SPACE

AI-Driven Multimodal Smart Home Manager

Jun Ho Uh
College of Engineering
Hanyang University
Dept. of Information Systems
Seoul, Korea
djwnsgh0248@hanyang.ac.kr

Yeon Seong Shin
College of Engineering
Hanyang University
Dept. of Data Science
Seoul, Korea
dustjd2651@gmail.com

Dogyeom Kim
College of Engineering
Hanyang University
Dept. of Computer Science
Seoul, Korea
dogyeom74@hanyang.ac.kr

Dong Hyun Lim
College of Engineering
Hanyang University
Dept. of Computer Science
Seoul, Korea
limdongxian1207@gmail.com

***Abstract*—Smart home technology has significantly enhanced household convenience through automation; however, it still faces limitations in user personalization and proactive responsiveness. This project proposes an Intelligent Smart Home Agent System that integrates three key functions: (1) user state analysis leveraging biometric and weather data, (2) persona-based prompting grounded in initial user preferences, and (3) an alert-and-call mechanism for anomaly detection. The system establishes an AI persona whose personality and response style are shaped by the user's basic information and lifestyle patterns. Through ongoing interaction and feedback, the system incrementally refines its behavior, adapting to the user's evolving routines and preferences. Furthermore, the system enables bi-directional voice communication between the user and the home AI. Ultimately, the proposed system—SPACE—aims to establish a more intelligent and empathetic home environment by implementing a context-aware, self-evolving agent that allows seamless interaction from any location, autonomously manages daily conditions, and continuously learns to align with the user's physical and emotional state.**

TABLE I: Role Assignments

Roles	Name	Task description and etc.
User, Customer, Development manager	Jun ho Uh	As the representative user and development manager, Jun Ho Uh identifies user requirements and ensures that system features meet real customer needs. He leads overall project planning, timeline management, and quality assurance. Additionally, he supervises the full software lifecycle—from design and implementation to testing—and guarantees alignment between user expectations and system functionality.
AI Developer	Yeon seong Shin	The AI developer is responsible for designing and implementing intelligent models that interpret user behavior and generate adaptive responses. Yeon Seong Shin develops algorithms for biometric and environmental data analysis, builds multi-agent conversational systems for personalized interaction, and ensures efficient data processing pipelines for continuous learning.
Software Developer (Back-end)	Dogyeom Kim	The backend developer is responsible for safety and efficiently handling user data and AI analysis results in SPACE platform development. Dogyeom Kim designs and builds FastAPI-based API servers and implements real-time data management systems using MySQL. Based on the cloud environment, it connects various APIs to manage data, and focuses on communication between users and AI core and ensuring the scalability and stability of the system's Backend infrastructure.
Software Developer (Front-end)	Dong Hyun Lim	The frontend developer is responsible for designing and implementing the user-facing interface of SPACE. Dong Hyun Lim develops interactive dashboards and responsive layouts that visualize user state information, integrates voice and persona interactions, and ensures seamless communication with backend APIs for real-time updates and feedback.

I. Introduction

A. Motivation

1) Development of Smart Homes

Over the past decade, smart home technology has developed rapidly. Initially, it was limited to simple voice-command-based appliance control, but with the spread of IoT technology, the level of inter-device connectivity and automation has gradually increased.

However, despite this progress, current smart home technologies still remain mechanical and function-centered. Systems simply respond and act according to user commands in a repetitive manner. At present, while "intelligent reactions based on commands" are possible, the system still lacks "emotional understanding" of those commands or awareness of the "context in which they occur."

Entering the 2020s, modern users no longer view smart devices as mere tools but as integral parts of their lives. Smartphones, homes, and furniture are expected to exist not merely as "objects that perform functions" but as "companions that understand personal rhythms and emotions." The true evolution of smart homes should therefore move beyond technological convenience toward expanding user experience and emotional depth. Against this backdrop, we propose the "House–Human Relationship" as a new core value of the smart home system.

2) House-Human Relationship

In the past, the house was a physical space and a base for daily living. In modern society, however, it has transformed into more than just a place of residence — it has become a central space for psychological stability, identity, and emotional recovery. With the rise of remote work, personalized services, and generative AI, the home is increasingly evolving into an interactive interface between users and their environment.

Despite this evolution, most current smart homes still fail to consider emotional connection. Emotional states such as fatigue, happiness, loneliness, or tension are not reflected in how systems operate. The paradigm of user experience (UX) is shifting from convenience to empathy. Today's technology focuses less on "what it can do" and more on "how well it understands me."

When a user says "I'm tired," the goal is not merely to respond with "You must be exhausted," but to dim the lighting to reduce eye strain, play soft music to relieve tension, and create an emotionally responsive environment. The home should provide a sense of emotional feedback — transforming itself from a neutral space into a caring and attentive companion.

The house–human relationship thus evolves from a one-way model, where humans set and control the home, into a mutual learning and adaptive relationship in which the home observes and learns about the user. Through repeated interactions in which the home observes, remembers, and reacts to the user's emotions and patterns — and the user, in turn, grows to trust and feel affinity toward the home — the system gradually develops its own persona. This evolution allows users to perceive their homes not as mere objects, but as beings they live together with.

3) Advancement of Generative AI and Self-Supervised Learning

The rise of generative AI and self-supervised learning has opened the possibility for smart home systems to evolve beyond simple data collection and response into autonomously learning and evolving entities. Large language models, in particular, have reached a level where they can understand emotions, intentions, and context from human voice and dialogue, while reinforcement learning helps optimize their behavioral decision-making.

Furthermore, recent AI services are developing not merely in functionality but toward having distinct personalities. Examples such as ChatGPT, Replika, and Character_AI demonstrate how AI systems can learn users' conversational habits, information needs, and behavioral patterns — generating personalized responses unique to each user. These represent early examples of emotional interaction with AI, illustrating how artificial intelligence can be perceived as a relational and empathetic presence rather than a purely functional one.

B. Problem Statement

1) Limitations of Current Smart Home Systems

Most current smart home systems are designed based on a command-driven architecture, in which device operations are primarily governed by event–trigger rules. This approach results in a structure that reacts to individual device-level inputs rather than managing the home as an integrated environment.

Consequently, the system struggles to process multi-variable contextual conditions—for example, the combination of indoor temperature, humidity, and lighting. As a result, when the temperature is high and the humidity is elevated, the system may activate the air conditioner but fail to coordinate with the dehumidifier, leading to inefficient or excessive energy use.

This limitation arises from the lack of a centralized hub or variable device capable of consolidating these factors into a unified decision-making framework. In essence, the actions of one device are not semantically linked to others, and the system cannot preserve or reference contextual information in subsequent interactions.

Another inherent limitation of command-based architectures lies in their inability to understand user context and emotion. Statements such as "I'm a bit tired today" do not correspond to explicit commands directed toward a particular device, and therefore elicit no system response. Moreover, identical phrases can have different meanings depending on context—saying "It's hot" in summer differs significantly

from saying the same in winter, or expressing "It's cold" at 1 p.m. versus at 3 a.m.—yet current systems fail to distinguish these nuances.

This issue originates from the reliance on speech-to-text conversion followed by purely textual analysis, which strips away emotional and situational cues. Additionally, incorporating multimodal contextual factors such as location, time, schedule, lifestyle patterns, recent behavior, and weather conditions requires a more sophisticated computational framework. This makes it difficult for existing systems—designed for lightweight, rule-based operation—to process emotional or contextual information while maintaining low hardware and processing overhead.

2) Information Deficiency

Although current smart home systems have access to a vast amount of user data, they lack the ability to interpret this information contextually or utilize it continuously. Sensors and IoT appliances collect enormous volumes of environmental and behavioral data; however, such information is used primarily for real-time reactions rather than for long-term learning or personalization.

As a result, the system must respond to each user action as if it were new, without reference to accumulated experience, making it impossible to form an adaptive or relational understanding of the user over time.

