Mood Manager

AI-Based Personalized Mood Management System for Smart Homes

Hyeokjin Ma
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
tema0311@hanyang.ac.kr

Hyunwoo Choi
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
hhyyrr0713@hanyang.ac.kr

Junseong Ahn
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
lljs1113@hanyang.ac.kr

Saeyeon Park
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
saeyeon0317@hanyang.ac.kr

Heejoo Chae
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
heeju0203@hanyang.ac.kr

***Abstract*—Smart homes today offer convenience and automation, yet their fragmented control of lighting, fragrance, and sound leaves the overall spatial experience disjointed and reactive. Existing systems focus on functional operation but lack the ability to anticipate user needs or create holistic, personalized environments. To address this gap, we present Mood Manager, an AI-driven virtual device that integrates ambient elements—light, scent, and audio—into a unified, context-aware system. Using few-shot prompting as its personalization engine, Mood Manager learns user preferences dynamically through real-time interactions, requiring no dedicated model retraining. By combining contextual awareness, feedback logging, and IoT-based prototyping, the system not only responds to explicit commands but also proactively curates atmospheres tailored to user states and environmental conditions. Over time, Mood Manager evolves into an autonomous "mood curator," transfo-rming ordinary spaces into hyper-personalized thera-peutic environments. This approach demonstrates how hyper persona-lization, powered by lightweight AI techniques, can redefine smart home experiences from passive control toward active well-being enhancement.**

TABLE I: Role Assignments

Roles	Name	Task description and etc.
Software Developer (Full-stack)	Hyeokjin Ma	Manages the end-to-end integration of the system across the web client, backend processes, and data synchronization layers. Designs the flow between WearOS inputs, Firebase streams, preprocessing logic, and AI inference endpoints to maintain stability and coherence. Oversees the rollout of core features, identifies bottlenecks, and ensures reliable communication across modules. Coordinates development schedules, enfor-ces shared data standards, and maintains documentation for architecture decisions. Produces integration diagrams, technical reports, and performance evaluations used throughout development.
Software Developer (Front-end)	Hyunwoo Choi	Designs and implements the user-facing interface with emphasis on clarity, respon-siveness, and a consistent visual system. Builds interactive layouts for mood control, device management, and user onboarding while ensuring smooth navigation across the application. Refines UI through testing, accessibility checks, and iterative design improvements. Collaborates with backend and AI developers to reflect real-time biometric and event-driven data in the dashboard. Produces component libraries, design guidelines, and interaction docum-enttation that support long-term scalability.
Software Developer (Back-end)	Heejoo Chae	Leads the development of the server-side architecture responsible for authentication, data processing, and API scalability. Designs the Prisma database schema, implements route logic for device operations and mood updates, and maintains secure communication between the web client, Firebase, and preprocessing workflows. Manages integration of datasets to support personalized recommendations. Coordinates closely with AI developers to ensure consistent data formatting and collaborates with the full-stack developer to maintain API coherence. Produces backend specifications, schema revisions, operational manuals, and monitoring procedures for system reliability.
AI Developer	Saeyeon Park	Works on building the LLM-based inference pipeline using structured few-shot prompting and adaptive reasoning techniques. Designs prompt templates that convert biometric signals, preference data, and contextual indicators into unified mood profiles. Tunes generation parameters to improve relevance, reduce variance, and maintain narrative consistency across moods. Cooperates with the multimodal AI developer to align semantic cues from physiological and audio data, and with backend engineers to validate JSON schema compatibility. Produces experiment summaries, iteration logs, and optimization guidelines that refine the mood-inference engine over time.
AI Developer	Junseong Ahn	Focuses on integrating physiological, acoustic, and environmental signals from WearOS and external APIs into a coherent representation of user state. Develops preprocessing routines for HRV, stress indices, sleep metrics, and audio-event classification to support personalized mood prediction. Works closely with backend developers to stabilize the data ingestion pipeline and ensure the real-time delivery of multimodal features. Collaborates with the LLM-focused AI developer to translate raw signals into interpretable inputs for the recommendation engine. Contributes to improving system robustness, context awareness, and privacy-preserving data handling throughout the full processing pipeline.

I. Introduction

A. *Motivation*

1) The Rise of Hyper-Personalization in AI

Artificial intelligence has transitioned from basic automation to highly adaptive and user-centric systems. Early applications of AI mainly focused on efficiency and scalability, offering generalized solutions designed for broad user groups. However, with the introduction of large language models (LLMs) and generative AI, the paradigm has shifted toward experiences that dynamically adapt to individuals. Users now expect technology not only to execute tasks but also to understand personal preferences, anticipate needs, and respond in intuitive, human-like ways. This shift marks the emergence of hyper-personalization — the ability to tailor services to the unique emotional and behavioral patterns of each user.

Among recent personalization techniques, few-shot prompting has become one of the most effective approaches for achieving real-time adaptation. Unlike traditional retraining or fine-tuning methods, few-shot prompting enables AI systems to learn user tendencies instantly through a small number of examples. This allows the model to refine its behavior continuously based on individual feedback without requiring extensive computation or model updates. As a result, AI evolves into a responsive companion capable of forming emotional resonance with users, delivering experiences that feel deeply personal and contextually aware — extending far beyond conventional automation.

2) The Current Gap in Smart Home Environments

a) Current State of Smart Home Systems

The global smart home market has expanded rapidly, offering consumers a wide range of connected devices that support automation, remote control, and energy management. From smart speakers and lighting systems to intelligent thermostats, these technologies have enhanced convenience and accessibility in daily life. However, most devices still operate independently, relying on user-initiated commands rather than seamless coordination between systems. Research indicates that while the adoption rate of smart devices is high, advanced features such as cross-device automation and contextual adaptation remain underutilized. More than half of users engage only with basic functions such as turning devices on or off, leaving advanced integrations unexplored. This reflects a fundamental gap between technological potential and the actual user experience—where automation exists, but intelligent interaction and emotional resonance do not.

b) Current Trends in the Home Fragrance Market

In parallel, the home fragrance market has shown remarkable growth, driven by consumer interest in wellness, self-care, and emotional comfort within living spaces. Scent has become an essential factor in shaping atmosphere and mood, complementing lighting and sound to create immersive environments. Recent developments include smart diffusers and app-controlled scent devices; however, these solutions largely remain standalone products with limited integration into broader smart home ecosystems. While lighting and audio technologies have achieved sophisticated automation and voice control, fragrance devices are still predominantly manual and lack context-aware functionality. This imbalance reveals an untapped opportunity—bridging the gap between emotional personalization and technological integration through intelligent, automated scent management.

c) Need for Integrated Multi-Sensory Control

To achieve a truly intelligent home environment, it is essential to technologically integrate lighting, sound, and fragrance into a single adaptive framework. Each sensory component influences human emotion in complementary ways: lighting affects focus and relaxation, sound shapes perception and mood, and fragrance modulates emotional comfort. By connecting these elements through AI-driven personalization and IoT synchronization, users can experience holistic, context-aware mood management rather than fragmented device control. This integration establishes the foundation for the proposed Mood Manager System, which coordinates sensory feedback and user interaction to deliver adaptive, emotionally intelligent smart home experiences.

3) Intelligent Mood Management System

Our project introduces the concept of the Mood Manager, a system designed to harmonize lighting, sound, and fragrance into a unified, intelligent experience. Unlike conventional smart home systems that require explicit commands, Mood Manager uses contextual cues and user feedback to infer preferences and deliver proactive adjustments. By employing few-shot prompting as its personalization engine, the system dynamically learns from minimal interaction data, adapting to individual users without the need for extensive retraining. This capability allows Mood Manager to evolve with each interaction, gradually curating an environment that feels increasingly attuned to the user's lifestyle and emotional needs.

The vision is to transform personal spaces into adaptive, therapeutic environments—"personalized therapy rooms" that respond seamlessly to mood and context. For example, on a rainy day, Mood Manager recalls the user's preferred warm lighting, calming fragrance, and soft background music, recreating a comforting atmosphere without requiring manual intervention. Ultimately, the project seeks to demonstrate how hyper-personalized AI, combined with IoT integration, can redefine smart homes as environments that enhance not only convenience but also emotional well-being and quality of life.

B. *Problem Statement (Client's needs)*

a. Fragmented Customization and Limited User Experience

Although smart home devices advertise convenience and personalization, the actual user experience remains fragmented. Lighting, fragrance, and sound—three core factors that shape the ambiance of a personal space—are often managed separately through individual applications or manual controls. According to market surveys, one of the top dissatisfaction factors among smart home users is the lack of seamless interoperability across devices. Users may configure settings individually but fail to fully leverage the advanced functions designed for cross-device scenarios. This leads to declining satisfaction despite increased device ownership. There is a clear need for improved customization and a differentiated user experience that can integrate these elements into a unified and meaningful interaction.

b. Static and Uniform Scenarios

Routine-based automation is a common feature in smart home systems, allowing users to schedule repetitive tasks such as turning on lights or playing background music at fixed times. While useful for simple patterns, such routines fall short in dynamic and unpredictable situations. For example, sudden changes in weather, irregular user moods, or unexpected guest visits are scenarios where rigid routines fail to adapt. Furthermore, current automation functions cannot flexibly manage complex, multi-sensory interactions across lighting, fragrance diffusion, and soundscapes. This limitation reveals the need for systems capable of real-time contextual awareness, leveraging AI and machine learning to adapt to diverse and complex living scenarios.

c. Overreliance on Passive and Manual Management

Despite their "smart" label, most home devices still require significant user intervention for configuration and operation. Users must decide how each device should behave in specific contexts, from adjusting an air conditioner's temperature to selecting fragrance intensity or curating playlists. Even when routines are available, users must manually design logical structures, which can lead to errors or inefficiencies. This overreliance on manual control burdens users with unnecessary complexity, keeping the focus on independent device operation rather than holistic optimization of the entire home environment.

d. Lack of Integrated Information and Orchestration

Smart devices individually generate rich data, such as energy consumption, device usage patterns, or even wellness-related metrics. However, this information is siloed within separate applications, requiring users to manually gather, analyze, and act on disparate data sources. For example, checking air quality data, lighting conditions, and ambient sound levels often involves switching between multiple apps without any overarching integration. The lack of information synthesis prevents users from receiving proactive and contextually relevant recommendations. Without an intelligent system to integrate and orchestrate these insights, the promise of a truly seamless smart home experience remains unfulfilled.

C. Research on Related Software

1. Home Assistant

Home Assistant is an open-source smart-home automation platform that allows users to connect and manage devices from multiple manufacturers in one interface. It supports automation through YAML scripts and plug-ins, enabling complex routines and data visualization dashboards. However, its automation remains rule-based rather than AI-adaptive, and it lacks mood-oriented personalization that learns user emotions or context dynamically.

2. openHAB (Open Home Automation Bus)

openHAB is a Java-based, vendor-neutral smart-home integration framework. It connects heterogeneous IoT devices through a modular "binding" system and provides local dashboards for user-defined automation rules. While it is powerful for interoperability, openHAB depends on manually defined logic and does not include AI-driven personalization or cross-sensory control such as lighting–sound–fragrance orchestration.

3. Philips Hue App + API

Philips Hue provides advanced smart-lighting control software that allows users to adjust brightness, color temperature, and dynamic scenes through a mobile app or API. It integrates with third-party platforms such as Spotify for light-music synchronization, creating partial multi-sensory experiences. Despite its mature lighting automation, Hue's software ecosystem lacks integration with fragrance or contextual AI personalization, limiting full environmental adaptation.

4. Pura Smart Fragrance App

Pura's mobile software manages Wi-Fi-connected fragrance diffusers, letting users schedule scents, monitor cartridge levels, and adjust intensity remotely. The app also connects Alexa and Google Home for voice-based control. Although Pura digitalizes scent management effectively, it operates as an isolated fragrance-control application and cannot coordinate scent with lighting or sound in an AI-driven manner.

5. Amazon Alexa Routines

Alexa Routines software enables users to automate tasks such as turning on lights or playing music based on time, sensor data, or voice triggers. It can execute multiple actions under a predefined "mood" routine (e.g., "Good Morning" or "Relax"). However, its logic is preprogrammed rather than contextual; it does not automatically adapt to environmental changes such as weather or user emotional state.

6. Google Home / Nest App

Google Home application serves as the control hub for devices powered by Google Assistant. It integrates voice commands, visual feedback, and multi-room media control, learning user habits to improve automation suggestions. While it demonstrates conversational intelligence, it mainly supports command execution and lacks true hyper-personalization or multi-sensory mood synthesis.

7. Recombee AI Recommendation Engine

   Recombee is a cloud-based recommendation-engine software that uses collaborative filtering and neural models to predict user preferences in real time. In the context of smart environments, its recommendation logic could inspire how the Mood Manager suggests optimal combinations of lighting, sound, and fragrance. Nevertheless, Recombee is oriented toward digital content recommendation rather than physical-environment orchestration.

8. Dynamic Yield Personalization Platform

   Dynamic Yield is an AI-driven personalization software that analyzes behavioral data to adapt content and optimize user experiences across digital interfaces. Its continuous learning framework and A/B testing tools illustrate how adaptive systems can refine personalization through feedback loops. Although designed for marketing applications, its approach parallels the Mood Manager's objective of refining environmental recommenddations based on user responses and context data.

II. Requirements

A. Sign up

Mood Manager requires essential personal and preference information to create a personalized smart-home profile. The system collects user credentials and baseline data to enable AI-driven customization from the first use.

1) Enter Email Address

   Users must provide a valid email address as their primary account identifier. The email is verified for format validity and checked for duplicates during registration.

