ALIVE

AI-driven multi-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-chat 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. 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-chat system 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

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 systems, where multiple systems collaborate, think, and learn independently. This evolution will likely lead to the integration of multiple 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. 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. 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]

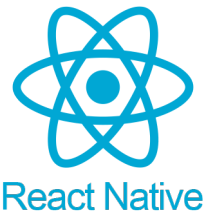


Figure 1: React Native

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]

Figure 2: Typescript

TypeScript is a superset of 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]

Figure 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) SpringBoot [5]

Figure 4: SpringBoot

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]

Figure 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]

Figure 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.

B. Software in use

1. visual Studio Code

Figure 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

Figure 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

Figure 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

Figure 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

Figure 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

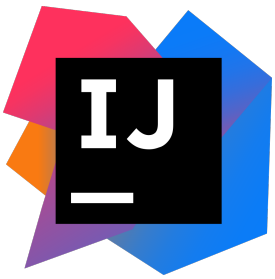


Figure 12: IntelliJ

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

Figure 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

Figure 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

Figure 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

Figure 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

Figure 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

Figure 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.

13. Amazon Web Services (AWS)

Figure 19: Amazon Web Services (AWS)

Amazon Web Services (AWS) is a comprehensive cloud computing platform developed by Amazon, offering a wide range of on-demand computing services and APIs to individuals, companies, and governments. It provides scalable and reliable solutions for application hosting, data storage, machine learning, and DevOps workflows. AWS's extensive global infrastructure supports high availability and low latency, while services like EC2, S3, and Lambda cater to diverse development needs. Its robust security features and pay-as-you-go pricing model make it an optimal choice for building and deploying scalable, cost-effective applications across various industries.

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. 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

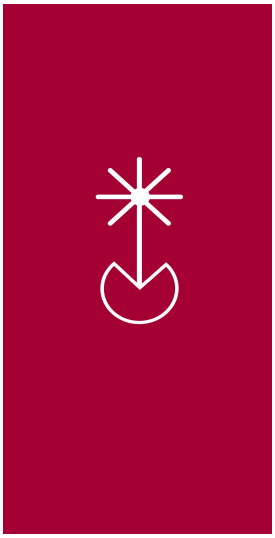


Figure 20: Entry Page

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

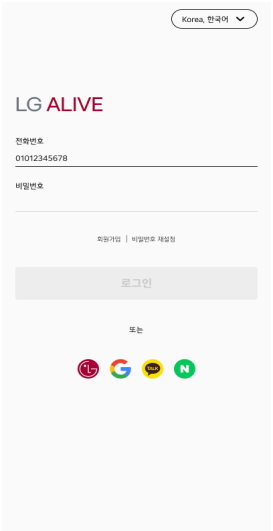


Figure 21: Login-Page

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

Figure 22: 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

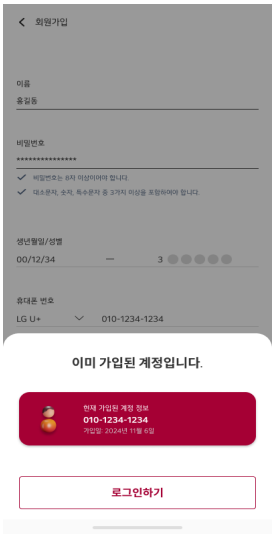


Figure 23: Sign up - Preventing Duplicate

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

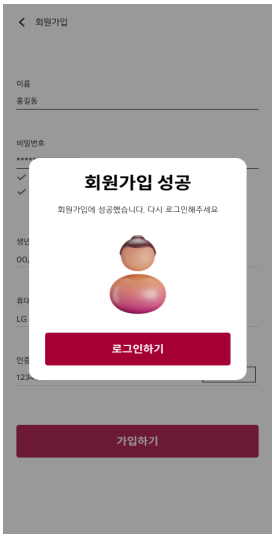


Figure 24: Sign up - Register Completed

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

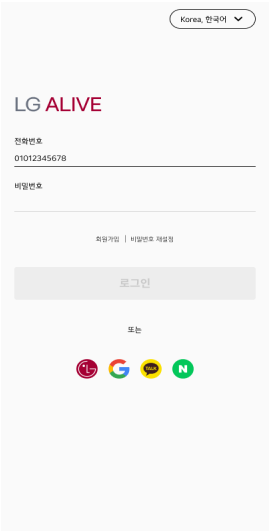


Figure 25: Login-Types

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

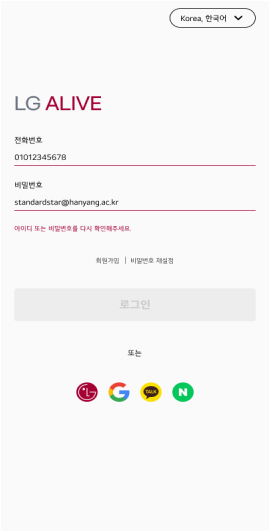


Figure 26: Login Failed

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 conveni-
   ently log in through these platforms.

