Faculty of Natural and Mathematical Sciences
Department of Informatics

King's College London
Strand Campus, London,
United Kingdom

7CCSMPRJ

Individual Project Submission 2024/25

Name:	Rajas Shrinand Mahajan
Student Number:	K24051160
Degree Programme:	MSc Advanced Computing
Project Title:	Using AI Tools for Better Accessibility and Engagement in mHealth and Wellbeing Apps - Variant 2
Supervisor:	Angelos Georgoulas
Word Count:	13,342

Signature: [illegible]

Date: August 6, 2025

Abstract

The rise of mobile technology has created new opportunities for digital health applications focused on personal wellness. This project presents the design and development of *MoodSip*, an AI-powered Android application that helps users track hydration and meal patterns while generating personalized insights into their mood and energy levels.

Unlike existing apps, MoodSip uniquely combines daily logging with mood-energy analysis, offering users a holistic view of their wellbeing. In doing so, it becomes the only mobile application within its sector to combine all mentioned metrics in a single system. Built using Jetpack Compose in Android Studio, the app integrates Firebase for event logging and uses machine learning models hosted on Google Cloud Run. A key feature is the real-time backend engine that processes user data and returns natural language insights based on patterns in hydration and nutrition.

The system also includes weather-based hydration goal adjustments, animated UI elements, personalized reminders and an accessibility friendly interface. By combining mobile sensing, AI-powered analysis and real-time feedback, MoodSip demonstrates how intelligent wellness apps can support healthier daily habits.

This project contributes a scalable and extensible framework for future mHealth solutions aiming to deliver a similar type of guidance using explainable AI.

Acknowledgements

I would like to express my sincere gratitude to my university, **King's College London**, for providing the academic environment and resources that made this project possible. My deepest thanks go to my project supervisor, **Mr.Angelos Georgoulas**, whose invaluable guidance, feedback and encouragement consistently supported me throughout this research. I am also immensely grateful to my family for their unwavering love, patience and support during this journey. Their belief in me has been a constant source of strength and motivation. This work would not have been possible without the contributions and inspiration of all the individuals and systems acknowledged above.

Contents

List of Figures

List of Tables

Nomenclature

AI	Artificial Intelligence
BCS	British Computer Society
CCPA	California Consumer Privacy Act
CM	Confusion Matrix
CSV	Comma-Separated Values
GDPR	General Data Protection Regulation
GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrated Development Environment
LLM	Large Language Model
MIT	Massachusetts Institute of Technology
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-ViewModel
OEM	Original Equipment Manufacturer
PII	Personally Identifiable Information
PRC	Precision Recall Curve
REST API	Representational State Transfer Application Programming Interface
SDK	Software Development Kit
SUS	System Usability Scale
TLS	Transport Layer Security
UAT	User Acceptance Testing

UI	User Interface
UX	User Experience
XML	eXtensible Markup Language

Chapter 1

Introduction

The advent of mobile health (mHealth) applications represents a significant advancement in the healthcare domain, providing users with tools that enhance their health management capabilities through technology. These applications, particularly those enhanced by artificial intelligence (AI), can offer personalized experiences that respond dynamically to individual user needs, ultimately improving user engagement and health outcomes. This report focuses on the development of an mHealth app prototype that integrates essential features such as water intake logging and meal quality assessment while harnessing AI capabilities to improve overall user experience.

1.1 Background

Informatics, at the intersection of computer science, human-computer interaction and data-driven systems, plays a critical role in addressing real-world problems in health and wellbeing. One of the most active subfields within this space is mobile health (mHealth), which involves the delivery of health services and information via mobile technologies. The global rise in chronic health conditions and lifestyle-related diseases has accelerated the demand for scalable, user-friendly and intelligent health support systems[1]. These applications are increasingly being used to promote behaviour change, manage symptoms and track daily wellness indicators such as hydration, diet, mood and physical activity.

Despite the popularization of health-tracking apps, a significant number remain underutilized due to poor user experience, cumbersome interfaces and lack of intelligent personalization. This issue reflects a broader challenge in digital health: the gap between technological capability and real user needs[2]. Academic research has shown that user adherence to wellbeing apps is directly correlated with usability, engagement and responsiveness.[3] As a result, integrating Artificial Intelligence (AI) into mobile health apps presents an important frontier for improving both user satisfaction and health outcomes.

1.2 Motivation

Many users disengage from mHealth apps shortly after initial use due to overwhelming input demands, unintuitive design, or lack of perceived value. Common issues include tedious manual data entry, non-adaptive reminders and a failure to account for personal routines or accessibility needs[2]. These shortcomings may reduce the effectiveness of such tools and negatively affect user trust and long-term health adherence.

This project is motivated by the hypothesis that AI-driven interaction models can significantly improve the usability and perceived intelligence of mHealth systems. By applying technologies such as smart recommendations, real-time mood inference, hydration goals and intelligent logging systems, my project seeks to bridge the gap between passive data collection and active behavioural support. Such systems can empower users to engage meaningfully with their own health data while reducing cognitive and physical effort, thereby enabling more consistent health routines.

1.3 Aims

The primary aim of this project is to design and develop an AI-enhanced mHealth application prototype, named *MoodSip*, focused on hydration and nutrition tracking with embedded real-time AI insights. The system is intended to explore how artificial intelligence can simplify user interaction, personalize health prompts and generate wellness feedback in a mobile-first, user-friendly environment.

To achieve this, several specific objectives were set. First, I aimed to investigate the usability limitations of current mHealth applications, both through literature review and user observations, to understand where users struggle or disengage. Based on these insights, I then focused on designing an user interface for logging hydration and meals, ensuring that visual feedback, accessibility features and minimal interaction were maintained. A core part of the application involved integrating AI components—specifically, machine learning and generative models—to analyze user data and provide personalized insights, notifications and predictions based on behavior patterns. Another major objective was to assess the impact of this AI-driven feedback on user engagement, satisfaction and routine adherence. Finally, the project aimed to contribute meaningfully to the growing field of personalized digital wellbeing by offering a modular, extensible prototype that could evolve into a broader wellness platform.

Chapter 2

Literature Survey

2.1 Existing Systems in mHealth and Wellbeing Applications

Mobile health (mHealth) applications have seen widespread adoption due to the increasing penetration of smartphones and growing interest in self-care and wellness. Popular applications such as MyFitnessPal[4], WaterMinder[5], Samsung Health[6] and Cronometer[7] offer users tools to track hydration, meals, physical activity and other wellbeing indicators. These apps are used in both clinical and consumer settings to support health routines and chronic condition management.

2.1.1 Common Features

Based on extensive personal experience with these apps, I can say the typical functionalities include manual logging of water and food intake, which allows users to maintain a detailed and structured log of their daily consumption habits. These applications often feature graphical progress tracking, providing users with visual feedback on their hydration or calorie goals, which can increase motivation and consistency. Goal-setting and reminder systems are also widely implemented to promote adherence by notifying users to drink water or eat at regular intervals. Many apps incorporate caloric and nutritional estimation, automatically calculating the nutritional value of meals based on logged entries. Furthermore, integration with wearable devices such as Fitbit or Apple Watch enables continuous data syncing and passive tracking, enhancing the overall utility and personalization of the app.

2.1.2 Usability Limitations

Despite their widespread adoption, engagement with these apps tends to decline over time. Studies show that mHealth apps often suffer from *feature fatigue* where users especially older adults become overwhelmed with non-intuitive interfaces and lack of adaptability to personal habits[8]. Majority of these apps generally assume a one-size-fits-all model, ignoring users' individual context, mood or cognitive load, which leads to disengagement. Several studies have noted that repeated prompts from hydration and nutrition logging applications may be perceived as intrusive which reduces user satisfaction[9].

Another crucial limitation is tedious manual input. Users are often required to input specific values or search large food databases which becomes a cognitive burden over time.

This barrier is particularly pronounced for hydration tracking, where users are expected to log each glass of water. They implement static rules (e.g., *8 glasses of water per day*) without accounting for user history, environmental conditions, or behavioral patterns.

Accessibility is also a challenge. Many apps fail to provide support for diverse populations including the elderly, those with visual or motor impairments or users who rely on assistive technologies. This lack of inclusive design excludes a significant segment of potential users from mHealth benefits[10].

2.1.3 Lack of Intuitive Features and Intelligence

MyFitnessPal has increasingly placed core features like barcode scanning, meal insights and macro breakdowns behind a paywall, frustrating users who previously relied on them for free. Many also report data loss or missing historical logs after updates, eroding trust in the app's reliability[11].

While apps like Cronometer excel in precise nutrient tracking and offer a clean interface for logging food, they fall short when it comes to supporting broader wellness metrics like mood and energy. From personal experience, I have found that Cronometer lacks integrated tools to log emotional states or fatigue levels that significantly influence dietary choices and overall well-being. There are no mood diaries, sliders or tagging options to connect meals with how you feel afterward.

Additionally, the application offers very little in the way of visualizations related to physical or mental health beyond nutrition. There are no trend lines over time for mood or energy nor are there any personalized observations connecting these variables to eating and hydration habits. This makes it difficult to spot correlations such as whether specific meals consistently leave the user feeling sluggish or whether low energy is related to specific food categories. For users interested in a more holistic view of health that includes mental and emotional aspects, Cronometer's narrow focus can feel limiting.

2.1.4 Proposed System

This project proposes the development of an intelligent wellness application, **MoodSip**, that moves beyond traditional tracking. It integrates AI-driven insights into hydration and meal logging, offering natural language-based feedback and adaptive recommendations that are both timely and personalized. Rather than focusing solely on collecting raw health data, MoodSip is designed to interpret and reason about user behavior to actively promote better wellbeing.

Key Innovations

- **Natural Language AI Insights:** Instead of presenting static data logs, MoodSip uses a combination of large language models (LLMs) and custom-trained AI/ML models to process hydration and meal logs. These models analyze mood and energy trends, recognize behavioral correlations and deliver feedback in clear, natural language. This transforms numerical data into actionable, human-friendly insights that encourage sustained behavioral changes.

- **Context-Aware Hydration Goals:** MoodSip adjusts daily water intake recommendations by analyzing historical consumption patterns, temperature, humidity and time of day. For example, on hot days or after low-hydration periods, the app intelligently suggests increased water intake. This adaptive goal setting replaces generic *8-glasses-per-day* advice with a contextually intelligent hydration plan.

- **Minimal Input Design:** To address the usability gap present in many wellness apps, MoodSip introduces a visual, tappable glass-based UI for hydration tracking and simplified tagging for meals. Users can log with a single tap or swipe, reducing the effort required to maintain logs. The goal is to make the process as frictionless as possible while maintaining data quality.

- **Accessibility Features:** The interface design follows accessibility-first principles—featuring high-contrast visuals, large tap targets and minimal text input. In the future, I am planning to include voice-based interactions and haptic feedback, ensuring the app is usable by individuals with visual, motor or cognitive impairments, as well as older adults.

- **Behavioral Reasoning Engine:** At the core of MoodSip is an AI-powered engine capable of understanding complex interactions between hydration, nutrition, mood and energy levels. It identifies nuanced relationships—for example, how consistent hydration might mitigate the negative mood impact of junk food or how late-night eating patterns can reduce morning energy. These insights form the basis for personalized daily suggestions and wellness summaries.

2.1.5 Conclusion of Survey

In short, although existing mHealth solutions possess adequate tracking capabilities, they lack sophisticated reasoning, personalization and accessible design. MoodSip's foundation lies in behavioral informatics, aiming not only to log actions but to influence healthier habits. The integration of AI not only improves personalization but transforms the application into a proactive wellness assistant which is able to provide targeted and timely suggestions.

Chapter 3

Dataset Creation and Preparation

3.1 Data Collection Methodology

In order to enable intelligent reasoning over hydration, nutrition and wellness behaviors, this project required a highly specific dataset capturing not only the foods and fluids consumed but also user-reported metrics like mood and energy. Upon reviewing existing mHealth and behavioral datasets, it became clear that no publicly available source combined meal data, hydration logs and emotional state annotations in a structured, timestamped manner. Given this gap, I undertook the task of creating a custom dataset, carefully designed to reflect real-world patterns and nuanced interactions between hydration, food intake, mood and energy levels.

The data was collected over a continuous period of 120 days through a combination of mobile applications and manual entries. Hydration and meal data were logged using the Cronometer app[7], which I chose for its reliable timestamped logging system. Alongside this, a parallel process was followed to capture subjective states: mood and energy levels were recorded before and after each meal using a structured Google Form deployed daily , utilizing a 5-point Likert scale (1 = Very Low, 5 = Very High). These responses were exported in CSV format and integrated with the meal and hydration logs.

3.2 Preprocessing and Feature Engineering

To convert the raw data into a usable form for modeling and analysis, an extensive preprocessing pipeline was developed using Python's `pandas` and `datetime` libraries within a Jupyter Notebook environment. First, hydration and meal logs were timestamp-aligned and merged by date. Manual mood/energy logs were joined based on time proximity to meals.

To transform these raw entries into insightful features, an extensive data processing pipeline was developed. This pipeline generated high-level attributes such as hydration status per day, hydration timing spread (measured using the standard deviation of intake times) and junk food ratios based on food categories. It also included derived metrics such as meal timing lateness — flagging lunch after 3 PM or dinner after 9 PM — and skipped meal detection by analyzing missing entries for known expected meal times. Weather data for each day was fetched via the OpenWeatherMap API, adding environmental context to hydration needs and goal personalization.

A particularly insightful derived feature was mood and energy variability. Using statistical aggregation, I computed the standard deviation in mood and energy scores across meals per day, capturing days of emotional instability or fatigue. Additionally, a hydration spread index was introduced to understand how well-distributed water intake was throughout the day rather than clustered at specific times, as often happens when users hydrate reactively.

Throughout the preprocessing phase, anomaly detection was applied to identify and handle implausible values. For example, entries where mood jumped from 1 to 5 within ten minutes for a single salad meal were flagged for verification. Weekly audits were performed by sampling 15 percent of the dataset and manually comparing logged data with original app records. These audits helped correct duplicate entries, timestamp inconsistencies and labeling errors.

3.3 Dataset Utility and Observed Patterns

The resulting dataset forms the backbone of my application. Unlike existing public health datasets, this dataset offers a unique combination of timestamped hydration events, meal classifications, mood-energy response pairs and weather context — all tied to real daily behavior.

Furthermore, the process of manually logging mood and energy before and after meals introduced granularity that automated apps rarely capture, providing a goldmine for pattern analysis and insight generation.

Finally, this dataset not only served as a training and inference base for my AI models but also offered real-world insight into behavioral tendencies. I observed for example, hydration was frequently done reactively rather than proactively and that energy levels often dipped after meals perceived to be emotionally comforting but nutritionally poor. Countless other findings laid the groundwork for the system's AI-driven insight generation logic.

Chapter 4

Specifications

4.1 Design

To bridge the gap identified in existing mHealth applications Section 2.1.2, particularly the lack of personalization and contextual intelligence, several architectural and behavioral diagrams were designed to underpin the proposed MoodSip system. These diagrams offer a structured justification of how each component contributes to this application.

4.1.1 Sequence Diagram 1: Meal Insight Flow

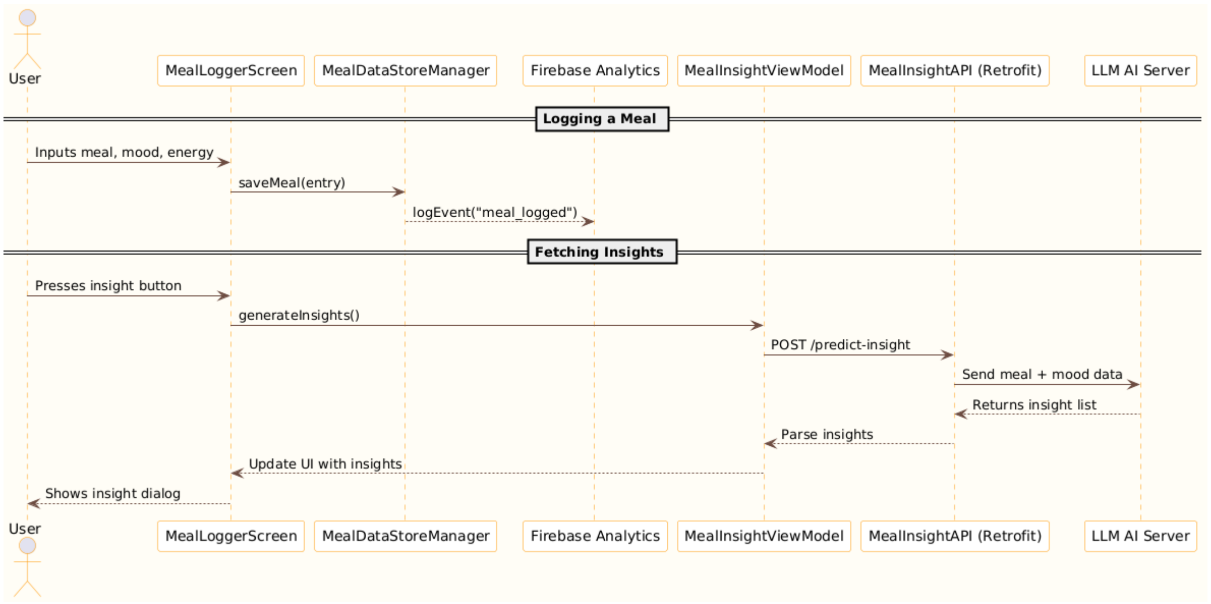


Figure 4.1: Sequence Diagram – Meal Logging and LLM Insight Generation Flow

This sequence diagram captures the flow triggered when a user logs a meal and requests insights. Compared to existing apps like Cronometer or MyFitnessPal, where food data remains passive, MoodSip actively interprets the data using a Large Language Model (LLM).

The sequence starts from the `MealLoggingScreen`, where a user submits meal and mood data. This is routed to the `MealInsightViewModel`, which in turn queries the local `MealDataStoreManager` for recent logs. The core rationale here is to create a lightweight, real-time inference flow. Upon fetching relevant data, the view model asynchronously invokes the `LLMInsightApi` via Retrofit to retrieve a generated interpretation of mood-energy-food correlations.

This embodies the innovation mentioned in Section 2.1.4 —transforming static logs into actionable feedback using natural language instead of charts.

4.1.2 Sequence Diagram 2: Hydration Intelligence and Analytics Flow

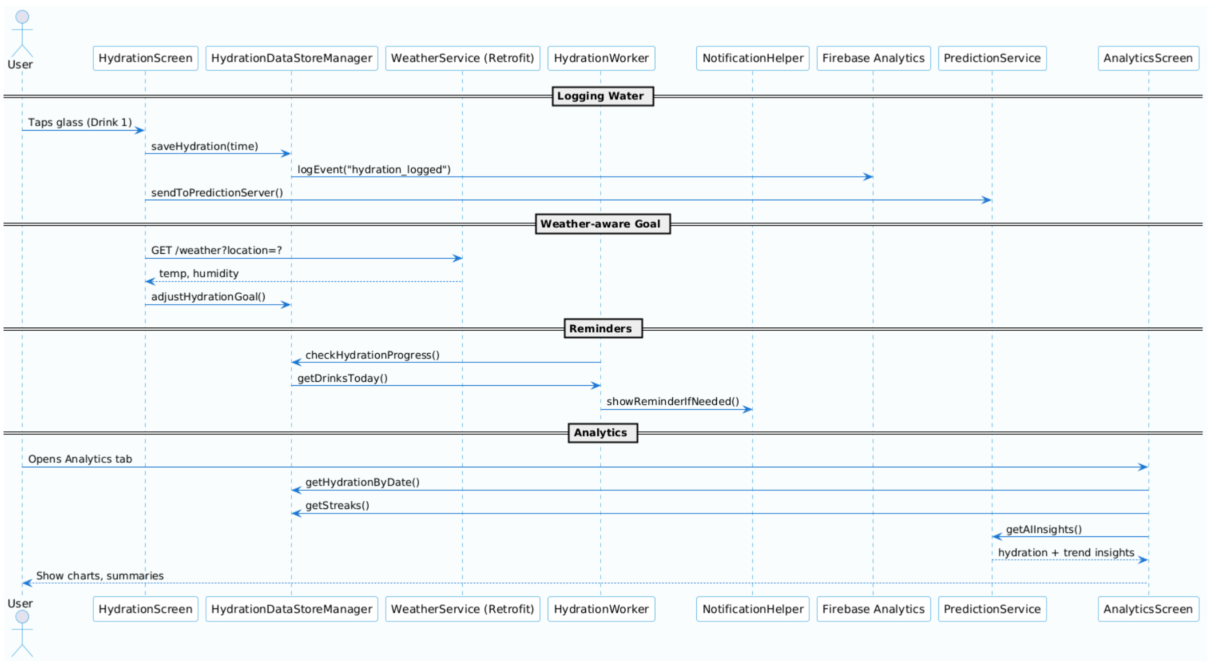


Figure 4.2: Sequence Diagram – Hydration Logging, Weather-Aware Goals and Analytics Insight

This diagram represents the hydration-related interactions across multiple subsystems. We interact with the `HydrationScreen`, tapping glasses to log intake. This data is written to the local `HydrationDataStoreManager` and is logged with Firebase Analytics. At scheduled intervals, a background worker (`HydrationReminderWorker`) wakes up to fetch the latest weather data via the `WeatherService`, reads hydration history and contacts the `PredictionService` to dynamically adjust the user's hydration goal.

The insight generation is triggered when the user opens the `AnalyticsScreen`, which compiles recent mood, energy and hydration data and calls the `InsightApi` for trend extraction.

The rationale here diverts from the generic *8 glasses* advice. As highlighted in Section 2.1.2 and Section 2.1.4, MoodSip incorporates external contextual variables like temperature and time of day to provide adaptive hydration suggestions. It also links hydration behavior with mood analytics, an integration absent in existing apps.