2) Create Password

   Users are required to create a password with at least six characters. As users input their password, a real-time indicator displays its strength (Weak / Medium / Strong) to enhance security awareness.

3) Enter Name

   The user's name is collected and automatically set as the default nickname for their account. This name is also used by the AI for conversational interactions and natural responses.

4) Select Gender

   Gender selection is optional and used solely to improve fragrance and sound personalization (e.g., preferred scent tone, music genre). Users may skip this step if they prefer not to disclose the information.

5) Enter Birth Date

   Users enter their full birth date (YYYY-MM-DD format). The system validates that users are at least 12 years old., not the full date of birth, to protect privacy while enabling broad personalization (e.g., age-based recommendation tend-encies).

6) Agree to Data Usage & Privacy Policy

   Users must read and consent to the data usage policy, explaining how their environmental preferences and feedback logs are stored and used to improve AI personalization.

B. Sign In

Mood Manager provides flexible login options to ensure convenience and security for users.

1) Local Login through Mood Manager Account

   Users can log in by entering their registered email and password. If authentication is successful, a secure session token (JWT) is issued to maintain access throughout the session. In case of failure, a pop-up message such as "Invalid ID or password" is displayed with an option to reset the password.

2) Social Login via SNS Integration

   Mood Manager supports quick login through external services such as Google, KakaoTalk (Kakao), or Naver. When users select a social login option, the system authenticates via the provider's OAuth 2.0 API and retrieves essential profile data (name, email, profile image). This data is automatically linked to the user's Mood Manager account.

3) Auto-Login and Session Management

   For convenience, the system offers an "Auto Login" toggle that securely stores encrypted session tokens on the device. When enabled, the user can access the system directly without repeated authentication, provided the session remains valid.

C. Device Registration

After signing in, users can register their smart devices and initialize personalization features in Mood Manager. Device registration serves as the foundation for AI-driven mood control, linking physical appliances with user preferences and environmental data. There are two stages: (1) registering connected IoT devices and (2) configuring personal and environmental settings for mood adaptation.

1) Device Registration

   Users can connect their smart devices to Mood Manager manually. Users can register devices by scanning the QR code printed on the product. Upon granting permission, the system automatically identifies the product model and connects it to the user's account. Alternatively, users may search by product name, serial number, or wireless protocol (Wi-Fi/Bluetooth)

2) Preference Survey

   After device registration, users complete a short preference survey to initialize the AI's personalization engine. This survey helps the system understand basic mood-related tendencies and sensory preferences.

   a) Lighting Preferences

      Users select their favorite light colors from preset options (warmWhite, skyBlue, softPink, etc.) and indicate any disliked colors. Multiple selections are stored as comma-separated strings in UserPreferences.colorLiked and colorDisliked fields. (e.g., warm, cool, natural). The AI later uses this data to suggest appropriate lighting scenes based on time, weather, or activity.

b) Fragrance Preferences

Users choose from basic fragrance families such as floral, woody, citrus, or fresh. They can also indicate sensitivities or allergies (e.g., "avoid strong musk scents").

c) Sound Preferences

Users specify their preferred audio genres or soundscapes (e.g., lo-fi, jazz, ambient nature sounds) and desired volume levels during specific moods (e.g., focus, relaxation).

d) Mood Mapping

Users can connect moods (e.g., relaxed, focused, romantic) with their chosen lighting, fragrance, and sound combinations. These mappings act as initial training data for the AI's few-shot prompting model, allowing it to predict suitable combinations in new contexts.

3) AI Initialization and Verification

After device registration, preference input, and environmental setup are complete, Mood Manager's AI Personalization Engine is initialized. The system verifies all connected devices and settings through a brief diagnostic step, ensuring successful linkage and readiness for automatic mood control.

D. Mood Selection

The Mood Selection feature enables users to experience AI-driven, hyper-personalized environments through the integrated control of lighting, fragrance, and sound. Based on the user's registered devices, preferences, and contextual data, the system automatically recommends or adjusts the optimal "mood scene." Users can also manually modify each element through intuitive interfaces within the application.

1) Mood Recommendation and Scene Generation

a) AI-Powered Context-Aware Analysis

The system continuously monitors environmental variables such as biometric data from WearOS devices (heart rate, HRV, stress levels), audio events from Firebase Firestore (laughter/sigh detection), and user preferences. This contextual information is processed by the AI Personalization Engine, which uses a Few-shot Prompting mechanism to infer the most appropriate combination of light, sound, and fragrance for the current situation.

b) Mood Scene Suggestion

The AI generates one or more mood scene options (e.g., Relaxation, Focus, Romantic Evening) in JSON format, containing recommended values for brightness, fragrance intensity, and music volume. Users can preview and select a preferred mood scene or allow the system to apply it automatically.

c) Manual Adjustment and Override

Users can adjust each element of the mood scene individually (e.g., slightly dimmer lighting, lower fragrance intensity). All adjustments are recorded as user feedback to refine future recommendations.

2) Lighting Control

a) Integration with Smart Lighting Devices

Lighting control is implemented through protocols such as Matter or MQTT, enabling direct communication with smart bulbs or lighting panels (e.g., Philips Hue, Nanoleaf).

b) Brightness and Color Temperature Adjustment

Based on the selected mood, the system modifies brightness and color tone—for example:

- Relax Mode: 30–40% brightness, 2800–3000K warm tone

- Focus Mode: 70–80% brightness, 5000–6000K cool tone

c) Dynamic Lighting Effects

For certain moods, dynamic transitions (fade-in/fade-out or rhythmic lighting) are used to enhance immersion. These effects are generated in real time through the lighting control API.

3) Fragrance Control

a) Smart Diffuser Integration

The system communicates with Wi-Fi or BLE-enabled fragrance diffusers that support programmable intensity and duration.

b) Intensity and Duration Management

The AI determines the appropriate scent intensity (Level 1–5) and dispersal duration based on user context and previous preferences. For example, during a Relax mood, a citrus or woody scent may be diffused at moderate intensity for 30 minutes.

c) Safety and Refill Monitoring

The system includes safety guardrails such as preventing over-diffusion, enforcing cooldown intervals, and monitoring cartridge capacity. If refill levels are low, the user receives an in-app notification.

4) Sound Control

a) Music and Soundscape Selection

Mood Manager connects to streaming APIs (e.g., Spotify, YouTube Music) or local storage to play background sounds aligned with the selected mood. Example pairings include:

- Focus: Lo-fi beats, white noise

- Relax: Nature sounds, jazz

- Romantic: Soft instrumental music

b) Volume and Playback Scheduling

The system controls speaker volume through the IoT layer, maintaining balance with lighting and fragrance intensity. Playback automatically starts and stops according to mood duration or time of day.

5) AI-driven Multi-Sensory Synchronization

All three sensory components—light, sound, and fragrance—are orchestrated simultaneously by the IoT Orchestration Module. This module ensures that transitions occur smoothly and synchronously, creating a cohesive experience rather than separate device reactions. For instance, when the user switches from

Focus to Relax, the lighting slowly dims while soft music and fragrance fade in together. The orchestration logic follows a transaction-based system that guarantees all connected devices receive the command successfully. If one device fails, the system automatically rolls back to the previous stable state to maintain harmony in the environment.

E. Chat-Based Feedback

The Chat-Based Feedback feature allows users to communicate naturally with the system after a mood scene has been applied. Through conversational feedback, users can express satisfaction, make adjustments, or request changes in real time. This interaction not only enhances user engagement but also serves as essential data for the AI Personalization Engine to refine future recommendations.

1) Chat Interface for Feedback Collection

a) Conversational Feedback Window

After a mood scene (lighting, fragrance, and sound) is applied, the system opens a chat interface with the AI assistant. The assistant asks short, context-aware questions such as:

- "Do you like this current mood setup?"
- "Would you prefer brighter lighting or a softer scent?"

b) User Responses

Users can respond freely in natural language (e.g., "Make it a bit warmer" or "The scent is too strong"). The system's natural language processor (NLP) interprets the intent and modifies the settings in real time.

c) Quick Feedback Buttons

For quick interaction, users can also select from predefined feedback buttons such as satisfied, unsatisfied, or adjust. These simple responses are recorded as sentimental data for learning purposes.

2) Real-Time Adjustment and Data Logging

When the user provides feedback, the system immediately modifies corresponding IoT parameters through the orchestration module. For instance, "Make it a little brighter" increases lighting brightness, while "Reduce the scent" decreases diffuser intensity. Each feedback session is stored in the feedback log, including contextual data (time, environment, and device states) and final config-uration results. These logs are vectorized and stored in the embedding database, forming reference examples for Few-shot Prompting.

3) AI Learning and Personalization Update

The system periodically retrieves stored feedback data to enhance personalization. Positive or frequently accepted configurations are treated as "preferred examples," guiding future inference in similar contexts. The AI dynamically updates user profiles based on this data, gradually shaping individual tendencies such as preferred lighting tone or scent strength. When conflicting prefe-rences occur, recency-based weighting ensures that the model adapts to the user's current lifestyle rather than outdated data.

4) Emotion and Sentiment Recognition

The NLP module analyzes the emotional tone of chat inputs to detect user sentiment (positive, neutral, or negative). If stress or fatigue-related language is detected (e.g., "I'm tired," "Too loud"), the system proactively adjusts to a Relax or Comfort mood scene. This sentiment-aware control enhances emotional intelligence and user satisfaction.

III. Development Environment

A. Choice of software development platform

1) Development Platform

a) Web Platform

The web platform serves as the central foundation of the Mood Manager system, offering high accessibility, scalability, and compatibility across multiple operating systems. Through modern web technologies such as Next.js and TypeScript, developers can efficiently build responsive interfaces and maintain consistent user experiences on any device. The web platform supports seamless collaboration among team members using different operating systems including macOS, Windows, and Linux, as it only requires a web browser for development and testing. It also provides easy integration with cloud services such as Firebase Firestore for real-time data synchronization and AWS for hosting and deployment. Continuous updates in web technologies ensure compatibility with emerging standards and APIs, enabling long-term maintainability. For these reasons, the web platform is recognized as a flexible and future-proof environment that supports the core visualization and control components of the Mood Manager system.

b) Wear OS Platform

Wear OS, built on the Android ecosystem, is employed as the data collection platform for biometric and acoustic information in the Mood Manager project. It provides robust SDK support and native integration with key APIs such as Health Connect, AudioRecord, and Porcupine, enabling efficient access to physiological data and on-device voice recognition features. Development in Kotlin enhances code safety, readability, and interoperability with Android libraries while ensuring optimal performance. The platform's Foreground-Service feature allows continuous background operation for uninterrupted data capture, even when the watch display is inactive. Collected data are securely transmitted to Firebase Firestore for real-time synchronization with the cloud and web dashboard. As an Android-based system, Wear OS benefits from a large developer community, wide hardware availability, and strong compatibility with Google services. Due to these capabilities, Wear OS was selected as the optimal platform for reliable, real-time physiological data acquisition and contextual awareness within the Mood Manager ecosystem.

2) Language

a) Kotlin

Figure 1: Kotlin

Kotlin is a modern, statically typed programming language officially supported by Google for Android and Wear OS development. It enhances code safety through null safety features and concise syntax, reducing boilerplate compared to Java. Kotlin provides seamless interoperability with existing Android libraries, enabling efficient use of APIs such as Health Connect and AudioRecord. Its coroutine-based concurrency allows smooth background processing and real-time data handling, making it ideal for continuous biometric and audio data collection in wearable environments.

b) TypeScript

Figure 2: Typescript

TypeScript is a superset of JavaScript developed by Microsoft, introducing a static type system that improves code stability and readability. It catches errors at compile time, reducing runtime issues and enhancing maintainability, especially in large-scale projects. TypeScript's strong type inference and modular structure help developers define clear code intent and foster collaboration across teams. It preserves JavaScript's flexibility while providing the safety of typed languages, making it a robust choice for building reliable and scalable web applications.

c) Python

Figure 3: Python

Python is a high-level, general-purpose programming language widely used for backend development, data analysis, and artificial intelligence applications. Its simple syntax and extensive library support make it suitable for rapid prototyping and deployment of complex systems. Python's ecosystem includes frameworks and tools optimized for web APIs, machine learning, and data processing, allowing seamless integration with cloud environments. Its flexibility and readability facilitate efficient backend logic implementation for data-driven decision-making in the Mood Manager system.

3) Frameworks

a) Next.js

Figure 4: Next.js

Next.js is a React-based web framework that provides server-side rendering and static site generation for fast and scalable web applications. It supports efficient routing and strong TypeScript integration, improving performance and maintainability. The framework simplifies both frontend and backend development through built-in API routes and automatic optimization, making it ideal for real-time interactive systems. It also provides excellent developer experience with hot reloading and simple deployment options, ensuring rapid iteration and stable production performance.

b) FastAPI

Figure 5: FastAPI

FastAPI is a lightweight Python framework designed for building APIs quickly and efficiently. It uses Python type hints for automatic validation and documentation while supporting asynchronous processing for high-speed performance. The framework enables clean, readable, and easily extensible code, allowing developers to implement stable and responsive backend services. It is highly compatible with modern AI libraries and cloud environments, making it suitable for intelligent and data-driven applications.

c) Tailwind CSS

Figure 6: Tailwind CSS

Tailwind CSS is a utility-first framework that allows fast and consistent front-end design. It provides reusable style classes and encourages clean, responsive layouts without custom CSS. The framework ensures visual consistency, reduces repetitive styling tasks, and helps developers rapidly build modern and adaptive user interfaces. Tailwind also integrates seamlessly with component-based frameworks, enabling scalable and maintainable front-end design systems.

d) Android JetPack

Figure 7: Android Jetpack

Android Jetpack is a collection of libraries and components that simplifies Android app development. It provides modules such as Lifecycle, ViewModel and WorkManager that help manage data, background tasks and UI states efficiently. Jetpack offers a standardized architecture, improves app stability, and enhances productivity with built-in tools for Wear OS integration. Its modular structure supports cleaner code organization and long-term maintenance for complex Android applications.

