PocketHome: AI-driven Environmental Optimization Through Multi-User Preference Mediation

Siwoong Lee
Dept. of Information System
Hanyang University
Seoul, Korea
bluewings02@hanyang.ac.kr

Jaebeom Park
Dept. of Information System
Hanyang University
Seoul, Korea
tony0604@hanyang.ac.kr

Yuanjae Jang
Dept. of Information System
Hanyang University
Seoul, Korea
semxe123@gmail.com

***Abstract*—'PocketHome' is an AI-based automated control system that creates an optimal environment for multiple people at once. It works by combining user preferences, observed behaviors, and real-time environmental data from sensors in the space. Using this information, the system automatically adjusts shared appliances like heating/cooling systems, air purifiers, and lighting. The AI's main task is to find a balance that keeps the largest number of people comfortable and satisfied. The system's core technology, reinforcement learning, treats any manual adjustments by users as feedback, allowing it to continuously improve how it operates. This learning process reduces the need for people to make changes themselves.**

***Index Terms*—Reinforcement Learning, Optimal Environment, Multi User, User Satisfaction**

I. Role Assignments

TABLE I
Role Assignments

Role	Name	Task Description
User	Siwoong Lee	The primary end-user who provides their unique User ID and personal environmental preferences (temperature, humidity, illumination) via the mobile app. Their preference data and any manual overrides are the core inputs for the AI model.
Customer	Siwoong Lee	The sponsoring entity (e.g., building owner, facility operator) who funds the project. Defines the key objectives (occupant satisfaction, energy efficiency) and approves the project scope and its integration with the building's infrastructure.
Software Developer	Jaebeom Park	The technical expert responsible for building the complete 'PocketHome' system as defined in the requirements, including the mobile app (iOS/Android), central database, synchronization APIs, and local presence-detection protocols.
Development Manager	Yuanjae Jang	The project lead responsible for managing the development plan, team, and risks. They ensure the 'Data Utilization (AI Core)', including MOP and RL models, is successfully implemented to meet the Customer's objective of balancing multi-user comfort.

II. Introduction

A. Motivation

At home, we effortlessly control temperature, humidity, and lighting via smartphones. Yet, we lose this personalized control in shared spaces like offices and libraries. This raises

a key question: “Why can we manage our home environment meticulously, but not the public spaces where we spend most of our time?” This discrepancy highlights a significant technological gap in which individual comfort is sacrificed for one-size-fits-all management.

This environmental mismatch becomes a personal burden. Individuals in shared spaces often resort to passive measures, such as adding clothing or changing seats, when uncomfortable. This directly impacts not only their comfort but also their focus and productivity. We must therefore move beyond the current, unresponsive centralized control model to a new paradigm that can intelligently accommodate diverse and individual needs.

B. Problem Statement

The fundamental flaw in shared environmental control is the requirement of enforcing a single setting on multiple individuals. This one-size-fits-all system cannot resolve the conflicting preferences that inevitably arise. Current feedback mechanisms, such as manual complaints, are inefficient, slow, and do not mediate these diverse needs. This leads directly to decreased comfort, reduced productivity, and energy waste. The core problem is the absence of an automated mechanism to intelligently mediate these conflicting demands and derive a balanced, optimal environment in real time.

C. Related Service

Siemens Comfy: A leading workplace experience platform. It enables employees to adjust temperature and lighting in their immediate vicinity through an app. It also includes features like booking meeting rooms or desks. Its acquisition by Siemens highlights the trend of merging with smart building technology.

HqO: An integrated app platform that building owners provide to their tenants. It extends beyond simple environmental control to improve the overall ’work-life experience,’ including booking building amenities, visitor registration, and viewing local information and announcements.

Honeywell Forge: An enterprise-level solution from Honeywell for optimizing building operations. Integrates data from various systems (HVAC, security, fire safety) for AI analysis to enable prediction of energy consumption and predictive maintenance. Its focus is more on operational efficiency than on employee-facing convenience.