6. Login-Password Reset Authentication

Figure 27: 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

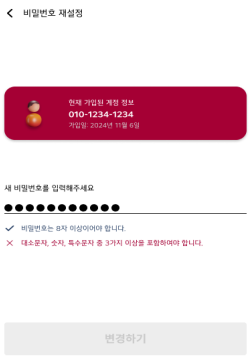


Figure 28: New Passowrd Failed

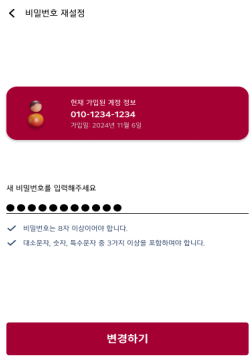


Figure 29: New Password Passed

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

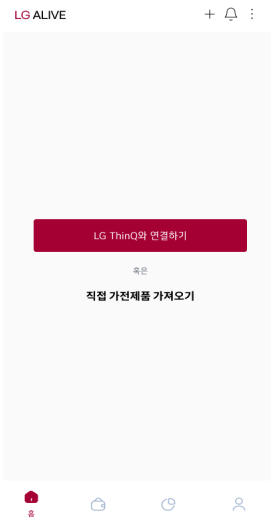


Figure 30: Default Not Imported

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

Figure 31: 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.

3. Import Directly

Figure 32: 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.

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.

6. Entire Chat

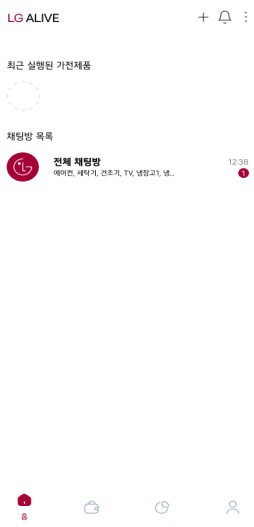


Figure 33: Entire Chat

The system should display the 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

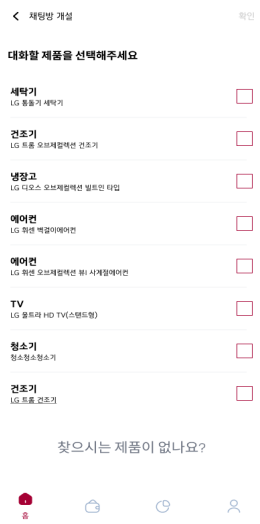


Figure 34: Select None

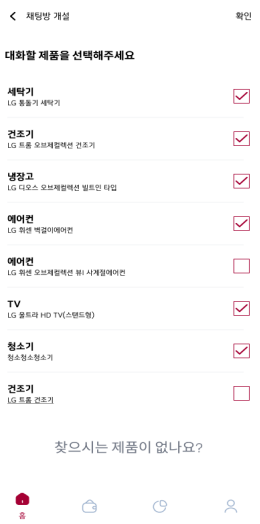


Figure 35: Select Done

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

Figure 36: Set Name

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

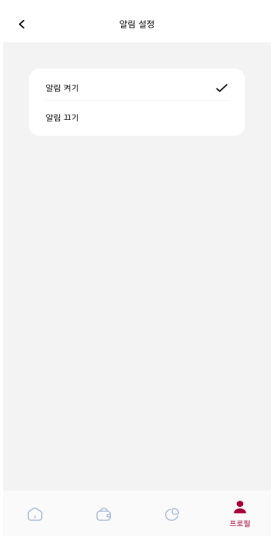


Figure 37: Notice

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

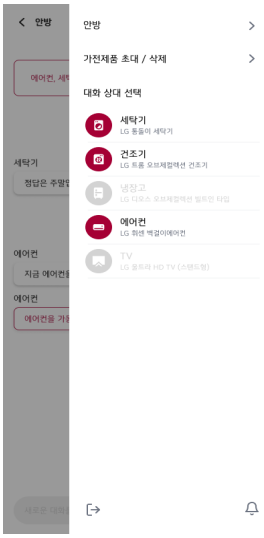


Figure 38: Others-Edit

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

Figure 39: 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

Figure 40: 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

Figure 41: 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

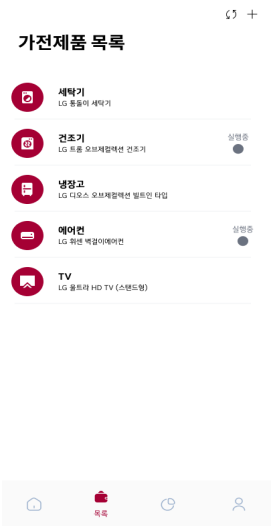


Figure 42: Appliances Default

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

Figure 43:
Appliance 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

Figure 44:
Appliance 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. Appliances-Detail

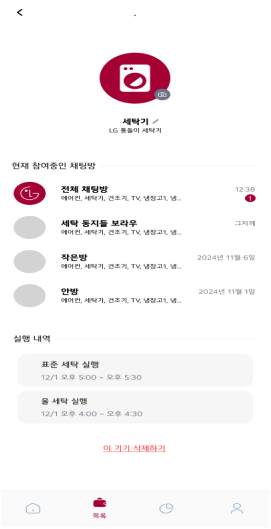


Figure 45: Appliance Detail

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

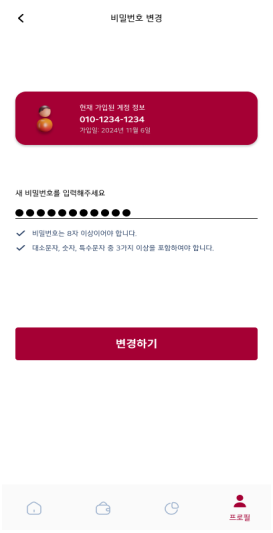


Figure 46:
Password Change

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

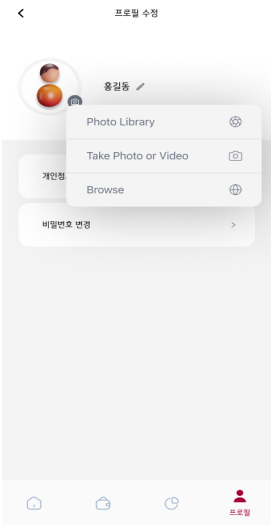


Figure 47:
Profile Update

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

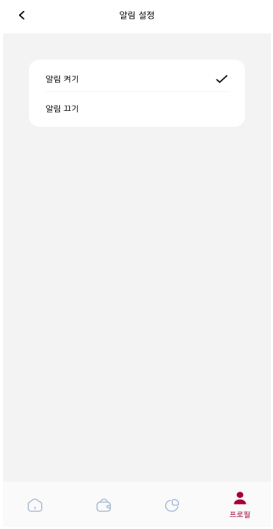


Figure 48: Notification
Settings

Users can enable or disable notifications for app events like messages, updates, and alerts. Preferences are saved immediately.