4) Cloud Platform

a) Amazon Web Services(AWS)

Figure 8: Amazon Web Services(AWS)

AWS is a global cloud platform that provides hosting, storage and deployment services for modern applications. It offers high reliability, automatic scaling and strong security for both web and backend systems. AWS supports continuous integration and deployment pipelines, ensuring efficient updates and consistent performance across environments. Its broad service ecosystem allows flexible configuration and cost optimization, making it suitable for both development and large-scale production.

b) Firebase Firestore

Figure 9: Firebase Firestore

Firebase Firestore is a cloud-based NoSQL database that supports real-time synchronization and secure data storage. It allows fast data access and easy integration with mobile and web SDKs. Firestore automatically scales with usage and minimizes server management, enabling developers to focus on application functionality rather than infrastructure. Its real-time listener feature ensures that all connected clients instantly reflect changes, creating a seamless live data environment.

5) Cost Estimation

a) Hardware

The hardware cost consists of laptops used by five team members for development and testing. Each device is estimated at approximately ₩1,500,000, which provides sufficient performance for running Android Studio, Next.js, and cloud-based tools. Since the team requires multi-platform compatibility across macOS and Windows environments, the total hardware expense amounts to ₩7,500,000 in total.

b) Software

The software cost consists of cloud computing and AI model usage. For backend hosting and deployment, an AWS EC2 t3.xlarge instance is used, which provides 4 vCPUs, 16 GiB of memory, EBS-only storage, and network bandwidth up to 5 Gbps. The on-demand usage rate is USD 0.2816 per hour, approximately ₩370 per hour. Assuming a total usage of about 135 hours during the project development period, the total estimated cost for AWS is around ₩50,000. For AI inference, the project uses the OpenAI GPT-5 API, which is priced at USD 1.25 per one million input tokens and USD 10 per one million output tokens. The project is expected to use under one million input tokens and around three million output tokens, leading to an estimated total cost of USD 31.25, equivalent to approximately ₩41,000. Combining both components, the total estimated software cost is ₩91,000, covering cloud infrastructure and AI API usage throughout the development and testing phases.

B. *Software in Use*

1. Android Studio

Figure 10: Android Studio

Android Studio is the official integrated development environment for Android and Wear OS application development. It provides powerful tools for code editing, debugging, and performance profiling, allowing developers to build optimized and reliable mobile applications. The environment supports Kotlin and Java natively and integrates seamlessly with Android Jetpack, Health Connect, and other Google APIs. It also includes an advanced emulator for testing across various devices and OS versions. Android Studio's strong support for Gradle automation and version control improves productivity and ensures stable app deployment.

2. Visual Studio Code

Figure 11: Visual Studio Code

Visual Studio Code is a lightweight and highly extensible code editor developed by Microsoft. It supports multiple programming languages including TypeScript, Python, and Kotlin through a rich extension ecosystem. The built-in IntelliSense, debugging tools, and terminal features make it suitable for both frontend and backend development. It offers real-time collaboration via Live Share, allowing developers to work together effectively across different environments. VS Code's flexibility and integration with GitHub and Docker make it an ideal all-purpose tool for modern software projects.

3. GitHub

Figure 12: GitHub

GitHub is a cloud-based version control and collaboration platform built on Git. It enables developers to manage code efficiently, track revisions, and collaborate across teams through pull requests and branches. Its project management tools, including issues and boards, help maintain workflow organization. GitHub also provides CI/CD pipeline integration and security scanning features for automated testing and code quality assurance. The platform's global community and open-source ecosystem promote collaboration and innovation across diverse development teams.

4. Figma

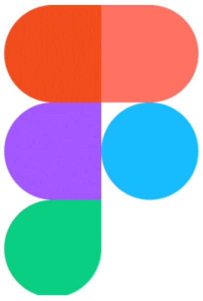


Figure 13: Figma

Figma is a web-based design and prototyping tool used to create user interfaces and interactive prototypes. It supports real-time collaboration, allowing multiple designers and developers to work simultaneously. Its reusable component system and responsive design capabilities streamline the UI/UX workflow. Figma's browser-based nature eliminates installation issues and ensures cross-platform accessibility. The tool's integration with design systems and plugins enhances team efficiency and visual consistency across products.

5. Notion

Figure 14: Notion

Notion is an all-in-one workspace that combines documentation, task management, and project collaboration. It allows teams to organize notes, schedules, and resources in a centralized environment. Customizable templates and database features make it adaptable to different workflows. Notion supports real-time collaboration and version tracking, reducing communication gaps between developers and designers. Its flexibility and minimal interface help maintain clear documentation throughout all stages of project development.

6. PostgreSQL

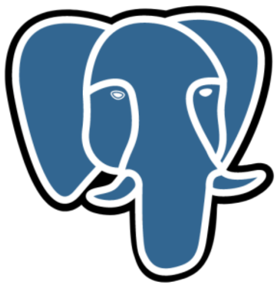


Figure 15: PostgreSQL

PostgreSQL is an open-source relational database management system known for its robustness and compliance with SQL standards. It provides advanced features such as transactions, indexing, and JSON support for hybrid data structures. The database ensures high data integrity and fault tolerance, making it suitable for mission-critical systems. PostgreSQL's scalability and strong community support enable continuous improvements and long-term stability. Its compatibility with cloud services and ORMs makes it a preferred choice for modern data-driven applications.

7. Docker

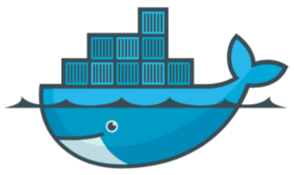


Figure 16: Docker

Docker is a containerization platform that enables applications to run consistently across different environments. It packages code and dependencies into lightweight containers that improve scalability and deployment speed. Docker eliminates environment inconsistencies, simplifying development and testing. Its support for microservices architecture allows modular and efficient system design. The platform's integration with orchestration tools like Kubernetes makes it a standard for modern cloud-native applications.

8. ChatGPT

Figure 17: ChatGPT

ChatGPT is an advanced conversational AI model developed by OpenAI, accessible through the OpenAI API for integration into software systems. It provides natural language understanding and generation capabilities that assist developers in automating documentation, debugging, and code generation tasks. Through the API, ChatGPT can be embedded directly into applications, enabling intelligent interactions such as question answering, summarization, and contextual reasoning. It supports multiple programming languages and frameworks, making it highly adaptable to various development environments. The API's scalability and pay-per-use pricing structure allow flexible usage without the need for dedicated infrastructure. By leveraging the GPT-5 architecture, ChatGPT delivers reliable, context-aware responses that improve development efficiency and enhance user experience in interactive systems.

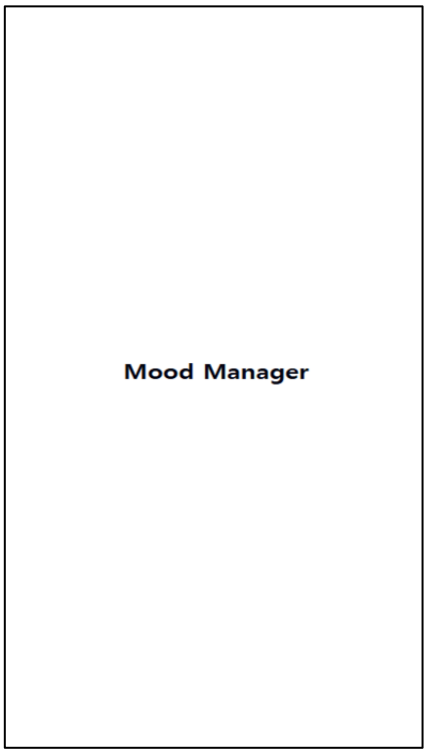
9. Gemini

Figure 18: Gemini

Gemini is Google's advanced AI model designed for reasoning, coding assistance, and data analysis. It provides multimodal capabilities, allowing text, image, and structured data processing within a single framework. The model assists developers with intelligent suggestions, code generation, and system optimization. Gemini's adaptability and integration with Google's ecosystem make it a valuable tool for research and advanced development workflows.

IV. Specification

A) Splashing Page



Mood Manager

Figure 19: Splashing Page

When the application is launched, a dedicated splash screen is displayed for approximately one to two seconds to prevent users from seeing an empty or uninitialized interface during data loading. This screen ensures a smooth and visually coherent startup experience by masking initial delays while core resources, user session data, and essential UI components are being prepared in the background. Once loading is complete, the app automatically transitions to the main interface without requiring any user interaction, creating a seamless and polished first impression.

B) *Sign Up Page*

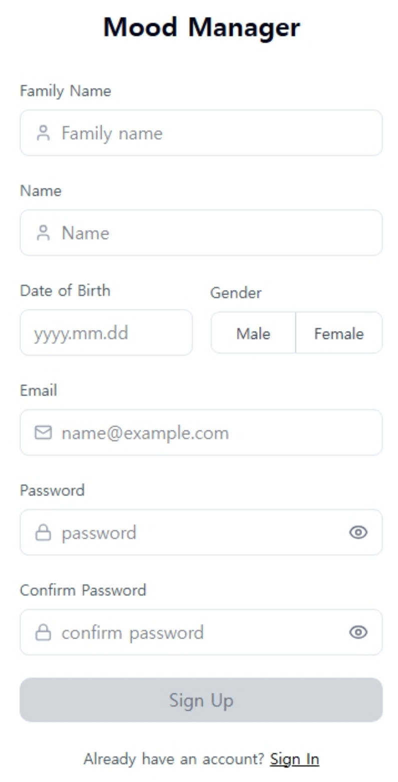


Figure 20: Sign Up Page

1) Email Registration

Users enter a valid email address as their primary account identifier. The system validates the email format in real-time and displays an error message if the format is incorrect. When users submit the registration form, the system checks whether the email is already registered in the database. If the email is already in use, a clear message informs the user and suggests logging in instead. The email field cannot be changed after initial account creation and serves as the permanent login identifier.

2) Password Creation

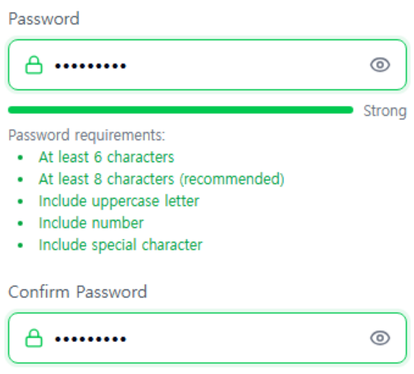


Figure 21: Password Creation

Users create a password with a minimum length of six characters. The password field displays asterisks by default to protect privacy, and a toggle button allows users to reveal or hide the entered text. Below the password field, a confirmation field requires users to re-enter the same password to prevent typos. If the two passwords do not match, the system displays a warning message and prevents submission. A brief hint below the field reminds users of the minimum character requirement.

3) Personal Information Input

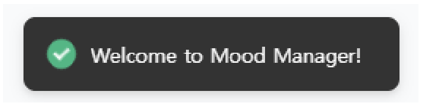
Users provide their family name and given name in separate fields, both of which are required for account creation. The birth date must be entered in YYYY-MM-DD format, and the system validates that users are at least twelve years old before allowing registration. Users select their gender from three options: Male, Female, or Other. An optional phone number field accepts Korean mobile numbers and automatically normalizes the format by removing hyphens and spaces. All personal information can be edited later through the profile management page.

4) SNS Sign Up

Figure 22: SNS Sign Up

Users can register using Google, Kakao, or Naver accounts by clicking the corresponding button on the registration page. After selecting a provider, users are redirected to the provider's consent screen where they approve sharing basic profile information such as email, name, and profile image. Upon returning to the application, the registration form pre-fills the email field with the authenticated email address, which becomes read-only. Users complete the registration by entering their family name, given name, birth date, and gender. No password is required for social login accounts, as authentication is handled by the OAuth provider.

5) Registration Completion



Welcome to Mood Manager!

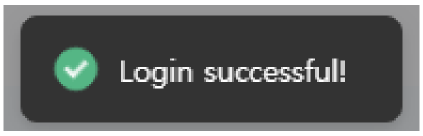
Figure 23: Registration Completion

A brief review step shows the entered information except the password so users can check it once more. Users agree to the terms and privacy policy by ticking checkboxes and tap Create account. A success message appears and the app moves to the login page automatically. If the network fails the page keeps the inputs and shows a Try again button. Clear guidance is shown for any error so users can finish without confusion.

C) Sign In Page

1) Local Sign In

a) Success

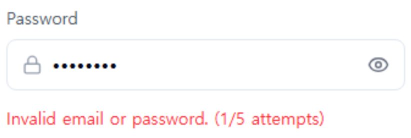


Login successful!

Figure 24: Sign In Success

Users enter their ID and password, which are hidden with star symbols and can be briefly shown by tapping the Show button. After pressing Sign In, if the information matches, the app opens the main page.

b) Wrong Password



Password

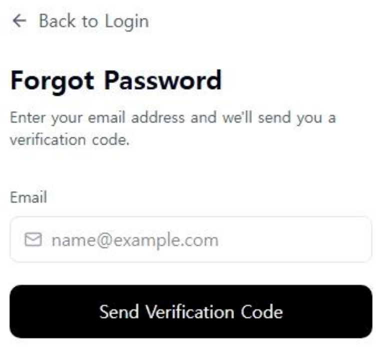
••••••••

Invalid email or password. (1/5 attempts)

Figure 25: Wrong Password

If the entered password is incorrect, the field border turns red, and a clear message appears asking the user to try again. The ID stays on the screen so users only reenter the password. A small hint message appears if Caps Lock is on, and the Forgot Password link guides users directly to reset. The screen limits repeated failed attempts and shows a short wait message before retrying.

c) Password Recovery



Back to Login

Forgot Password

Enter your email address and we'll send you a verification code.

