ALIVE

An AI-Driven Multi-Agent Chat System for Intuitive Smart Home Management

Jun Seong Pyo
College of Engineering
Hanyang University
Dept.of Information Systems
Seoul, Korea
standardstar@hanyang.ac.kr

Byeong Hyun Yang
College of Engineering
Hanyang University
Dept.of Information Systems
Seoul, Korea
zxvm5962@hanyang.ac.kr

Dong Hun Kang
College of Engineering
Hanyang University
Dept.of Information Systems
Seoul, Korea
kdu3840@hanyang.ac.kr

Hye Jin Bae
College of Engineering
Hanyang University
Dept.of Information Systems
Seoul, Korea
cats5565@hanyang.ac.kr

Abstract—Smart home provides automated control and convenience in home environments. Although smart homes have improved convenience in daily life, their rigid and uniform structures limit customization and integration of diverse information sources, leading to passive and inflexible management systems. We propose a new multi-turn AI based Chat Room system, Alive, which enables users to manage and control home appliances. With Alive, users can register home appliances directly or import them from compatible apps, allowing them to create custom chat rooms for seamless device management. Through our appliance chat rooms, users can identify the most suitable appliances for specific situations and execute them simultaneously, simplifying the process of smart home control. Users can efficiently manage and control their appliances at any time through the chat interface. Furthermore, we've integrated a human feedback mechanism to improve the system. Users can evaluate their experience after the end of process, providing valuable insights for continuous enhancement of the application. This data is systematically stored and utilized for learning and refining the system's performance. Alive represents a significant step forward in smart home technology, addressing the need for more flexible, user-centric solutions in the evolving landscape of home automation.

TABLE I: Role Assignments

Roles	Name	Task description and etc.
User, Customer, Development manager	Pyo Jun Seong	Users/customers consider what features should be added from the perspective of users or customers. They think about what needs exist and what solutions could address these needs. The Development Manager takes charge of the project's comprehensive elements, including timeline creation, strategic planning, and maintaining the quality standards of products and services. Furthermore, they ensure a thorough understanding of user specifications and oversee the complete software engineering lifecycle, encompassing phases such as design, implementation, and quality assurance testing.
AI Developer	Yang Byeong Hyun	AI developers create customized programs tailored to business needs based on collected and analyzed data. In this service, they build AI systems that present various scenarios using users' historical data and user requirements. Specifically, they apply multi-agent technology to provide accurate and personalized responses to user requirements. AI developers are responsible for the design, development, implementation, and monitoring of the entire AI system, focusing on building efficient data collection and transformation architectures.
Software Developer (Back-end)	Kang Dong hun	Software developers (backend) design and manage server-side infrastructure and database systems required for project development. This includes building and maintaining robust backend solutions that support the capabilities of web and mobile applications, and ensuring seamless integration with front-end components. This includes database operations, efficient query generation, and API endpoint implementation. It also involves leveraging Java, Kotlin, and Spring programming languages to develop scalable and maintainable systems that meet the requirements of the project.
Software Developer (Front-end)	Bae Hye jin	Software Developers (Front-end) are responsible for designing and implementing the overall UI/UX of applications to create exceptional user experiences. They craft visual layouts and interactive elements, ensuring interfaces are aesthetically pleasing, intuitive, and user-friendly. Using tools like Figma, they create and refine designs, which they then implement using technologies such as React-Native, TypeScript, and CSS. The role involves creating responsive interfaces, optimizing performance, and ensuring accessibility throughout the development process.

I. INTRODUCTION

A. Motivation

1) The Advancement of AI Technology and the Rise of Multi-Agent Systems

Artificial intelligence technology has been developing at an astonishing rate over the past few years. In 2016, Google DeepMind's AlphaGo defeating the world's top Go player marked the beginning of the AI era. Since then, AI technology has become even more sophisticated, with the release of OpenAI's ChatGPT in late 2022 ushering in the era of generative AI. ChatGPT reached 1 million users within a week of its launch and saw explosive growth, surpassing 100 million active users in just two months.

In 2023, GPT-4 was released, showcasing even more advanced features. GPT-4 evolved into a multimodal model that can simultaneously process various input data such as text, images, audio, and video, utilizing a dataset about 500 times larger than its previous model. These advancements reflect a broader trend in AI development, where the technology is moving beyond simple content generation to more complex, interactive, and adaptive capabilities.

The future of the AI industry points to evolve from generative models to autonomous agents, ushering in an era of multi-agents where multiple agents collaborate, think, and learn independently. This evolution will likely lead to the integration of multi-agent systems within the Artificial Intelligence of Things (AIoT), which combines AI and the Internet of Things (IoT), and is expected to spread to various industries, centered around manufacturing.

2) IoT Technology Market Trends and the Current Status of Smart Homes

The spread of IoT technology is dramatically changing all aspects of daily life, with 'smart homes' being a prime example. In smart homes, home appliances and various devices are interconnected via the internet, allowing users to control them remotely and automate daily tasks. The smart appliance market continues to grow rapidly, fueled by increasing consumer interest in convenience, energy efficiency, and personalized living environments.

Despite widespread awareness and adoption of smart home devices, many users are not fully utilizing their potential. Over Half of users (53.1%) still rely on basic functions like power control and status monitoring, while advanced capabilities such as automation or device interlinking remain underutilized. This indicates a gap between the technology's capabilities and how users engage with it, suggesting a need for more intuitive interfaces and seamless integration to encourage deeper interaction.

As smart home technology continues to evolve, there is a growing opportunity to move beyond simple control features and offer more intelligent, adaptive experiences that anticipate users' needs. The next step for smart homes involves leveraging AI to enable dynamic collaboration among devices, creating environments that actively respond to user behavior and preferences.

Our team's goal is to push the boundaries of smart home technology by advancing from passive AI systems to dynamic and intelligent interactions. We aim to deliver optimal results for users by implementing the ability for smart devices to communicate with each other using Large Language Models (LLMs). Through this solution, we aim to address the current gap between the potential of smart home technology and its real-world use to increase access and smoothness to advanced automation, ultimately improving the daily life experience of individual users.

B. Problem Statement

a. Need for improved Customization and Differentiated User Experience

According to the Smart Home Trend Report 2023, major dissatisfaction factors for smart home appliances include 'lack of additional functions when using devices in conjunction'. The response 'lack of additional functions when using devices in conjunction' suggests that users are not fully utilizing the functions of smart home appliances. In fact, while smart appliances offer various functions through applications, customer satisfaction is declining as users fail to effectively utilize these functions. This indicates the need for improved Customization and User Experience so that users can effectively use them.

b. Limited and Uniform Scenarios

While the current routine functions of smart appliances are useful for automating users' daily patterns, they have limitations in responding to complex and diverse real-life scenarios. For example, in situations with many variables such as sudden weather changes, pre-set routines alone are insufficient for appropriate responses. Moreover, it's difficult to flexibly handle complex interactions between various appliances or immediate changes in user demands. Therefore, for a truly smart home solution, more advanced technologies such as real-time situation awareness and machine learning are needed. This requires the development of intelligent systems that can understand users' living patterns more deeply and respond according to the situation.

c. Limitations of Passive and Individual Management

Despite being labeled as 'smart', current smart home

appliances still heavily rely on passive and individual management by users. In most cases, users need to adjust settings for each device separately and make judgments and decisions for various situations directly. For example, user intervention is required in many aspects such as adjusting the air conditioner's temperature, selecting the washing machine's cycle, or setting the robot vacuum's cleaning schedule. Furthermore, even in routine settings, users have to perform logical structure design themselves, which can lead to logical errors. This indicates that the focus is on the independent operation of each device rather than on device interconnection and overall home environment optimization.

d. Lack of Information Integration

Smart home applications individually offer rich features and information. Each app provides useful data such as energy usage monitoring, device status checks, and even general lifestyle information. However, these diverse information and functions are not connected into a single integrated system, which limits the user experience. As a result, users must open each section separately to check information and make their own judgments to decide on their next actions based on this information. This hinders the convenience and efficiency of smart homes, causing users the inconvenience of having to manually integrate and analyze multiple pieces of information.

C. Solution

a. Providing Customized Services and improving user Experience

It is necessary to improve the user experience by providing customized services that meet user needs. This can be achieved by using AI to provide user-customized interfaces and improve user experience through more user interactions.

Specifically, in this system, when the user simply instructs a task via chat or voice, AI coordinates the 'conversation' between home appliances to derive the optimal execution plan. It obtains user confirmation before execution, suggests alternatives based on changing situations, and learns user preferences. It also collects feedback after task completion for continuous improvement.

This approach goes beyond simply controlling devices, providing a true smart home experience that deeply understands and reflects the user's intentions and values. As a result, users can enjoy a home environment perfectly tailored to their lifestyle and preferences without complex settings, making the interaction between technology and humans more natural and efficient.

b. AI-based Integrated Control and Scenario Optimization

LLM (Large Language Model) technology can generate and analyze various complex scenarios. Based on tasks instructed by users via chat or voice, LLM can establish comprehensive preparation plans.

For example, with just the simple information that "friends are coming over tonight," LLM can establish a comprehensive preparation plan including lighting settings, indoor temperature adjustment, music playback, and activating cleaning robots. In this process, LLM suggests optimal solutions considering the user's past preferences, current situation, and energy efficiency.

If changes are needed during execution, LLM immediately generates alternatives and provides options to the user. Additionally, by analyzing feedback collected after task completion, it can provide better solutions in similar situations in the future. In this way, LLM acts as an intelligent mediator between users and appliances, contributing to more accurately understanding and realizing the user's intentions.

c. Integrated Control Between Devices and Adaptation to Changing Environments

The optimization of environment-adaptive scenarios through integrated control systems and collaboration between products greatly enhances the efficiency and user experience of smart homes. Each device performs status checks and transmissions, and the central control system monitors this and comprehensively analyzes data collected from each device.