4. Mypage-App Version Check

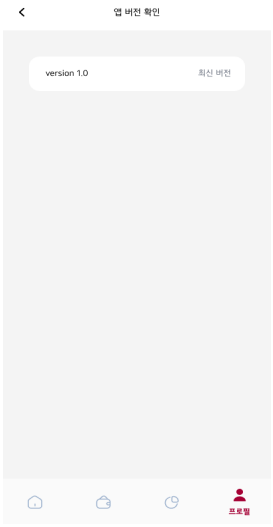


Figure 49: App
Version Check

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

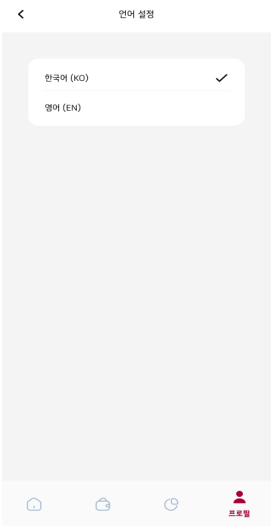


Figure 50:
Language Setting

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

6. Mypage-Personal Information

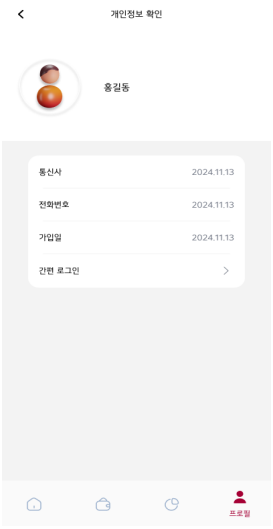


Figure 51:
Personal Info

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

Figure 52:
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

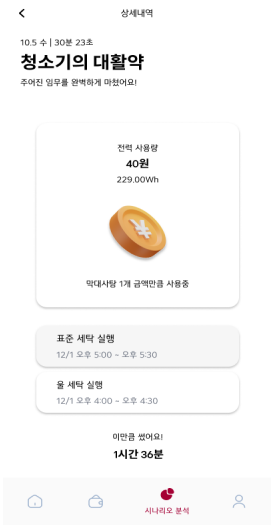


Figure 53:
Result Analyze

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

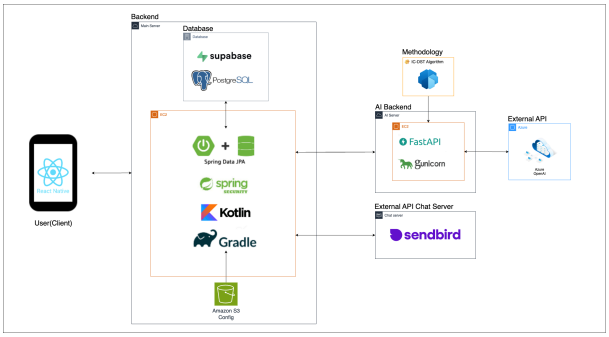


Figure 54: Overall Architecture

ALIVE is composed of four main modules: the Frontend, the Backend (Main Server), the Chat Server, 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 dialog 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" 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 directly with the frontend, 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/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/app/ (tabs)/exeAnalysis	[exeKey].tsx
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/constant	color.constant.ts device.constant.ts typography.constant.ts
ALIVE/frontend/src/utils	calendar.ts date.ts

Table V: DIRECTORY-ORGANIZATION-BACKEND-1

Directory	File Name
ALIVE/backend/src/main/ kotlin/com/example/demo	Application.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ controller	AuthController.kt AIMessageController.kt MessageController ChannelController.kt DeviceController.kt DeviceUsageRecordController.kt UserController.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 MessageService.kt dialogUserService.kt dialogChannelService.kt UserService.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ model	Channel.kt ChannelDevice.kt Device.kt DeviceCategory.kt DeviceUsageRecord.kt User.kt QueryMessagesRequest.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 SendMessageRequest.kt QueryMessageRequest.kt InviteUserRequest.kt CreateChannelRequest.kt CreateChannelRequestDTO.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto/ai	AIMessageRequest.kt AIMessageResponse.kt ContextDTO.kt DialogRequest.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ dto/user	UserCreateRequestDTO.kt UserDTO.kt UserInviteRequestDTO.kt CreateUserRequest.kt CreateUserResponse.kt UserResponseDTO.kt
ALIVE/backend/src/main/ kotlin/com/example/demo/ exception	DeviceNotFoundInChannel.kt Exception.kt DeviceNotInProgressException.kt UserDeviceNotFoundException.kt UserNotFoundException.kt

Table VI: DIRECTORY-ORGANIZATION-BACKEND-2

Directory	File Name
ALIVE/backend/src/main/kotlin/com/example/demo/config	AppConfig.kt AIServerConfig.kt GlobalExceptionHandler.kt SecurityConfig.kt dialogConfig.kt
ALIVE/backend/src/main/kotlin/com/example/demo/util	ApiHelper.kt

Table VII: DIRECTORY-ORGANIZATION-AI

Directory	File Name
ALIVE/AI	requirements.txt
ALIVE/AI/display	monitor.py simulator.py state_manager.py status_viewer.py
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

1) 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.

