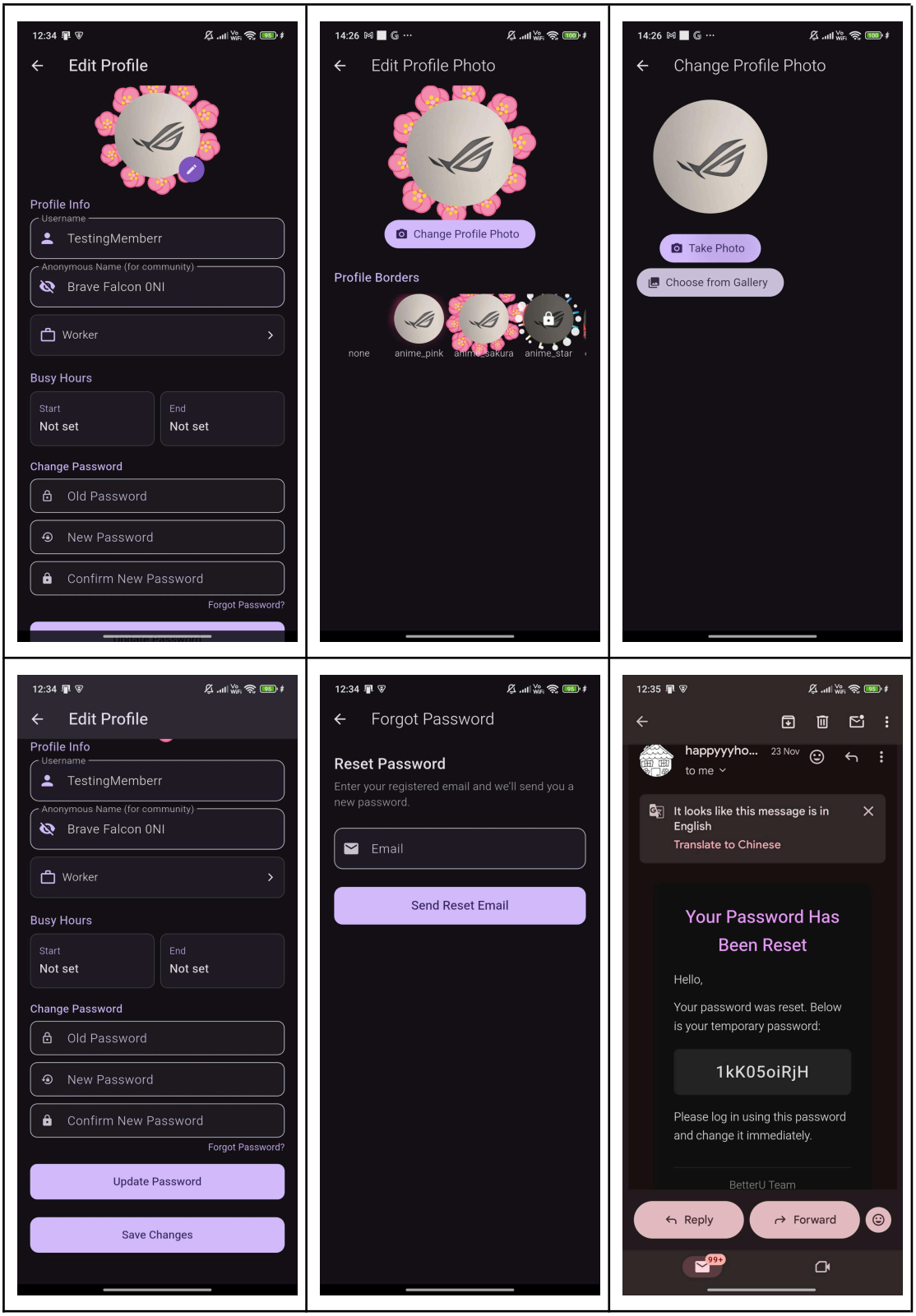
Email verification page



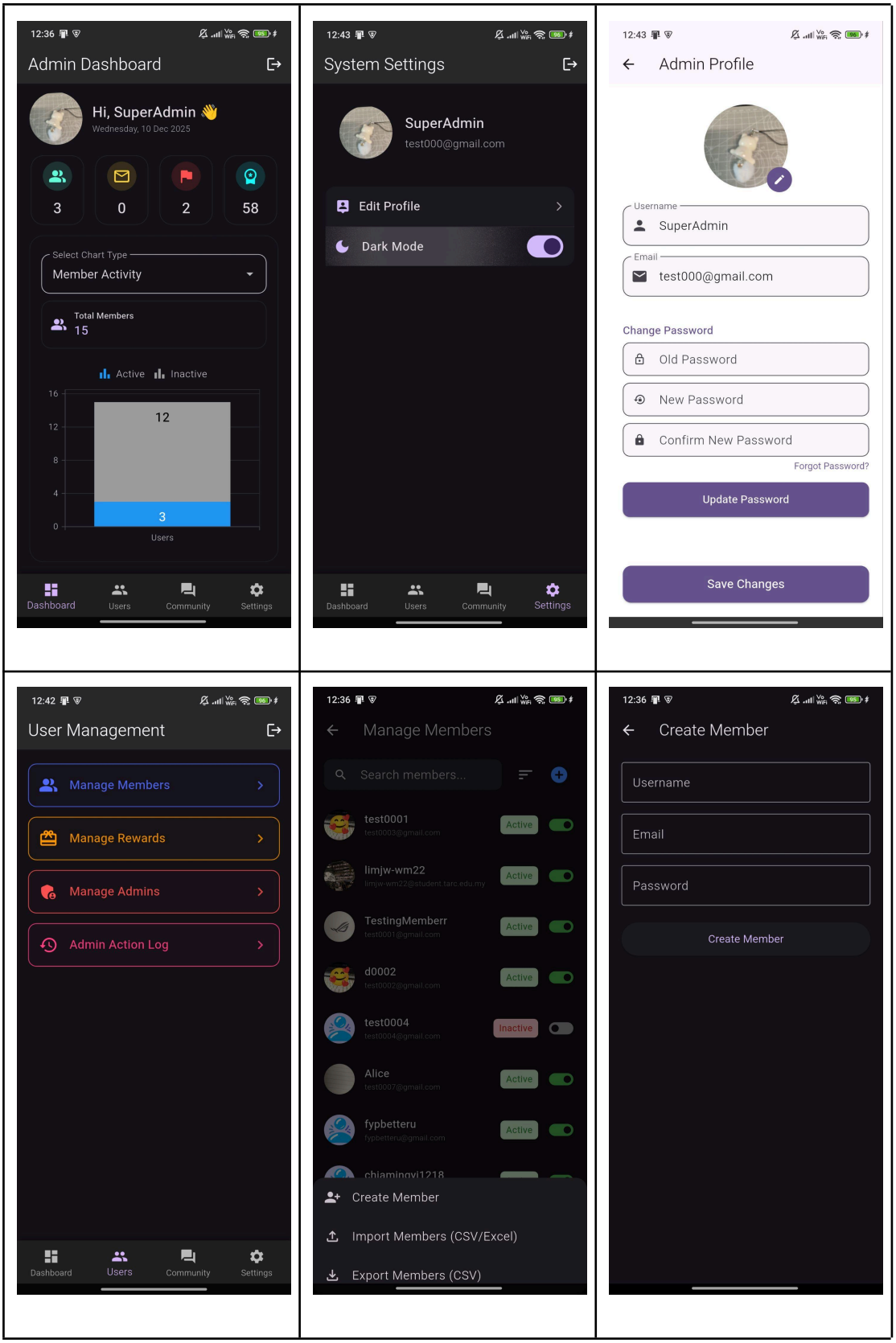
Email content of Email verification

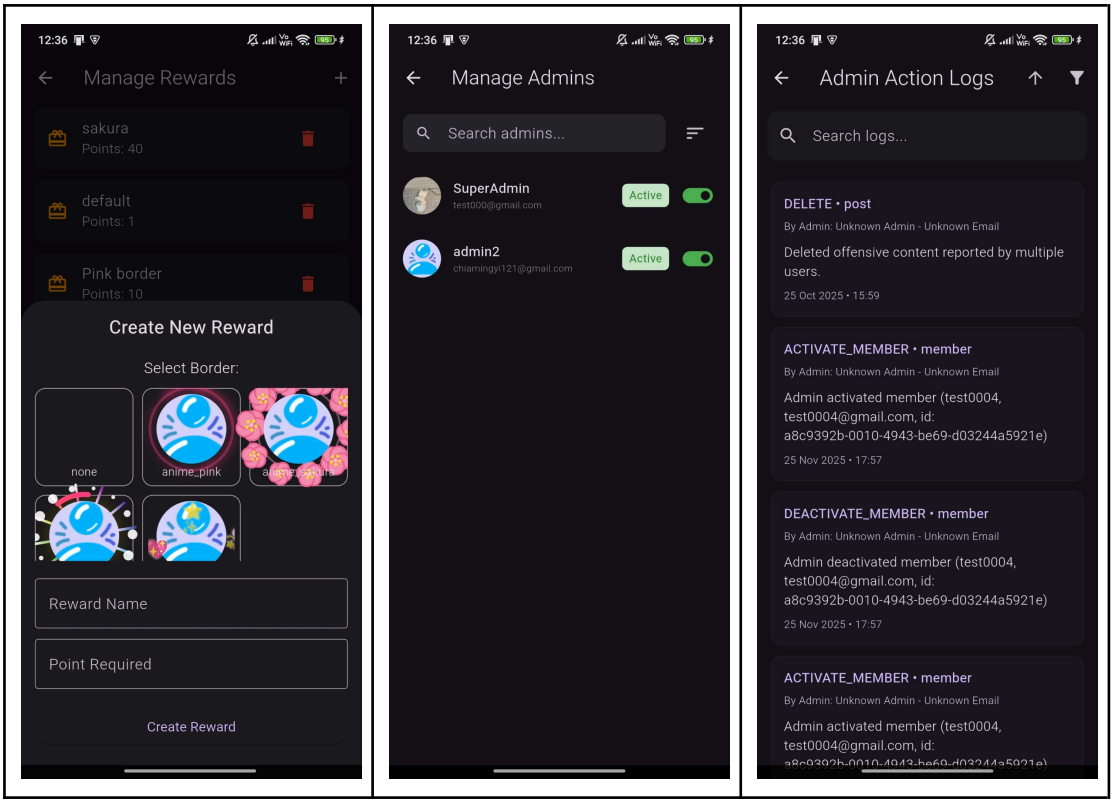

14:16
Profile
WED - 10/12/2025
TestingMemberr
test0001@gmail.com
Occupation
Worker
Points
1049
Busy Hours
Not Set
Log Out
Me
14:16
Settings
Preference
Dark Mode
Enable Biometric Lock
Privacy & Anonymity
Chat Anonymity
Hide your real name when others search for you in personal chat.
Community Anonymity
Appear anonymous in all communities (posts, comments, member list).
Forum Anonymity
Hide your real identity when posting or answering questions in forums.
Anonymous Identity Name
Brave Falcon 0NI

Edit Member Profile:



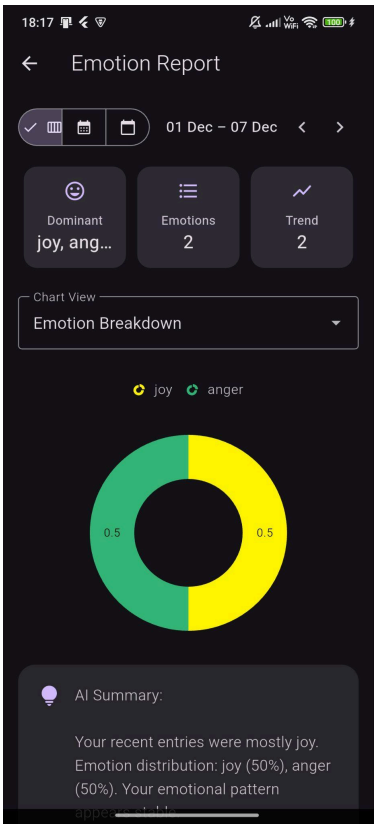
Admin Side Design:

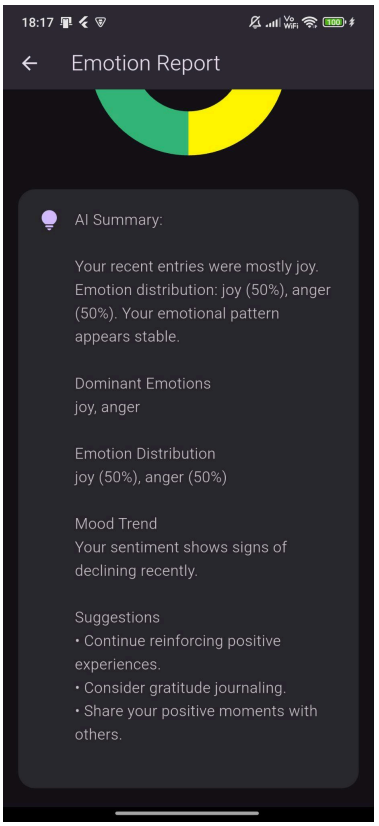


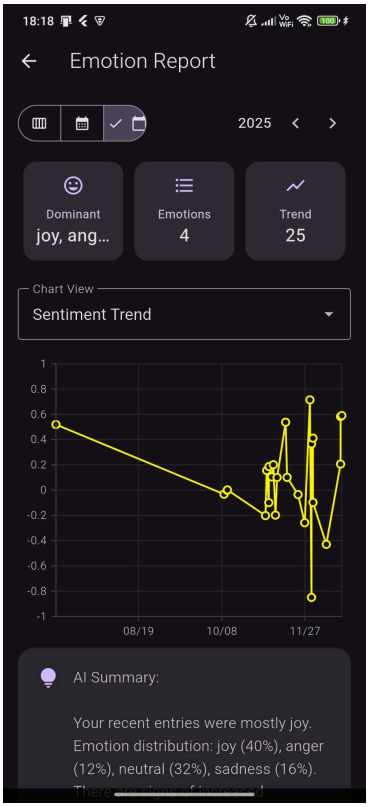

Manage Rewards
sakura
Points: 40
default
Points: 1
Pink border
Points: 10
Create New Reward
Select Border:
none
anime_pink
Reward Name
Point Required
Create Reward
Manage Admins
Search admins...
SuperAdmin
Active
admin2
Active
Admin Action Logs
Search logs...
DELETE • post
By Admin: Unknown Admin - Unknown Email
Deleted offensive content reported by multiple users.
25 Oct 2025 • 15:59
ACTIVATE_MEMBER • member
By Admin: Unknown Admin - Unknown Email
Admin activated member (test0004, test0004@gmail.com, id: a8c9392b-0010-4943-be69-d03244a5921e)
25 Nov 2025 • 17:57
DEACTIVATE_MEMBER • member
By Admin: Unknown Admin - Unknown Email
Admin deactivated member (test0004, test0004@gmail.com, id: a8c9392b-0010-4943-be69-d03244a5921e)
25 Nov 2025 • 17:57
ACTIVATE_MEMBER • member
By Admin: Unknown Admin - Unknown Email
Admin activated member (test0004, test0004@gmail.com, id:

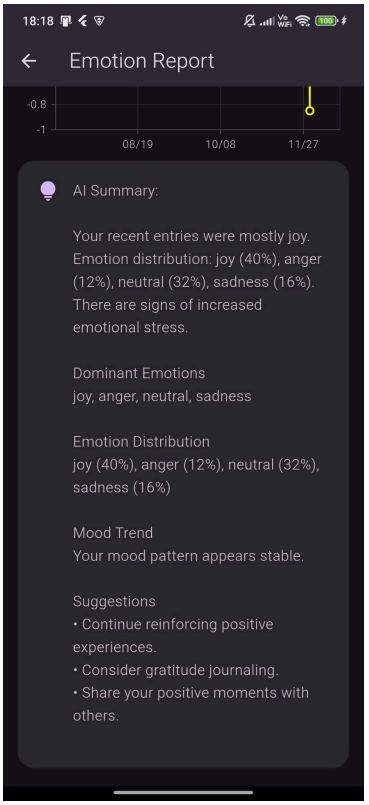
Report Module

Member side (Emotion Report):

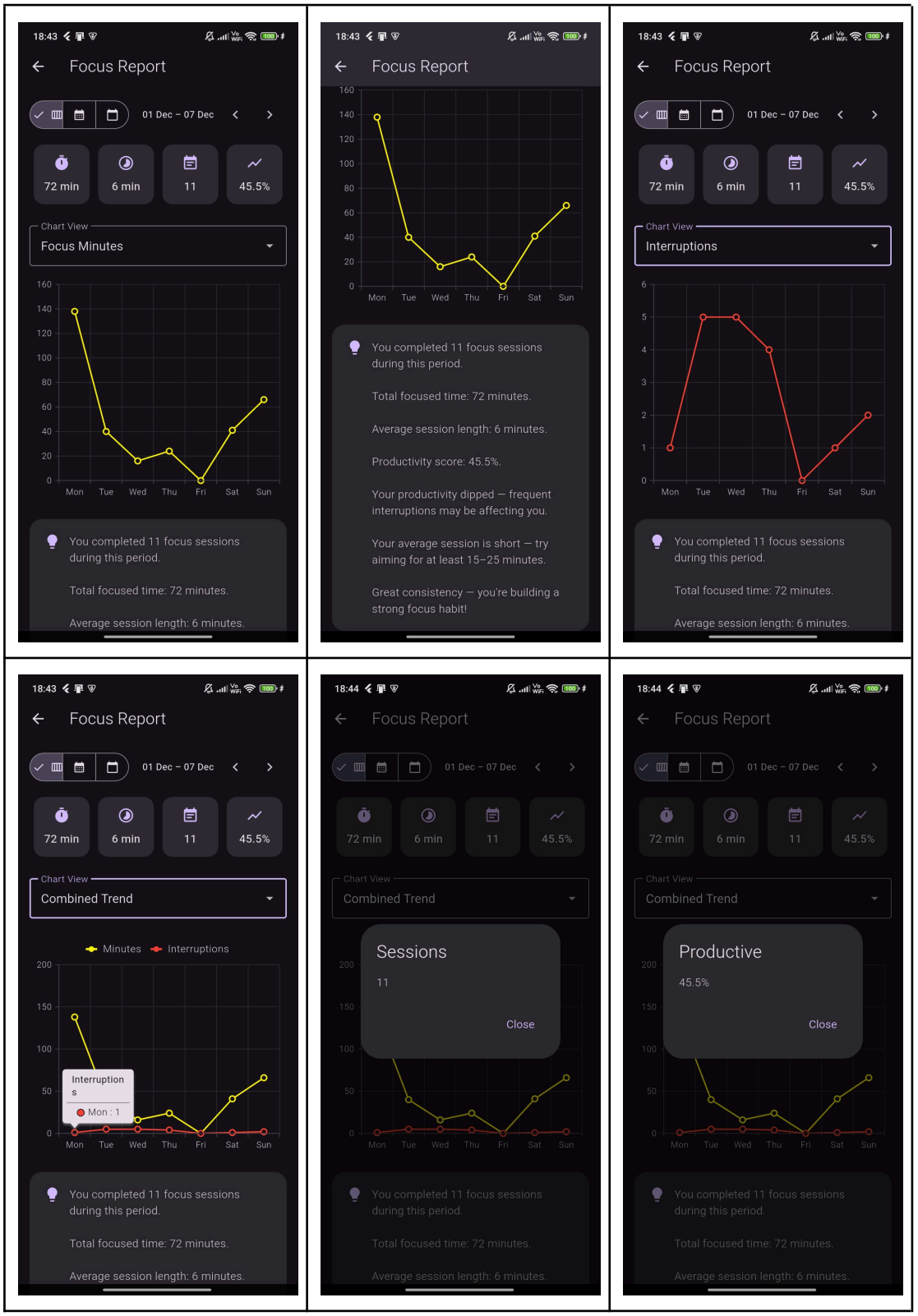




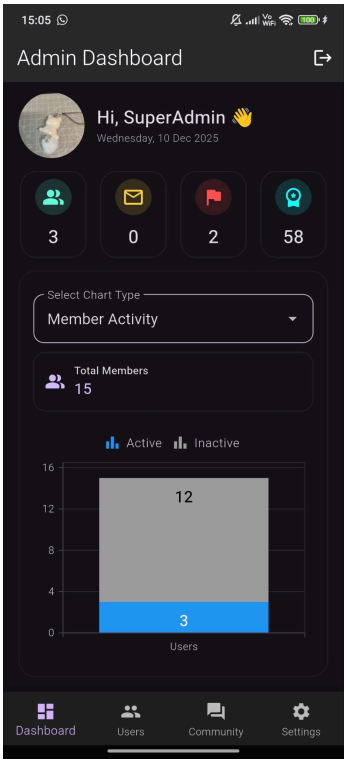
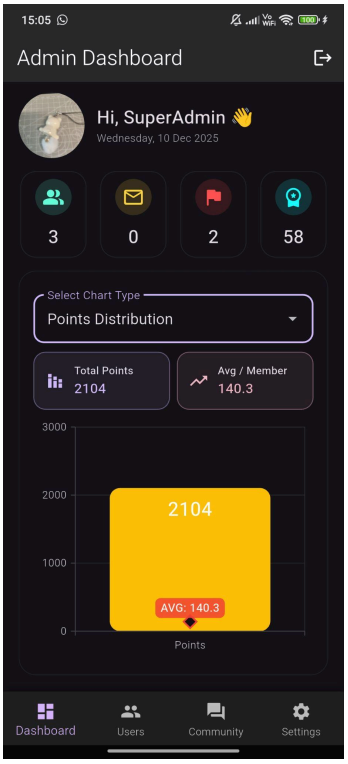
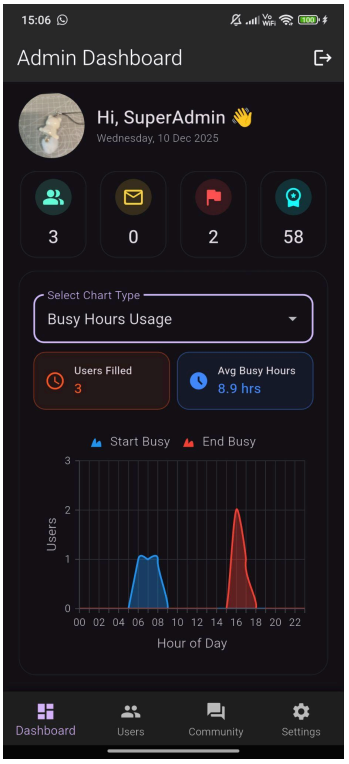




Member side (Focus Report):



Admin side:

		
Gain insight about member activity (check last login at)	Gain insight about focus point distribute to member	Gain insight about busy hour duration of member

4.4 Data Design

4.4.1 Class Diagram

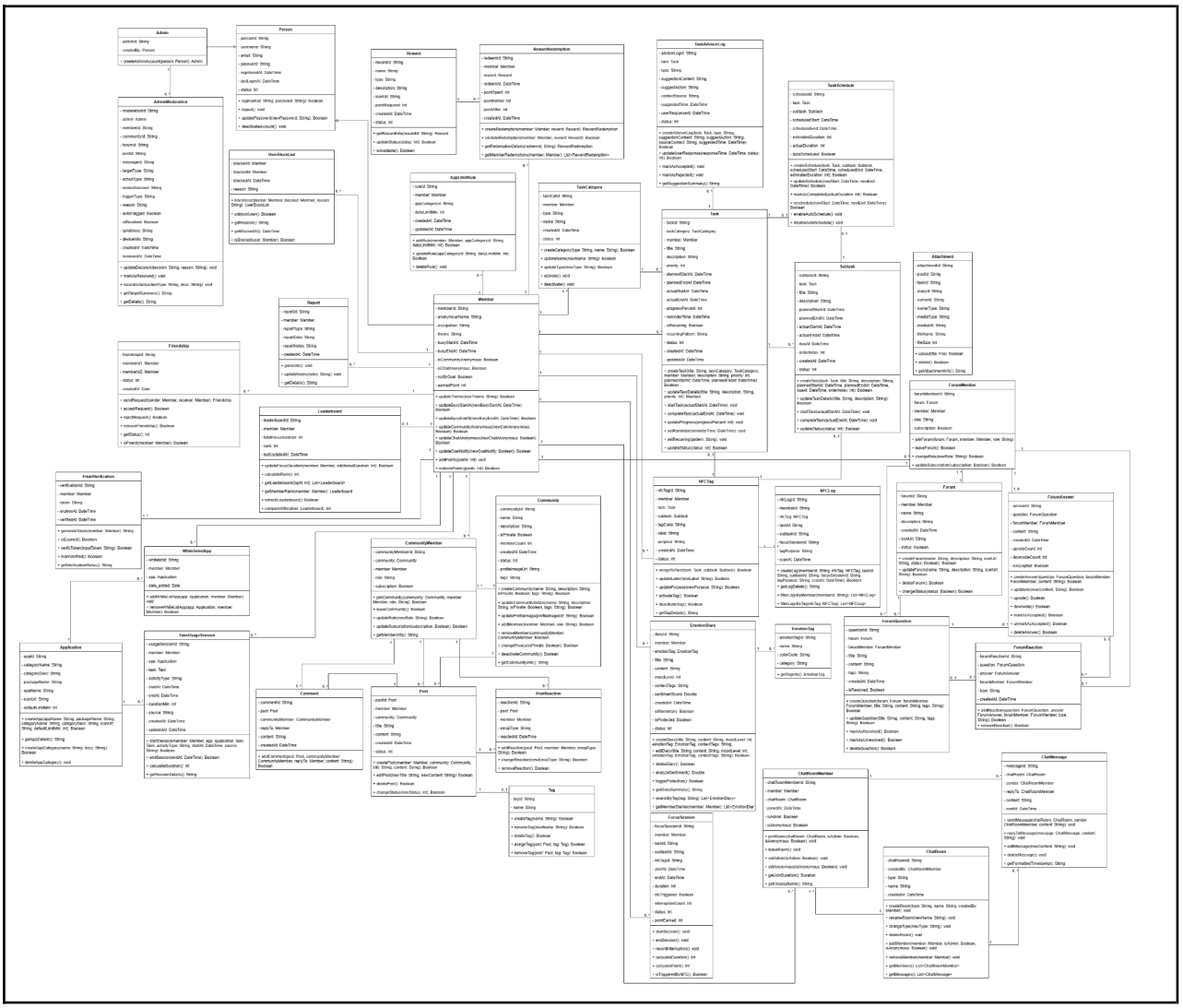


Diagram 4.4.1: Class Diagram of BetterU Mobile Application

4.4.2 Entity Relationship Diagram

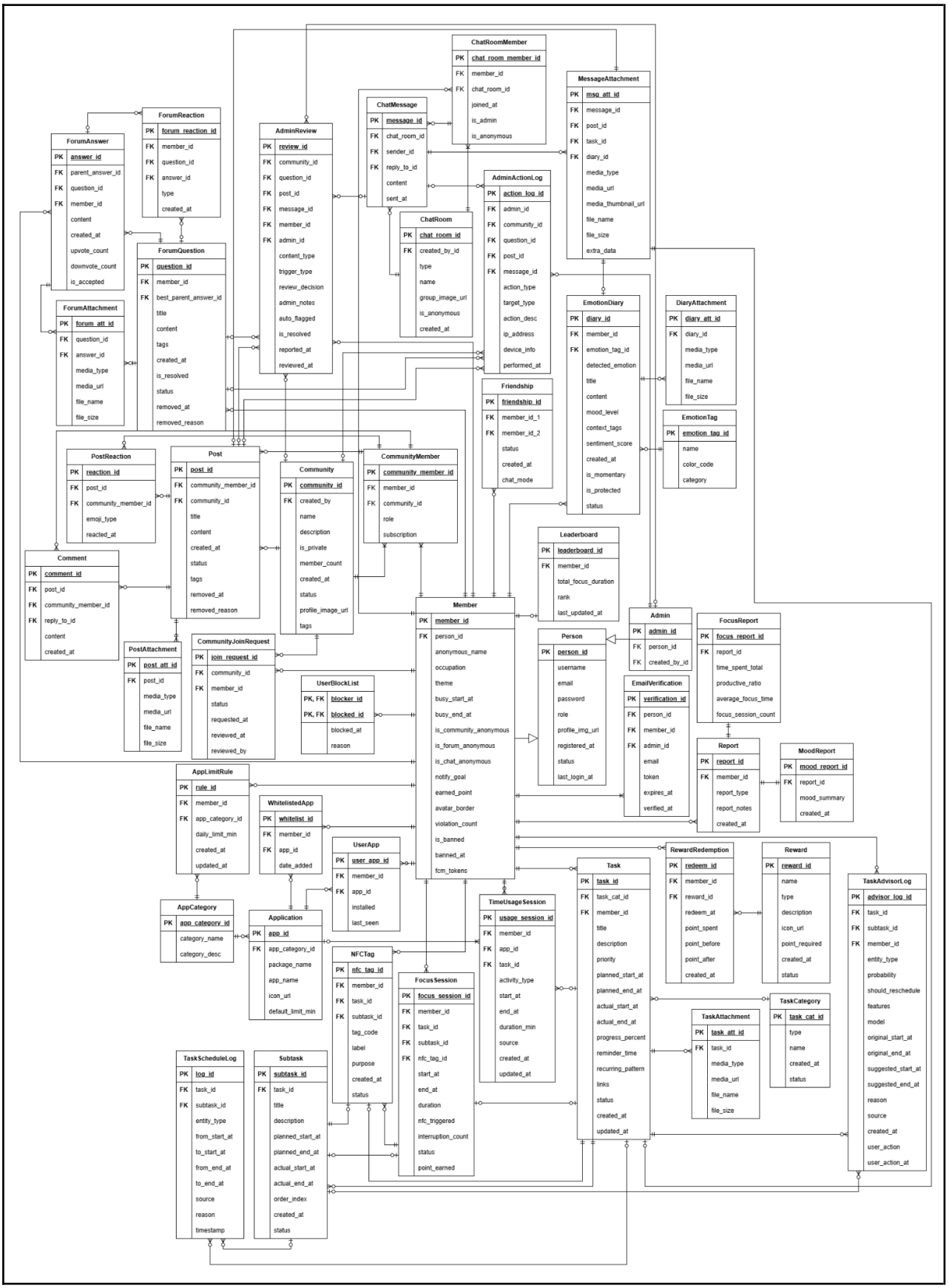


Diagram 4.4.2: Entity Relationship Diagram of BetterU Mobile Application

4.4.3 Data Dictionary

Person

Field Name	Data Type	Size/Constraint	Key	Default	Description
person_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each person (primary key).
username	String	VARCHAR(50), Unique, Not Null	-	-	Display name chosen by the user for login and identification.
email	String	VARCHAR(100), Unique, Not Null	-	-	User's email address, used for login and communication.
password	String	VARCHAR(255), Not Null (hashed)	-	-	User's encrypted/hashed password for authentication.
role	String	VARCHAR(20), Not Null	-	-	Defines user's system role such as member or admin
profile_img_url	String	VARCHAR(255), Nullable	-	NULL	URL path pointing to the user's profile image stored
registered_at	DateTime	Not Null	-	Current timestamp	The date and time when the user registered.
last_login_at	DateTime	Nullable	-	NULL	Timestamp of the admin's

					most recent successful login.
status	Int	Enum: {0=Inactive, 1=Active}	-	1 (Active)	Account status indicator.