4.1.3 Component Diagram

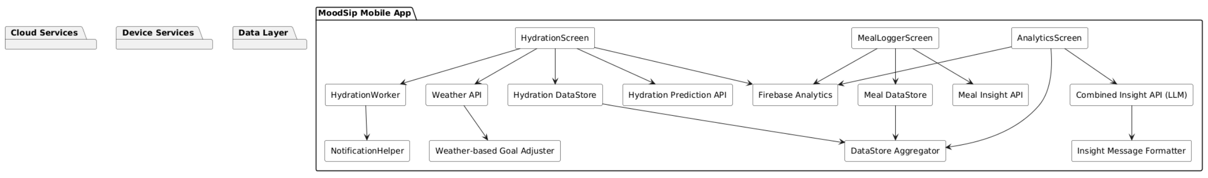


Figure 4.3: Component Diagram – Modular System Architecture of MoodSip

The component diagram provides a modular overview of the system. It divides the application into layers—UI, ViewModel, Local Storage, Remote APIs and Workers. Each are encapsulated into composable units.

Key differentiators in this structure include:

- **Dual Insight API Clients:** MoodSip maintains separate APIs for statistical insight extraction and LLM-based narrative generation, enabling hybrid intelligence as described in Section 2.1.4.
- **Shared ViewModel-Oriented Design:** Screens delegate logic entirely to view models, allowing asynchronous data transformations while keeping the UI reactive and clean.
- **Worker-Aware Intelligence Loop:** The hydration worker is not simply a notifier but a micro-intelligence unit that reasons with the user state before alerting which leads to adaptive reminders, resolving a key drawback of existing systems as mentioned in Section 2.1.2.

This ensures scalability, testability and aligns with accessibility goals, such as decoupling UI updates from logic and enabling voice or haptic inputs in future layers.

4.1.4 Use Case Diagram

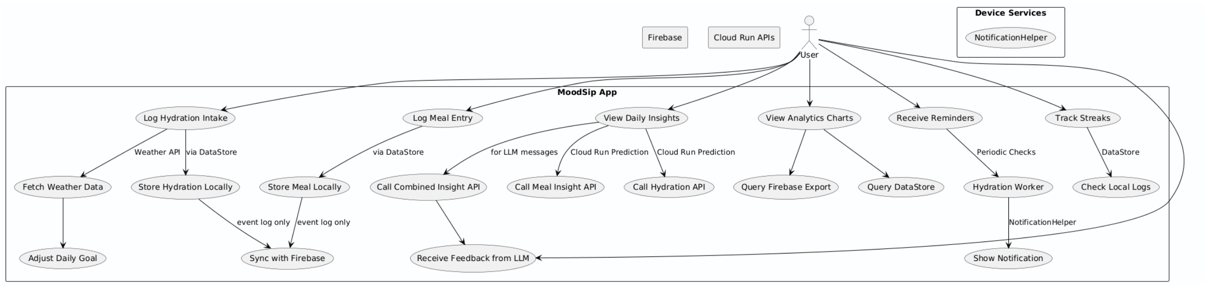


Figure 4.4: Use Case Diagram – User, Worker and AI Interactions in MoodSip

The core use cases in the MoodSip system can be grouped into four main categories. The first is **Active Tracking**, where users are involved in manually logging their hydration and meal data throughout the day. This also includes the adjustment of sliders to input their mood and energy levels before and after meals, enabling a personalized feedback loop.

Next is **Insight Generation**, where users can request both daily and weekly summaries through the dedicated analytics tab. These insights help them understand how their behaviors influence well-being over time, offering a reflective component to the app experience.

Another important category is **Adaptive Logic**. This refers to the app's ability to automatically update hydration goals based on current weather conditions and the user's past drinking patterns. Such adaptability ensures that recommendations remain relevant and personalized across changing conditions.

Finally, **Engagement Triggers** form the proactive component of the system. Users receive personalized reminders driven by AI-based predictions, nudging them to take action at critical points—such as when they're falling behind on hydration or likely to experience a mood dip. These triggers aim to sustain consistent engagement without being intrusive.

4.2 Architecture

This section outlines the reason behind selecting **Model–View–ViewModel (MVVM)** over other architectural patterns.

4.2.1 Comparison of Architectural Patterns

Model–View–Controller (MVC) was one of the earliest architectural patterns applied in mobile development. While conceptually sound, in Android practice it leads to "Massive View Controllers," where Activities or Fragments absorb both UI logic and business logic, resulting in tightly coupled, hard-to-maintain code[12].

Model–View–Presenter (MVP) improves separation by introducing a Presenter that handles presentation logic independently of the view. However, in large-scale apps, MVP introduces significant boilerplate and struggles with additional overhead, making it suboptimal for Compose and coroutine-based architectures [13].

Model–View–ViewModel (MVVM), in contrast, aligns cleanly with *Android Architecture Components*. It introduces a **ViewModel** that retains UI-related state across configuration changes and allows for one-way data binding and asynchronous flows. It supports observable patterns via `LiveData` or `StateFlow`, ensuring better testability and state management[12].

4.2.2 MVVM Justification for MoodSip

To manage MoodSip's stateful and reactive features efficiently, it adopts the Model-View-ViewModel (MVVM) architecture. The unidirectional data flow it enforces where the `ViewModel` emits state and the `View` consumes it, allows for a clean abstraction between UI and business logic. This architecture aligns naturally with Jetpack Compose's declarative style, enabling dynamic UI updates as the underlying state changes.

The app architecture also benefits from seamless integration with **DataStore**, which persistently stores user preferences and hydration logs, ensuring continuity between sessions. **Kotlin Coroutines** are extensively used for asynchronous operations such as hydration reminders and API fetches, keeping the UI responsive. Moreover, lifecycle awareness is maintained using `viewModelScope`, which ensures coroutine jobs are automatically cancelled when the ViewModel is cleared, thus avoiding memory leaks. Collectively, these architectural decisions ensure that MoodSip remains reactive, scalable and maintainable.

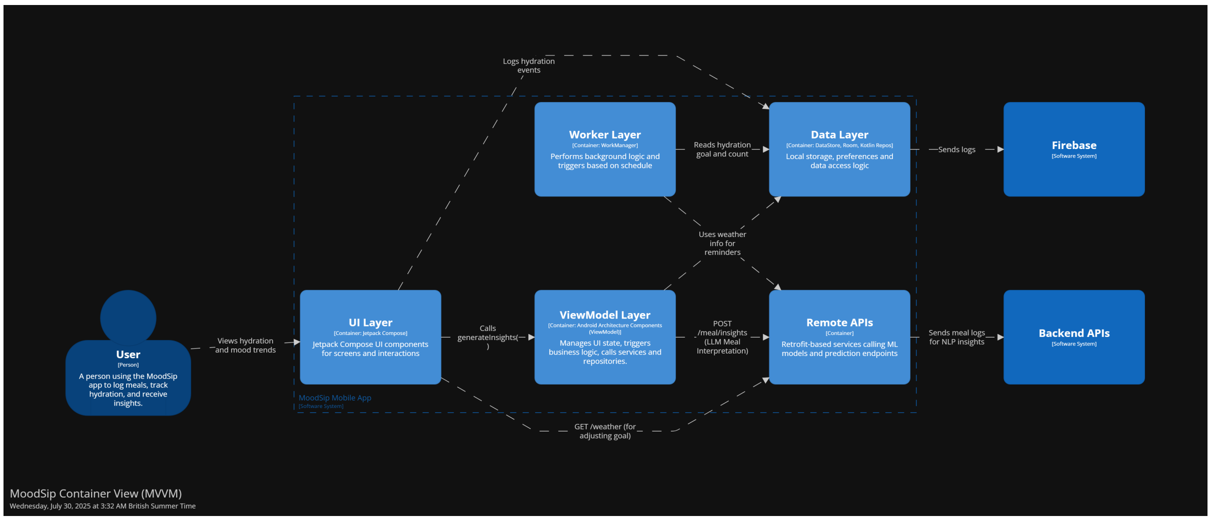

Logs hydration events
Worker Layer
[Container: WorkManager]
Performs background logic and triggers based on schedule
Reads hydration goal and count
Data Layer
[Container: DataStore, Room, Kotlin Repos]
Local storage, preferences and data access logic
Sends logs
Firebase
[Software System]
Uses weather info for reminders
User
[Person]
A person using the MoodSip app to log meals, track hydration, and receive insights.
Views hydration and mood trends
UI Layer
[Container: Jetpack Compose]
Jetpack Compose UI components for screens and interactions
Calls generateInsights()
ViewModel Layer
[Container: Android Architecture Components (ViewModel)]
Manages UI state, triggers business logic, calls services and repositories.
POST /meal/insights (LLM Meal Interpretation)
Remote APIs
[Container]
Retrofit-based services calling ML models and prediction endpoints
Sends meal logs for NLP insights
Backend APIs
[Software System]
MoodSip Mobile App
[Software System]
GET /weather (for adjusting goal)
MoodSip Container View (MVVM)
Wednesday, July 30, 2025 at 3:32 AM British Summer Time

Figure 4.5: MVVM Architectural Diagram

The implemented MVVM design in MoodSip maps as follows:

- **Model Layer**: Manages hydration and meal logs via `DataStore`, interacts with Firebase Analytics, weather API and LLM APIs.
- **ViewModel Layer**: Serves as the mediator. For example, `HydrationViewModel` manages hydration state and logic, while `MealInsightViewModel` queries the backend to generate daily insights.
- **View Layer**: Fully declarative using Compose. It subscribes to `StateFlow` or `collectAsState()` from ViewModels and remains free of business logic.

4.3 Requirements

This section contains a thorough analysis of each requirement and its relative significance to the project's completion. A high significance requirement must be implemented first to finish the project, followed by the other requirement effectively.

Requirement Details	**Specification**	**Significance**
The application shall support AI-driven insights.	The system should integrate backend AI models (LLMs and trained Python models) deployed via Google Cloud Run. These models must provide contextual recommendations and trend analysis on hydration, nutrition and mood.	High
The system shall support seamless hydration and meal logging.	Users should be able to log meals and hydration with minimal input using intuitive UI components such as tappable glasses, sliders and category selectors. Data entry must be fast and frictionless.	High
The system shall maintain persistent user data.	All hydration and meal logs should be stored using Android Jetpack DataStore for local persistence, with Firebase Analytics capturing behavioral metadata for further analysis.	High
The application shall support a modular architecture.	The system should be built using MVVM architecture, allowing clean separation of concerns, easier testing and scalable integration of new features such as reminders and future wearable syncing.	Medium
The app shall deliver context-aware hydration goals.	Daily hydration targets should dynamically adjust based on prior history and real-time weather data obtained via external APIs and rendered in the UI.	High
The solution should enable insightful visualization and summaries.	Analytics features must present hydration, mood and energy trends through intuitive graphs, streaks and summaries, allowing users to reflect on long-term behavior.	High
The app shall include accessibility-aware design.	Visual elements such as large tap targets, high-contrast themes and minimal text input should be used to ensure accessibility for elderly users and those with impairments.	Medium
The app should be extensible with backend interoperability.	The architecture must support RESTful communication with backend Python APIs that expose insights from ML models for mood/meal prediction and allow future extensions with BigQuery or Firebase Remote Config.	Medium
The system shall support reminder notifications.	Custom hydration reminders should be scheduled using Android WorkManager or AlarmManager and respond to both system state and prediction model feedback.	High

Table 4.1: Functional and Architectural Requirements of the MoodSip Application

4.4 Tools and Technologies Used

4.4.1 Platforms

1. Kotlin

MoodSip was built entirely in **Kotlin**[14], the official language for Android development. Kotlin was chosen for its null safety, coroutine-based concurrency model, concise syntax, full Jetpack Compose support and improved maintainability compared to Java. Another reason it was chosen was to challenge myself and build in a language I had never used before.

2. Operating System

Development was carried out primarily on **Windows 11**, with Android Studio configured. This ensured consistent Gradle and SDK behavior.

3. IDE

Android Studio Narwhal (v2025.1.1)[15] served as the primary IDE. It was selected due to its robust Kotlin/Jetpack Compose support, integrated emulator, live previews, Gradle build system and full Firebase integration.

4. Google Cloud Platform (Cloud Run)

Google Cloud Run[16] was used to deploy the hydration AI, LLM meal insight and analytics insight backends. These services scaled automatically, accepted REST calls and supported secure and scalable interaction from the mobile client without manual server provisioning.

5. Firebase Platform

The app integrated the following Firebase services[17]:

- **Firebase Analytics**: To log hydration and meal logging behavior for later trend analysis.
- **Firebase DebugView**: Used to live-preview analytics events during development. This ensured hydration, meal and screen-view logs were correctly recorded for later insight generation.
- **Firebase Cloud Messaging (FCM)**: Planned for future notification delivery.

Firebase was chosen for its seamless integration with Android, scalability and documentation.

6. OpenWeatherMap API

To retrieve live weather data (temperature and humidity) to adjust hydration goals dynamically. The API was accessed via Retrofit and integrated into the top card UI of the hydration screen[18].

4.4.2 Frameworks

1. Jetpack Compose

Jetpack Compose[19] was the UI toolkit used throughout. It enabled declarative UI development, state management and animation control. Compose also supported themes, accessibility and responsive layouts with fewer XML constraints.

2. MVVM Architecture

Model-View-ViewModel[20] was adopted for its decoupling of UI and logic. This pattern facilitated easier testing, clearer responsibilities and scalability across hydration, meals and analytics modules.

3. WorkManager

Used for scheduling background hydration reminders, WorkManager[21] handles retries, battery constraints and guarantees execution even after reboots. It was selected over AlarmManager for modern compatibility.

4.4.3 Libraries

1. Retrofit

Used for all API calls to the weather, hydration prediction,and meal insight models. Retrofit[22] offers powerful URL parsing, automatic response mapping and seamless coroutine support.

2. Kotlin Coroutines + Flow

Coroutines[23] and Flow were used to handle asynchronous data operations, such as fetching DataStore values, calling APIs and managing user state. Flow's reactive paradigm was especially useful for the hydration/mood/energy logs.

3. Jetpack DataStore

DataStore[24] was employed for storing hydration and meal logs. It supports asynchronous, lifecycle-aware persistence in Proto/Preferences format and replaces SharedPreferences with modern safety.

4. MPAndroidChart

Used in the Analytics screen to render hydration, mood and energy trends over time. The library supports line, bar and pie charts with real-time updates and tooltips, improving interpretability of behavioral data[25].

5. Maxkeppeler Calendar

Calendar views for analytics tab to view user logs was implemented using this as it aided date blocking and easy integration to kotlin labels[26].

6. Emoji2 Library

To ensure consistent emoji rendering across Android versions and device manufacturers, the **Emoji2** Jetpack library[27] was used. It enables backward-compatible emoji support for visual logs and feedback overlays.

7. Lottie

Used for hydration and info button animations[28].

4.4.4 Other Tools

1. Python (FastAPI)

All AI/ML backends were implemented using Python with FastAPI[29]. These included:

- Hydration Prediction API: Ingests logs and weather to generate risk scores.
- LLM Insight API: Generates meal-based mood/energy insights using prompt-tuned LLMs.
- Analytics Summary API: Provides insights from user logs.

2. PlantUML

Used to prototype several use-case, architectural and sequence diagrams until the right fit was found and implemented[30].

3. Git & GitHub

GitHub[31] hosted the full source code, including versioned branches for UI experiments, backend logic and model integration. Pull requests, releases and issue tracking were actively used.

4. Postman

Postman[32] was used to test hydration prediction and meal insight REST endpoints before mobile integration.

Chapter 5

Implementation of Mood-Energy Quantification

5.1 Mood Sensing Approaches

Understanding and quantifying mood states is a complex task and several psychological models have been proposed to define and structure emotions. Two dominant perspectives guided the design of the MoodSip application: Discrete Emotion Theories and Multidimensional Emotion Models.

According to the *Discrete Emotions Theory*, emotions are considered distinct and universal categories. Each emotion is thought to possess a unique physiological and psychological profile. Robert Plutchik expanded this theory through his well-known ”Wheel of Emotions,” which includes eight primary emotions—joy, trust, fear, surprise, sadness, disgust, anger and anticipation—and illustrates how their intensities and combinations form secondary emotional states[33]. Although this model is rich in detail, its categorical nature limits its scalability in digital systems.

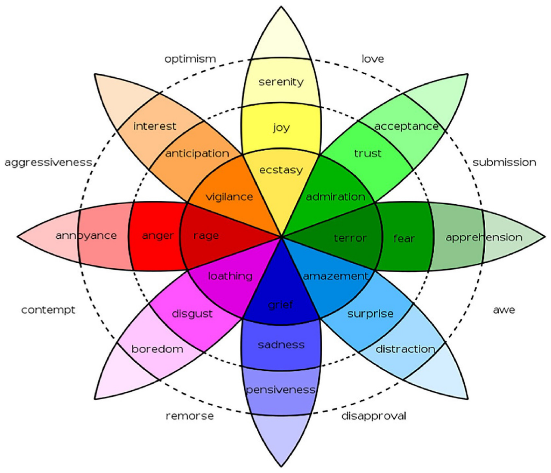


Figure 5.1: Plutchik Wheel of Emotions

In contrast, the *Multidimensional Theory of Emotion* proposes that emotional states lie within a continuous space, typically described by two or three dimensions. The 2D model, adopted in this project, focuses on valence (positive vs. negative) and arousal (high vs. low activation). This model offers a smoother gradient of affective experience, making it ideal for computational and user-friendly design[34].

To support real-time affect logging, various sensing modalities exist. Questionnaire-based self-reports are among the most common, including tools such as the Positive and Negative Affect Schedule (PANAS)[35], the Self-Assessment Manikin (SAM)[36] and the Photographic Affect Meter (PAM)[37]. These techniques allow users to manually assess their emotional states in a structured manner and have been widely validated in emotion research. PANAS provides dual scales for positive and negative affect[35], while SAM presents a non-verbal method to assess valence and arousal directly[36]. PAM allows visual affect logging by selecting representative photos[37] and the Experience Sampling Method (ESM) collects momentary emotional snapshots in real-world settings[38].

5.2 The Circumplex Model and Mood Quantification

In order to enable machine learning-based analysis and personalization within the MoodSip application, it was necessary to transform subjective mood inputs into structured numerical values. This transformation was guided by the well-established **Valence-Arousal Model of Affect** [34], which organizes emotions along two key axes: *Valence* (ranging from unpleasant to pleasant) and *Arousal* (ranging from calm to activated). Emotions can thus be conceptualized as occupying specific regions within a circular space, as shown in Figure 5.2.

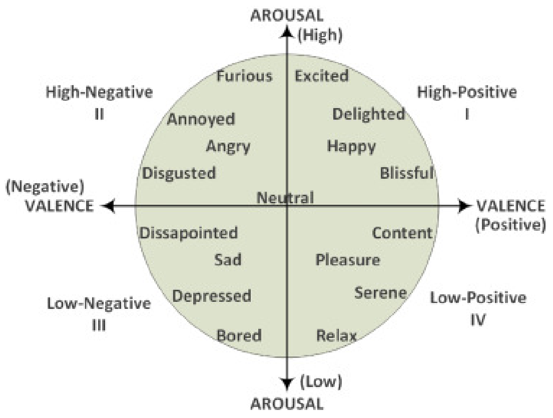


Figure 5.2: Russell's Circumplex Model of Affect

The quadrant-based interpretation of this model provides an effective structure for dividing mood into ordinal categories suitable for computational systems.

5.3 Mapping Mood Categories to Numerical Levels

To operationalize this model for use within the MoodSip data pipeline, I adopted a five-level ordinal scale that aligns semantically with the emotional valence-arousal space. The specific mapping is as follows:

- **Level 5 – High Positive Arousal:** Emotions such as *Excited*, *Delighted* and *Happy* occupy the high-arousal, high-valence quadrant (I). These are mapped to mood level 5.

- **Level 4 – Low Positive Arousal:** States like *Content*, *Relaxed*, or *Serene* (quadrant IV) are assigned to mood level 4.
- **Level 3 – Neutral:** Affective neutrality is represented at the origin of the circumplex and is mapped to mood level 3.
- **Level 2 – Low Negative Arousal:** Emotions such as *Sad*, *Depressed* and *Bored* (quadrant III) are mapped to mood level 2.
- **Level 1 – High Negative Arousal:** Highly aroused negative states like *Furious*, *Annoyed*, or *Disgusted* (quadrant II) correspond to mood level 1.

This division enables user moods to be logged via intuitive UI sliders while producing ordinal data usable for statistical analysis and machine learning modeling.

5.4 Why Numerical Encoding is Essential

Quantifying emotional states using a bounded ordinal scale from 1 to 5 serves several essential purposes in the MoodSip system. Firstly, it ensures **data consistency** across all user entries. By representing mood numerically, a standardized format is maitained that simplifies downstream data handling and avoids ambiguity when interpreting emotional input.

Secondly, this numerical encoding enables **behavioral correlation** between mood levels and various lifestyle factors such as hydration patterns, food categories or even the time of meal consumption. These relationships can then be explored using common analytical methods like regression, classification and trend analysis.

Another critical benefit is in the **training of AI models**. Structured, low-dimensional inputs are ideal for both the hydration risk prediction model and the LLM-based meal insight generator. Mood level encoding enables these models to easily learn, allowing for more accurate predictions and personalized insights.

Lastly, the abstraction of mood into numerical form contributes to **user privacy and model generalization**. Unlike raw text descriptions, which might reveal sensitive context or include subjective language, numeric values provide a depersonalized abstraction. This not only protects user identity but also improves the generalizability of the AI models.

5.5 Energy Level Quantification and Interpretation

While mood has been extensively studied in affective science literature, subjective energy levels have comparatively less standardization. During the development of the MoodSip application, it became evident that no universally accepted framework existed for quantifying self-reported energy in the same manner as mood valence-arousal models. As a result, I adopted a user-centered approach to define energy levels on a five-point ordinal scale.

This choice was primarily guided by the need for simplicity, ease of self-assessment and compatibility with machine learning pipelines. Each level was designed keeping in mind how an individual might assess their body and cognitive vitality at a given moment. Essentially, it encourages users to ask themselves simple introspective questions like, "Do I feel refreshed or drained right now?" or "Can I stay productive if I continue like this?"