2) 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.

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

4) 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.

- colors.constant.ts : This is a file containing color-related constants and theme definitions.

- devices.constant.ts : This is a file containing device -related types.

- typography.constant.ts : This is a file containing typography-related constants like font sizes, weight s, and styles.

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

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

D. Module 2: Backend

1) 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 (dialog), 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.

2) 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.

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

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

- 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.

- MessageController.kt : Processes HTTP requests related to messaging functionalities.

- AIMessageController.kt : Handles HTTP requests specifically for AI-driven message processing.

- 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

- repository : Handles database interactions for various entities.

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

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

- 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.

- 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.

- MessageService.kt : Implements business logic for processing and managing messages

- DialogUserService.kt : Manages user-related operations within dialog contexts.

- DialogChannelService.kt : Handles channel-related operations within dialog contexts.

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

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

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

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

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

- InviteUserRequest.kt : Handles data for inviting users to a dialog channel.

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

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

- AIMessageRequest.kt : Defines the data structure for AI message requests sent to the AI server.

- AIMessageResponse.kt : Defines the data structure for AI message responses received from the AI server.

- ContextDTO.kt : Represents the context information used throughout the application's conversational processes.

- DialogRequest.kt : Represents requests related to in itiating or managing dialogs within the application.

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

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

- CreateUserRequest.kt : Represents data for creating a new user in dialog.

- CreateUserResponse.kt : Represents the response data after creating a user in dialog.

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

- 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.

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

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

- AIServerConfig.kt : Manages configuration settings for integrating with the AI server.

- GlobalExceptionHandler.kt : Handles exceptions gl obally across the application.

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

- dialogConfig.kt : Configures integration settings for the dialog chat service.

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

- dialogApiHelper.kt : Provides helper functions for interacting with the dialog API.

E. Module 3: AI

1) 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.

2) 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.

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

4) 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. USE CASE

A. Loading

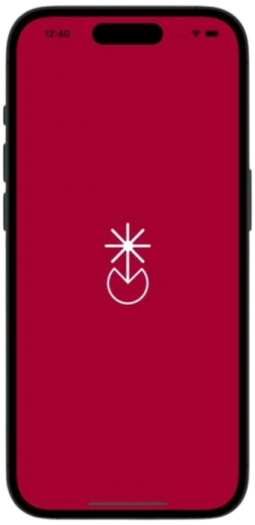


Figure 55: Loading

When the app is launched, an initial loading screen appears. This page is turned on only while the application loads the elements needed to operate, and automatically moves on to the import page at the end of loading.

B. Sign Up

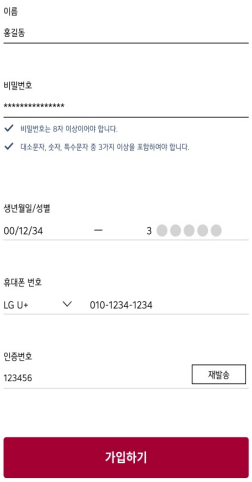


Figure 56: Sign Up

The user enter name, password, birthdate and authenticate their phone number through the telecommunications provider to use it as their ID. After entering information, they can proceed to the login page and attempt to log in with the newly created account.

C. Login

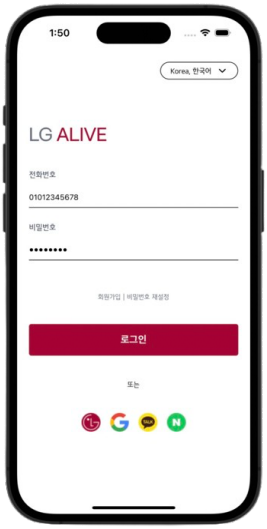


Figure 57: Login

After signing up, the login page is displayed. User then enters their registered phone number and password to log in and access the system.

D. Import

The user can select the Import option to upload data into the system. There are two options available, Import ThinQ and Import Directly.

Figure 58: Import ThinQ

1) Import ThinQ : User can Import product by connecting ThinQ and retrieves registered smart home devices.

Figure 59:
Import Directly

2) Import Directly : User can Import products by scanning a QR code or using Wi-Fi and Bluetooth to detect nearby devices.

E. Main Page

Figure 60: Main Page

After the product import is completed, the user is directed to the main page, where an initial chatroom for all devices is automatically created. On the Main page, users can create a chatroom to manage devices and communicate.

F. Create Chatroom

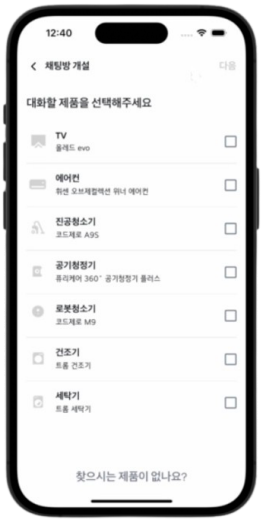


Figure 61:
Create ChatRoom

The user can access and view a comprehensive list of all the devices they currently own. This list is organized to display detailed information, including the category each device belongs to and the specific name of each device.

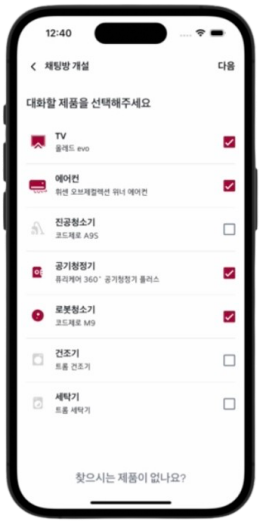


Figure 62: Select
Devices

1) Select Devices : The user chooses devices from the list to include in the chatroom, after which an animation is displayed for the selected devices.