Based on this, dynamic adjustments between devices are made according to changing environmental conditions (e.g., weather changes, indoor air quality, user activity patterns). For example, when the temperature rises, the air conditioner and blinds can work together to efficiently maintain indoor temperature, or the air purifier and ventilation system can be linked to create an optimal indoor air environment.

This system maintains the optimal living environment continuously with minimal user intervention and can flexibly respond to unpredictable situations, thus implementing a smart home in the true sense of the term. Furthermore, by entrusting logical structure design to AI, it can solve the problem of logical errors that may occur when users set it directly.

d. Integration of IoT Technology and Information

Integrating and utilizing the diverse information and functions provided by IoT devices greatly improves the efficiency of smart home systems. Current smart home appliances, i.e., IoT devices, offer a wide variety of convenient functions, but users are not fully utilizing them.

To solve this problem, we propose managing various lifestyle information such as energy consumption, indoor

environmental data, user schedules, and weather information provided by apps comprehensively on a single platform. This integrated information is used for smarter decision-making by home appliances.

For example, a washing machine can analyze the user's schedule and power usage patterns to suggest optimal operating times. This integrated approach enables complex scenario settings that optimize the entire home environment, going beyond simple automation of individual devices.

As a result, users can enjoy a more convenient and energy-efficient smart home experience. Additionally, it allows users to easily obtain information and maximize the interaction effect with IoT devices and, by extension, with the smart home.

D. Research on Related Software

1. LG ThinQ ON
   LG ThinQ ON is an AI home hub equipped with Furon, which integrates various large language models (LLM) into the LG ThinQ platform. This system continuously monitors the home environment and appliances, engaging in conversations with users to assess situations and optimally control devices. Unlike traditional voice recognition speakers that provide simple responses and execute predefined commands, LG ThinQ ON offers a more interactive and intelligent experience.

2. SmartThings
   SmartThings is Samsung's integrated smart home platform that allows users to easily control and manage various IoT devices through a mobile app. This platform include SmartThings Energy, an AI-based automated energy management service that optimizes energy usage through the "Energy AI Save Mode," and SmartThings Air Care, which controls air flow recommends appropriate air purification methods.

3. Amazon Alexa Hunches
   Amazon Alexa Hunches is a feature that enables Alexa to learn user patterns related to smart home devices. It proactively suggest actions or automatically perform tasks based on these behaviors. Additionally, users can configure Alexa to execute actions automatically with their consent. Hunches also helps optimize energy consumption by monitoring the status of devices such as lights, thermostats, and plugs, thereby assisting in energy savings.

4. Vivint Smart Home
   Vivint Smart Home is a home automation and security platform that provides comprehensive smart home solutions, including security cameras, smart locks, lighting, and energy management systems. It integrates various smart devices into a single system, enabling users to control their home environment via the Vivint app or voice commands through smart assistants. Vivint's security-focused features, such as professional monitoring and customizable alerts, ensure safety, while its smart home automation enhances convenience and energy efficiency for users.

5. Apple HomeKit
   Apple HomeKit is a software framework that allows users to configure, communicate with, and control smart appliances through Apple devices such as iPhones and macOS computers. It supports both Apple's own products and those compatible with Matter, enabling users to register devices using their setup codes. HomeKit facilitates the management of these devices, allowing for control and the registration of detailed routines.

6. Google Nest Hub
   Smart display developed by Google, this provides a range of smart home features based on Google Assistant. It extends the voice-centric capabilities of Google Home by adding conversational interactions and learning user behavior patterns to better understand and anticipate their needs. In addition, its display-based interface offers visual feedback, enhancing user interaction with smart home devices and delivering a more intuitive and engaging experience.

7. Azure IoT Hub
   Azure IoT Hub is a versatile and scalable cloud platform (IoT PaaS) that caters to multiple tenants. It comprises an IoT device registry, data storage, and robust security features. It also offers a service interface to facilitate IoT application development.

8. ChatThinQ
   ChatThinQ is an AI feature within LG's ThinQ platform that enhances smart home interactions. By using large language models, it supports natural, context-aware conversations for controlling appliances, offering personalized recommendations, and providing status updates. It can proactively suggest actions, optimize device settings, and create custom routines based on user preferences. The multi-turn dialogue capability ensures fluid interactions, making the smart home experience more intuitive and engaging.

II. REQUIREMENT

A. Sign up
ALIVE needs five types of information to sign up. These are phone numbers, passwords, name, email, and birth dates.

1) Enter phone numbers
   The phone number must be entered, and the phone number is verified through the carrier's authentication system to confirm whether the phone number is valid

for membership registration. The phone number serves as an ID in the subsequent login process.

2) Enter passwords
Passwords must be entered and must be at least 8 characters long in a combination of 3 or more of English uppercase, English lowercase, numbers, or special characters. When the user enters the desired password, it is displayed in the form of '****' on the screen, with each condition changing color to green when it is satisfied, and red when it is not.

3) Enter a name
The name must be entered, and it is subsequently set as the default nickname at the first login. The name is also used in ID search.

4) Enter birth dates
The birth date must be entered, and a pop-up window is displayed every year to celebrate the user's birthday. The date of birth is also used in ID search.

B. Sign in
There are two types of logins: 1) Local logins through ALIVE membership, 2) SNS logins through SNS linkage.

1) Local logins through ALIVE membership
   a) The system checks whether the ID and password entered by the user have been filled.
   b) When the ID and password input by the user exist in the member database, the user succeeds in logging in. After that, it moves to the main page.
   c) If the phone number and password entered by the user do not exist in the member database, the user fails to log in and a "Non-existent member" message is displayed in the pop-up window.

C. Register home appliances You can register your appliances in ALIVE in two ways: 1) Register your appliances directly, 2) Import your registered appliances from the LG ThinQ app.

1) Register your appliances directly
   a) via QR code: When the camera access permission request screen appears, simply follow the instructions to grant access. After that, locate the QR code on each product. Scan the QR code along the guide lines on the edge.
   b) find appliances on your own: you can search for appliances by two methods:
      • Wi-Fi or Bluetooth
      • Product name and serial number

2) Import from LG ThinQ
You can import a list of appliances pre-registered in the LG ThinQ app. After clicking the 'Import from LG ThinQ' button, the LG ThinQ app will open. The user must agree to the following:
   • Agree to import the information of the registered home appliances into the app.
   • Agree to import the settings from the ThinQ app into the app.

If information is imported from the LG ThinQ app, it may include the following:
   • Registered home appliances
   • Room list and list of appliances added to the room
   • Smart routines

After all the settings are imported, the following chat rooms will be opened automatically:
   • A chat room with all your appliances
   • A room-specific chat room

D. Creation of Appliance Chat Rooms
1) Integration with LG ThinQ App
Enable users to retrieve all connected electronic appliances by integrating with the LG ThinQ app. Ensure seamless synchronization so that any changes in connected devices are reflected in the chat application.

2) Automatic Generation of Default Chat Room
Automatically generate default chat rooms during the initial setup, including all connected appliances and grouping them by room. This provides users with immediate access to control all their devices without additional configuration.

3) Custom Chat Room Creation
Allow users to create new chat rooms through the "+" button at the top right corner of the screen, and invite only the appliances they wish to include.

4) Support for Multiple Chat Rooms
Permit the creation of multiple chat rooms to manage appliances based on rooms, functions, or user preferences.

5) Dynamic Management of Appliances
Enable users to later invite additional appliances or remove existing ones. Provide easy-to-use interfaces for managing device participation within chat rooms.

By incorporating these features, users can efficiently manage their connected appliances through customizable chat rooms. This approach enhances usability and provides personalized control over their smart home environment, allowing for a more intuitive and flexible user experience.

III. DEVELOPMENT ENVIRONMENT

A. Choice of software development platform
1 Development Platform
   1) Windows
   Windows provides a wide range of development tools and integrated development environments (IDEs) for creating various types of applications, including web applications, desktop applications, mobile apps, and games. This supports effective code editing, debugging, testing, deployment, and collaboration, ultimately enhancing developers'

productivity. Furthermore, Windows supports multiple programming languages and frameworks, allowing developers to choose their preferred languages and technologies to flexibly meet project requirements. Windows offers a user-friendly and intuitive interface, making it easy for developers to configure and manage their development environments. A robust community and support system enable developers to share experiences and receive assistance. Lastly, Windows continuously updates and improves, ensuring access to the latest technologies and tools, empowering developers to stay current and modernize their applications. Windows is recognized as a versatile platform suitable for various software development fields, playing a crucial role in turning developers' ideas into reality.

2) macOS

macOS is a highly regarded operating system in the field of software development, known for its user-friendly interface and exceptional versatility. This operating system offers several advantages to developers, and let's explore some of them. Firstly, macOS provides essential development tools and an integrated development environment (IDE) for creating a wide range of applications, including web applications, desktop applications, mobile apps, and games. Official IDEs like Xcode are powerful tools for application development across various platforms such as macOS, iOS, watchOS, and tvOS. They support tasks like code writing, debugging, testing, deployment, and collaboration, significantly enhancing developer productivity. Additionally, macOS supports a variety of programming languages and frameworks, allowing developers to choose their preferred languages and technologies, making it flexible to adapt to project requirements. macOS offers an intuitive and user-friendly interface that simplifies development environment setup and project management. The active macOS developer community provides a platform for sharing experiences and collaboration among developers. Finally, macOS ensures access to the latest technologies and tools through continuous updates and improvements. Apple's dedication to innovation provides developers with the necessaryfeatures to leverage the latest technologies and modernize their applications. For these reasons, macOS is recognized as an essential platform for software development, playing a significant role in turning ideas into reality.

2 Language / Framework

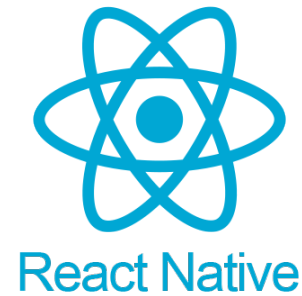1) Programming Languages

(1) React Native [2]

Fig. 1: React Native

React Native is a cross-platform framework developed by Facebook, enabling developers to build iOS and Android applications simultaneously using JavaScript. React Native ensures a consistent UI/UX across platforms while providing high compatibility with native modules, allowing optimal user experiences without compromising performance. Leveraging React's component-based architecture, it maximizes code reusability, which enhances project efficiency and manageability. Consequently, React Native supports fast release cycles and delivers high-quality mobile applications, making it a powerful solution for efficient mobile app development.