- **Level 5 – Fully Energized:** The user feels exceptionally active, focused and physically refreshed. This could correspond to states following good rest, exercise or peak motivation.
- **Level 4 – Alert and Engaged:** The user feels productive and attentive, though not at peak capacity. They are likely functioning well with minor fatigue.
- **Level 3 – Neutral or Average:** The user does not perceive any strong sensations of tiredness or energy. This level represents a baseline or default energy state.
- **Level 2 – Tired or Sluggish:** The user may feel mentally foggy or physically slow, possibly needing rest, hydration or stimulation to return to a higher state.
- **Level 1 – Drained or Fatigued:** The user feels depleted, possibly unable to focus or move efficiently and is likely in need of recovery or nourishment.

5.6 UI Implementation

5.6.1 Pulsating Information Icon on Sliders

To improve user understanding of what each slider represents, an animated pulsating `i` button is placed next to both the Mood and Energy sliders. When tapped, this button opens a detailed tooltip explaining the levels, interpretation range and user intent behind each option. This ensures users can make informed self-assessments.

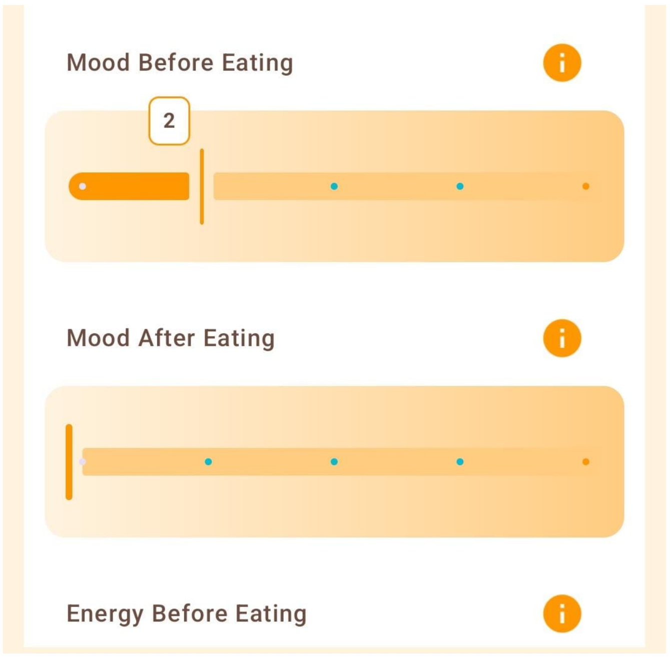
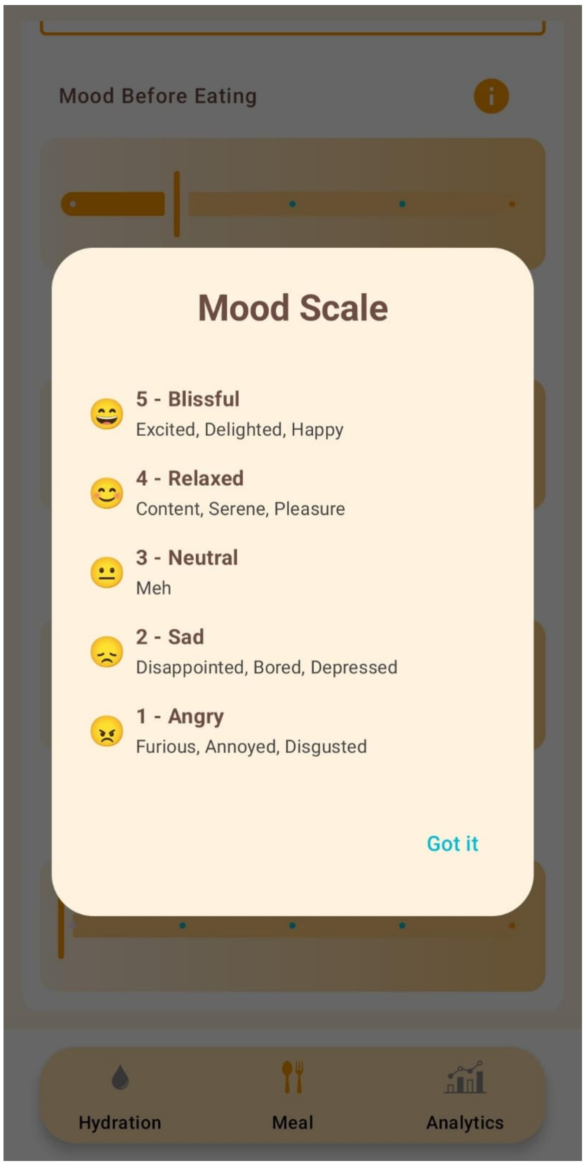


Figure 5.3: Information icon on mood/energy sliders for interactive help.

5.6.2 Mood and Energy Scales

The Mood scale is based on the Russell Circumplex Model of Affect and maps emotional states across valence and arousal dimensions. Energy levels are assessed using a slider ranging from 1 (fully drained) to 5 (fully energized). This scale is inspired by everyday perceptions of mental and physical vitality. Users are encouraged to reflect on how tired or alert they feel before and after a meal or hydration event.



Mood Before Eating

Mood Scale

5 - Blissful
Excited, Delighted, Happy

4 - Relaxed
Content, Serene, Pleasure

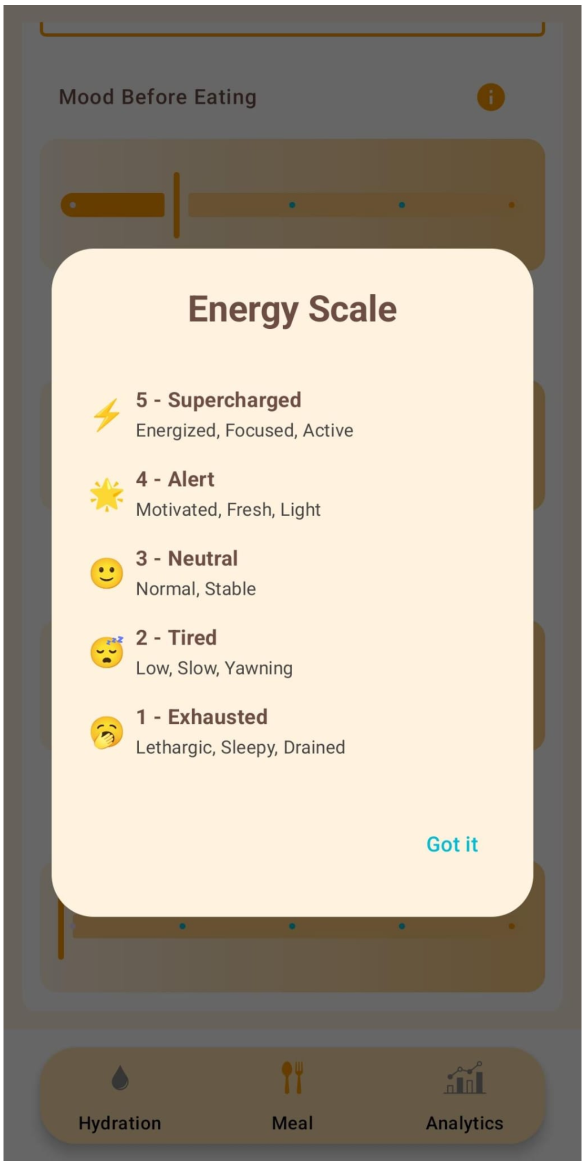
3 - Neutral
Meh

2 - Sad
Disappointed, Bored, Depressed

1 - Angry
Furious, Annoyed, Disgusted

Got it

(a) Mood scale based on the valence-arousal model.



Mood Before Eating

Energy Scale

5 - Supercharged
Energized, Focused, Active

4 - Alert
Motivated, Fresh, Light

3 - Neutral
Normal, Stable

2 - Tired
Low, Slow, Yawning

1 - Exhausted
Lethargic, Sleepy, Drained

Got it

(b) Energy scale reflecting subjective physical or mental vitality, from drained to fully energized.

Figure 5.4: Mood and energy self-assessment scales used in MoodSip.

Chapter 6

Implementation of Front-end

6.1 Core Frontend Development Principles

The frontend of MoodSip was designed to uphold principles that promote usability, accessibility, responsiveness and low cognitive load. A primary objective was to create an intuitive interface that could cater to users across diverse age groups and technological familiarity levels, while also accommodating those with physical or cognitive impairments [3]. In this pursuit, a combination of **Material Design 3** guidelines, **Compose for UI declarativity** and **accessible color theory** were applied throughout the interface. These principles are recommended by the World Wide Web Consortium (W3C) for modern interface design, especially in mobile health (mHealth) applications [39].

One of the core philosophies behind MoodSip's frontend architecture is rooted in **minimal input design**. Conventional wellness and tracking applications often overwhelm users with complex logging interfaces [9]. This design flaw frequently leads to reduced engagement and poor long-term adherence. In contrast, MoodSip provides a tap-based interface for hydration, visual selectors for meal categorization and sliders for mood and energy thus reducing friction in the logging process. Every touchpoint was designed to convey **immediate visual feedback**, reducing uncertainty and improving perceived control, a key factor in behavioral retention models [40].

The interface was built using Jetpack Compose, which offered **declarative UI construction** and simplified the **state management of reactive components**. This not only enhanced code maintainability but also allowed animations like the dynamic hydration droplet to update seamlessly based on user state.

MoodSip prioritizes **high-contrast visual hierarchy**, adopting a palette rooted in **warm tones** to draw attention to action areas without overwhelming the interface. **Typography hierarchy** was also preserved to guide user focus logically from headings to inputs. These decisions were influenced by **Nielsen's usability heuristics** which emphasize visibility of system status and recognition over recall [41].

6.2 Hydration Logging Flow

MoodSip follows a linear-yet-modular navigation architecture, where users move through three primary flows: **Hydration Logging**, **Meal Logging** and **Analytics & Insights**. These are accessible via a persistent bottom navigation bar, adhering to modern mobile usability patterns which favor minimal taps for frequent actions [41].

6.2.1 Loading Screen and Hydration Screen

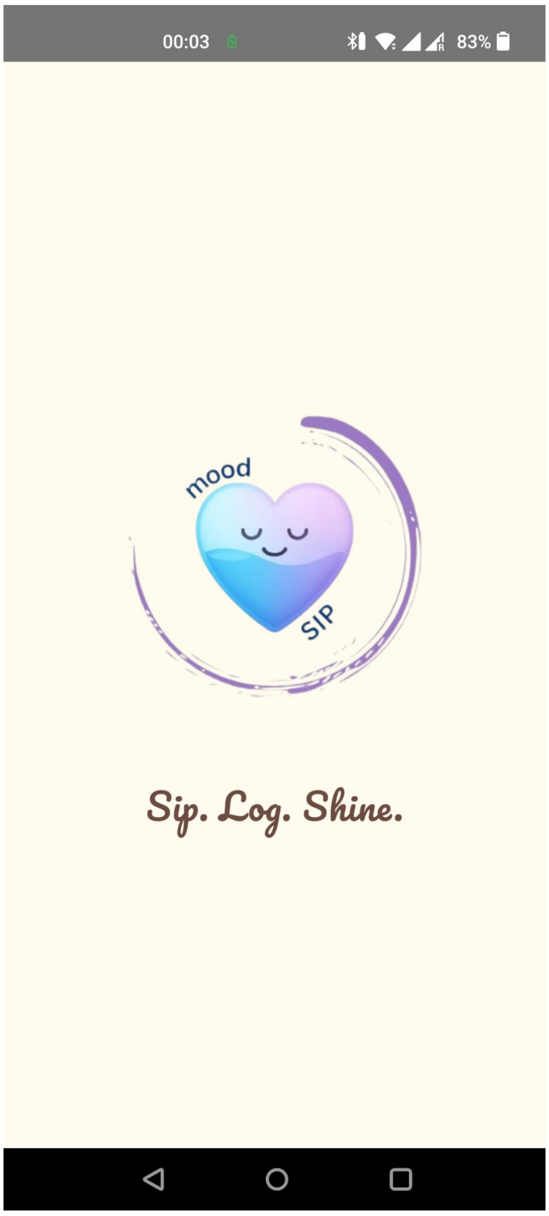


Figure 6.1: Animated Loading Screen

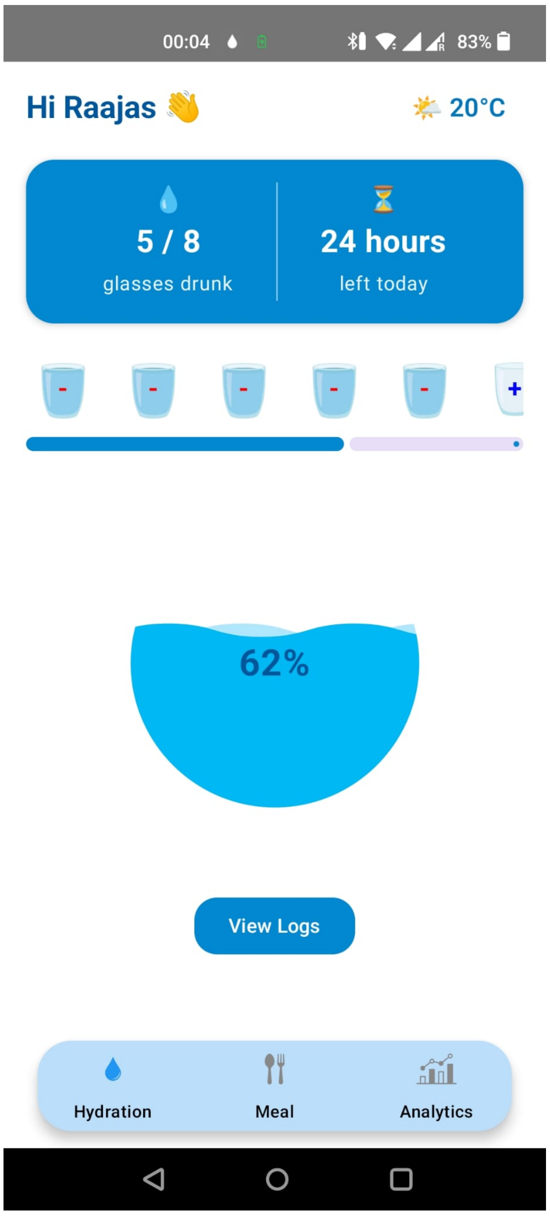
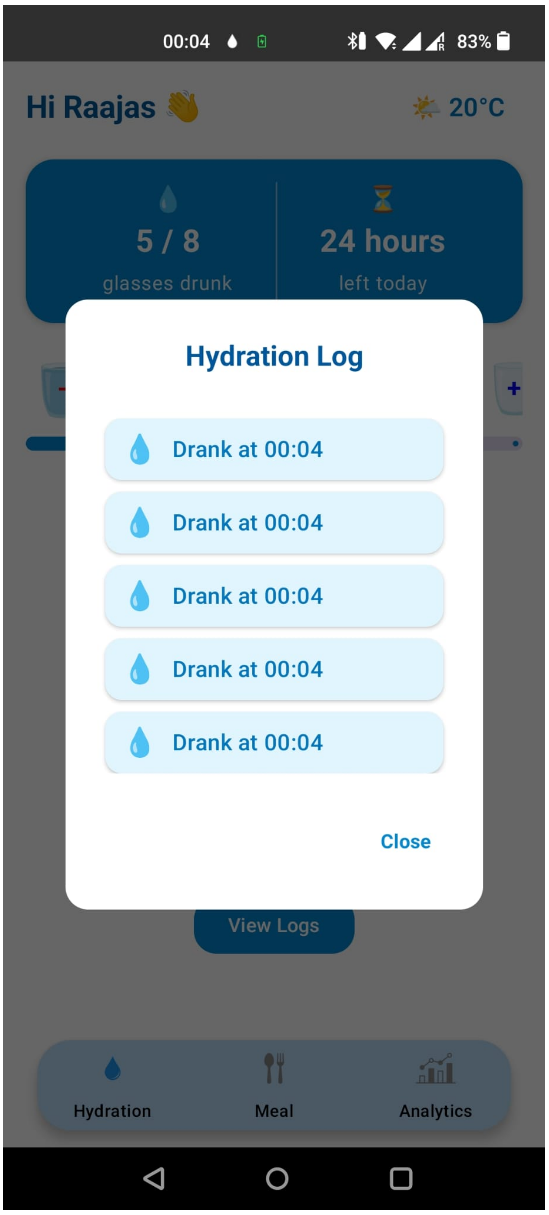


Figure 6.2: Hydration Tracking UI

Upon launching, users are greeted with a screen featuring a pulsing and spinning logo animation Figure 6.1. This not only provides a sense of brand identity but also manages load time perception, as recommended in mobile UI design best practices[39].

The hydration screen integrates several feedback elements to guide and motivate the user. It features a top card that displays current weather conditions (retrieved from the OpenWeatherMap API) and real-time hydration goal progress. Below it, dynamic glass icons fill interactively when tapped to represent the number of glasses consumed. A central animated water droplet visually reflects the user's intake.

6.2.2 Hydration Log and Celebration Overlay



Hydration Log

Drank at 00:04

Drank at 00:04

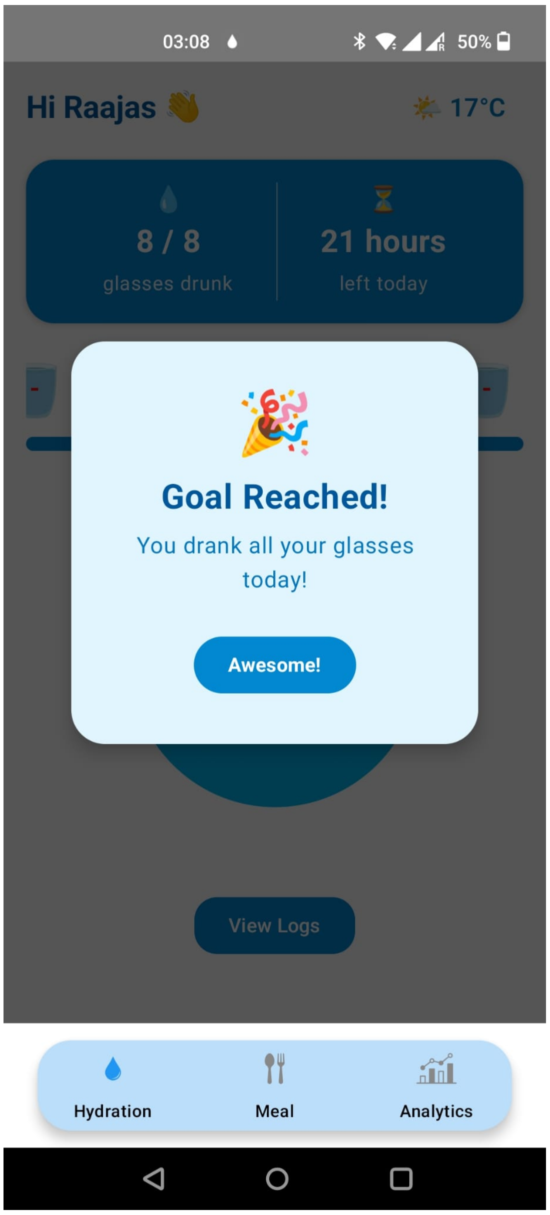
Drank at 00:04

Drank at 00:04

Drank at 00:04

Close

Figure 6.3: Hydration Log with Time and Emojis



🎉

Goal Reached!

You drank all your glasses today!

Awesome!

Figure 6.4: Goal Completion Celebration Overlay

A time-stamped hydration history is displayed below the tracker as seen in Figure 6.3. Upon completing the hydration goal, a celebration overlay is triggered, as seen in Figure 6.4, designed to create a reward loop and encourage habit formation through positive visual reinforcement.

6.3 Meal Logging Flow

6.3.1 Meal Logging and Insight Overview

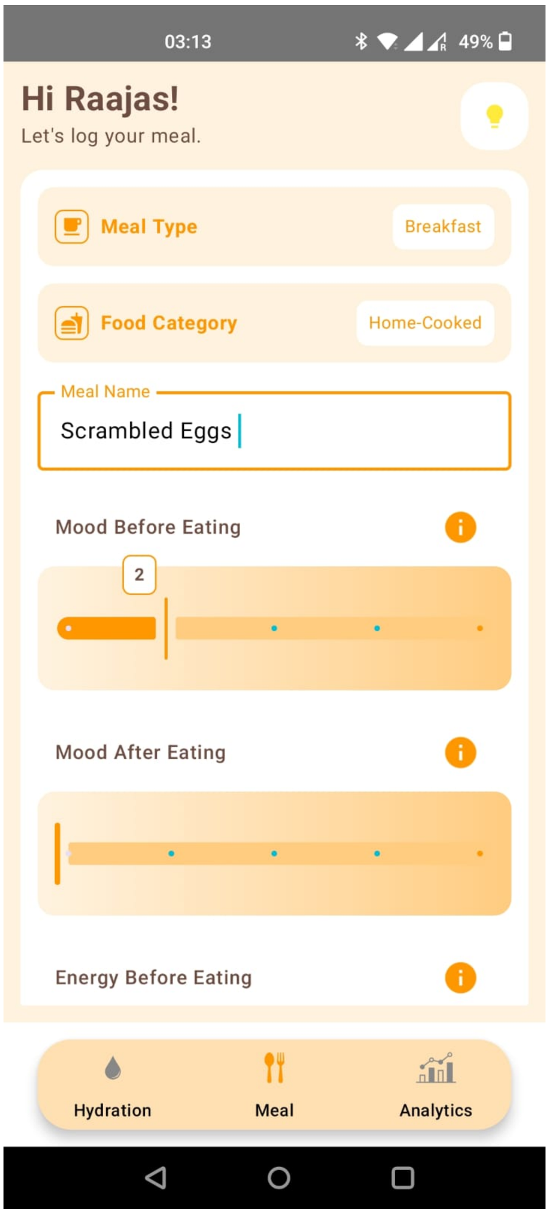
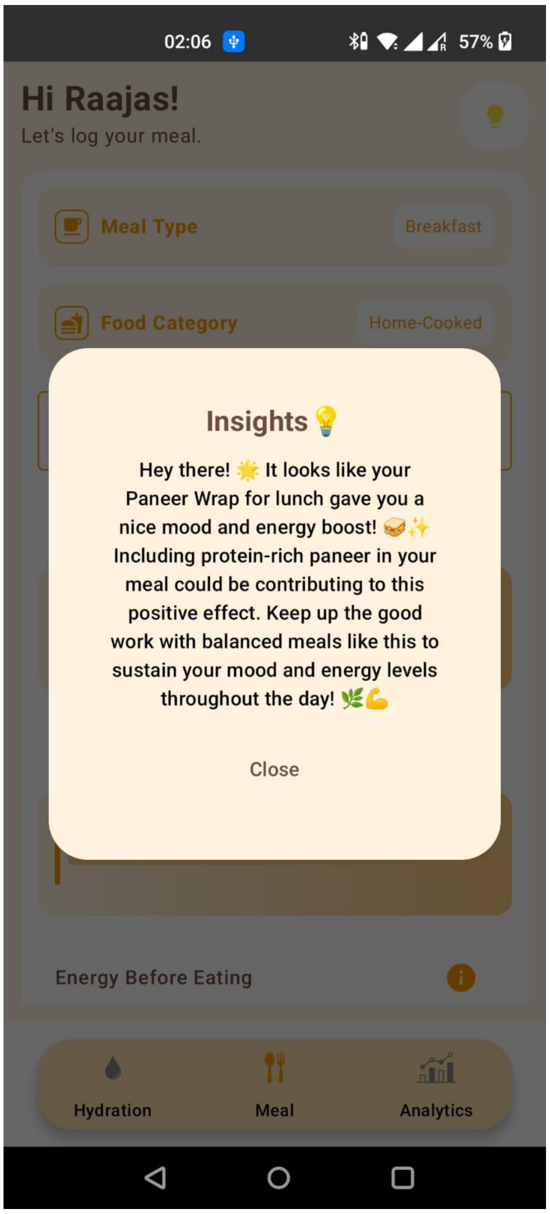


Figure 6.5: Meal Logging Interface



Insights 💡

Hey there! 🌟 It looks like your Paneer Wrap for lunch gave you a nice mood and energy boost! 🥪✨ Including protein-rich paneer in your meal could be contributing to this positive effect. Keep up the good work with balanced meals like this to sustain your mood and energy levels throughout the day! 🌿💪

Close

Figure 6.6: AI-Generated Insight Screen

The meal logging screen allows users to enter meal details using minimal input methods. Users can enter a meal name, choose its category and type and assign pre- and post-meal mood and energy levels using sliders. Figure 6.5 shows the core logging interface, while Figure 6.6 highlights the LLM-generated insights screen accessible after logging.