In most existing architectures, event logs and state data are retained only for a predefined short duration and are subsequently deleted. Consequently, if the system fails to learn from these data within that limited window, the information is never integrated into the AI's internal state, and thus has no influence on its future behavior.

Moreover, because each device typically stores its data on separate, vendor-specific servers, the system as a whole cannot consolidate relationships across devices or evolve a coherent, unified model of the household. This fragmented and short-term data management leads to a fundamental absence of memory, context, and continuity, preventing the smart home from developing a truly personalized and evolving intelligence.

3) Lack of Awareness toward Personal Management Systems

The development of smart home technologies has so far focused primarily on the automation of device control. As a result, most smart home systems operate from a device-centered perspective rather than a human-centered one. Current platforms recognize the user merely as the issuer of control commands, not as an active participant whose physical and emotional states influence the home environment.

Consequently, human-centered factors such as emotion, condition, daily rhythm, and behavioral patterns are rarely considered as core variables in system design. This structural limitation causes smart homes to provide only functional convenience, failing to deliver the psychological stability and lifestyle management that users increasingly expect from intelligent living spaces.

Such a limitation prevents the smart home from evolving into a Personal Management Platform—a system that not only automates tasks but also understands, supports, and manages the user's well-being. For smart homes to mature into truly intelligent systems, they must recognize the user as more than a source of commands—as a subject of management and learning, whose behavioral and emotional data continually shape the home's adaptive intelligence.

C. Solution

1) Persona-Based User Understanding and Relationship Formation

The most fundamental issue with current smart home systems is that they recognize the user merely as a command generator. To address this, we introduce an AI Persona System designed to learn from and remember all interactions with the user.

The persona continuously evolves based on the user's preferences, lifestyle patterns, conversational style, and emotional expression. When a user says, "I'm having a tough day," the system doesn't just process that statement. Instead, it comprehensively considers the environmental settings the user preferred in similar situations, the corresponding time of day and weather, and the user's satisfaction afterward.

Unlike traditional systems that delete data after a set period, this approach accumulates all experiences as part of the system's personality and knowledge.

As a result, the user gains a life partner that understands them deeply, without needing to repeat the same settings each time. This forms the core foundation for the smart home's evolution from a simple device-control platform into a Personal Management Platform.

2) Voice-Based Contextual Understanding and Real-Time Communication

When the system detects unusual events in the home—such as a fire alarm or the sound of a fall—it immediately calls the user to confirm the situation. During this call, it analyzes the user's tone, speech rate, and intonation to determine whether a real emergency is occurring.

Furthermore, the system distinguishes and responds differently to the same phrase—for example, "It's a bit cold"—depending on whether it's spoken with a trembling voice at 3 a.m. or casually at 1 p.m.

This approach goes beyond simple text-based command processing, advancing to a level that integrates the user's emotional state and situational context. Consequently, the system can respond appropriately to implicit expressions like "I'm feeling worn out today," allowing users to create the desired environment through natural conversation rather than precise commands—fulfilling the role of a genuine life companion.

The system leverages LG's existing home cameras for real-time detection and voice call functions. If a camera detects abnormal sounds—such as a fall, glass breaking, or unusual pet behavior—it stores a 10–20 second pre-buffer and calls the user to confirm the situation. The user can immediately respond during the call with "I'm okay" or "I need help," and this feedback continuously improves the model's accuracy.

All recorded data is automatically deleted upon user confirmation, with only the short buffer segment temporarily stored to ensure privacy protection.

3) User-Centric Environment Optimization Through Multi-Dimensional Data Integration

A major inefficiency in current smart homes is that they control temperature, humidity, and lighting on a per-device basis—often leading to contradictions such as the air conditioner and dehumidifier running simultaneously. These systems also tend to react solely to sensor data, regardless of the user's actual comfort level.

By integrating and analyzing biometric data and weather information, the proposed system can understand the user's real condition and recommend appliance operations accordingly.

The model predicting the user's condition doesn't just process the fact that "the temperature is 28°C." Instead, it interprets the context: "the user feels uncomfortable due to 28°C and high humidity while feeling tired." Based on this, it comprehensively adjusts the air conditioner, dehumidifier, and lighting brightness to create an environment optimized for recovery.

The process of confirming the system's predictions with the user enables continuous improvement, overcoming the "single-use data" problem in existing systems.

This signifies a shift from a device-centric to a user-centric paradigm. The decision-making center is no longer which appliance to turn on but what the user needs right now.

As a result, the user experiences an environment optimized for their condition without complex manual settings, while the system operates efficiently without wasting energy.

4) Predictive Lifestyle Management Through Habit Pattern Learning

Existing smart homes rely on fragmented information—such as whether the user is currently home or not—failing to grasp the broader context or continuity of daily actions. The User Routine Tracking System analyzes travel routes, dwell times, and recurring lifestyle patterns to understand the user's full day. It combines this with biometric and weather data to enable predictive lifestyle management.

For example, the system might determine that the user is returning home later than usual, had high travel activity, and shows signs of fatigue based on biometric data.

In response, it proactively creates an environment for recovery by dimming the lights, lowering the indoor temperature, and preparing calming music before the user arrives.

Conversely, if the system detects an unusual outing pattern on a weekend morning, it suggests environmental settings suitable for an active day.

This establishes a system that manages and cares for the user's overall life, moving beyond simple automation. It transitions from a reactive structure (event occurs $\rightarrow$ immediate response) to a proactive one (pattern learning $\rightarrow$ situation prediction $\rightarrow$ preemptive suggestion).

Consequently, the user experiences having what they need prepared even before issuing a command, and the smart home functions as a true Personal Management Platform.

Moreover, long-term pattern learning can detect subtle changes in the user's life rhythm, offering meaningful insights for health management and lifestyle stability.

D. Research on related software

1) LG ThinQ

LG ThinQ is an integrated smart home platform that connects and controls all LG appliances within the household through a centralized hub architecture. It provides the highest level of interoperability and system stability within the LG ecosystem.

In particular, the ThinQ API enables precise monitoring and control of appliance states such as those of washing machines, air conditioners, and air purifiers. Furthermore, the UP System adopts a modular design that supports continuous enhancement of appliance functionality through over-the-air (OTA) updates, allowing devices to evolve even after deployment.

These technical features constitute a core operational foundation for this project, enabling the behavioral policies generated by the AI persona to be executed reliably at the appliance level. In other words, LG ThinQ serves as the most essential interface that ensures the accurate and dependable realization of the persona's decisions within the physical smart home environment.

2) Samsung SmartThings

Samsung SmartThings is a representative multi-brand smart home platform that enables users to control devices from various manufacturers within a unified application environment. At the hub level, it employs Edge Driver technology to support local device control, thereby minimizing network latency and instability while reducing dependence on cloud processing.

In addition, its Energy Management System provides real-time visualization and control of power consumption for individual appliances, achieving a high level of completeness in terms of energy efficiency management.

This high degree of interoperability and data integration capability makes SmartThings particularly effective for this

project, which aims to manage and unify appliances from multiple brands under a single policy framework.

Moreover, since the SmartThings ecosystem is natively integrated with the Matter standard, it provides a strong infrastructural foundation for expanding the proposed system's policies and control mechanisms into a universally compatible smart home environment.

3) Google Home (Assistant)

Google Home is Google's smart home platform, developed through the integration of Nest devices and Chromecast, and is characterized by its deep connectivity with the broader Google ecosystem—including Google Account, Calendar, and Maps. Through this contextual integration, the platform can directly incorporate the user's schedule, location, and habitual behaviors into automated routines and scenario generation.

In particular, Google Assistant's advanced speech recognition and natural language processing (NLP) technologies have reached a level where the system can not only interpret explicit commands but also understand conversational expressions in context and translate them into corresponding automated actions.

This natural language–based automation mechanism closely aligns with the conversational engine structure of our proposed system, serving as a direct reference for designing models that convert dialogue-based instructions into policy-level inputs. Furthermore, by linking with calendar and location data, Google Home can provide valuable contextual and emotional cues, enhancing the system's ability to accurately perceive the user's current state and situational context.