Email

name@example.com

Send Verification Code

Figure 26: Password Recovery

Users who forget their password can reset it through a secure three-step verification process using email-based authentication. The process ensures account security while providing a straightforward recovery experience.

1. Email Verification Request

   Users enter their registered email address on the password recovery page. After submitting, the system sends a six-digit verification code to the provided email address. A confirmation message informs users that the code has been sent and instructs them to check their inbox. If the email is not registered, the system displays a generic success message to avoid revealing account existence. If the email belongs to a social login account, the system displays a specific error message explaining that password reset is not available and directs the user to log in with their provider.

2. Code Verification

   Users enter the six-digit code received via email into the verification field. The code remains valid for ten minutes, and a countdown timer displays the remaining time. If users do not receive the code, they can click a "Resend Code" button after a brief waiting period. When the code is submitted, the system validates it against the stored value and checks the expiration time. If the code is correct and not expired, users proceed to the password reset step. If the code is incorrect, expired, or already used, an error message appears with appropriate instructions.

3. Password Reset

Users create a new password that meets the minimum six-character requirement. A confirmation field requires users to re-enter the password to prevent typos. The password field includes a toggle button to show or hide the entered text. After submitting, the system updates the stored password and displays a success message. Users are then automatically redirected to the login page where they can sign in with their new credentials.

2) SNS Sign In

Users can sign in with social accounts such as Google, Naver, or Kakao by tapping the provider button. After confirming account access on the provider's page, users return to the app where basic information like name and email are automatically filled. If the account is new, the app asks for additional fields such as email address or age before completion. When sign-in succeeds, the app moves to the main page and shows a welcome notification.

D) Main Page

1) Preference Survey

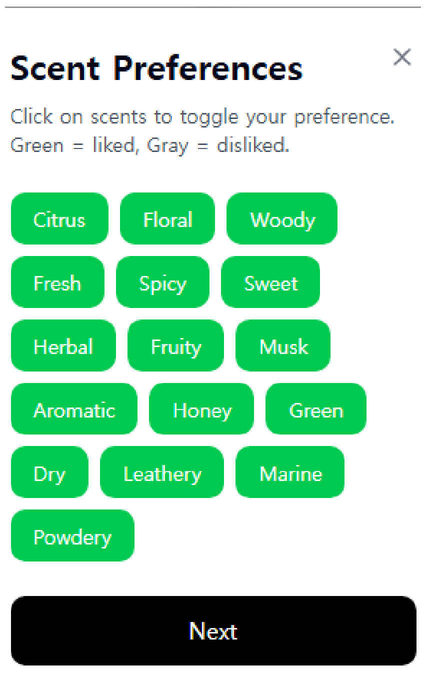


Figure 27: Preference Survey

When users log in for the first time, a full-screen survey overlay appears before the home page content. The survey consists of three steps: scent preferences, lighting preferences, and music preferences. Each step presents multiple-choice questions with visual icons representing each option. Users can select up to three favorite items and any number of disliked items. Navigation buttons at the bottom allow users to move between steps or skip the survey entirely. If users skip, default preferences are applied automatically. After completing or skipping the survey, the overlay closes and does not appear again unless users reset their preferences.

2) Mood Dashboard

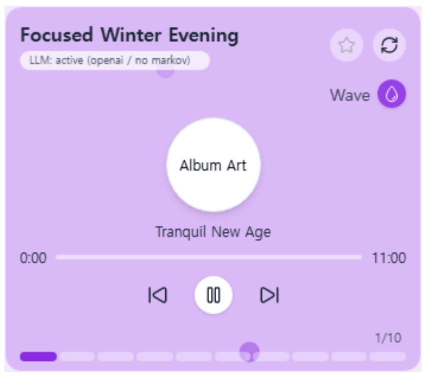


Figure 28: Mood Dashboard

The mood dashboard occupies the upper section of the home page and displays the current mood name, scent type, lighting color, and music genre. Each element is presented with descriptive labels and visual indicators such as color switches for lighting. Four control buttons allow users to change the entire mood, scent only, song only, or color only. When users click any of these buttons, a selection interface appears with available options. After selecting, the dashboard updates to reflect the new mood settings, and all connected devices adjust accordingly.

3) Device Management

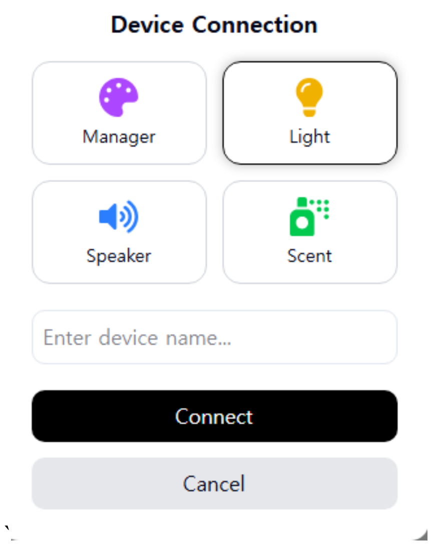


Figure 29: Device Connection

a) Device Type Selection

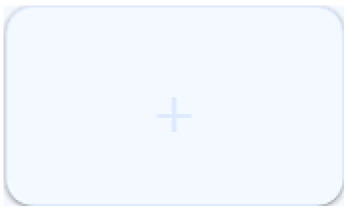


Figure 30: Device Add Button

Clicking the "Add Device" button opens a modal displaying four device type options: Manager, Light, Scent, and Speaker. Each option is represented by a large card with an icon, label, and brief description of its purpose. Users select a type by clicking the corresponding card, which then proceeds to the naming step. The modal can be closed at any time by clicking the background overlay or a close button.

b) Device Name Setup

After selecting a type, the modal displays a text input field where users can enter a custom device name. A default name is pre-filled based on the device type and the number of existing devices of that type. For example, the first light device defaults to "Light #1" and the second to "Light #2". Users can accept the default name or replace it with a custom value. A "Confirm" button submits the registration, while a "Cancel" button returns to the type of selection screen.

c) Device Addition

When users confirm the device name, the modal closes and the new device immediately appear in the device grid. A brief success notification displays the device name to confirm the addition. The newly added device starts in the powered-off state with full battery. Users can immediately click the device card to expand it and adjust settings.

d) Device Deletion

Figure 31: Device Deletion Button

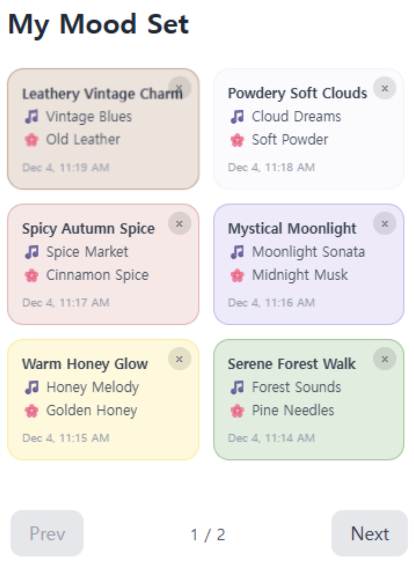
Each expanded device card includes a delete button in the footer section. When clicked, a confirmation dialog appears asking users to verify the deletion. The dialog displays the device name and warns that this action cannot be undone. If users confirm, the device is removed from the grid and all associated data is deleted. If users cancel, the dialog closes and no changes occur.

4) AI-Based Recommended Mood

A prominent "Generate Mood" button triggers the AI-based mood recommendation system. When clicked, a loading indicator appears with a message explaining that the system is analyzing biometric data. The generation process considers recent stress levels, sleep quality, emotion events, weather conditions, and user preferences. After processing, the dashboard updates with the newly generated mood, and a notification displays the mood name. Users can regenerate the mood at any time by clicking the button again.

E) Mood Page

1) Mood Set Display



My Mood Set

Leathery Vintage Charm ×
Vintage Blues
Old Leather
Dec 4, 11:19 AM

Powdery Soft Clouds ×
Cloud Dreams
Soft Powder
Dec 4, 11:18 AM

Spicy Autumn Spice ×
Spice Market
Cinnamon Spice
Dec 4, 11:17 AM

Mystical Moonlight ×
Moonlight Sonata
Midnight Musk
Dec 4, 11:16 AM

Warm Honey Glow ×
Honey Melody
Golden Honey
Dec 4, 11:15 AM

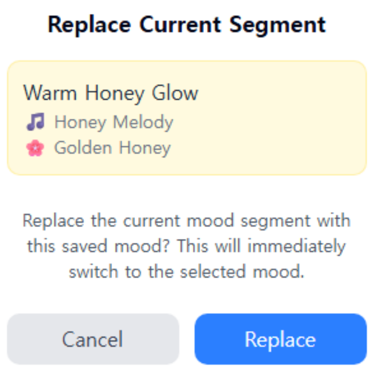
Serene Forest Walk ×
Forest Sounds
Pine Needles
Dec 4, 11:14 AM

Prev 1 / 2 Next

Figure 32: Mood Set Display

The mood page displays all saved moods in a grid layout across multiple pages. Each saved mood is represented as a colored card showing its name, music track, scent type, and timestamp. Users can navigate through their collection using Prev and Next buttons at the bottom of the page. Each mood card includes a small (×) delete icon in the top-right corner, allowing quick removal of saved moods.

2) Current Segment Replacement


Replace Current Segment

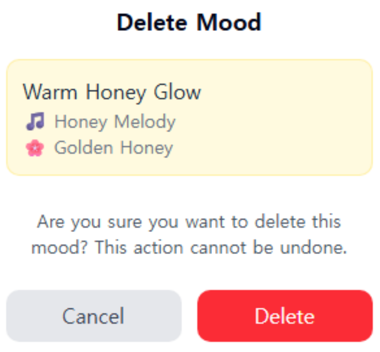
Warm Honey Glow
Honey Melody
Golden Honey

Replace the current mood segment with this saved mood? This will immediately switch to the selected mood.

Figure 33: Current Segment Replacement

When the user selects a saved mood card while viewing the Mood Dashboard, a Replace Current Segment modal appears. The modal shows a preview of the selected mood—its name, music, and scent—and asks whether the user wants to replace the current segment of the active mood stream. Choosing Replace immediately switches the current segment to the selected mood, updating music and scent on all connected devices.

3) Saved Mood Deletion


Delete Mood

Warm Honey Glow
Honey Melody
Golden Honey

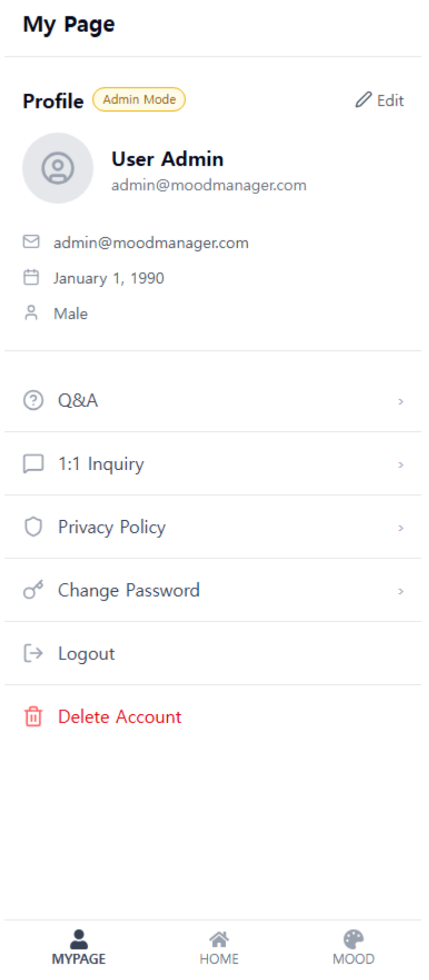
Are you sure you want to delete this mood? This action cannot be undone.

Figure 34: Saved Mood Deletion

When the user clicks the delete icon on a saved mood card, a Delete Mood confirmation modal opens. The modal displays the mood's name, music, and scent and asks for final confirmation. Pressing Delete permanently removes the mood from the user's saved list. A success message is shown after deletion, and the Mood Set grid updates accordingly.

F) My Page

1) Profile Display


My Page

Profile Admin Mode Edit

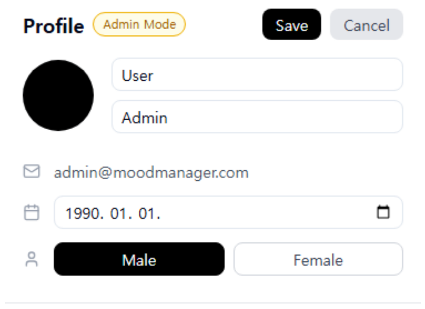
User Admin
admin@moodmanager.com

admin@moodmanager.com
January 1, 1990
Male

Figure 35: Profile Display

The profile page displays the user's current information including email, family name, given name, birth date, gender, phone number, and profile image. The email field is read-only and cannot be changed. If the account uses social login, a badge displays the provider's name such as "Connected with Google". An "Edit Profile" button at the top of the page enables editing mode for modifiable fields.