6.3.2 Selecting Food Category and Meal Type

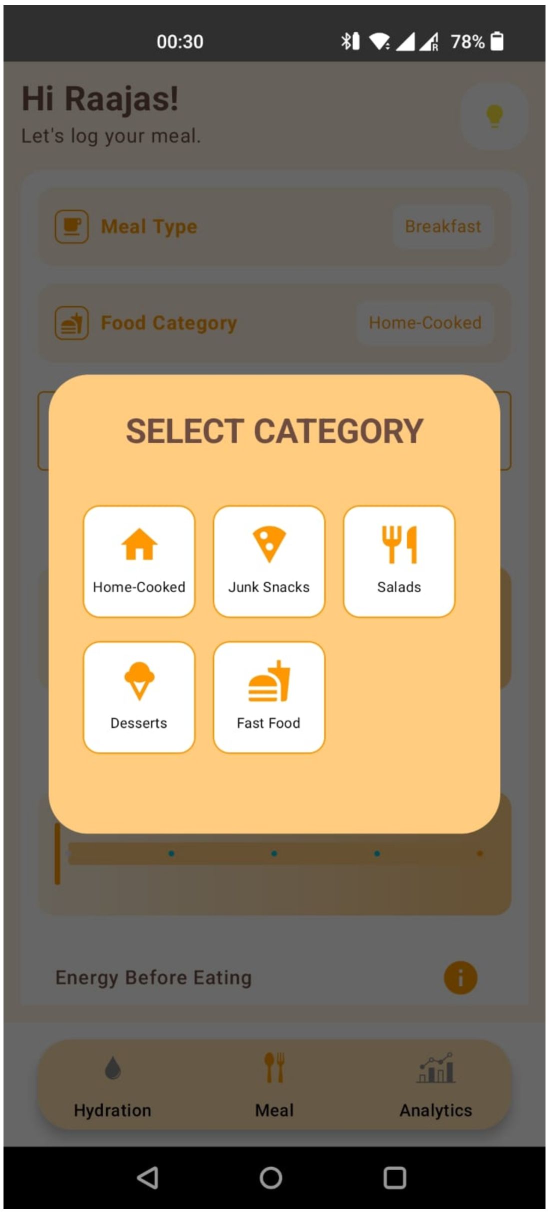


Figure 6.7: Select Food Category

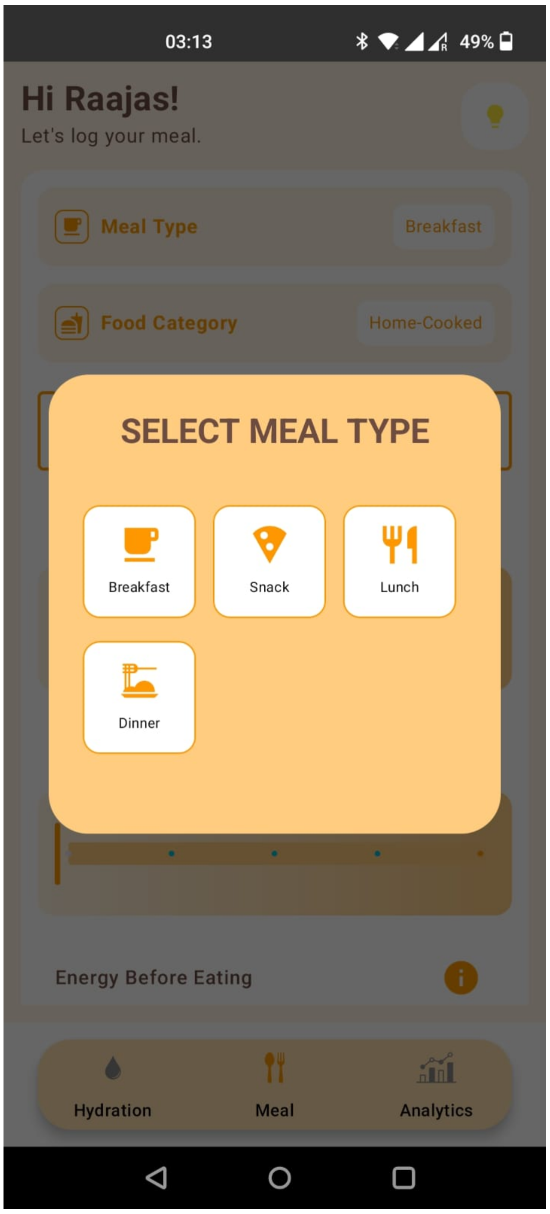
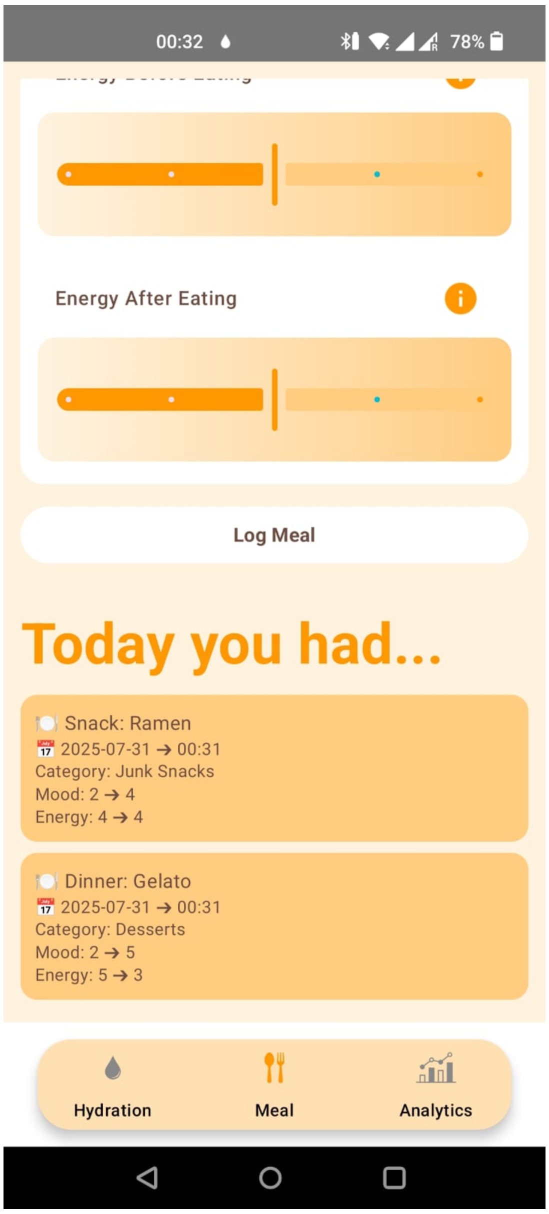


Figure 6.8: Select Meal Type

To further structure meal data, MoodSip uses tappable cards to select food category (e.g., home-cooked, junk) and meal type (e.g., breakfast, dinner). Figure 6.7 and Figure 6.8 show these UI elements, respectively. Users are then asked to reflect and log their emotional and physical states before and after meals using sliders Figure 5.3.

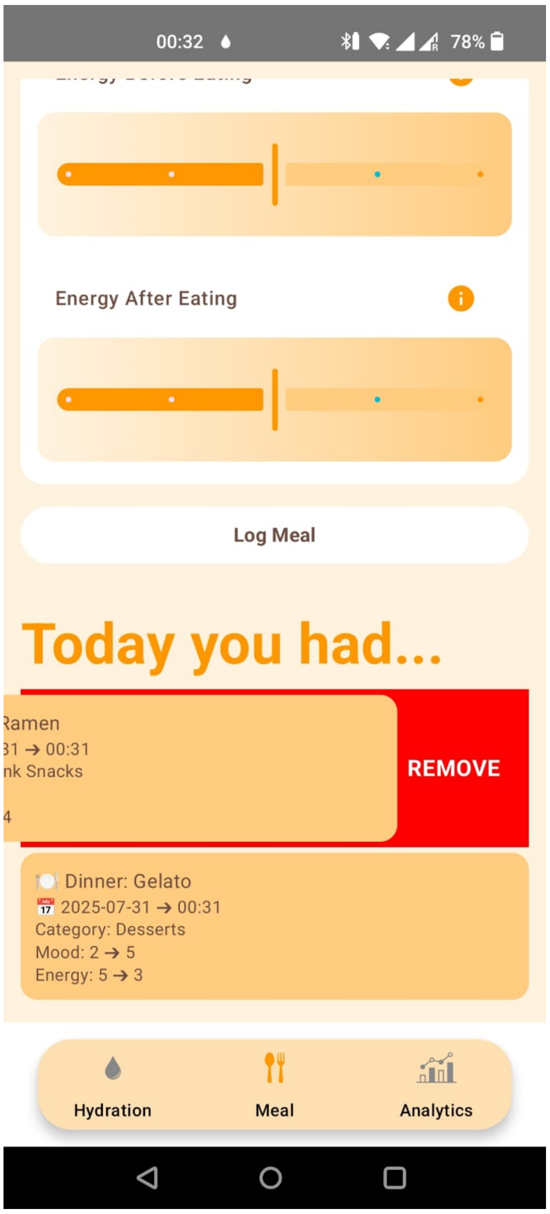
6.3.3 Meal Log and Entry Removal



🍽️ Snack: Ramen
📅 2025-07-31 → 00:31
Category: Junk Snacks
Mood: 2 → 4
Energy: 4 → 4

🍽️ Dinner: Gelato
📅 2025-07-31 → 00:31
Category: Desserts
Mood: 2 → 5
Energy: 5 → 3

Figure 6.9: Meal Log of the Day



Ramen
31 → 00:31
nk Snacks
4

🍽️ Dinner: Gelato
📅 2025-07-31 → 00:31
Category: Desserts
Mood: 2 → 5
Energy: 5 → 3

Figure 6.10: Swipe to Remove Entry

Once meals are logged, they appear in a scrollable list showing name, type and timestamp. Users can swipe left on an entry to delete it instantly. This lightweight interaction model keeps the log clean while offering a sense of control. Figure 6.9 and Figure 6.10 illustrate the daily log and swipe gesture.

6.4 Analytics Tab and Trends

The Analytics tab in MoodSip is designed to present behavioral patterns and trends in hydration, mood and energy. Unlike static logs, this screen emphasizes change over time, empowering users to self-reflect and course-correct based on visual cues. Calendars, tooltips, modular graphs and streak indicators work together to provide an engaging yet actionable overview.

6.4.1 Calendar and Weekly Logs

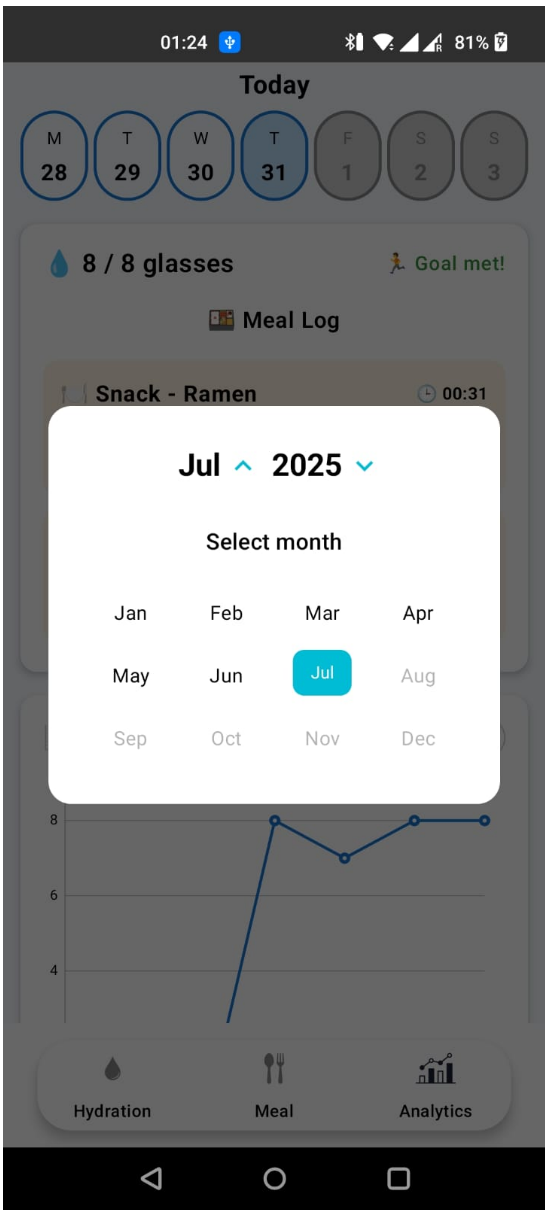
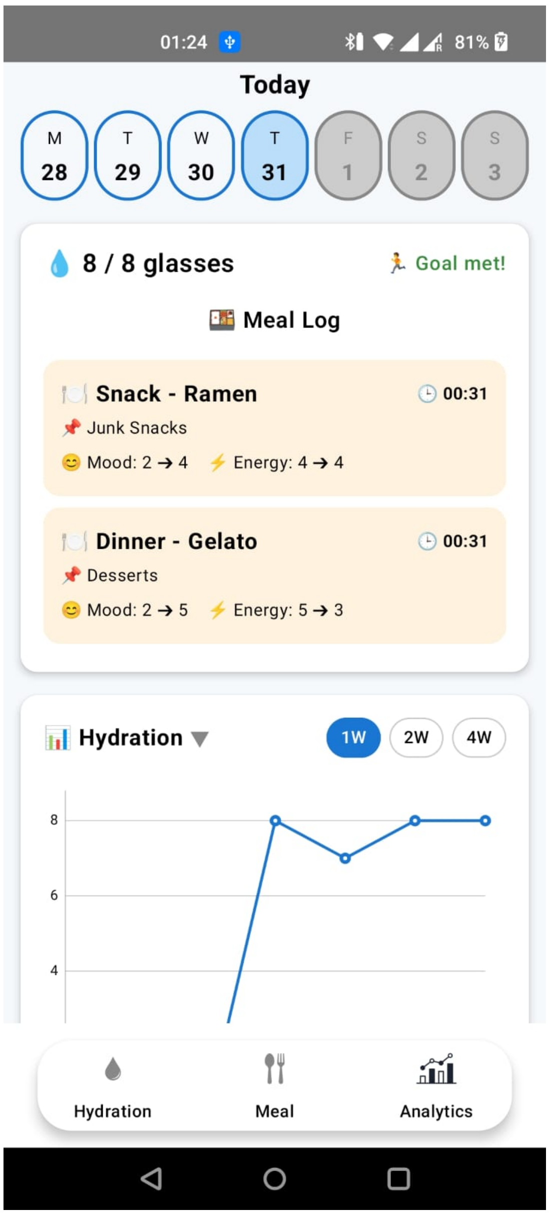


Figure 6.11: Calendar-Based Log Selection



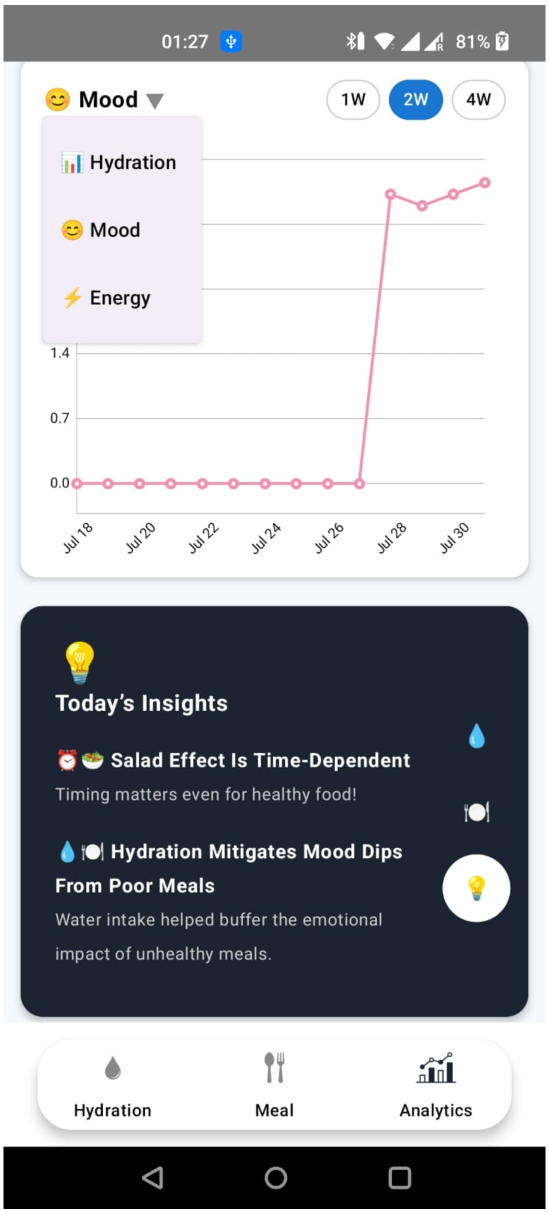
Snack - Ramen 00:31
Junk Snacks
Mood: 2 → 4 Energy: 4 → 4

Dinner - Gelato 00:31
Desserts
Mood: 2 → 5 Energy: 5 → 3

Figure 6.12: Current Week Log Summary

We begin by selecting a date from a calendar view, which dynamically filters the hydration and meal logs for that day. The accompanying week view summarizes recent behavior. Together, they support short-term tracking and retrospective analysis.

6.4.2 Chart Selector and AI Insights



💡

Today's Insights

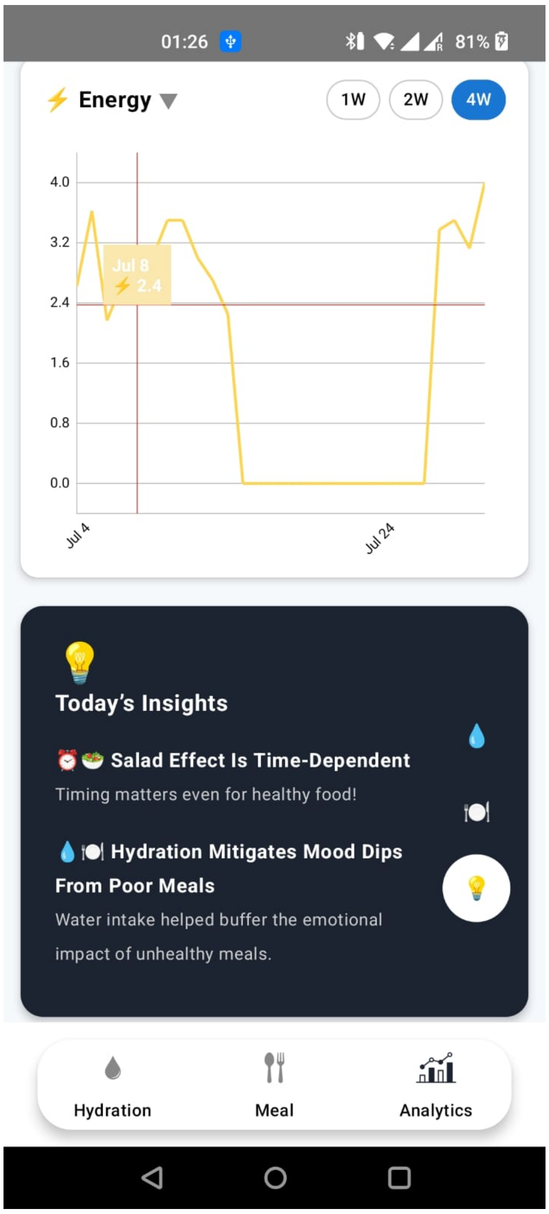
⏰🥗 Salad Effect Is Time-Dependent

Timing matters even for healthy food!

💧🍽 Hydration Mitigates Mood Dips From Poor Meals

Water intake helped buffer the emotional impact of unhealthy meals.

Figure 6.13: Interactive Graph Selector and Legend



💡

Today's Insights

⏰🥗 Salad Effect Is Time-Dependent

Timing matters even for healthy food!

💧🍽 Hydration Mitigates Mood Dips From Poor Meals

Water intake helped buffer the emotional impact of unhealthy meals.