4) openHAB

openHAB is an open-source home automation hub designed to minimize dependence on cloud infrastructure, specializing in local control and on-premise automation. Because it can be executed directly on a user's own server or single-board computer, openHAB allows data to be securely stored within the household, offering strong advantages in terms of data privacy and user sovereignty.

At the same time, it maintains compatibility with major cloud-based platforms such as Google Assistant, Amazon Alexa, and Apple HomeKit, thereby supporting a flexible hybrid operating environment.

Currently, openHAB provides integration with over 3,000 global brands and devices, including Qualcomm AllPlay, Android TV, BenQ, and AIRTHINGS. In addition to its extensive interoperability, the platform enables local learning and inference, making it a valuable experimental base for testing reinforcement learning (RL) models or persona-driven policy frameworks within a controlled, privacy-preserving environment.

This combination of openness, scalability, and on-device intelligence positions openHAB as a practical research platform for developing and validating next-generation smart home systems.

5) Apple HomeKit

Apple Home is a smart home platform deeply integrated with the iOS ecosystem, built upon an architecture that prioritizes security and privacy above all else. Through the HomeKit protocol, only certified and authenticated devices are permitted to connect to the network, ensuring a highly reliable and secure environment.

More recently, Apple Home has adopted support for the Matter standard and Thread networking, significantly enhancing the stability and interoperability of low-power devices such as sensors and lighting systems.

In addition, Apple's characteristic consistent iOS user experience (UX) and the seamless integration of the Siri voice assistant provide users with high accessibility and an intuitive automation setup process. These features directly align with our project's design philosophy, which emphasizes on-device learning and local data protection.

In particular, HomeKit's local inference architecture serves as a concrete reference model for developing the project's on-device persona intelligence strategy, where personalized behaviors and contextual reasoning are executed securely within the user's own environment.

6) Amazon Alexa

Amazon Alexa is a voice-assistant platform built around smart speaker technology and is currently one of the most widely adopted voice interfaces for smart home ecosystems. Alexa offers thousands of extensions (Skills) and automation scenarios (Routines) that enable a broad range of functions, including appliance control, music playback, and information retrieval.

In recent developments, Alexa has integrated large language model (LLM)-based natural language routine generation, allowing users to configure automation simply through conversational expressions such as "When this happens, do that."

This advancement in voice user experience (UX) and affective language processing represents one of the most mature commercial implementations of an emotionally responsive home environment. In this regard, Alexa serves as a practical reference for the vision of our system—a home that understands and responds to the user's emotions and contextual needs, bridging natural conversation with intelligent, adaptive automation.

7) Matter

Matter is a global standard protocol designed to ensure interoperability among smart home devices, allowing them to operate seamlessly regardless of brand or platform.

By utilizing Thread networking technology, Matter enables mesh communication among low-power devices such as sensors and switches, while its simplified commissioning process enhances both the scalability of smart home environments and the consistency of user experience.

For the proposed system, Matter serves as a foundational technological framework that ensures the persistence and portability of learned behaviors. When the AI persona generates automation scenarios based on its learned behavioral policies, Matter allows these policies to be transferred and maintained across newly added or replaced devices.

This interoperability provides the essential infrastructure for realizing a continuously evolving and universally compatible intelligent home environment.

8) Hume AI

Hume AI is an affective artificial intelligence research company that quantifies users' emotional states by analyzing subtle variations in vocal tone, speed, intensity, and prosody. Beyond simple positive–negative sentiment classification, Hume AI's models identify underlying vocal patterns to estimate complex emotional indicators such as stress, fatigue, and arousal levels.

Furthermore, the company's technology can infer emotional states using microphone input alone, without requiring visual data from cameras, making it particularly effective and privacy-compliant in environments where visual information collection is sensitive or restricted.

9) Replika

Replika is an emotionally adaptive chatbot that continuously learns from users' conversational data to develop individualized personalities and emotional responses. By analyzing each user's linguistic patterns, emotional expressions, and conversational history, Replika maintains relational consistency while gradually evolving its persona over time.

This architecture provides a direct conceptual foundation for the design of smart home agents that move beyond simple command execution—enabling systems that continuously optimize their behavioral policies and response styles through ongoing user feedback and long-term interaction.

10) Affectiva

Affectiva is a pioneer in emotion recognition (Emotion AI) technology that analyzes facial expressions and vocal cues to interpret human emotions with high precision.

Its research and commercial applications span multiple industries—including automotive systems, robotics, and advertising analytics—focusing on designing interactive systems that adapt their responses based on users' emotional states.

This approach offers valuable insights for the development of multimodal persona architectures in smart home agents, enabling them to interpret and respond to emotional signals not only from voice data but also from visual inputs, thereby enhancing the depth and empathy of human–AI interaction.

11) Twilio

Twilio provides a comprehensive cloud-based voice communication infrastructure through its Programmable Voice API, enabling applications to initiate, receive, and manage phone calls programmatically.

The platform supports multiple communication protocols, including WebRTC, SIP (Session Initiation Protocol), and PSTN (Public Switched Telephone Network), allowing the implementation of reliable bidirectional voice communication between the home AI system and the user—whether the user calls the home AI or the home AI initiates a call to the user.

Twilio's API further supports real-time speech recognition (STT), DTMF input detection, and IVR (Interactive Voice Response) flow control, while the use of TwiML (Twilio Markup Language) allows developers to model detailed call scenarios directly at the code level.

When integrated with the Model Context Protocol (MCP), this architecture enables the AI model to dynamically manage call flows, process real-time call events as contextual inputs, and perform conversational decision-making within an ongoing voice interaction.

12) Vonage

Vonage is a global communication platform centered on its Voice API, supporting bidirectional voice streaming through WebRTC and SIP-based protocols. The Vonage API allows real-time transmission of audio streams to external AI endpoints during a call, making it highly suitable for AI-integrated call processing tasks such as speech recognition, emotion analysis, and intent understanding.

In particular, the Voice Connectors feature enables direct linkage with the Model Context Protocol (MCP) via a WebSocket interface, allowing the AI model to interpret user utterances in real time and generate immediate, context-aware responses during an ongoing conversation.

For instance, in a Vonage Voice API–enabled call, if the user's voice tone rises, the MCP can recognize this as a change in emotional state, triggering an emergency response mode, or prompting the conversational persona to respond in a calmer, more empathetic tone.

In this way, Vonage transforms call audio from a mere sound stream into a contextual communication medium, providing a crucial technological foundation for implementing the persona-based bidirectional communication framework proposed in this project.

13) Google Maps Timeline

Google Maps Timeline is a location-tracking service that records users' movement paths, visited places, transporta-

tion modes, dwell times, and travel distances based on a combination of GPS and Wi-Fi signal data. By integrating additional contextual information from Google's web and app activity as well as photo metadata, the system constructs a comprehensive log of the user's daily mobility and behavioral patterns.

This multi-source dataset provides valuable contextual signals that, when combined with biometric and weather data, significantly enhance the accuracy of lifestyle and condition analysis models—enabling fine-grained estimation of user activity levels, fatigue, and rest cycles.

Within the proposed system, the location and activity data from Google Maps Timeline are incorporated as contextual inputs for adaptive decision-making. For instance, if the system detects higher-than-usual mobility or a delayed return home, it can infer potential fatigue and proactively suggest personalized environmental adjustments, such as "Would you like to dim the lights?" or "Shall I activate sleep mode?"

To ensure privacy, raw location data are anonymized and aggregated into grid-based representations stored locally on the device, while users maintain full control over the scope and duration of timeline recording and sharing preferences.