LG Electronics B2B Integrated Solutions: LG provides B2B smart office solutions focused on central air conditioning (HVAC), digital signage, and meeting technologies. The key distinction is that these solutions have not yet integrated LG’s consumer home appliances to provide personalized convenience for individual employees.

III. Requirements

A. User Roles

- **General User:** Any individual present in a space where ’PocketHome’ is installed. They register their unique ID via the app and transmit their personal environmental preferences to the system in real-time.

B. Account Management

- **User Identification:** User enters their unique `User ID` once, and can delete their profile with `User ID`. After registration, the app stores the `User ID` on the device, allowing the user to transmit preferences without logging in each time.

C. Core Features (Single Screen Interface)

- **Single Screen UI:** All core functions (ID verification, preference input) are provided on a single main screen. It displays `User ID`, `Temperature`, `Humidity`, `Illumination (Brightness)` those can be manually changed, and other preferences such as `Optional Biometric Data Consent` button and so on.

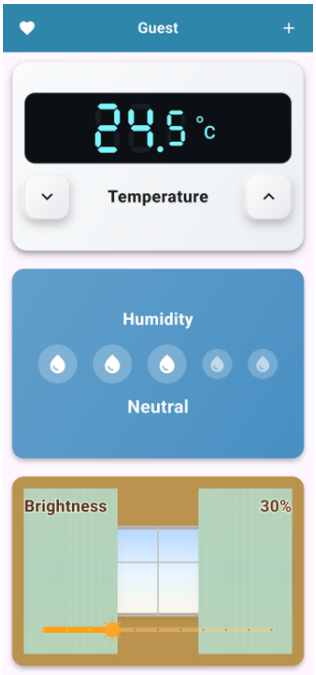

Fig. 1. Main Screen

D. Optional Data Integration

1) Biometric Data:

- **Heart Rate Sensor:** With user consent, integrates with the smartphone’s heart rate sensor (on compatible devices).
- **Stress Index:** With user consent, integrates with the smartphone’s stress index sensor (on compatible devices).
- **Real-time Transmission:** When activated, periodically transmits the average value of each heart rate and blood oxygen every 5 minutes.

2) Personal Profile Attributes:

- **Optional Input:** Allows users to voluntarily provide additional personal attributes, such as MBTI type, self-assessed sensitivity to cold/light, or general work patterns (e.g., "Focus work", "Collaborative work").
- **Data Utilization:** This supplementary, non-biometric data can be used by the AI model as another feature to find correlations and improve the accuracy of its satisfaction predictions (e.g., "Users with MBTI type 'INXX' may correlate with a preference for lower illumination").

E. System & Space Integration

1) Presence Detection:

- **Network Communication:** When a user enters a specific 'PocketHome' enabled zone, their smartphone begins communication with the local control device(Bluetooth Advertisement).

2) Preference Transmission:

- **Interval-based Communication:** While in space, the smartphone transmits its `User ID` and current preference data to the local space system at regular intervals (e.g., every 5 minutes). This allows the space system to identify in real-time which users (User IDs) with which preferences are currently present in this space.

F. Data Utilization (AI Core)

1) Central AI Model (Multi-Objective Optimization):

- **Data Aggregation:** The central AI system queries the database to retrieve two key data sets: (1) All users' registered preferences (from Data Synchronization) and (2) The list of `User IDs` currently present in a specific zone (from Preference Transmission).
- **Optimization Goal:** The AI model (e.g., using Genetic Algorithms or MORL) processes this aggregated data. Its goal is to compute the single optimal environmental setting (temp, humidity, illumination) that best satisfies a defined objective function (e.g., the 'Max-Min' principle, maximizing the satisfaction of the least-satisfied user).
- **Control Command:** The resulting optimal setting is then sent as a command to the building's control system (e.g., HVAC, lighting controllers) for that specific zone.

2) Reinforcement Learning Feedback Loop:

- **Monitoring for Manual Overrides:** The system must log any manual adjustments made by users (e.g., via a physical thermostat or an admin override) after the AI has set an environment.
- **Model Refinement:** A manual override is treated as strong negative feedback ("penalty"). The AI's reinforcement learning component uses this feedback to retrain its models, refining its understanding of true user preferences and discomfort thresholds to avoid this outcome in the future.