Figure 6.14: Four-Week Energy Chart with AI Insight

The graph selector enables users to toggle between hydration, mood and energy visualizations. Tooltip-supported line charts improve interpretability of micro-fluctuations, while AI-generated insight cards offer human-readable summaries for the week.

6.4.3 Trends and Streaks

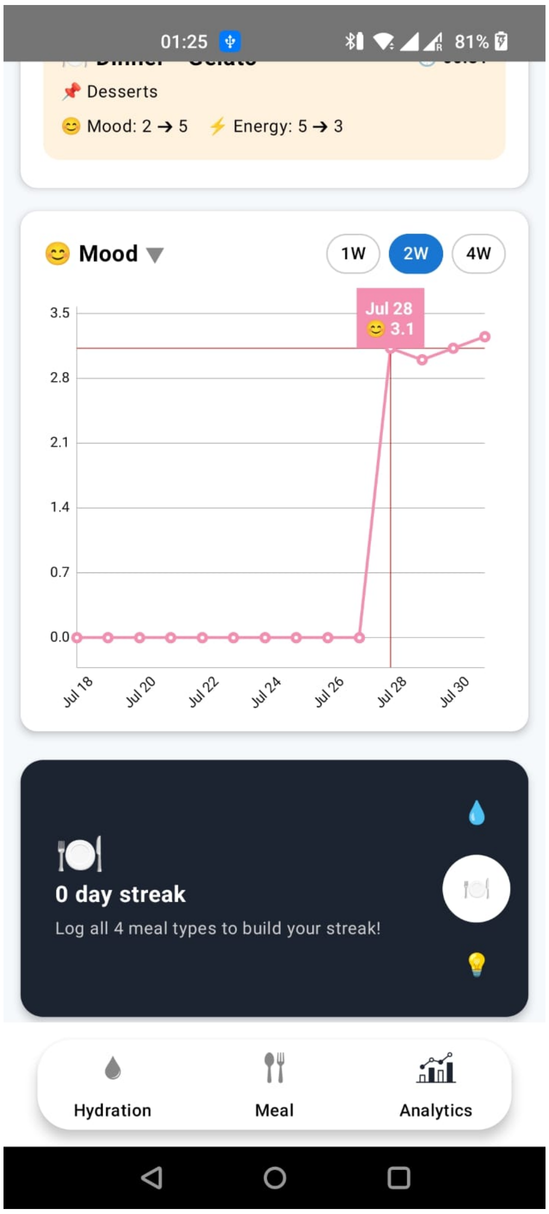


Figure 6.15: Meal Streak and Mood Trend (2 Weeks)

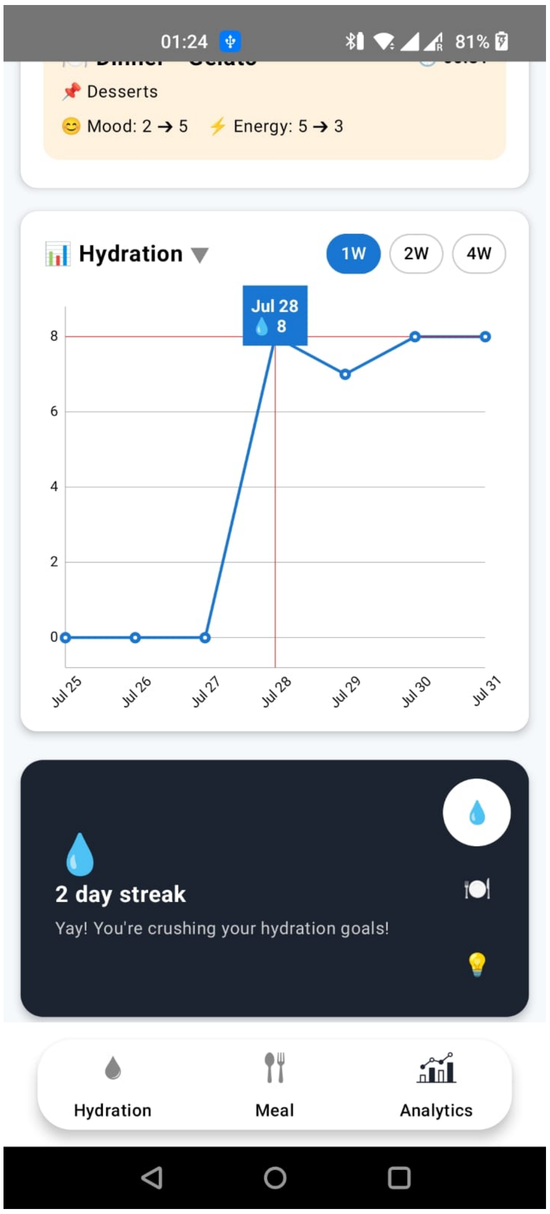


Figure 6.16: Hydration Trend and Streak (1 Week)

6.5 Summary

In summary, the front-end implementation demonstrates a clearly planned approach, using Jetpack Compose and MVVM architecture to effectively manage the reactive state. It reflects a strong understanding of current methodologies, incorporating principles from Material Design 3, W3C accessibility guidelines, Nielsen heuristics and cognitive load theory. The design is one-of-a-kind in its integration of hydration, mood and energy tracking into a unified UI. The challenge of translating subjective emotion into structured input was addressed through design choices backed by theory.

Chapter 7

Implementation of Backend

MoodSip integrates three core backend methodologies. These systems are built on a combination of traditional logic, AI models and LLM-powered pipelines, deployed as RESTful APIs via FastAPI and hosted on Google Cloud Run for scalability and performance.

7.1 Hydration Prediction Engine

7.1.1 Overview

The hydration backend model processes real-time user behavior data and weather data (temperature) retrieved from the OpenWeatherMap API. This hybrid input allows the system to compute a personalized hydration risk score, identifying whether the user is likely to underhydrate on a given day. The model is continuously refined using behavioral history and operates asynchronously in the background via WorkManager, enabling the app to deliver proactive notifications.

7.1.2 Input Data and Features

To accurately model hydration risk, the backend uses multiple user-centric and environment-aware features. The training dataset consisted of daily hydration logs compiled as discussed in Section 3.1. The following variables were extracted for each day:

- `streak`: Number of consecutive days the user completed their hydration goal.
- `avg_glasses`: The mean number of glasses consumed by the user over the last seven days.
- `missed_days`: Count of days the user failed to meet their hydration goal in the last week.
- `temp`: Temperature (in °C) recorded at exactly 15:00 hours from the OpenWeatherMap API.
- `time_of_day`: Categorical feature representing whether the entry occurred in the `morning`, `afternoon`, or `evening`.

The features `avg_glasses` and `missed_days` serve as behavioral indicators of commitment, while `temp` and `time_of_day` act as environmental and temporal modifiers. The `streak` reflects habit strength, a well-known behavioral reinforcement metric.

7.1.3 Model Output

The model predicts a single binary label:

- `hydration_risk`: Takes value 1 if the user is at risk of underhydrating on the current day and 0 otherwise.

This label serves as the basis for both UI overlays and backend-triggered hydration reminders.

7.1.4 Machine Learning Approach and Model Training

A **Random Forest Classifier** was selected due to its robustness against overfitting, ability to handle mixed feature types and interpretability. Random forests have shown high performance in health behavior prediction tasks [42].

The dataset is denoted as:

$$X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}, \quad y = \{y_1, y_2, \ldots, y_n\}$$

where $\mathbf{x}_i$ contains the input features described above and $y_i \in \{0, 1\}$ is the hydration risk label.

Given decision trees $T_1, T_2, \ldots, T_k$, the forest predicts risk as:

$$\hat{y} = \text{majority\_vote}(T_1(\mathbf{x}), T_2(\mathbf{x}), \ldots, T_k(\mathbf{x}))$$

The classifier was trained using $k = 100$ estimators and a fixed seed of 42 to ensure reproducibility. One-hot encoding was used for the `time_of_day` feature through a pipeline approach and the model was serialized using `joblib`.

7.1.5 Hydration Goal Calculation: Weather and Risk Adjustment

The hydration goal displayed to the user on the mobile interface is dynamically adjusted based on the ambient temperature at 15:00 hours using the OpenWeatherMap API. Let:

- T = Temperature in Celsius retrieved from the weather API
- $G_{\text{base}} = 8$ = Default hydration goal (in glasses)
- $\Delta G = \left\lfloor \frac{T - 25}{5} \right\rfloor^{+}$ = Temperature adjustment factor
- Predicted hydration risk `HRisk` $\in \{0, 1\}$ (extra glass if hydration risk is high)
- G_{final} = Adjusted hydration goal displayed to the user

Here, the operator $\lfloor \cdot \rfloor^{+}$ denotes a *non-negative floor*, implemented via the Kotlin method `coerceAtLeast(0)`. This ensures no decrement in hydration goal for temperatures below the baseline threshold.

$$\Delta G = \max\left(\left\lfloor \frac{T - 25}{5} \right\rfloor, 0 \right) \tag{7.1}$$

$$G_{\text{final}} = G_{\text{base}} + \Delta G + HRisk \tag{7.2}$$

This rule assumes 25°C as the thermal comfort baseline. For every 5°C above this baseline, the user is advised to drink one additional glass of water. This discrete step function reflects hydration science findings that suggest fluid needs increase approximately linearly with ambient heat stress up to moderate thresholds [43].

This computed goal, G_{final}, is saved in the local DataStore using a date key to persist across restarts, enabling the analytics system to later evaluate goal adherence.

The temperature normalization ensures dynamic goal setting based on local climate variability, while the risk flag acts as a behavioral penalty to encourage compensatory drinking.

7.1.6 Real-Time Weather Integration

Only the temperature at `15:00 hours` is used as a hydration reference marker. This choice is motivated by the fact that mid-afternoon usually corresponds to peak temperature of the day and only based on this temperature should the hydration goal of the day be calculated. I have used the `/forecast` feature of the OpenWeatherMap API.

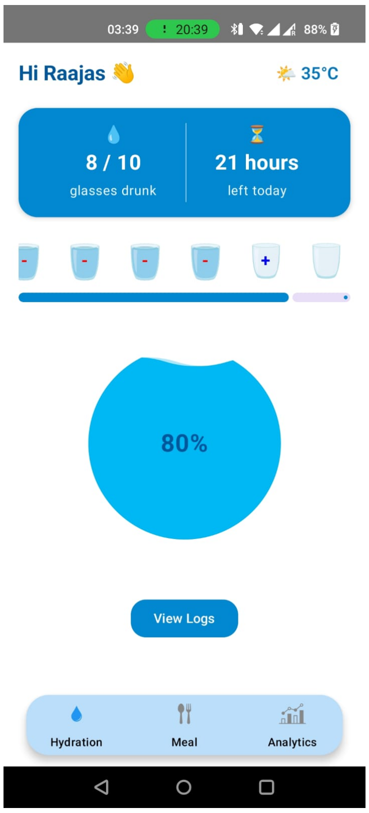
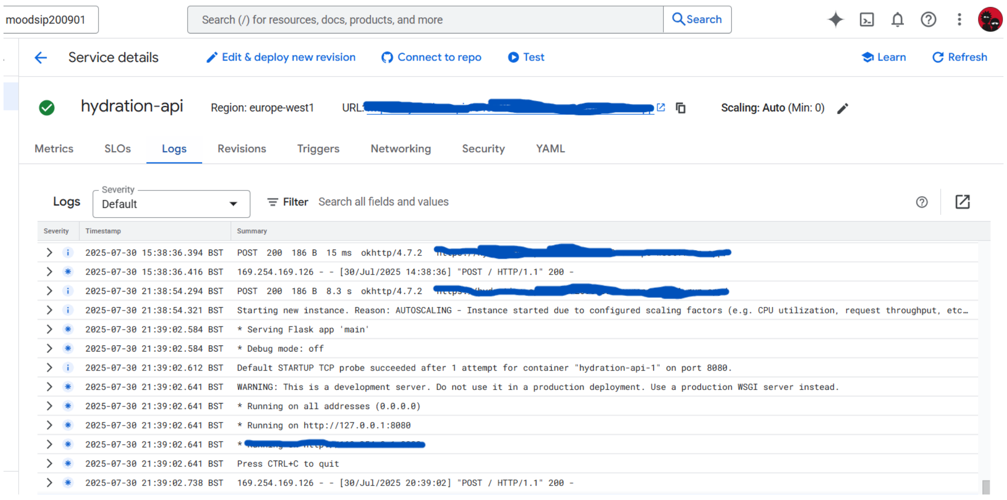


Figure 7.1: Increased daily goal when temp = 35 °C

7.1.7 Deployment via Cloud Run

The model is hosted as a REST endpoint using `FastAPI`. It accepts a JSON payload with user features and returns the predicted risk. The API is deployed on Google Cloud

Run, allowing automatic scaling, secure HTTPS and minimal cold start times. I have also ensured the pipeline is containerized with Docker for reproducibility.



hydration-api Region: europe-west1 URL: [redacted] Scaling: Auto (Min: 0)

Logs Severity: Default

```
Severity  Timestamp                    Summary
>  i  2025-07-30 15:38:36.394 BST  POST  200  186 B  15 ms  okhttp/4.7.2  [redacted]
>  *  2025-07-30 15:38:36.416 BST  169.254.169.126 - - [30/Jul/2025 14:38:36] "POST / HTTP/1.1" 200 -
>  i  2025-07-30 21:38:54.294 BST  POST  200  186 B  8.3 s  okhttp/4.7.2  [redacted]
>  i  2025-07-30 21:38:54.321 BST  Starting new instance. Reason: AUTOSCALING - Instance started due to configured scaling factors (e.g. CPU utilization, request throughput, etc…
>  *  2025-07-30 21:39:02.584 BST  * Serving Flask app 'main'
>  *  2025-07-30 21:39:02.584 BST  * Debug mode: off
>  i  2025-07-30 21:39:02.612 BST  Default STARTUP TCP probe succeeded after 1 attempt for container "hydration-api-1" on port 8080.
>  *  2025-07-30 21:39:02.641 BST  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
>  *  2025-07-30 21:39:02.641 BST  * Running on all addresses (0.0.0.0)
>  *  2025-07-30 21:39:02.641 BST  * Running on http://127.0.0.1:8080
>  *  2025-07-30 21:39:02.641 BST  * [redacted]
>  *  2025-07-30 21:39:02.641 BST  Press CTRL+C to quit
>  *  2025-07-30 21:39:02.738 BST  169.254.169.126 - - [30/Jul/2025 20:39:02] "POST / HTTP/1.1" 200 -
```

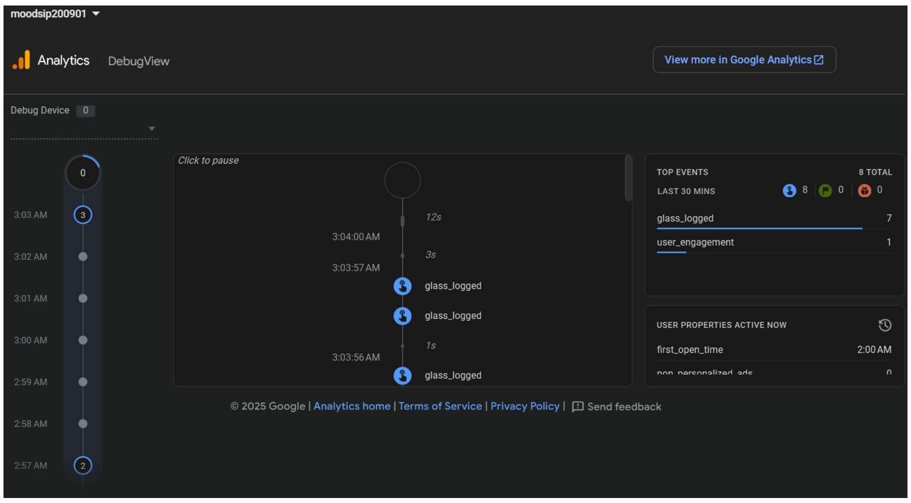
Figure 7.2: Deployment of Hydration Model on Google Cloud Run

7.1.8 Firebase Analytics Integration for Hydration Logging

To enable real-time user behavior tracking, **Firebase Analytics** was integrated into the hydration logging workflow. Each time the user taps on a glass to record a water intake event, a custom analytics event named `glass_logged` is fired. This integration is mainly setup when MoodSip comes out of the prototype phase with multiple users active.

Purpose of Logging: The event serves multiple goals:

1. Tracks daily hydration activity per user for behavior analysis.
2. Powers DebugView in Firebase Console to validate real-time logging



TOP EVENTS LAST 30 MINS	8 TOTAL
glass_logged	7
user_engagement	1

USER PROPERTIES ACTIVE NOW	
first_open_time	2:00 AM
non_personalized_ads	0

Figure 7.3: Firebase Analytics DebugView showing the `glass_logged` event.

7.1.9 Hydration Notification System

The MoodSip application integrates a smart notification system to guide users throughout their hydration journey. All notifications are triggered programmatically via `WorkManager` in such way that they are not intrusive (see Section 2.1.2).

The notification system includes the following categories:

1. Intake-based Notifications

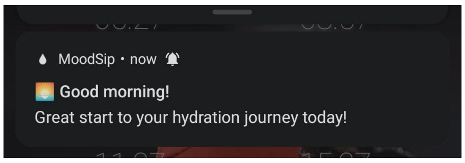

MoodSip • now
🌅 Good morning!
Great start to your hydration journey today!

Figure 7.4: Morning Notification: Encouraging first glass

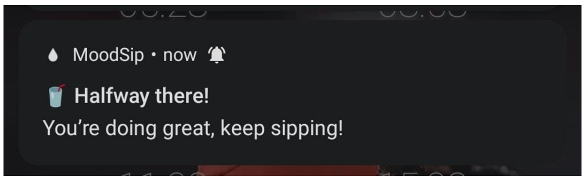

MoodSip • now
🥤 Halfway there!
You're doing great, keep sipping!

Figure 7.5: Progress Notification: Halfway Milestone

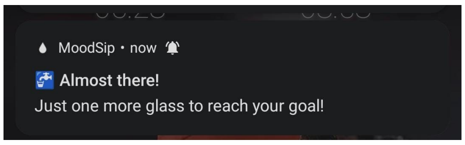

MoodSip • now
🚰 Almost there!
Just one more glass to reach your goal!

Figure 7.6: Progress Notification: Final Stretch

2. Weather-Triggered Alert

On days when the temperature exceeds a threshold (35°C), a `Heat Alert` notification is sent.

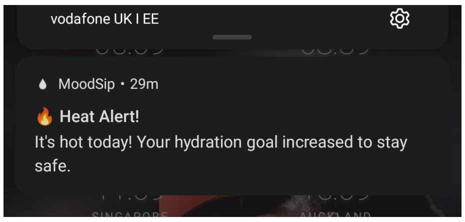

vodafone UK I EE
MoodSip • 29m
🔥 Heat Alert!
It's hot today! Your hydration goal increased to stay safe.

Figure 7.7: Weather-Based Notification: Heat Alert

3. AI-Based Hydration Risk Notification

Based on the output of the hydration risk prediction model (see Section 7.1.3), a contextual message is triggered:

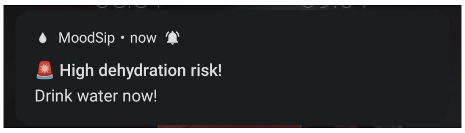

MoodSip • now
🚨 High dehydration risk!
Drink water now!

Figure 7.8: Risk-Based Notification: High

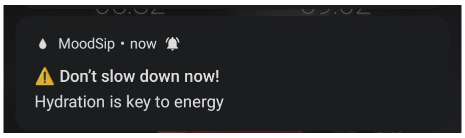

MoodSip • now
⚠️ Don't slow down now!
Hydration is key to energy

Figure 7.9: Risk-Based Notification: Low

7.2 Meal Insight Engine (LLM-Powered)

On the meal logging screen, users can access intelligent suggestions by tapping the button located at the top right corner. **This feature invokes an LLM-powered backend API that consumes the user's last five days of meal logs and generates personalized insights**. These insights are not rule-based; instead, they are generated in natural language using a custom-tuned prompt that connects mood and food trends.

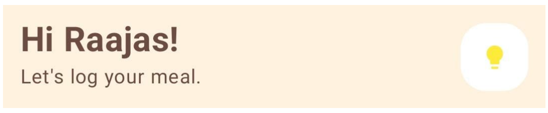


Figure 7.10: Meal Insight Button Trigger on Logging Screen

7.2.1 Data Structure and Prompt Engineering

The meal insight backend uses the `OpenAI GPT-3.5-turbo` model deployed via a `FastAPI` server to generate personalized wellness insights based on a user's last five days worth of meals history. The model is accessed using the official `OpenAI Python SDK` with authentication via a secure `.env` configuration.

Each API call expects a JSON body structured like so:

- `mealType` – (e.g., Breakfast, Lunch, Snack)
- `mealName` – (e.g., Chicken Wrap, Salad)
- `foodCategory` – (e.g., Junk, Home-cooked)
- `time` – ISO-formatted time of meal
- `moodBefore`, `moodAfter` – Integers from 1–5
- `energyBefore`, `energyAfter` – Integers from 1–5

This data is then transformed into a natural language block and injected into a carefully crafted **user prompt**. The system prompt remains constant and instructs the LLM to act as a *"wellness assistant"*, focusing on interpreting behavioral patterns and emotional changes from the input. The user prompt includes a sequential representation of the recent meals, in the format:

Lunch – Grilled Cheese at 13:00 — Mood: 3→2, Energy: 4→2

This natural language format facilitates stronger few-shot inference in the LLM, allowing it to generate empathetic, human-like reflections. The response typically includes 2–3 bullet-point insights such as:

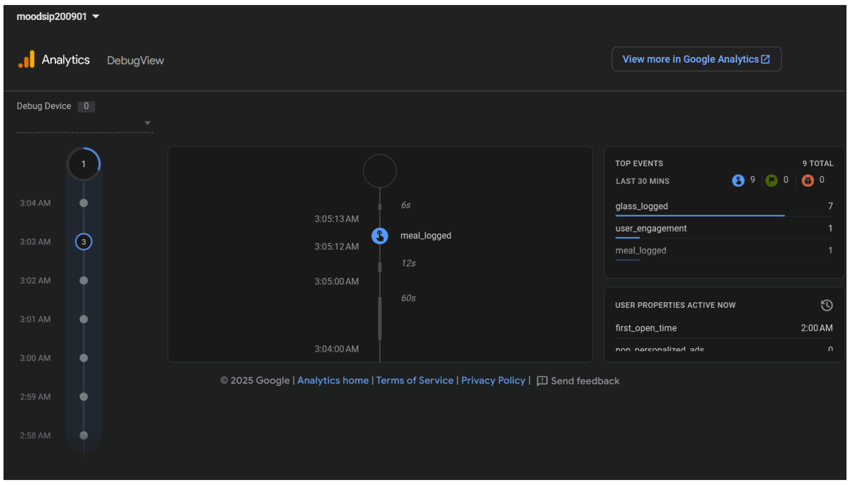
• Skipping breakfast may have affected your energy levels • Junk food for lunch led to a drop in mood

7.2.2 Choice of Model and Platform Justification

I chose `OpenAI GPT-3.5-turbo`[44] over open-source LLMs such as `Mistral`[45], `LLaMA`[46], or `Gemma`[47] due to its superior few-shot reasoning capabilities, reliability under structured prompt constraints and support for chat-completion APIs.