(2) TypeScript [3] TypeScript is a superset of

Fig. 2: Typescript

JavaScript developed by Microsoft, introducing a static type system that improves code stability and readability. TypeScript catches errors at compile time, reducing runtime issues and enhancing maintainability, especially for large-scale projects. Its rich type inference allows developers to clearly define code structure and intent, fostering better collaboration and improving code quality. Ultimately, TypeScript enables the writing of robust,

high-performance applications while retaining JavaScript's flexibility.

(3) Kotlin [4]

Fig. 3: Kotlin

Kotlin is a modern language developed by JetBrains and widely adopted as a Java alternative, particularly in Android development. With concise syntax and a strong type system, Kotlin simplifies code writing and maintenance, allowing developers to produce efficient, maintainable code. Its seamless Java interoperability supports integrating new code into legacy projects, improving productivity and minimizing runtime issues. Kotlin empowers Android developers to build efficient applications with better user experiences, making it an ideal choice for modern, high-quality programming.

2) Frameworks

(1) Spring Boot [5]

Fig. 4: Spring Boot

Spring Boot is a framework designed to simplify the development of Java-based web applications and microservices. This framework offers streamlined configuration, an embedded web server, automatic setup, starter dependencies, monitoring and management tools, robust microservices support, and integration with a vast ecosystem of libraries and tools. By using Spring Boot, developers can accelerate application development, simplify complex configurations, and boost overall productivity, making it an ideal choice for scalable and

maintainable applications.

(2) Hibernate [6]

Fig. 5: Hibernate

Hibernate is an open-source Object-Relational Mapping (ORM) framework for Java, facilitating seamless interaction between Java objects and relational databases. Hibernate offers database independence, automatic schema generation, an Object-Oriented Query Language (HQL), caching support, integrated transaction management, and a well-established community and ecosystem. In essence, Hibernate optimizes database operations in Java applications, providing flexibility, efficiency, and cross-database portability.

(3) FastApi [7]

Fig. 6: FastApi

FastAPI is a modern Python-based web framework designed for fast and efficient API development. FastAPI offers asynchronous support and data validation through Pydantic, maximizing developer productivity. Its automatic OpenAPI and JSON Schema documentation simplifies API testing and enhances collaboration. Renowned for its high performance, FastAPI is widely adopted across fields like machine learning model deployment and data analysis, making it a trusted choice for designing reliable APIs.
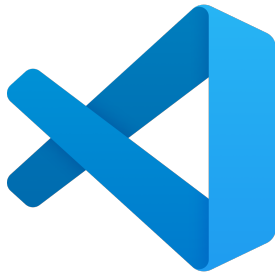
1 visual Studio Code

Fig. 7: Visual Studio Code

Visual Studio Code is a lightweight yet powerful source code editor developed by Microsoft, optimized for modern development workflows. It offers essential features including IntelliSense code completion, debugging support, and Git integration, while maintaining high performance. The editor's extensive marketplace provides diverse extensions supporting various programming languages and frameworks, particularly beneficial for React Native development. Its customizable interface and integrated terminal enhance development efficiency, making it an ideal choice for cross-platform mobile application development.

2 Xcode

Fig. 8: Xcode

Xcode is Apple's integrated development environment (IDE) essential for iOS app development and simulation. It provides comprehensive tools including iOS simulators, debugging capabilities, and Interface Builder for UI design. While primarily used for native iOS development, it serves as a crucial tool for React Native developers by offering high-fidelity iOS simulation and testing environments. The IDE integrates seamlessly with Apple's development ecosystem, providing essential features like device management, performance profiling, and automated testing capabilities for ensuring iOS app quality.

3 Github

Fig. 9: Github

GitHub is a cloud-based platform for version control and collaborative software development using Git. It provides essential features including repository hosting, branch management, pull requests, and issue tracking. The platform enables smooth team collaboration through code review tools, project management features, and continuous integration/deployment capabilities. With its extensive documentation support and robust security features, GitHub serves as a centralized hub for maintaining code quality and managing development
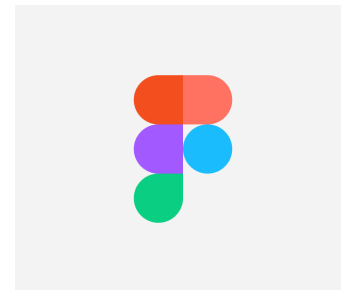
4 Figma

Fig. 10: Figma

Figma is a collaborative web-based interface design tool for creating user interfaces and prototypes. It offers powerful design features including component-based systems, auto-layout, and interactive prototyping capabilities. The platform allows real-time collaboration among designers and developers, streamlining the design-to-development workflow with efficient handoff tools and design system management. With

its cloud-based nature and extensive plugin ecosystem, Figma enhances team productivity and ensures consistent design implementation across projects.

5 Jira

Fig. 11: Jira

Jira is an agile project management tool developed by Atlassian that streamlines software development processes. It provides comprehensive features for issue tracking, sprint planning, and workflow customization. The platform offers robust reporting tools, integration capabilities with development tools, and flexible board views for different agile methodologies. With its detailed task management and progress tracking features, Jira enables teams to effectively monitor project progress and maintain development efficiency.

6 IntelliJ
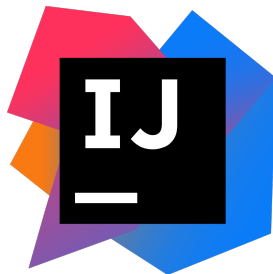
Fig. 12: IntelliJ

IntelliJ IDEA is a comprehensive Java-based integrated development environment (IDE) developed by JetBrains. It offers advanced code analysis, intelligent code completion, and powerful refactoring tools that enhance development productivity. The IDE provides robust debugging capabilities, seamless integration with various frameworks, and extensive plugin support for multiple programming languages. With its smart code navigation and efficient build tools,

IntelliJ IDEA accelerates development processes while maintaining code quality and consistency.

7 Notion

Fig. 13: Notion

Notion is a versatile collaborative workspace that unifies note-taking, task management, and documentation tools. The platform excels in organizing information through customizable databases, interconnected pages, and flexible templates. Its all-in-one approach combines wiki-style documentation, project tracking, and team coordination capabilities. With its adaptable interface and cross-platform accessibility, Notion simplifies knowledge sharing and enhances team communication across projects.

8 PostgreSQL

Fig. 14: PostgreSQL

PostgreSQL is an advanced open-source relational database management system known for its reliability and data integrity. The system supports complex queries, custom functions, and multi-version concurrency control for efficient data handling. Its architecture enables handling of diverse workloads, from single machines to distributed systems, while maintaining robust security features. With extensive support for SQL standards and scalable performance, PostgreSQL stands as a preferred choice for managing structured data in modern

applications.

9 Overleaf

Fig. 15: Overleaf

Overleaf is a web-based LaTeX editor that facilitates academic writing and document preparation. The platform combines real-time preview capabilities, extensive template libraries, and reference management tools for scholarly publications. Its browser-based environment enables collaborative writing among researchers, supporting version control and simultaneous editing features. With its integrated compilation engine and comprehensive documentation, Overleaf streamlines the creation of professional academic documents and research papers.

10 Postman

Fig. 16: Postman

Postman is a specialized API development and testing platform that simplifies the API lifecycle management. The tool offers intuitive request building, automated testing sequences, and detailed response validation capabilities. Its environment management system facilitates API testing across different configurations while supporting team collaboration through shared workspaces. With comprehensive documentation generation and mock server features, Postman accelerates API development and ensures reliable endpoint functionality.

11 ChatGPT

Fig. 17: ChatGPT

ChatGPT is an AI-powered API that integrates generative AI capabilities into applications through OpenAI's endpoints. The service, based on GPT-3.5 and GPT-4 models trained on extensive datasets, provides natural language processing functionalities. Through REST API integration, it enables features like text generation, language translation, and content summarization in applications. Its flexible token-based system and documented endpoints allow developers to implement sophisticated AI features effectively.

12 Azure

Fig. 18: Azure

Azure is a comprehensive cloud computing platform designed to streamline application development, deployment, and management. The platform provides a wide range of services, including virtual machines, databases, and AI tools, enabling scalable and secure solutions. Its resource management capabilities allow seamless integration across hybrid environments, while built-in DevOps tools enhance collaboration and automation.