G. Technical Requirements

- **Platform:** Mobile application (Android, iOS).
- **Database:** A central, real-time database capable of receiving and updating all user `User IDs` and their changing preference data.
- **Network:**
  - *Central:* Stable internet connection (Wi-Fi or Cellular) for database updates.
  - *Local:* Local communication protocol (Bluetooth Advertisement) for the detection and transmission of user presence within a space.
- **Privacy & Consent:**
  - *Initial Consent:* A clear notification and consent process is mandatory upon `User ID` registration regarding the collection of preference and optional data.

IV. Development Environment

A. Software Development Platform

- **Windows**

  Windows 11 was selected as the primary developer operating system due to its robust and high-performance support for the project's essential toolchain. Its main role is to host Android Studio, enabling the stable compilation, debugging, and emulation of the native Kotlin application. This environment allows for efficient build management via Gradle and simultaneously supports the local development and testing of the Python-based AI Core services before their eventual deployment to the Firebase cloud.

B. Programming Language

- **Dart**

  Dart was chosen as the primary language, utilized through the Flutter framework, because it is Google's modern toolkit for building natively compiled applications for mobile, web, and desktop from a single code base. This approach significantly accelerates development and ensures a consistent user experience across platforms. The Dart language itself promotes safer and more concise code through features like robust null safety.

- **Python**

  Python was chosen to power server-side Data Utilization (AI Core) because it is the definitive industry standard for the complex AI modeling this project requires. Its primary role is to execute the Multi-Objective Optimization (MOP) and Reinforcement Learning (RL) models that form the system's intelligence. This decision leverages Python's unparalleled ecosystem of mature libraries, such as Pymoo for optimization and TensorFlow Agents for reinforcement learning, which are essential for processing aggregated user data. This stack enables a

flexible architecture where heavy model training (using feedback logs) is performed offline, while the resulting optimized models are deployed to a serverless environment (like Google Cloud Run) for efficient real-time inference.

C. Software in Use

- **Visual Studio Code**

  Visual Studio Code was selected as the development environment due to its unparalleled integration with the Dart language and Flutter framework, providing a lightweight, fast, and highly-optimized workflow. Its best-in-class support for stateful Hot Reload is critical, as it dramatically accelerates the development and debugging cycle. This choice is central to our cross-platform strategy, offering robust tooling to efficiently build, test, and manage the Platform Channels required for deep native integration, thereby ensuring reliable access to core hardware-centric features like the Bluetooth LE (BLE) API for Presence Detection and the Health Connect API for Biometric Data Transmission from a single, unified codebase.

- **Flutter**

  Flutter was chosen for its ability to deliver a high-fidelity, consistent UI across both Android and iOS from a single codebase. Its rich, declarative widget system allows for deep customization and rapid prototyping. This speed is accelerated by Hot Reload, enabling instant testing and refinement of the Preference Input Dashboard. This method ensures a polished user experience and avoids duplicating design efforts for each platform.

- **Firebase**

  Firebase was selected as the backend stack due to its direct alignment with the project's core requirements and its seamless integration with the native Kotlin app. Its Cloud Firestore component provides the essential real-time database capability specified in the Data Synchronization requirement, allowing for instantaneous preference updates from the app with minimal latency. Furthermore, Firebase's serverless architecture, through Cloud Functions, is ideal for executing the Data Utilization (AI Core) logic, enabling scalable, event-driven inference for our MOP/RL models. This comprehensive platform, combined with the cost-free prototyping offered by the Spark Plan and a robust Kotlin SDK, makes it the most rapid and efficient solution for development.

- **Google Cloud Compute Engine**

  Google Cloud Compute Engine was selected to provide the high-performance computing infrastructure necessary for the intensive computational loads of the MOP/RL models. Unlike serverless environments, it offers full control over virtual machine configurations, enabling the specific GPU acceleration and custom runtime environments required for deep learning tasks. This choice ensures the stable and scalable execution of the AI Core while maintaining low-latency connectivity with the Firebase database, guaranteeing real-time responsiveness for complex inference operations.