7.2.3 Firebase Analytics Event Logging

Each time a meal is logged within the mobile interface, a custom analytics event `meal_logged` is fired using `Firebase Analytics`. This logs the timestamp and meal type for downstream user analysis.



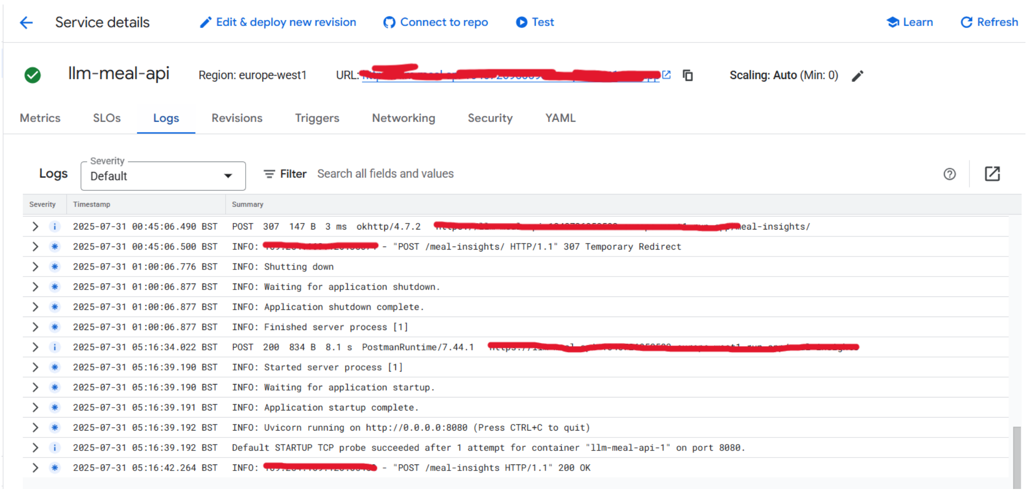
TOP EVENTS	9 TOTAL
LAST 30 MINS	9 0 0
glass_logged	7
user_engagement	1
meal_logged	1

USER PROPERTIES ACTIVE NOW	
first_open_time	2:00 AM
non_personalized_ads	0

Figure 7.11: Firebase DebugView of `meal_logged` event

7.2.4 Cloud Run Deployment

The backend is containerized via `Docker` and deployed using `Google Cloud Run`, allowing serverless execution with autoscaling and HTTPS support. The deployed endpoint is stateless and uses cold start optimization for prompt responses.



llm-meal-api Region: europe-west1 URL: [redacted] Scaling: Auto (Min: 0)

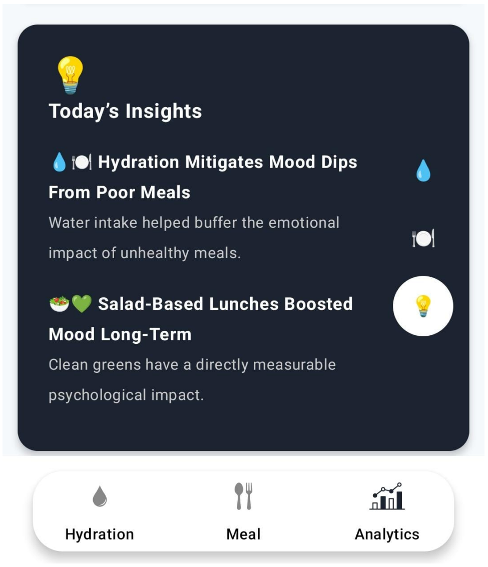
```
Severity  Timestamp                    Summary
>  i  2025-07-31 00:45:06.490 BST  POST  307  147 B  3 ms  okhttp/4.7.2  [redacted]eal-insights/
>  *  2025-07-31 00:45:06.500 BST  INFO: [redacted] - "POST /meal-insights/ HTTP/1.1" 307 Temporary Redirect
>  *  2025-07-31 01:00:06.776 BST  INFO: Shutting down
>  *  2025-07-31 01:00:06.877 BST  INFO: Waiting for application shutdown.
>  *  2025-07-31 01:00:06.877 BST  INFO: Application shutdown complete.
>  *  2025-07-31 01:00:06.877 BST  INFO: Finished server process [1]
>  i  2025-07-31 05:16:34.022 BST  POST  200  834 B  8.1 s  PostmanRuntime/7.44.1  [redacted]
>  *  2025-07-31 05:16:39.190 BST  INFO: Started server process [1]
>  *  2025-07-31 05:16:39.190 BST  INFO: Waiting for application startup.
>  *  2025-07-31 05:16:39.191 BST  INFO: Application startup complete.
>  *  2025-07-31 05:16:39.192 BST  INFO: Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
>  i  2025-07-31 05:16:39.192 BST  Default STARTUP TCP probe succeeded after 1 attempt for container "llm-meal-api-1" on port 8080.
>  *  2025-07-31 05:16:42.264 BST  INFO: [redacted] - "POST /meal-insights HTTP/1.1" 200 OK
```

Figure 7.12: Meal Insight Backend running on Google Cloud Run

7.3 Analytics Insight Model

7.3.1 Overview and Motivation

The `Analytics Insight Backend` is designed to generate a holistic snapshot of the user's wellness profile for a given day, based on both hydration and meal behaviors. This is triggered when the user presses the `insight bulb button` in the Analytics tab. Unlike the real-time hydration or recent meal LLM systems, this backend considers complete logged data for the day and synthesizes insights using a machine learning model trained on historical data.



Today's Insights

💧🍽️ Hydration Mitigates Mood Dips From Poor Meals

Water intake helped buffer the emotional impact of unhealthy meals.

🥗💚 Salad-Based Lunches Boosted Mood Long-Term

Clean greens have a directly measurable psychological impact.

Figure 7.13: Triggering insight generation from Analytics tab

7.3.2 Daily Log Preprocessing and Feature Engineering

I first transformed user data like hydration timestamps, mood/energy levels and meal entriesn into structured numerical features. This transformation is executed using a custom preprocessing pipeline written in Python.

The following derived features are calculated:

- `hydration_status`: Fraction of goal completed, defined as:

$$H_s = \frac{H_a}{H_g}$$

  where H_a is hydration actual and H_g is hydration goal.

- `hydration_time_spread`: Temporal variance of water intake (in minutes) to assess distribution throughout the day:

$$\sigma_h^2 = \frac{1}{n}\sum_{i=1}^{n}(t_i - \bar{t})^2$$

- `meal_distribution`: Number of unique meals logged from {Breakfast, Lunch, Snack, Dinner}.

- `junk_food_ratio`: Proportion of meals classified as `junk` or `fast food`.
- `mood_change` and `energy_change`:

$$\Delta M = M_{\text{after}} - M_{\text{before}}, \quad \Delta E = E_{\text{after}} - E_{\text{before}}$$

- `hydration_impact_on_energy`:

$$HIE = H_s \cdot E_{\text{after}}$$

- `meal_timing_lateness`: Number of meals taken after 21:00.
- `skipped_meals`: Missed meals from {Breakfast, Lunch, Dinner}.
- `salad_mood_avg`: Mean post-meal mood score for meals containing `salad`.

This results in a unified row of features for each day. Sample dataset structure:

`hydration_goal`	8
`hydration_actual`	5
`hydration_status`	0.625
`hydration_time_spread`	95.2
`meal_distribution`	3
`junk_food_ratio`	0.5
`mood_change`	1
`energy_change`	-1
...	...

7.3.3 Modeling Architecture and Outputs

I formulated three learning tasks:

1. Multi-Output Regression

Using the extracted features $\mathbf{X} \in \mathbb{R}^{n \times d}$, we can predict multiple continuous targets simultaneously:

$$\hat{\mathbf{y}} = f_r(\mathbf{X}), \quad \text{where } f_r \text{ is a Random Forest Regressor}$$

The target vector $\mathbf{y} \in \mathbb{R}^7$ includes:

$$\{\texttt{hydration\_status}, \texttt{hydration\_time\_spread}, \texttt{mood\_change}, \texttt{energy\_change}\}$$

$$\{\texttt{hydration\_impact\_on\_energy}, \texttt{daily\_mood\_variability}, \texttt{daily\_energy\_variability}\}$$

I used the **Mean Absolute Error (MAE)** as the evaluation metric:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

2. Multi-Label Classification

Several binary insight flags are predicted using:

$$\hat{\mathbf{z}} = f_c(\mathbf{X}), \quad \mathbf{z} \in \{0,1\}^4$$

These include:

- `low_hydration`: $H_s < 0.40$
- `junk_heavy_day`: junk_food_ratio > 0.5
- `late_meals`: At least one meal after 9pm
- `skipped_breakfast`: Breakfast not logged

The model uses a **MultiOutputClassifier** with `RandomForestClassifier`.

3. Meta Day-Quality Classifier

I have defined a label $\mathcal{Y}_{\text{quality}} \in \{\text{optimal}, \text{recovering}, \text{stable}, \text{at\_risk}\}$ with the following rule-based derivation:

$$\mathcal{Y}_{\text{quality}} = \begin{cases} \texttt{at\_risk} & \text{if low\_hydration and junk\_heavy\_day} \\ \texttt{optimal} & \text{if } H_s > 0.9 \wedge \Delta M > 0 \wedge \Delta E > 0 \\ \texttt{recovering} & \text{if } \Delta M > 0 \vee \Delta E > 0 \\ \texttt{stable} & \text{otherwise} \end{cases}$$

Encoded numerically using a `LabelEncoder` and trained using another `RandomForestClassifier`.

7.3.4 Training, Model Selection and Output Interpretation

The analytics insight system utilizes multiple machine learning models trained on a unified daily dataset to infer behavioral insights from user logs. These models were trained using an 80/20 train-test split to ensure generalizability, with fixed random seeds for reproducibility.

Why Random Forest?

All three models, the regression engine, multi-label insight classifier and day quality predictor are based on the Random Forest algorithm due to its robustness, interpretability and performance on tabular health data[42]. Random Forests are ensemble methods that average predictions from multiple decision trees to reduce variance and overfitting. For regression, the output is:

$$\hat{y}_{\text{reg}} = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

For classification, each tree votes and the final prediction is the majority vote:

$$\hat{y}_{\text{cls}} = \text{mode}(\{h_t(x)\}_{t=1}^{T})$$

This technique is highly suitable for the mixed feature types (numeric + categorical) and non-linear dependencies present in mHealth behavioral data.

Model Outputs and How They Power Insights

Each model contributes to the overall understanding of the user's wellness behavior:

- **MultiOutput Regressor**: Predicts continuous wellness trends such as:

  $$\hat{M}_{\text{change}}, \hat{E}_{\text{change}}, \hat{H}_{\text{impact}}, \hat{\sigma}_h, \ldots$$

  These are used to generate insights like:

  > "Your mood increased significantly today—keep up the balanced hydration!" or
  > "You drank all your water in a short time window. Try spacing it out more."

- **MultiOutput Classifier**: Flags binary conditions like:
  - `low_hydration` → "You were underhydrated today."
  - `junk_heavy_day` → "High intake of junk food noticed today."
  - `skipped_breakfast` → "Skipping breakfast correlated with low energy."

- **Day Quality Classifier**: Synthesizes all metrics into a high-level label:

  $$\text{DayQuality} \in \{\texttt{optimal}, \texttt{recovering}, \texttt{stable}, \texttt{at\_risk}\}$$

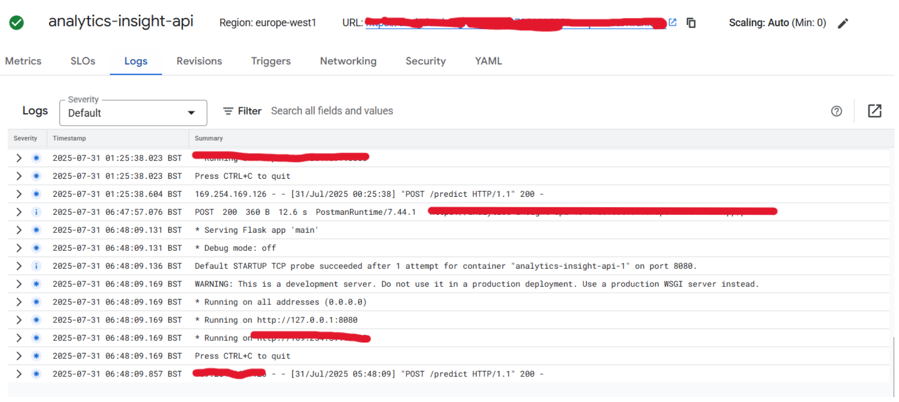
  This label governs the framing of the daily summary card and drives reinforcement-based feedback:

  > "**Optimal day:** Everything you did today aligned well—hydration, meals and mood!"
  > "**At-risk day:** Low water intake and heavy junk food—your body needs balance tomorrow."

Above examples are just for understanding. For actual insights implemented, `InsightUtils.kt` can be referred in the codebase.

7.3.5 Deployment via Google Cloud Run

To ensure scalability, reliability and low-latency, analytics insight models were containerized and deployed to **Google Cloud Run**. The trained models and preprocessing logic were bundled using `FastAPI` inside a Docker container and deployed as a REST endpoint with HTTPS. This endpoint is invoked directly from the MoodSip Android app using Retrofit.



analytics-insight-api Region: europe-west1 URL: [redacted] Scaling: Auto (Min: 0)

```
Severity  Timestamp                    Summary
>         2025-07-31 01:25:38.023 BST  [redacted]
>         2025-07-31 01:25:38.023 BST  Press CTRL+C to quit
>         2025-07-31 01:25:38.604 BST  169.254.169.126 - - [31/Jul/2025 00:25:38] "POST /predict HTTP/1.1" 200 -
>   i     2025-07-31 06:47:57.076 BST  POST  200  360 B  12.6 s  PostmanRuntime/7.44.1  [redacted]
>         2025-07-31 06:48:09.131 BST  * Serving Flask app 'main'
>         2025-07-31 06:48:09.131 BST  * Debug mode: off
>   i     2025-07-31 06:48:09.136 BST  Default STARTUP TCP probe succeeded after 1 attempt for container "analytics-insight-api-1" on port 8080.
>         2025-07-31 06:48:09.169 BST  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
>         2025-07-31 06:48:09.169 BST  * Running on all addresses (0.0.0.0)
>         2025-07-31 06:48:09.169 BST  * Running on http://127.0.0.1:8080
>         2025-07-31 06:48:09.169 BST  * Running on [redacted]
>         2025-07-31 06:48:09.169 BST  Press CTRL+C to quit
>         2025-07-31 06:48:09.857 BST  [redacted] - - [31/Jul/2025 05:48:09] "POST /predict HTTP/1.1" 200 -
```

Figure 7.14: Cloud Run Deployment of Analytics Insight Engine

7.3.6 Average Mood and Energy Computation for Chart Tooltips

To improve the interpretability of the weekly trend charts on the Analytics screen, each data point displays a tooltip that shows the *average mood* or *energy level* for that specific day. These values are computed dynamically by aggregating each meal's `moodBefore`, `moodAfter`, `energyBefore`,and `energyAfter` fields.

Computation Logic:

Let $M = \{m_1, m_2, ..., m_n\}$ be the set of meal logs for a given day. Then:

$$\text{Average Mood}_{\text{day}} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{m_i.\text{moodBefore} + m_i.\text{moodAfter}}{2} \right)$$

$$\text{Average Energy}_{\text{day}} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{m_i.\text{energyBefore} + m_i.\text{energyAfter}}{2} \right)$$

These averages are displayed when a user taps on a data point in the chart

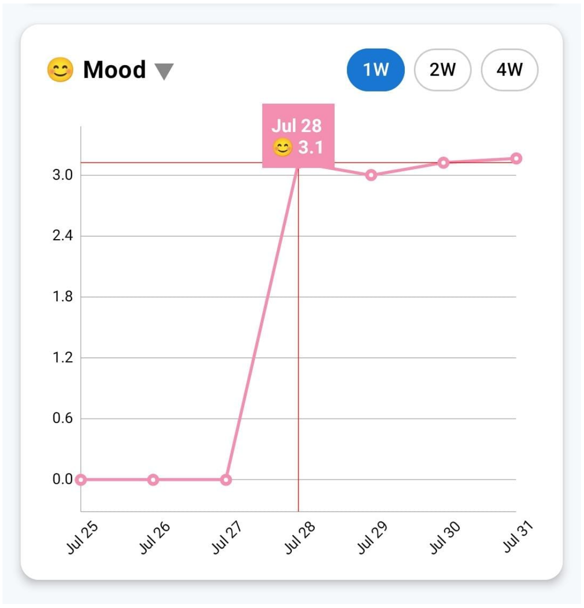


Figure 7.15: Tooltip Showing Average Mood Level

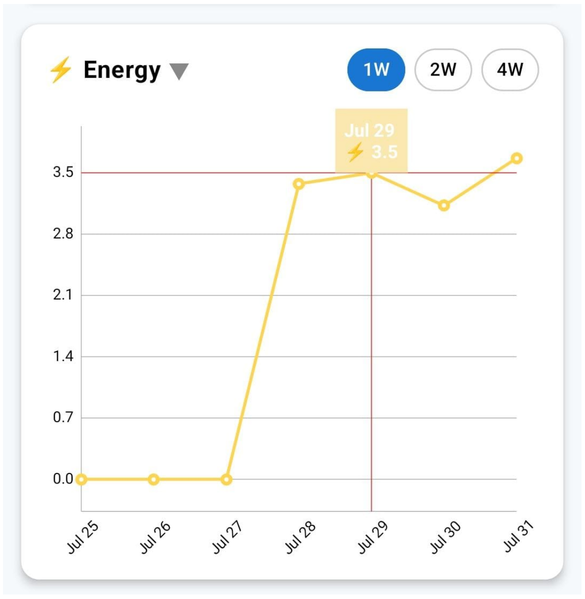


Figure 7.16: Tooltip Showing Average Energy Level

Chapter 8

System Summary

Given the complexity and breadth of functionalities integrated, it is understandable that navigating through the various modules may appear overwhelming at first. Therefore, this chapter is intended to provide a clear and structured overview of the system as a whole.

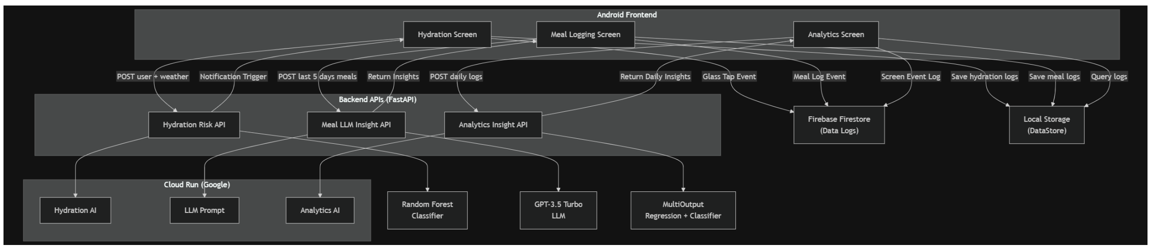


Figure 8.1: Diagram illustrating interactions between Android screens, backend APIs, ML/LLM deployments on Cloud Run and data storage layers.

This diagram outlines the architectural flow between the frontend Android screens, backend FastAPI services and the deployed AI/ML modules on Google Cloud Run. The Hydration Screen interacts with the Hydration Risk API, which uses a Random Forest Classifier to predict user hydration risk based on behavior and weather data. The Meal Logging Screen sends recent meals to the LLM Insight API, which uses GPT-3.5 to return wellness insights. The Analytics Screen invokes the Analytics Insight API, powered by a multi-output regression and classification model, to deliver personalized daily insights. All interactions involve reading from and writing to Firebase and Local DataStore, ensuring persistent and synced user data across modules.

Chapter 9

Testing

9.1 Backend Model Evaluation

9.1.1 Metrics used

To evaluate the performance of the trained machine learning models, I used the following standard metrics:

- **Confusion Matrix**: A tabular summary of true vs. predicted values. It helps visualize the model's ability to distinguish between classes.
- **Precision**: The ratio of true positives to all predicted positives. Indicates correctness of positive predictions.
- **Recall**: The ratio of true positives to all actual positives. Indicates completeness of positive predictions.
- **F1 Score**: Harmonic mean of precision and recall. A balanced measure of accuracy for imbalanced datasets.

9.1.2 Classification Performance

The classification performance for each binary label was evaluated using precision, recall and F1-score.

Class	Precision	Recall	F1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00

Table 9.1: Classification metrics for `low_hydration`.

Class	Precision	Recall	F1-score
0	0.95	0.99	0.97
1	0.00	0.00	0.00

Table 9.2: Classification metrics for `late_meals`.

Class	Precision	Recall	F1-score
0	0.77	0.82	0.79
1	0.50	0.42	0.45

Table 9.3: Classification metrics for `junk_heavy_day`.

Class	Precision	Recall	F1-score
0	0.95	0.69	0.80
1	0.87	0.98	0.92

Table 9.4: Classification metrics for `skipped_breakfast`.