II. Requirement

A. User Management

1) Sign Up

The registration process requires the following information:

- Mobile number: Verified through carrier authentication and later used as the login ID
- Password: Must be at least 8 characters long and include three of the following: uppercase letters, lowercase letters, numbers, and special characters. When requirements are met, indicators turn green; when unmet, they turn red. Password input is masked for security
- Name: Required field; used as the default nickname upon first login and for ID recovery
- Date of birth: Required field; triggers birthday pop-up notification annually and used for ID recovery
- Email: Used for account recovery, additional authentication, and receiving important notices

2) Sign In

The system supports two authentication methods:

- Local login: Users enter their ID (mobile number) and password. If credentials match, the system redirects to the main page; otherwise, a "Member does not exist" popup is displayed.
- SNS login: Users can authenticate via Apple, Google, Kakao, or other social platforms. After consent, the system connects to the same user profile based on mobile key linkage.

B. Device Management

1) Register Home Appliance

The system provides two registration methods:

- QR scan (requires camera permission)
- Device search via Wi-Fi/BLE scan or manual input of model and serial number

Import from LG ThinQ: After OAuth consent, the system imports registered appliances, room layout, and smart routines. Imported data is used exclusively for control and automation suggestions.

C. Data Collection and Privacy

1) Data Sources and Permissions

The system collects data from multiple sources with explicit user consent

- Biometric and activity data: Collected from wearable devices and phone sensors, including heart rate, activity levels, sleep patterns, and step counts. Data collection is minimal and requires explicit consent.
- Weather and environment data: Obtained from external weather APIs and indoor sensors measuring temperature, humidity, CO2 levels, and other environmental factors.
- Required permissions: The system requests notifications, location (for weather data), and health/activity data access. The purpose and retention period for each permission are clearly disclosed to users.

D. AI-Driven User State Inference

1) Data Integration and Modeling Framework

The system integrates biometric signals from Apple HealthKit, contextual data from GPS-based routine tracking, and environmental factors such as temperature and humidity to infer the user's overall physical and mental condition. At the early stage, it operates on scientifically validated rule-based algorithms, and as sufficient data accumulates, it transitions into a machine-learning–based model for personalized prediction.

2) Condition Indicators

The core of the system consists of five condition indicators: fatigue, stress, thermal comfort, light preference, and recovery need. Fatigue is calculated from sleep duration, deep-sleep ratio, HRV (Heart Rate Variability), and circadian rhythm patterns. Stress is derived from increased heart rate, decreased HRV, respiratory rate, and environmental noise. Thermal comfort reflects the optimal temperature (18–28 °C) adjusted according to activity intensity, outdoor climate, and humidity. Light preference dynamically adapts brightness and color temperature based on time of day, fatigue, and sleep readiness. Finally, recovery need evaluates post-exercise physical recovery using heart-rate recovery speed, HRV rebound, and respiration rate. Each indicator is normalized by the user's personal baseline, minimizing physiological variability among individuals.

3) Baseline Construction and Adaptive Learning

The baseline is established after at least seven days of data collection and includes metrics such as HRV, resting heart rate, sleep duration, and activity levels. It is automatically updated on a weekly basis, allowing the system to adjust to the user's long-term physiological trends. Once a sufficient dataset ($\geq$30 days and $\geq$200 feedback samples) has been accumulated, the rule-based engine is replaced by a XGBoost model, enabling adaptive, data-driven condition estimation.

4) Generative AI Condition Interpretation

The analyzed condition output is passed to a Generative AI module, which converts it into a structured prompt describing the user's current physiological and environmental context. The AI then generates personalized appliance-control commands—for example, "activate dehumidifying mode at 27 °C after exercise," or "set lighting to 50% brightness and 3500 K color temperature for relaxation." Each recommendation includes concise scientific rationale (e.g., HRV recovery facilitation or melatonin preservation) derived from established sources such as WHO, ISO 7730, and AASM sleep guidelines.

5) RLHF Feedback Loop and Continuous Optimization

These AI-generated suggestions are presented to the user in a card interface, allowing quick responses such as Run now, In 30 min, or Skip today. All user interactions feed back into a Reinforcement Learning from Human Feedback (RLHF) loop, where approval rates, adjustment patterns, and contextual variables are used to continuously refine prompt templates, decision thresholds, and model confidence levels. Over time, the system evolves into a personalized predictive assistant that learns from each interaction and aligns closely with the user's preferences and habits.

E. Apple Watch Integration

1) GPS and Location Awareness

The Apple Watch integration enables real-time tracking of the user's geographical context through GPS data. By continuously monitoring location, the system determines whether the user is near home or traveling. This spatial awareness allows the agent to adapt its behavior — for instance, adjusting home device readiness when the user is approaching or providing location-based reminders. Additionally, accumulated GPS traces are visualized on an adaptive map, supporting routine pattern analysis such as daily commuting routes, time spent outdoors, and regional activity trends.

2) Health Data Synchronization

Through Apple's HealthKit framework, the agent securely retrieves physiological and activity data including heart rate, sleep duration, step count, and exercise sessions. These metrics are continuously analyzed to infer the user's physical condition, stress levels, and lifestyle patterns. Based on the analysis, the system can generate personalized insights — such as recommending rest after elevated stress detection or adjusting environmental conditions (e.g., temperature or lighting) following high activity levels.

3) Behavioral Mapping and Wellness Insights

By combining GPS-derived mobility data with real-time health indicators, the agent constructs a comprehensive behavioral map of the user's daily life. This integration allows it to identify correlations — for example, linking reduced step count with poor sleep quality or detecting stress patterns associated with long commutes. Ultimately, the Apple Watch module transforms raw sensor data into actionable wellness intelligence, enhancing personalization, safety, and long-term health awareness.

F. Personalization

1) Persona Configuration

The personalization framework is centered on the concept of an AI persona configured during the onboarding phase. Upon initial setup, the user selects the agent's tone and communication style — for instance, friendly, professional, or concise — and provides basic profile information such as name, date of birth, occupation, sleep–wake cycle, and weekly exercise frequency. This data directly influences the tone of notifications, phrasing of suggestions, and communication style in voice or text interfaces.

2) Adaptive Personalization and Behavioral Learning

As the system observes real user behavior and feedback, it gradually increases personalization depth. For example, if a user consistently approves "cooler settings after exercise," the agent autonomously learns to trigger air conditioning suggestions post-workout. Similarly, recurring actions such as "dim lights at night" become embedded as personal automation rules, minimizing the need for repeated confirmations.

3) Safety and Transparency

High-risk or low-confidence actions (e.g., oven or gas control) always require explicit user confirmation. Every automation remains overrideable at any time, ensuring user safety, transparency, and trust. Through continuous learning, the agent transitions from a static rule-based assistant to an adaptive, user-aware smart companion.

G. Proactive Automation

1) Autonomous Appliance Adjustment

Instead of waiting for user confirmation, the system autonomously adjusts appliances based on real-time user condition analysis. When the AI detects significant fatigue, elevated stress, or temperature discomfort, it immediately

configures connected devices such as lighting, air conditioning, or air purifiers to match the predicted comfort range without requiring prior user input. This enables seamless environmental adaptation that aligns with the user's physiological and contextual state.

2) User Feedback and Re-adjustment

After an automatic adjustment is applied, users can freely modify or reject the new settings. For instance, if the user changes the air conditioner temperature or disables lighting adjustments, the system interprets this as a negative feedback signal. All such interactions are recorded as behavioral feedback data and used to update the internal preference model.

3) Adaptive Learning Loop

Each user correction or rejection is treated as a learning instance. The reinforcement loop continually analyzes these signals to fine-tune decision thresholds, preferred control parameters, and contextual mappings. Over time, this process transforms the automation from a static ruleset into a personalized predictive controller that autonomously optimizes appliance behavior based on accumulated feedback and evolving user preferences.

4) Safety and Transparency

For safety-critical or high-impact actions (e.g., oven, gas, or electrical control), the system enforces reconfirmation or multi-factor authentication. All adjustments remain transparent and reversible; users can override or reset automated actions at any time, ensuring reliability, trust, and control in the automation process.