Figure 63: Set
Chatroom Name

2) Set Chatroom Name : The user assigns a name to the chatroom for easier identification

G. ChatRoom

Figure 64: ChatRoom

The user interacts with others and controls devices in a chatroom. Within the chatroom, Upon entering the chatroom, the invited devices are introduced.

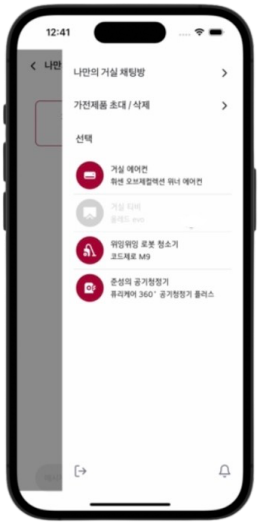


Figure 65: Select Conversation Partner

1) Select Conversation Partner : In chatroom, user selects a device to communicate with in the chatroom. Devices that are not participating in the chat are displayed with a gray indicator to differentiate them.

Figure 66: Chat Message

2) Chat Message : User types a message and sends it through the chat interface. The message format could be direct, indirect, and simultaneous commands, allowing flexible communication with multiple devices through the chat interface.

Figure 67: Execute Device

3) Execute Device : After sending commands through the chat interface, system recognizing the input and ensuring the appropriate devices and execute the specified actions.

H. Devices

The user can manage device information through the system. This allows the user to track and manage records for each device individually.

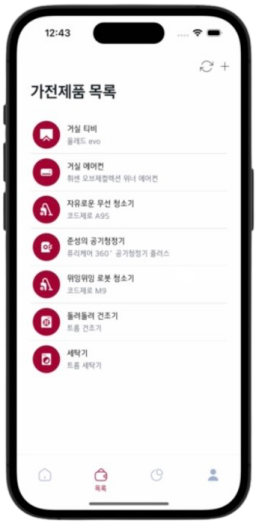


Figure 68: Device List

1) Device List : This page shows a list of all available devices connected to the system.

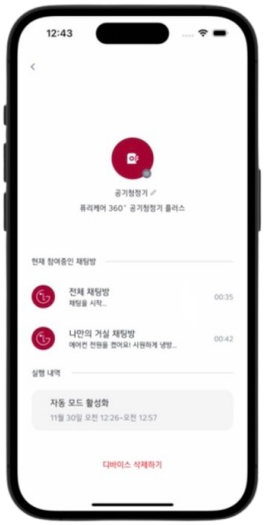


Figure 69:
Device Details

2) Device Details : The user selects a device from the list to view its detailed information, such as participating chatrooms and its activity list.

I. History

The user checks the records of past activities in the system, where they can view the execution history of devices organized by date.

Figure 70: Calender

1) Calender : The first section displayed is the calendar, where dates with execution history are marked with a dot under the date.

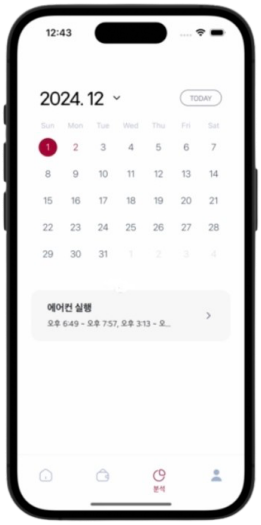


Figure 71: Overview
By Date

2) Overview by Date : When the user clicks on a specific date, the summary of activities for that date appears.

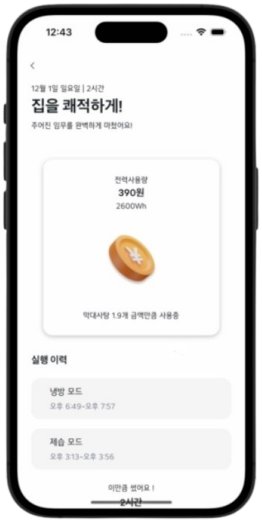


Figure 72:
Detailed Record

3) Detailed Records : User can select a specific activity to see detailed information. Clicking on an activity shown in the Overview by Date page displays details such as power consumption and usage time.

J. Mypage

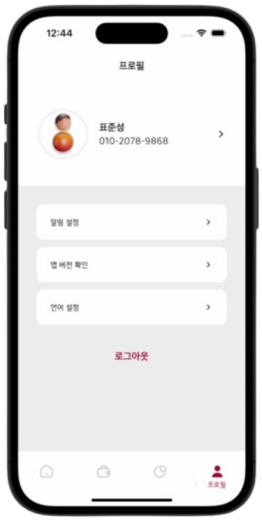


Figure 73: Mypage

Users can manage their personal settings and preferences by accessing the 'My Page' section through the menu bar. On the 'My Page' they can update their profile, personal information and password change along with accessing other personal customization options.

1) Change Profile
   User can update their profile information, such as name or profile picture by clicking profile picture

2) Personal Information
   Users can check their personal information stored in the system, including details such as their mobile carrier, phone number, registration date, and quick login settings.

3) Change Password
   The user updates their password for account security. The password can be changed only if it meets the specified requirements.