3 Task Distribution

TABLE II: Role Assignments

Tasks	Name	Descriptions
Project Manager	Kang DongHun	A Project Manager functions as the operational orchestrator responsible for planning, executing, and delivering projects within specified constraints of scope, time, and budget. Their role involves developing comprehensive project plans, establishing critical milestones, and implementing methodologies such as Agile or Waterfall to ensure efficient project delivery. They are accountable for resource allocation, timeline management, and budget control, utilizing project management tools and methodologies to track progress, identify bottlenecks, and maintain project momentum. Critical responsibilities include conducting regular status meetings, managing project documentation, tracking deliverables, and ensuring quality standards are met throughout the project lifecycle. Their success is measured through project completion metrics, team performance, and adherence to initial project parameters, requiring strong leadership skills and the ability to adapt to changing project demands while maintaining team cohesion and project focus.
Frontend Developer	Pyo JunSeong, Bae Hyejin	Frontend developers utilize languages like React Native and TypeScript to create applications. They are responsible for designing the interfaces that users interact with, such as tapping buttons and swiping through screens. Their primary objective is to create a user experience that is both accessible and engaging, while adhering to the specified design. Additionally, frontend developers are responsible for transferring user-entered information to the backend developers. The reason for having three frontend developers is that two of them write code for each screen, while the third developer reviews and optimizes the code for the screens that users see. This organizational structure requires effective teamwork, clear role allocation, excellent communication skills, and collaborative synergy.
Backend Developer	Kang DongHun, Yang Byeong Hyun	Backend developers are responsible for designing the database and application architecture, as well as writing the APIs used by frontend developers. When working with APIs, backend developers need to be able to receive information from application users through the frontend and provide the correct return value to the API.
Backend Developer	Kang DongHun, Yang Byeong Hyun	They also need to design APIs that interact with the backend to leverage generative AI and machine learning features, and make them accessible to application users. This role requires a strong understanding of the central database and software structure, and ensuring that software development aligns with that structure.moment
UI-UX Designer	Pyo JunSeong, Bae Hyejin	The UI-UX designer, using Figma, is responsible for determining how the application screens are presented to users. This role involves deciding which screens will be more engaging and comfortable for users to use. As a UI-UX designer, the goal is to create a design that keeps users engaged and encourages them to return to the application. Once the UI-UX decisions are finalized, they can be communicated to the front-end developers
AI Developer	Pyo JunSeong, Yang Byeong Hyun	A machine learning software developer works with algorithms, data, and artificial intelligence. Their role involves researching, building, and designing artificial intelligence software specifically for machine learning purposes. They primarily focus on applying artificial intelligence systems to various applications. The responsibilities of this role include collecting, cleaning, and preprocessing data to extract meaningful value. They then use this data to train models and deploy them in software. Additionally, the machine learning software developer must appropriately implement machine learning algorithms into software functions, conduct experiments and tests of AI systems, and determine the most suitable models for the application's functions. They are also responsible for designing and developing machine learning systems, as well as performing statistical analysis.

IV. SPECIFICATION

A. *Splash Screen Page*
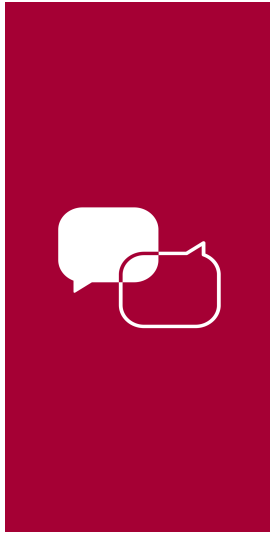
1. Entry-splashing

Fig. 19: Entry Splashing-Page

When the application is launched, the startup page should be displayed for a duration of 1 to 2 seconds to prevent an empty page from being shown while the application is loading its data. This ensures a smooth and visually appealing user experience during the app's startup process.
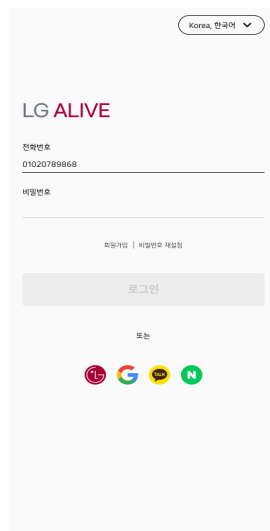
B. *SignUp Page*

1. Login-Page

Fig. 20: Login-Page

Users should be able to use the following features in the login page: Sign Up, Log In, Reset Password, SNS Login, and Language Change.

2. Sign Up-Phone Number
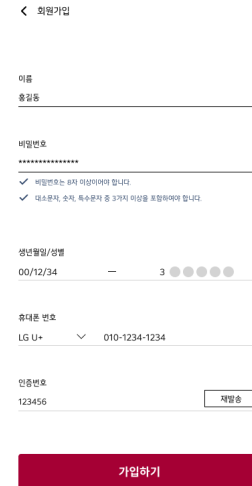
Fig. 21: Sign Up Page

Users are required to enter their phone number. The phone number will serve as the user's ID during the login process after registration. The validity of the phone number should be verified through the authentication system of the mobile service provider.

3. Sign Up-Password
Users must enter a password. The password should be at least 8 characters long and must contain a combination of at least 3 of the following: uppercase letters, lowercase letters, numbers, and special characters. When the user enters their desired password, it is displayed in the form of '****' on the screen, with each condition changing color to green when it is satisfied, and red when it is not.

4. Sign Up-Name and Birthdate
Users must enter their name and date of birth. The date of birth should be entered in the 'YY/MM/DD' format, and gender will be verified based on the first digit of the resident registration number.

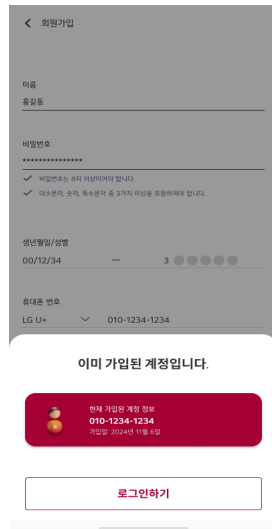5. Sign Up-Preventing Duplicated PhoneNumber

Fig. 22: Sign Up - Preventing Duplicated Phone Number

Registration with duplicate phone numbers must be prevented. Attempting to register with a phone number that is already in use should not be allowed.
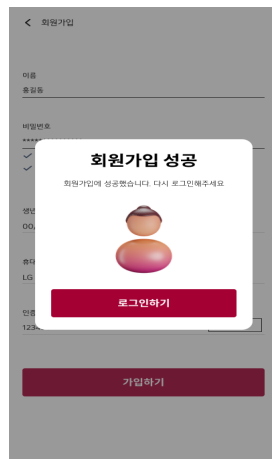
6. Sign Up-Registration Completed

Fig. 23: Sign Up - Registration Completed

Upon successful registration, a notification should be displayed to the user, and they should be automatically redirected to the login process. If the phone number is already registered, the screen displays existing account's information including phone number and registering date.

C. Login Page

1. Login-Types
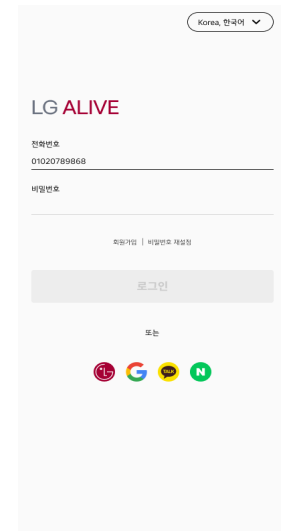
Fig. 24: Login-Types

Users should be able to log in using two types of login methods: (1) Local login via ALIVE membership, (2) SNS login via social media integration.

2. Login-Local Success
If the ID and password entered by the user exist in the member database, the user will successfully log in and be directed to the main page.
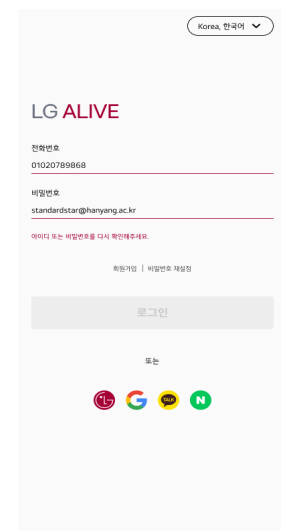
3. Login-Local Failed

Fig. 25: Login - Local Failed

If the user enters their ID and password, but either the ID or the password is incorrect or the user does not exist in the member database, a popup window will request the user to check their ID and password again.

4. Login-SNS Login
   The system utilizes SNS registration APIs such as Google, Naver, Kakao, and more. Users should be able to conveniently log in through these platforms.

5. Login-MyLG Login
   The system utilizes registration through MyLG. Users should be able to conveniently log in through these platforms.

6. Login-Password Reset Authentication

Fig. 26: Password Reset
- Authentication

The system should offer users guidance on setting a new password through carrier-based verification. Upon successful carrier authentication, users will be directed to the 'Password Reset' page; in case of failure, they will return to this page.

7. Login-New Password

Fig. 27: New Password Failed

Fig. 28: New Password Success

When setting a new password, users should be provided with appropriate security requirements, such as a combination of at least 8 characters, including at least three of uppercase letters, lowercase letters, numbers, and special characters.

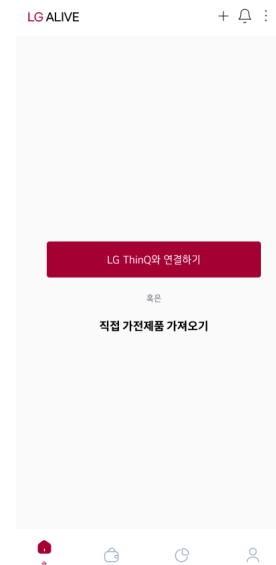*D. Home Page*

1. Default-Not Imported

Fig. 29: Default - Not Imported

System must initialize with import options when no devices are registered. System should display two primary action buttons: 'Import ThinQ' and 'Import Directly' in the main interface. These buttons must remain prominently visible until initial device registration is completed.

2. Import ThinQ

Fig. 30: Import ThinQ

When users tap the Import ThinQ button, a connection popup should immediately appear on screen. The system should then automatically initiate ThinQ service connection while displaying a loading indicator to show progress. Upon completion, the system must show either a success or failure message to inform the user of the connection status.

Fig. 31: Import Directly

5. Import Directly-WiFi or Bluetooth
When users choose the WiFi/Bluetooth import method, the system should check and request any necessary device permissions. The interface should display a list of available devices for connection, clearly indicating the connection status throughout the process. Users should be provided with clear, step-by-step connection instructions to ensure successful device pairing.

3. Import Directly
When users select the Import Directly option, the system should display a modal popup presenting two distinct import methods: QR scanning and Wi-Fi/Bluetooth connection.

4. Import Directly-QR
Upon selecting the QR method, the system should first request camera permissions if not previously granted. The QR scanner should immediately open, displaying a clear scanning area in the viewfinder. The system should display appropriate error messages for invalid or unrecognized QR codes.