2) Profile Editing



User
Admin
admin@moodmanager.com
1990. 01. 01.
Male
Female

Figure 36: Profile Editing

Clicking the "Edit Profile" button transforms display fields into editable inputs. Users can upload a new profile image by clicking the current image and selecting a file. Text fields for family name, given name, phone number become active for editing. The birth date field opens a date picker when clicked. The gender field displays a dropdown menu with three options. A "Save Changes" button submits the modifications, while a "Cancel" button discards changes and returns to display mode. After saving, a success message appears and the page refreshes to show the updated information.

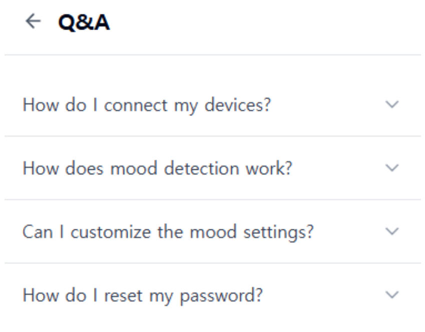
3) Account Deletion

A "Delete Account" button appears in the danger zone section of the account settings. When clicked, a confirmation dialog warns that this action is permanent and will delete all associated data including devices, presets, and preferences. Users must confirm by clicking "Delete" in the dialog or cancel to abort. If users confirm deletion, they are immediately logged out and redirected to the login page with a notification that the account has been permanently deleted.

4) Logout

The Logout button signs the user out of the current account immediately. When the button is clicked, app returns to the sign-in page and clears saved session data for security.

5) Q&A

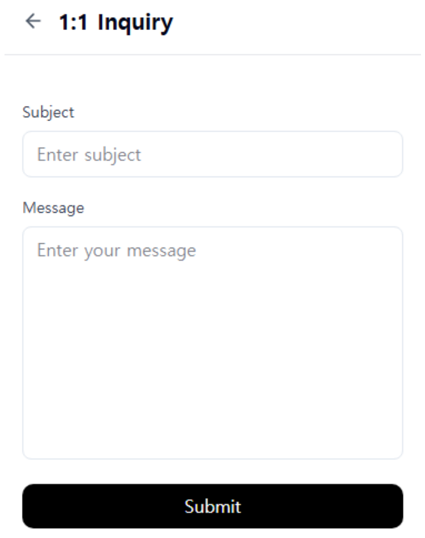


How do I connect my devices?
How does mood detection work?
Can I customize the mood settings?
How do I reset my password?

Figure 37: Q&A

The Q&A page displays a list of frequently asked questions organized in an accordion format. Each question appears as a clickable header that expands to reveal the answer. Topics include how the mood recommendation system works, what biometric data is collected, how to customize presets, and how to add devices. Users can expand multiple questions simultaneously to compare answers. The page is accessible through a link in the Mypage menu.

6) 1:1 Inquiry



Subject
Enter subject
Message
Enter your message

Figure 38: 1:1 Inquiry

The inquiry page provides a form where users can submit support questions or feedback. Users enter a subject line and a detailed message in a multi-line text area. Both fields are required before submission. After submitting, a success message confirms that the inquiry has been received and explains that a response will be provided via email. The form clears after successful submission, allowing users to submit another inquiry if needed.

7) Privacy Policy

The privacy policy page displays comprehensive information about data collection, storage, usage, and user rights. The content is organized into sections with clear headings covering topics such as biometric data collection, third-party service integration, data retention policies, and account deletion procedures. Users can scroll through the entire policy without any interactive elements. The page is accessible through a link in the Mypage menu.

V. Architecture Design & Implementation

A. *Overall Architecture*

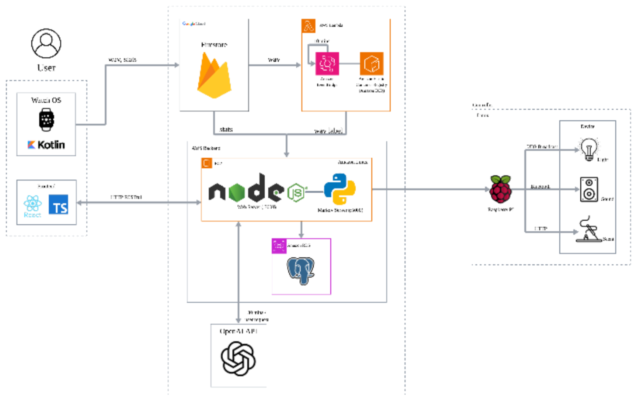


Figure 39: Overall Architecture

The Mood Manager architecture is composed of six primary modules: the WearOS Client, the Cloud Firestore Sync Layer, the ML Event Classification Server, the Web Application, the Backend Service Layer, and the AI Inference Module. Together, these components form an end-to-end pipeline that collects multimodal signals from the user, processes them through cloud-connected services, and generates personalized mood outputs through AI reasoning. Each module plays an essential role in ensuring that biometric, acoustic, and contextual information flows smoothly across the system.

The first module is the WearOS Client, implemented as a Kotlin-based native application that operates continuously through a foreground service. It gathers biometric signals such as heart rate, HRV, stress indicators, and movement counts, while also capturing short audio segments for emotional context. This module serves as the system's data origin, formatting and transmitting periodic measurements to Cloud Firestore with strict timing to maintain consistency for downstream processing.

The second module is the Cloud Firestore Sync Layer, which acts as a real-time communication bridge between the wearable device, the ML service, and the web application. Firestore stores biometric samples and audio-event data under structured user collections, enabling event-driven updates and a stable time-series dataset. This layer removes the need for direct device–server links and supports the continuous flow of information throughout the system.

The third module, the ML Event Classification Server, is deployed on AWS and processes audio events sent from Firestore. It retrieves Base64-encoded WAV inputs, applies inference logic to classify laughter, sigh, or noise, and posts validated results back to the Web Application. By running on a containerized environment, the ML server maintains scalable event processing while ensuring that multimodal emotional signals are available for integrated mood inference.

At the center of the workflow is the Web Application, built with Next.js, React, and TailwindCSS. It collects biometric samples from Firestore, retrieves classification results from the ML server, merges them with user preferences and weather information, and orchestrates the logic required to determine mood attributes. This module also delivers the primary interface for mood visualization, device control, and user management, acting as the core processing and interaction layer of the system.

Supporting these operations is the Backend Service Layer, implemented using Node.js and Prisma. It manages database access, validates operations, and provides structured endpoints for authentication, preference management, device operations, and mood updates. This module ensures consistent interaction between the Web Application and the persistent storage, while also enabling clean integration points for advanced analytics and future system extensions.

The final component is the AI Inference Module, which uses OpenAI's few-shot prompting to convert processed biometric and contextual inputs into structured mood profiles. It generates recommendations for lighting, scent, and sound, along with a descriptive mood name that reflects the inferred emotional state. This module encapsulates the system's adaptive reasoning and provides the core intelligence behind the personalized mood experience.

B. *Directory Organization*

TABLE III: Directory Organization – BACKEND

Directory	File Name
/src/app/api/auth/[...nextauth]	route.ts
/src/app/api/auth/register	route.ts
/src/app/api/auth/profile	route.ts
/src/app/api/auth/account	route.ts
/src/app/api/auth/survey-status	route.ts
/src/app/api/auth/forgot-password	route.ts
/src/app/api/auth/verify-reset-code	route.ts
/src/app/api/auth/reset-password	route.ts
/src/app/api/ai/background-params	route.ts
/src/app/api/ai/chat	route.ts
/src/app/api/devices	route.ts
/src/app/api/devices/[deviceId]	route.ts
/src/app/api/devices/[deviceId]/power	route.ts

/src/app/api/devices/[deviceId]/name	route.ts
/src/app/api/devices/[deviceId]/scent-interval	route.ts
/src/app/api/devices/[deviceId]/scent-level	route.ts
/src/app/api/moods	route.ts
/src/app/api/moods/current	route.ts
/src/app/api/moods/current/scent	route.ts
/src/app/api/moods/current/song	route.ts
/src/app/api/moods/current/color	route.ts
/src/app/api/moods/current/refresh	route.ts
/src/app/api/moods/current/generate	route.ts
/src/app/api/moods/preference	route.ts
/src/app/api/moods/saved	route.ts
/src/app/api/moods/saved/[savedMoodId]	route.ts
/src/app/api/inquiry	route.ts
/src/app/api/preferences	route.ts
/src/app/api/preprocessing	route.ts
/src/app/api/sleep/today	route.ts
/src/app/api/sleep/generate	route.ts
/src/app/api/ml/emotion	route.ts
/src/app/api/markov/daily-preprocessed	route.ts
/backend/listener	periodicListener.ts
/backend/jobs	preprocessPeriodic.ts, processEmotion.ts, calcTodaySleepScore.ts, fetchTodayPeriodicRaw.ts, fetchTodaySleepRaw.ts, generateDummySleepData.ts, sleepSessionDetector.ts, sleepWakeClassifier.ts
/backend/cache	periodicCache.ts
/prisma	schema.prisma, seed.ts, update-seed-data.ts

TABLE IV: Directory Organization – FRONTEND

Directory	File Name
/src/app	layout.tsx, page.tsx
/src/app/(auth)	login/page.tsx, register/page.tsx, forgot-password/page.tsx
/src/app/(auth)/register/components	RegisterForm.tsx, EmailSection.tsx, PasswordSection.tsx, ConfirmPasswordSection.tsx, PersonalInfoSection.tsx, BirthDateGenderSection.tsx
/src/app/(auth)/register/hooks	useRegisterForm.ts
/src/app/(auth)/forgot-password/components	ForgotPasswordSteps.tsx
/src/app/(auth)/forgot-password/hooks	useForgotPassword.ts
/src/app/(main)/home	page.tsx
/src/app/(main)/home/components	HomeContent.tsx
/src/app/(main)/home/components/MoodDashboard	MoodDashboard.tsx
/src/app/(main)/home/components/MoodDashboard/components	MoodHeader.tsx, AlbumSection.tsx, ScentControl.tsx, MusicControls.tsx, MoodDuration.tsx, HeartAnimation.tsx
/src/app/(main)/home/components/MoodDashboard/hooks	useMoodDashboard.ts, useMoodColors.ts, useHeartAnimation.ts, useSegmentSelector.ts
/src/app/(main)/home/components/MoodDashboard/utils	moodStreamConverter.ts, moodUtils.ts
/src/app/(main)/home/components/Device	DeviceGrid.tsx, DeviceCardSmall.tsx, DeviceCardExpanded.tsx, AddDeviceCard.tsx, DeviceAddModal.tsx, DeviceTypeSelectModal.tsx, DeviceNameInputModal.tsx, DeviceDeleteModal.tsx
/src/app/(main)/home/components/Device/components	DeviceControls.tsx, DeviceNameEditor.tsx
/src/app/(main)/home/components/Device/hooks	useDeviceCard.ts, useDeviceHandlers.ts
/src/app/(main)/home/components/Device/utils	deviceUtils.tsx
/src/app/(main)/home/components/SurveyOverlay	SurveyOverlay.tsx
/src/app/(main)/mood	page.tsx
/src/app/(main)/mypage	page.tsx
/src/app/(main)/mypage/components	MenuSection.tsx, ProfileSection.tsx, DeleteAccountModal.tsx
/src/app/(main)/mypage/hooks	useProfile.ts
/src/app/(main)/mypage/inquiry	page.tsx
/src/app/(main)/mypage/privacy	page.tsx
/src/app/(main)/mypage/qna	page.tsx
/src/components	ErrorBoundary.tsx
/src/components/navigation	TopNav.tsx, BottomNav.tsx
/src/components/ui	Skeleton.tsx, ScentBackground.tsx

TABLE V: Directory Organization – CORE MODULES

Directory	File Name
/src/lib/auth	password.ts, session.ts
/src/lib/firebase	client.ts, admin.ts, firebase.ts
/src/lib/aws	aws.ts
/src/lib/cache	llmCache.ts memoryCache.ts
/src/lib/constants	mood.ts
/src/lib/openai	index.ts, formatPayload.ts, openai.ts
/src/lib/llm	generatePrompt.ts, optimizePrompt.ts, prepareLLMInput.ts, validateResponse.ts
/src/lib/preprocessing	preprocess.ts, prepareOpenAIInput.ts, index.ts
/src/lib/sleep	detectSleepStage.ts, calculateSleepScore.ts, calculateDailySleepScore.ts, utils.ts, index.ts
/src/lib/stress	calculateStressIndex.ts, utils.ts, index.ts
/src/lib/weather	index.ts, fetchWeather.ts, mapGrid.ts
/src/lib/preferences	getPreferences.ts, updatePreferences.ts, mapPreferencesForAI.ts, preferenceUtils.ts
/src/lib/moodSignals	fetchDailySignals.ts
/src/lib/moodStream	scheduler.ts
/src/lib/types	periodic.ts, sleep.ts
/src/lib/utils	validation.ts, time.ts utils.ts
/src/lib/prisma	prisma.ts
/src/hooks	useMood.ts, useSurvey.ts, useMoodStream.ts, useMoodStreamManager.ts, useBackgroundParams.ts, useDevices.ts, useDeviceSync.ts
/src/types	mood.ts, device.ts, preset.ts, emotion.ts

TABLE VI: Directory Organization – ML

Directory	File Name
mood-manager-ML/mood-manager-ML/apps/ml_classification	predict.py, sigh_laugh_neg_cls.py, convert_onnx.py, dataset_download.py, ml_settings.py, fix_config.py, requirements_docker.txt, Dockerfile, .dockerignore
mood-manager-ML/mood-manager-ML/apps/ml_classification/onnx_model	model.onnx, model_quantized.onnx, config.json, ort_config.json, preprocessor_config.json
mood-manager-ML/mood-manager-ML/apps/ml_classification/saved_model	model.safetensors, config.json, preprocessor_config.json
markov	build_yesterday_many.py, realtime_inference_many.py

TABLE VII: Directory Organization – WearOS

Directory	File Name
Watch	build.gradle.kts, gradle.properties, settings.gradle.kts
Watch/app	build.gradle.kts, lint.xml
Watch/app/src/main	AndroidManifest.xml
Watch/app/src/main/java/com/moodmanager/watch/presentation	MainActivity.kt, AudioEventService.kt, PeriodicDataService.kt, FirebaseViewModel.kt
Watch/app/src/main/java/com/moodmanager/watch/presentation/theme	Theme.kt
Watch/app/src/main/res	drawable/, mipmap-*, values/, values-round/

C. Module 1: Backend

1) Purpose

The Backend Module of Mood Manager provides the core server-side logic responsible for authentication, device management, mood retrieval, data preprocessing, and system-wide orchestration. Acting as the integration point between the Next.js frontend, Firestore data streams, and the AI inference module, it centralizes application logic within structured API routes to ensure consistency, security, and reliable communication throughout the system. In addition to handling requests, the backend manages time-series aggregation, biometric preprocessing, and synchronization of user state for mood inference. Listener, job, and cache components support continuous processing workflows—from WearOS data ingestion to database persistence—forming the operational backbone that enables system stability, scalability, and coherent data flow.