VII. AI SPECIFICATION

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

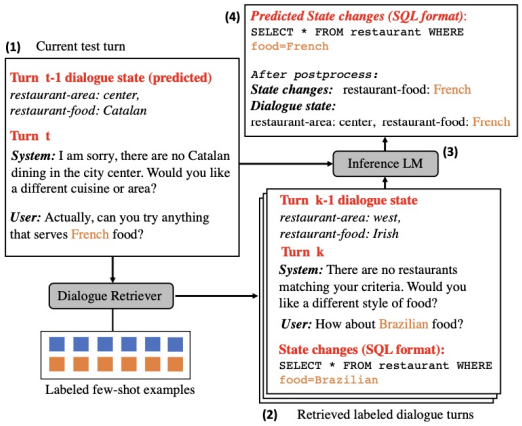


Figure 74:
ICL-DST Architecture

1. Introduction
   Dialogue State Tracking (DST) is a crucial component in dialogue systems, tracking user requirements and guiding actions. Traditional DST approaches often require retraining models whenever a new domain is introduced, leading to high costs and inefficiencies.
   To overcome these challenges, we propose a novel process that applies In-Context Learning (ICL) for DST, leveraging our custom-built dataset, a dialogue retriever for selecting relevant examples, and the powerful pre-trained model, Qwen-Coder 2.5 7B.
   Our approach extends ICL-DST by introducing a dialogue retriever that retrieves labeled examples most relevant to the current dialogue turn. These retrieved examples are then provided to the pre-trained model as in-context examples, enabling efficient adaptation to new domains with minimal retraining. By combining the retriever with in-context learning, we ensure flexible dialogue state prediction and support scalable updates with domain-specific data.

2. Background
   1) 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.

2) 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, enhancing the effectiveness of state prediction.

3) 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.

4) 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

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.

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

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

3. Comparison with multiWOZ
   1) 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.
   2) 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.
   3) 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.

C. Methodology

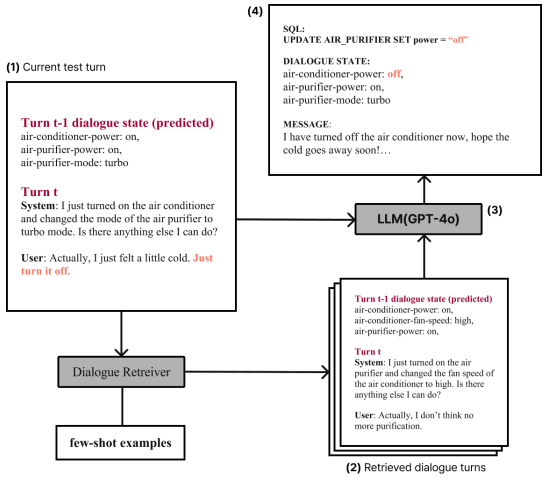


Figure 75:
ALIVE ICL-DST Architecture

Our approach is based on the principles of In-Context Learning for Dialogue State Tracking (IC-DST). By leveraging labeled examples and pre-trained models, we dynamically generate SQL queries and provide natural language responses to user inputs. The process can be summarized as follows:

1. Parsing Current Test Turn
   1) The system processes the user's latest utterance and the system's previous response.
   2) This step provides the context for the current dialogue turn, forming the basis for

understanding the user's request.

2. Retrieving Relevant Examples
   1) Relevant labeled examples are retrieved from a dataset to serve as few-shot prompts for the model.
   2) These examples include the dialogue context, sample system responses, and the corresponding SQL queries.
   3) The retrieved examples guide the model to handle similar requests and produce consistent outputs.

3. Combining the Prompt
   1) The current test turn, labeled examples, and the SQL structure are combined into a single prompt.
   2) This structured prompt ensures the model has the necessary context for accurate inference.

4. Using Pre-trained Model for Completion
   1) A pre-trained model, such as CodeGen, GPT-4o is used to process the prompt and generate SQL queries and natural language responses.
   2) The model's inference aligns with the dialogue context and examples provided.

5. Communication Between Main and AI Servers
   1) The main server sends a structured request to the AI server, which includes dialogue information and updated slot values.
   2) The AI server processes the request, generates an SQL query, and provides a response back to the main server.

D. Language Models

In our approach, we utilize two distinct configurations of GPT-4o: GPT-4o-mini and GPT-4o, each serving a specific role in the system.

1. GPT-4o-mini
   This configuration is employed for dataset generation, ensuring that the examples used for in-context learning are of high quality, contextually accurate, and well-aligned with the task requirements. By leveraging GPT-4o-mini's efficient generative capabilities, we can create diverse labeled dialogue examples while maintaining computational efficiency.

2. GPT-4o
   This configuration is used to generate the final outputs, such as SQL queries and natural language responses. Its advanced architecture is well-suited for producing precise and context-sensitive outputs, enabling accurate real-time inference based on the provided prompts and examples.

Our use of these two configurations aligns with the methodology outlined in the IC-DST paper, where separate models or configurations are strategically employed to optimize different stages of the workflow. GPT-4o-mini facilitates the creation of robust and efficient training data, while GPT-4o ensures the system generates accurate and context-aware outputs during real-time inference.