6. Entire Chat The system should display the

Fig. 32: Entire Chat

main chat room interface that includes all connected appliances. The header section must contain four essential buttons: a chat room selection toggle for switching between different chat rooms, a create chat room button for establishing new chat spaces, a notification button for accessing system alerts, and an others button for additional settings.
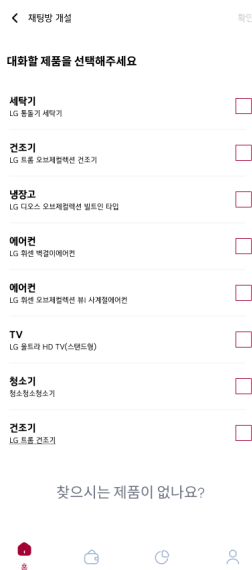
7. Select ChatRoom

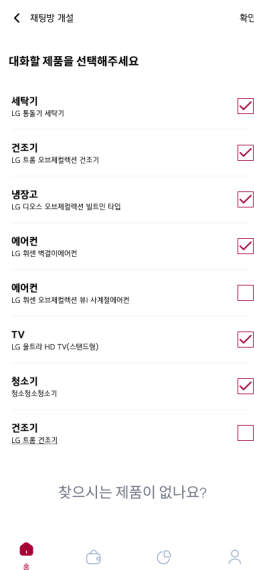
Fig. 33: Select Chat-Room - None

Fig. 34: Select Chat-Room - Available

The system should allow users to navigate

between different appliance chat rooms through the chat room toggle button. Users should be presented with a list of available chat rooms and can easily switch to their desired room.
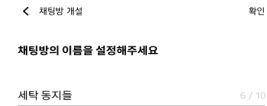
8. Create ChatRoom

Fig. 35: Create Chat-Room

When users tap the create chat room button, the system should display a comprehensive list of all connected appliances. Users can assign a chat room name with a maximum of 10 letters.

9. Notice

Fig. 36: Notice

The notification system should provide users with important app alerts and updates. A red dot indicator must appear on the notification icon when there are unread alerts, ensuring users are aware of new notifications. Upon clicking the notification button, the system should display a chronological list of all notifications, distinguishing between read and unread messages.
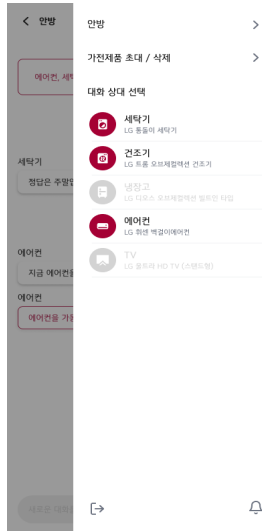
10. Others-Edit

Fig. 37: Others-Edit

Users should be able to edit chat room information. They can rename chat rooms, and add or delete appliances. The minimum number of appliances in a chat room must be at least one. Users should also be able to toggle the power status (ON/OFF) of devices in the chat room. Only active (ON) devices can receive commands and participate in interactions.

11. Chat-Appliance-Tag
Users should be able to tag and command home appliances within chat rooms using the "@" symbol (e.g., "@WashingMachine"). The system must provide real-time validation of device availability.

12. Voice-Command

Fig. 38: Voice Command

Users should be able to communicate with their appliances through voice commands. For example, users can say, "Air conditioner, I'm feeling a bit warm," and receive conversational responses about adjusting the temperature settings.

13. Command-Verification

Fig. 39: Command Verification

The appliance should respond within the chat with a message asking if the user would like to perform the suggested action. There should be small "Confirm" (O) and "Cancel" (X) icons for user approval or rejection. The appliance will only execute the action after the user confirms.

14. ChatRoom-Search
Users should be able to search for past conversations within the chat room by typing keywords or device names. The search results should display relevant chat history with context on when the conversations occurred.

15. ChatRoom-Exit

Fig. 40: ChatRoom Exit

Users should be able to delete an appliance chat room they have created. Upon deletion, the user will be prompted to confirm, and all contents will be permanently removed. Users should have the option to archive the chat history before deletion if desired.
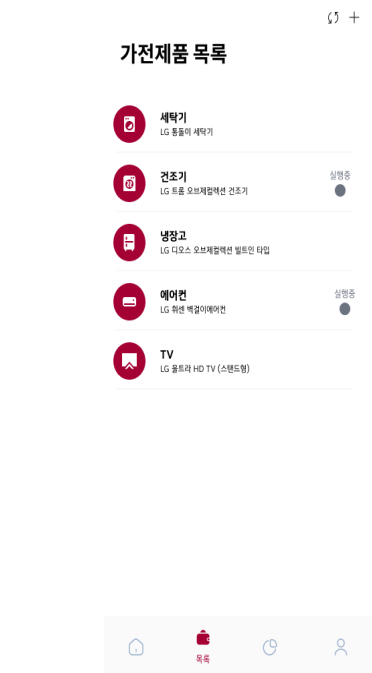
E. Appliance Page

1. Appliances-Default

Fig. 41: Appliances Default

The system should display a comprehensive list of all connected appliances, providing users with a clear overview of their smart home ecosystem. Each appliance entry should show essential information, including connection status, device name, and current operational status. The list must be updated whenever device status changes or new devices are added.

2. Appliances-Sync

Fig. 42: Appliances Sync

When users activate the sync button, the system should initiate synchronization with ThinQ services and the application list in the internal app to update the appliance list. Upon successful synchronization of new devices, the system must automatically add them to the entire chat room and display a welcome message in the format "Appliance [Name] joined default chat room". The synchronization process should show a clear progress indicator and confirmation of completion or any errors encountered.

3. Appliances-Import

Fig. 43: Appliances Import

The appliance import functionality should mirror the import options available on the main page, providing users with two distinct import methods: QR code scanning and WiFi/Bluetooth connection. For QR scanning, the system must activate the device camera and provide clear scanning guidance. The WiFi/Bluetooth method should display available devices and guide users through the connection process with clear step-by-step instructions.
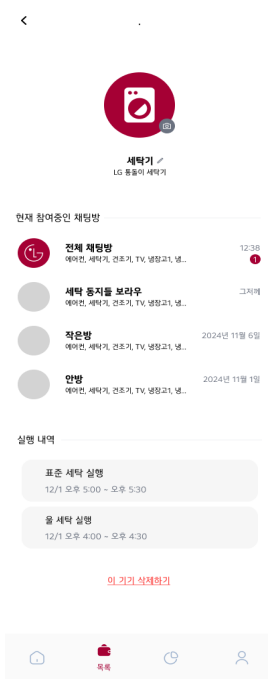
4. Applications-Detail

Fig. 44: Applications Detail

When users select an appliance from the list, the system should display a detailed information page including comprehensive usage statistics and interaction history. This detail view must show a list of participating chat rooms, total operation time, user feedback and ratings, and a chronological history of interactions with other appliances. The interface should present this information in clearly organized sections, allowing users to easily track and analyze their appliance usage patterns and performance history.

F. MyPage
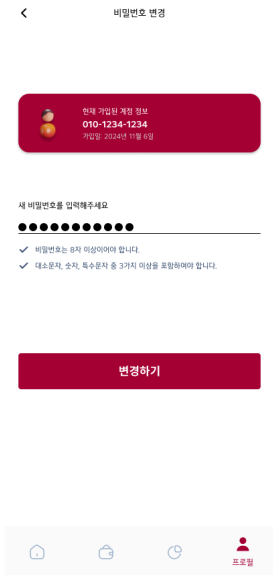
1. Mypage-Password Change

Fig. 45: Password Change

Users must enter their current password to verify their identity before changing it to a new one. The new password must be at least 8 characters long, including special characters. If the new password is identical to the current one, a notification will inform the user that it cannot be reused. Passwords must include at least 3 of the following: uppercase letters, lowercase letters, numbers, and special characters.

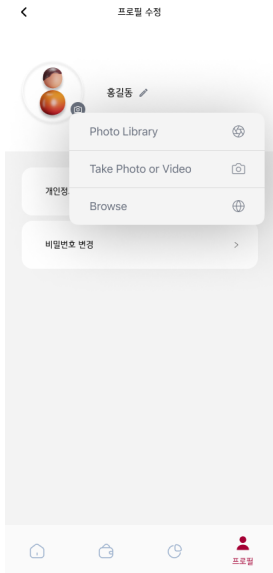2. Mypage-Profile Picture Update

Fig. 46: Profile Picture Update

Users can upload a new profile picture or select from existing images on their device. The updated profile picture will appear throughout the app.
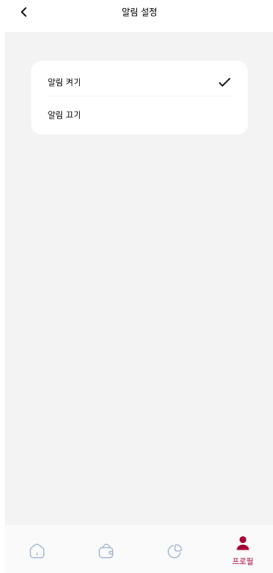
3. Mypage-Notification Settings

Fig. 47: Notification Settings

Users can enable or disable notifications for app events like messages, updates, and alerts. Preferences are saved immediately.
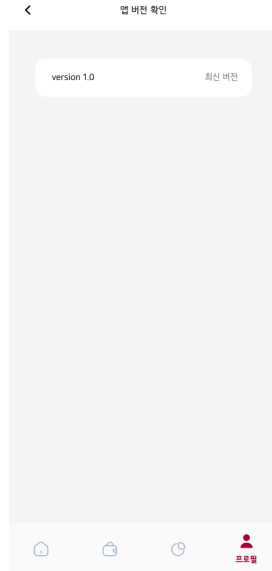
4. Mypage-App Version Check

Fig. 48: App Version Check

The app displays the current version installed on the user's device. Users can check if they have the latest version.

5. Mypage-Language Settings
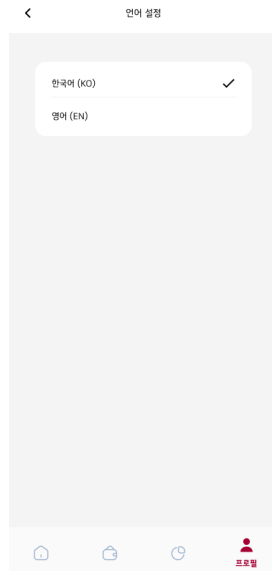
Fig. 49: Language Settings

Users can choose their preferred language for the app interface, which updates immediately upon selection.