2) Functionality

The Backend Module implements comprehensive REST-style API endpoints for authentication, password recovery, survey status management, device control, and mood customization. These APIs support power toggling, scent interval adjustments, mood refreshing, and mood attribute updates, while additional routes process biometric inputs, compute sleep metrics, and accept emotion classification results from external ML services. Listener modules and scheduled jobs handle periodic physiological data uploaded from WearOS, performing sleep detection, stress calculation, normalization, and daily metric generation. Cache utilities reduce Firestore load by storing intermediate biometric states, and Prisma serves as the database interface for structured access to user profiles, device configurations, and mood history records.

3) Location of Source Code

https://github.com/Ma-Hyekjin/mood-manager/Web

4) Class Components

- auth/[...nextauth]/route.ts : Handles NextAuth session initialization, OAuth login, and session callbacks.
- auth/register/route.ts : Processes new user registration and validation.
- auth/profile/route.ts : Returns and updates authenticated user profile data.
- auth/account/route.ts : Manages account deletion and sensitive account settings.
- auth/survey-status/route.ts : Checks whether the user has completed the preference survey.
- auth/forgot-password/route.ts : Sends reset codes to user email during password recovery.
- auth/verify-reset-code/route.ts : Validates the reset code submitted by the user.
- auth/reset-password/route.ts : Resets the user's password upon successful verification.
- ai/background-params/route.ts : Generates and manages dynamic background visual parameters based on mood state.
- ai/chat/route.ts : Handles conversational AI interactions for mood feedback and adjustments.
- devices/route.ts : Lists all registered devices for the user and creates new devices.
- devices/[deviceId]/route.ts : Retrieves or deletes a specific device.
- devices/[deviceId]/power/route.ts : Toggles device power state.
- devices/[deviceId]/name/route.ts : Updates device name.
- devices/[deviceId]/scent-interval/route.ts : Updates fragrance output interval.
- devices/[deviceId]/scent-level/route.ts : Adjusts fragrance intensity level.
- moods/route.ts : Returns list of available mood presets.
- moods/current/route.ts : Returns the current inferred mood profile.
- moods/current/scent/route.ts : Updates the scent component of the current mood.
- moods/current/song/route.ts : Updates music selection within the mood.
- moods/current/color/route.ts : Updates lighting color attributes.
- moods/current/refresh/route.ts : Forces regeneration of a new mood recommendation.
- moods/current/generate/route.ts : Generates new AI-based mood stream with multiple segments.
- moods/preference/route.ts : Manages user mood preferences and weights.
- moods/saved/route.ts : Lists and creates saved mood configurations.
- moods/saved/[savedMoodId]/route.ts : Retrieves, updates, or deletes a specific saved mood.
- inquiry/route.ts : Submits user inquiries for customer support.
- preferences/route.ts : Gets or updates user preference data.
- preprocessing/route.ts : Preprocesses periodic biometric data for mood inference.
- sleep/today/route.ts : Retrieves today's computed sleep metrics.
- sleep/generate/route.ts : Generates synthetic sleep data for testing.
- ml/emotion/route.ts : Receives emotion classification results from the ML microservice.
- markov/daily-preprocessed/route.ts : Provides daily preprocessed data for Markov-based mood prediction models.
- periodicListener.ts : Listens for new periodic biometric entries and triggers preprocessing.
- preprocessPeriodic.ts : Converts raw biometric data into normalized metrics.
- processEmotion.ts : Integrates emotion data into the mood pipeline.
- calcTodaySleepScore.ts : Computes daily sleep score using HRV and movement metrics.
- fetchTodayPeriodicRaw.ts : Fetches today's raw periodic biometric dataset.
- fetchTodaySleepRaw.ts : Fetches raw sleep sessions for processing.
- generateDummySleepData.ts : Creates mock sleep datasets for debugging.
- sleepSessionDetector.ts : Detects sleep intervals based on movement and HR patterns.
- sleepWakeClassifier.ts : Classifies segments into sleep or wake states.

- periodicCache.ts : Provides in-memory caching for recent biometric computations.
- schema.prisma : Defines SQL schema for users, devices, preferences, and mood logs.
- seed.ts : Populates the initial database with baseline test data.
- update-seed-data.ts : Updates or extends seeded database entries.

D. Module 2: Frontend

1) Purpose

The Frontend Module of Mood Manager provides the user-facing interface through which individuals can authenticate, view their personalized mood state, manage smart-home devices, and update preferences in a clear and responsive manner. Developed using Next.js with a focus on usability and consistent design, the frontend transforms complex biometric and AI-generated insights into intuitive visual elements. It communicates with the backend APIs, renders real-time mood updates, manages user sessions, and drives various guided flows such as onboarding, survey collection, mood presentation, and device control. As the presentation layer of the system, the frontend ensures that every interaction—from mood adjustments to profile edits—operates smoothly and remains accessible across devices.

2) Functionality

The Frontend Module delivers a complete set of interface-level features, including registration, login, password recovery, and survey collection for preference initialization. It displays real-time mood information such as scent, music, lighting, and duration, and provides device management tools for adding, renaming, deleting, and configuring smart-home devices. It also supports user profile management, account deletion, navigation layout rendering, and visual effects like animated heartbeats and dynamic mood backgrounds. Custom hooks abstract the management of mood states, surveys, animations, and device operations, while shared UI components ensure uniform styling and interactions throughout the system. The frontend acts as the hub for user interaction, coordinating data flow between backend logic, preprocessing workflows, and AI inference results.

3) Location of Source Code

https://github.com/Ma-Hyekjin/mood-manager/Web

4) Class Components

- layout.tsx : Defines the global layout wrapping all pages.
- page.tsx : Application entry splash page.
- (auth)/login/page.tsx : Login interface for user authentication.
- (auth)/register/page.tsx : User registration page.
- (auth)/forgot-password/page.tsx : Password recovery request page.
- register/components/RegisterForm.tsx : Controller for the registration form flow.
- register/components/EmailSection.tsx : Email field and validation UI.
- register/components/PasswordSection.tsx : Password creation interface.
- register/components/ConfirmPasswordSection.tsx : Password confirmation form.
- register/components/PersonalInfoSection.tsx : Name and personal information fields.
- register/components/BirthDateGenderSection.tsx : Birth date selector and gender input.
- register/hooks/useRegisterForm.ts : Registration form logic and validation.
- forgot-password/components/ForgotPasswordSteps.tsx : Multi-step UI for password reset.
- forgot-password/hooks/useForgotPassword.ts : Hook controlling password recovery flow.
- (main)/home/page.tsx : Main dashboard displaying inferred mood and devices.
- home/components/HomeContent.tsx : Primary container for dashboard contents.
- MoodDashboard/MoodDashboard.tsx : Wrapper for mood presentation UI.
- MoodDashboard/components/MoodHeader.tsx : Displays current mood name and short description.
- MoodDashboard/components/AlbumSection.tsx : Music artwork and audio-related visuals.
- MoodDashboard/components/ScentControl.tsx : Interface for adjusting scent intensity.
- MoodDashboard/components/MusicControls.tsx : Controls for play, next song, and selection.
- MoodDashboard/components/MoodDuration.tsx : Remaining duration for the current mood.
- MoodDashboard/components/HeartAnimation.tsx : Animated heart visual synced with mood.
- MoodDashboard/hooks/useMoodDashboard.ts : Fetches and manages the overall mood state.
- MoodDashboard/hooks/useMoodColors.ts : Computes theme colors based on mood attributes.
- MoodDashboard/hooks/useHeartAnimation.ts : Animation controller for heartbeat visuals.
- MoodDashboard/hooks/useSegmentSelector.ts : Handles selection logic for segmented UI.
- MoodDashboard/utils/moodStreamConverter.ts : Converts mood stream data formats.

- MoodDashboard/utils/moodUtils.ts : Utility functions for mood calculations.
- Device/DeviceGrid.tsx : Grid view displaying all devices.
- Device/DeviceCardSmall.tsx : Compact visualization for each device.
- Device/DeviceCardExpanded.tsx : Expanded card with full controls.
- Device/AddDeviceCard.tsx : Entry component for adding new devices.
- Device/DeviceAddModal.tsx : Modal for creating a new device.
- Device/DeviceTypeSelectModal.tsx : Modal for choosing device type.
- Device/DeviceNameInputModal.tsx : Modal for device naming.
- Device/DeviceDeleteModal.tsx : Confirmation modal for device removal.
- Device/components/DeviceControls.tsx : Power, scent interval, and control buttons.
- Device/components/DeviceNameEditor.tsx : Inline device rename interface.
- Device/hooks/useDeviceCard.ts : Logic for device card interactions.
- Device/hooks/useDeviceHandlers.ts : Encapsulates API operations for devices.
- Device/utils/deviceUtils.tsx : Utility functions used across device components.
- SurveyOverlay/SurveyOverlay.tsx : Popup survey for collecting user preferences.
- (main)/mood/page.tsx : Mood management and saved mood list page.
- mypage/page.tsx : User profile main page.
- mypage/components/MenuSection.tsx : Navigation menu inside MyPage.
- mypage/components/ProfileSection.tsx : User profile information and editing UI.
- mypage/components/DeleteAccountModal.tsx : Account deletion confirmation modal.
- mypage/hooks/useProfile.ts : Manages user profile data and updates.
- mypage/inquiry/page.tsx : 1:1 inquiry submission page.
- mypage/privacy/page.tsx : Privacy policy and account management page.
- mypage/qna/page.tsx : Frequently asked questions page.
- ErrorBoundary.tsx : Catches and displays global errors.
- navigation/TopNav.tsx : Top navigation bar for the application.
- navigation/BottomNav.tsx : Bottom navigation bar with main menu.
- ui/Skeleton.tsx : Loading placeholder component.
- ui/ScentBackground.tsx : Dynamic visual background reflecting scent category.

E. Module 3: Core Modules

1) Purpose

The Core Modules of Mood Manager provide the foundational logic shared across all parts of the system, including authentication utilities, Firebase access layers, OpenAI integration, biometric preprocessing, sleep and stress analytics, weather retrieval, preference mapping, and global utility functions. These modules abstract low-level operations into reusable building blocks, enabling the frontend and backend to maintain clean, modular structures. By centralizing cross-cutting functionality such as data formatting, AI prompt preparation, physiological scoring logic, and input validation, the Core Modules ensure consistency, reduce duplication, and provide a stable computational base for the entire system's operations. This layer serves as the system's internal backbone, supporting computation-heavy tasks and guaranteeing coherent behavior across different application workflows.

2) Functionality

The Core Modules offer a wide spectrum of reusable functionalities essential for mood inference, device interaction, and user state management. Authentication helpers securely manage password hashing and session handling, while Firebase clients handle client-side and admin-level Firestore communication. OpenAI modules prepare structured prompts, format inputs, and trigger LLM-based mood generation. Preprocessing utilities convert raw biometric streams into normalized metrics, and sleep/stress algorithms compute meaningful indicators from heart rate, HRV, and movement patterns. Weather and preference modules enrich context with environmental and personal information, while moodSignal utilities aggregate daily emotional events. Shared validation, time utilities, hooks, and TypeScript types standardize logic across pages and API routes. Collectively, these modules maintain the computational, analytical, and structural consistency that the higher-level system relies on.

3) Location of Source Code

https://github.com/Ma-Hyekjin/mood-manager/Web/src/lib

4) Class Components

- auth/password.ts : Hashes and verifies passwords using bcrypt.
- auth/session.ts : Manages session creation and validation with NextAuth.
- firebase/client.ts : Handles client-side Firebase initialization.