E. Prompt Examples

1. Table Prompt

Robot Cleaner

```
CREATE TABLE robot_cleaner(
    name text,
    power text CHECK (power IN ('on', 'off')),
    mode text CHECK (mode IN ('ZigZag',
        'detail', 'intense', 'normal'))
);
```

```
3 example rows:
SELECT * FROM robot_cleaner LIMIT 3;
name                                power      mode
LG 코드제로 로보킹 AI 올인원 (프리스탠딩)   ON        Zigzag
LG 코드제로 오브제컬렉션 M9              ON       Meticulous
LG 코드제로 R5                          ON       Focused
```

Air Conditioner

```
CREATE TABLE air_conditioner(
    name text,
    power text CHECK (power IN ('on', 'off')),
    temperature int CHECK (temperature BETWEEN
        16 AND 30),
    mode text CHECK (mode IN ('cooling', '
        heating', 'auto', 'dehumidification'))
        ,
    fan_speed text CHECK (fan_speed IN ('low',
        'normal', 'high', 'auto'))
);
```

```
3 example rows:
SELECT * FROM air_conditioner LIMIT 3;
name                          power  temperature      mode           fan_speed
LG 휘센 오브제컬렉션 위너 에어컨    ON        18         Cooling          High
LG 휘센 벽걸이에어컨              OFF       16           Auto           Normal
LG 휘센 오브제컬렉션 타워I 에어컨   ON        22      Dehumidification     Low
```

Air Purifier

```
CREATE TABLE air_purifier(
    name text,
    power text CHECK (power IN ('on', 'off')),
    mode text CHECK (mode IN ('auto', 'sleep',
        'turbo')),
    fan_speed text CHECK (fan_speed IN ('low',
        'normal', 'high', 'auto'))
);
```

```
3 example rows:
SELECT * FROM air_purifier LIMIT 3;
name                              power   mode     fan_speed
LG 퓨리케어 360° 공기청정기 플러스      ON     Auto      Normal
LG 퓨리케어 오브제컬렉션 에어로타워       ON    Sleep       Low
LG 퓨리케어 오브제컬렉션 에어로퍼니처     OFF   Turbo       High
```

TV

```
CREATE TABLE tv(
    name text,
    power text CHECK (power IN ('on', 'off')),
    volume int CHECK (volume BETWEEN 0 AND
        100),
    channel int CHECK (channel BETWEEN 1 AND
        999),
    mode text CHECK (mode IN ('standard', '
        movie', 'sports', 'game'))
);
```

```
3 example rows:
SELECT * FROM tv LIMIT 3;
name                    power   volume   channel    mode
LG 올레드 evo (스탠드형)      ON      30       15       Standard
LG 올레드 TV (스탠드형)      OFF      0        2        Movie
LG SIGNATURE OLED 8K      ON      45       78       Sports
```

2. System Prompt

You are an AI that conducts natural conversations with users to control home appliances and generates SQL queries based on the dialogue state. Refer to the following instructions to handle the operations of each appliance:

a. Previous Dialogue

- User: User's previous dialogue
- System: System's previous dialogue

b. Generation Rules

Before any appliance function is executed, the power must be turned on first.

1) Context

a. Previous Context: Previous context
b. Based on the previous context, generate the updated list of all slot values affected by the user's utterance.
c. The domain must be one of the following: List of domains.
d. If the user's utterance is unrelated to the previous context, retain the previous context as is.

2) Messages

```
[
    {
        "domain": "appliance domain",
        "message": "Response message from the
                    appliance"
    }
]
```

a. The appliances should respond directly to the user in a personified manner, as if they are speaking naturally. Responses should be structured as follows

3) SQL

- Generate an SQL query that reflects the updated dialogue state.

VIII. DISCUSSION

A. Technical Difficulties

1) Limitations in Dataset Quality

One of the significant challenges we encountered was the limitation of the existing MultiWoz 2.2 dataset, which did not encompass the specific domains required for our app. To address this, we generated a custom dataset using a large language model (LLM) to create initial data samples. Afterward, we conducted a manual labeling process to assign appropriate labels to the data and performed thorough human review to ensure its quality and accuracy. However, this workflow was time-consuming and labor-intensive, highlighting the need for an automated pipeline to streamline the dataset creation, labeling, and validation process. In the future, we aim to utilize ChatThinQ's user data to build an automated system that can dynamically generate and refine datasets tailored to specific domains.

2) Simplifying Actions with Routine Integration

Integrating pre-set routines into the chat system could also be seamlessly enhance user convenience. By enabling users to trigger actions with a simple command like "Start deep cleaning," the process could be greatly simplified, providing a more intuitive and efficient way to execute tasks directly within the chat interface.

B. Non-Technical Difficulties

1) Frontend-Backend Collaboration

During the development process, maintaining seamless communication between the frontend and backend teams proved challenging. While we utilized

Postman to define API endpoints and data formats, real-time communication regarding API updates and JSON format changes was not always efficient. To address this issue, we introduced Jira as a coordination tool to improve task tracking and communication. However, limited familiarity with the tool prevented us from fully leveraging its potential. Through this project, we recognized the critical importance of adopting and mastering collaboration tools like Jira to enhance communication and workflow efficiency in future endeavors.

2) 3d animation creation

During the post-development phase, we planned a promotional video to effectively convey the theme of ”conversations with appliances.” To bring a sense of vitality to the video, we utilized Blender to create 3D animations, aiming to give the impression that the appliances were alive. However, creating these animations presented challenges, particularly in implementing smooth and natural motion sequences. Additionally, syncing the animations seamlessly with other video elements required extensive revisions, significantly increasing the time and resources needed for production. Despite the difficulties, using Blender allowed us to deliver a more engaging and dynamic visual experience.