Table 4.4.1: Data Dictionary - Person Entity

Member

Field Name	Data Type	Size/Constraint	Key	Default	Description
member_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each member (primary key).
person_id	String	UUID (36), Not Null	FK	-	References Person.person_id (foreign key).
anonymous_name	String	VARCHAR(50), Nullable	-	NULL	Pseudonym used in community/forum features if anonymity is enabled.
occupation	String	VARCHAR(100), Nullable	-	NULL	Member's occupation or role (optional).
theme	String	Enum: {Light, Dark, System}	-	System	Preferred UI theme for the app interface.
busy_start_at	Time	Nullable	-	NULL	Start time of daily busy period (used for scheduling/notifications).

busy_end_at	Time	Nullable	-	NULL	End time of daily busy period.
is_community_anonymous	Boolean	-	-	false	If true, the member appears anonymous in community posts.
is_forum_anonymous	Boolean	-	-	false	If true, the member appears anonymous when posting in the forum.
is_chat_anonymous	Boolean	-	-	false	If true, the member appears anonymous in chats.
notify_goal	Boolean	-	-	true	Whether the member receives goal-related notifications.
earned_point	Integer	>=0	-	0	Accumulated reward points earned through app activities.
avatar_border	String	VARCHAR(50), Nullable	-	NULL	The selected decorative border style for the member's avatar.
violation_count	Integer	Not Null, >= 0	-	0	Total number of approved violations committed by the member.

is_banned	Boolean	Not Null	-	False	Indicates whether the member is banned due to repeated violations.
banned_at	DateTime	Nullable	-	NULL	Timestamp of when the member was banned.
fcm_tokens	Array	Array of VARCHAR(255), Nullable	-	Empty list ([])	List of Firebase Cloud Messaging (FCM) tokens for the member's devices for pushing notifications.

Table 4.4.2: Data Dictionary - Member Entity

Admin

Field Name	Data Type	Size/Constraint	Key	Default	Description
admin_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each admin (primary key).
person_id	String	UUID (36), Not Null	FK	-	References Person.person_id (foreign key). Links the admin role to a person.
created_by_id	String	UUID (36), Nullable	FK (self-ref)	NULL	References Admin.admin_id who created this admin account (for tracking delegation).

Table 4.4.3: Data Dictionary - Admin Entity

EmailVerification

Field Name	Data Type	Size/Constraint	Key	Default	Description
verification_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each email verification record.
person_id	String	UUID (36), Not Null	FK	-	References Person.person_id as a universal identifier for the individual whose email is being verified.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id (the member whose email is being verified).
admin_id	String	UUID (36), Nullable	FK	-	References Admin.admin_id when the email verification is initiated or approved by an administrator.
email	String	VARCHAR (255), Not Null	-	-	The email address undergoing verification.
token	String	VARCHAR(255), Unique, Not Null	-	-	Randomly generated secure token used for email verification.

expires_at	DateTime	Not Null	-	-	Timestamp indicating when the verification token becomes invalid.
verified_at	DateTime	Nullable	-	NULL	Timestamp of when the email was successfully verified.

Table 4.4.4: Data Dictionary - EmailVerification Entity

Friendship

Field Name	Data Type	Size/Constraint	Key	Default	Description
friendship_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each friendship record.
member_id_1	String	UUID (36), Not Null	FK	-	References Member.member_id. Represents one side of the friendship.
member_id_2	String	UUID (36), Not Null	FK	-	References Member.member_id. Represents the other side of the friendship.
status	String	Enum: {0=Blocked, 1=Pending, 2=Accepted, 3=Declined}	-	1	Current status of the friendship request.
created_at	DateTime	Not Null	-	Current timestamp	The date and time when the friendship

					request was initiated.
chat_mode	String	Enum: {"real", "anonymous"}, Not Null	-	real	Preferred chat identity mode selected by the requester.

Table 4.4.5: Data Dictionary - Friendship Entity

UserBlockList

Field Name	Data Type	Size/Constraint	Key	Default	Description
blocker_id	String	UUID (36), Not Null	PK, FK	-	References Member.member_id. The member who initiates the block.
blocked_id	String	UUID (36), Not Null	PK, FK	-	References Member.member_id. The member being blocked.
blocked_at	DateTime	Not Null	-	Current timestamp	The date and time when the block action occurred.
reason	String	VARCHAR(255), Nullable	-	NULL	Optional description or reason provided for blocking.

Table 4.4.6: Data Dictionary - UserBlockedList Entity

TaskCategory

Field Name	Data Type	Size/Constraint	Key	Default	Description
task_cat_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each task category.

type	String	Enum: {System, User}	-	User	Indicates whether the category was system pre-created (System) or created by a member (User).
name	String	VARCHAR(100), Not Null	-	-	Name of the category (e.g., Work, Study, Fitness, etc.).
created_at	DateTime	Not Null	-	Current timestamp	The date and time when the category was created.
status	Integer	Enum: {0=Inactive, 1=Active}	-	Active	Current status of the task category.

Table 4.4.7: Data Dictionary - TaskCategory Entity

Task

Field Name	Data Type	Size/Constraint	Key	Default	Description
task_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each task.
task_cat_id	String	UUID (36), Nullable	FK	NULL	References TaskCategory.task_cat_id. Defines which category the task belongs to.
member_id	String	UUID (36), Not Null	FK	—	References Member.member_id. The member who created/owns the task.

title	String	VARCHAR(150), Not Null	—	—	Title or short name of the task.
description	Text	Nullable	—	NULL	Detailed description of the task.
priority	Integer	Enum: {1=High, 2=Medium, 3=Low}	—	Medium	Importance level of the task.
planned_start_at	DateTime	Nullable	—	NULL	Planned start date/time of the task.
planned_end_at	DateTime	Nullable	—	NULL	Planned deadline/end date of the task.
actual_start_at	DateTime	Nullable	—	NULL	Actual start time of the task (when user begins).
actual_end_at	DateTime	Nullable	—	NULL	Actual completion time of the task.
progress_percent	Integer	Range 0–100	—	0	Percentage progress of the task.
reminder_time	DateTime	Nullable	—	NULL	Date/time when a reminder notification should be sent.
recurring_pattern	String	Nullable (e.g., Daily, Weekly, Monthly)	—	NULL	Pattern definition for recurrence (if is_recurring = true).
links	Array of String	Nullable	-	NULL	List of related links or URLs associated

					with the task.
status	Integer	Enum: {0=Cancelled, 1=Pending, 2=Ongoing, 3=Completed, 4=Overdue}	—	Pending	Current lifecycle state of the task.
created_at	DateTime	Not Null	—	Current timestamp	When the task was created.
updated_at	DateTime	Not Null	—	Current timestamp (auto-update)	When the task record was last modified.

Table 4.4.8: Data Dictionary - Task Entity

Subtask

Field Name	Data Type	Size/Constraint	Key	Default	Description
subtask_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each subtask.
task_id	String	UUID (36), Not Null	FK	-	References Task.task_id. Defines the parent task this subtask belongs to.
title	String	VARCHAR(150), Not Null	-	-	Title or short name of the subtask.
description	Text	Nullable	-	NULL	Detailed description of the subtask.
planned_start_at	DateTime	Nullable	-	NULL	Planned start date/time of the subtask.
planned_end_at	DateTime	Nullable	-	NULL	Planned deadline/end

					date of the subtask.
actual_start_at	DateTime	Nullable	-	NULL	Actual start time of the subtask.
actual_end_at	DateTime	Nullable	-	NULL	Actual completion time of the subtask.
order_index	Integer	>= 0	-	0	Defines the display or execution order of subtasks within a task.
created_at	DateTime	Not Null	-	Current timestamp	Timestamp when the subtask was created.
status	Integer	Enum: {0=Pending, 1=Ongoing, 2=Completed, 3=Overdue}	-	Pending	Current lifecycle state of the subtask.

Table 4.4.9: Data Dictionary - Subtask Entity

TaskAttachment

Field Name	Data Type	Size/Constraint	Key	Default	Description
task_att_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each task attachment.
task_id	String	UUID (36), Not Null	FK	-	References Task.task_id. Defines the task this attachment belongs to.
media_type	String	Enum: {Image, Video}	-	-	Type of media

					attached to the task.
media_url	String	VARCHAR(255), Not Null	-	-	URL where the attachment is stored.
file_name	String	VARCHAR(150), Not Null	-	-	Filename of the uploaded attachment.
file_size	Integer	Size in KB, >=0	-	0	Size of the attachment file.

Table 4.4.10: Data Dictionary - TaskAttachment Entity

TaskScheduleLog

Field Name	Data Type	Size/Constraint	Key	Default	Description
log_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each schedule change log entry.
task_id	String	UUID (36), Not Null	FK	-	References Task.task_id. Identifies the task that this log entry belongs to.
subtask_id	String	UUID (36), Not Null	FK	-	References Subtask.subtask_id when the change belongs to a subtask; NULL otherwise.
entity_type	String	Enum: {"task", "subtask"}, Not Null	-	-	Indicates whether the schedule changed applies to a task or a subtask

from_start_at	DateTime	Not Null	-	-	Previous scheduled start time before the change occurred.
to_start_at	DateTime	Not Null	-	-	Updated scheduled start time after the change.
from_end_at	DateTime	Not Null	-	-	Previous scheduled start time before the change.
to_end_at	DateTime	Not Null	-	-	Updated scheduled end time after the change.
source	String	Enum: {"user", "system_ai"}, Not Null	-	user	Indicates who triggered the schedule change (user or AI system).
reason	String	Nullable (Text)	-	NULL	Optional explanation or justification for the schedule adjustment.
timestamp	DateTime	Not Null	-	Current timestamp	The date and time when the rescheduling event occurred.

Table 4.4.11: Data Dictionary - TaskScheduleLog Entity

TaskAdvisorLog

Field Name	Data Type	Size/Constraint	Key	Default	Description

advisor_log_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each advisor log entry.
task_id	String	UUID (36), Not Null	FK	-	References Task.task_id. The task that the suggestion is related to.
subtask_id	String	UUID (36), Not Null	FK	-	References Subtask.subtask_id when the suggestion is for a subtask.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. Represents the user receiving the advisor suggestion
entity_type	String	Enum: {"task", "subtask"}, Not Null	-	task	Indicates whether the advisor suggestion relates to a task or a subtask
should_reschedule	Boolean	Not Null	-	-	Indicates whether the AI predicts that the task's schedule should be adjusted.
features	JSON	Not Null	-	{}	Dictionary of input features used by the AI model to generate the suggestion.

model	String	VARCHAR (100), Not Null	-	-	Name or identifier of the AI model that generated the suggestion.
original_start_at	DateTime	Nullable	-	NULL	Previously scheduled start time before the advisor-generated update.
original_end_at	DateTime	Nullable	-	NULL	Previously scheduled end time before the advisor-generated update.
suggested_start_at	DateTime	Nullable	-	NULL	AI-suggested start time after analyzing schedule conflicts or productivity patterns.
suggested_end_at	DateTime	Nullable	-	NULL	AI-suggested end time for the task or subtask
reason	String	Nullable (Text)	-	NULL	Explanation of why the AI suggested the change.
source	String	Enum: {"ai_optimizer", "manual", "calendar_conflict"}, Not Null	-	ai_optimizer	Origin of the suggestion.
created_at	DateTime	Not Null	-	Current timestamp	Timestamp when the advisor log entry was created.

user_action	String	Enum: {"pending", "accepted", "rejected", "ignored"}, Not Null	-	pending	Indicates how the user responded to the advisor's suggestion.
user_action_at	DateTime	Nullable	-	NULL	Timestamp of the user's response to the suggestion.

Table 4.4.12: Data Dictionary - TaskAdvisorLog Entity

AppCategory

Field Name	Data Type	Size/Constraint	Key	Default	Description
app_category_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each app category.
category_name	String	100, Not Null, Unique	-	-	Name of the category (e.g., Productivity, Education, Health).
category_desc	Text	Nullable	-	NULL	Description of the category, explaining its purpose or scope.

Table 4.4.13: Data Dictionary - AppCategory Entity

AppLimitRule

Field Name	Data Type	Size/Constraint	Key	Default	Description
rule_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each app usage limit rule.
member_id	String	UUID (36)	FK	-	References the Member

					who owns this rule.
app_category_id	String	UUID (36)	FK	-	References the AppCategory this limit applies to.
daily_limit_min	Integer	>= 0	-	0	Maximum daily allowed usage (in minutes) for apps in this category.
created_at	DateTime	Not Null	-	Current time	Timestamp when the rule was created.
updated_at	DateTime	Not Null	-	Auto-updated	Timestamp when the rule was last modified.

Table 4.4.14: Data Dictionary - AppLimitRule Entity

Application

Field Name	Data Type	Size/Constraint	Key	Default	Description
app_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each application record.
app_category_id	String	UUID (36)	FK	-	References AppCategory that this application belongs to.
package_name	String	150	Unique	-	System package identifier (e.g., com.whatsapp, org.mozilla.firefox).

app_name	String	100	-	-	Display name of the application (e.g., WhatsApp, Chrome).
icon_url	String	255	-	-	URL or file path to the app's icon.
default_limit_min	Integer	≥ 0	-	0	Default recommended daily limit (in minutes).

Table 4.4.15: Data Dictionary - Application Entity

UserApp

Field Name	Data Type	Size/Constraint	Key	Default	Description
user_app_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each user-application record.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. Identifies the member who installed or uses the application.
app_id	String	UUID (36), Not Null	FK		References Application.app_id. Indicates which third-party app or integrated service is being tracked.
installed	Boolean	Not Null	-	True	Indicates whether the app is currently

					installed by the member.
last_seen	DateTime	Not Null	-	Current timestamp	Timestamp of the most recent time the user interacted with or opened the app.

Table 4.4.16: Data Dictionary - UserApp Entity

TimeUsageSession

Field Name	Data Type	Size/Constraint	Key	Default	Description
usage_session_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each usage session.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. The member performing the activity.
app_id	String	UUID (36), Nullable	FK	NULL	References Application.app_id. Optional, used if session involves app usage.
task_id	String	UUID (36), Nullable	FK	NULL	References Task.task_id. Optional, used if session involves task focus.
activity_type	Enum	{AppUsage, TaskFocus, Other}	-	-	Type of activity being tracked in this session.

start_at	DateTime	Not Null	-	-	Start timestamp of the session.
end_at	DateTime	Nullable	-	NULL	End timestamp of the session. Can be NULL if session is ongoing.
duration_min	Integer	>= 0	-	0	Duration of the session in minutes. Can be auto-calculated from end_at – start_at.
source	String	VARCHAR(50)	-	Device/App	Source of the activity data (e.g., MobileApp, Task, NFCTrigger).
created_at	DateTime	Not Null	-	Current timestamp	Timestamp when the session record was created.
updated_at	DateTime	Not Null	-	Auto-updated	Timestamp when the session record was last updated.

Table 4.4.17: Data Dictionary - TimeUsageSession Entity

WhitelistedApp

Field Name	Data Type	Size/Constraint	Key	Default	Description
whitelist_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each whitelist entry.

member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. The member who set the whitelist.
app_id	String	UUID (36), Not Null	FK	-	References Application.app_id. The whitelisted application.
date_added	DateTime	Not Null	-	Current timestamp	Date and time when the app was whitelisted.

Table 4.4.18: Data Dictionary - WhiteListedApp Entity

NFCTag

Field Name	Data Type	Size/Constraint	Key	Default	Description
nfc_tag_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each NFC tag record.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. The owner of the NFC tag.
task_id	String	UUID (36), Nullable	FK	-	References Task.task_id. Task linked to this NFC tag (optional if tied directly to a subtask).
subtask_id	String	UUID (36), Nullable	FK	-	References Subtask.subtask_id. Subtask linked to this NFC tag.

tag_code	String	255, Unique, Not Null	-	-	The physical NFC tag's unique code/UID read from the chip.
label	String	100	-	-	User-friendly label given to the tag (e.g., "Study Desk Tag").
purpose	String	50	-	-	Describes the tag's intended use (e.g., Focus, TaskStart, TaskEnd, Reminder).
created_at	DateTime	Not Null	-	Current timestamp	The time the NFC tag was registered in the system.
status	Integer	Enum: {0=Inactive, 1=Active}	-	1	Current availability/usage state of the tag.

Table 4.4.19: Data Dictionary - NFCTag Entity

Community

Field Name	Data Type	Size/Constraint	Key	Default	Description
community_id	String	UUID (36)	PK	Auto-generated	Unique identifier for the community.
created_by	String	UUID (36), Not Null	FK	-	References Member.member_id. Identifies the member who created the community.
name	String	150, Not Null, Unique	-	-	Display name of the community.

description	Text	-	-	-	Brief overview or details about the community's purpose.
is_private	Boolean	Not Null	-	false	Defines if the community is private (invite/join request only) or public.
member_count	Integer	>= 1, Not Null	-	1	Total number of members in the community (starts at 1 for the creator).
created_at	DateTime	Not Null	-	Current timestamp	Timestamp of when the community was created.
status	Integer	Enum: {0 = inactive, 1 = active}	-	1	Current status of the community.
profile_image_url	String	255	-	NULL	URL for the community's profile picture or banner.
tags	String	-	-	NULL	List of tags/keywords associated with the community.

Table 4.4.20: Data Dictionary - Community Entity

CommunityJoinRequest

Field Name	Data Type	Size/Constraint	Key	Default	Description
join_request_id	String	UUID (36)	PK	Auto-generated	Unique identifier for

					each community join request.
community_id	String	UUID (36), Not Null	FK	-	References Community.community_id. The community the member wants to join.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. The member who submitted the join request.
reviewed_by	String	UUID (36), Not Null	FK	NULL	References Member.member_id. The admin who reviewed the join request.
status	Integer	Enum: {1=Pending, 2=Approved, 3=Rejected}, Not Null	-	1	Current review status of the join request.
requested_at	DateTime	Not Null	-	Current timestamp	Timestamp when the join request was submitted.
reviewed_at	DateTime	Nullable	-	NULL	Timestamp when the request was reviewed (approved or rejected).

Table 4.4.21: Data Dictionary - CommunityJoinRequest Entity

CommunityMember

Field Name	Data Type	Size/Constraint	Key	Default	Description
community_member_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each community membership record.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. The member who joined the community.
community_id	String	UUID (36), Not Null	FK	-	References Community.community_id. The community this member belongs to.
role	Integer	Enum: {1=member, 2=creator}, Not Null	-	-	Role of the member in the community, referencing their authority level.
subscription	Boolean	Not Null	-	true	Whether the member is subscribed to community updates/notifications.

Table 4.4.22: Data Dictionary - CommunityMember Entity

Post

Field Name	Data Type	Size/Constraint	Key	Default	Description
post_id	String	UUID (36)	PK	Auto-generated	Unique identifier for the post.

community_member_id	String	UUID (36), Not Null	FK	-	References CommunityMember.community_member_id. Identifies the member who created the post.
community_id	String	UUID (36), Not Null	FK	-	References Community.community_id. Identifies the community where the post belongs.
title	String	200, Not Null	-	-	Title or headline of the post.
content	Text	Not Null	-	-	Main body/content of the post.
created_at	DateTime	Not Null	-	Current timestamp	Timestamp when the post was created.
status	Integer	Enum: {0 = inactive, 1 = active}, Not Null	-	1	Current status of the post.
tags	Array of String	Nullable	-	Empty list ([])	List of tag names used for search and filtering.
removed_at	DateTime	Nullable	-	NULL	Timestamp indicating when the post was soft-deleted or removed by an admin.
removed_reason	String	Nullable (Text)	-	NULL	Explanation provided by an admin stating why

					the post was removed.

Table 4.4.23: Data Dictionary - Post Entity

PostAttachment

Field Name	Data Type	Size/Constraint	Key	Default	Description
post_att_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each post attachment.
post_id	String	UUID (36), Not Null	FK	-	References Post.post_id. The post that this attachment belongs to.
media_type	String	50, Not Null	-	-	Type of the media (e.g., image, video).
media_url	String	255, Not Null	-	-	URL to the uploaded media file.
file_name	String	150, Not Null	-	-	File name of the attachment.
file_size	Integer	Size in KB, Not Null	-	-	Size of the file in bytes.

Table 4.4.24: Data Dictionary - PostAttachment Entity

PostReaction

Field Name	Data Type	Size/Constraint	Key	Default	Description
reaction_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each reaction.
post_id	String	UUID (36), Not Null	FK	-	References Post.post_id. The post that

					received the reaction.
community_member_id	String	UUID (36), Not Null	FK	-	References CommunityMember.community_member_id. The member who reacted to the post.
emoji_type	String	50, Not Null	-	-	Type of reaction/emoji used (e.g., like, love, laugh, sad, angry).
reacted_at	DateTime	Not Null	-	Current timestamp	Timestamp when the reaction was made.

Table 4.4.25: Data Dictionary - PostReaction Entity

Comment

Field Name	Data Type	Size/Constraint	Key	Default	Description
comment_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each comment.
post_id	String	UUID (36), Not Null	FK	-	References Post.post_id. The post that this comment belongs to.
community_member_id	String	UUID (36), Not Null	FK	-	References CommunityMember.community_member_id. The member who made the comment.

reply_to_id	String	UUID (36), Nullable	FK	NULL	References Comment.comment_id. If the comment is a reply, this points to the parent comment.
content	Text	Not Null	-	-	Text content of the comment.
created_at	DateTime	Not Null	-	Current timestamp	Timestamp when the comment was created.

Table 4.4.26: Data Dictionary - Comment Entity

ForumQuestion

Field Name	Data Type	Size/Constraint	Key	Default	Description
question_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each forum question.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. Identifies the member who posted the forum question.
best_parent_answer_id	String	UUID (36), Nullable	FK	NULL	References ForumAnswer.answer_id. Stores the ID of the accepted/best answer for this question.
title	String	255, Not Null	-	-	Title of the forum question.

content	Text	Not Null	-	-	Detailed description of the forum question.
tags	String	255, Nullable	-	-	Comma-separated tags for categorization of the question.
created_at	DateTime	Not Null	-	Current timestamp	When the question was created.
is_resolved	Boolean	Not Null	-	false	Indicates whether the question has been resolved.
status	Integer	Enum: {0=Inactive, 1=Active}, Not Null	-	1	Indicates the current lifecycle state of the forum question.
removed_at	DateTime	Nullable	-	NULL	Timestamp when the question was removed or soft-deleted by a moderator.
removed_reason	String	Nullable (Text)	-	NULL	Reason provided by a moderator for removing the question.

Table 4.4.27: Data Dictionary - ForumQuestion Entity

ForumAnswer

Field Name	Data Type	Size/Constraint	Key	Default	Description
answer_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each forum answer.

parent_answer_id	String	UUID (36), Not Null	FK	NULL	References ForumAnswer.answer_id when this answer is a reply to another answer.
question_id	String	UUID (36), Not Null	FK	-	References ForumQuestion.question_id. The question this answer belongs to.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. Identifies the member who posted the forum answer.
content	Text	Not Null	-	-	The textual content of the forum answer.
created_at	DateTime	Not Null	-	Current timestamp	When the answer was created.
upvote_count	Integer	Not Null	-	0	Number of upvotes the answer has received.
downvote_count	Integer	Not Null	-	0	Number of downvotes the answer has received.
is_accepted	Boolean	Not Null	-	false	Indicates whether the answer has been accepted as the solution.

Table 4.4.28: Data Dictionary - ForumAnswer Entity

ForumAttachment

Field Name	Data Type	Size/Constraint	Key	Default	Description
forum_att_id	String	UUID (36)	PK	Auto-generated	Unique identifier for the forum attachment.
question_id	String	UUID (36), Nullable	FK	-	References ForumQuestion.question_id. Required if the attachment belongs to a question.
answer_id	String	UUID (36), Nullable	FK	-	References ForumAnswer.answer_id. Required if the attachment belongs to an answer.
media_type	String	Enum: {image, video}, Not Null	-	-	The type of media file attached.
media_url	String	255, Not Null	-	-	URL of the attachment.
file_name	String	255	-	NULL	File name of the uploaded attachment.
file_size	Integer	≥ 0	-	NULL	File size in bytes (for validation or limits).

Table 4.4.29: Data Dictionary - ForumAttachment Entity

ForumReaction

Field Name	Data Type	Size/Constraint	Key	Default	Description
forum_reaction_id	String	UUID (36)	PK	Auto-generated	Unique identifier for the reaction.

question_id	String	UUID (36), Nullable	FK	-	References ForumQuestion.question_id.
answer_id	String	UUID (36), Nullable	FK	-	References ForumAnswer.answer_id.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id. Identifies the member who voted to this forum question.
type	Integer	Enum: {0=Downvote, 1=Upvote}	-	-	Reaction type.
created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Timestamp when the reaction was added.

Table 4.4.30: Data Dictionary - ForumReaction Entity

ChatRoom

Field Name	Data Type	Size/Constraint	Key	Default	Description
chat_room_id	String	UUID (36)	PK	Auto-generated	Unique identifier for the chat room.
created_by_id	String	UUID (36), Not Null	FK	-	References ChatRoomMember.chat_room_member_id (the creator participant).
type	Integer	Enum: {1=Direct, 2=Group}	-	-	Defines the type of chat room.

name	String	255, Nullable	-	NULL	Room name (only required for group/community chats).
group_image_url	String	VARCHAR (255), Nullable	-	NULL	URL of the group chat's image/avatar. Only applicable for group chat rooms.
is_anonymous	Boolean	Not Null	-	false	Indicates whether this chat room hides member identities.
created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Timestamp when the chat room was created.

Table 4.4.31: Data Dictionary - ChatRoom Entity

ChatRoomMember

Field Name	Data Type	Size/Constraint	Key	Default	Description
chat_room_member_id	String	UUID (36)	PK	Auto-generated	Unique identifier for a chat room participant.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id (the actual user).
chat_room_id	String	UUID (36), Not Null	FK	-	References ChatRoom.chat_room_id.
joined_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Timestamp when the member joined the chat room.

is_admin	Boolean	Not Null	-	FALSE	Indicates if the member has admin privileges in the chat room.
is_anonymous	Boolean	Not Null	-	FALSE	Defines whether the member is visible by identity or hidden (e.g., anonymous posting/chat).

Table 4.4.32: Data Dictionary - ChatRoomMember Entity

ChatMessage

Field Name	Data Type	Size/Constraint	Key	Default	Description
message_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each chat message.
chat_room_id	String	UUID (36), Not Null	FK	-	References ChatRoom.chat_room_id, links the message to a chat room.
sender_id	String	UUID (36), Not Null	FK	-	References ChatRoomMember.chat_room_member_id, ensures only valid room members can send messages.
reply_to_id	String	UUID (36), Nullable	FK (self-reference)	-	References ChatMessage.message_id, allows threaded replies.

content	Text	Not Null	-	-	Message body text (supports plain text, markdown, or rich text depending on design).
sent_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Timestamp when the message was sent.

Table 4.4.33: Data Dictionary - ChatMessage Entity

MessageAttachment

Field Name	Data Type	Size/Constraint	Key	Default	Description
msg_att_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each attachment.
message_id	String	UUID (36), Not Null	FK	-	References ChatMessage.message_id, links the attachment to a specific message.
post_id	String	UUID (36), Nullable	FK	-	References Post.post_id, links the attachment to a specific community post.
task_id	String	UUID (36), Nullable	FK	-	References Task.task_id, links the attachment to a specific personal task.
diary_id	String	UUID (36), Nullable	FK	-	References EmotionDiary.diary_id, links the

					attachment to a specific emotion diary note.
media_type	Enum	{image, video}	-	-	Type of media uploaded.
media_url	String	255, Not Null	-	-	Storage location (cloud URL or local path).
media_thumbnail_url	String	VARCHAR (255), Nullable	-	NULL	URL of the thumbnail image generated for media files.
file_name	String	255, Nullable	-	-	Original name of the file uploaded.
file_size	Integer	≥ 0	-	NULL	File size in bytes (for validation or limits).
extra_data	JSON	Nullable	-	NULL	Optional structured data used for non-media attachments.

Table 4.4.34: Data Dictionary - MessageAttachment Entity

FocusSession

Field Name	Data Type	Size/Constraint	Key	Default	Description
focus_session_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each focus session.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id, identifies the user.

task_id	String	UUID (36), Nullable	FK	-	References Task.task_id, links to a main task.
subtask_id	String	UUID (36), Nullable	FK	-	References Subtask.subtask_id, links to a subtask (optional).
nfc_tag_id	String	UUID (36), Nullable	FK	-	References NFCTag.nfc_tag_id, indicates if session was tied to a tag.
start_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	When the session started.
end_at	DateTime	Nullable	-	NULL	When the session ended (NULL if ongoing).
duration	Integer	In minutes	-	-	Total focus time, derived as end_at - start_at.
nfc_triggered	Boolean	{true, false}	-	false	Whether session started via NFC tag.
interruption_count	Integer	Default 0	-	0	Number of interruptions recorded.
status	Integer	Enum: {0=cancelled, 1=active, 2=ongoing, 3=completed}	-	0	Session lifecycle state.
point_earned	Integer	Default 0	-	0	Reward points earned for completing the session.

Table 4.4.35: Data Dictionary - FocusSession Entity

Reward

Field Name	Data Type	Size/Constraint	Key	Default	Description
reward_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each reward.
name	String	150 chars, Not Null	-	-	Display name of the reward.
type	String	50 chars, Not Null	-	-	Category of reward (e.g., voucher, badge, coupon, gift).
description	Text	Optional	-	NULL	Detailed explanation of the reward.
icon_url	String	255 chars, Nullable	-	NULL	URL for reward icon image.
point_required	Integer	Not Null	-	-	Number of points a member must redeem to claim this reward.
created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time the reward was created.
status	Integer	Enum: {0=inactive, 1=active}	-	1	Indicates reward availability.