- **FastAPI**

  FastAPI was adopted to establish a high-performance model serving layer within Google Cloud Compute Engine, enabling direct deployment of MOP/RL inference capabilities to client terminals. Its native support for asynchronous I/O is critical for efficiently managing concurrent requests without blocking the computational resources required by the underlying Python-based models. This framework provides a lightweight, standardized REST interface that bridges the AI Core with the external network, ensuring low-latency data transmission and reliable API contracts for the end-user devices.

D. Cost Estimation

- **Software Costs: $0.** All primary development tools are free (Flutter, Git, VS Code).
- **Cloud Costs (Prototype): $100.** We will utilize the **Firebase (GCP) Free Tier (Spark Plan)** and **Google Cloud Compute Engine**. This plan provides sufficient capacity for the `Cloud Firestore` database (real-time data sync) and `Virtual Machine` (API/AI logic) needed for development and initial testing.
- **Hardware Costs: $0.** Development will use existing laptops (PC/Mac) and Android test devices.

E. Development Environment Details

- **Workstation OS:** Windows 11 (Host for Android Emulator & Development)
- **IDE:** Visual Studio Code
- **Framework & Language:** Flutter (Dart SDK 3.x), Python 3.10+
- **Version Control:** Git (Repository: `LeeSiwoong/PocketHome`)
- **Target Platform:** Android (Min SDK 26, Target SDK 34/35), iOS (Latest)
- **Backend & AI Infrastructure:**
  - **BaaS:** Firebase (Firestore, Auth, Functions)
  - **Compute:** Google Cloud Compute Engine (GPU-accelerated instances for AI models, Ubuntu)
- **Key Libraries & Dependencies:**
  - **Flutter Client (Dart):**
    - `firebase_core` / `cloud_firestore`: For app initialization and real-time NoSQL database sync.
    - `flutter_blue_plus`: For Bluetooth Low Energy (BLE) scanning and 'Presence Detection'.
    - `http`: For asynchronous REST API requests to the Python AI Core.

    - `permission_handler`: For managing runtime permissions (BLE, Health, Location).
  - **AI Core (Python/FastAPI):**
    - `FastAPI / Uvicorn`: For high-concurrency ASGI server implementation.
    - `sklearn`: For random forest regressor.
    - `NumPy`: For data preprocessing and manipulation.

F. Software in Use (Existing Algorithms)

We leverage established, high-performance libraries to ensure system reliability and scalability. The client application is built using the **Flutter SDK**, utilizing **`flutter_blue_plus`** for BLE-based presence detection and **`firebase_core`** for real-time data synchronization. For the Python AI Core, we utilize **scikit-learn** to implement the Random Forest Regressor for user preference prediction and **NumPy** to execute a custom Genetic Algorithm for multi-user environmental optimization. These models are exposed via **FastAPI** and hosted on **Google Cloud Compute Engine** to handle real-time inference requests efficiently.

G. Task Distribution

TABLE II
ROLE ASSIGNMENTS

Role	Name	Task Description
Frontend Developer	Siwoong Lee	The Frontend Developer constructs the cross-platform mobile application using Flutter and Dart, implementing the user interface for ID management and environmental controls while handling local Bluetooth Low Energy (BLE) scanning for presence detection.
Backend Developer	Jang Yuanjae	The Backend Developer manages the server-side infrastructure using Firebase and Google Cloud Compute Engine to ensure real-time data synchronization between the app and the central database.
AI Developer	Park Jaebeom	The AI Developer engineers the system's core intelligence using Python, deploying Multi-Objective Optimization and Reinforcement Learning models to aggregate user data and calculate optimal environmental settings.

V. SPECIFICATIONS

Technology Stack: Kotlin (Android App), Firebase (Backend/DB), Python (AI Core)

A. User Roles

- **User Identification:** When the app is opened for the first time, the user must enter a unique `User ID`. It must not be in the database. If the user wants to delete or change `User ID`, it can be done by clicking the ID in ID section in main screen. The app must not run if there is no valid ID on user's app.