H. Communication System

1) Communication Channels

Push notifications serve as the default communication method, with SMS and email available as alternatives. For urgent alerts or complex issues, the system can initiate voice calls to establish direct agent-user phone connection through Voice call (VoIP/ARS). In addition, Users can configure do-not-disturb hours, priority levels, guardian designation, and language preferences through contact settings.

I. Security and Compliance

1) Privacy, Security, and Consent

The system implements comprehensive privacy and security measures:

Data handling: Minimal data collection principle is enforced, with encryption applied during both transmission and storage. Personally identifiable information (PII) is separated from telemetry data.

ThinQ integration: Minimal permission scope is requested from LG ThinQ, with prior disclosure of all retrieved items.

User rights: Users can download or delete their data, unlink ThinQ integration, and deactivate their account at any time.

Audit trail: Every automation execution and contact attempt is logged for audit purposes.

2) Reliability, Error Handling, and Offline Support

The system implements robust error handling mechanisms:

Device availability: Offline devices are detected and indicated, with recovery guidance provided. Non-critical commands are queued for later execution.

Conflict resolution: When ThinQ and local status information conflict, the system presents a resolution prompt or applies a single-source policy.

Schema compatibility: Core controls are limited to finalized schemas, with older values displayed as legacy indicators.

III. Development Environment

A. Choice of Software Development Platform

1) Development Platform

a) macOS Development Environment

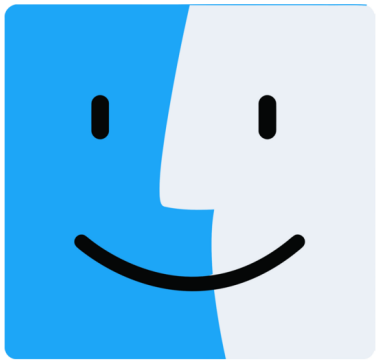

Fig. 1: macOS

macOS served as the primary development environment for iOS and watchOS applications. All implementations were carried out using **Xcode 16.0** on **macOS Sonoma 15.0**, running on an Apple MacBook Pro (M2, 16 GB RAM). This platform ensured full compatibility with Apple's SDKs and provided integrated tools for debugging, WatchKit simulation, and real-device testing. The macOS environment also enabled seamless SwiftUI preview and build automation through Apple's native developer ecosystem.

b) Windows Environment for Backend Testing

Fig. 2: windows11

Windows 11 was utilized as a secondary development environment to verify backend communication and data exchange between the FastAPI server and client devices. Docker containers were configured to host FastAPI and MySQL instances, providing a consistent testbed across both macOS and Windows. This setup ensured that the backend could be validated independently from the Apple development environment and maintained cross-platform compatibility.

c) iOS Platform (iPhone)

The iOS application served as the main interface for collecting health, GPS, and contextual data. Developed using **Swift 5.10** and **SwiftUI**, it integrates Apple frameworks such as `HealthKit`, `CoreLocation`, and `MapKit`. Testing and optimization were performed on an **iPhone 14 (iOS 18.0)** device, ensuring stable background synchronization and low-latency data transfer through the `WCSession` framework. The iOS platform also enabled direct visualization of health metrics and movement trajectories within the app's dashboard.

d) watchOS Platform (Apple Watch)

Fig. 3: WatchOS

The watchOS component was responsible for collecting real-time biometric and activity data through the Apple Watch sensors. An **Apple Watch SE (2nd generation, watchOS 10.1)** was paired with the iPhone for continuous synchronization of heart rate, HRV, and exercise sessions via the `HealthKit` and `WatchConnectivity` frameworks. The lightweight watchOS design enabled background data capture with minimal power consumption, ensuring continuous monitoring without user intervention. All collected data were serialized and securely transmitted to the iOS host app, forming the foundation of the system's health monitoring pipeline.

2) Language/Framework

a) Docker

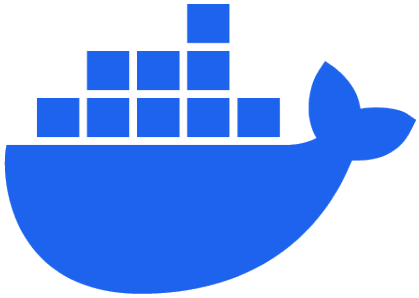

Fig. 4: Docker

Docker is an open-source containerization platform that enables developers to package applications and their dependencies into lightweight, portable containers. By isolating software components from the host system, Docker ensures consistent performance across development, testing, and production environments. It simplifies deployment pipelines and promotes scalability through container orchestration tools such as Docker Compose and Kubernetes. Docker's image-based architecture allows reproducible builds, rapid scaling, and seamless collaboration among distributed teams.

b) MySQL

Fig. 5: MySQL

MySQL is a widely used open-source relational database management system based on Structured Query Language (SQL). It provides reliable data storage, indexing, and querying capabilities for web and enterprise applications. MySQL ensures data integrity through ACID compliance and supports multi-user access with transaction control and authentication mechanisms. In this project, MySQL was used to manage structured data

efficiently, offering strong scalability and compatibility with Docker-based deployments.

c) Mermaid

Fig. 6: Mermaid

Mermaid is a JavaScript-based diagramming and visualization tool that converts text definitions into dynamic charts and diagrams. It supports flowcharts, sequence diagrams, entity-relationship diagrams (ERD), and Gantt charts, enabling engineers to describe complex architectures in a simple, markdown-style syntax. Within this project, Mermaid was employed to visualize database schemas and backend architecture, enhancing documentation clarity and maintainability.

d) Google Cloud Platform

Fig. 7: Google Cloud Platform

Google Cloud Platform (GCP) is a comprehensive suite of cloud computing services provided by Google. It offers scalable infrastructure for computing, storage, networking, and machine learning applications. GCP enables developers to deploy and manage containerized applications using services such as Cloud Run, Compute Engine, and Kubernetes Engine. In this project, GCP was used to host backend APIs and manage containerized environments with high scalability and reliability.

e) FastAPI

Fig. 8: FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python 3.7+ based on standard type hints. It supports asynchronous I/O and automatic request validation using Python type annotations, making it both efficient and developer-friendly. FastAPI automatically generates interactive API documentation using OpenAPI and Swagger UI, improving testing and maintainability. In this project, FastAPI served as the backend framework responsible for connecting user interfaces, databases, and external services such as the Korea Meteorological Administration (KMA) API.

f) KMA OpenAPI

Fig. 9: Korea Meteorological Administration

The KMA OpenAPI provides official weather data released by the Korea Meteorological Administration. It offers various endpoints including ultra-short-term observation, ultra-short-term forecast, and village forecast, which provide weather information at intervals of 10 minutes to 1 hour based on a 5 km × 5 km grid (nx, ny) system. This API enables real-time access to weather conditions such as temperature, humidity, precipitation, wind speed, and sky condition across South Korea. In this project, the KMA API was integrated into the backend system to collect localized weather data, which was later combined with user health and location information to enhance contextual analysis and AI-driven stress prediction.

g) Swift / SwiftUI

Fig. 10: Swift / SwiftUI

Swift is Apple's modern programming language optimized for safety, performance, and expressiveness. In this project, Swift 5.10 and SwiftUI were used to implement both the iPhone and Apple Watch interfaces. SwiftUI's declarative syntax simplified reactive UI updates in response to changing health and GPS

data, while seamless state synchronization between watchOS and iOS enabled real-time visualization of sensor readings. The combination of Swift and SwiftUI improved maintainability and reduced boilerplate code, allowing faster iteration and debugging throughout the development cycle.

B. Software in Use

1) Apple Fitness+ / Apple Fitness (Apple)

Apple Fitness+ and its companion Apple Fitness app provide structured workout programs, personalized recommendations, and motivational tracking features based on data collected through the Apple Watch. The service offers real-time visualization of workout metrics—such as heart rate, calories burned, and activity ring progress—and synchronizes seamlessly across iPhone, iPad, and Apple TV. However, its core focus remains on promoting exercise engagement rather than integrating multi-dimensional data sources like GPS trajectories or meteorological context. In contrast, the proposed system expands upon this foundation by combining health, spatial, and environmental data to support adaptive and contextual behavior analysis.