Table 4.4.36: Data Dictionary - Reward Entity

RewardRedemption

Field Name	Data Type	Size/Constraint	Key	Default	Description
redeem_id	String	UUID (36)	PK	Auto-generated	Unique identifier for

					each reward redemption record.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id; identifies the member who redeemed the reward.
reward_id	String	UUID (36), Not Null	FK	-	References Reward.reward_id; specifies which reward was redeemed.
redeem_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time when the reward was redeemed.
point_spent	Integer	Not Null	-	0	Number of points spent for this redemption.
point_before	Integer	Not Null	-	0	Member's point balance before redemption.
point_after	Integer	Not Null	-	0	Member's point balance after redemption.
created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Timestamp of when the redemption record was created in the system.

Table 4.4.37: Data Dictionary - RewardRedemption Entity

Leaderboard

Field Name	Data Type	Size/Constraint	Key	Default	Description
leaderboard_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each leaderboard record.
member_id	String	UUID (36), Not Null	FK	-	References Member.member_id; identifies the member participating in the leaderboard.
total_focus_duration	Integer	Not Null	-	0	Total accumulated focus duration (in minutes) by the member.
rank	Integer	Not Null	-	-	Current ranking position of the member based on focus duration.
last_updated_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time when the leaderboard record was last updated.

Table 4.4.38: Data Dictionary - Leaderboard Entity

EmotionDiary

Field Name	Data Type	Size/Constraint	Key	Default	Description
diary_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each diary entry.
member_id	String	UUID (36), Not Null	FK	-	References Member.me

					mber_id; the member who created the diary entry.
emotion_tag _id	String	UUID (36), Nullable	FK	NULL	References EmotionTag. emotion_tag _id; identifies the tagged emotion for this diary entry.
title	String	50 chars, Not Null	-	-	Title of the diary entry.
content	Text	Optional	-	NULL	Main content or description of the diary entry.
mood_level	Integer	Range: 1–5, Not Null	-	5	Numerical scale representing the intensity of mood (e.g., 1 = very low, 5 = very high).
context_tags	String	255 chars, Nullable	-	NULL	Comma-sepa rated tags or keywords describing context (e.g., “work, family, exam”).
sentiment_sc ore	Float	Range: -1.0 to 1.0, Nullable	-	NULL	Sentiment analysis score derived from content (-1 = negative, 0 = neutral, 1 = positive).
created_at	DateTime	Not Null	-	CURRENT_ TIMESTAM P	Date and time the

					diary entry was created.
is_momentary	Boolean	{true, false}	-	false	Indicates whether the diary entry was a quick, momentary note.
is_protected	Boolean	{true, false}	-	false	Marks the diary entry as private/protected (not shared).
status	Integer	Enum: {0 = inactive, 1 = active}	-	1	Status of the diary entry.

Table 4.4.39: Data Dictionary - EmotionDiary Entity

EmotionTag

Field Name	Data Type	Size/Constraint	Key	Default	Description
emotion_tag_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each emotion tag.
name	String	100 chars, Not Null	-	-	Display name of the emotion (e.g., Happy, Sad, Angry).
color_code	String	7 chars, Not Null	-	-	Hexadecimal color code (e.g., #FFD700) associated with the emotion for UI visualization.
category	String	50 chars, Nullable	-	NULL	Emotion category/group (e.g., Positive,

					Negative, Neutral).

Table 4.4.40: Data Dictionary - EmotionTag Entity

DiaryAttachment

Field Name	Data Type	Size/Constraint	Key	Default	Description
diary_att_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each diary attachment.
diary_id	String	UUID (36), Not Null	FK	-	References the EmotionDiary entry this attachment belongs to.
media_type	String	{image, video]	-	-	Type of media
media_url	String	255 chars, Nullable	-	NULL	Storage URL or file path to the uploaded media file.
file_name	String	150 chars, Nullable	-	NULL	Original name of the uploaded file.
file_size	Integer	Size in KB, Nullable	-	NULL	File size of the attachment.

Table 4.4.41: Data Dictionary - DiaryAttachment Entity

AdminReview

Field Name	Data Type	Size/Constraint	Key	Default	Description
review_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each admin review record.

community_id	String	UUID (36), Nullable	FK	NULL	References the Community where the flagged content was posted (if applicable).
question_id	String	UUID (36), Nullable	FK	NULL	References ForumQuestion.question_id. Identifies the forum question that triggered the admin review.
post_id	String	UUID (36), Nullable	FK	NULL	References the Community Post that triggered the review.
message_id	String	UUID (36), Nullable	FK	NULL	References the Message that was flagged for review.
member_id	String	UUID (36), Nullable	FK	NULL	References the Member who created the flagged content.
admin_id	String	UUID (36), Not Null	FK	-	References the Admin who performed the review action.
content_type	String	50 chars, Not Null	-	-	Type of content under review (e.g., post, message, forum_answer).

trigger_type	String	50 chars, Not Null	-	-	Reason for triggering review (e.g., report, auto-flag, manual-check).
review_decision	Integer	Enum: {0=pending, 1=approved, 2=removed}	-	0	Decision outcome of the review.
admin_notes	Text	Optional	-	NULL	Notes or remarks provided by the reviewing admin.
auto_flagged	Boolean	{false, true}	-	-	Indicates if the content was auto-flagged by the system.
is_resolved	Boolean	{false, true}	-	false	Whether the review case has been resolved.
reported_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time when the content was reported or flagged.
reviewed_at	DateTime	Nullable	-	NULL	Date and time when the admin reviewed the case.

Table 4.4.42: Data Dictionary - AdminReview Entity

AdminActionLog

Field Name	Data Type	Size/Constraint	Key	Default	Description
action_log_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each admin

					action log entry.
admin_id	String	UUID (36), Not Null	FK	-	References the Admin who performed the action.
community_id	String	UUID (36), Nullable	FK	NULL	References the Community where the action occurred (if applicable).
question_id	String	UUID (36), Nullable	FK	NULL	References ForumQuestion.question_id. Identifies the forum question involved in the admin action.
post_id	String	UUID (36), Nullable	FK	NULL	References the Post involved in the action (if applicable).
message_id	String	UUID (36), Nullable	FK	NULL	References the Message affected by the action (if applicable).
action_type	String	50 chars, Not Null	-	-	Type of action performed (e.g., delete, warn, suspend, approve).
target_type	String	50 chars, Not Null	-	-	Type of content targeted (e.g., post, message, forum, member).

action_desc	Text	Optional	-	NULL	Detailed description or notes of the action taken.
ip_address	String	45 chars (IPv4/IPv6)	-	NULL	IP address from which the admin performed the action.
device_info	String	255 chars	-	NULL	Device details of the admin during the action.
performed_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time when the action was carried out.

Table 4.4.43: Data Dictionary - AdminActionLog Entity

Report

Field Name	Data Type	Size/Constraint	Key	Default	Description
report_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each report entry.
member_id	String	UUID (36), Not Null	FK	-	References the Member for whom the report is generated.
report_type	String	Enum: {daily, weekly, monthly}, Not Null	-	-	Specifies the reporting frequency/type.
report_notes	Text	Optional	-	NULL	Additional comments, remarks, or context about the report.

created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time when the report was generated.

Table 4.4.44: Data Dictionary - Report Entity

FocusReport

Field Name	Data Type	Size/Constraint	Key	Default	Description
focus_report_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each focus report.
report_id	String	UUID (36), Not Null	FK	-	References the Report to which this focus report belongs.
time_spent_total	Integer	Not Null, ≥0	-	0	Total time (in minutes or seconds) spent in focus sessions within the report period.
productive_ratio	Decimal	(5,2), Not Null	-	0.00	Ratio of productive vs. total focus time (expressed as percentage or decimal fraction).
average_focus_time	Integer	Not Null, ≥0	-	0	Average duration of focus sessions (in minutes or seconds).
focus_session_count	Integer	Not Null, ≥0	-	0	Total number of focus

					sessions recorded in the report period.

Table 4.4.45: Data Dictionary - FocusReport Entity

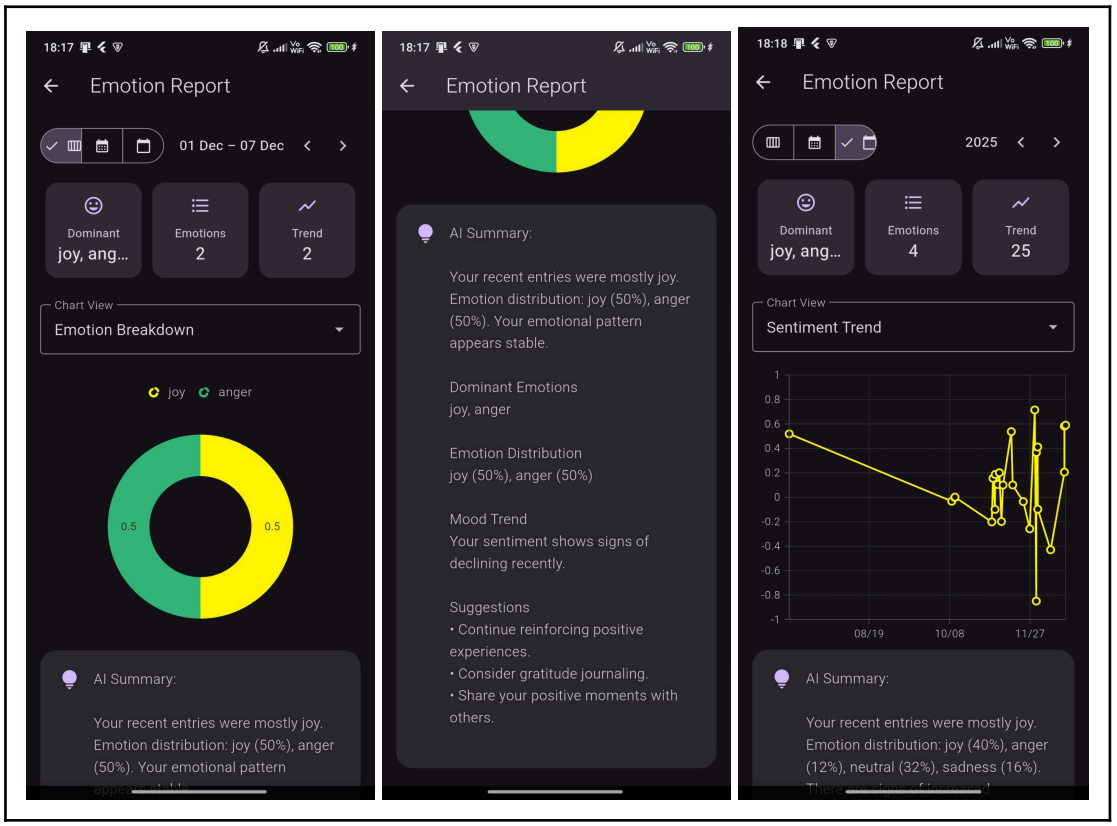
MoodReport

Field Name	Data Type	Size/Constraint	Key	Default	Description
mood_report_id	String	UUID (36)	PK	Auto-generated	Unique identifier for each mood report.
report_id	String	UUID (36), Not Null	FK	-	References the associated report record.
mood_summary	Text	Nullable	-	NULL	Summary of mood patterns or aggregated mood insights for the reporting period.
created_at	DateTime	Not Null	-	CURRENT_TIMESTAMP	Date and time the mood report was generated.

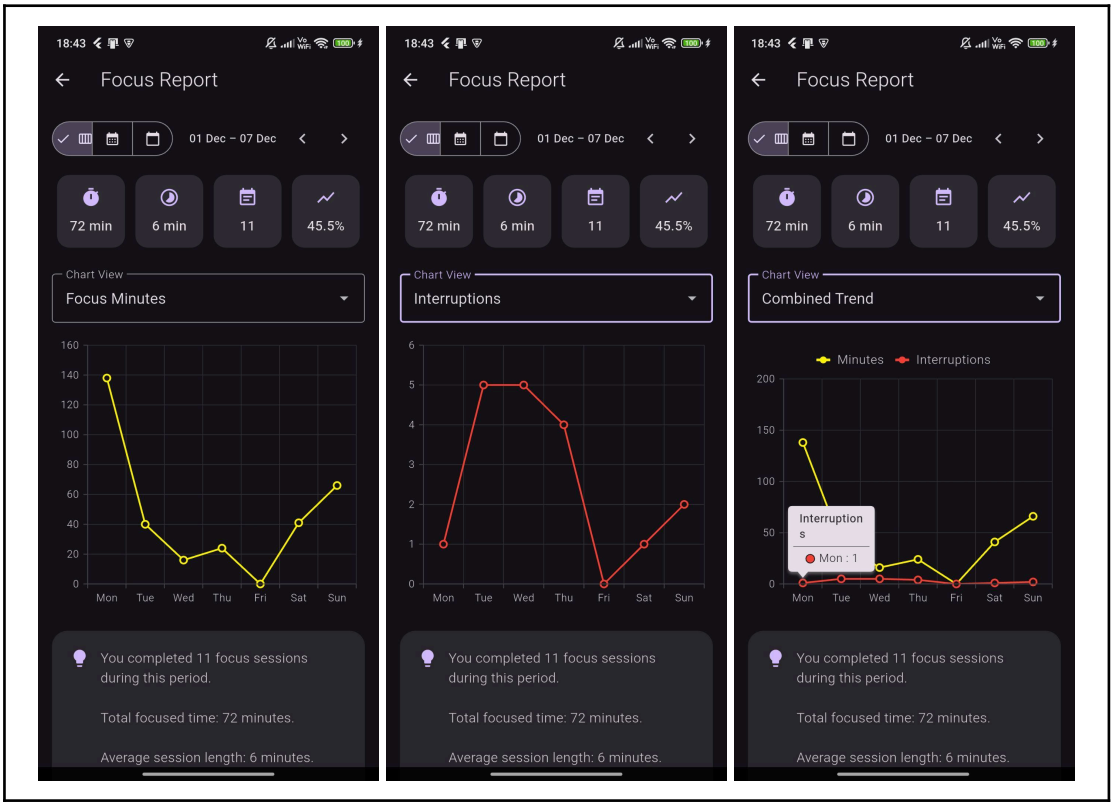
Table 4.4.46: Data Dictionary - MoodReport Entity

4.5 Reports Design

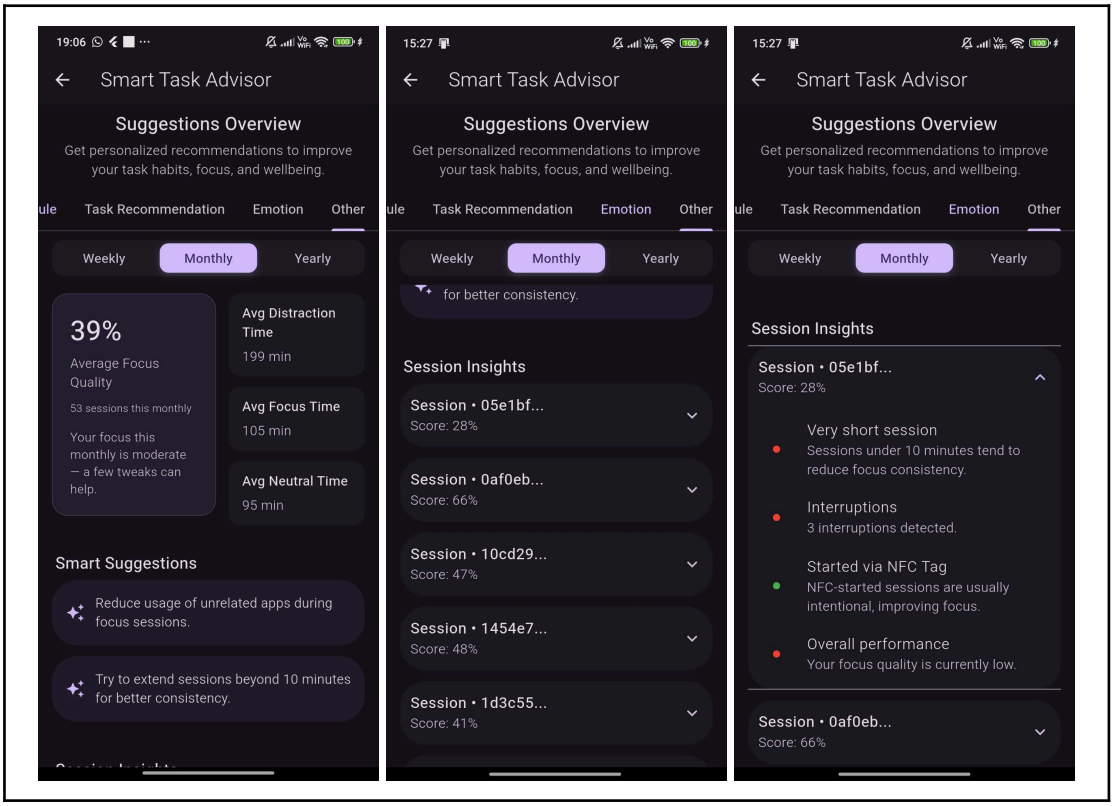
4.5.1 Emotion Summary Report



4.5.2 Focus Performance Report



4.5.3 Goal Tracking Report



4.6 Process Design

Goal Assistance Module

Create Task and Subtask

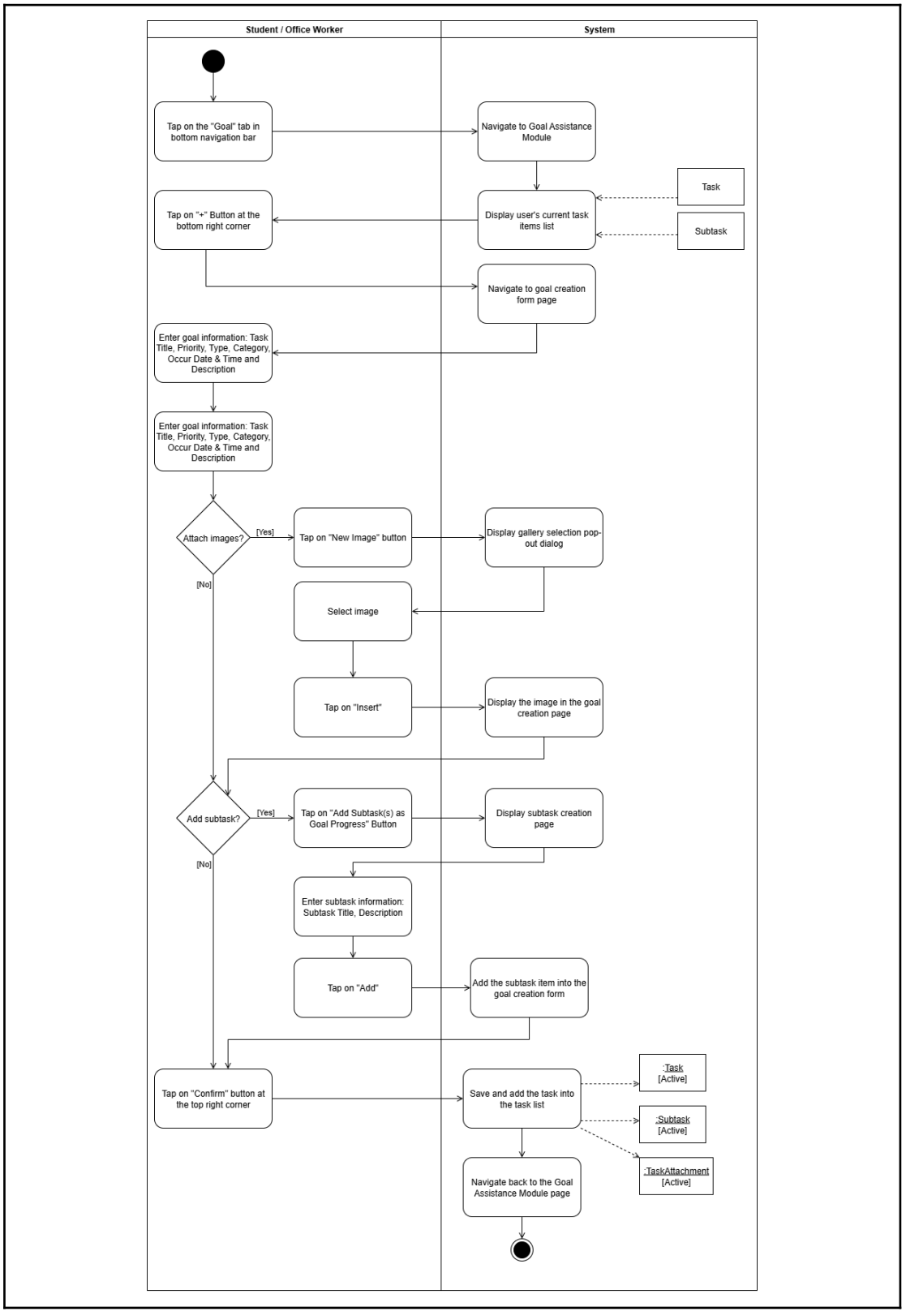


Diagram 4.6.1: Activity Diagram - Create Task and Subtask (Goal Assistance Module)

View Goal Notification

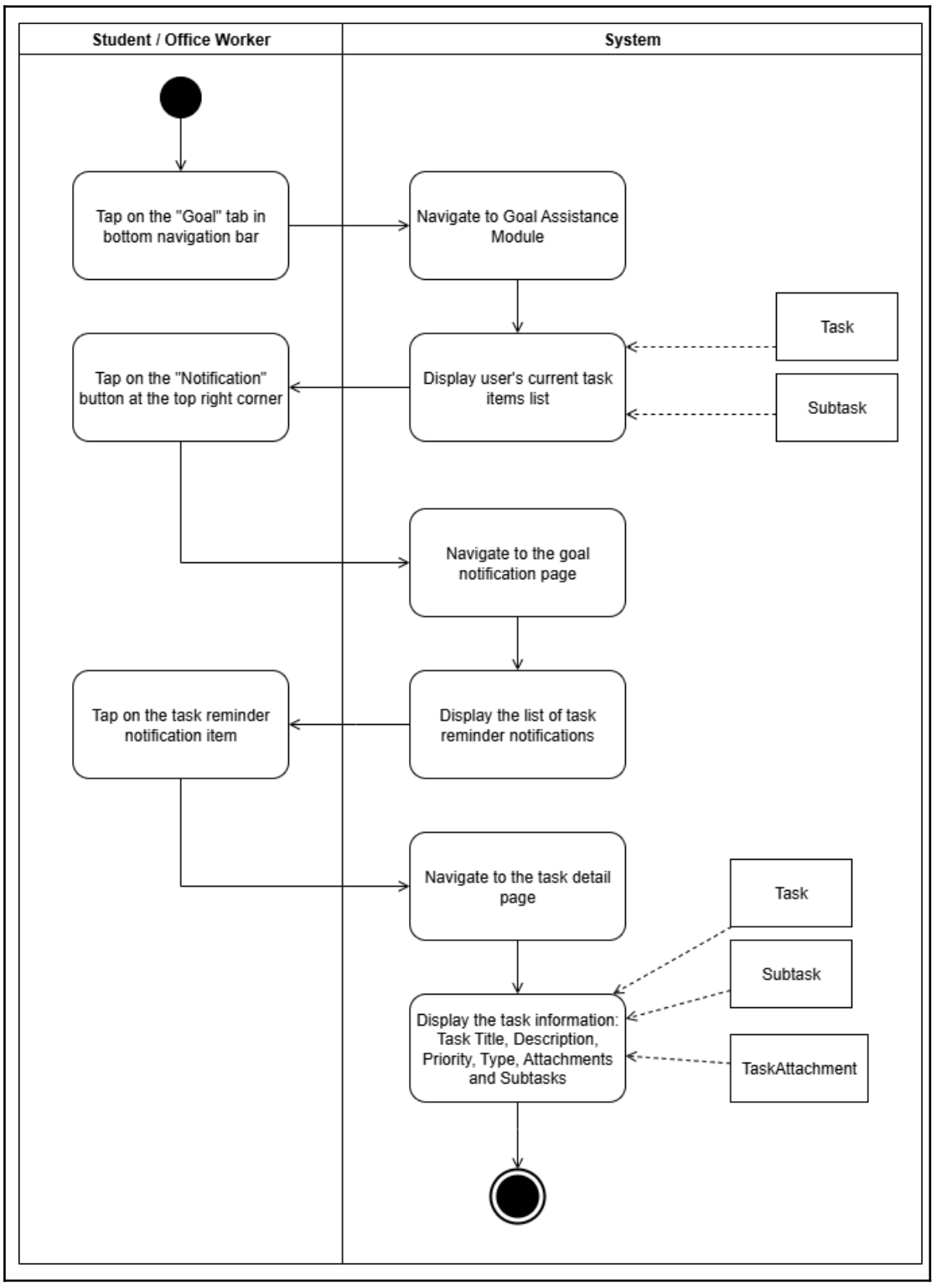


Diagram 4.6.2: Activity Diagram - View Goal Notification (Goal Assistance Module)

Manage Smart Advisor

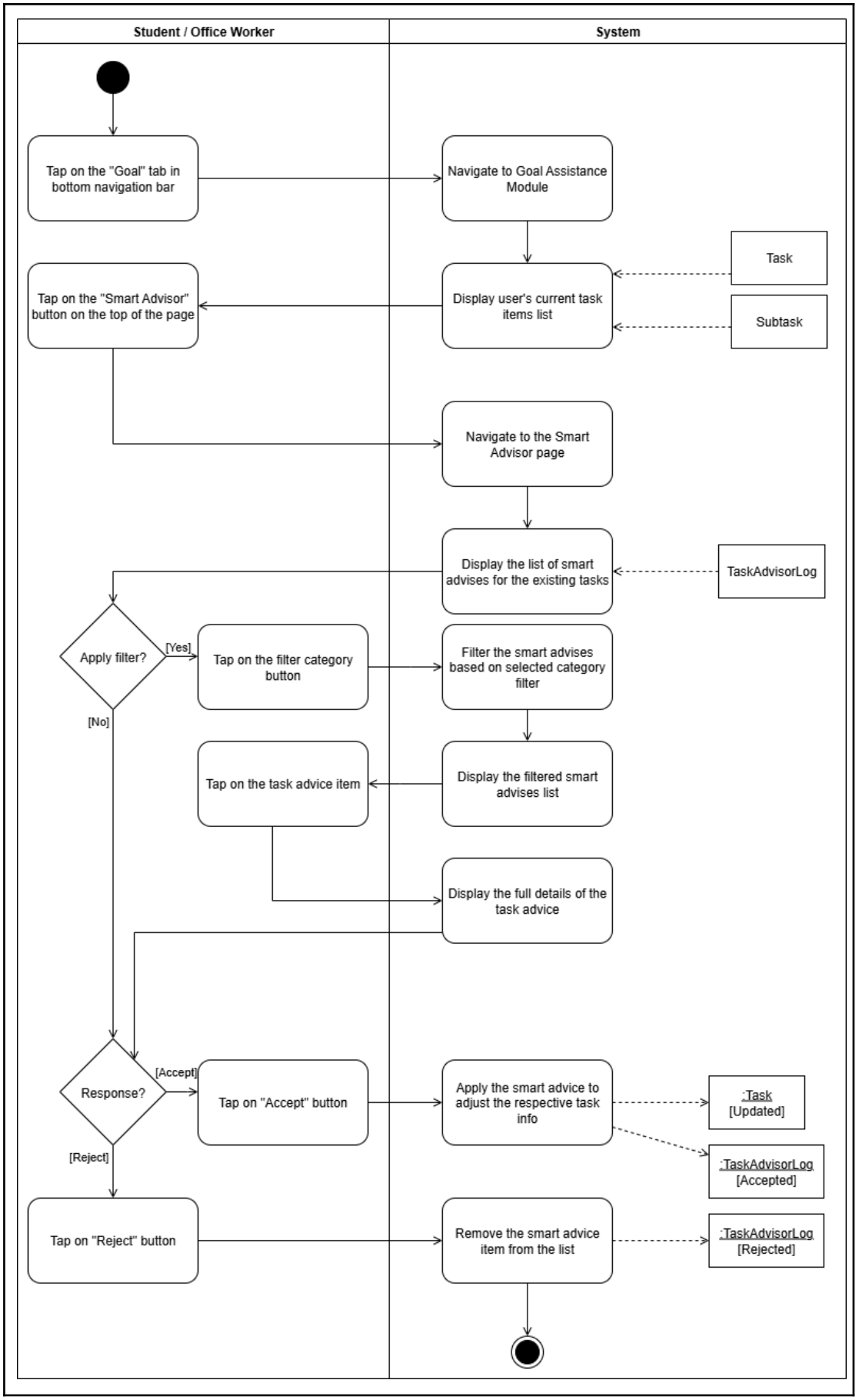


Diagram 4.6.3: Activity Diagram - Manage Smart Advisor (Goal Assistance Module)