These results demonstrate high performance for `low_hydration` and `skipped_breakfast`, while performance for `late_meals` is low due to extreme class imbalance.

9.1.3 Confusion Matrices

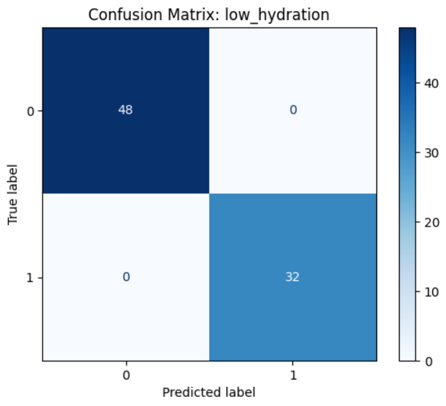


Figure 9.1: CM: `low_hydration`

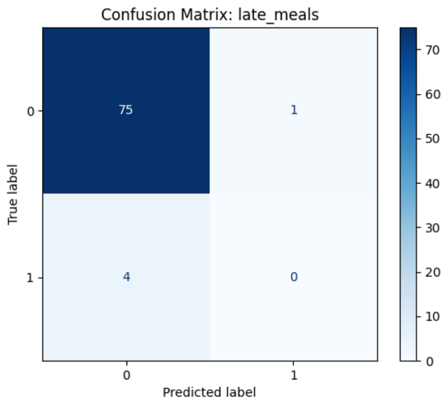


Figure 9.2: CM: `late_meals`

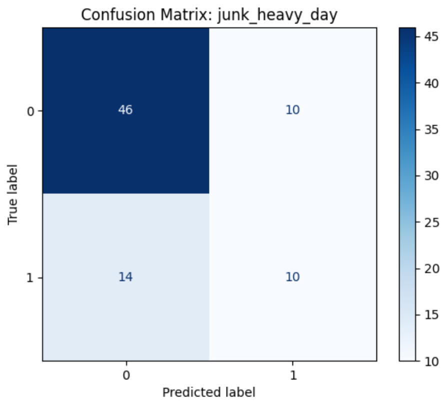


Figure 9.3: CM: `junk_heavy_day`

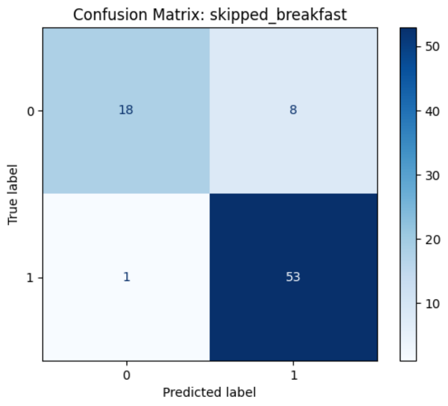


Figure 9.4: CM: `skipped_breakfast`

Each matrix visualizes true vs. predicted labels. Most models show strong diagonal dominance, with minor confusion in imbalanced classes.

9.1.4 Precision-Recall Curves

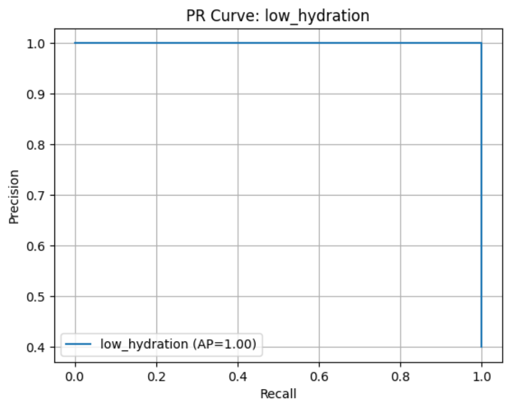


Figure 9.5: PRC: `low_hydration`

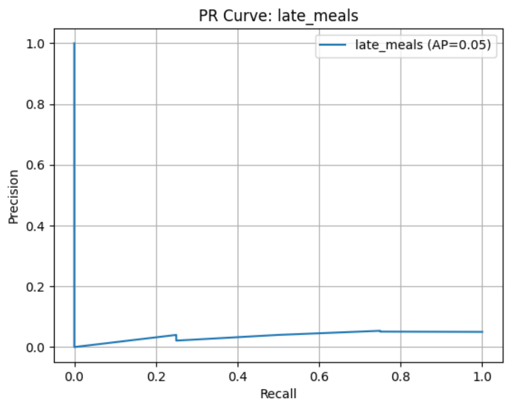


Figure 9.6: PRC: `late_meals`

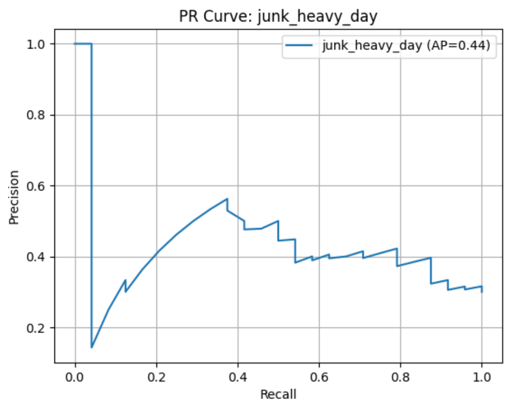


Figure 9.7: PRC: `junk_heavy_day`

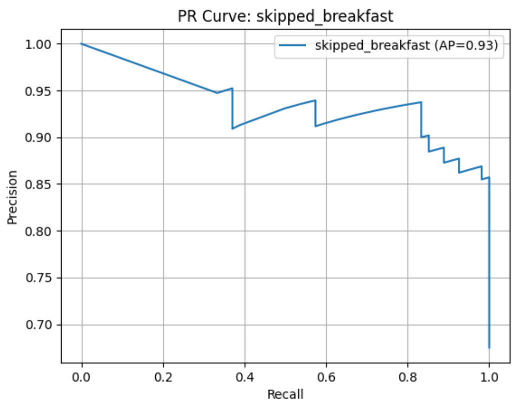
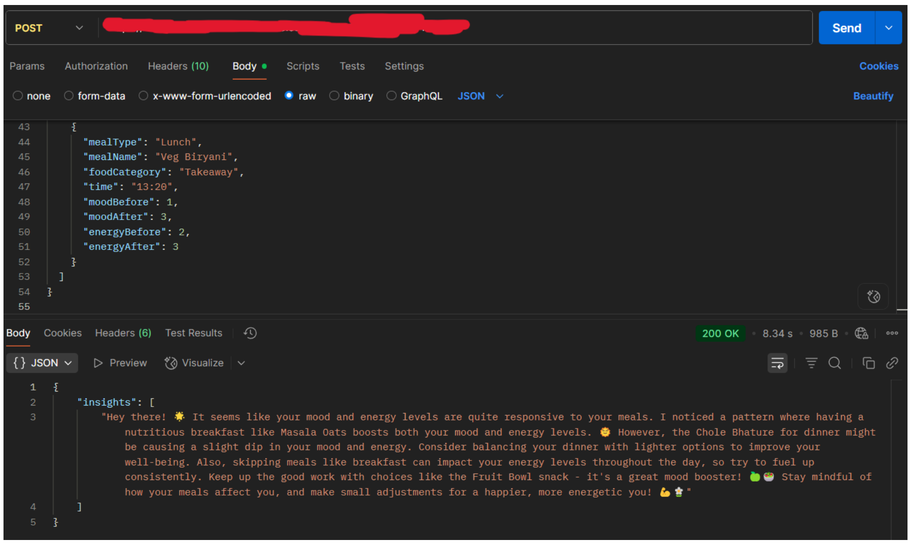


Figure 9.8: PRC: `skipped_breakfast`

The curves confirm model confidence. `low_hydration` and `skipped_breakfast` show strong separability, while `late_meals` struggles due to data imbalance.

9.1.5 Postman-Based Endpoint Testing

To verify backend behavior during development, I tested the API with `Postman`. This allowed simulation of payloads and inspection of returned insights before app-side integration.



POST [redacted] Send

Params | Authorization | Headers (10) | Body | Scripts | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON | Beautify

```
    {
      "mealType": "Lunch",
      "mealName": "Veg Biryani",
      "foodCategory": "Takeaway",
      "time": "13:20",
      "moodBefore": 1,
      "moodAfter": 3,
      "energyBefore": 2,
      "energyAfter": 3
    }
  ]
}
```

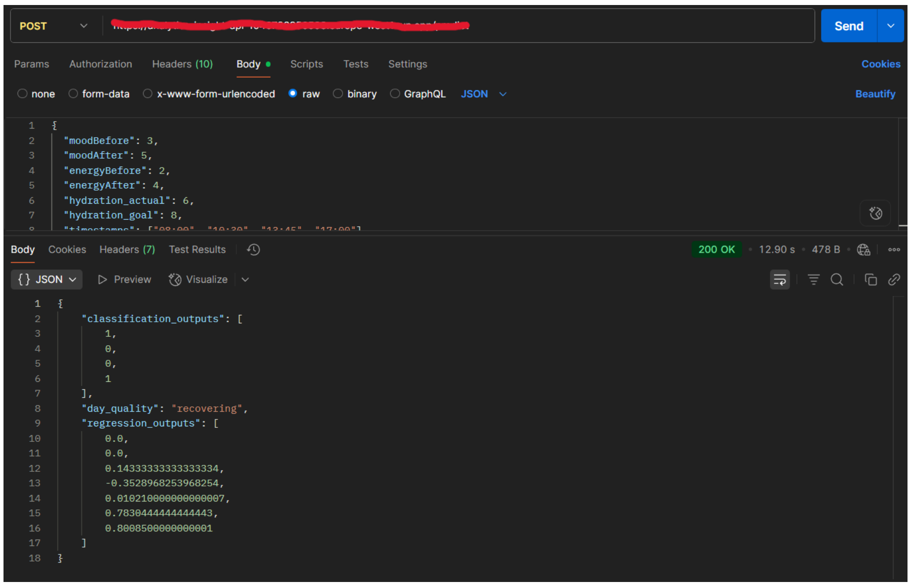
Body | Cookies | Headers (6) | Test Results — 200 OK • 8.34 s • 985 B

{} JSON | Preview | Visualize

```
{
    "insights": [
        "Hey there! ☀️ It seems like your mood and energy levels are quite responsive to your meals. I noticed a pattern where having a
            nutritious breakfast like Masala Oats boosts both your mood and energy levels. 😊 However, the Chole Bhature for dinner might
            be causing a slight dip in your mood and energy. Consider balancing your dinner with lighter options to improve your
            well-being. Also, skipping meals like breakfast can impact your energy levels throughout the day, so try to fuel up
            consistently. Keep up the good work with choices like the Fruit Bowl snack - it's a great mood booster! 🍏🥗 Stay mindful of
            how your meals affect you, and make small adjustments for a happier, more energetic you! 💪🌟"
    ]
}
```

Figure 9.9: Postman Test for LLM Meal Insight API

The analytics insight API was also thoroughly tested using **Postman**.



POST [redacted] Send

Params | Authorization | Headers (10) | Body | Scripts | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON | Beautify

```
{
  "moodBefore": 3,
  "moodAfter": 5,
  "energyBefore": 2,
  "energyAfter": 4,
  "hydration_actual": 6,
  "hydration_goal": 8,
  "timestamps": ["08:00", "10:30", "13:45", "17:00"]
```

Body | Cookies | Headers (7) | Test Results — 200 OK • 12.90 s • 478 B

{} JSON | Preview | Visualize

```
{
    "classification_outputs": [
        1,
        0,
        0,
        1
    ],
    "day_quality": "recovering",
    "regression_outputs": [
        0.0,
        0.0,
        0.14333333333333334,
        -0.3528968253968254,
        0.010210000000000007,
        0.7830444444444443,
        0.8008500000000001
    ]
}
```

Figure 9.10: Postman Test for Analytics Insight API

9.2 Software Testing

Functional testing emerged as the most practical and impactful strategy during the development phase. This approach focused on verifying that entire features and workflows operated as intended when used through the user interface.

Throughout development, the application was run repeatedly on both emulators and physical Android devices. Each interactive path was tested with varied data inputs, simulating realistic user behaviour. For example, the hydration screen was validated to ensure the correct number of glasses was displayed and the progress bar was updated in sync with backend records. Similarly, meal entries were created, edited and verified for accurate logging with the correct timestamps.

Insight retrieval was also functionally tested by simulating meal and hydration logs and observing the resulting AI-driven cards. In each case, I ensured that the features were responsive, displayed relevant content and handled edge cases (like incomplete logs).

9.2.1 Challenges with Unit Testing in MoodSip

Although unit testing is a cornerstone of software quality assurance, implementing it in MoodSip proved to be both time-consuming and less effective, given the architectural decisions and relatively tight coupling between UI and logic(despite MVVM).

Much of the application's behaviour is tightly integrated within Jetpack Compose's reactive state system. The ViewModels manage logic that depends on mutable state and composable lifecycle scopes. Honestly, testing these without running a full Compose environment required complex mocking of state holders, coroutine dispatchers and live data flows.

Furthermore, several features rely on local persistence via DataStore and cloud services like Firebase and Google Cloud Run. While technically mockable, isolating these for pure unit tests required significant infrastructure just to simulate realistic environments, which in some cases would make the tests harder to maintain than the features themselves.

As such, I deprioritized unit testing in favour of broader functional tests. This decision, while not ideal for long-term regression coverage, was deliberate and based on the fast iteration cycle of the app.

9.2.2 Opportunities for Further Testing

Despite the focus on functional validation during initial development, MoodSip can benefit from deeper testing layers in future iterations. Based on my experience as a developer, I can say one such strategy is **User Acceptance Testing (UAT)**, where the application is deployed to a group of real users for daily use under natural conditions. UAT helps uncover issues not visible to the developer, such as usability hurdles, accessibility concerns or unexpected behaviours across device ecosystems.

In addition, automated UI testing using tools such as Espresso or Jetpack Compose Testing APIs could be integrated. These tools allow developers to write reusable test scenarios that simulate user gestures and validate UI responses, helping ensure that future changes do not break existing functionality.

For a wellness-focused app, performance and memory testing is also important. The hydration tracker includes animated progress views, sound effects and notification systems, all of which can impact battery and resource usage. Using **Android Profiler**, one can measure memory leaks, slow frame renders and CPU spikes to optimise the app for long-term daily use.

Lastly, testing across a device farm such as **Firebase Test Lab** would ensure compatibility and robustness across Android versions and manufacturer-specific behaviours. Given that background processes and battery optimisations vary across OEMs, such testing can help catch inconsistencies in notification timing or background task failures.

9.3 System Usability Scale (SUS)

9.3.1 What is the SUS Framework?

The System Usability Scale (SUS), developed by John Brooke in 1986, is a quick and reliable tool for measuring the perceived usability of a software system. It consists of a 10-item Likert-scale questionnaire with responses ranging from "Strongly Disagree" to "Strongly Agree". The SUS is technology-agnostic and applicable across interfaces, making it a staple in usability evaluation research[48].

Each item alternates between positive and negative phrasing, reducing response bias.

9.3.2 SUS Questionnaire Items

Participants respond to the following standardized 10 items[48]:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need technical support to use this system.
5. I found the various functions in this system well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Odd-numbered questions are positively worded and even-numbered questions are negatively worded. This affects the scoring formula.

9.3.3 Alternatives to SUS

- **NASA-TLX:** Focuses on workload rather than usability.
- **UMUX-Lite:** A two-question metric with high correlation to SUS.
- **SUPR-Q:** Measures usability plus trust and appearance (for websites).
- **QUIS:** Questionnaire for User Interface Satisfaction, more granular but longer.

SUS remains the industry standard due to its simplicity and reliability across domains[49].

9.3.4 SUS Data Collection for MoodSip

I conducted an internal usability survey with 15 participants aged between 21 and 27. Each of the participants interacted with MoodSip's UI and understood the rationale behind every single feature. Each of these interactions lasted for about 15 minutes wherein the participants were made aware of what MoodSip could offer to their wellness habits when it would be deployed to a full-scale. Their responses were recorded on a 5-point scale (1 = Strongly Disagree, 5 = Strongly Agree) using a Google Form linked here.[50].

Table 9.5: SUS Questionnaire Responses (Raw Scores)

User	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	**Raw Total**
P1	5	2	5	1	4	2	5	1	4	2	31
P2	4	2	4	2	4	1	4	2	4	1	28
P3	5	1	5	1	5	1	5	1	5	1	30
P4	4	2	4	2	3	2	4	1	4	1	27
P5	5	1	5	1	4	1	5	1	5	1	29
P6	4	2	4	2	4	2	4	2	4	2	30
P7	3	3	4	2	3	3	4	3	4	2	31
P8	4	1	4	1	4	2	5	1	4	1	27
P9	5	2	5	2	5	2	5	1	5	1	33
P10	4	2	4	2	4	2	4	2	4	2	30
P11	5	1	5	1	5	1	5	1	5	1	30
P12	5	1	5	1	4	1	5	1	4	1	28
P13	4	2	5	2	4	2	5	2	4	1	31
P14	3	3	4	3	3	2	4	3	4	2	31
P15	5	1	5	1	5	1	5	1	5	1	30

9.3.5 SUS Score Calculation

Step 1: Score Transformation

- For odd-numbered items (Q1, Q3, ..., Q9): score = response - 1
- For even-numbered items (Q2, Q4, ..., Q10): score = 5 - response

Example for P1:

$$\text{Positive Qs (Q1, Q3, Q5, Q7, Q9)} : (5-1)+(5-1)+(4-1)+(5-1)+(4-1) = 15$$
$$\text{Negative Qs (Q2, Q4, Q6, Q8, Q10)} : (5-2)+(5-1)+(5-2)+(5-1)+(5-2) = 14$$
$$\text{Total SUS Score} = (15+14)\times 2.5 = \mathbf{72.5}$$

Step 2: Repeat for all participants

We apply this to each row to compute total SUS score.

Table 9.6: Final SUS Scores Per Participant

Participant	**SUS Score (/100)**
P1	72.5
P2	65.0
P3	75.0
P4	62.5
P5	72.5
P6	70.0
P7	67.5
P8	67.5
P9	80.0
P10	70.0
P11	75.0
P12	70.0
P13	72.5
P14	67.5
P15	75.0
Mean SUS Score	**70.83**

9.3.6 Interpretation of Results

A SUS score of **70.83** indicates a high degree of perceived usability. Based on industry SUS benchmarks[49]:

- **Scores above 80** are excellent.
- **Scores between 68 - 79** are considered good.
- **Scores between 51–67** indicate average usability with potential for refinements.
- **Scores below 51** are considered unusable.

This aligns with the output that MoodSip is intuitive, quick to learn and satisfying for daily usage.

9.3.7 Conclusion from SUS Analysis

The SUS evaluation validates that the frontend design decisions like modular navigation and reactivity, translate well to user experience. It also identifies areas (e.g., complexity or learnability) that can benefit from refinement, particularly for new users.

This structured approach to usability validation I believe, enhances the credibility and reliability of the frontend design and supports its integration into wider public deployments.

Chapter 10

Legal, Social, Ethical and Professional Issues

Throughout the development of the MoodSip mobile application, careful attention was paid to ensuring full alignment with the British Computer Society (BCS) Code of Conduct and Code of Good Practice. Rather than viewing these principles as just checklists, I consciously embedded them into my design choices, development practices and deployment workflows. This chapter explores how legal compliance, social responsibility, ethical decision-making and professional integrity have been actively considered during the course of this project.

10.1 Data Collection, Security and Privacy

MoodSip has been designed from the ground up to minimize data collection and avoid unnecessary exposure of user information. The data that is collected includes hydration logs (specifically, the number of glasses consumed and the timestamp), meal records with mood and energy ratings before and after eating and the current weather temperature. Importantly, no personally identifiable information (PII) such as names, email addresses or biometric data is stored or required by the prototype.

Backend communications are made via secure HTTPS calls to stateless APIs hosted on Google Cloud Run. These endpoints do not retain any session information and user logs remain only on the client or in Firebase Firestore under access-controlled conditions. To further reinforce security, all external API requests are rate-limited and validated. The app follows the principle of least privilege, ensuring that only the required permissions are requested. In terms of transparency, the entire codebase is publicly available and version-controlled, allowing for audits and collaborative feedback. I have made a deliberate effort to keep the backend architecture closed-loop, meaning that it introduces no new vulnerabilities on the network.

10.2 Intellectual Property and Licensing

MoodSip respects intellectual property boundaries at every level. All third-party libraries—such as Jetpack Compose, Retrofit and Firebase SDKs are used in accordance with their permissive licenses (mostly Apache or MIT). AI capabilities rely on OpenAI's GPT-3.5 API, which is accessed through the appropriate commercial usage agreement.

The original work within this project including the logic behind hydration goal adjustments, insight pipelines, UI design and AI model training is authored by me and made available under an open academic license. This transparency ensures that others can replicate, critique or build upon my work. If MoodSip were ever commercialized, a new licensing review would be essential, particularly for the use of proprietary APIs and open-source redistribution rules under UK Copyright and Patent law.

10.3 Accessibility and Inclusivity

Making MoodSip usable for as many people as possible was a design goal. The app interface features high contrast colors and scalable text sizes for readability. Touch targets are appropriately sized for different screen resolutions, which is essential for mobile accessibility. In the hydration and meal logging screens, indicators and sliders are clearly labeled and consistent iconography is used to support users with cognitive or visual impairments.

I also added tooltips and context-sensitive visual cues where possible to enhance clarity. While these are initial steps, future versions of MoodSip could expand accessibility with support for screen readers, voice-based input and color-blind-friendly palettes. This section also aligns with *"You make IT for everyone"*, a core pillar of the British Computer Society code.

10.4 Sustainability and Environmental Impact

Sustainability has been integrated not only at the technical level but also through architectural decisions. Most computations are performed on the client side, reducing server load and energy consumption. When remote processing is required—such as for AI insights—it is delegated to Google Cloud Run, a platform that is working toward fully carbon-neutral operations by 2030[51].

Moreover, by using Jetpack DataStore for local data caching, the app avoids making excessive network calls, thereby conserving bandwidth and reducing the environmental impact associated with cloud data transfers. These optimizations contribute to a greener and more energy-efficient application footprint.