2) Fitbit-based Health Tracking Platforms

Fitbit's health tracking ecosystem is among the most established wearable platforms, enabling users to monitor daily activity, sleep, and heart rate through connected devices and a cloud-based mobile interface. Its APIs and SDKs facilitate health research, allowing correlation studies on movement, sleep quality, and stress management. Nonetheless, Fitbit's data model primarily concentrates on individual domains—such as physical activity or sleep—without capturing contextual environmental data. Our system advances this approach by integrating real-time GPS and weather data, constructing a unified schema that relates health trends with environmental and behavioral factors.

3) AI-driven Wearable Data Recommendation Systems

Recent AI-powered fitness systems, such as the "Privacy-Preserving Personalized Fitness Recommender System (P3FitRec)," employ deep learning to analyze multi-sensor data and generate personalized exercise recommendations. These systems extract temporal dependencies from physiological signals like heart rate and activity duration, while preserving user privacy through federated learning and encrypted communication. However, their focus largely remains on recommendation accuracy rather than contextual integration. The present project enhances this paradigm by incorporating geolocation, environmental, and biometric data streams into a single inference pipeline, enabling the analysis of stress and wellness dynamics beyond exercise routines.

4) Figma

Fig. 11: figma

Figma is a collaborative cloud-based design platform optimized for UI/UX design, prototyping, and team collaboration. In this project, Figma was used to design and prototype both the iPhone and Apple Watch interfaces with pixel-perfect precision. Figma's component system and auto-layout features enabled consistent design patterns across different screen sizes, while real-time collaboration capabilities allowed seamless feedback exchange between designers and developers throughout the design process. The combination of Figma's prototyping tools and developer handoff features improved design-to-code accuracy and reduced implementation time, allowing faster validation of user interactions and visual refinements during the development cycle.

5) Visual Studio Code

Fig. 12: Visual Studio Code

Visual Studio Code is a lightweight source code editor optimized for cross-platform development, extension customization, and multi-language support. In this project, VSCode was used for editing configuration files, writing documentation in Markdown, and managing Git operations with an intuitive interface. VSCode's extensive extension marketplace and integrated terminal enabled streamlined workflows for linting, formatting, and version control without switching contexts, while workspace settings allowed consistent coding standards across team members. The combination of VSCode's IntelliSense and multi-cursor editing improved productivity for repetitive tasks and code navigation, allowing efficient handling of non-Swift files and supporting scripts throughout the development cycle.

6) Xcode

Fig. 13: Xcode

Xcode is Apple's integrated development environment optimized for building, testing, and debugging iOS, watchOS, and macOS applications. In this project, Xcode was used as the primary IDE for Swift development, Interface Builder integration, and device-specific testing on iPhone and Apple Watch simulators. Xcode's Instruments profiling suite and live preview functionality enabled real-time performance monitoring and UI debugging without repeated builds, while built-in HealthKit and CoreLocation frameworks simplified integration with native sensor APIs. The combination of Xcode's code completion and refactoring tools improved development speed and code consistency, allowing rapid prototyping with immediate visual feedback throughout the development cycle.

7) Github

Fig. 14: Github

GitHub is a cloud-based version control platform optimized for collaborative software development, code review, and CI/CD integration. In this project, GitHub was used to manage source code repositories, facilitate pull request reviews, and automate testing pipelines for Swift codebases. GitHub's branching strategy and merge conflict resolution enabled parallel feature development without disrupting the main codebase, while Actions workflows automated build verification and deployment processes for both iOS and watchOS targets. The combination of GitHub's code review tools and issue linking improved code quality and traceability, allowing rapid iteration with confidence and maintaining a clean commit history throughout the development cycle.

8) Jira

Fig. 15: Jira

Jira is a project management platform optimized for agile development, issue tracking, and sprint planning. In this project, Jira was used to manage user stories, track bugs, and coordinate development tasks across iOS and watchOS implementations. Jira's customizable workflow and sprint

board visualization enabled clear prioritization of features and real-time progress monitoring, while integration with GitHub allowed automatic status updates based on pull request activities. The combination of Jira's reporting dashboards and backlog management improved team velocity and accountability, allowing data-driven sprint planning and faster identification of blockers throughout the development cycle.

9) Notion

Fig. 16: Notion

Notion is an all-in-one workspace platform optimized for documentation, knowledge management, and team collaboration. In this project, Notion was used to centralize project documentation, meeting notes, and design specifications in a single accessible repository. Notion's flexible database system and nested page structure enabled organized tracking of feature requirements and research findings, while cross-linking capabilities allowed seamless navigation between related documents and resources. The combination of Notion's collaborative editing and commenting features improved knowledge sharing and reduced information silos, allowing team members to stay aligned on project goals and technical decisions throughout the development cycle.

10) OpenAI

Fig. 17: OpenAI

OpenAI's GPT-5 Nano API is a lightweight language model optimized for efficient natural language processing with minimal latency and resource consumption. In this project, the GPT-5 Nano API was integrated to provide intelligent health insights and personalized workout recommendations based on user activity data. The API's low-latency response time and compact model size enabled real-time text generation on-device without compromising battery life, while fine-tuning capabilities allowed customization for health and fitness domain-specific responses. The combination of GPT-4 Nano's natural language understanding and cost-effective pricing improved user engagement with contextual suggestions, allowing scalable AI-powered features without significant infrastructure overhead throughout the deployment cycle.

11) Overleaf

Fig. 18: Overleaf

Overleaf is a collaborative cloud-based LaTeX editor optimized for academic writing, technical documentation, and publication-ready typesetting. In this project, Overleaf was used to author the research paper, format technical diagrams, and maintain consistent academic citation standards. Overleaf's real-time collaborative editing and version history enabled simultaneous contributions from multiple authors without file conflicts, while rich LaTeX template library and integrated compiler provided professional formatting for figures, equations, and bibliographies. The combination of Overleaf's track changes feature and comment system improved peer review efficiency and writing quality, allowing seamless revision cycles and faster preparation of camera-ready manuscripts throughout the publication process.

C. Task Distribution

1) Project Manager - Junho Uh

A Project Manager orchestrates the development and delivery of the smart health-home integration system within specified constraints of scope, time, and budget. Their role involves developing comprehensive project plans, establishing critical milestones, and implementing Agile methodologies to ensure efficient delivery. They coordinate resource allocation and timeline management using Jira for issue tracking, Notion for documentation, and GitHub for version control.

Critical responsibilities include managing deliverables across iOS/watchOS health monitoring, backend infrastructure, LLM-based home automation logic, and smart appliance integration. They ensure seamless coordination between health data collection, AI-driven state prediction, and home environment optimization through connected devices. Their success is measured through project completion metrics, team performance, and the system's ability to accurately predict user states and automatically adjust home conditions for optimal comfort and wellness.

2) Frontend Developer - Donghyun Lim

Frontend developers utilize Swift 5.10 and SwiftUI to create iOS and watchOS applications that bridge health monitoring with smart home control. They design interfaces for real-time health data visualization (heart rate, HRV, sleep patterns) alongside home appliance status and control panels. The development integrates HealthKit for biometric data collection, CoreLocation for spatial context, and custom APIs for communicating with smart home devices.

They implement the WCSession framework for seamless data synchronization between Apple Watch and iPhone, ensuring that health metrics captured on the wearable trigger appropriate home automation responses. The role requires expertise in creating intuitive control interfaces where users can monitor their predicted wellness state and manually override or customize automated home adjustments. They work closely with UI-UX designers to ensure the health-to-home automation workflow is accessible and engaging, while coordinating with backend developers to implement real-time bidirectional communication between health sensors, prediction models, and connected appliances.