- firebase/admin.ts : Provides server-side Firestore admin access.
- firebase/firebase.ts : Unified Firebase configuration module.
- aws.ts : AWS SDK configuration and S3 operations.
- cache/llmCache.ts : Caches LLM responses to reduce API calls.
- cache/memoryCache.ts : In-memory caching utility for temporary data storage.
- constants/mood.ts : Defines mood-related constants and enumerations.
- openai/index.ts : Sends requests to OpenAI for mood inference.
- openai/formatPayload.ts : Formats data into AI-ready JSON.
- openai/openai.ts : OpenAI client configuration.
- llm/generatePrompt.ts : Generates structured prompts for LLM mood inference.
- llm/optimizePrompt.ts : Optimizes prompt structure and token usage.
- llm/prepareLLMInput.ts : Prepares input data for LLM processing.
- llm/validateResponse.ts : Validates and sanitizes LLM response data.
- preprocessing/preprocess.ts : Normalizes biometric raw data.
- preprocessing/prepareOpenAIInput.ts : Generates structured input for LLM mood generation.
- preprocessing/index.ts : Main preprocessing orchestration module.
- sleep/detectSleepStage.ts : Identifies sleep stages from physiological patterns.
- sleep/calculateSleepScore.ts : Computes score for a sleep session.
- sleep/calculateDailySleepScore.ts : Aggregates daily sleep score.
- sleep/utils.ts : Shared sleep-related math and helpers.
- sleep/index.ts : Sleep analysis module entry point.
- stress/calculateStressIndex.ts : Produces stress index from HRV and movement.
- stress/utils.ts : HRV-related helper functions.
- stress/index.ts : Stress analysis module entry point.
- weather/index.ts : Main module for retrieving weather data.
- weather/fetchWeather.ts : Queries external weather APIs.
- weather/mapGrid.ts : Maps coordinates to grid-based weather models.
- preferences/getPreferences.ts : Retrieves stored user preferences.
- preferences/updatePreferences.ts : Updates user preference dataset.
- preferences/mapPreferencesForAI.ts : Converts preferences into AI-weighted values.
- preferences/preferenceUtils.ts : Helper functions for preference logic.
- moodSignals/fetchDailySignals.ts : Retrieves daily emotional events for mood inference.
- moodStream/scheduler.ts : Manages mood stream scheduling and transitions.
- types/periodic.ts : Defines biometric periodic data types.
- types/sleep.ts : Defines sleep session and stage types.
- utils/validation.ts : Common validation helpers for input checking.
- utils/time.ts : Time formatting and manipulation utilities.
- utils/utils.ts : General-purpose utility functions.
- prisma.ts : Prisma client initialization and configuration.
- hooks/useMood.ts : Fetches and updates mood states.
- hooks/useSurvey.ts : Manages survey completion and state.
- hooks/useMoodStream.ts : Handles real-time mood stream updates.
- hooks/useMoodStreamManager.ts : Manages mood stream lifecycle and persistence.
- hooks/useBackgroundParams.ts : Controls dashboard background effects.
- hooks/useDevices.ts : Manages device list and operations.
- hooks/useDeviceSync.ts : Synchronizes device states with backend.
- types/mood.ts : Defines mood-related data structures.
- types/device.ts : Defines device-related types.
- types/preset.ts : Defines environment preset types.
- types/emotion.ts : Defines emotion classification types.

F. Module 4: ML

1) Purpose

The ML Module of Mood Manager provides the machine learning infrastructure responsible for classifying audio-based emotional cues, specifically laughter, sighs, and neutral sounds. This module operates as an external microservice that processes WAV/Base64 audio streams uploaded from WearOS, transforming them into discrete emotional events used by the backend to refine mood inference. Built with lightweight ONNX models for efficient execution, the ML Module ensures fast, reliable classification across devices and environments. It acts as the system's perceptual layer, extending biometric sensing with voice-based signals and supplying essential multimodal data that enhances personalization.

2) Functionality

The ML Module downloads, preprocesses, and classifies audio samples using trained neural networks exported in ONNX format. It handles dataset acquisition, model conversion, inference execution, and configuration management while providing a REST endpoint for returning classification results to the backend. Preprocessors standardize audio inputs, and model settings configure sampling rates, thresholds, and feature extraction. Saved models and quantized ONNX versions ensure optimized runtime performance. Configuration files allow reproducible model behavior, and utilities support dataset generation, ONNX export, and debugging in local or Dockerized environments. By converting raw acoustic signals into structured emotional labels, the ML Module contributes directly to the Mood Manager's contextual awareness.

3) Location of Source Code

https://github.com/Ma-Hyekjin/mood-manager/markov

4) Class Components

- predict.py : Runs inference on audio samples and returns predicted emotion class.
- sigh_laugh_neg_cls.py : Implements the main classifier for sigh/laugh/negative categories.
- convert_onnx.py : Converts trained PyTorch/TensorFlow models into ONNX format.
- dataset_download.py : Downloads and organizes external datasets for emotion classification.
- ml_settings.py : Configuration file defining model hyperparameters and thresholds.
- fix_config.py : Applies patches or adjustments to existing model configuration files.
- requirements_docker.txt : Lists dependencies for running ML service in Docker or isolated environments.
- Dockerfile : Container configuration for deploying ML inference service.
- .dockerignore : Specifies files to exclude from Docker build context.
- onnx_model/model.onnx : Base ONNX model for audio emotion classification.
- onnx_model/model_quantized.onnx : Optimized quantized ONNX model for runtime inference.
- onnx_model/config.json : Model configuration parameters.
- onnx_model/ort_config.json : ONNX Runtime-specific configuration.
- onnx_model/preprocessor_config.json : Rules for audio preprocessing prior to model inference.
- saved_model/model.safetensors : Model weights in SafeTensors format for secure serialization.
- saved_model/config.json : Full model configuration for reproducibility.
- saved_model/preprocessor_config.json : Preprocessor settings for locally saved model.
- build_yesterday_many.py : Builds Markov-based mood prediction models from yesterday's raw data using DTW clustering.
- realtime_inference_many.py : Flask server for real-time mood state prediction using pre-trained Markov models.

G. *Module 5: WearOS*

1) Purpose

The WearOS Module of Mood Manager serves as the primary data acquisition layer, continuously collecting multimodal physiological and acoustic signals directly from the user's wrist-worn device. Implemented as a native Kotlin application for Wear OS, this module operates autonomously in the background through foreground services, ensuring uninterrupted data capture even when the device screen is off or the user is engaged in other activities. By integrating with Android's Health Connect API and native audio recording capabilities, the WearOS app gathers essential biometric metrics—including heart rate, heart rate variability, stress indices, and respiratory rate—alongside contextual audio events that indicate emotional states such as laughter or sighs. All collected data is transmitted in real time to Firebase Firestore, establishing a reliable and structured data stream that feeds into the backend preprocessing pipeline and ultimately drives AI-powered mood inference. As the foundational sensor layer of the Mood Manager ecosystem, the WearOS module bridges the gap between raw physiological measurements and intelligent mood prediction, enabling the system to respond dynamically to the user's emotional and physical state throughout the day.

2) Functionality

The WearOS Module implements two independent foreground services that run continuously to ensure robust data collection without interruption. The PeriodicDataService queries the Health Connect API at regular intervals to retrieve biometric measurements such as instantaneous heart rate, heart rate variability metrics, stress levels derived from HRV analysis, and respiratory rate estimates. These data points are normalized, timestamped, and uploaded to the Firestore collection `*users/{userId}/raw_periodic*`, creating a time-series dataset that supports sleep detection, stress tracking, and daily wellness scoring. In parallel, the AudioEventService activates the device microphone every minute for a brief two-second recording window, capturing ambient sound and analyzing it for emotional cues. Using lightweight on-device heuristics based on audio amplitude and frequency characteristics, the service classifies detected events as laughter, sighs, or background noise, encoding the raw audio as Base64-encoded WAV and uploading validated events to `users/{userId}/raw_events`. The MainActivity coordinates permission management for sensitive data access—requesting notifications, body sensors, and audio recording permissions—and initializes both services upon app launch. All operations comply with Android's battery optimization and privacy constraints by leveraging foreground service notifications and maintaining minimal resource consumption. The WearOS app's data flows directly into the backend's real-time Firestore listeners, ensuring seamless integration with the broader Mood Manager architecture and enabling context-aware mood recommendations driven by continuous physiological and emotional monitoring.

3) Location of Source Code

https://github.com/Ma-Hyekjin/mood-manager/Watch

4) Class Components

- MainActivity.kt : Main activity that handles permission requests and starts foreground services.
- AudioEventService.kt : Foreground service that records audio every minute and detects laughter/sigh events.
- PeriodicDataService.kt : Foreground service that collects biometric data periodically from Health Connect API.
- FirebaseViewModel.kt : ViewModel for managing Firebase Firestore data synchronization.
- Theme.kt : Wear OS Material Design theme configuration.
- AndroidManifest.xml : App configuration including permissions and service declarations.
- build.gradle.kts : Gradle build configuration for Watch app module.
- gradle.properties : Gradle project-wide properties.
- settings.gradle.kts : Gradle settings for multi-module project structure.

VI. USE CASE

A. Sign Up

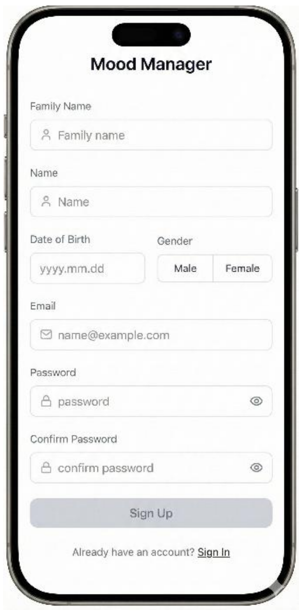


Figure 40: Sign Up

The user enters name, password, birthdate and gender to create a new account. Alternatively, users can sign up through social authentication (Google, Kakao, or Naver), which automatically fills in email and name. After entering information, they can proceed to the login page and attempt to log in with the newly created account.

B. Sign In

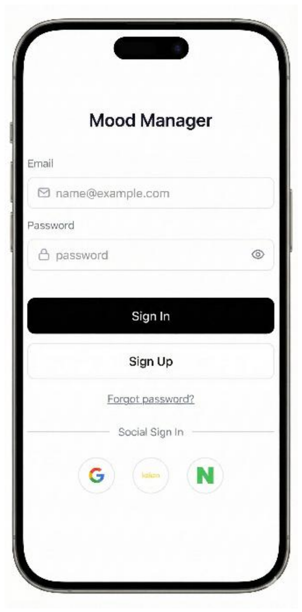
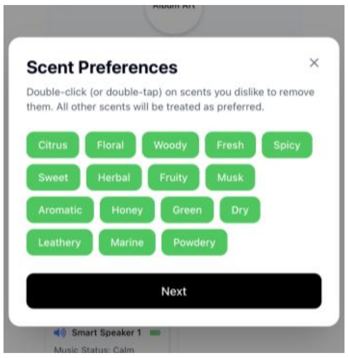


Figure 41: Sign In

After signing up, the login page is displayed. User then enters their registered email and password to log in and access the system. Social login options are also available. The system implements rate limiting, allowing a maximum of 5 login attempts before locking the account for 15 minutes.

C. *Survey*


Scent Preferences

Double-click (or double-tap) on scents you dislike to remove them. All other scents will be treated as preferred.

Smart Speaker 1

Music Status: Calm

Figure 42: Survey

When the user first accesses the home page after registration, a survey overlay appears to set up mood preferences. The survey collects favorite scents, lighting colors, and music genres to personalize AI recommendations. Users can complete the survey or skip it to use default preferences.

D. *Device Registration*

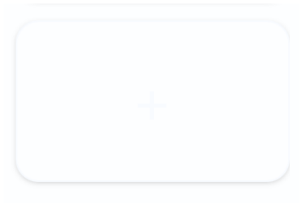
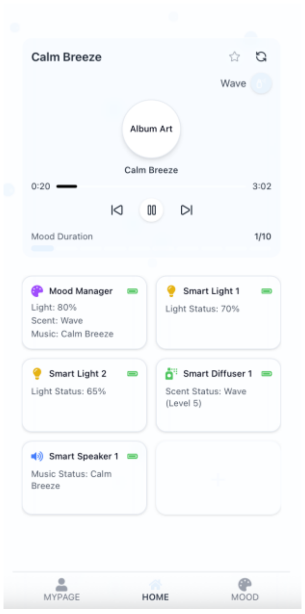


Figure 43: Device Registration

The user clicks the "+ Add Device" card to open the device registration modal. The modal displays four device type options: Manager, Light, Speaker, and Scent. After selecting a device type and entering a name, the system registers the device and displays it in the device grid.

E. *Mood Dashboard*


Calm Breeze

Wave

Calm Breeze

0:20 3:02

Mood Duration 1/10

Mood Manager
Light: 80%
Scent: Wave
Music: Calm Breeze

Smart Light 1
Light Status: 70%

Smart Light 2
Light Status: 65%

Smart Diffuser 1
Scent Status: Wave (Level 5)

Smart Speaker 1
Music Status: Calm Breeze

Figure 44: Mood Dashboard

The mood dashboard displays the current AI-generated mood with a creative name and circular album art showing the associated music track. The dashboard features a 30-minute mood stream timeline divided into 10 segments. Users can interact with the timeline by clicking individual segments to jump to different mood states.

F. *Refresh Mood*

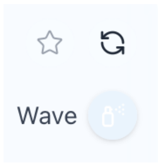


Figure 45: Refresh Mood

The user clicks the refresh button to request a new AI mood recommendation. The system fetches latest biometric data from Firestore, preprocesses it, and calls OpenAI API to generate a new 30-minute mood stream. The dashboard updates with the new AI-generated mood name, music, scent, and lighting combination.

G. Scent Control

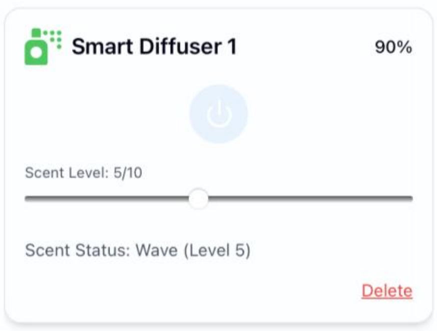


Figure 46: Scent Control

The user can control scent settings through the mood dashboard and device cards. Users can change fragrance type, adjust intensity level, and configure spray interval.