6. Mypage-Personal Information
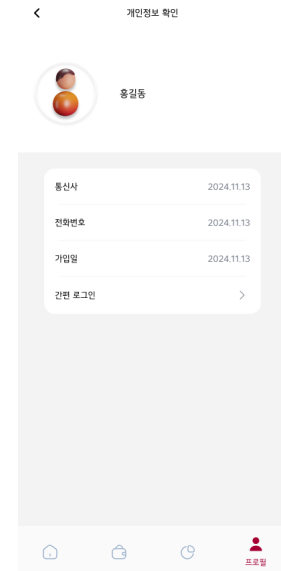
Fig. 50: Personal Information

Users can view personal details like carrier, phone number, and registration date, and set up simplified login options.

G. Result Page

1. ResultAnalysis-Calendar View

Fig. 51: Result Calender

The Result Analysis page should display a calendar-based view that shows all interaction outcomes from appliance conversations. Each calendar date must present the specific choices and results

that were provided to users during their interactions. The system should maintain a comprehensive record of all interactions, displaying them in a chronological format that allows users to easily navigate through their history.
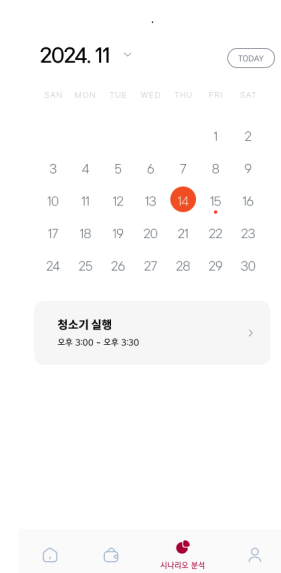
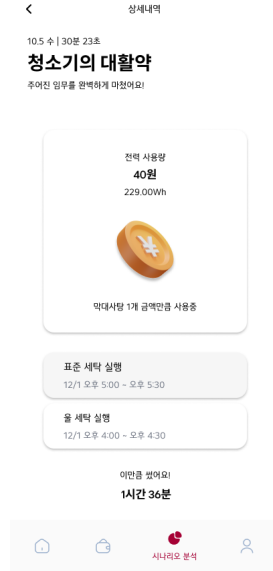2. ResultAnalysis-Data Presentation

Fig. 52: ResultAnalysis-Data

The system must present all interaction data in an organized and easy-to-understand format within the calendar view. Each date with recorded interactions should be clearly marked, and users should be able to view detailed information about the interactions that occurred on any specific date. The interface should allow users to track patterns in their appliance usage and decision-making over time, providing valuable insights into their interaction history. This calendar-based presentation format ensures that users can efficiently review and analyze their past interactions, understanding how the system responded to different situations across various timeframes.

V. ARCHITECTURE DESIGN

A. Overall Architecture
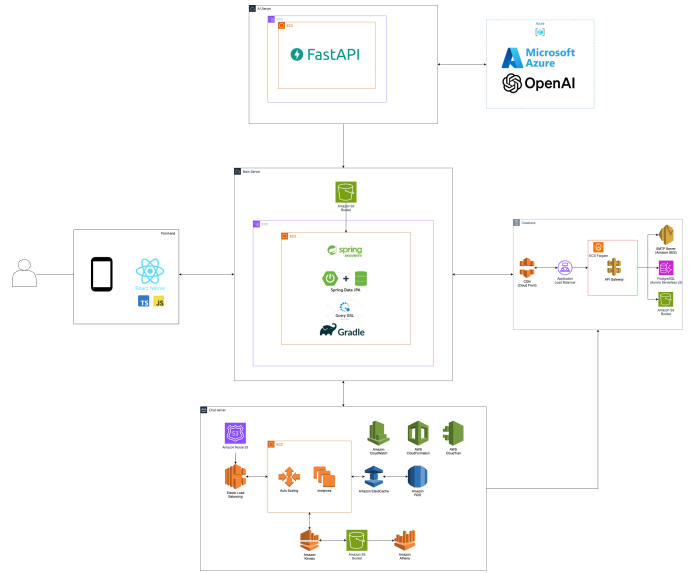
Fig. 53: Overall Architecture

ALIVE is composed of four main modules: the Frontend, the Backend (Main Server), the Chat Server (Sendbird), and the AI Server, each playing a distinct role in delivering a seamless user experience.

The first module "Frontend." is built using React-native with TypeScript, JavaScript and serves as the user interface. It allows users to interact with the system by providing features such as data retrieval, input, and chat functionality, while also displaying AI-generated results. The frontend communicates with the backend server via APIs, connects to the chat server using WebSocket or the Sendbird SDK for real-time messaging, and visualizes responses from the AI server.

The second module, "Backend (Main Server)", implemented using Spring Boot and Kotlin, acts as the core processing unit of the system. It manages user authentication, processes chat-related operations such as creating and managing chat rooms, channels, and channel users, and routes requests to the appropriate components like the database, AI server, or chat server. Additionally, it provides APIs to the frontend and ensures smooth data management through integration with Supabase and PostgreSQL for storing user information, logs, and chat-related data.

The third module "Chat Server" powered by Sendbird, facilitates real-time messaging between users. It supports one-on-one and group chats, manages message storage, and provides notifications and user status updates. The chat server connects with the frontend through the Sendbird SDK, ensuring reliable and real-time communication. It also synchronizes chat data with the backend and database as needed.

The fourth module "AI Server" is implemented using FastAPI. It is deployed on Azure and integrates with OpenAI to provide intelligent responses. Its primary function is to generate contextually appropriate replies tailored for conversations in chat rooms involving smart appliances. By leveraging OpenAI's generative AI, the server ensures that the responses align with the unique interactions and requirements of smart device communication.

B. Directory Organization

TABLE III: DIRECTORY-ORGANIZATION-FRONTEND-1

Directory	File Name
ALIVE/frontend/src/app	_layout.tsx +html.tsx +not-found.tsx index.tsx
ALIVE/frontend/src/app/ (tabs)	_layout.tsx analysis.tsx inbox.tsx index.tsx list.tsx profile.tsx
ALIVE/frontend/src/app/ (tabs)/appDetail	appKey.tsx
ALIVE/frontend/src/app/ (tabs)/chat	[channel_url].tsx create_chat_name.tsx create_chat_room.tsx index.tsx
ALIVE/frontend/src/app/ (tabs)/exeAnalysis	[exeKey].tsx
ALIVE/frontend/src/ components/analysis	ContentBox.tsx CustomCalendar.tsx DatePickerModal.tsx energyCard.tsx RecordsList.tsx
ALIVE/frontend/src/ components/chat	ChatBubble.tsx ChatContainer.tsx ChatDrawer.tsx ChatInput.tsx ChatRoomCard.tsx

TABLE IV: DIRECTORY-ORGANIZATION-FRONTEND-2

Directory	File Name
ALIVE/frontend/src/ components/common	AppBar.tsx Bold.tsx Button.tsx ClickBox.tsx DeviceCard.tsx InputField.tsx Loading.tsx ThemedText.tsx ThemedView.tsx
ALIVE/frontend/src/ components/icons	Analysis.tsx Bell.tsx BellOff.tsx DotsCircle.tsx Home.tsx MoreVertical.tsx Plus.tsx Profile.tsx Sync.tsx Wallet.tsx
ALIVE/frontend/src/ components/icons/devices	AirConditioner.tsx AirPurifier.tsx Cleaner.tsx Dryer.tsx Refrigerator.tsx Tv.tsx WashingMachine.tsx
ALIVE/frontend/src/ components/list	deviceList.tsx ExeLogBox.tsx ParticipatingChatRoom.tsx
ALIVE/frontend/src/ components/login	toggle.tsx
ALIVE/frontend/src/ components/navigation	TabBarIcon.tsx
ALIVE/frontend/src/hooks	useColorScheme.ts useColorScheme.web.ts useLogin.ts useModal.ts useThemeColor.ts
ALIVE/frontend/src/service	auth.service.ts axios-instance.ts channel.service.ts chat.service.ts device.service.ts message.service.ts
ALIVE/frontend/src/stores	useDeviceStore.ts useUserStore.ts
ALIVE/frontend/src/types	chat.ts device.ts icon.ts images.d.ts record.types.ts
ALIVE/frontend/src/utils	calendar.ts date.ts

TABLE V: DIRECTORY-ORGANIZATION-BACKEND

Directory	File Name
ALIVE/backend/src/main/ kotlin/com/example/demo	Application.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ config	AppConfig.kt GlobalExceptionHandler.kt SecurityConfig.kt SendbirdConfig.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ controller	AuthController.kt ChannelController.kt DeviceController.kt DeviceUsageRecordController.kt UserController.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ controller/sendbird	SendbirdChannelController.kt SendbirdMessageController.kt SendbirdUserController.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ repository	ChannelDeviceRepository.kt ChannelRepository.kt DeviceRepository.kt DeviceUsageRecordRepository.kt UserDeviceRepository.kt UserRepository.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ service	ChannelService.kt DeviceService.kt DeviceUsageRecordService.kt UserService.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ service/sendbird	SendbirdChannelService.kt SendbirdMessageService.kt SendbirdUserService.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ model	Channel.kt ChannelDevice.kt Device.kt DeviceCategory.kt DeviceUsageRecord.kt User.kt UserDevice.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto	ActionDTO.kt LoginRequestDTO.kt UserDeviceResponseDTO.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto/channel	ChannelDTO.kt ChannelDeviceDTO.kt ChannelResponseDTO.kt CreateChannelRequestDTO.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto/sendbird	QueryMessagesRequest.kt SendMessageRequest.kt SendbirdChannelCreateRequest.kt SendbirdUserCreateRequest.kt SendbirdUserCreateResponse.kt SendbirdUserInviteRequest.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto/user	UserCreateRequestDTO.kt UserDTO.kt UserInviteRequestDTO.kt UserResponseDTO.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ exception	DeviceNotFoundInChannel Exception.kt DeviceNotInProgressException.kt UserDeviceNotFoundException.kt UserNotFoundException.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/util	SendbirdApiHelper.kt