3) Backend Developer - Dogyeom Kim, Junho Uh

Backend developers design the database schema and API architecture that connects health data collection, AI prediction models, LLM-based decision making, and smart home device control. They manage a MySQL database analyzing relationships among user health states, environmental conditions, location contexts, and home appliance configurations. Using FastAPI, they create RESTful endpoints that receive health samples and GPS data from iOS/watchOS applications, process them through prediction models, and generate LLM prompts that determine optimal home states.

They integrate OpenAI's GPT API to translate predicted user conditions (stress levels, fatigue, sleep readiness) into natural language commands for controlling temperature, lighting, humidity, and other smart appliances. The backend utilizes Docker for containerization and Google Cloud Platform for scalable deployment, while implementing secure protocols for IoT device communication. They design the feature-engineering pipeline that aggregates time-series health data with environmental factors to feed both prediction models and LLM context, ensuring the system maintains data consistency throughout the health-to-home automation workflow.

4) UI-UX Designer - Donghyun Lim, Dogyeom Kim

UI-UX designers, using Figma, determine how the integrated health monitoring and smart home control interface is presented across iPhone and Apple Watch platforms. They create a unified design system that seamlessly blends health data visualization with home automation controls, ensuring users can easily understand their current wellness state and corresponding home environment adjustments.

The design process includes creating mockups that display predicted user conditions alongside automated appliance responses, with intuitive override controls and customization options. Designers must balance information density between health metrics (heart rate, sleep quality, stress indicators) and home status displays (temperature, lighting, air quality) within the limited screen space of wearable devices. They leverage Figma's component system to maintain consistent design patterns across health monitoring dashboards and appliance control interfaces. Once designs are finalized, they communicate specifications to frontend developers through Figma's developer handoff features, ensuring pixel-perfect implementation of the health-to-home automation user experience.

5) AI Developer - Yeonseong Shin, Junho Uh

AI developers build machine learning pipelines and LLM integration systems that predict user wellness states and generate optimal home automation commands. They collect and preprocess multi-modal data from HealthKit (heart rate, HRV, sleep, activity), CoreLocation (GPS, location patterns), and environmental sensors to extract features for stress prediction and fatigue detection models. The AI workflow implements machine learning algorithms that analyze correlations between physiological patterns and contextual factors, then feeds these predictions into OpenAI's GPT API to generate natural language commands for smart home devices.

They design the integration pipeline where predicted states (e.g., "user is stressed and approaching home") trigger LLM prompts that determine appropriate environmental adjustments (dimmed lighting, cooler temperature, calming music). AI developers train and evaluate models using metrics such as prediction accuracy and user comfort scores, while optimizing the LLM prompt engineering to ensure reliable and contextually appropriate appliance control. They ensure the GPT API's responses are parsed correctly and translated into specific device commands, maintaining low latency between state prediction and home automation execution to provide seamless, anticipatory environmental optimization based on real-time health intelligence.

IV. Specification

A. Backend Data Management

1) User and Device Information Management

The backend data system functions as an integrated database schema that analyzes the relationships among users' health states, weather conditions, and spatial contexts. Data are categorized into four domains—user, health, location, and weather—each interconnected through a unified identifier hierarchy.

User and device information constitute the fundamental entities of the service, serving as the top-level reference keys for all subsequent data. The User table stores each user's primary information, while a user may register multiple devices through the Device table. Each device record contains model name, platform type (e.g., iOS watch, iOS phone), and registration time, and links to the DeviceModel table to trace hardware and OS-level capabilities. The Consent table records the scope of user consent for health, location, and weather data collection, along with the timestamps of consent and revocation (granted_at, revoked_at), thereby ensuring ethical and legal transparency in data access.

2) Health Data Schema and Time-Series Management

Health-related data are stored in a per-user time-series structure, designed to capture continuous physiological measurements. The HealthSample table stores quantitative biosignals such as heart rate, heart rate variability(HRV), blood oxygen saturation, and respiratory rate. Each record is uniquely identified by a composite key of user_id, type_code, observed_at, and device_id, forming a natural unique constraint to prevent redundant uploads from the same device and timestamp.

To analyze long-term physiological patterns, dedicated tables for sleep and workout sessions are defined. The SleepSession table records the start and end time of each sleep episode, durations of deep, REM, and core sleep, and overall sleep efficiency. The WorkoutSession table manages high-intensity exercise sessions, logging activity type (running, cycling, strength, etc.), average heart rate, and active energy expenditure. In addition, the ActivitySummary table aggregates daily activity features—such as step count, exercise minutes, active calories, and standing hours—which later serve as inputs for the DailyFeature table used in model training.

This hierarchical structure is compatible with Apple's HealthKit data model and is automatically normalized and stored via the FastAPI-based backend service.

3) Environmental Context Integration

To represent users' physical and environmental contexts, the system integrates location and weather information in a unified schema. The LocationFix table stores raw GPS samples collected at regular intervals, including latitude, longitude, altitude, speed, true heading, and both horizontal and vertical accuracy values. These raw samples are clustered to identify frequently visited places (Place), each defined by its center coordinates, radius, confidence level, and visit count. Each place is also mapped to the 5-km grid system (nx, ny) provided by the Korea Meteorological Administration(KMA) for efficient weather retrieval. Address and category data (e.g., home, office, gym) are stored in a features JSON field obtained through Apple Maps reverse geocoding, enabling higher-level behavioral analysis.

Weather data are organized into WeatherTile and WeatherObservation tables. The WeatherTile table defines regional grid cells based on KMA's standardized nx, ny coordinates, while WeatherObservation stores hourly or 10-minute-interval meteorological measurements. Observed attributes include temperature, apparent temperature, humidity, wind speed, cloud cover, precipitation type (none/rain/snow/sleet), precipitation probability, and air-quality indicators. These data are periodically updated through KMA's ultra-short-term forecast and observation APIs and are linked to each Place via the grid_nx and grid_ny attributes for spatiotemporal environmental analysis.

All tables employ UUID (CHAR(36)) primary keys, and major tables include a foreign key (user_id) to ensure data ownership consistency. Cascade deletion rules (ON DELETE CASCADE) guarantee referential integrity when a user or parent record is removed. High-frequency time-series tables are indexed by (user_id, observed_at DESC) to enable efficient temporal queries, while (tile_id, as_of) indexing optimizes spatiotemporal weather lookups.

This ERD-based schema enables seamless integration of user health, location, and environmental data under a common key structure. By aligning all domains through the unified user_id and temporal fields (observed_at, as_of, recorded_at), the system ensures data consistency throughout the feature-engineering and stress-prediction pipelines. Furthermore, linking KMA's grid-based tile system directly to user places allows efficient three-