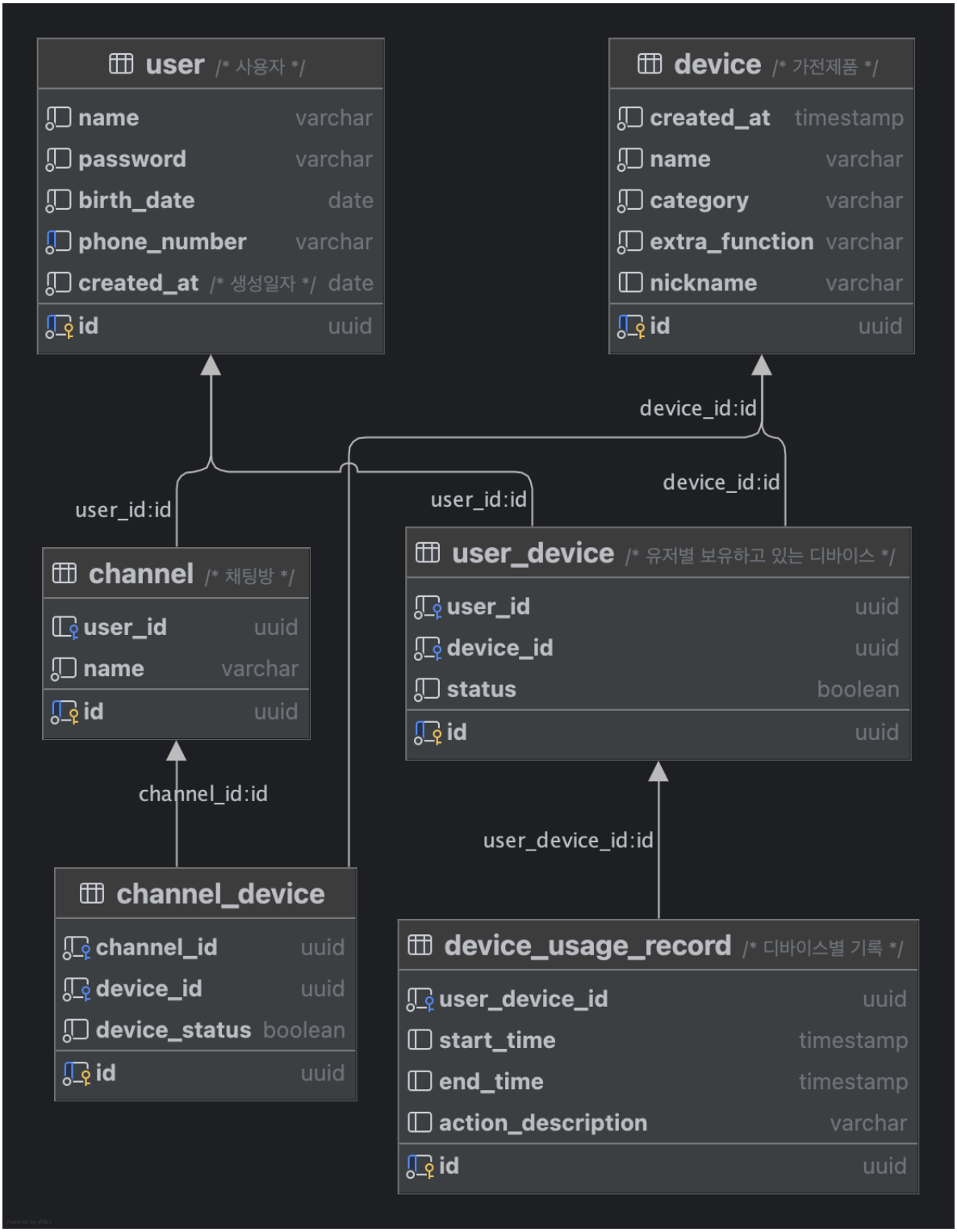
C. Conclusion

At the start of the project, under the topic of ”Implementing Smart Appliance Services with Generative AI,” we focused on two key questions. The first was how to make managing home appliances more convenient, and the second was how to shift AI from being perceived as passive to feeling more active and autonomous. We believed that managing home appliances could be more effective through a service that allows users to communicate with them, similar to having conversations with people.

Over the course of approximately four months, from planning to development, we encountered various challenges, including communication issues, and not everything went as initially intended. However, through these efforts, we successfully developed ALIVE, the device chat application for appliances.

Nowadays, AI technology continues to advance toward creating more intelligent systems, and it will soon become an even deeper part of our everyday lives. This progress will not be limited to external applications but will also profoundly impact home services, driving innovation in smart home systems. At the heart of this transformation will be communication with appliances, enabling a new level of interaction and convenience —a concept brought to life through ALIVE. We look forward to further evolving smart home services through ALIVE.

IX. APPENDIX


user /* 사용자 */
name varchar
password varchar
birth_date date
phone_number varchar
created_at /* 생성일자 */ date
id uuid
device /* 가전제품 */
created_at timestamp
name varchar
category varchar
extra_function varchar
nickname varchar
id uuid
device_id:id
user_id:id
user_id:id
device_id:id
channel /* 채팅방 */
user_id uuid
name varchar
id uuid
user_device /* 유저별 보유하고 있는 디바이스 */
user_id uuid
device_id uuid
status boolean
id uuid
channel_id:id
user_device_id:id
channel_device
channel_id uuid
device_id uuid
device_status boolean
id uuid
device_usage_record /* 디바이스별 기록 */
user_device_id uuid
start_time timestamp
end_time timestamp
action_description varchar
id uuid

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.

[8] Léo Jacqmin, Lina M. Rojas-Barahona, Benoît Favre. *Do you follow me? A Survey of Recent Approaches in Dialogue State Tracking*. Orange Innovation, Aix-Marseille Université, 2023.

[9] Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A. Smith, and Mari Ostendorf. *In-Context Learning for Few-Shot Dialogue State Tracking*. University of Washington, University of Hong Kong, Allen Institute for Artificial Intelligence, 2023.

[10] ”Sendbird Chat SDK,” https://sendbird.com/docs/chat/sdk/v4/android/getting-started/send-first-message