TABLE VI: DIRECTORY-ORGANIZATION-AI

Directory	File Name
ALIVE/AI	requirements.txt
ALIVE/AI/app	__init__.py main.py retriever.py
ALIVE/AI/app/api	routes.py
ALIVE/AI/app/data	create_dataset_azure.py create_dataset_openai.py dataset.json domain_slots.json
ALIVE/AI/app/helper	turn_data_to_str.py
ALIVE/AI/app/prompts	system_prompt.txt table_prompt.txt

C. Module 1: Frontend

i) Purpose
The Frontend Module of ALIVE serves as the client-facing interface, enabling users to interact with the system seamlessly. It handles the display of data, user input, and interactions, ensuring a user-friendly experience across devices. Built using modern web technologies, the frontend communicates with the backend, AI server, and chat server to present real-time data, manage user settings, and facilitate smart device interactions within chat rooms.

ii) Functionality
The frontend provides a range of features, including user authentication, profile management, appliance list views, and device usage analysis presented in calendar and card formats. It enables real-time chat functionality for interacting with smart appliances and supports dynamic routing for detailed views such as chat rooms and execution analysis. Additionally, it includes reusable components like buttons, modals, and charts for efficient UI development. Hooks and services are used to manage states and make API calls for authentication, device control, and messaging, ensuring smooth communication with other modules.

iii) Location of source code:
https://github.com/SWE-ALIVE/frontend

iv) Class component

- _layout.tsx : This is the main layout component that wraps the entire application.

- +html.tsx : This is a file that defines the HTML template structure.

- +not-found.tsx : This is a file that handles 404 error page rendering.

- index.tsx : This is the main entry page of the application.

- _layout.tsx : This is a file that defines the layout structure for tabs.

- profile.tsx : This is a file for user profile management.

- list.tsx : This is a file for displaying appliance list views.

- inbox.tsx : This is a file for handling alarm inbox functionality.

- analysis.tsx : This is a file for displaying application analysis records in calendar format.

- appDetail/[appKey].tsx : This is a dynamic route file for showing detailed application information.

- [channel_url].tsx : This is a dynamic route file for chat room functionality.

- exeAnalysis/[exeKey].tsx : This is a dynamic route file for execution analysis views.

- ContentBox.tsx : This is a component for displaying analysis content in a structured format.

- RecordsList.tsx : This is a component for showing records in a list format.

- energyCard.tsx : This is a component for displaying energy usage information.

- DatePickerModal.tsx : This is a modal component for date selection.

- analysisContentBox.tsx : This is a component for displaying analysis content in a structured format.

- CustomCalendar.tsx : This is a component that provides calendar functionality.

- ChatBubble.tsx : This is a component for displaying chat messages.

- ChatDrawer.tsx : This is a drawer component for chat room side bar.

- ChatInput.tsx : This is a component for chat message input.

- ChatRoomCard.tsx : This is a component for displaying chat room information.

- ChatContainer.tsx : This is a container component for chat functionality.

- AppBar.tsx : This is a component for the top application bar.

- Button.tsx : This is a reusable button component.

- InputField.tsx : This is a reusable input field component.

- Loading.tsx : This is a component for displaying loading states.

- ThemedText/View.tsx : These are components for theme-aware text and view rendering.

- Analysis.tsx, Bell.tsx : These are files containing various icon components.

- devices : This is a folder containing appliance-specific icons.

- useColorScheme.ts: This is a hook for managing color scheme preferences.

- useLogin.ts: This is a hook for handling login functionality.

- useModal.ts: This is a hook for modal state management.

- useThemeColor.ts: This is a hook for theme color management.

- auth.service.ts: This is a file for authentication-related API calls.

- channel.service.ts: This is a file for channel-related API calls.

- chat.service.ts: This is a file for chat-related API calls.

- device.service.ts: This is a file for device-related API calls.

- message.service.ts: This is a file for message-related API calls.

- useDeviceStore.ts: This is a file for managing device-related states.

- useUserStore.ts: This is a file for managing user-related states.

- chat.ts, device.ts, etc.: These are files containing various type definitions.

- calendar.ts: This is a file containing calendar-related utilities.

- date.ts: This is a file containing date handling utilities.

D. Module 2: Backend

i) Purpose
The backend for ALIVE is responsible for managing the server-side operations and databases. It stores, processes, and manages data generated from user interactions on the client-side application and facilitates the retrieval of necessary information. The backend also ensures seamless communication between the frontend, chat server (Sendbird), and AI server. To implement the backend for ALIVE, we utilized Spring Boot with Kotlin for its modern syntax and compatibility. For data storage, we used Supabase and PostgreSQL, which ensure reliable and efficient data management.

ii) Functionality
The ALIVE backend processes user requests from the frontend, chat server, or AI server, managing chatrooms, IoT device synchronization, and user settings. It logs actions in the database to ensure traceability and supports device communication, routine automation, and intelligent chat-based interactions. Additionally, it generates reports and synchronizes settings to deliver a seamless and scalable user experience.

iii) Location of source code:
https://github.com/SWE-ALIVE/backend

iv) Class component
- Application.kt : Entry point for launching the Spring Boot application.

- config : Directory for configuration files and classes, such as application settings and beans.

- AppConfig.kt : Contains general application-wide configuration and bean definitions.

- GlobalExceptionHandler.kt : Handles exceptions globally across the application.

- SecurityConfig.kt : Configures security settings, such as authentication and authorization.

- SendbirdConfig.kt : Configures integration settings for the Sendbird chat service.

- controller : Directory for handling incoming API requests and routing them to the appropriate services.

- AuthController.kt : Manages authentication-related endpoints, such as login and logout.

- ChannelController.kt : Handles endpoints related to channel creation and management.

- DeviceController.kt : Manages device-related operations, such as adding or removing IoT devices

- DeviceUsageRecordController.kt : Handles endpoints for managing and retrieving device usage records.

- UserController.kt : Manages user-related operations, such as profile updates and user information retrieval

- service : Contains business logic for handling core application functionality.

- ChannelService.kt : Handles business logic for channel management.

- DeviceService.kt : Manages IoT device-related operations.

- DeviceUsageRecordService.kt : Handles Processes device usage records.

- UserService.kt : Implements business logic for user management.

- repository : Handles database interactions for various entities.

- ChannelRepository.kt : Manages data access for channel entities.

- ChannelDeviceRepository.kt : Handles data access for channel-device relationships.

- DeviceRepository.kt : Manages database operations for IoT devices.

- DeviceUsageRecordRepository.kt : Handles data access for device usage records.

- UserDeviceRepository.kt : Manages user-device relationships in the database.

- UserRepository.kt : Handles database interactions for user entities.

- model : Defines the core data entities used throughout the application.

- ChannelDevice.kt : Represents the relationship between channels and devices.

- DeviceUsageRecord.kt : Represents records of IoT device usage.

- UserDevice.kt : Represents the relationship between users and their devices.

- dto : Contains Data Transfer Objects for handling requests and responses.

- ActionDTO.kt : Represents data for specific actions performed by users or devices.

- LoginRequestDTO.kt : Handles login request data from clients.

- UserDeviceResponseDTO.kt : Represents response data for user-device relationships.

- dto/sendbird : Contains Data Transfer Objects specifically for Sendbird API integration.

- QueryMessagesRequest.kt : Represents a request to query messages from a Sendbird channel.

- SendMessageRequest.kt : Handles data for sending messages to a Sendbird channel.

- SendbirdUserCreateRequest.kt : Represents data for creating a new user in Sendbird.

- SendbirdUserCreateResponse.kt : Represents the response data after creating a user in Sendbird.

- SendbirdUserInviteRequest.kt : Handles data for inviting users to a Sendbird channel.

- dto/channel : Contains Data Transfer Objects for managing channel-related operations.

- CreateChannelRequestDTO.kt : Handles data for creating a new channel.

- ChannelResponseDTO.kt : Represents response data for channel-related requests.

- ChannelDeviceDTO.kt : Represents data for devices associated with a channel.

- dto/user : Contains Data Transfer Objects for user-related operations.

- UserCreateRequestDTO.kt : Handles data for creating a new user.

- UserResponseDTO.kt : Represents response data for user-related requests.

- UserInviteRequestDTO.kt : Handles data for inviting a user to a specific channel or group.

- exception : Contains custom exceptions for handling application-specific errors.

- DeviceNotFoundInChannel : Thrown when a device is not found in the specified channel.

- Exception.kt : Base class for defining custom exceptions in the application.

- DeviceNotInProgressException.kt : Thrown when a device is not in a valid progress state.

- UserDeviceNotFoundException.kt : Thrown when a user-device relationship is not found.

- UserNotFoundException.kt : Thrown when the specified user is not found.

- util : Contains utility classes and helpers for reusable functionality.

- SendbirdApiHelper.kt : Provides helper functions for interacting with the Sendbird API.

E. Module 3: AI

i) Purpose
The AI module serves as the core of intelligent interactions within the ALIVE system. Its primary purpose is to process user inputs, generate context-aware responses, and handle dataset creation and interaction with AI models. By leveraging technologies such as FastAPI, Azure, and OpenAI, the AI module ensures real-time communication and provides dynamic, personalized outputs to enhance the user experience.

ii) Functionality
The AI module interacts with the backend to process requests and utilizes OpenAI models to generate tailored, context-aware responses. It supports dataset creation using Azure and OpenAI APIs, manages predefined AI prompt templates, and handles natural language processing tasks. Additionally, it provides utilities for data preprocessing and model input preparation, ensuring efficient integration of AI services within the system.

iii) Location of source code:
https://github.com/SWE-ALIVE/AI

iv) Class component
- app : Core application logic, including the entry point.

- __init__.py : Initializes the app package for the AI server.

- main.py : Serves as the entry point for the FastAPI server, defining the application's startup configuration.