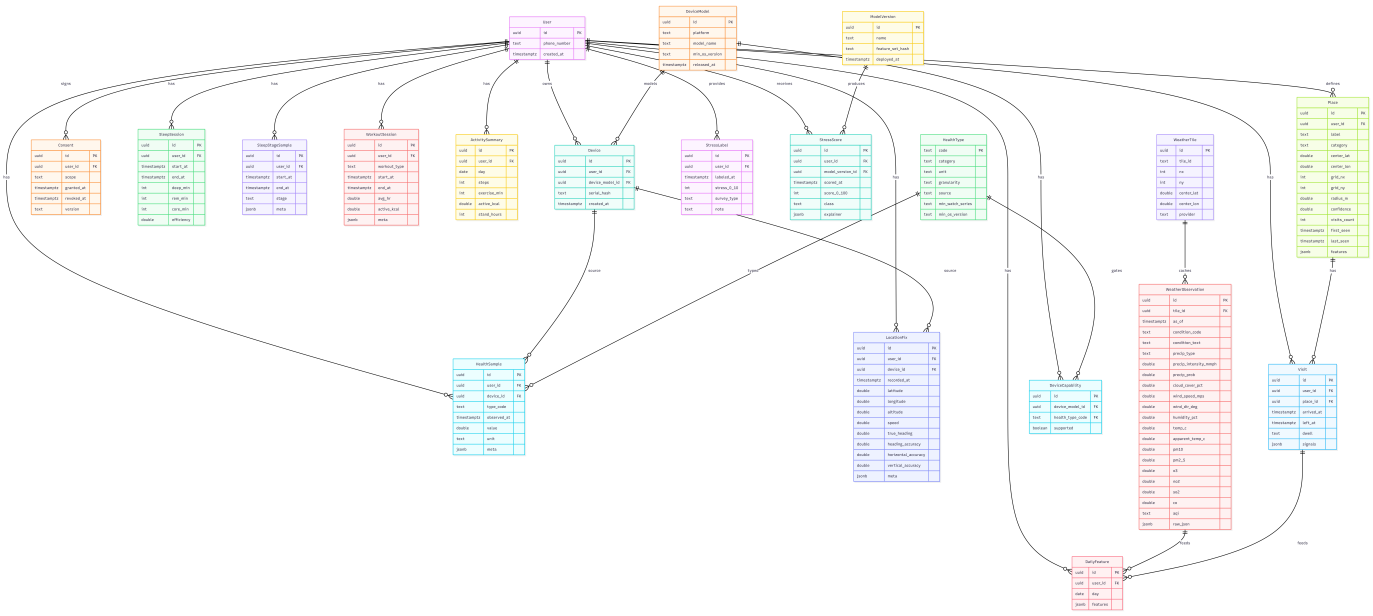

Fig. 19: Entity–Relationship Diagram of the backend database schema

way integration among weather, location, and health data. This architecture maintains high extensibility for incorporating additional environmental factors (e.g., air quality, ambient noise, indoor conditions), and serves as the core backend infrastructure connecting user health, contextual environment, and AI-driven inference in a unified pipeline.

B. Apple Watch Integration and Frontend Architecture

1) Health and Sensor Data Acquisition

The mobile application interfaces with Apple Watch via the HealthKit and CoreLocation frameworks to collect continuous biometric and spatial data in real time. Through the HealthKit API, the system retrieves categorized health data across multiple domains — including heart rate, heart rate variability (HRV), oxygen saturation, respiratory rate, and sleep analysis. Each data type is defined under Apple's standardized schema (`HKQuantityType`, `HKCategoryType`, and `HKWorkoutType`), ensuring compatibility with the user's privacy settings and consent scope.

Collected metrics are automatically normalized into time-series format before being transmitted to the backend pipeline. This structure allows the system to infer physical conditions such as fatigue, stress, and recovery status. To preserve transparency and user control, data synchronization strictly follows HealthKit's permission model, where the user explicitly authorizes or revokes each category (e.g., heart rate, step count, sleep). The synchronized data are later used to generate contextual insights, such as detecting elevated stress from HRV or reduced activity from step count trends.

2) GPS-Based Context Recognition

The CoreLocation module continuously monitors the user's GPS position to determine spatial context. Raw data points including latitude, longitude, altitude, speed, and heading accuracy — are captured at configurable accuracy levels (`kCLLocationAccuracyBestForNavigation`, `kCLLocationAccuracyNearestTenMeters`, etc.), balancing precision with energy efficiency.

Using these coordinates, the application determines whether the user is near home, commuting, or in a specific habitual zone. This context awareness enables anticipatory automation; for example, preparing home device states when the user is approaching or logging environmental conditions along outdoor routes. Each location update is timestamped and locally cached until successful transmission to the backend.

3) Map Visualization and Routine Pattern Analysis

The MapKit framework is utilized to visualize spatial trajectories and reconstruct the user's daily movement timeline. GPS samples are rendered as polylines on a map interface, where clusters of frequent stops are highlighted as potential key places (home, workplace, gym, etc.). By aggregating temporal and spatial data, the application generates an interactive timeline that represents not only the user's movement patterns but also correlates them with physiological data such as heart rate and activity levels.

This visual feedback helps users interpret lifestyle trends — for instance, linking increased stress levels with prolonged commuting or identifying sedentary periods during the day. Map-based insights are presented in both

static (daily summary) and dynamic (real-time tracking) modes, with user control over privacy and data visibility.

4) Integration Architecture and Data Flow

All frontend modules (HealthKit, GPS, and MapKit) are connected through a unified data pipeline within the iOS app architecture. The WatchKit extension communicates sensor updates via background tasks and delegates, which are received by the iOS host app using the `WCSession` framework. The host app then packages these data points into a consistent JSON schema before sending them to the backend through the FastAPI endpoint.

The entire flow ensures high modularity and real-time responsiveness:

1) Apple Watch collects health and motion data via sensors.
2) Data are serialized and transferred to the paired iPhone through `WCSession`.
3) The iPhone app integrates HealthKit and GPS samples, tagging them with timestamps and user identifiers.
4) Processed data are transmitted to the backend, where normalization and database insertion occur.

This modular integration enables continuous synchronization between the user's wearable, smartphone, and backend analytics system, forming the foundation of personalized health and behavior intelligence.

C. AI-Driven User State Prediction Model

1) Model Architecture and Design Principles

The user state prediction model employs a hybrid deep learning architecture that integrates time-series health data with contextual environmental information to forecast physiological and psychological conditions. The model outputs four primary state indicators: stress level (0-10 scale), fatigue level (0-10 scale), sleep readiness (0-1 probability), and comfort index (0-10 scale). These predictions serve as the foundation for generating LLM-based commands that automatically adjust smart home appliances to optimize the user's living environment.

The architecture combines Long Short-Term Memory (LSTM) networks for temporal pattern recognition in health metrics with dense neural layers for contextual feature encoding. An attention mechanism identifies critical time periods that significantly influence current user states, while a fusion layer integrates both data streams into a unified representation. Multiple output heads enable simultaneous prediction of different state dimensions, allowing the system to capture complex interdependencies between stress, fatigue, and environmental comfort.

2) Input Feature Engineering

The model processes two distinct feature categories: time-series health data and contextual environmental data. Health features include 24-hour rolling windows of heart rate statistics (mean, standard deviation, maximum), heart rate variability (HRV) as a primary stress indicator, activity metrics (step count, active energy expenditure, exercise duration), and sleep quality measures aggregated over the past seven days (sleep duration, efficiency, deep sleep ratio, REM sleep ratio). Additional physiological signals such as respiratory rate and blood oxygen saturation supplement the primary health indicators.

Contextual features capture temporal, spatial, and environmental dimensions that influence user states. Temporal features include hour of day, day of week, and cyclic encodings using sine and cosine transformations to represent daily and weekly patterns. Spatial context is derived from GPS clustering that identifies location types (home, office, commute, other) and calculates time spent at each location. Weather data retrieved from the Korea Meteorological Administration API provides temperature, humidity, precipitation, and air quality indices (PM10, PM2.5) that correlate with physiological comfort and stress responses. Current smart home device states, including indoor temperature, humidity, and lighting levels, are also incorporated to model feedback loops between environmental conditions and user comfort.

3) Real-Time Inference and LLM Integration

The pipeline executes continuously with configurable prediction intervals, typically ranging from 15 to 60 minutes depending on computational constraints and user preferences. When triggered, the feature engineering module queries the MySQL database to extract recent health samples, GPS locations, and weather observations, constructing the input tensors required by the neural network. The trained model performs forward propagation in under 100 milliseconds on the server, producing state predictions that are immediately formatted into a structured natural language prompt for the OpenAI API.

The LLM prompt includes the predicted stress level, fatigue level, sleep readiness, and current contextual information such as time of day, location type, outdoor weather conditions, and current indoor environment settings. The prompt instructs the agent to generate a JSON-formatted response containing recommended adjustments for temperature, lighting intensity, humidity control, and any additional appliance actions. The LLM's reasoning capability enables contextually appropriate decisions. The parsed JSON commands are then transmitted to the smart home controller for execution, with confirmation feedback sent to the iOS interface for user transparency and manual override capability.