Emotion Diary Note Module

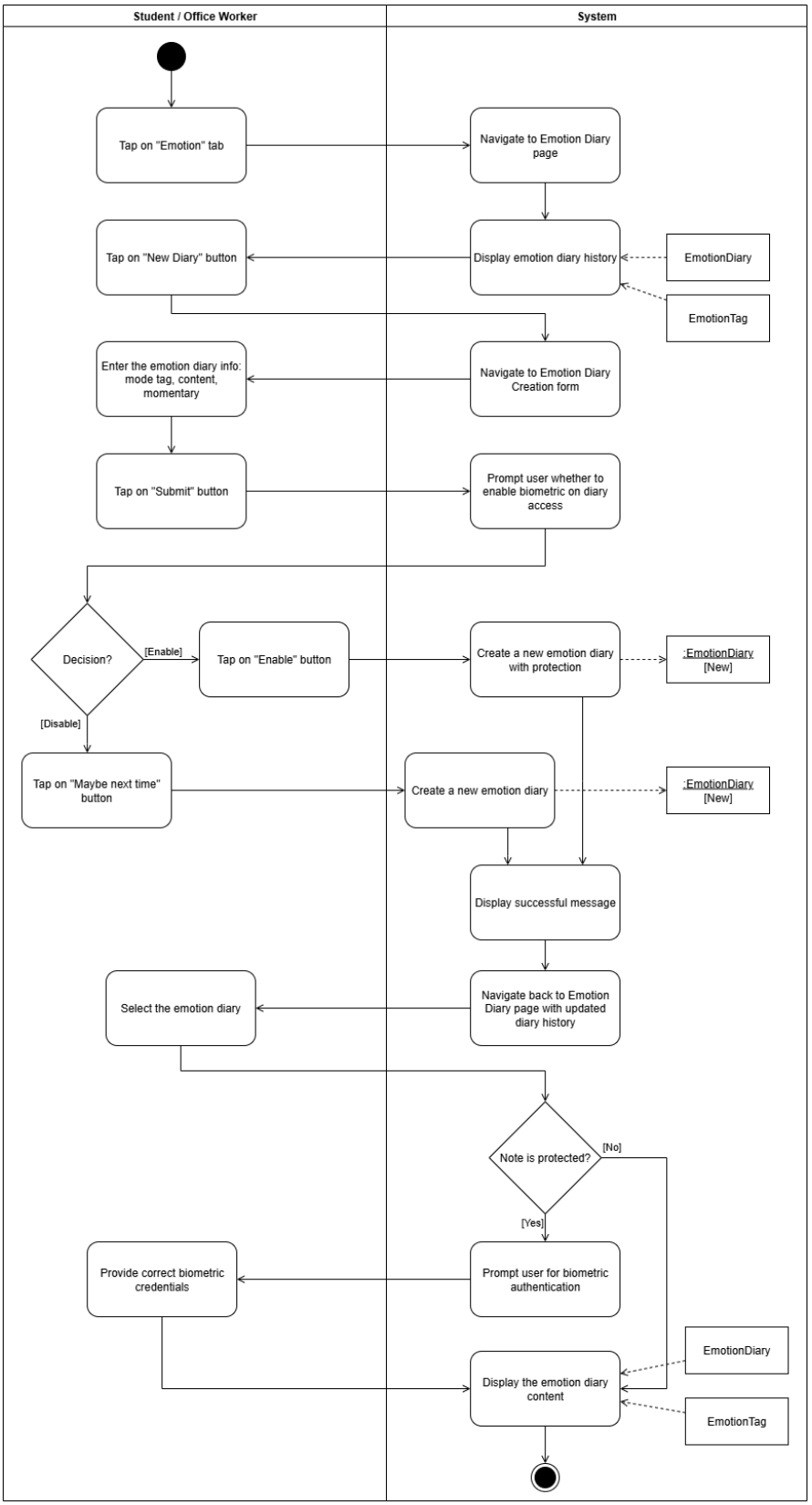


Diagram 4.6.4: Activity Diagram - Emotion Diary Note Module: Create Emotion Diary