- retriever.py : Implements logic for retrieving and processing data, such as querying data sources or preparing input for AI models.

- routes.py : Defines the API endpoints for the AI server, managing requests and responses for client interactions.

- app/data : Scripts and files for dataset creation and management.

- create_dataset_azure.py : Generates datasets by interacting with Azure services.

- create_dataset_openai.py : Creates datasets using OpenAI APIs for training or inference.

- dataset.json : Stores preprocessed dataset information used by the AI server.

- domain_slots.json : Contains slot information for domain-specific AI interactions, helping structure conversational data.

- app/helper : Utilities for data preprocessing.

- turn_data_to_str.py : A utility script to convert structured data into string format, facilitating easier processing or integration with models.

- app/prompts : Predefined templates for AI prompts.

- system_prompt.txt : Stores the base

system prompt for guiding AI interactions.

- table_prompt.txt : A predefined prompt template tailored for interpreting or generating table-based outputs.

VI. AI SPECIFICATION

A. In-Context Learning for Few-Shot Dialogue State Tracking(ICL-DST)

1) Introduction
   Dialogue State Tracking (DST) is a crucial component in dialogue systems, responsible for tracking user requirements and determining subsequent actions. Traditional DST approaches require retraining the model each time a new domain is added, leading to high costs and inefficiency. To address these challenges, ICL-DST introduces a zero-shot and few-shot learning framework that leverages In-Context Learning by incorporating in-context examples into a pre-trained language model, enabling flexible prediction of dialogue states across new domains.

2) DST System Design
   The ICL-DST system is designed to handle multi-domain scenarios, enabling the tracking and prediction of dialogue states for each domain. Each domain, such as user-appliance interactions or appliance-appliance communications, has a predefined set of slots like "appliance-name" or "appliance-status". The dialogue state is represented as a set of slot-value pairs, potentially spanning multiple domains.

3) DST Task Framing
   Dialogue consists of a series of turns alternating between user utterances and system responses. The DST task is to predict the current dialogue state at each turn based on the dialogue context. The dialogue state is represented as a set of slot-value pairs and can include slots and values from multiple domains. To mitigate data sparsity in few-shot learning, state changes occurring at each turn are used as labels for prediction.

4) DST as Text-to-SQL
   ICL-DST transforms DST into a Text-to-

SQL problem by representing dialogue states in SQL format. Each domain is defined as a table, and each slot is defined as a column. All slots and values are included in the WHERE clause. For turns involving multiple domains, each domain is renamed accordingly. This approach aligns well with Azure's OpenAI GPT-4o-mini, enhancing the effectiveness of state prediction.

5) Dialogue Context Representation
   Instead of including the entire dialogue history, only the most recent N turns are considered. ICL-DST leverages the fact that the dialogue state is a summary of the dialogue history by representing the dialogue context as $[y_{t-1}, A_t, U_t]$, where y_{t-1} denotes the accumulated dialogue state after the previous user turn.

6) In-Context Learning
   In-Context Learning (ICL) involves keeping the parameters of a pre-trained language model fixed and providing a prompt that includes task descriptions, examples, and test instances. This allows the model to learn patterns within the dialogue context without requiring additional training. By structuring the dialogue state in SQL format, the model can effectively predict the dialogue state based on this representation.

7) Schema Prompting
   To represent the dialogue schema within the prompt, SQL tables are used for each domain. Each table consists of rows for slot names and example values associated with each slot. For slots with many possible values, only a few example values are listed to maintain prompt efficiency. In experiments, SQL tables for all domains are generated and concatenated to form part of the input.

8) In-Context Examples
   In few-shot scenarios, a retriever system takes the dialogue context $[y_{t-1}, A_t, U_t]$ as input and retrieves similar examples from a labeled training set. Using the dialogue state y_{t-1} instead of the full dialogue history allows for shorter inputs, enabling the inclusion of more examples and more effective similarity scoring. In

zero-shot scenarios, in-context examples are formatted as example turns.

9) Dialogue Retriever
The success of ICL in few-shot environments heavily depends on the quality of in-context examples. ICL-DST extends retrieval to encompass the entire dialogue history by utilizing embedding models like S-BERT to compute the similarity of dialogue states. This allows for the effective extraction of relevant example dialogues that exhibit similar state changes, ensuring accurate state prediction.

B. Dataset

We created a custom dataset for appliance conversations using Azure OpenAI: GPT-4-mini, following the structure of the well-known multiWOZ dataset. The dataset is specifically designed to suit interactions between smart appliances while maintaining the schema and format of multiWOZ.

1) Dataset Creation Conditions
   i) Each dialogue involves at least one and up to three appliances participating in the conversation.
   ii) OpenAI's structured output functionality was utilized to ensure the dataset adheres to a consistent JSON format.

2) Dataset Volume
   i) A total of 1,000 dialogues were generated, far exceeding the 80 dialogues that are considered sufficient for many research applications.
   ii) This larger volume addresses potential accuracy concerns arising from automated generation, providing robustness through diversity and scale.

3) Comparison with multiWOZ
   i) The multiWOZ dataset was manually curated, ensuring high accuracy for its dialogues. However, its design is tailored for human-to-human or human-to-machine interactions.
   ii) In contrast, our dataset is adapted specifically for appliance-to-appliance and appliance-to-human interactions, replicating multiWOZ's structural

principles while introducing scenarios unique to smart home environments.
   iii) By focusing on quantity and domain-specific relevance, we offset any limitations from the automated generation process, ensuring the dataset remains effective and practical for AI training.

This approach ensures the dataset is not only scalable but also tailored to the unique needs of smart home technology, bridging the gap between manual curation and large-scale automation.
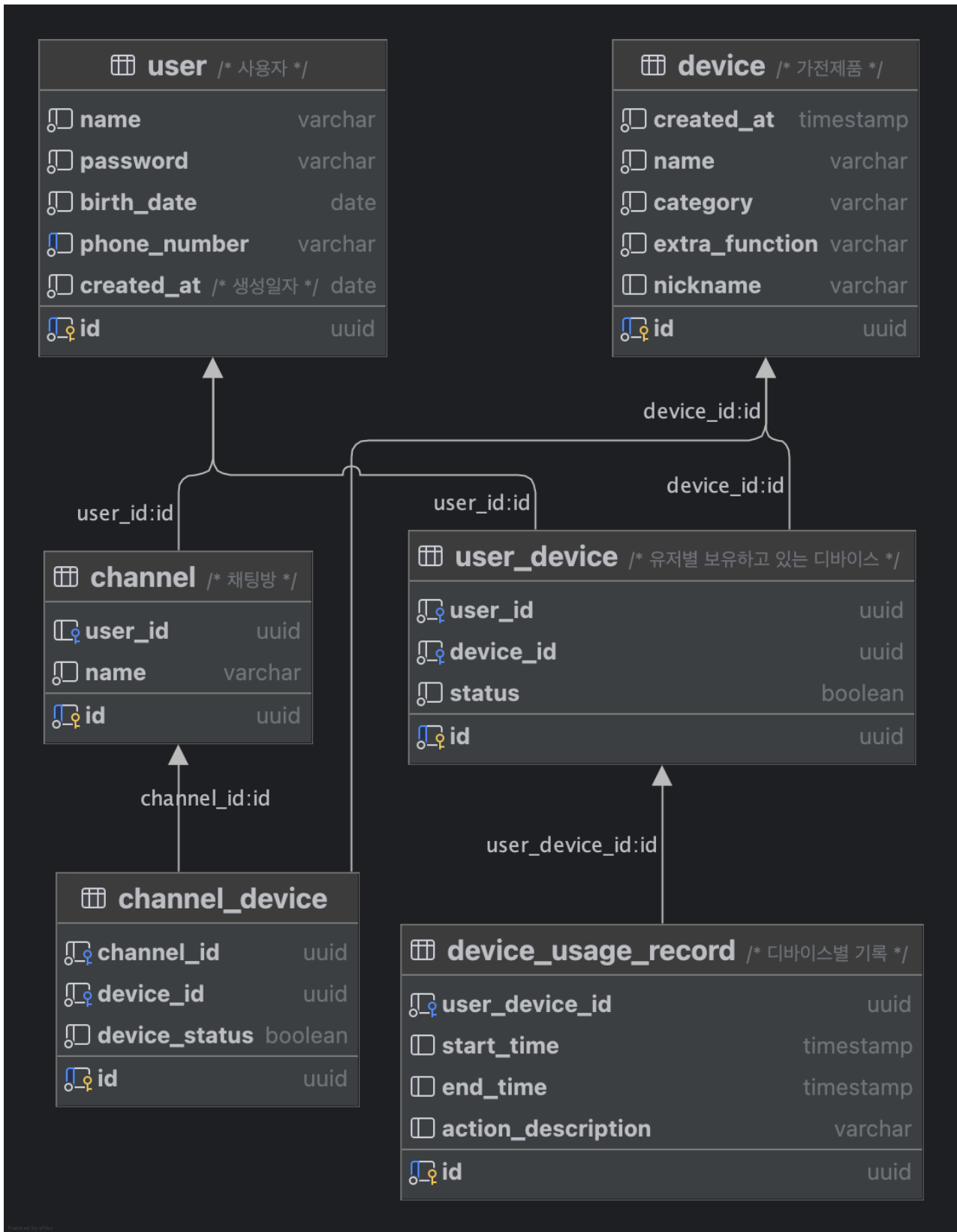
VII. APPENDIX

Fig. 54: Database Entity Relationship Diagram

REFERENCES

[1] "opensurvey,"https://blog.opensurvey.co.kr/trendreport/smart-home-2023/, 2023.

[2] "React Native," https://reactnative.dev/, 2023.

[3] "Typescript," https://www.typescriptlang.org/, 2023.

[4] "Kotlin," https://kotlinlang.org/, 2023.

[5] "SpringBoot," https://spring.io/projects/spring-boot, 2023.

[6] "Hivernate," https://hibernate.org/, 2023.

[7] "FastAPI," https://fastapi.tiangolo.com/, 2023.