10.5 Economic and Commercial Considerations

MoodSip is primarily intended as a research prototype, but I also evaluated its commercial potential. One of the strengths of the architecture is its cost-efficiency. It uses infrastructure like Cloud Run and Firestore, which scale well and have free usage tiers suitable for low-traffic applications. This makes it feasible to operate MoodSip at minimal cost.

If a commercial route is explored, premium features such as advanced analytics, goal suggestions or personalized nutrition planning could be offered in a freemium model. This shift, however, would require revisiting data handling policies, particularly around GDPR and user consent. Any monetization strategy must go hand-in-hand with responsible data governance.

10.6 Software Trustworthiness

The integrity and trustworthiness of the software were core values throughout development. I ensured that all insights delivered to the user were either derived from explainable models or from clearly documented rule-based logic. No opaque decision-making is introduced and users are given visual explanations of what influences the AI outputs.

Additionally, the project's open-source nature means anyone can inspect the code, verify the implementation and suggest improvements. This transparency, combined with consistent functional testing and a well-modularized MVVM architecture, helps minimize bugs and reinforces trust. No hidden analytics or manipulative patterns (dark UX) are used—what you see is truly what you get.

I believe my application demonstrates alignment with ethical, legal and professional standards. It respects user autonomy, protects personal data, promotes inclusivity and remains transparent about its AI usage. I have tried my best to practically implement them into the software's architecture and development workflow and not just understand them as pure theory.

Chapter 11

Conclusion

The core objective of this project was to design and implement an intelligent, habit-forming wellness application, one that could help users stay hydrated and better understand the effects of their meals on mood and energy levels. This idea stemmed from a real gap in the current wellness app ecosystem: most trackers either log hydration or meals, but very few attempt to uncover the correlation between the two, even fewer leverage AI to deliver personalized insights rooted in actual behavioral data and none focus on both of the 2 most fundamental metrics- Mood and Energy; the ones who actually determine our daily health and wellness.

From the outset, the goal was not just to log, but to interpret those logs in meaningful ways. With this, MoodSip was built as a modular, AI-powered mobile application that does more than just track habits. It analyzes user behavior, adapts goals based on weather and patterns, predicts risk of dehydration and reflects on the emotional consequences of daily meal choices. The integration of LLM-based insights, calendar-based analytics and visualizations like hydration spread and mood/energy change plots created a wellness dashboard that's both data-rich and user-friendly. The analytics insight especially which takes into account, the amount of junk food the user has had, meals they have skipped, ratio of healthy food consumed etc is truly life-changing.

Achieving this involved numerous components working in unison — from Firebase for real-time data logging, to Cloud Run-hosted ML models, to prompt-engineered OpenAI APIs and of course, a Kotlin Jetpack Compose-based UI. A significant amount of testing and refinement ensured that the app remained responsive, privacy-respecting and modular enough for future expansion. In doing so, I strongly believe MoodSip is at complex intersection of user experience, data analysis and AI.

11.1 Limitations

Despite the robust architecture and intelligent backend, there are still several limitations to acknowledge. First, the current meal insights have a recency bias of 5 days. The backend LLM would be able to generate even more personalized insights if meal data could be sent from the UI not just as a simple JSON Array but rather a closely condensed package of information.

Secondly, user authentication has not yet been implemented, which limits the ability to sync data across devices or maintain long-term user profiles. Moreover, while the app integrates weather-aware hydration adjustment, it does not consider physical activity or illness, both of which are major factors in hydration and energy needs.

Accessibility was carefully considered in the design however real-world usability tests with people with disabilities were limited, meaning some edge cases may not yet be fully supported.

Finally, although Firebase was used for data security, full end-to-end encryption, consent management or GDPR-style data export features are not yet in place. These are essential for compliance if MoodSip is ever scaled commercially or released in multiple jurisdictions.

11.2 Future Work

There are many promising directions for future development. The first priority is to implement secure user authentication and user-specific data sync. Once users can log in and store their data on the cloud securely, pattern detection and health trend modeling will become far more powerful.

Another enhancement would be the introduction of macro-nutrient logging — allowing users to specify carbohydrate, protein and fat content of their meals. With this, the model can deliver insights not just on food categories but on nutritional composition.

Voice-based logging and AI chatbot integration could also make the app even more accessible, particularly for visually impaired users. Support for haptic feedback and ambient reminders (e.g., hydration nudges during low activity) would make the experience more immersive.

On the AI side, moving beyond pre-trained models to real-time personalization, federated learning or reinforcement-based models could greatly improve the contextual accuracy of the insights. Additional sensors like wearables or integration with fitness trackers could also provide deeper signals for stress, sleep or exertion.

Most of the limitations listed earlier can be systematically addressed through these future enhancements.

11.3 Personal Reflection

This project has been one of the most challenging experiences I have had. I chose to implement this in Kotlin with the aim of learning a completely new programming language for improving my skillset. As a result, the learning curve was steep and required a lot of technical prep work that was not even directly related to this project. My own background is mostly backend development so building a full-featured, responsive and visually appealing mobile UI from scratch required me to step far outside of my comfort zone. Looking back, I can confidently say MoodSip represents a complete blend of design, logic, data science and user empathy. It's not just a tracker, it's an intelligent companion with serious potential in the health and wellness tracking market. I look forward to seeing where this app can go next.

Bibliography

[1] F. Alkhuzaimi, D. Rainey, C. B. Wilson, and J. Bloomfield, "The impact of mobile health interventions on service users' health outcomes and the role of health professions: a systematic review of systematic reviews-protocol," *Syst. Rev.*, vol. 13, no. 1, p. 199, Jul. 2024.

[2] A. Mustafa, N. Ali, J. S. Dhillon, G. Alkawsi, and Y. Mohamed, "User engagement and abandonment of mhealth: A cross-sectional survey," *Healthcare*, vol. 10, p. 221, 01 2022.

[3] C. Tu, A. Russo, and Y. Zhang, "A checklist for the usability evaluation of artificial intelligence (ai) mhealth applications graphical user interface," in *Design, User Experience, and Usability*, A. Marcus, E. Rosenzweig, and M. M. Soares, Eds. Cham: Springer Nature Switzerland, 2024, pp. 324–337.

[4] "MyFitnessPal - Wikipedia — en.wikipedia.org," https://en.wikipedia.org/wiki/MyFitnessPal.

[5] WaterMinder, "WaterMinder - best water tracker app for your hydration needs! — waterminder.com," https://waterminder.com/#features.

[6] "samsung-health — samsung.com," https://www.samsung.com/us/support/owners/app/samsung-health.

[7] "Features — cronometer.com," https://cronometer.com/features/.

[8] Y. Cao, J. Li, X. Qin, and B. Hu, "Examining the effect of overload on the MHealth application resistance behavior of elderly users: An SOR perspective," *Int. J. Environ. Res. Public Health*, vol. 17, no. 18, p. 6658, Sep. 2020.

[9] L. Dennison, L. Morrison, G. Conway, and L. Yardley, "Opportunities and challenges for smartphone applications in supporting health behavior change: qualitative study," *J. Med. Internet Res.*, vol. 15, no. 4, p. e86, Apr. 2013.

[10] D. X. Yu, B. Parmanto, B. E. Dicianno, V. J. Watzlaf, and K. D. Seelman, "Accessibility needs and challenges of a mhealth system for patients with dexterity impairments," *Disability and Rehabilitation: Assistive Technology*, vol. 12, no. 1, pp. 56–64, 2017, pMID: 26153097. [Online]. Available: https://doi.org/10.3109/17483107.2015.1063171

[11] A. G. D. Benedetto, "Counting calories is going to get harder with MyFitnessPal — theverge.com," https://www.theverge.com/2022/8/25/23321408/myfitnesspal-weight-loss-app-barcode-scanning-premium-paywall.

[12] P. Kumar, “MVVM & MVC — pradeepgpre,” https://medium.com/@pradeepgpre/mvvm-mvc-7e36f508018c.

[13] “Architectural Pattern - Model–view–presenter (MVP) — dev.to,” https://dev.to/binoy123/architectural-pattern-model-view-presenter-mvp-28hl.

[14] “Basic syntax — Kotlin — kotlinlang.org,” https://kotlinlang.org/docs/basic-syntax.html.

[15] “Write your app — Android Studio — Android Developers — developer.android.com,” https://developer.android.com/studio/write.

[16] “Google Cloud Documentation — cloud.google.com,” https://cloud.google.com/docs.

[17] “Firebase Documentation — firebase.google.com,” https://firebase.google.com/docs.

[18] OpenWeatherMap, “Weather api - openweathermap,” https://openweathermap.org/api, 2023.

[19] Google Android Developers, “Jetpack compose — android developers,” https://developer.android.com/jetpack/compose, 2024.

[20] ——, “Guide to app architecture - mvvm,” https://developer.android.com/topic/architecture, 2023.

[21] ——, “Workmanager api reference,” https://developer.android.com/topic/libraries/architecture/workmanager, 2024.

[22] I. Square, “Retrofit: Type-safe http client for android and java,” https://square.github.io/retrofit/, 2023.

[23] JetBrains, “Kotlin coroutines and flow,” https://kotlinlang.org/docs/coroutines-overview.html, 2023.

[24] Google Android Developers, “Jetpack datastore,” https://developer.android.com/topic/libraries/architecture/datastore, 2024.

[25] PhilJay, “Mpandroidchart - a powerful chart library for android,” https://github.com/PhilJay/MPAndroidChart, 2023.

[26] M. Keppeler, “Modern android calendar components,” https://github.com/maxkeppeler/sheets, 2024.

[27] Google Jetpack, “Emoji2 — android jetpack library,” https://developer.android.com/jetpack/androidx/releases/emoji2, 2024.

[28] A. Engineering, “Lottie for android,” https://airbnb.io/lottie/#/android, 2023.

[29] S. Ramírez, “Fastapi: Modern, fast (high-performance), web framework for building apis,” https://fastapi.tiangolo.com/, 2024.

[30] P. Team, “Plantuml - visualize your diagrams as code,” https://plantuml.com/, 2023.

[31] G. Inc., “Github: Where the world builds software,” https://github.com/, 2025.

[32] P. Inc., “Postman api platform,” https://www.postman.com/, 2024.

[33] R. Plutchik and H. Kellerman, *Theories of emotion.* Academic press, 2013, vol. 1.

[34] L.-C. Yu, L.-H. Lee, S. Hao, J. Wang, Y. He, J. Hu, K. R. Lai, and X. Zhang, “Building Chinese affective resources in valence-arousal dimensions,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Knight, A. Nenkova, and O. Rambow, Eds. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 540–545. [Online]. Available: https://aclanthology.org/N16-1066/

[35] V. Tran, “Positive affect negative affect scale (panas),” in *Encyclopedia of behavioral medicine.* Springer, 2020, pp. 1708–1709.

[36] M. M. Bradley and P. J. Lang, “Measuring emotion: the self-assessment manikin and the semantic differential,” *Journal of behavior therapy and experimental psychiatry*, vol. 25, no. 1, pp. 49–59, 1994.

[37] J. P. Pollak, P. Adams, and G. Gay, “Pam: a photographic affect meter for frequent, in situ measurement of affect,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2011, pp. 725–734.

[38] S. Kang, C. Y. Park, A. Kim, N. Cha, and U. Lee, “Understanding emotion changes in mobile experience sampling,” in *Proceedings of the 2022 CHI conference on human factors in computing systems*, 2022, pp. 1–14.

[39] “Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile — w3.org,” https://www.w3.org/TR/mobile-accessibility-mapping/.

[40] D. Norman, *The Design of Everyday Things.* Basic Books, 2013.

[41] J. Nielsen, “10 usability heuristics for user interface design,” 1994. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

[42] F. Ndikumana, J. Izabayo, J. Kalisa, M. Nemerimana, E. C. Nyabyenda, S. H. Muzungu, I. Komezusenge, M. Uwase, S. Ndagijimana, C. Twizere, and V. Sezibera, “Machine learning-based predictive modelling of mental health in rwandan youth,” *Scientific Reports*, vol. 15, no. 1, p. 16032, May 2025. [Online]. Available: https://doi.org/10.1038/s41598-025-00519-z

[43] R. W. Kenefick and S. N. Cheuvront, “Hydration for recreational sport and physical activity,” *Nutr. Rev.*, vol. 70 Suppl 2, pp. S137–42, Nov. 2012.

[44] “OpenAI Platform — platform.openai.com,” https://platform.openai.com/docs/models.

[45] "Models Overview — Mistral AI — docs.mistral.ai," https://docs.mistral.ai/getting-started/models/models_overview/.

[46] "Unmatched Performance and Efficiency — Llama 4 — llama.com," https://www.llama.com/models/llama-4/.

[47] "Gemma — deepmind.google," https://deepmind.google/models/gemma/.

[48] J. Brooke, "Sus: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, 11 1995.

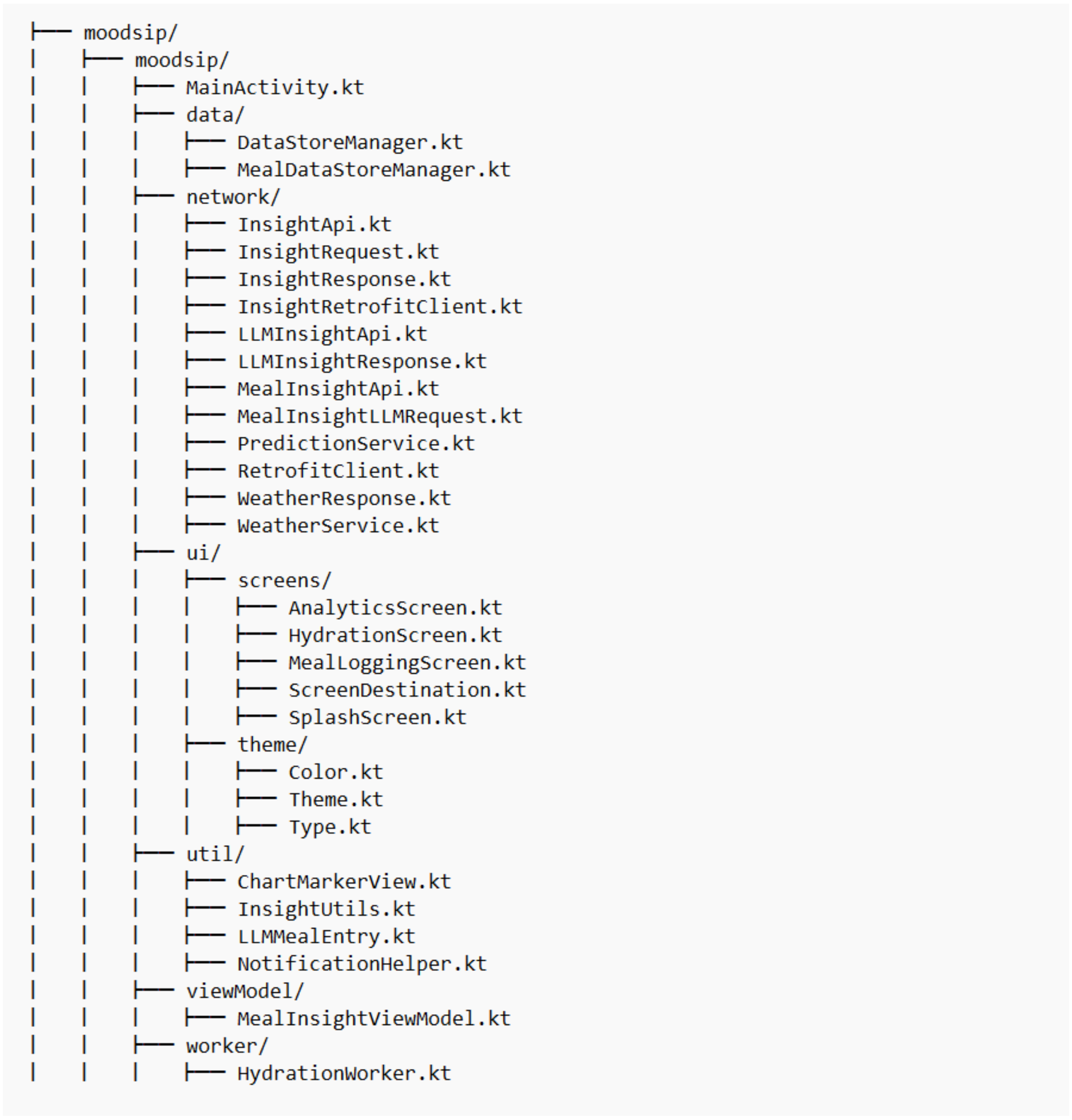
[49] "How To Use The System Usability Scale (SUS) To Evaluate The Usability Of Your Website - Usability Geek — usabilitygeek.com," https://usabilitygeek.com/how-to-use-the-system-usability-scale-sus-to-evaluate-the-usability-of-your-website/.

[50] "App Usability Survey — docs.google.com," https://docs.google.com/forms/d/e/1FAIpQLScLVuGy4-09gFeIyXFhiUAyet7PSj-0YUZvbkG3pshVXjjQyQ/viewform?usp=header.

[51] "Operating sustainably &ndash; Google Data Centers — datacenters.google," https://datacenters.google/operating-sustainably/.

Appendix A

File Structure

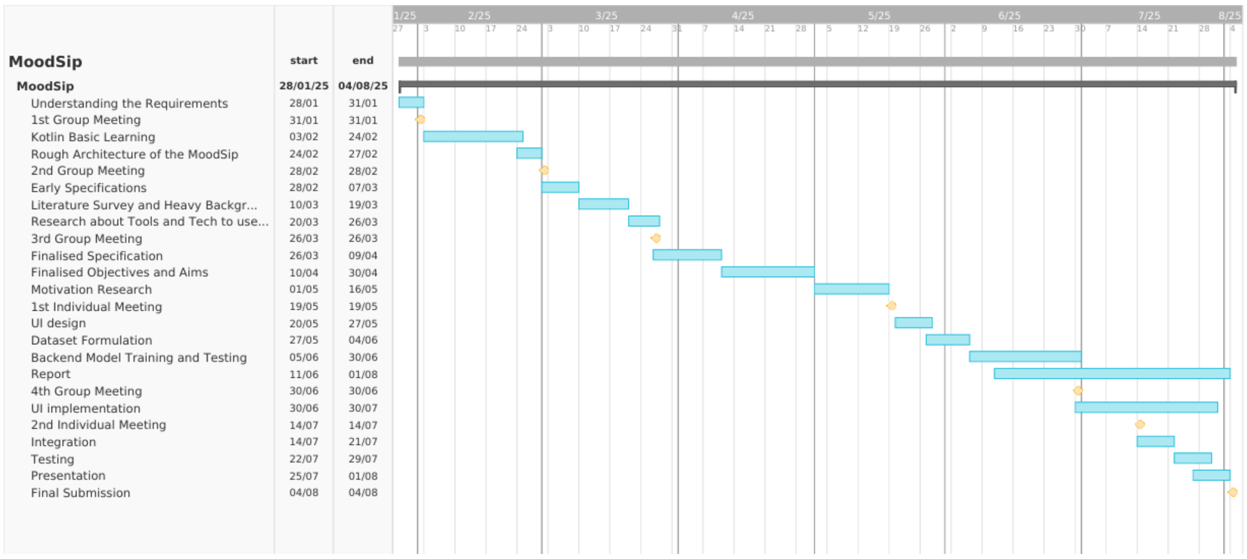


```
├── moodsip/
|   ├── moodsip/
|   |   ├── MainActivity.kt
|   |   ├── data/
|   |   |   ├── DataStoreManager.kt
|   |   |   ├── MealDataStoreManager.kt
|   |   ├── network/
|   |   |   ├── InsightApi.kt
|   |   |   ├── InsightRequest.kt
|   |   |   ├── InsightResponse.kt
|   |   |   ├── InsightRetrofitClient.kt
|   |   |   ├── LLMInsightApi.kt
|   |   |   ├── LLMInsightResponse.kt
|   |   |   ├── MealInsightApi.kt
|   |   |   ├── MealInsightLLMRequest.kt
|   |   |   ├── PredictionService.kt
|   |   |   ├── RetrofitClient.kt
|   |   |   ├── WeatherResponse.kt
|   |   |   ├── WeatherService.kt
|   |   ├── ui/
|   |   |   ├── screens/
|   |   |   |   ├── AnalyticsScreen.kt
|   |   |   |   ├── HydrationScreen.kt
|   |   |   |   ├── MealLoggingScreen.kt
|   |   |   |   ├── ScreenDestination.kt
|   |   |   |   ├── SplashScreen.kt
|   |   |   ├── theme/
|   |   |   |   ├── Color.kt
|   |   |   |   ├── Theme.kt
|   |   |   |   ├── Type.kt
|   |   ├── util/
|   |   |   ├── ChartMarkerView.kt
|   |   |   ├── InsightUtils.kt
|   |   |   ├── LLMMealEntry.kt
|   |   |   ├── NotificationHelper.kt
|   |   ├── viewModel/
|   |   |   ├── MealInsightViewModel.kt
|   |   ├── worker/
|   |   |   ├── HydrationWorker.kt
```

Figure A.1: File Structure

Appendix B

Gantt Chart



MoodSip	start	end
MoodSip	**28/01/25**	**04/08/25**
Understanding the Requirements	28/01	31/01
1st Group Meeting	31/01	31/01
Kotlin Basic Learning	03/02	24/02
Rough Architecture of the MoodSip	24/02	27/02
2nd Group Meeting	28/02	28/02
Early Specifications	28/02	07/03
Literature Survey and Heavy Backgr...	10/03	19/03
Research about Tools and Tech to use...	20/03	26/03
3rd Group Meeting	26/03	26/03
Finalised Specification	26/03	09/04
Finalised Objectives and Aims	10/04	30/04
Motivation Research	01/05	16/05
1st Individual Meeting	19/05	19/05
UI design	20/05	27/05
Dataset Formulation	27/05	04/06
Backend Model Training and Testing	05/06	30/06
Report	11/06	01/08
4th Group Meeting	30/06	30/06
UI implementation	30/06	30/07
2nd Individual Meeting	14/07	14/07
Integration	14/07	21/07
Testing	22/07	29/07
Presentation	25/07	01/08
Final Submission	04/08	04/08

Figure B.1: GANTT Chart