Focus Timer Module

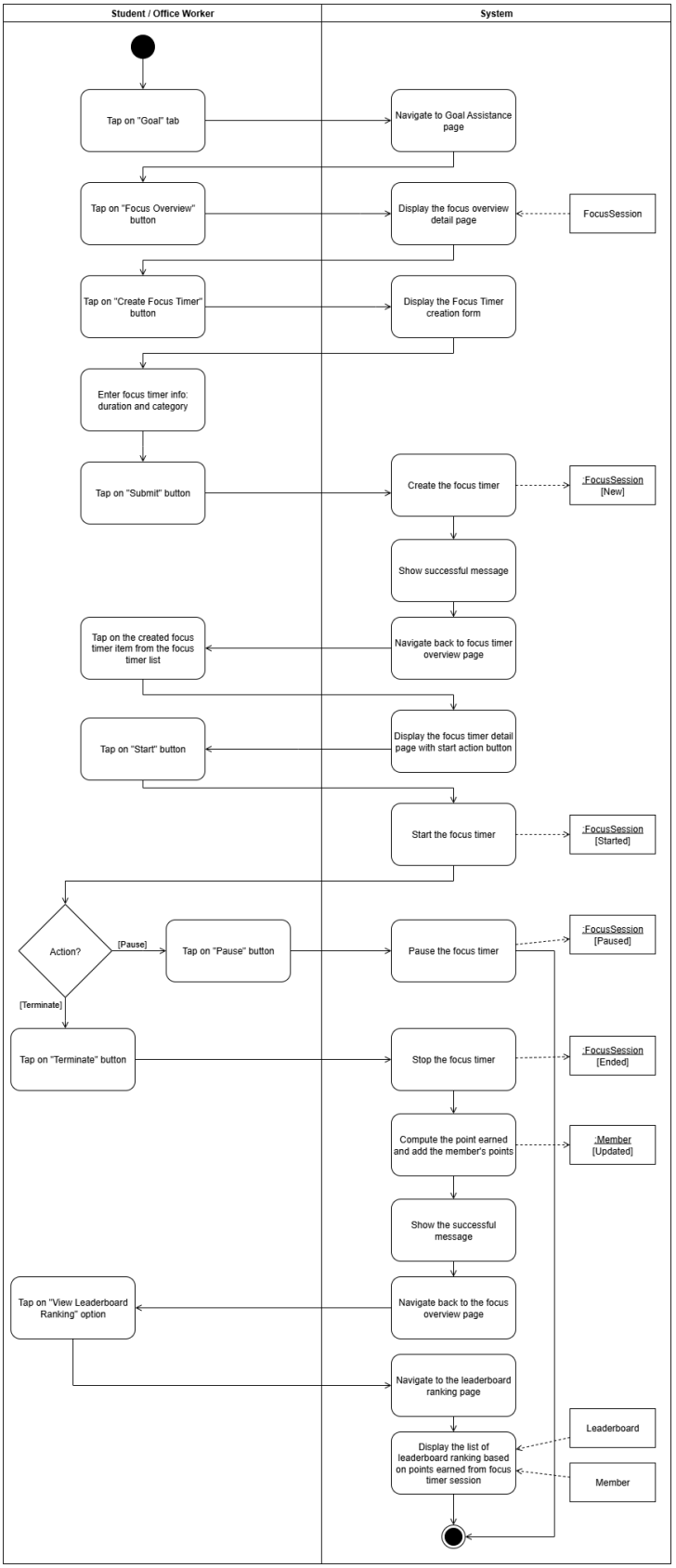


Diagram 4.6.5: Activity Diagram - Focus Timer Module: Manage Focus Timer Session

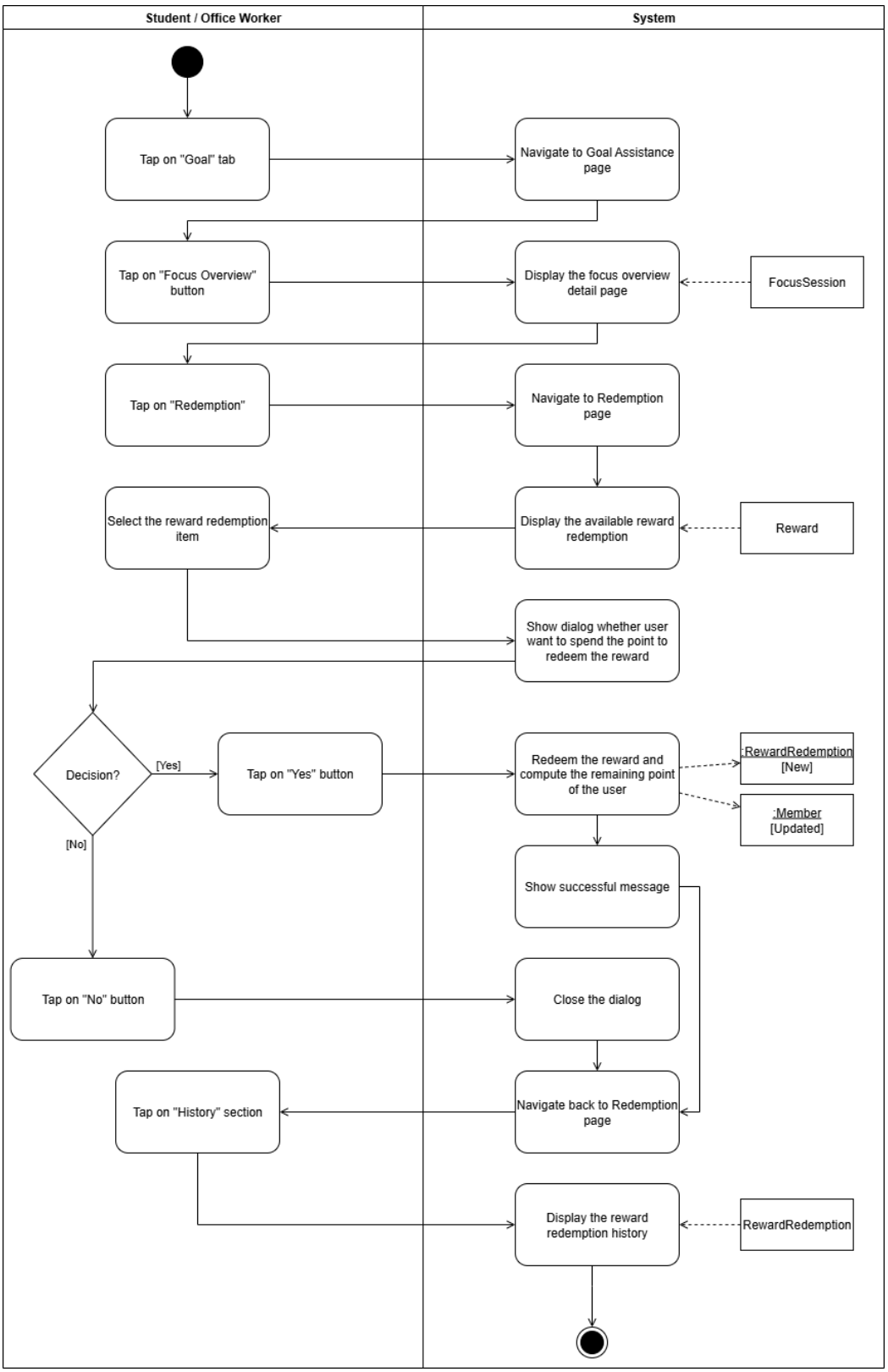


Diagram 4.6.6: Activity Diagram - Focus Timer Module: Collect Rewards

User Module

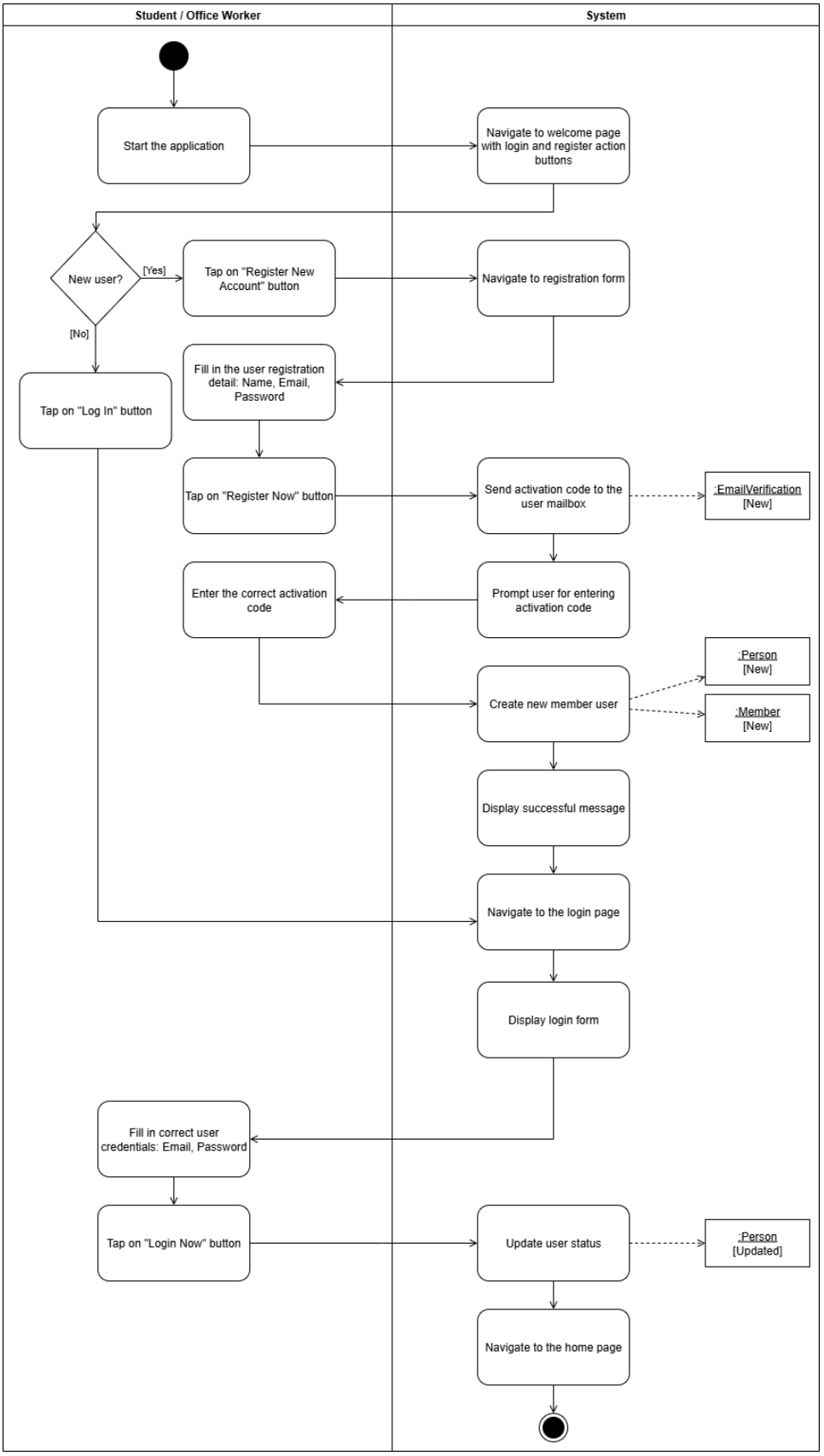


Diagram 4.6.7: Activity Diagram - User Module: Register and Login Account

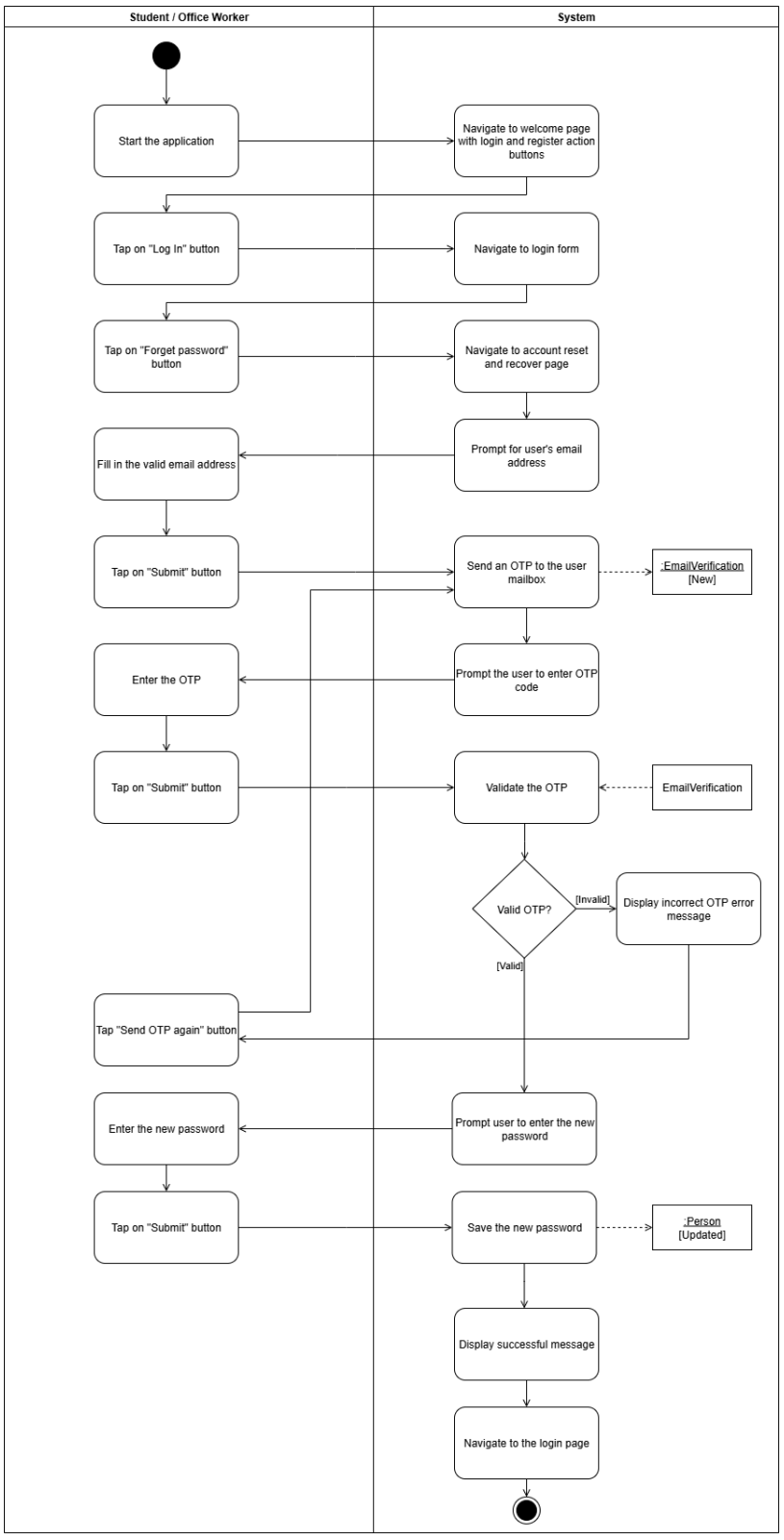


Diagram 4.6.8: Activity Diagram - User Module: Reset or Recover Password

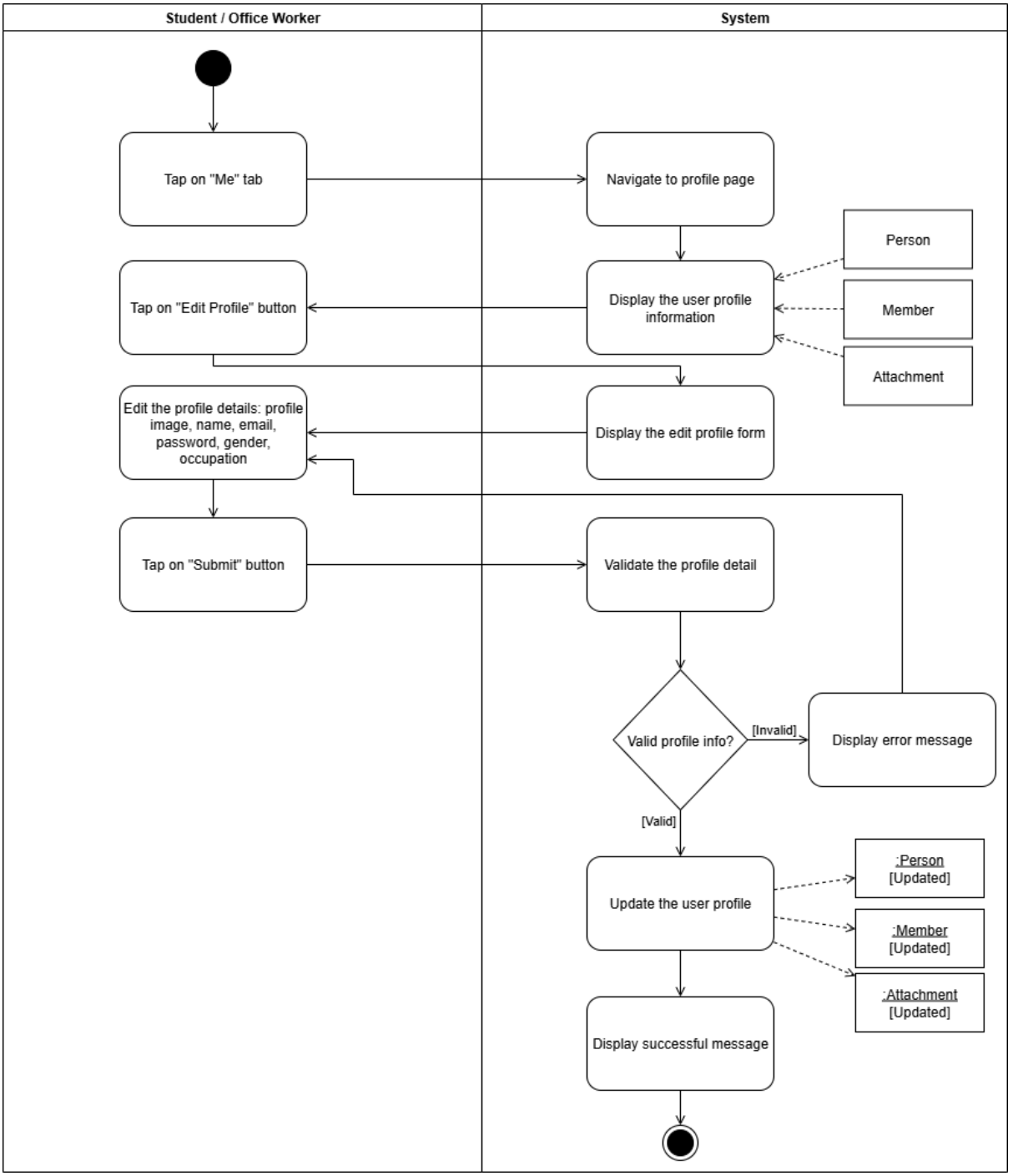


Diagram 4.6.9: Activity Diagram - User Module: Update Profile

Report module

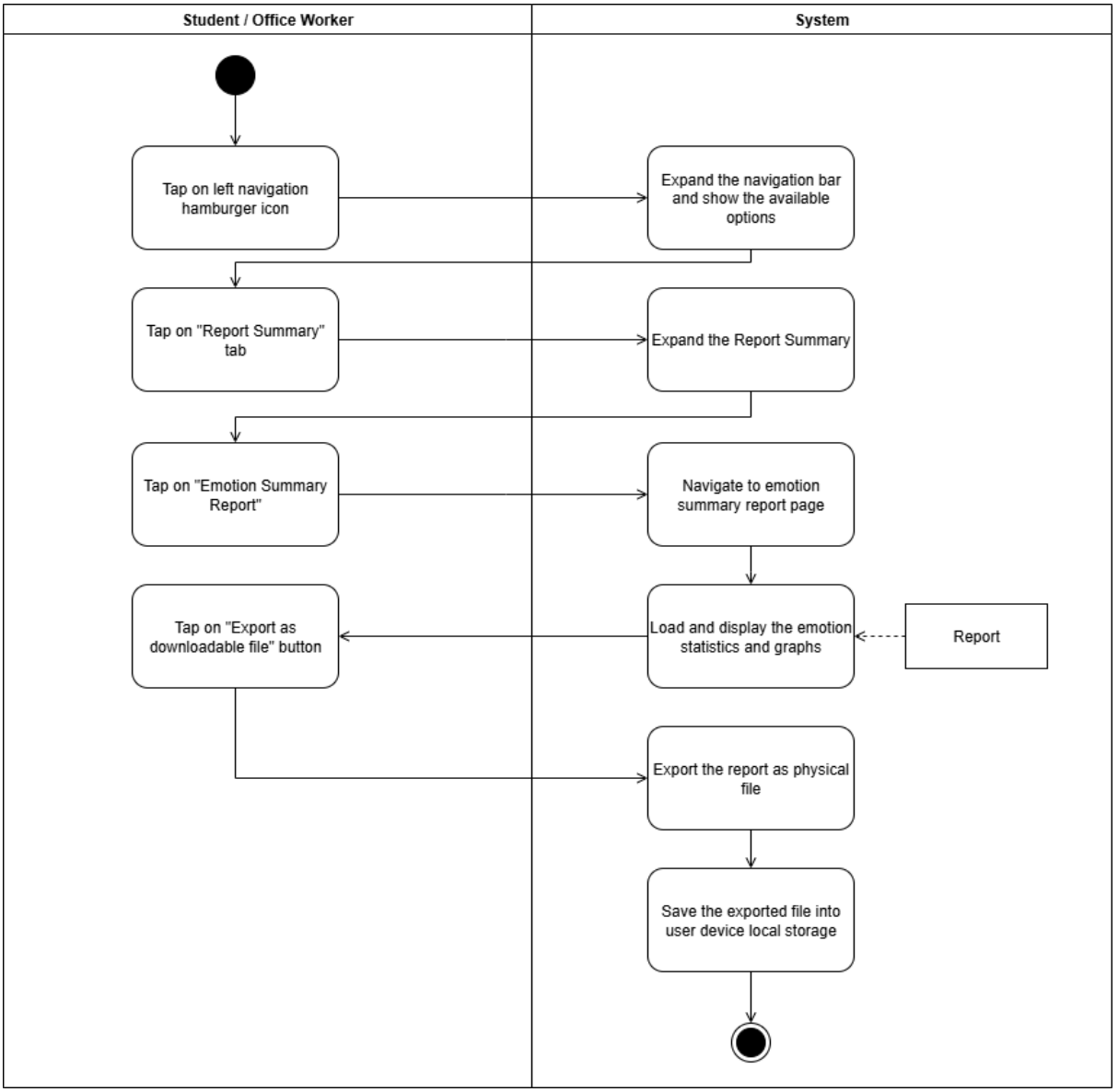


Diagram 4.6.10: Activity Diagram - Report Module: View Emotion Summary Report

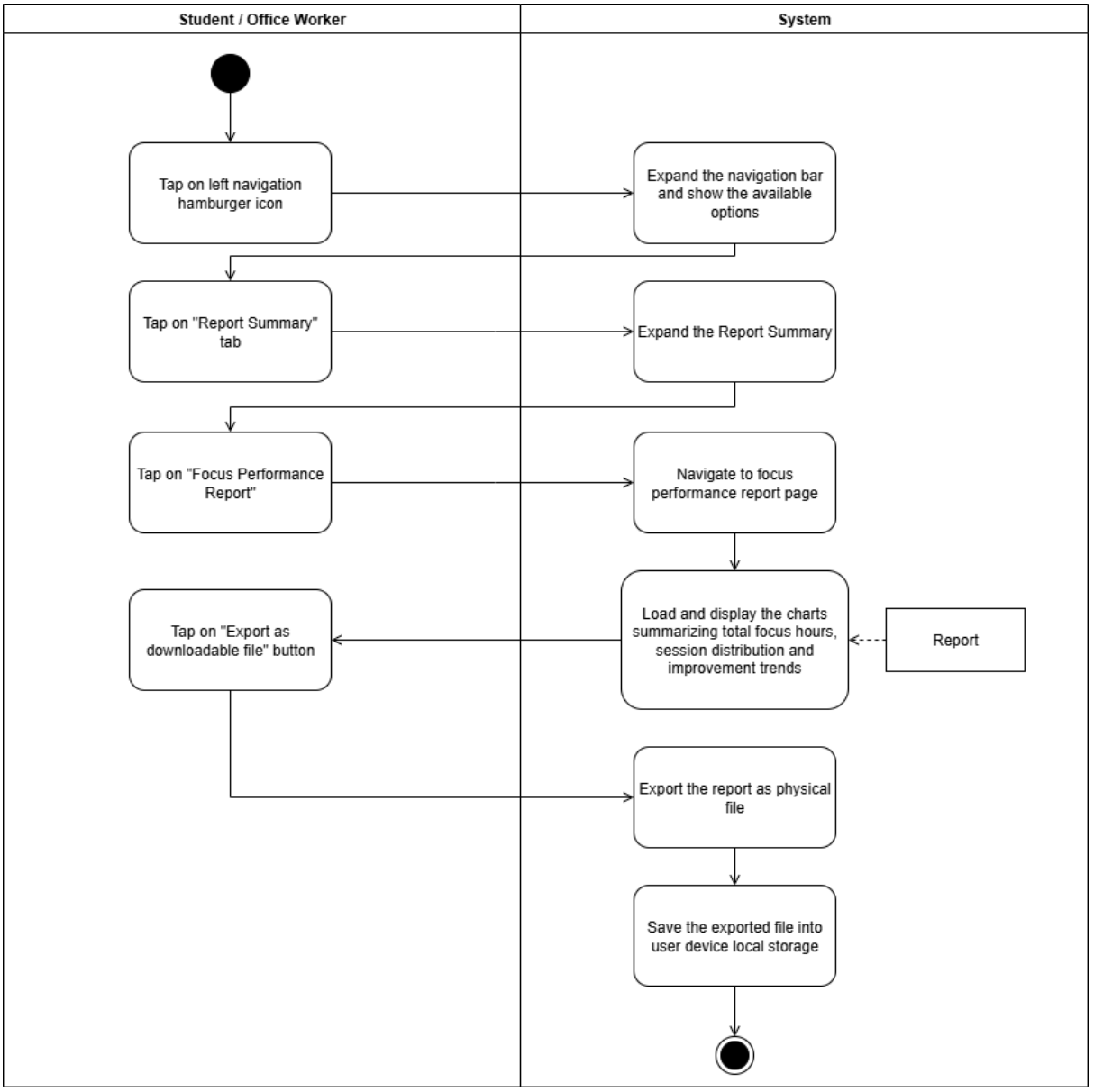


Diagram 4.6.11: Activity Diagram - Report Module: View Focus Performance Report

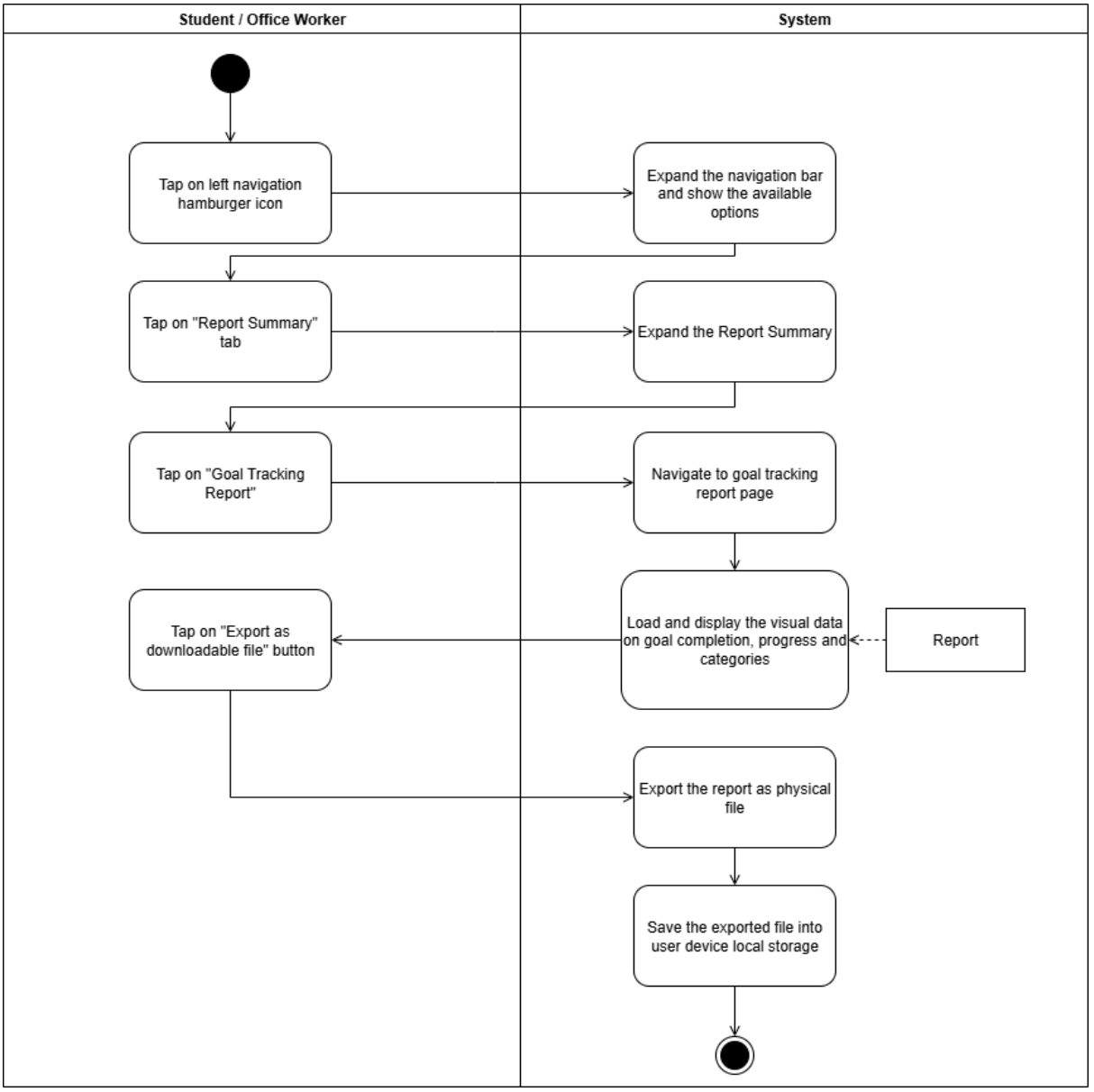


Diagram 4.6.12: Activity Diagram - Report Module: View Goal Tracking Report

4.7 Software Architecture Design

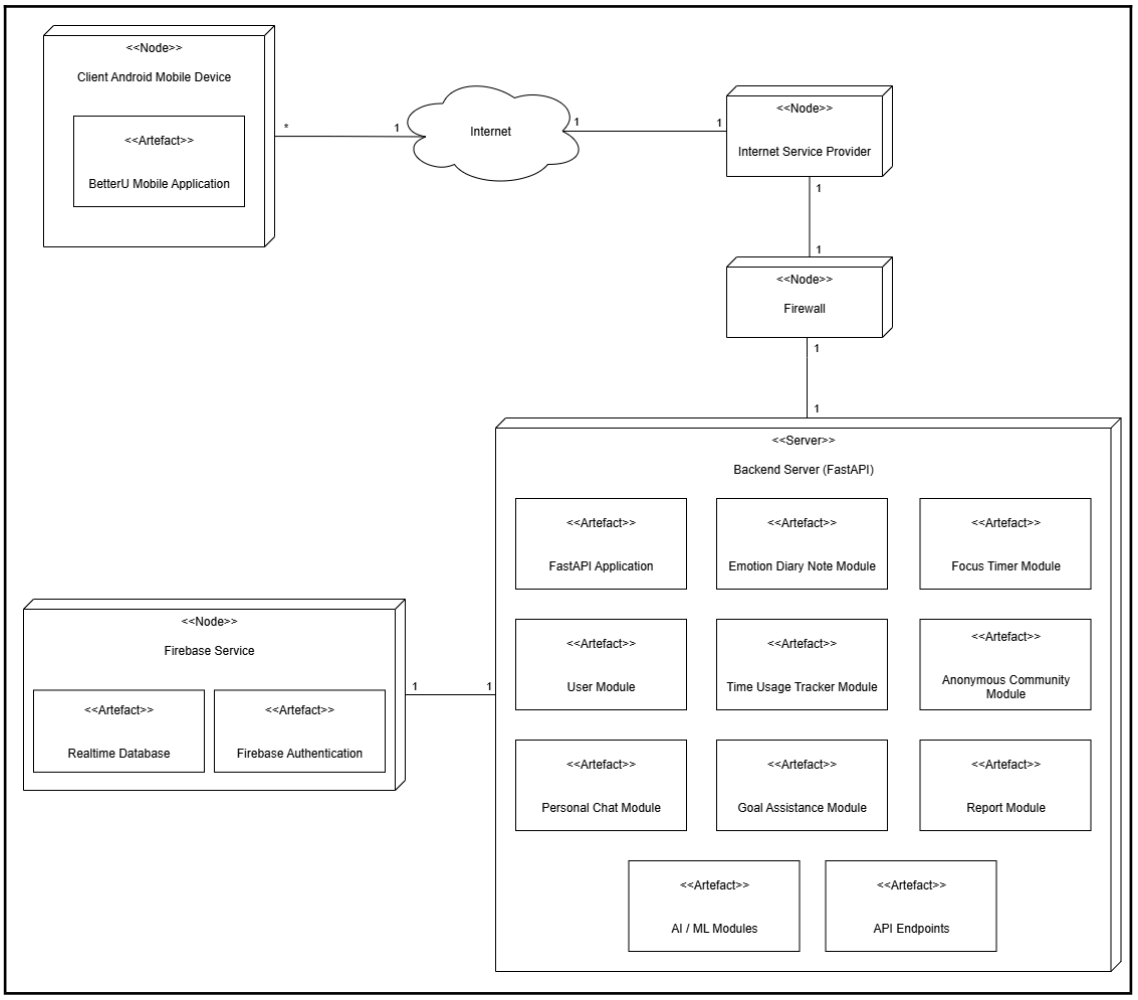


Diagram 4.7.1: Deployment Diagram of BetterU Mobile Application

4.8 AI Algorithms

Task Categorization and Prioritization

- Algorithm: Random Forest Classifier
- Application: This algorithm helps for predicting the user task input's category and priority levels after training the model. This can efficiently reduce the users' effort to manually set the task category and priority level. Meanwhile, all the urgent tasks can be immediately detected and classified.
- Synthetic dataset: 3270 labeled task entries. Each task is annotated with category (Study, Working, Entertainment, Exercise, Personal) and priority level (1, 2, 3, 4)
- Task Category and Priority Dataset:

task_text	category	priority
watch tutorial on computer science	Study	4
complete chemistry assignment	Study	1
prepare for biology quiz	Study	1
revise biology notes	Study	2
organize study materials for english	Study	3
organize study materials for english	Study	3
prepare for physics quiz	Study	1
prepare for biology quiz	Study	1
summarize history readings	Study	2
review english flashcards	Study	3
call client for feedback	Working	1
update task tracker	Working	3
attend team meeting	Working	3
send project update email	Working	1
write marketing report	Working	2
debug application issue	Working	1

design presentation layout	Working	4
design presentation layout	Working	4
update task tracker	Working	3
review marketing documents	Working	2
attend live event	Entertainment	3
scroll through social media	Entertainment	4
chat with friends	Entertainment	2
watch funny YouTube videos	Entertainment	4
chat with friends	Entertainment	2
watch romantic movie	Entertainment	3
watch funny YouTube videos	Entertainment	4
play console games	Entertainment	3
attend live event	Entertainment	3
watch documentary movie	Entertainment	3
go for a short run	Exercise	1
cycle around the neighborhood	Exercise	2
go for a short run	Exercise	1
lift weights at gym	Exercise	1
attend marketing fitness class	Exercise	3
do 30-minute workout at home	Exercise	2
practice football skills	Exercise	2
follow PC fitness video	Exercise	2
follow console fitness video	Exercise	2
follow console fitness video	Exercise	2
Buy groceries for the week	Personal	1

Call grandparents to check in	Personal	1
Clean out your closet	Personal	2
Schedule a dentist appointment	Personal	1
Write in your personal journal	Personal	2
Plan your monthly budget	Personal	2
Do laundry and fold clothes	Personal	2
Organize your photo gallery	Personal	3
Declutter your workspace	Personal	2
Backup important files to the cloud	Personal	1

Table 4.8.1: Task Categorization and Prioritization - Task Category and Priority Dataset

Smart Task Advisor

- Algorithm: Hybrid Recommendation Model - LightFM (collaborative + content-based filtering) with Matrix Factorization and Embeddings
- Application: The algorithm will learn the user preferences and recommends the suitable tasks or scheduling strategies. This can help the system to intelligently analyze user's tasks info and user preferences for recommending a better schedule of task execution time.
- Dataset:
    - Users dataset: 300 users with various ages, occupation, preferences and productivity styles.
    - Tasks dataset: 600 tasks data records with task name, task type, priority and estimated duration.
    - Interaction dataset: 6000+ interactions data records which records the user-task interaction such as completed, rescheduled, in-progress and skipped. Besides, the interaction score is labelled for each interaction record.
- Sample dataset content:
    - Users Dataset

user_id	age	occupation	preferences	productivity_styles	task_preferences

user_001	22	student	study;fitness	Morning-person	short-tasks
user_002	28	office worker	health;personal	Night-owl	high-priority-first
user_003	19	student	study;entertainment	Balanced	long-tasks
user_004	24	office worker	fitness;working	Morning-person	flexible
user_005	21	student	study;health	Night-owl	short-tasks
user_006	27	office worker	personal;entertainment	Balanced	high-priority-first
user_007	23	student	study;fitness	Morning-person	long-tasks
user_008	26	office worker	working;health	Night-owl	flexible
user_009	20	student	study;personal	Balanced	short-tasks
user_010	25	office worker	fitness;entertainment	Morning-person	high-priority-first

Table 4.8.2: Smart Task Advisor - Users Dataset

- Tasks Dataset

task_id	task_name	task_type	priority	estimated_duration	created_at
T001	Morning Jog	exercise	3	30	1/6/2024 7:00
T002	Read a Book	personal	2	45	1/6/2024 9:00
T003	Complete Assignment	study	5	120	1/6/2024 10:00
T004	Team Meeting	working	4	60	1/6/2024 11:00
T005	Watch Documentary	entertainment	2	90	1/6/2024 20:00

T006	Evening Walk	exercise	2	25	1/6/2024 18:00
T007	Write Journal	personal	1	20	1/6/2024 21:00
T008	Prepare Presentation	working	5	90	2/6/2024 9:00
T009	Study Math	study	4	60	2/6/2024 14:00
T010	Listen to Podcast	entertainment	1	30	2/6/2024 19:00

Table 4.8.3: Smart Task Advisor - Tasks Dataset

- Interaction Dataset

interaction_id	**user_id**	**task_id**	**interaction_type**	**interaction_score**	**timestamp**
int_000001	user_001	T143	completed	5	13/7/2024 6:05
int_000002	user_001	T262	rescheduled	3	25/7/2024 7:02
int_000003	user_001	T547	rescheduled	3	26/9/2024 10:38
int_000004	user_001	T063	completed	5	20/6/2024 9:45
int_000005	user_001	T043	completed	5	13/6/2024 8:11
int_000006	user_001	T198	completed	5	9/8/2024 7:03
int_000007	user_001	T252	completed	5	24/7/2024 10:40
int_000008	user_001	T024	skipped	1	15/6/2024 8:41
int_000009	user_001	T530	completed	5	9/10/2024 8:12
int_000010	user_001	T246	completed	5	25/7/2024 6:38

Table 4.8.4: Smart Task Advisor - Interaction Dataset

Task Rescheduling Prediction

- Algorithm: Random Forest (for task success / delay prediction) and LSTM (Long Short-Term Memory neural network) for temporal pattern learning
- Application: The algorithm can predict the likelihood that a task will be delayed or missed. If there is a delay predicted, the system will propose a rescheduling to a more optimal time based on the user's past behavior and daily rhythms.
- Synthetic Dataset:
    - Interactions with reschedules features: This dataset inherits the Interactions dataset and expands the attributes such as previous_task_outcome, scheduled_start_time, actual_start_time, delay_minutes and other scheduling event related attributes.
- Interaction with Rescheduling Dataset in Text Format:

```
interaction_id,user_id,task_id,interaction_type,interaction_score,timestamp,previous_task_outcome,scheduled_start_time,actual_start_time,delay_minutes,delay_status,reschedule_count,task_priority,user_engagement_score,day_of_week,time_of_day,task_duration_estimate,completion_rate_past_week,moving_avg_delay_7d
int_000005,user_001,T043,completed,5,2024-06-13 08:11:55,None,2024-06-13 08:11:55,2024-06-13 08:11:55,0,On-time,0,High,1.0,Thursday,Morning,40,0.93,7.1
int_003170,user_001,T034,completed,5,2024-06-14 06:48:47,Completed,2024-06-14 06:48:47,2024-06-14 06:48:47,0,On-time,0,Low,1.0,Friday,Morning,15,0.98,2.5
int_000008,user_001,T024,skipped,1,2024-06-15 08:41:23,Completed,2024-06-15 08:41:23,2024-06-15 08:41:23,0,Missed,0,Low,1.0,Saturday,Morning,60,0.85,6.1
int_000004,user_001,T063,completed,5,2024-06-20 09:45:06,Delayed,2024-06-20 09:44:06,2024-06-20 09:45:06,1,On-time,0,High,1.0,Thursday,Morning,30,0.97,2.4
int_003475,user_001,T091,completed,5,2024-06-21 06:33:55,Completed,2024-06-21 06:32:55,2024-06-21 06:33:55,1,On-time,0,Low,1.0,Friday,Morning,20,0.72,8.9
int_003584,user_001,T109,skipped,1,2024-06-25 06:18:18,Completed,2024-06-25 06:16:18,2024-06-25 06:18:18,2,Missed,0,Low,1.0,Tuesday,Morning,90,0.66,3.3
int_003495,user_001,T094,completed,5,2024-07-07 08:43:26,Delayed,2024-07-07 08:43:26,2024-07-07 08:43:26,0,On-time,0,Low,1.0,Sunday,Morning,15,0.86,3.8
int_000001,user_001,T143,completed,5,2024-07-13 06:05:16,Completed,2024-07-13 05:55:16,2024-07-13 06:05:16,10,Delayed,0,High,1.0,Saturday,Morning,25,0.95,3.5
int_003897,user_001,T169,postponed,2,2024-07-19 11:50:00,Completed,2024-07-19 11:50:00,2024-07-19 11:50:00,0,On-time,1,Low,1.0,Friday,Morning,90,0.79,2.7
int_000007,user_001,T252,completed,5,2024-07-24 10:40:56,Delayed,2024-07-24 10:30:56,2024-07-24 10:40:56,10,Delayed,0,High,1.0,Wednesday,Morning,60,0.63,1.4
```

Table 4.8.5: Task Rescheduling Prediction - Interaction with Rescheduling Dataset

Time Optimization and Adaptive Scheduling

- Algorithm: Constraint Programming (Google OR-Tools) and Reinforcement Learning (RLib)

- Application: The OR-Tools will generate the initial conflict-free schedules based on the logical constraints. Meanwhile, the Reinforcement Learning will then adapt the schedule over time by analyzing which patterns will cause a higher productivity. This can enable the system to continuously improve its scheduling strategies.
- Synthetic Dataset:
    - User time availability: This dataset mainly records each user's available and unavailable day of week, start time and end time. So, it can ease the system to distribute the suitable tasks at suitable time periods without clashing with users' unavailable time period after implementing the features into the system.
    - Constraints: This dataset defines the scheduling constraints for tasks with consideration of hard and soft rules. It contains task id, constraint type (e.g. mush_finish_before, cannot_overlap_with, min_duration_minutes), value for the constraint and constraint strength (e.g. soft, hard).
    - Schedule history: This dataset simulates the history and outcomes for each user's task assignment. It contains user id, task id, scheduled_start, scheduled_end, outcome and productivity score.
- Sample dataset content:
    - User Time Availability

user_id	**day_of_week**	**start_time**	**end_time**	**availability_status**
user_001	Monday	6:00	10:00	available
user_001	Monday	10:00	12:00	available
user_001	Monday	13:00	16:00	available
user_001	Monday	12:00	13:00	unavailable
user_001	Tuesday	6:00	10:00	available
user_001	Tuesday	10:00	12:00	available
user_001	Tuesday	13:00	16:00	available
user_001	Tuesday	12:00	13:00	unavailable
user_001	Wednesday	6:00	10:00	available
user_001	Wednesday	10:00	12:00	available

Table 4.8.6: Time Optimization and Adaptive Scheduling - User Time Availability Dataset

    - Constraints

task_id	constraint_type	value	constraint_strength
T001	min_duration_minutes	30	hard
T002	min_duration_minutes	45	hard
T003	must_finish_before	20:00	hard
T003	cannot_overlap_with	long_task	hard
T003	min_duration_minutes	120	hard
T004	must_finish_before	20:00	hard
T004	min_duration_minutes	60	hard
T005	cannot_overlap_with	long_task	hard
T005	min_duration_minutes	90	hard
T006	min_duration_minutes	25	hard

Table 4.8.7: Time Optimization and Adaptive Scheduling - Constraints Dataset

- Schedule History

user_id	task_id	scheduled_start	scheduled_end	actual_start	actual_end	outcome	productivity_score
user_001	T043	13/6/2024 8:11	13/6/2024 8:51	13/6/2024 8:11	13/6/2024 8:51	completed	0.93
user_001	T034	14/6/2024 6:48	14/6/2024 7:03	14/6/2024 6:48	14/6/2024 7:03	completed	0.98
user_001	T024	15/6/2024 8:41	15/6/2024 9:41			missed	0.85

user_001	T063	20/6/2024 9:44	20/6/2024 10:14	20/6/2024 9:45	20/6/2024 10:15	completed	0.97
user_001	T091	21/6/2024 6:32	21/6/2024 6:52	21/6/2024 6:33	21/6/2024 6:53	completed	0.72
user_001	T109	25/6/2024 6:16	25/6/2024 7:46			missed	0.66
user_001	T094	7/7/2024 8:43	7/7/2024 8:58	7/7/2024 8:43	7/7/2024 8:58	completed	0.86
user_001	T143	13/7/2024 5:55	13/7/2024 6:20	13/7/2024 6:05	13/7/2024 6:30	completed	0.95
user_001	T169	19/7/2024 11:50	19/7/2024 13:20	19/7/2024 11:50	19/7/2024 13:20	delayed	0.79
user_001	T252	24/7/2024 10:30	24/7/2024 11:30	24/7/2024 10:40	24/7/2024 11:40	completed	0.63

Table 4.8.8: Time Optimization and Adaptive Scheduling - Schedule History Dataset

4.9 Chapter Summary and Evaluation

This chapter has covered the main system design of the BetterU mobile application with support of useful explanation, diagrams and sample datasets. It mainly described both structural and behavioral aspects of the mobile application. For example, the sequence diagram, state chart diagram, activity diagram and user interface diagram has provided the users with a brief overview of the system processes and user interactions. Meanwhile, the data design was also documented through the class diagram, entity relationship diagram and data dictionary. This can ensure a clear understanding of the system entities and relationships offered. From the aspect of report design, the productivity report and social performance report were planned with detailed UI components and data granularity. Moreover, the deployment diagram and process design were also involved for providing an architectural view of how the application components are organized. Eventually, the AI algorithms that will be integrated into the BetterU mobile application are also demonstrated since they are used for providing intelligent and personalized features. In overall, the design has provided a comprehensive blueprint of BetterU mobile application requirements including functional and

non0-functional requirements. Via the well-defined system design, the modularity, scalability and maintainability of the system can be promised.

Chapter 5

Implementation and Testing

5 Implementation and Testing

This chapter presents the complete technical implementation of the BetterU mobile application, describing the underlying architectures, AI components and integrated subsystems that enable the platform's intelligent productivity and emotional well-being features. Each major module is supported by a combination of advanced backend services, machine-learning models, natural language processing pipelines and real-time mobile workflows, all designed to work together seamlessly to create a cohesive user experience.

The chapter begins by detailing the system's speech-to-text processing pipeline, which employs a hybrid approach using Vosk for low-latency partial transcription and Whisper for accurate final transcription. This dual-engine design allows users to create tasks naturally through voice input while ensuring transcription accuracy. The output from speech processing is then passed into the task extraction module, which uses spaCy-based NLP rules, SetFit classification models and datetime parsing strategies to convert user instructions into structured tasks with identified activities, time ranges, categories and priorities.

Subsequent sections describe the Emotion Diary Note subsystem, which integrates a fine-tuned DistilBERT model, sentiment scoring logic and keyword-based refinement to interpret diary entries and generate emotional insights. The system also supports AI-driven diary autocomplete suggestions designed to help users express their feelings more effectively. These emotional signals are further combined with task-related emotional analysis inside the Unified Emotion Advisor Engine, which fuses task emotions, diary sentiment trends and behavioural patterns to produce personalised, cross-module recommendations.

The chapter concludes by outlining the testing methodology used to validate the interconnected behaviour of all modules. Integration testing was adopted to ensure that the system's components, including authentication, AI models, task assistants, focus sessions and emotion analytics which interact reliably and reflect real-world usage conditions. Together, these implementations form the intelligent backbone of BetterU and ensure consistent performance across its productivity and emotional well-being features.

5.1 Implementation and Coding

5.1.1 Speech-to-Text Processing

The speech-to-text feature is used in the Goal Assistance module (shared module) to allow users to create tasks using voice input. For example, when a user says "Play badminton on next Wednesday from 3 PM to 5 PM", the system will convert the speech into text and later send it to the task extraction module to identify the activity, date and time details.

In order to achieve both real-time responsiveness and high transcription accuracy, the system applied a hybrid speech recognition pipeline:

- The offline **Vosk** model processes streaming audio chunks during recording in order to provide low-latency and real-time text previews.
- After the user finishes speaking, the online **Whisper** model will reprocess the full audio to generate a more accurate final transcription.

Vosk Warm-Up

```
vosk_model = None
vosk_ready = False
_warmup_started = False

def warmup_vosk():
    global vosk_model, vosk_ready, _warmup_started
    if _warmup_started:
        return
    _warmup_started = True

    print("[VOSK] Background warm-up started...")
    try:
        vosk_model = vosk.Model("vosk-model-small-en-us-0.15")
        vosk_ready = True
        print("[VOSK] Model ready in background!")
    except Exception as e:
        print("[VOSK] Failed to load model:", e)
```

Partial Transcription Pipeline

```
def transcribe_partial(audio_path: str, session_id: str, sample_rate: int =
16000):
    # read_audio_frames returns (bytes, sr)
    data_bytes, sr = read_audio_frames(audio_path)

    # Normalize and ensure correct byte-order & length for VAD
    norm_bytes = normalize_audio_pcm16(data_bytes, in_rate=sr, out_rate=sr)

    # ensure frames are multiples of frame_length for VAD
    frame_duration = 30
    frame_length = int(sr * frame_duration / 1000) * 2
    if len(norm_bytes) % frame_length != 0:
```

```
        pad_len = frame_length - (len(norm_bytes) % frame_length)
        norm_bytes = norm_bytes + (b"\x00" * pad_len)

    # create/get recognizer with actual sr
    rec = get_or_create_recognizer(session_id, sr)

    # VAD check on normalized bytes
    if not is_speech(norm_bytes, sr):
        return {"text": ""}

    # Feed the normalized bytes to vosk
    if rec.AcceptWaveform(norm_bytes):
        res = json.loads(rec.Result())
        text = res.get("text", "")
    else:
        res = json.loads(rec.PartialResult())
        text = res.get("partial", "")

    print(f"Vosk partial transcription for {session_id}: {text}")
    return {"text": text}
```

Whisper Language Detection

```
def detect_language(audio_path: str) -> str:
    audio = whisper.load_audio(audio_path)
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(model.device)

    _, probs = model.detect_language(mel)
    detected_lang = max(probs, key=probs.get)
    print(f"Detected language: {detected_lang}")

    if detected_lang in ["en", "zh"]:
        return detected_lang
    return "en"  # default to English
```

Finalization Logic

```
def finalize_session(session_id: str):
    # Find recognizer matching this session id prefix
    recog_key = None
    for k in list(recognizers.keys()):
        if k.startswith(f"{session_id}-"):
            recog_key = k
            break

    final_text = ""
    if recog_key:
        final_text = json.loads(recognizers[recog_key].FinalResult()).get("text",
"")
        del recognizers[recog_key]
        print(f"Vosk finalized for {session_id}: {final_text}")
    else:
        print(f"No recognizer found for session {session_id}")
```

```
    session_audio_path = os.path.join(tempfile.gettempdir(), f"{session_id}.wav")

    if not os.path.exists(session_audio_path):
        return {"final_text": final_text}

    print(f"Found audio for session {session_id}, proceeding with Whisper
transcription.")

    try:
        lang = detect_language(session_audio_path)
        whisper_text = transcribe_with_whisper(session_audio_path, language=lang)
        print(f"Whisper transcription: {whisper_text}")
        return {"final_text": final_text, "refined_text": whisper_text,
"language": lang}
    except Exception as e:
        print("Whisper transcription error:", e)
        return {"final_text": final_text, "refined_text": "", "language": "en"}
```

5.1.2 Task Extraction

The task extraction feature will work together with the speech-to-text service to automatically understand and structure the user's task instructions. After the speech has been transcribed into text, this task extraction service will process the sentence using the combination of spaCy NLP, rule-based parsing and machine-learning classification to extract meaningful task details.

The system identifies key information such as:

- Task activity
- Date and time expressions
- Start and end datetime
- Task category
- Task priority

Task Phrase Extraction

```
def extract_task_phrase_from_doc(doc, time_token_idxs) -> str:
    TIME_CONNECTORS = {"from", "until", "between", "and", "around", "about"}
    ABSOLUTE_EXCLUDE = {"am", "pm"} | TIME_CONNECTORS

    # Ending verbs to remove from task phrase
    ENDING_VERBS = {"end", "finish", "stop", "complete", "conclude", "wrap"}

    root = None
    for tok in doc:
        if tok.head == tok:
            root = tok
            break
```

```
    if root is None:
        return doc.text.strip()

    task_tokens = set()
    task_tokens.add(root)

    def add(token):
        for t in token.subtree:

            # skip time tokens
            if t.i in time_token_idxs:
                continue
            # skip AM/PM and connectors entirely
            if t.lower_ in ABSOLUTE_EXCLUDE:
                continue
            # skip "to" ONLY if it is actually part of a time expression
            if t.lower_ == "to" and (t.i-1 in time_token_idxs or t.i+1 in
time_token_idxs):
                continue
            # skip numbers that are part of time
            if t.pos_ == "NUM" and t.i in time_token_idxs:
                continue
            # skip "for" in "for X hours"
            if t.lower_ == "for" and re.search(r"\bfor \d+", doc.text.lower()):
                continue
            # skip ending verbs ("end", "finish", etc.)
            if t.lemma_.lower() in ENDING_VERBS:
                continue

            task_tokens.add(t)

    # add main verb + complements
    add(root)

    # add coordinated verbs but NOT ending verbs
    for child in root.children:
        if child.dep_ in ("conj", "xcomp", "ccomp") and child.i not in
time_token_idxs:
            if child.lemma_.lower() not in ENDING_VERBS:
                add(child)

    # assemble phrase
    sorted_tokens = sorted(task_tokens, key=lambda t: t.i)
    phrase = " ".join(tok.text for tok in sorted_tokens)
    phrase = re.sub(r"\s+", " ", phrase).strip()

    # Remove trailing single junk words like "am", "finish", "end"
    trailing_junk = {"am", "pm"} | ENDING_VERBS
    words = phrase.split()

    if len(words) > 1 and words[-1].lower() in trailing_junk:
        words = words[:-1]

    phrase = " ".join(words).strip()

    return phrase
```

Datetime Phrase Extraction

```
def extract_datetime_phrase(doc, time_token_idxs) -> str:
    # Special handling for "between X and Y"
    text = doc.text.lower()
    if "between" in text and "and" in text:
        return re.search(r"between (.+?) and (.+)", doc.text,
flags=re.I).group(0)

    # expand by including "from", "to", "until", "between", "and"
    connectors = {"from", "to", "until", "between", "and"}

    expanded = set(time_token_idxs)
    for i in list(time_token_idxs):
        # expand left
        if i > 0 and doc[i - 1].lower_ in connectors:
            expanded.add(i - 1)
        # expand right
        if i < len(doc) - 1 and doc[i + 1].lower_ in connectors:
            expanded.add(i + 1)

    time_token_idxs = expanded

    if not time_token_idxs:
        return ""

    sorted_ids = sorted(time_token_idxs)

    spans = []
    current = [sorted_ids[0]]
    for idx in sorted_ids[1:]:
        if idx == current[-1] + 1:
            current.append(idx)
        else:
            spans.append(current)
            current = [idx]
    spans.append(current)

    parts = []
    for span in spans:
        tokens = [doc[i].text for i in span]
        parts.append(" ".join(tokens))

    # keep exact structure
    phrase = " ".join(parts)
    return phrase.strip(" ,.")
```

Main Structured Task Extraction Function

```
def extract_structured_task(clause: str, base_date: datetime | None = None):
    if base_date is None:
        base_date = datetime.now()

    doc = nlp(clause)

    # mark all time-related tokens
```

```
    time_token_idxs = _get_time_token_indices(doc)

    # build task phrase and datetime phrase
    task_phrase = extract_task_phrase_from_doc(doc, time_token_idxs)
    datetime_phrase = extract_datetime_phrase(doc, time_token_idxs)

    from setfit import SetFitModel

    category_model =
SetFitModel.from_pretrained("app/core/task_extraction/model_task_category")
    priority_model =
SetFitModel.from_pretrained("app/core/task_extraction/model_task_priority")

    def classify_task(task_phrase):
        raw_cat = category_model(task_phrase)
        raw_pri = priority_model(task_phrase)

        # Convert numpy -> Python string
        category = str(raw_cat)

        # Convert tensor -> Python int
        if hasattr(raw_pri, "item"):
            priority = raw_pri.item()
        else:
            priority = int(raw_pri)

        return category, priority

    task_category, task_priority = classify_task(task_phrase)

    # parse datetime
    datetime_info = extract_datetime_info(datetime_phrase, base_date)

    return {
        "task": task_phrase,
        "category": task_category,
        "priority": task_priority,
        "start": datetime_info["start"],
        "end": datetime_info["end"],
        "date": datetime_info["date"],
        "raw": datetime_info["raw"],
    }
```

5.1.3 Emotion Diary Note AI Model

The Emotion Diary Note subsystem is powered by a custom fine-tuned transformer-based model combined with a lightweight sentiment and keyword-rule engine. This hybrid architecture enables the system to analyse diary entries, classify emotional states, calculate sentiment levels, suggest personalised emotional expression, and provide suitable cross-module action. The following subsections describe the full implementation based solely on the developed code.