1) Change Scent Type

User clicks the spray bottle icon on the mood dashboard to open a scent selection modal displaying three fragrance options. Upon selection, the system updates the mood preset while keeping music and lighting unchanged.

2) Adjust Scent Intensity

On Manager or Scent device cards, a scent level slider allows users to adjust fragrance intensity from 1 to 10.

3) Configure Spray Interval

Users can set spray frequency through device settings with options for 5, 10, 15, 30, or 60-minute intervals.

H. Music Control

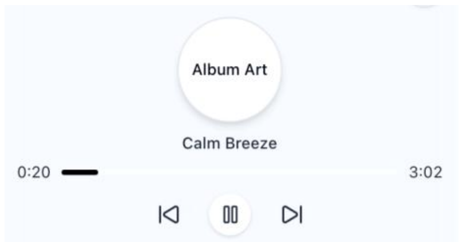


Figure 47: Music Control

The music control section features play/pause, previous, and next track buttons below the album art. A progress bar displays current playback position with time indicators. Users can click the album art to open a song change modal and select from three song options.

I. Lighting Control

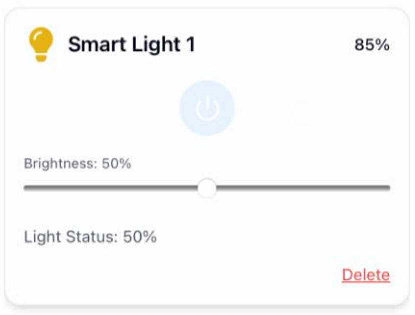


Figure 48: Lighting Control

The lighting color is automatically generated by the AI mood system. Users can adjust brightness level through a slider control (0-100%) on Light or Manager device cards. Manager devices also provide a color picker for manual color override.

J. Mood Duration Bar

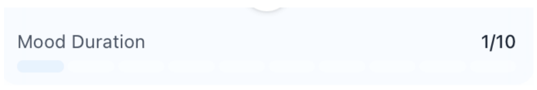


Figure 49: Mood Duration Bar

The mood duration bar visualizes the 30-minute mood stream as an interactive timeline with 10 colored segments. Users can click any segment to jump to that specific mood state in the stream.

K. Device Management

The user can manage devices through the device grid section. This allows control of device power, settings, and deletion.

Figure 50: Device Power Toggle

1) Device Power Toggle

Each device card displays a power toggle switch. Users can click the toggle to turn the device on or off. When powered off, control inputs are hidden and the device stops producing output.

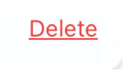


Figure 51: Delete Device

2) Delete Device

Users can delete devices by clicking the delete button on expanded device cards. A confirmation modal appears to prevent accidental deletion.

L. Save Favorite Mood

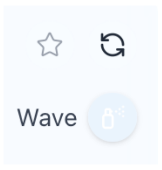


Figure 52: Save Favorite Mood

The user can save the current mood as a favorite by clicking the star icon on the mood dashboard. Double-clicking the dashboard triggers a heart animation effect. Saved moods can be accessed later from the saved moods list.

M. My Page

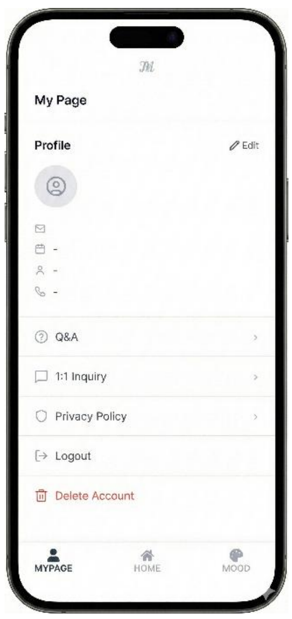


Figure 53: My Page

Users can manage their personal settings and preferences by accessing the 'My Page' section through the bottom navigation bar. On the 'My Page' they can update their profile, view personal information, submit inquiries, and access account management options.

1) Change Profile

User can update their profile information, such as name, birthdate, gender, or profile picture by clicking the "Edit" button. After modifying fields, users click "Save" to persist changes.

2) View Personal Information

Users can check their personal information stored in the system, including email address, registration date, account provider (general/Google/Kakao/Naver), and survey completion status.

3) View Q&A

The Q&A tab displays frequently asked questions about the Mood Manager system, providing answers to common questions about mood recommendations and device management.

4) Submit 1:1 Inquiry

Users can submit support requests through the Inquiry tab. The form requires a title and content. After submission, the inquiry appears in the user's inquiry history with status indicators.

5) Account Management

The Privacy tab provides access to privacy policy and account deletion functionality. Users can permanently delete their account by clicking the "Delete Account" button, which requires password confirmation.

N. Password Reset

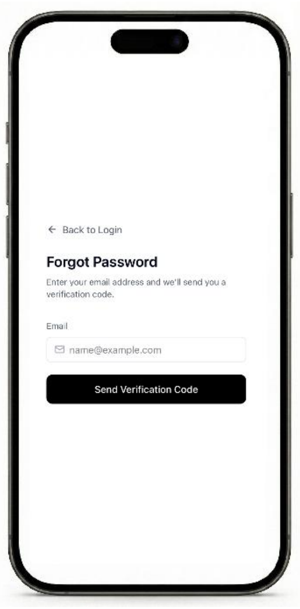


Figure 54: Password Reset

The user clicks the "Forgot password?" link on the login page to access password recovery. User enters their email to receive a 6-digit verification code. After verifying the code, user can set a new password and return to the login page.

VII. AI SPECIFICATIONS

A. Overview

1) System Role

The Mood Manager AI pipeline integrates multiple computational components to infer, stabilize, and articulate user mood states by combining acoustic emotion cues, temporal biometric dynamics, and generative language reasoning. Each subsystem contributes a distinct analytical perspective: the audio classifier provides valence indicators, the DTW–Markov engine models state evolution over time, and the LLM transforms internal outputs into high-level semantic labels.

2) Information Flow

Emotion signals detected from WearOS audio are first classified and encoded as discrete emotional events. These events augment biometric-derived state traces processed through DTW similarity matching, which situates the user in a behavioral cluster. Markov transition modeling then predicts short-term state progression and produces a coherent sequence of mood segments that form a 30-minute stream.

3) Output Integration

The final step uses an LLM to convert the system's internal representation—consisting of selected lighting, scent, and music parameters—into descriptive segment names optimized for clarity and user engagement. This layered approach ensures that the mood stream is simultaneously data-driven, temporally stable, and linguistically interpretable.

B. *Emotional ML*

1) Classification Model

The system employs a lightweight ONNX-based audio classifier to distinguish laughter, sighs, and negative cues from short acoustic bursts captured by WearOS. By leveraging a fine-tuned Wav2Vec2 feature extractor paired with a compact prediction head, the model achieves efficient low-latency inference suitable for serverless deployment.

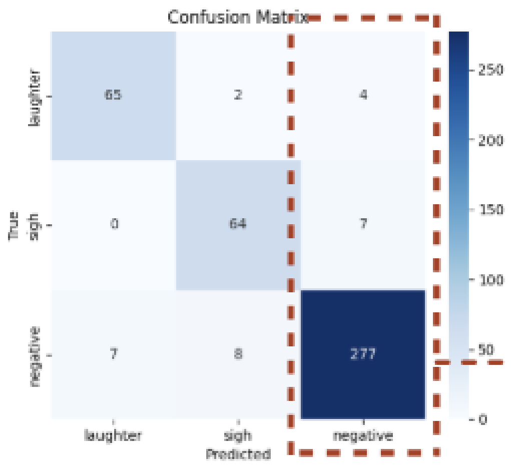


Figure 55: Emotional ML Confusion Matrix

2) Training Process

Training incorporates noise augmentation, class-balanced sampling, and progressive learning-rate scheduling to increase robustness against environmental variability. The resulting classifier demonstrates high precision and stability, providing reliable valence signals that contribute to the user's emotional profile.

3) Runtime Behavior

Inference occurs in AWS Lambda, where the uploaded audio snippet is analyzed and mapped to an emotion label. These labels act as discrete emotional events that influence DTW clustering and Markov transition probabilities in subsequent stages.

C. *DTW-Markov*

1) DTW Similarity

Dynamic Time Warping is used to compare the user's current biometric pattern—such as stress variability or short-term physiological drift—to prior time-aligned sequences. DTW's elasticity supports meaningful clustering even under irregular sampling or nonuniform durations.

2) Markov Modeling

A first-order discrete Markov chain models transitions between coarse user states (calm, tense, fatigued, energized). Transition probabilities are derived from longitudinal DTW-based clustering histories and emotion events, supporting stable prediction of how the user's state is likely to evolve over the next time window.

3) Segment Construction

The Markov-predicted trajectory determines a sequence of mood segments that collectively form a 30-minute mood stream. Each segment encodes a consistent combination of scent, lighting, and music attributes, ensuring that the emotional flow is smooth rather than abrupt.

D. *Large Language Model (LLM)*

1) Naming Generation

A large-language model is employed to generate descriptive and coherent names for each generated mood segment based on the underlying sensory configuration and inferred emotional direction. This enhances the interpretability of the mood stream without affecting functional inference.

2) Stylistic Coherence

The LLM ensures that multi-segment mood streams maintain consistent tone and thematic continuity, producing labels that feel narratively linked rather than isolated presets. This improves user engagement by framing the mood sequence as a cohesive emotional journey.

3) Controlled Scope

The system restricts LLM usage to surface-level linguistic generation. It does not influence DTW computation, Markov transitions, or classifier outputs, thereby maintaining determinism and transparency in the core inference pipeline.

VIII. Discussion

A. *Key Functionalities*

1) DTW & K-menas Clustering

Our project explored several algorithmic methods to capture user state patterns, including DTW-based distance analysis and k-means clustering for grouping similar biometric patterns. These techniques were particularly useful for identifying trends in heart-rate–

derived features and segmenting daily behavioral rhythms.

2) Discrete Markov

We also experimented with discrete Markov chains to model transitions between inferred mood states. This probabilistic approach provided a structured way to represent sequential changes but proved sensitive to sparse or irregular data, revealing limitations in real-world use.

3) Multimodal

Beyond these mathematical models, we attempted a multimodal integration strategy that combined physiological signals, audio emotion detection, and contextual metadata. This fusion allowed us to reason about user conditions more holistically, but it also highlighted the difficulty of balancing heterogeneous data sources within a single inference pipeline.

B. *Difficulties*

1) Recommendation Logic

One of the most significant challenges was developing a reliable logic for mood recommendation. Although we successfully collected biometric, audio, and contextual data from WearOS and Firestore, constructing a stable mapping from these inputs to meaningful mood outputs required more nuanced feature engineering than initially anticipated.

2) AWS Deployment

On the engineering side, our AWS deployment frequently encountered build failures caused by dependency mismatches, environment inconsistencies, and container-layer errors. These issues slowed iteration and emphasized the need for stricter version control and standardized tooling.

3) System Integration

Collaboration introduced additional complexity, as each subsystem—WearOS data collection, server-side preprocessing, ML classification, and frontend UI—evolved at different speeds. Coordinating interface contracts and debugging integration points proved more time-consuming than expected.

C. *Future Work*

Future work will focus on strengthening the mood inference pipeline through more advanced probabilistic and embedding-based models, improving multimodal feature extraction by expanding audio and sleep-related datasets, and enhancing system reliability by stabilizing CI/CD workflows, containerizing ML components more explicitly, and introducing automated integration testing. These improvements aim to deliver a more accurate, scalable, and maintainable system for real-world deployment.

References

[1] "Kotlin Logo," *TechIcons*. https://techicons.dev/icons/ kotlin

[2] "TypeScript Logo," *TechIcons*. https://techicons.dev/icons/ty pescript

[3] "Python Logo," *TechIcons*. https://techicons.dev/ico ns/python

[4] "Next.js Logo," *TechIcons*. https://techicons.dev/icon s/nextjs

[5] "FastAPI Logo," *TechIcons*. https://techicons.dev/ico ns/fastapi

[6] "TailwindCSS Logo," *TechIcons*. https://techicons.dev/icons /tailwindcss

[7] "Android Jetpack Architecture Diagram," *Android Developers Blog*, May 2019. https://android-developers.googleblog.com/2019/05/ whats-new-with-android-jetpack.html

[8] "Amazon Web Services Logo," *TechIcons*. https://techicons. dev/icons/amazonwebservices

[9] "Firebase Logo," *TechIcons*. https://techicons.dev/icons/firebase

[10]"Android Studio Logo," *TechIcons*. https://techicons.dev/icons/ androidstudio

[11]"Visual Studio Code Logo," *TechIcons*. https://techiconsdev/ icons/vscode

[12]"GitHub Logo," *TechIcons*. https://techicons.dev/ico ns/github

[13]"Figma Logo," *TechIcons*. https://techicons.dev/icon s/figma

[14]"Notion Logo," *Wikimedia Commons*. https://commons. wikimedia.org/wiki/File:Notion-logo.svg

[15]"PostgreSQL Logo," *TechIcons*. https://techicons.dev/icons/ postgresql

[16]"Docker Logo," *TechIcons*. https://techicons.dev/icons /docker

[17]"ChatGPT Logo," *Wikimedia Commons*. https://upload.wiki media.org/wikipedia/commons/e/ef/ChatGPT-Logo.svg

[18]"Google Gemini Logo," *Wikimedia Commons*. https://upload.wikimedia.org/wikipedia/commons/8/8f/Google-gemini-icon.svg