1. Dataset Preparation and Emotion Class Mapping

To develop this model, the GoEmotions dataset was first loaded and converted from 27 fine-grained labels into seven broader classes (joy, gratitude, sadness, anger, fear, stress, and neutral) using a custom mapping function. Additional domain-specific samples related to academic stress were manually added to improve performance in student-related scenarios. A fixed label encoder was created to ensure consistent class ordering during both training and inference. The dataset was then balanced to 6000 samples per emotion category to prevent bias toward majority classes. Text samples were tokenised using a DistilBERT tokenizer, and the DistilBERT model was fine-tuned for emotion classification using HuggingFace's Trainer API, with evaluation performed at each epoch using accuracy and macro-F1 metrics. After training, the final model, tokenizer, and label encoder were exported as a reusable inference package for integration into the Emotion Diary Note engine. The complete model training script used for this subsystem is shown below.

```
import random
from collections import Counter

import joblib
import numpy as np
import torch
from datasets import load_dataset, Dataset
from sklearn.preprocessing import LabelEncoder
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    set_seed,
)

print("CUDA available:", torch.cuda.is_available())

# Reproducibility
set_seed(42)
random.seed(42)
np.random.seed(42)

# 1️⃣ Load GoEmotions dataset
dataset = load_dataset("go_emotions")

# 2️⃣ Define emotion mapping (GoEmotions → 7 classes)
label_map = {
    "joy": [1, 2, 20, 32],
    "gratitude": [10, 11],
```

```
    "sadness": [14, 15, 16],
    "anger": [0, 3, 4],
    "fear": [7, 8],
    "stress": [18, 22, 5, 6, 9, 12, 26],
    "neutral": [27],
}

def map_emotion(example):
    labels = example["labels"]

    for emotion, ids in label_map.items():
        if any(label in ids for label in labels):
            return {"text": example["text"], "label": emotion}

    # Fallback
    return {"text": example["text"], "label": "neutral"}

print("Mapping dataset...")
train_clean = dataset["train"].map(map_emotion,
remove_columns=dataset["train"].column_names)
val_clean = dataset["validation"].map(map_emotion,
remove_columns=dataset["validation"].column_names)

# extra stress samples (domain-specific)
extra_stress_samples = [
    {"text": "I feel overwhelmed by work and deadlines.", "label": "stress"},
    {"text": "Too many tasks today, I'm exhausted.", "label": "stress"},
    {"text": "I can't focus because I'm so stressed lately.", "label": "stress"},
    {"text": "My mind feels messy and pressured.", "label": "stress"},
    {"text": "I feel anxious about my project presentation.", "label": "stress"},
]

# 3 Encode labels — SINGLE SOURCE OF TRUTH
# Keep this order fixed forever; reuse the same encoder at inference.
CLASS_ORDER = ["joy", "gratitude", "sadness", "anger", "fear", "stress",
"neutral"]

encoder = LabelEncoder()
encoder.fit(CLASS_ORDER)  # encoder.classes_ will be in this exact order

# 4 Tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def preprocess(batch):
    tokens = tokenizer(
        batch["text"],
        truncation=True,
```

```
        padding="max_length",
        max_length=128,
    )
    tokens["label"] = encoder.transform(batch["label"]).tolist()
    return tokens

# 5 Balance classes in train split
print("Preparing balanced train dataset...")
train_list = train_clean.to_list()
print("Before balancing:", Counter(x["label"] for x in train_list))

TARGET = 6000  # number of samples per class (tune if needed)
balanced = []

for emotion in CLASS_ORDER:
    group = [x for x in train_list if x["label"] == emotion]

    if not group:
        continue

    if len(group) >= TARGET:
        balanced.extend(random.sample(group, TARGET))
    else:
        repeats = TARGET // len(group)
        remainder = TARGET % len(group)
        balanced.extend(group * repeats + random.sample(group, remainder))

# Add domain stress samples
balanced.extend(extra_stress_samples)

print("After balancing:", Counter(x["label"] for x in balanced))

train_balanced = Dataset.from_list(balanced)

print("Tokenizing...")
train_enc = train_balanced.map(preprocess, batched=True)
val_enc = val_clean.map(preprocess, batched=True)

train_enc.set_format(type="torch", columns=["input_ids", "attention_mask",
"label"])
val_enc.set_format(type="torch", columns=["input_ids", "attention_mask",
"label"])

# 6 Load model
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=len(CLASS_ORDER),
```

```
)

# Attach label mapping to config (for safety)
id2label = {i: CLASS_ORDER[i] for i in range(len(CLASS_ORDER))}
label2id = {v: k for k, v in id2label.items()}
model.config.id2label = id2label
model.config.label2id = label2id

device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

# 7️⃣ Training arguments
args = TrainingArguments(
    output_dir="emotion_model",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    fp16=torch.cuda.is_available(),
    logging_steps=500,
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
)

# 8️⃣ Metrics (optional, simple accuracy)
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    acc = accuracy_score(labels, preds)
    f1 = f1_score(labels, preds, average="macro")
    return {"accuracy": acc, "macro_f1": f1}

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_enc,
    eval_dataset=val_enc,
    compute_metrics=compute_metrics,
)

# 9️⃣ Train model
trainer.train()
```

```
print("Final train metrics:", trainer.state.log_history[-1])

# 🔟 Save model + tokenizer + encoder
model.save_pretrained("emotion_model")
tokenizer.save_pretrained("emotion_model")
joblib.dump(encoder, "emotion_model/label_encoder.pkl")

print("Training complete! Model + tokenizer + encoder saved in emotion_model~~/")
```

2. Generate Emotion Prediction

The emotion inference component loads the fine-tuned DistilBERT emotion model, tokenizer and label encoder, then processes the user's diary text to generate an emotion prediction. The text is tokenised and passed through the model to obtain raw probabilities for each emotion class. Since diary entries often contain short or direct emotional expressions, a post-processing logic is applied to refine the prediction. This includes keyword checks for common emotional words and a sentiment score using VADER to adjust results that the model may misclassify. The final output includes the refined emotion label, the model's original label, confidence score and full probability distribution.

```
import joblib
import torch
from functools import lru_cache
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

@lru_cache()
def load_model():
    device = "cuda" if torch.cuda.is_available() else "cpu"

    model =
AutoModelForSequenceClassification.from_pretrained("emotion_model").to(device)
    tokenizer = AutoTokenizer.from_pretrained("emotion_model")
    encoder = joblib.load("emotion_model/label_encoder.pkl")

    return model, tokenizer, encoder, device

def _contains(words, text):
    t = text.lower()
    return any(w in t for w in words)

def _final_emotion_logic(raw_label: str, text: str) -> str:
    """Heuristic on top of model prediction to handle simple phrases like
```

```
'happy'/'angry' better."""
    score = analyzer.polarity_scores(text)["compound"]
    t = text.lower().strip()

    # --- Strong overrides by keywords ---
    if _contains(["happy", "happier", "happiness"], t):
        return "joy"

    if _contains(["glad", "thankful", "grateful", "appreciate"], t):
        return "gratitude"

    if _contains(["stress", "stressed", "overwhelmed", "pressure", "deadline",
"assignment"], t):
        return "stress"

    if _contains(["scared", "afraid", "worried", "anxiety", "anxious"], t):
        return "fear"

    if _contains(["angry", "annoyed", "frustrated", "pissed", "mad"], t):
        return "anger"

    # --- Sentiment-based rules ---
    if score >= 0.4:
        # strongly positive → treat as joy unless model is clearly something
else
        if raw_label in ["neutral", "sadness", "fear", "stress"]:
            return "joy"

    if score <= -0.4:
        # strongly negative; if not fear, likely sadness/stress/anger
        if raw_label == "neutral":
            return "sadness"

    # --- Neutral fix ---
    if raw_label == "neutral":
        if score > 0.2:
            return "joy"
        if score < -0.2:
            return "sadness"

    return raw_label

def predict_emotion(text: str) -> dict:
    """
    Main inference function used by FastAPI.

    Returns:
```

```
    {
      "emotion": final_label,
      "confidence": float,
      "raw_label": model_label,
      "scores": {label: prob, ...}
    }
    """
    try:
        model, tokenizer, encoder, device = load_model()

        inputs = tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            padding=True,
        ).to(device)

        with torch.no_grad():
            outputs = model(**inputs)
            probs = torch.softmax(outputs.logits, dim=-1)[0]

        label_id = torch.argmax(probs).item()
        raw_label = encoder.inverse_transform([label_id])[0]

        final_label = _final_emotion_logic(raw_label, text)
        confidence = probs[label_id].item()

        scores = {
            encoder.inverse_transform([i])[0]: float(probs[i].item())
            for i in range(len(probs))
        }

        return {
            "emotion": final_label,
            "confidence": float(confidence),
            "raw_label": raw_label,
            "scores": scores,
        }

    except Exception as e:
        print("❌ ERROR in predict_emotion:", e)
        return {
            "emotion": "neutral",
            "confidence": 0.0,
            "raw_label": "neutral",
            "scores": {},
        }
```

```
if __name__ == "__main__":
    tests = [
        "I'm happy with my FYP progress",
        "I'm happy but tired",
        "I'm scared of failing my exam",
        "Too many deadlines, I'm overwhelmed",
        "I feel grateful for today",
        "I'm sad for no reason",
        "I'm angry at my friend",
        "Nothing special today.",
    ]

    for t in tests:
        print(t, " → ", predict_emotion(t))
```

3. Analyse User Content and Suggest Cross-module Action

The diary analysis component uses the trained emotion model to classify the user's diary entry and then generates a refined emotional label using additional keyword checks and VADER sentiment scoring. The system also computes a continuous sentiment score by combining the model's probabilities, the VADER polarity result and the user's selected mood level. The diary text is first cleaned using a Quill delta parser to extract plain text before analysis. Once the emotional state and sentiment score are obtained, the system generates suggested actions that link the Emotion Diary Note with other modules such as tasks, community and forum. These suggestions are based on the user's emotional intensity, helping the application recommend supportive or meaningful follow-up actions across different subsystems.

```
import json
from functools import lru_cache

import joblib
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from firebase_config import db

analyzer = SentimentIntensityAnalyzer()

# ------------------------------------------------------------
# LOAD MODEL + TOKENIZER + ENCODER (cached)
# ------------------------------------------------------------
@lru_cache()
```

```
def load_emotion_model():
    device = "cuda" if torch.cuda.is_available() else "cpu"

    model =
AutoModelForSequenceClassification.from_pretrained("emotion_model").to(device)
    tokenizer = AutoTokenizer.from_pretrained("emotion_model")
    encoder = joblib.load("emotion_model/label_encoder.pkl")  # MUST match
training

    return model, tokenizer, encoder, device

# ------------------------------------------------------------
# HELPER — KEYWORD OVERRIDE + VADER FIXES
# ------------------------------------------------------------
def contains(words, text):
    t = text.lower()
    return any(w in t for w in words)

def final_emotion_logic(raw_label, text, scores):
    score = analyzer.polarity_scores(text)["compound"]
    t = text.lower().strip()

    # --- Strong keyword override ---
    if contains(["happy", "happier", "happiness"], t):
        return "joy"

    if contains(["glad", "thankful", "grateful", "appreciate"], t):
        return "gratitude"

    if contains(["stress", "stressed", "overwhelmed", "pressure", "deadline",
"assignment"], t):
        return "stress"

    if contains(["scared", "afraid", "worried", "anxiety"], t):
        return "fear"

    if contains(["angry", "annoyed", "frustrated", "pissed", "mad"], t):
        return "anger"

    # --- Sentiment-based overrides ---
    if score >= 0.4:
        return "joy"

    if score <= -0.4 and raw_label != "fear":
        return "sadness"

    # Neutral soft fix
```

```
    if raw_label == "neutral":
        if score > 0.2:
            return "joy"
        if score < -0.2:
            return "sadness"

    return raw_label

# MAIN MODEL INFERENCE
def predict_emotion(text: str):
    try:
        model, tokenizer, encoder, device = load_emotion_model()

        inputs = tokenizer(text, return_tensors="pt", truncation=True,
padding=True).to(device)

        with torch.no_grad():
            outputs = model(**inputs)
            probs = torch.softmax(outputs.logits, dim=-1)[0]

        label_id = torch.argmax(probs).item()
        raw_label = encoder.inverse_transform([label_id])[0]

        scores = {
            encoder.inverse_transform([i])[0]: float(probs[i].item())
            for i in range(len(probs))
        }

        # apply rules
        final_label = final_emotion_logic(raw_label, text, scores)
        confidence = probs[label_id].item()

        return {
            "emotion": final_label,
            "raw_label": raw_label,
            "confidence": float(confidence),
            "scores": scores,
        }

    except Exception as e:
        print("❌ ERROR in predict_emotion:", e)
        return {
            "emotion": "neutral",
            "raw_label": "neutral",
            "confidence": 0.0,
            "scores": {},
        }
```

```python
# ------------------------------------------------------------
# SENTIMENT SCORE (NEW)
# ------------------------------------------------------------
def compute_sentiment_score(text: str, mood_level: int, scores):
    vader_score = analyzer.polarity_scores(text)["compound"]

    joy_p = scores.get("joy", 0.0)
    sad_p = scores.get("sadness", 0.0)
    model_sentiment = joy_p - sad_p

    mood_factor = (mood_level - 3) / 2  # from -1 → +1

    final_score = (
        0.4 * vader_score +
        0.4 * model_sentiment +
        0.2 * mood_factor
    )

    return max(-1, min(1, final_score))


# ------------------------------------------------------------
# CROSS MODULE ACTION SUGGESTIONS
# ------------------------------------------------------------
def generate_cross_module_actions(sentiment, emotion):
    actions = []

    # Severe negative
    if sentiment <= -0.5:
        actions += [
            {"module": "tasks", "action": "create_self_care_task"},
            {"module": "community", "action": "join_support_group"},
            {"module": "forum", "action": "explore_similar_topics"},
        ]

    # Mild negative
    elif -0.5 < sentiment < -0.1:
        actions.append({"module": "tasks", "action": "small_break_task"})
        if emotion in ["sadness", "fear", "stress"]:
            actions.append({"module": "community", "action":
"join_wellness_group"})

    # Neutral zone
    elif -0.1 <= sentiment <= 0.1:
        actions += [
            {"module": "community", "action": "discover_new_groups"},
```

```
            {"module": "tasks", "action": "add_small_goal"},
        ]

    # Positive
    else:
        actions += [
            {"module": "tasks", "action": "progress_goal"},
            {"module": "community", "action": "share_positive_moment"},
        ]

    return actions

# ------------------------------------------------------------
# PARSE QUILL
# ------------------------------------------------------------
def extract_plain_text(delta_string: str):
    try:
        delta = json.loads(delta_string)
        return "".join(op.get("insert", "") for op in delta).strip()
    except Exception:
        return delta_string

# ------------------------------------------------------------
# MAIN ENTRY FOR DIARY ANALYSIS (NEW)
# ------------------------------------------------------------
def analyze_diary_entry(title: str, content: str, mood_level: int):
    text = extract_plain_text(content)

    pred = predict_emotion(text)
    emotion = pred["emotion"]
    scores = pred["scores"]

    sentiment = compute_sentiment_score(text, mood_level, scores)
    actions = generate_cross_module_actions(sentiment, emotion)

    return {
        "emotion": emotion,
        "sentiment_score": sentiment,
        "scores": scores,
        "suggested_actions": actions,
    }
```

4. Emotion-Based Diary Autocomplete Suggestion

The diary autocomplete component provides users with emotionally appropriate sentence suggestions based on the detected emotion. A large set of handcrafted writing templates is organised into emotion groups such as stress, sadness, joy, anger, fear, gratitude and neutral.

When the user begins writing, the system selects the correct emotion category and retrieves a random set of four templates from the corresponding list. The function also ensures the user's input ends cleanly before appending the suggested text, producing smooth and expressive sentence completions. This helps users articulate their feelings more accurately and reduces the effort needed to describe their emotional experiences.

```
import random

# Emotion-based sentence templates (sample only)
OPTIONS = {
    "stress": [
        {"style": "reflective", "text": "I've been trying to manage everything,
but the pressure keeps building."},
        {"style": "soft warm", "text": "I tried reminding myself to breathe, but
the stress still lingered."},
        {"style": "casual", "text": "Honestly, everything just felt a bit too
much today."}
    ],
    "sadness": [
        {"style": "reflective", "text": "I felt this heaviness follow me
throughout the day."},
        {"style": "soft warm", "text": "My heart felt a little fragile today."}
    ],
    "joy": [
        {"style": "bright", "text": "There were little sparks of happiness that
made the day lighter."},
        {"style": "casual", "text": "I actually felt pretty happy today, which
was nice."}
    ],
    "anger": [
        {"style": "raw honest", "text": "I felt irritation build up inside me
more than I expected."}
    ],
    "fear": [
        {"style": "soft vulnerable", "text": "I felt a quiet fear sit with me,
making things uncertain."}
    ],
    "gratitude": [
        {"style": "warm", "text": "I felt grateful for the small moments that
reminded me I'm not alone."}
    ],
    "neutral": [
        {"style": "simple", "text": "It was just a regular day with nothing out
of the ordinary."}
    ]
}
```

```
def autocomplete_diary(emotion: str, starter: str):
    emotion = emotion.lower().strip()
    starter = starter.strip()

    if not starter.endswith((" ", ".", ",", "!", "?")):
        starter += " "

    items = OPTIONS.get(emotion, OPTIONS["neutral"])
    chosen = random.sample(items, min(4, len(items)))

    return {
        "options": [
            {"id": f"{emotion}_{i}", "style": item["style"], "text": starter +
item["text"]}
            for i, item in enumerate(chosen)
        ]
    }
```

5.1.4 Task Emotion Classification Model and Unified Emotion Advisor Engine

This section explains the development of the emotion-aware analytics pipeline used to analyse user tasks and diaries. This module consists of three core components:

1. the Task Emotion Classification Model,
2. the Task Emotion Inference Service, and
3. the Unified Emotion Advisor Engine that fuses both task and diary emotional signals.

The following subsections describe the full implementation based solely on the developed code.

1. Task Emotion Classification Model (Fine-Tuned Transformer)

A DistilBERT-based model was fine-tuned using a custom BetterU dataset (betteru_tasks.jsonl) containing labelled task titles and descriptions. The model predicts one of seven emotion categories: joy, gratitude, sadness, anger, fear, stress, or neutral.

Key implementation steps include:

- Loading and encoding the task dataset using HuggingFace datasets.
- Encoding labels using a fixed class order via LabelEncoder.
- Tokenising input text with distilbert-base-uncased.
- Fine-tuning a AutoModelForSequenceClassification for 5 epochs.
- Evaluating performance using accuracy and macro-F1 score.
- Saving the trained model, tokenizer and label encoder into task_emotion_model/.

A simplified training code snippet is shown below:

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=len(CLASS_ORDER)
)
trainer = Trainer(model=model, args=args, train_dataset=enc)
trainer.train()
model.save_pretrained("task_emotion_model")
```

2. Task Emotion Inference Service

After training, the system provides a lightweight inference function that predicts the emotion of any task using the fine-tuned model. Both task titles and descriptions are combined to form the input text. This classifier is used whenever BetterU needs to evaluate the emotional load of a user's tasks.

Example of the inference logic:

```
inputs = task_tokenizer(text, return_tensors="pt")
outputs = task_model(**inputs)
pred = torch.argmax(outputs.logits)
emotion = task_encoder.inverse_transform([pred])
```

3. Unified Emotion Advisor Engine

This component combines:

- Task emotion patterns (from all tasks within weekly/monthly/yearly ranges)
- Sentiment scores from diary entries

The advisor pipeline performs:

1. Fetching tasks and subtasks from Firestore.
2. Analysing each task's emotional category, confidence, priority level, overdue status and subtask count.
3. Computing aggregated task metrics such as dominant emotion, average priority, overdue count and emotional distribution.
4. Fetching diary entries and computing:
    a. average sentiment score
    b. diary emotion distribution
    c. dominant diary emotion
5. Fusing both sources using a weighted emotional score:
6. final = 0.6 * task_score + 0.4 * diary_score
7. Producing personalised cross-module recommendations (tasks, community, forums).

Example rule:

```
if final_score <= -0.3:
    suggest("tasks", "create_self_care_task")
if dominant_task == "stress":
    suggest("tasks", "break_down_tasks_more")
if diary_sentiment >= 0.3:
    suggest("tasks", "progress_goal")
```

These suggestions help users manage workload and well-being in an integrated way.

5.2 Testing

5.2.1 Testing Strategies

Integration testing was adopted as the main testing strategy for the BetterU mobile application because each module consists of multiple interdependent components that must function cohesively. Rather than evaluating components in isolation, this testing approach verifies that the internal elements within each module interact correctly, exchange data as intended, trigger the appropriate logic, and collectively produce the expected results.

Integration testing for the **User Module** focused on validating authentication and user-preference workflows. This involved ***verifying the communication between Firebase Authentication, local session control, and Firestore user profiles to ensure accurate identity verification and secure data access***. Tests also confirmed that ***user settings, including preferences and profile details***, were successfully loaded into other modules and influenced their behaviours, such as task defaults, theme settings, and personalisation options.

For the **Goal Assistance Module**, integration testing was carried out across the ***entire task-processing workflow***. The real-time transcription produced by Vosk had to be accurately refined by the Whisper model, and the resulting text was then evaluated to ensure it was correctly interpreted by the NLP task-extraction engine. The structured task data generated from this process was validated to ***confirm that it could be applied to the task-creation form without any loss of information***. In addition, integration tests were conducted to ***verify that task editing actions correctly triggered the task-delay prediction process, updated the rescheduling suggestions, and refreshed the task recommendations accordingly***.

Integration testing for the **Focus Timer Module** validated the ***interaction between session timers, linked task data, and gamification features***. Tests ensured that starting a focus session correctly retrieved the related task, updated countdown timers, activated distraction-prevention logic, and synchronized session data with Firebase. Gamification components, including streak tracking and reward updates, were verified to ensure that they responded correctly to completed and interrupted sessions. The real-time updates rendered through Flutter state management were also tested to confirm smooth transitions between focus, break, and completion states.

Testing for the **Emotion Diary Note Module** examined ***diary entry handling, sentiment analysis, emotion classification, and privacy controls***. The end-to-end flow of processing diary content was validated, starting from Quill rich-text extraction through the emotion-classification model and sentiment-scoring engine. The diary auto-suggestion feature was also tested to confirm that it selected appropriate sentence templates based on the user's

detected emotion. Privacy features, including encrypted entries, access restrictions, and visibility options, were tested to ensure that the user's personal emotional records remained protected and hidden from unauthorized access.

5.2.2 Test Plan

Test Objectives

The objective of this test plan is to verify that each module operates correctly when its internal components are integrated and interacting with the related subsystems. The focus is to ensure that all workflows run smoothly, data flows correctly between modules, and the system behaves as expected under real usage conditions. The modules covered in this testing include the User Module, Goal Assistance Module, Focus Timer Module, and Emotion Diary Note Module.

Testing Process

Integration Testing

Integration testing was applied throughout this test plan because each module contains multiple internal components that must work together to support a complete end-to-end workflow. The main purpose is to validate cross-component interactions and confirm that integrated behaviours remain stable and consistent.

Examples of interactions validated include:

- User authentication and preference loading interacting with task creation workflows.
- Speech-to-text and NLP task extraction collaborating with task creation in the Goal Assistance Module.
- Focus session timers synchronizing task metadata, break prompts, and gamification counters.
- Diary sentiment analysis interacting with mood tagging and secure user privacy controls.

Through integration testing, the system ensures that combined behaviours of components across modules function reliably and accurately reflect real-world usage of the BetterU application.

Test Plan Sequence

1. User Module

   The User Module will be tested first because it provides the essential foundation required by all other modules. This includes user registration, authentication, profile

configuration, and session handling. Ensuring that user accounts and preferences function correctly is critical before testing any feature that relies on personalised settings or user identity.

2. Goal Assistance Module (shared)

   The Goal Assistance Module will be tested after the User Module because it supports the core productivity workflow of the BetterU application. It covers speech-to-text processing, task extraction using NLP, and task creation mechanisms. Other modules, such as the Focus Timer and Emotion Diary Note, rely on the tasks and data generated here. For example, the Focus Timer uses task details for session tracking, and the Emotion Diary Note may reference task-related emotional patterns.

3. Focus Timer Module

   The Focus Timer Module will be tested next because it depends on the task data produced by the Goal Assistance Module. This includes focus session control, countdown functionality, distraction prevention logic and gamification elements. By validating this module after tasks are already functioning, the integrity of session tracking and productivity workflows can be ensured.

4. Emotion Diary Note Module

   The Emotion Diary Note Module will be tested after the Focus Timer because it integrates both emotional analysis and user activity context. It includes diary entry management, sentiment analysis, mood tagging, and privacy control. Some insights generated here relate to task stress levels and productivity habits, making it suitable to test after earlier modules are stable.

Tested Items

- User Module:
  - User registration with email verification
  - User login and session token validation
  - Forgot password function sending a one-time password to the registered email
  - Loading user profile and preference settings
  - Updating user profile details
  - Updating user preference configurations
  - Session timeout handling and secure logout
  - Access control ensuring users only view their own data (tasks, diaries, sessions)

- Admin ability to view, edit, deactivate or delete members
- Admin functions for importing and exporting member data
- Admin tools for managing members, admin accounts, and reward settings
- Admin actions recorded in the system log

- Goal Assistance Module:
  - Integration of Vosk partial transcription with Whisper final transcription
  - Whisper transcription output forwarded into the NLP task extraction pipeline
  - Extracted task details mapped into the task creation form
  - Task editing triggering task delay prediction update
  - Task editing triggering task rescheduling suggestions
  - Task editing triggering task recommendation generation
  - Updated task information reflected in attachment selection lists used by other modules
- Focus Timer Module:
  - Start, pause, resume, and complete functions for focus sessions
  - Countdown timer accuracy and stable state management in Flutter
  - Linking focus sessions to the selected task
  - Pause action increasing the interruption counter
  - Gamification features including earn reward points from complete sessions, leaderboard ranking by reward points and reward redemption for profile borders
  - Distraction prevention logic where leaving BetterU triggers session cancellation
  - Updating task-related information upon session completion
- Emotion Diary Note Module:
  - Biometric access settings enabled in member preferences to prevent unauthorized entry
  - Extraction of diary content from Quill rich-text format
  - Emotion classification using the custom-trained emotion model
  - Sentiment score calculation based on model probabilities and mood level
  - Automatic sentence suggestions generated according to detected emotion
  - Saving diary entries with correct metadata such as mood, sentiment score, and timestamp

Testing Environment

Hardware and Devices

Mobile Phone	Device Model: POCO F5	● Processor: Qualcomm Snapdragon 7+ Gen 2 ● RAM: 12 GB ● Storage: 256 GB ● Operating System: Xiaomi HyperOS 1.0.18.0.UMRMIXM (Android 14) ● NFC Support: Yes
Laptop	Device Model: ROG Strix G614JV_G614JV	● Processor: 13th Gen Intel® Core™ i7-13650HX, 2.60 GHz (14-core) ● RAM: 32 GB ● Storage: 1 TB ● Graphics: ○ GPU 0: Intel® UHD Graphics ○ GPU 1: NVIDIA GeForce RTX 4060 Laptop GPU ● Operating System: Windows 11 Home (64-bit)

Backend and Services

- FastAPI backend running in development environments
- Google Firestore for storing all the BetterU mobile application's data
- Firebase Cloud Messaging for push notifications on user's mobile devices

Languages and Framework Environment

- Flutter 3.35.6 (channel stable)
- Dart 3.9.2
- Python 3.10.0

Network Environment

- Laptop connected to home mesh router via wired Ethernet
- Mobile device connected to Wi-Fi on the same mesh network

5.2.3 Test Cases and Results

User Module

Tester's Name	Chia Ming Yi	Date Tested	4-Dec-2025	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Stable internet connection
2	User has access to the BetterU mobile app
3	Test admin account is available
4	Email service (SMTP/Email API) is functional
5	Test email inbox available to verify OTP and verification email
6	App permissions granted (notifications, secure storage)

S #	Test Data
1	Member email to register: test0001@gmail.com
2	Member password to register: Test@12345
3	New password for reset: Abc123
4	Profile update data: new username, anonymous name, occupation, busy hours
5	Preference settings: Dark Mode ON, Biometric Lock ON, Chat Anonymity ON, Community Anonymity ON, Forum Anonymity ON
6	Admin email: test000@gmail.com; Admin password: Abc123
7	Member list import file (CSV)

Test Scenario	This scenario validates the complete User Module workflow, including registration, email verification, secure login with session token, password recovery using OTP, user profile editing, and full preference configuration (Dark Mode, Biometric Lock, Chat/Community/Forum Anonymity). It also includes access control validation, session logout, and all administrator capabilities such as viewing members, deactivating accounts, importing/exporting member lists, updating reward settings, and managing admin accounts. All changes should be recorded in the system logs.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open the BetterU app and tap “Go to Login”.	Login and Registration page is displayed successfully.	Login and Registration page was displayed successfully.	Pass
2	Enter a valid new email and password, tap “Register”.	New member are registered successfully.	New member was registered successfully.	Pass
3	Tap on the "Email Verification".	Verification email sent successfully to the registered email address.	Verification email was sent successfully.	Pass
4	Open the registered email inbox and check for verification email.	Registered email is received with a valid verification link.	Verification email received with valid link.	Pass
5	Tap the verification link.	New member is verified successfully.	Account verified successfully.	Pass
6	Return to the app and log in using a verified account.	Login successful; session token created.	Logged in successfully; token generated.	Pass

7	Tap “Forgot Password”, enter registered email, submit request.	OTP is sent to the registered email.	OTP email received successfully.	Pass
8	Enter OTP and set a new password.	Password is reset successfully.	Password reset successfully.	Pass
9	Navigate to Profile Page after login.	User profile and preference settings load correctly.	Profile and preference settings loaded correctly.	Pass
10	Edit profile details (username, anonymous name, occupation, busy hours) and tap “Save Changes”.	Profile details updated successfully.	Profile details updated successfully.	Pass
11	Update preference configurations: enable Dark Mode, enable Biometric Lock, enable Chat/Community/Forum anonymity.	Preferences saved; UI updates instantly (Dark Mode applied). Biometric Lock enabled for next login. Anonymity settings updated.	Preferences saved correctly; Dark Mode applied, Biometric Lock enabled, anonymity options updated.	Pass
12	Navigate to the Emotion Diary Note tab.	Biometric prompt appears	Biometric Lock activated; fingerprint required to proceed.	Pass
13	Tap the “Logout” button.	Session token invalidated, user redirected to login page.	Logged out successfully, token cleared.	Pass
14	Attempt to access another user’s data via UI (tasks/diaries/sessions).	Access is denied; only the user's own data is visible.	Access restricted successfully; only own data displayed.	Pass
15	Log in as Admin and view the Member List.	All members displayed correctly.	Admin successfully viewed the member list.	Pass
16	Admin deactivates a member account.	Member status changes to inactive; login blocked.	Members deactivated and blocked from access.	Pass
17	Admin imports member data using CSV.	Member data imported successfully with validation.	Member data imported successfully.	Pass
18	Admin exports member data.	Downloadable export file generated successfully.	Export file generated and downloaded successfully.	Pass
19	Admin updates reward settings (points, badges).	Reward settings updated and applied system-wide.	Reward settings updated successfully.	Pass
20	Admin updates other admin accounts.	Admin accounts updated successfully.	Admin accounts managed successfully	Pass
21	Verify administrative actions appear in system logs.	All admin actions are recorded with timestamps.	System logs updated correctly.	Pass

Goal Assistance Module

Tester's Name	Chia Ming Yi	Date Tested	5-Dec-2025	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	User is logged into the BetterU mobile app as member
2	Device microphone permission granted
3	Mobile device connected to same Wi-Fi network with the laptop connected network
4	Stable Internet connection

S #	Test Data
1	Speech input sentence = “Play badminton next Wednesday from 3 PM to 5 PM.”
2	Reminder time = 30 minutes before task start
3	Task description = “Weekly badminton practice with friends.”
4	Attachment (link) = "https://example.com/badminton-rules"
5	Attachment (image) = Any test image file
6	Subtask 1 title = “Prepare sports equipment”; Start: 2:00 PM; End: 2:30 PM
7	Subtask 2 title = “Warm-up session”; Start: 2:30 PM; End: 2:50 PM
8	Edited task time = Change to 4 PM - 6 PM
9	Edited category = Exercise
10	Edited priority = P3

Test Scenario	Creates a new task using the AI Task Extraction feature by recording a speech input, obtains the extracted task details, applies them into the task form, completes the remaining fields, and submits the task. Then, checks the Smart Advisor page for rescheduling suggestions and recommendations, edits the task to trigger updated predictions, and finally verifies that the created task is available in the Personal Chat attachment picker.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open the Goal Assistance module.	Goal Assistance dashboard is displayed successfully.	Goal Assistance dashboard was displayed successfully.	Pass
2	Tap the "+" button to create a new goal.	Create New Goal page is displayed with goal creation form.	Create New Goal page loaded with the goal creation form.	Pass
3	Tap on the "AI Task Extraction" card.	AI Task Extraction dialog is loaded.	AI Task Extraction dialog opened successfully.	Pass
4	Tap "Start Recording" to begin speech input.	Vosk begins showing partial real-time transcription.	Vosk began showing partial real-time transcription.	Pass
5	Speak the test sentence: "Play badminton next Wednesday from 3 PM to 5 PM."	Partial transcription updates dynamically during speech.	Partial transcription updated dynamically during speech input.	Pass
6	Tap "Stop Recording".	Whisper processes	Whisper processed the audio and	Pass

		audio and returns final accurate transcription.	returned a final accurate transcription.	
7	Tap "Extract Task Details".	NLP extracts task title, category, priority, start date/time and end date/time.	NLP successfully extracted title, category, priority, start time, and end time.	Pass
8	Tap "Apply" to insert extracted task details into the form.	Task creation form fields are automatically populated with extracted values.	Task form fields were populated automatically with extracted values.	Pass
9	Complete additional fields such as reminder time, description, links and attachments.	All entered details are accepted and validated by the form.	All additional details (reminder, description, links, attachments) were accepted and displayed correctly.	Pass
10	Add subtasks with subtask title, description, start date/time and end date/time.	Subtasks are successfully added.	Subtasks were added successfully with valid date and time ranges.	Pass
11	Submit the task using the submit button.	Task is created successfully and appears in the task list with all details intact.	Task was created and appeared in the task list with complete details.	Pass
12	Open the Smart Advisor page and check the Reschedule tab.	Task rescheduling suggestions are displayed based on task details.	Task rescheduling suggestions were displayed based on the task details.	Pass
13	Open the Task Recommendation tab.	Recommended tasks to complete first are displayed.	Recommended tasks to complete first were displayed successfully.	Pass
14	Edit the task by modifying the date, time, category or priority.	Delay prediction, rescheduling suggestions and recommendations update automatically.	Delay prediction, rescheduling suggestions, and recommendations updated according to edited task information.	Pass
15	Navigate to the Personal Chat module and select a conversation.	Chat interface loads successfully.	Chat interface loaded without issues.	Pass
16	Open the attachment picker and select the "Task" option.	The newly created task appears in the task selection list.	The newly created task appeared correctly in the task attachment selection list.	Pass

Focus Timer Module

Tester's Name	Chia Ming Yi	Date Tested	10-Dec-2025	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:
1	Logged into the BetterU mobile app as a member.
2	At least one task with subtasks exists
3	Device notification permission is granted for the BetterU app.
4	Stable internet connection and correct device date/time settings.

S #	Test Data
1	Selected Task = “Testing Project for ft”
2	Selected Subtask = “writing”
3	Focus duration = 5 mins
4	Initial total points = value shown in Focus header
5	Expected bonus points when no interruption = additional points (total shown as 55 pts on completion screen)
6	Interruption counter initial value = 0 for a new session.
7	

Test Scenario	Starts a focus session for an existing task and subtask, verifies the start, pause, resume, cancellation, and completion flows, checks countdown timer accuracy and state stability in Flutter, ensures that the focus session is correctly linked to the selected task, validates that pausing increases the interruption counter, confirms gamification behaviour (reward points, leaderboard, and ability to redeem profile borders), verifies distraction prevention logic where leaving the BetterU app cancels the session, and finally checks that task-related information, calendar views, and recent sessions are correctly updated after a completed session.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open the Focus Timer module from the bottom navigation bar.	Focus Timer dashboard is displayed with calendar, + Focus, Report, and Search / Filter buttons and the current points value in the header.	Focus Timer dashboard was displayed successfully with calendar, buttons, and points header visible.	Pass
2	Tap the “+ Focus” button.	Start Focus Session page is displayed.	Start Focus Session page was displayed successfully.	Pass
3	On the Start Focus Session page, select the task “Testing Project for ft” and subtask “writing”.	Selected task and subtasks appear in the session card.	Task “Testing Project for ft” and subtask “writing” were selected and shown in the card.	Pass
4	Select duration 5 min and verify the Points of this Focus: 50 pts label.	5 min duration is highlighted and points label shows 50 pts.	5 min option was highlighted and “Points of this Focus: 50 pts” was displayed correctly.	Pass
5	Tap “Start Focus”.	An ongoing focus session is created; Ongoing Focus Timer page opens showing selected task/subtask, 5-minute countdown,	Ongoing Focus Timer page opened with correct task, 5-minute countdown and controls. “Focus Started” notification appeared and a new recent session entry was created.	Pass

		bonus bar, and Pause / Stop controls. A “Focus Started” notification appears and the session is listed under Recent Sessions with correct duration.		
6	Observe the countdown for several seconds and then navigate back to the Focus dashboard and re-enter the ongoing session.	The countdown timer decreases smoothly in seconds; when leaving and re-entering the screen, remaining time is preserved (state not reset).	Countdown decreased correctly and showed the same remaining time after navigating away and back to the session screen.	Pass
7	Switch to another bottom navigation tab (e.g., Me), then return to the Focus tab.	Timer continues from the correct remaining time, confirming stable state management within Flutter navigation.	After switching tabs and returning, the timer continued from the correct remaining time without resetting.	Pass
8	Pull down the device notification shade during the running session.	A persistent notification shows “Focus Started” with the selected task and subtask, indicating the session is active.	Notification shade showed an active “Focus Started” notification with correct task and subtask details.	Pass
9	From the Ongoing Focus Timer screen, tap the Pause button.	Session is paused; countdown stops; bonus bar changes to indicate interruption; a “Timer Paused” notification is shown; on-screen banner updates to “Interrupted 1 time(s)” and interruption counter increases from 0 to 1.	Timer stopped at the current remaining time, “Timer Paused” notification appeared, and the screen showed “Interrupted 1 time(s)” with the interruption counter updated.	Pass
10	Tap the Play / Resume button to continue the session.	Countdown resumes from the paused time; session status changes back to running; a “Focus Resumed” notification is shown; bonus indicator shows that the no-interruption bonus is no longer active.	Timer resumed from the paused time, “Focus Resumed” notification appeared, and the bonus banner indicated the bonus state correctly.	Pass
11	While the resumed session is running, press the Home button or switch to another app for a short period, then open the notification shade.	Distraction prevention logic cancels the session because the user left BetterU; “Session Cancelled” notification appears; cancelled session is recorded appropriately (e.g., as cancelled) without awarding points.	Leaving the app triggered “Session Cancelled” notifications, and the interrupted session was marked as cancelled with no reward points granted.	Pass

12	Return to BetterU, go back to the Focus Timer module, and start a new focus session for "Testing Project for ft – writing" with 5 min duration, then tap Start Focus.	A fresh focus session begins with 5-minute countdown and bonus active (no interruptions yet).	A new focus session started successfully with a 5-minute countdown and active bonus banner.	Pass
13	Let the second session run until the countdown reaches 00:00, without pausing or leaving the app.	Session completes automatically; a "Focus Completed!" notification appears; a celebration screen is shown with confetti, stating that a 5-minute focus session was finished and 55 points (base + bonus) were earned.	The session ran to 00:00, "Focus Completed!" notification appeared, and the completion screen showed a 5-minute session finished with 55 points earned.	Pass
14	From the completion screen, tap "Go to Focus Overview".	Focus Timer dashboard opens; weekly view shows a block for the completed 5-minute session at the correct date/time; Recent Sessions list shows the new session with task name, date, time range, duration, and +50 pts / +55 pts entry.	Focus Overview displayed a new 5-minute block in the weekly grid at the correct timeslot and Recent Sessions listed "Task #Testing Project for ft • writing" with 5-minute duration and points gained.	Pass
15	Switch to Monthly view in the Focus Timer and tap the date of the completed session.	Tooltip or details show "5 mins focused" for the selected date; monthly statistics reflect the completed session.	Monthly calendar displayed the selected date with tooltip showing "5 mins focused" and statistics reflected the completed session.	Pass
16	Check total points in the Focus header and open Leaderboard & Points.	Total points increase by the reward amount from the completed session; leaderboard ranking updates according to the new total points.	Total points increased accordingly and leaderboard showed the updated ranking after earning focus points.	Pass
17	Navigate to the Profile module and verify member points and profile border rewards. Redeem or apply a profile border if available.	Profile points match the updated total from Focus Timer; profile border reward can be redeemed/selected and displayed around the profile photo, confirming reward redemption flow.	Profile showed the updated points total and the selected profile border was applied successfully around the profile photo.	Pass
18	Open the related task and review its details.	Task information is updated to reflect the new focus time (e.g., increased total focused minutes, recent session shown in history or	Task details reflected the additional focused duration and recent session, confirming successful linkage between focus sessions and the task.	Pass

		progress indicators), confirming the focus session is correctly linked to the selected task.		

Emotion Diary Note Module

Tester's Name	Chia Ming Yi	Date Tested	11-Dec-2025	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:	S #	Test Data
1	Logged into the BetterU mobile app as a member.	1	Diary Title = “good day”
2	Biometric access setting is enabled in Member Preferences	2	Diary Content (Quill format): *“Im happy to eat some food 😋”*
3	Stable internet connection	3	Mood Level = 😀 (Happy)
4	Device storage permission granted (for diary photos)	4	Photos = 2 test images

Test Scenario	User creates a new diary entry using the Emotion Diary Note Module. The system extracts and interprets Quill rich-text content, classifies the emotion using the trained emotion model, calculates sentiment scores, displays Smart Advisor suggestions, generates AI sentence suggestions, and saves the diary with correct metadata. The user then verifies the diary listing, deletes entries, and checks emotion analytics (breakdown chart, AI summary) and Smart Advisor integration.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open Emotion Diary Note Module from bottom navigation	Emotion Diary Note dashboard loads successfully	Emotion Diary Note dashboard displayed successfully	Pass
2	Face/Tap biometric prompt to enter diary section	Biometric authentication successful, user enters module	Authentication passed, diary module opened	Pass
3	Tap "+ Diary" button	“New Diary” editor loads with all fields	New Diary page loaded correctly	Pass
4	Set diary date and time	Selected date/time displayed on header	Date/time updated correctly	Pass
5	Enter title: “good day”	Title accepted and displayed	Title displayed correctly	Pass
6	Insert 2 photos into “Momentary Photo” section	Photos appear as thumbnails with remove (X) icon	Thumbnails loaded with remove icon	Pass
7	Type text in content: *“Im happy to eat some food 😋”*	Quill editor captures full text including emoji	Text captured successfully	Pass
8	Select Mood Level = 😀 Happy	Mood level indicator updates	Mood level highlighted as Happy	Pass
9	Tap “Analyze”	Backend processes text then returns detected emotion	Emotion detected: joy displayed at bottom	Pass
10	Tap “Auto-complete”	AI Suggestions panel appears with 4+ sentences	AI suggestions displayed correctly	Pass

11	Tap on one suggested sentence to insert	Suggested sentence appended to existing diary text	Sentence appended successfully	Pass
12	Tap ✓ to save entry	Diary is saved with correct metadata (title, mood, sentiment, photos, content, timestamp)	Diary saved with complete metadata	Pass
13	Return to diary list	Newly created diary appears under selected date	Diary entry visible in list	Pass
14	Verify thumbnail photos in diary card	Photos appear in correct order	Both thumbnails displayed	Pass
15	Open the diary entry	Full content appears with AI-added sentences included	All saved content displayed correctly	Pass
16	Tap “⋮” and tap “Delete”	Diary removed from list	Entry deleted successfully	Pass
17	Open Report page	Monthly/Weekly emotion metrics displayed	Report page loaded with analytics	Pass
18	Check Emotion Breakdown chart	Correct proportions shown (joy 50%, anger 50% based on dataset)	Chart displayed correctly	Pass
19	Scroll to AI Summary	Summary includes: dominant emotions, distribution, mood trend, suggestions	AI summary displayed fully and correctly	Pass
20	Open Smart Advisor and select Emotion tab	Emotion-based task insights appear	Insights loaded correctly	Pass
21	Verify sentiment score aggregated	Dashboard shows sentiment average and emotion counts	Data reflected accurately	Pass

5.3 Chapter Summary and Evaluation

This chapter provided a comprehensive explanation of the implementation and integration of the BetterU system. It described how the hybrid Vosk–Whisper speech-to-text pipeline enables accurate real-time transcription, how the NLP task extraction engine structures user input into meaningful task data, and how emotion analysis is accomplished through fine-tuned transformer models combined with sentiment logic. The Emotion Diary Note module, alongside the diary autocomplete feature, supports emotional expression and generates actionable cross-module insights. The Unified Emotion Advisor Engine further consolidates emotional signals from both tasks and diary entries to provide personalised recommendations.

In addition, the chapter outlined the integration testing strategy used to validate the workflows across all system modules, confirming that authentication, task creation, focus sessions and diary analytics operate cohesively. Overall, this chapter demonstrated the depth of the system's technical foundation and ensured that each module functions correctly within the broader BetterU ecosystem.

Chapter 6

Discussions and Conclusion

6 Discussions and Conclusion

6.1 Chapter Summary

This chapter presented a **comprehensive review of the achievements, contributions, limitations and problem-solving processes involved in the development of the BetterU mobile application.** The project was designed to address the fragmented nature of existing productivity and mental-well-being applications, where most systems focus on only one or two functions such as emotion tracking, task management or focus enhancement. The proposed solution was to develop an integrated platform that combines emotional monitoring, intelligent goal assistance, focus management and community interaction into a single cohesive system. This holistic approach supports users in managing both their emotional states and daily responsibilities, addressing the core problem of disconnected self-improvement tools.

The tools, techniques and methodologies adopted in this project were selected based on technical suitability and alignment with the system requirements. Firestore was chosen as the primary database because it provides real-time synchronisation and seamless integration with Flutter, which is essential for modules that frequently update user-generated data such as emotion diary note entries, forum interactions and focus sessions. FastAPI was selected as the backend framework due to its lightweight architecture, strong performance and compatibility with machine learning libraries required for AI-driven features such as speech transcription and NLP task extraction. **Methodologies such as modular development, clear folder structuring, incremental AI integration and consistent naming conventions were applied to ensure maintainability and scalability.**

Throughout development, the system achieved the core objectives of each major module. The Goals Assistance Module successfully implemented speech-to-task extraction, automatic schedule generation and adaptive rescheduling. The Emotion Diary Note Module enabled multi-format emotion input, mood tagging and trend analysis, while the Focus Timer Module provided distraction-free sessions, interruption detection and gamified motivation. These accomplishments demonstrate that the system meets its intended purpose of supporting productivity, emotional awareness and behavioural consistency. Strong planning and UI/UX design contributed to creating a clear and user-friendly experience, although the backend development required additional time due to the complexity of AI processing and data integration.

The chapter **also highlighted several key contributions of BetterU**, particularly its unique integration of emotional intelligence, productivity support, behavioural insights and

community features in one unified architecture. This aligns with current research trends in digital mental health, which emphasize multi-dimensional support and AI-driven personalisation. BetterU therefore presents a meaningful contribution by bridging emotional monitoring and task performance within a single ecosystem.

Despite these achievements, several limitations were identified. The backend is not yet deployed on a public cloud service, which restricts real-world testing and multi-user accessibility. Personalisation features and monetisation options are still limited, reducing opportunities for long-term scalability. Technical challenges such as Python dependency conflicts and the complexity of integrating speech and NLP models also affected development efficiency. These issues were addressed through environment restructuring, incremental AI development and improved backend configuration, but they remain areas for future enhancement.

In summary, the chapter reflects how BetterU successfully met its objectives while introducing innovative digital well-being features grounded in research evidence. It also demonstrates how the choice of tools, technologies and methodologies supported the system's development. Although several limitations remain, the solutions implemented and the proposed future improvements provide a clear pathway for extending BetterU into a fully deployable and scalable application.

6.2 Achievements

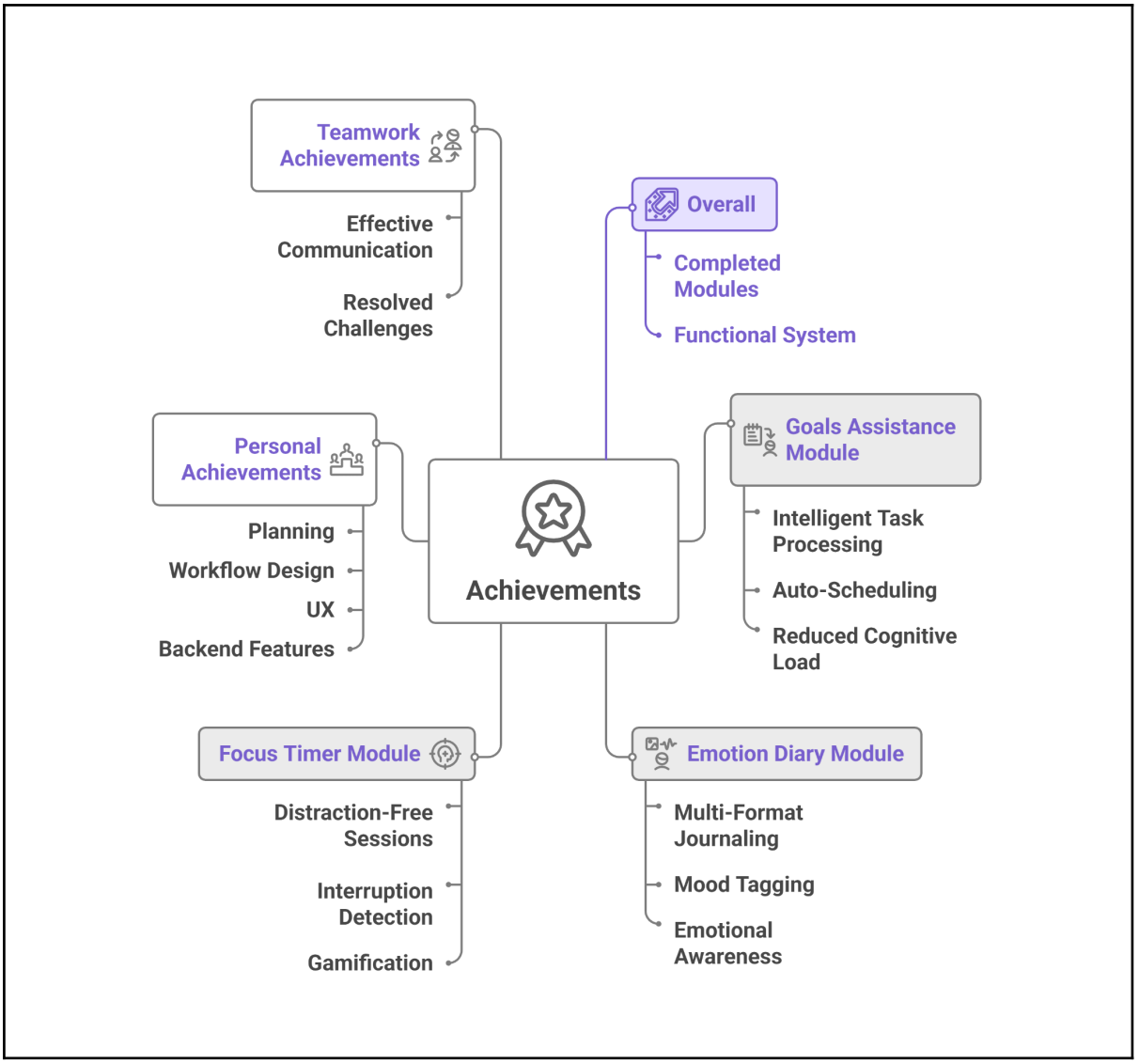


Diagram 6.2.1: Achievements Overview of BetterU Mobile Application

The current BetterU has successfully achieved the objectives defined for the BetterU application, and the modules under my responsibility were completed according to the planned functionality. Each module reached its intended goals and contributed to the overall purpose of improving productivity, emotional well-being and task organisation for users.

For the **shared Goals Assistance Module**, the main objective was to **provide intelligent task management through speech extraction, auto-scheduling and progress tracking.** These objectives were achieved as the module is able to convert user speech into actionable tasks, generate personalised schedules and adjust them when changes occur. This outcome demonstrates that the module is capable of reducing users' cognitive load and helping them organise their responsibilities more efficiently.

The **Emotion Diary Note Module** aimed to **support emotional awareness by allowing users to record emotions using text, speech or images, and by identifying mood trends.** This objective was fulfilled as the module successfully captures diary entries in multiple formats, applies mood tagging and analyses emotional patterns over time. Users can review these insights to better understand their emotional well-being.

For the **Focus Timer Module**, the **goal was to encourage sustained concentration and reduce distractions**. This objective was achieved through features such as distraction-free focus sessions, interruption detection and a gamification system that rewards consistent productivity. These functions help users build discipline and maintain focus during tasks.

Across the project, one of my strengths was planning, workflow design and ensuring that the user experience remained clear and intuitive. I was able to structure the modules well, maintain consistent design patterns and align features with user needs. **However, one of my weaknesses was backend development, which required more time and effort due to the complexity of the logic and the unfamiliarity of implementing advanced data processing.** Despite these challenges, **I managed to complete all assigned backend features, integrate them with the mobile app and ensure they worked as intended.**

There were also limitations during development, especially related to technical constraints, workload distribution and time pressure. Fortunately, **communication and collaboration with my teammate were strong, which allowed us to solve issues more effectively and maintain steady progress throughout the project.** This teamwork played an important role in overcoming challenges and ensuring the project was completed successfully.

Overall, the modules under my responsibility achieved their intended objectives, and the project was delivered as a fully functional system. The development process strengthened my technical skills, improved my problem-solving ability and gave me valuable experience in handling a multi-module application from start to completion.

6.3 Contributions

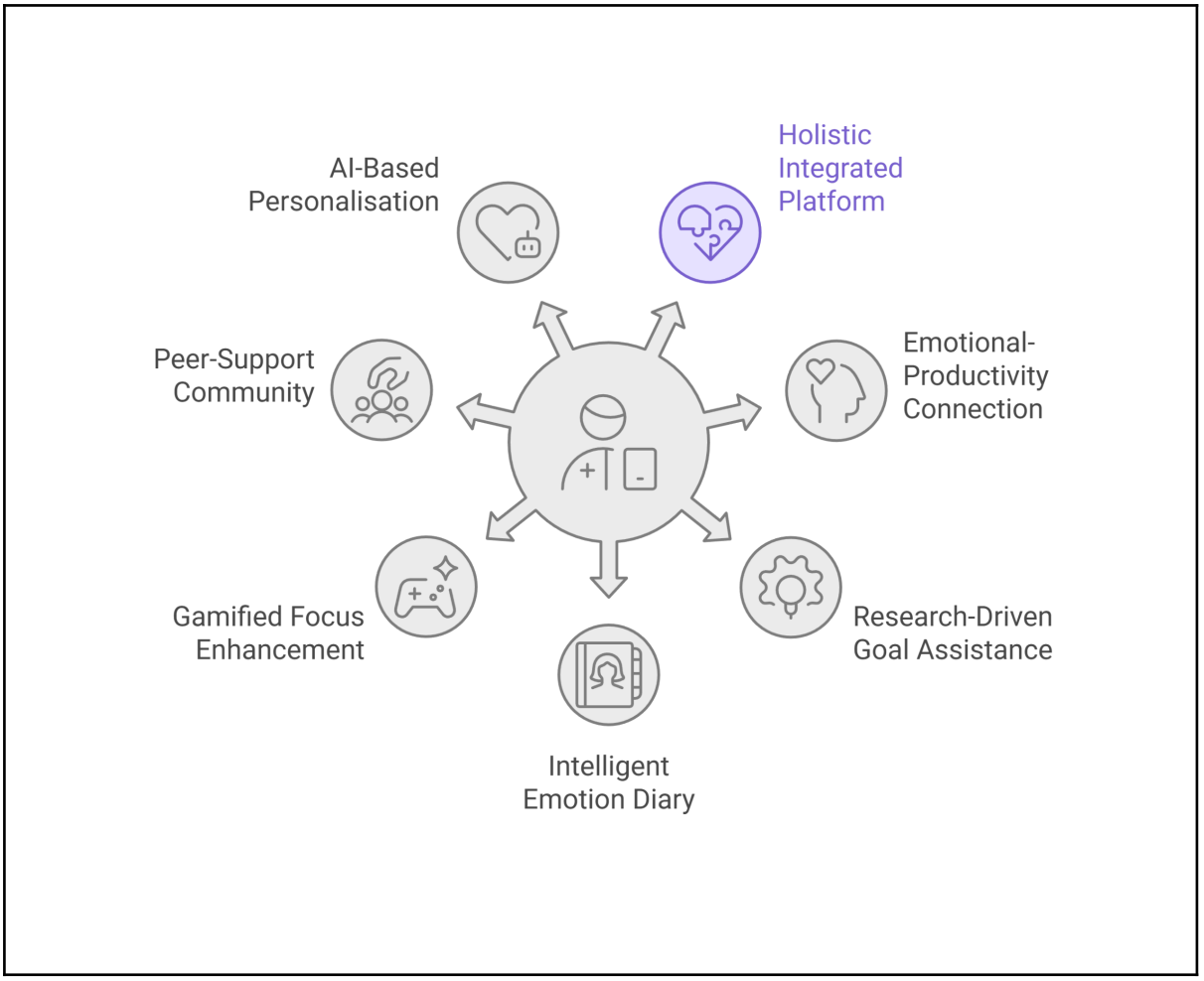


Diagram 6.3.1: Contributions Overview of BetterU Mobile Application

The BetterU system provides a significant contribution to the digital mental health and productivity domain by integrating multiple evidence-supported features into a unified and user-centric platform. Existing research shows that many mental health and productivity applications focus on singular functions, such as mood tracking, habit formation, distraction blocking, or social support, which often results in fragmented user experiences (JMIR Mental Health, 2020; PLOS Digital Health, 2023). BetterU contributes by overcoming this fragmentation through a holistic design that brings emotional well-being, productivity enhancement, behavioural guidance, and personal development into one cohesive system. This integrated model aligns with emerging directions in digital health research that emphasize multi-faceted support systems for improving daily functioning and psychological resilience (Frontiers in Public Health, 2017; JMIR, 2018).

1. **A Unified Emotional and Productivity-Driven Architecture**

   Research consistently highlights the strong connection between emotional states, task performance, and motivational stability (BMJ Open, 2024; Frontiers in Psychiatry, 2021). Despite this, many existing productivity apps do not incorporate emotional insight, and many

mood-tracking apps do not guide users in behaviour execution (Habitify, 2024; Verywell Mind, 2023). BetterU offers a creative contribution by connecting these two domains in a structured and meaningful way. Through emotional journaling, mood analysis, and behavioural insights, users gain a deeper understanding of how their emotional patterns influence their productivity and goal progress. This aligns with scientific findings suggesting that integrated models lead to stronger behavioural engagement and improved daily regulation (Kooth Insights, 2023; JMIR Mental Health, 2025).

2. **Goal Assistance and Behavioural Support Grounded in Research Evidence**

   Goal-setting frameworks have been shown to enhance mental well-being, decision-making, and self-management (Kooth Insights, 2023; IDTech, 2024). BetterU's Goals Assistance Module contributes by operationalizing these principles using AI-driven features such as speech-to-task extraction, schedule generation, dynamic rescheduling, and personalised task suggestions. These functions reduce cognitive effort and support users in making progress toward their goals, reflecting research demonstrating that structured digital goal interventions strengthen consistency and reduce planning fatigue (Frontiers in Public Health, 2017; PMC, 2024).

   By automating task breakdown and providing adaptive behavioural support, BetterU showcases a novel application of goal-setting science within a unified self-improvement ecosystem.

3. **Emotion Diary Note Intelligence and Reflective Mental Health Support**

   Emotional self-tracking tools increasingly adopt natural language processing and sentiment analysis to guide reflective practice (PLOS Digital Health, 2023; PMC, 2025). BetterU expands upon this research by enabling multi-format emotional journaling, trend visualization, and contextual emotional insights. Studies show that emotional reflection contributes to stress reduction, self-awareness, and improved coping strategies (JMIR Mental Health, 2021; PubMed, 2022). BetterU's contribution lies in combining emotional reflection with behavioural data, allowing users to understand how emotions relate to habits, tasks, and daily focus patterns. This creates deeper personal insight compared to traditional mood diary apps (PMC, 2021; JMIR Mental Health, 2025).

4. **Gamified Focus Management and Digital Behaviour Enhancement**

   Gamification is widely recognized for improving engagement, motivation, and adherence to digital interventions (ScienceDirect, 2025; PMC, 2017). BetterU's Focus Timer Module extends this concept by integrating distraction-free sessions with real-time interruption monitoring and reward-based elements such as treasure digging, avatar frames, and challenge points. Research supports that gamified systems encourage sustained behaviour and improve

productivity outcomes (JMIR Mental Health, 2021; PMC, 2023). The module's combination of behavioural reinforcement and emotionally contextualized focus tracking positions BetterU as an innovative productivity solution grounded in behavioural science.

5. **Community Interaction Inspired by Evidence-Based Peer Support**
   Peer-support environments have been shown to enhance emotional resilience, motivation, and sustained engagement in digital wellness tools (TalkLife, 2024; Digital Peer Support Database, 2023). BetterU contributes to this area by introducing a streamlined community interaction feature that supports sharing, encouragement, and positive engagement. This aligns with literature indicating that even minimal forms of peer connection can strengthen user well-being and encourage behavioural maintenance (WEconnect Health, 2024; PMC, 2018). The feature reflects modern digital mental health trends that incorporate social connectedness as a protective and motivating factor (UCDavis CHAT Lab, 2024).

6. **Advancement in AI-Driven Personalisation and Digital Coaching**
   AI-supported personalisation has become a growing trend in digital therapeutic tools, mental health platforms, and workplace coaching applications (LinkedIn Workplace AI Coach, 2024; arXiv, 2024). BetterU contributes creatively by integrating personalised recommendations, emotional insights, adaptive scheduling, and conversational guidance within its core modules. These elements demonstrate an application of persuasive digital health principles that enhance user motivation and behavioural reliability (JMIR Mental Health, 2025; ClinicalTrials.gov, 2024). This positions BetterU as an advanced, research-informed solution that aligns with emerging AI-driven wellness technologies.

BetterU stands out through its comprehensive integration of emotional monitoring, goal support, productivity management, reflective insights, gamification, and community connection. This unified architecture is supported by evidence-based practices drawn from digital mental health, behavioural science, and AI-driven personalisation research. **Rather than limiting users to isolated functions, BetterU delivers a seamless and holistic experience that supports personal growth from both emotional and behavioural perspectives.** This positions BetterU as a creative, complete, and impactful contribution within the digital self-improvement ecosystem.

6.4 Limitations and Future Improvements

1. **Backend Not Deployed on a Public Cloud Environment**

   At this stage of development, the FastAPI backend is only running locally on the developer's machine. Because it is not deployed to any public cloud environment, the mobile application cannot access backend services outside the development environment. This limits the system's ability to perform real-world testing, multi-user concurrency testing and performance monitoring under realistic conditions. In addition, external testers or potential users cannot interact with the system, which restricts usability evaluation and scalability assessment. The lack of a hosted backend also prevents features such as real-time updates, multi-device synchronisation and remote authentication from being evaluated in a true production environment.

   Improvement:

   Future development should focus on deploying the backend to a cloud hosting platform such as Render, AWS, Google Cloud or Azure. Cloud deployment will enable live access for users, allow real-time usage testing and support wider accessibility. These platforms also provide essential features such as scalability options, load balancing, logging services, performance monitoring dashboards and automatic backups. Any potential dependency conflicts can be resolved through environment isolation, the use of Docker containers or by adjusting the Python dependency list to match cloud requirements. Deploying the backend will significantly increase the system's reliability, testing accuracy and long-term viability.

2. **Limited Feature Set for User Monetisation and Premium Services**

   The current system does not include any form of payment module or subscription-based features. All functionalities are freely accessible and there is no mechanism to differentiate between standard users and premium users. This limits opportunities for revenue generation, which is an important factor for long-term sustainability and continuous system improvement. Without a proper monetisation structure, the project may face difficulty covering server costs, maintenance efforts and future feature expansion.

   Improvement:

   To support sustainable growth, a payment module should be introduced in future development. This could include subscription-based membership tiers such as normal and VIP users. VIP members may receive advanced analytics, exclusive content, additional customisation options or enhanced community privileges. Introducing monetisation not only supports system maintenance costs but also motivates continuous development. Secure payment gateways such as Stripe, PayPal or direct in-app purchases can be integrated to ensure a safe and smooth payment experience.

3. **Lack of Advanced AI-driven Personalisation**

   Although BetterU already incorporates AI for task extraction and emotion analysis, the system's personalisation features remain limited. The application does not yet provide adaptive recommendations, personalised growth plans or predictive analytics that respond to changes in user behaviour. As a result, the system's support remains generic rather than tailored to individual user needs.

   Improvement:

   In future versions, machine learning models can be used to generate personalized suggestions such as recommended tasks, mood-improvement activities, study habits or behavioural insights based on user patterns. Predictive analytics can also enhance the goal assistance module by forecasting potential delays or identifying stress-related behaviour. These improvements will lead to a more engaging and supportive user experience.

6.5 Issues and Solutions

1. **Difficulty in Selecting the Appropriate Database and Project Architecture**

   At the early stage of the project, determining a suitable database and designing the overall system architecture became a major challenge. The market offers a wide range of cloud databases, each with different capabilities in terms of scalability, real-time data handling, platform support and pricing. Selecting a solution that could support the requirements of modules such as the Emotion Diary Note, Focus Timer, Community Forum and Goal Assistance Module required extensive comparison and evaluation. Furthermore, defining a consistent coding structure and identifying the most suitable architectural pattern for both Flutter and FastAPI demanded substantial research and careful planning.

   Solution:

   After evaluating several options, Firestore was chosen as the primary database because it supports real-time synchronisation, provides cross-platform compatibility and integrates smoothly with Flutter. These features align well with modules that rely on frequently updated data, such as emotion logging and community interactions. For backend processing tasks, especially those involving AI, FastAPI was selected due to its lightweight design, high performance and excellent compatibility with Python-based machine learning libraries. Clear folder structures, modular code organisation and consistent naming conventions were also established at the start to ensure maintainability and long-term scalability of the entire BetterU system.

2. **Challenges in Integrating Flutter with the FastAPI Backend**

   Establishing a stable connection between the Flutter frontend and the FastAPI backend proved to be more complex than initially expected. While the connection worked smoothly in the emulator, testing on real Android devices introduced several issues. Common problems included network communication failures, Cross-Origin Resource Sharing (CORS) restrictions and accessibility limitations related to the local IP address. These issues caused interruptions when modules such as emotion analysis, task submission and forum interactions attempted to communicate with the backend.

   Solution:

   To resolve these issues, the backend was updated with appropriate CORS configurations to allow secure request handling from the Flutter app. The correct device-accessible IP addresses were used instead of the emulator's loopback address. API testing tools such as Postman were also utilized to verify request responses from each endpoint. These adjustments significantly improved the stability of communication between the mobile application and the backend services.

3. **Complexity in Developing AI-related Features**

   The development of AI functionalities presented one of the most challenging aspects of the entire project. With limited prior exposure to machine learning, it was difficult to understand how various AI components needed for the system should interact. For example, the Goal Assistance Module required NLP-based task extraction, Vosk and Whisper, and other analytic features required text processing and recommendation logic. Understanding these technologies, preparing the data flow and integrating them into the overall system required significant time and effort.

   Solution:

   To overcome these challenges, a large amount of self-learning was carried out through documentation, research papers and online tutorials. Instead of building all AI features at once, a gradual and modular development approach was adopted. Lightweight models were tested first before integrating more complex ones. Each AI function, such as transcription, NLP processing or task prediction, was developed as a smaller standalone component before being combined into the final pipeline. This incremental approach made debugging more manageable and ensured that each AI module worked correctly before moving to the next stage.

4. **Python Dependency Conflicts During Backend Development**

   During backend development, several Python libraries required for AI processing did not function correctly with the initially selected Python version, which was Python 3.13. Many machine learning and NLP-related dependencies either failed to install or produced runtime errors due to incompatibility issues. This resulted in an unstable development environment, especially for modules involving transcription, NLP and emotion analysis.

   Solution:

   The backend environment was reconfigured using Python 3.10.0, which provides much better compatibility with major AI libraries such as spaCy, Whisper, Vosk and SetFit. A clean virtual environment was created, and specific library versions were pinned to prevent version conflicts. This provided a stable and predictable environment for the development of AI-intensive modules and ensured smoother integration of machine learning features into the overall system.

6.6 Conclusion

The BetterU mobile application successfully achieves its objective of **providing an integrated platform that enhances users' productivity and emotional well-being**. The system **addresses core challenges such as cognitive overload, inconsistent task organisation and limited emotional awareness by combining several key modules under a single application**. Through the Goal Assistance Module, the system leverages AI-driven task extraction and intelligent scheduling to reduce users' planning effort and support more structured daily routines. The Emotion Diary Note Module enables users to record and reflect on their emotions through text, speech or images, allowing them to identify mood patterns and develop greater emotional insight. The Focus Timer Module promotes sustained concentration through distraction-free sessions, interruption detection and gamified motivation.

The development applied an **incremental model**, which allowed each module to be built, tested and refined in manageable stages. This approach **ensured that every feature aligned closely with user needs and worked reliably with the backend and database components.** While certain limitations still exist, such as backend deployment constraints and the lack of advanced personalisation, the overall outcome demonstrates that BetterU is a meaningful and impactful solution. It **has the potential to support students and working adults in managing their tasks more effectively, maintaining emotional wellness and staying motivated through a well-designed and user-centred digital ecosystem**.

References

Abdul Rashid, M. R., Syed Mohamad, S. N., Tajjudin, A. I. A., Roslan, N., Jaffar, A., Mohideen, F. B. S., Addnan, F. H., Baharom, N., & Ithnin, M. (2023). COVID-19 pandemic fatigue and its sociodemographic, mental health status, and perceived causes: A cross-sectional study nearing the transition to an endemic phase in Malaysia. *International Journal of Environmental Research and Public Health, 20*(5), 4476. Retrieved June 18, 2025, from https://doi.org/10.3390/ijerph20054476

Abdul Yazid, N. N. (2023). *Prevalence of mental health help-seeking attitudes among public and private universities students: A case study in Petaling*. Academia.edu. Retrieved June 20, 2025, from https://www.academia.edu/118504834/Prevalence_of_Mental_Health_Help_Seeking_Attitudes_among_Public_and_Private_Universities_Students_A_Case_Study_in_Petaling

Ahmad Adlan, A. S. (2022). *Innovative educational initiatives to train psychodynamic psychiatrists in underserved areas of the world*. Academia.edu. Retrieved June 20, 2025, from https://www.academia.edu/87468843/Innovative_Educational_Initiatives_to_Train_Psychodynamic_Psychiatrists_in_Underserved_Areas_of_the_World

Airbyte. (n.d.). *Firebase vs Firestore: A comparison*. Retrieved June 20, 2025, from https://airbyte.com/data-engineering-resources/firebase-vs-firestore

Amplework. (2024). *AI automation for repetitive tasks and resource efficiency*. Retrieved June 20, 2025, from https://www.amplework.com/blog/ai-automation-repetitive-tasks-resource-efficiency/

Android Developers. (n.d.). *Create and manage virtual devices*. Retrieved June 20, 2025, from https://developer.android.com/studio/run/managing-avds

Anuyat. (n.d.). *Using Flutter for health and wellness apps: A game-changer in user experience*. Retrieved June 20, 2025, from https://anuyat.com/flutter-health-wellness

arXiv. (2024). *Artificial intelligence–driven workplace coaching models and productivity support* (Preprint 2406.07485). Retrieved December 8, 2025, from https://arxiv.org/abs/2406.07485

Bahmad, H. (2021). *PTSD in the COVID-19 era*. Academia.edu. Retrieved June 20, 2025, from https://www.academia.edu/104673949/PTSD_in_the_COVID_19_Era

Bateman, P. (2024). *App fatigue: How too many apps are hurting productivity*. LinkedIn. Retrieved June 20, 2025, from https://www.linkedin.com/pulse/app-fatigue-how-too-many-apps-hurting-productivity-paul-bateman-jr--in7ce

BMJ Open. (2024). *Self-directed digital interventions for improving emotional regulation: Systematic review* (Article e081556). Retrieved December 8, 2025, from https://bmjopen.bmj.com/content/14/4/e081556

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. Retrieved June 20, 2025, from https://doi.org/10.1023/A:1010933404324

Chamrah, Y. (2023). *SetFit unpacked: When sentence transformers go to classification gym*. Medium. Retrieved June 21, 2025, from https://medium.com/@youssef.chamrah/setfit-unpacked-when-sentence-transformers-go-to-classification-gym-52f42f5a0d8e

Clustox. (2023). *Mood tracker apps: Features, challenges, and market potential*. Retrieved December 8, 2025, from https://www.clustox.com/blog/mood-tracker-apps/

Code Carbon. (2023). *Major advantages and disadvantages of Dart language*. Retrieved June 20, 2025, from https://codecarbon.com/major-advantages-and-disadvantages-of-dart-language/

Dart. (n.d.). *Dart overview*. Retrieved June 20, 2025, from https://dart.dev/overview

Daylio. (2025). *Daylio mood and habit tracking app*. Retrieved December 8, 2025, from https://daylio.net

Design Project. (2023). *Figma collaboration and teamwork*. Retrieved June 20, 2025, from https://designproject.io/blog/figma-collaboration-teamwork

Digital Peer Support. (2023). *Digital Peer Support Database*. Retrieved December 8, 2025, from https://digitalpeersupport.org/peer-support-database/

Estuary Dev. (2023). *Firestore vs Realtime Database: Which to choose?* Retrieved June 20, 2025, from https://estuary.dev/blog/firestore-vs-realtime-database

Explosion AI. (2023). *spaCy: Industrial-strength NLP in Python*. Retrieved June 20, 2025, from https://spacy.io/

FastAPI. (n.d.). *FastAPI documentation*. Retrieved June 20, 2025, from https://fastapi.tiangolo.com

Flutter. (n.d.). *Flutter documentation*. Retrieved June 20, 2025, from https://flutter.dev/

Frontiers in Psychiatry. (2021). *Digital mental health tools for emotion and behaviour tracking: Systematic review*. Retrieved December 8, 2025, from https://www.frontiersin.org/articles/10.3389/fpsyt.2021.672397/full

Frontiers in Public Health. (2017). *Mental health and behavioural support through digital interventions: Public health perspectives*. Retrieved December 8, 2025, from https://www.frontiersin.org/articles/10.3389/fpubh.2017.00249/full

GeeksforGeeks. (2022). *GitHub Desktop*. Retrieved June 20, 2025, from https://www.geeksforgeeks.org/github-desktop/

Google. (n.d.). *Firebase overview*. Retrieved June 20, 2025, from https://blog.google/products/admob/firebase-expands-to-become-unified-app/

Google OR-Tools. (2023). *Operations research tools*. Retrieved June 20, 2025, from https://developers.google.com/optimization

Habitify. (2024). *Habitify habit and task tracker*. Retrieved December 8, 2025, from https://www.habitify.me

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. Retrieved June 20, 2025, from https://doi.org/10.1162/neco.1997.9.8.1735

Human Resource Development Fund (HRDF). (2024). *The state of work-life balance in Malaysia: A report on employee well-being (2023)*. Human Resource Development Fund. Retrieved June 20, 2025, from https://hrdcorp.gov.my/wp-content/uploads/2024/06/HRD-Corp-Annual-Report-2023-Digital-Version.pdf

IDTech. (2024). *Technology tools that help you achieve goals: Digital self-improvement systems*. Retrieved December 8, 2025, from https://www.idtech.com/blog/technology-to-help-you-achieve-your-goals

IJSRD. (2024). *Task prioritization issues in mobile productivity apps*. Retrieved June 17, 2025, from https://www.ijsrd.com/articles/IJSRDV12I90019.pdf

Java Code Geeks. (2022). *Why Flutter picked Dart: A deeper dive*. Retrieved June 20, 2025, from https://www.javacodegeeks.com/2022/02/why-flutter-picked-dart.html

Jimmy Viquez. (n.d.). *Figma resources*. Retrieved June 20, 2025, from https://www.jimmyviquez.design/resources/figma

JMIR. (2018). *Digital self-help applications for mental health: Review of capabilities*. Retrieved December 8, 2025, from https://www.jmir.org/2018/2/e20/

JMIR. (2025). *Privacy issues in mental health apps*. Retrieved June 20, 2025, from https://www.jmir.org/2025/1/e66762

JMIR. (2025a). *Emerging trends in digital behavioural health systems* (Article e69639). Retrieved December 8, 2025, from https://www.jmir.org/2025/1/e69639

JMIR. (2025b). *User engagement challenges in mobile mental health apps* (Article e79850). Retrieved December 8, 2025, from https://www.jmir.org/2025/1/e79850

JMIR Mental Health. (2020). *Smartphone-based self-monitoring for youth psychiatric disorders: Systematic review* (Article e17453). https://doi.org/10.2196/17453

JMIR Mental Health. (2021). *Mobile mental health interventions and mood tracking trends* (Article e20424). Retrieved December 8, 2025, from https://mental.jmir.org/2021/4/e20424/

JMIR Mental Health. (2025). *Emotion-aware digital wellbeing applications* (Article e67785). Retrieved December 8, 2025, from https://mental.jmir.org/2025/1/e67785

Kiesler, S., Rainie, L., & Madden, M. (2013). *Anonymity, privacy, and security online*. Pew Research Center. Retrieved June 20, 2025, from https://www.pewresearch.org/internet/2013/09/05/anonymity-privacy-and-security-online/

Kooth Insights. (2023). *The role of goal-setting in digital mental health interventions for young people*. Retrieved December 8, 2025, from https://connect.kooth.com/insights/the-role-of-goal-setting-in-digital-mental-health-interventions-for-young

Kula, M. (2015). Metadata embeddings for user and item cold-start recommendations. *RecSys 2015 Workshop on New Trends in Content-Based Recommender Systems (CBRecSys)*. Retrieved June 20, 2025, from https://doi.org/10.48550/arXiv.1507.08439

Liang, X., Wang, X., & Zhang, Y. (2018). A reinforcement learning approach to personalized schedule planning. *Proceedings of the AAAI Conference on Artificial Intelligence, 32*(1). Retrieved June 16, 2025, from https://doi.org/10.48550/arXiv.1801.00209

Lindy.ai. (2024). *What is an AI planner and why you'll want one*. Retrieved June 16, 2025, from https://www.lindy.ai/blog/ai-scheduling-assistant

Ly, K. H., Trull, T. J., & Ross, R. M. (2020). The role of digital mental health interventions in promoting psychological well-being: A systematic review. *Frontiers in Psychology, 11*, 1–14. Retrieved June 16, 2025, from https://doi.org/10.3389/fpsyg.2020.00123

Malaysian Employers Federation (MEF). (2023). *MEF Salary Survey for Executives 2023*. Malaysian Employers Federation. Retrieved June 16, 2025, from https://mefacademy.mef.org.my/publications/publication_info.aspx?ID=75

Microsoft. (2021). *Microsoft To Do*. Retrieved June 16, 2025, from https://todo.microsoft.com

Microsoft. (2022). *Visual Studio Code documentation*. Retrieved June 16, 2025, from https://code.visualstudio.com/docs

Ministry of Health Malaysia (MOH). (2024). *National Health and Morbidity Survey (NHMS) 2023: Mental health and the COVID-19 pandemic*. Ministry of Health Malaysia. Retrieved June 16, 2025, from https://www.moh.gov.my

MobileAppDaily. (n.d.). *Python for mobile apps development*. Retrieved June 20, 2025, from https://www.mobileappdaily.com/knowledge-hub/python-for-mobile-apps-development

MoldStud. (2023). *How to create a mobile app using Dart programming language?* Retrieved June 20, 2025, from https://moldstud.com/mobile-app-development/how-to-create-a-mobile-app-using-dart-programming-language/

Monterail. (n.d.). *Python for mobile app development*. Retrieved June 20, 2025, from https://www.monterail.com/blog/python-for-mobile-app-development

Monterail. (n.d.). *Why choose Flutter? 6 top reasons to use Flutter for mobile app development*. Retrieved June 20, 2025, from https://www.monterail.com/blog/why-flutter

Nelson, E. (2020). *Collaborative design tools: How Figma is changing the future of UI/UX design*. UX Planet. Retrieved June 16, 2025, from https://uxplanet.org/collaborative-design-tools-how-figma-is-changing-the-future-of-ui-ux-design-8fa42f81a1fa

Nielsen Norman Group. (2024). *Prioritization methods*. Retrieved June 16, 2025, from https://www.nngroup.com/articles/prioritization-methods/

Notion Labs Inc. (2021). *Notion*. Retrieved June 16, 2025, from https://www.notion.so

OpenAI. (n.d.). *Whisper speech recognition toolkit*. Retrieved June 20, 2025, from https://github.com/openai/whisper

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. Retrieved June 16, 2025, from https://dl.acm.org/doi/pdf/10.5555/1953048.2078195

Pew Research Center. (2013). *Anonymity, privacy, and security online*. Retrieved June 16, 2025, from https://www.pewresearch.org/internet/2013/09/05/anonymity-privacy-and-security-online/

PLOS Digital Health. (2023). *Digital mental health tools for self-reflection and emotional awareness: Systematic review*. Retrieved December 8, 2025, from https://journals.plos.org/digitalhealth/article?id=10.1371/journal.pdig.0000123

PMC. (2017). *Gamification and digital behaviour change: Review of applications*. Retrieved December 8, 2025, from https://pmc.ncbi.nlm.nih.gov/articles/PMC5526809/

PMC. (2021). *Emotion journaling and language analysis in mental health apps*. Retrieved December 8, 2025, from https://pmc.ncbi.nlm.nih.gov/articles/PMC8257937/

PMC. (2023). *Digital adherence and engagement in behavioural health systems*. Retrieved December 8, 2025, from https://pmc.ncbi.nlm.nih.gov/articles/PMC9331057/

PMC. (2024). *Youth digital mental health: Emotional trends and behaviour patterns*. Retrieved December 8, 2025, from https://pmc.ncbi.nlm.nih.gov/articles/PMC12641403/

ProcessMaker. (2024). *Repetitive tasks at work: Research and statistics 2024*. Retrieved June 16, 2025, from https://www.processmaker.com/blog/repetitive-tasks-at-work-research-and-statistics-2024/

PubMed. (2022). *Cognitive and emotional regulation using mobile health applications: Clinical review*. Retrieved December 8, 2025, from https://pubmed.ncbi.nlm.nih.gov/35499098/

PubMed. (2023). *Mobile distraction and productivity impairment research*. Retrieved December 8, 2025, from https://pubmed.ncbi.nlm.nih.gov/36826990/

PwC Malaysia. (2024). *Asia Pacific Workforce Hopes and Fears Survey 2024: Malaysia highlights*. PwC. Retrieved June 18, 2025, from https://www.pwc.com/my/en/assets/publications/2024/pwc-my-hopes-and-fears-malaysia-findings.pdf

Radford, A., Gao, L., & Goh, G. (2023). *Whisper: Robust speech recognition via large-scale weak supervision* [Technical report]. OpenAI. Retrieved June 16, 2025, from https://openai.com/research/whisper

Reddit. (2024). *Overwhelmed by decision fatigue – Help me pick a tool*. Retrieved June 16, 2025, from https://www.reddit.com/r/ProductivityApps/comments/1j6eype/overwhelmed_by_decision_fatigue_help_me_pick_a/

Reddit. (2025). *What self-help apps have genuinely made a difference?* (r/productivity discussion). Retrieved December 8, 2025, from https://www.reddit.com/r/productivity/comments/1kvnuzs/what_selfhelp_apps_have_genuinely_made_a/

ScienceDirect. (2025). *Behavioural reinforcement mechanisms in digital self-help systems*. Retrieved December 8, 2025, from https://www.sciencedirect.com/science/article/pii/S0005796725000531

Scikit-learn. (n.d.). *Scikit-learn documentation*. Retrieved June 20, 2025, from https://scikit-learn.org/

Smartsheet. (2023). *Workers waste a quarter of the workweek on manual, repetitive tasks*. Retrieved June 16, 2025, from https://www.smartsheet.com/content-center/product-news/automation/workers-waste-quarter-work-week-manual-repetitive-tasks

spaCy. (n.d.). *spaCy NLP library*. Retrieved June 20, 2025, from https://spacy.io/

TalkLife. (2024). *Peer-support digital mental health app overview*. Retrieved December 8, 2025, from https://intuitionlabs.ai/software/telepsychiatry-digital-mental-health/peer-support-apps/talklife

TeamDynamix. (2024). *Calculating the time wasted on repetitive manual tasks*. Retrieved June 16, 2025, from https://www.teamdynamix.com/blog/calculating-the-time-wasted-on-repetitive-manual-tasks/

Ticina, E., & Cheben, J. (2025). *The use of gamification in time management*. ResearchGate. Retrieved June 16, 2025, from https://www.researchgate.net/publication/389492629_The_Use_of_Gamification_in_Time_Management

UCDavis CHAT Lab. (2024). *Persuasive chatbot interventions for behavioural change*. Retrieved December 8, 2025, from https://chatrlab.ucdavis.edu/current-research/persuasive-chatbots-project

Unfolding the Advantages. (n.d.). *10 benefits of Flutter for app development*. Retrieved June 20, 2025, from https://unfoldingtheadvantages.com/benefits-of-flutter

Verywell Mind. (2023). *Best mood tracker apps of 2023*. Retrieved December 8, 2025, from https://www.verywellmind.com/best-mood-tracker-apps-5212922

Vosk API. (2020). *Offline speech recognition toolkit*. Retrieved June 16, 2025, from https://alphacephei.com/vosk/

Vosk API. (n.d.). *Vosk offline speech recognition toolkit*. Retrieved June 20, 2025, from https://github.com/Puddot/vosk-api

WEconnect Health. (2024). *Digital support communities for wellness and recovery*. Retrieved December 8, 2025, from https://www.weconnecthealth.io

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Retrieved June 16, 2025, from https://doi.org/10.18653/v1/2020.emnlp-demos.6

World Health Organization. (2023). *Mental health: Strengthening our response*. Retrieved June 20, 2025, from https://www.who.int/news-room/fact-sheets/detail/mental-health-strengthening-our-response

Appendices

Development Environment Requirements

- IDEs: Android Studio, Visual Studio Code
- Python 3.10.0
- Flutter 3.35.6 (Channel Stable)
- Dart 3.9.2

Backend Setup

1. Install Python 3.10.0 via the URL if not installed: https://www.python.org/ftp/python/3.10.0/python-3.10.0-amd64.exe
2. Install Node.js if not installed: https://nodejs.org/
3. Open Visual Studio Code.
4. Open the "betteru-frontend" project folder.
5. Create a python environment using command: python -m venv env310
6. Activate the environment from the project directory: .\env310\Scripts\activate
7. Install all the required python dependencies: pip install -r 310-requirements.txt
8. Start the backend server
    - If using emulator, enter this command: uvicorn main:app --reload
    - If using real device, enter this command: uvicorn main:app --reload --host 0.0.0.0 --port 8000

Frontend Setup

1. Open Android Studio
2. Open the "betteru-backend" project folder
3. Open terminal section in Android Studio
4. Clean up the flutter environment: flutter clean
5. Install all the packages required: flutter pub get
6. Establish connection with backend server:
    - If using an emulator, open the "dio_client.dart" file and replace the "DEVICE_LOCAL_BASE_URL" with "EMULATOR_BASE_URL".
    - If using a real device, make sure your real device is connected to the network same with your computer connected network. Check your computer connected IPv4 address by opening the windows terminal and entering "ipconfig". Then, open the ".env" file and replace the value of DEVICE_LOCAL_BASE_URL with "http://[your-IPv4-address]:8000".
    - On the top panel, select your connected real device.

7. Click on the "Play" button to start the flutter application.
8. If your real device screen popped out the installation permission request, please allow them to proceed with the BetterU application installation.
9. Once the BetterU application has been installed on the device, it will be automatically started and opened.

User Account in BetterU Mobile Application

- Member Account:
    - Member 1:

Email Address	test0001@gmail.com
Password	Abc123

    - Member 2:

Email Address	test0003@gmail.com
Password	Abc123

    - Member 3:

Email Address	test0004@gmail.com
Password	Abc123

    - Member 3:

Email Address	fypbetteru@gmail.com
Password	Abc123

- Admin Account:

Email Address	test000@gmail.com
Password	Abc123

User Guide

1. Member Site
    - Login
        - Open the BetterU app
        - Enter email and password from the provided test member accounts.
    - Goal & Tasks
        - View tasks in Card or Calendar view.

        - Tap the bottom right floating button "+", select "Create Goal" to create a task.
        - Use AI Task Assistant to record voice and auto-fill task details.
    - Smart Task Advisor
        - Shows AI suggestions for task rescheduling and prioritisation.
        - Tap Apply Suggestion to accept recommended changes.
    - Time Usage Tracker
        - View daily screen-on time and focus duration.
        - Set Screen Time Limits and manage Whitelisted Apps.
        - Use Scan NFC to register an NFC tag or start a focus session.
    - Social (Personal and Group Chat)
        - Chat with friends or group members.
        - Send messages, images, tasks, posts, or diary entries.
        - Group creators can invite/remove members and toggle anonymous mode.
    - Q&A Forum
        - Browse questions, filter by tags, or search.
        - View question details and post answers with attachments.
    - Community
        - View joined communities.
        - Create a new community (name, tags, description).
        - Browse community posts and search by title or tag.
    - Reports
        - View task productivity and social performance summaries in Card or Chart view.
        - Export reports to PDF
2. Admin Site
    - Login
        - Open the BetterU app
        - Enter email and password from the provided test member accounts.
    - Community Moderation
        - Review community creation requests.
        - Approve or reject submissions.
    - Chat Moderation
        - Review reported chat messages.
        - Approve or remove messages

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.