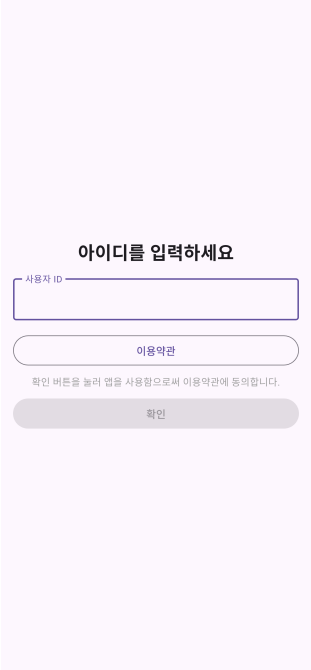

Fig. 2. ID Initialization

Fig. 3. ID Section

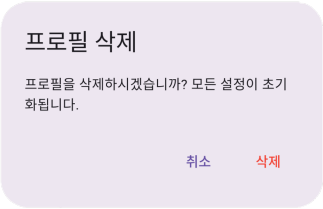

Fig. 4. ID Deletion Pop-up

B. Core Features

- **Send Modified Data:** Automatically send modified data when the user changes any of them.

C. Environmental Preference Controls

- **Temperature:**
  - *Control:* Stepper control.
  - *Range:* 18.0°C ~ 28.0°C.
  - *Increment:* Adjustable in units of 0.5°C by clicking buttons.

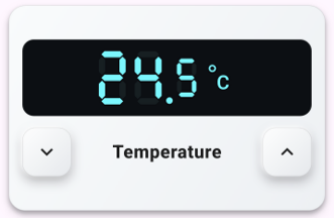

Fig. 5. Temperature Controller

- **Humidity:**
  - *Control:* 5-step segmented button group.
  - *Steps:* 5 levels (e.g., “Very Dry”, “Dry”, “Neutral”, “Humid”, “Very Humid”).

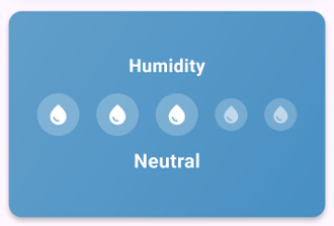

Fig. 6. Humidity Controller

- **Illumination (Brightness):**
  - *Control:* Continuous slider.
  - *Range:* 0% (Darkest) ˜ 100% (Brightest) in units of 10

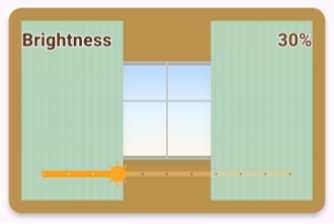

Fig. 7. Illumination Controller

D. Optional Data Integration

- **Optional Biometric Data Consent:** A heart shaped icon in the top-left corner opens a pop-up window where users can consent to the collection of optional biometric data. Pop-up window can be closed by clicking outside of it.

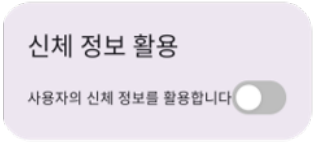

Fig. 8. Biometric Data Consent Pop-up

- **Optional Profile Access:** A + (plus) button in the top-right corner. Tapping this button opens an overlay screen (modal) where users can input or delete their optional ‘Personal Profile Attributes’ (e.g., MBTI, sensitivity to cold). Overlay screen can be closed by clicking outside of it.

Fig. 9. MBTI Enter Overlay

E. System & Space Integration

1) Presence Detection & Preference Transmission:

a) Platform:: Flutter (Dart - Background Isolate/Service)

b) Description:: Detects current zone via BLE scan using platform channels and periodically updates user’s presence/preferences to the `presence_logs` collection.

```
FUNCTION startService() async:
    // 1. Start BLE scan for 'PocketHome_BEACON_UUID'
    // Using a stream listener for continuous scanning
    FlutterBluePlus.startScan(
        withServices: [ PocketHome_BEACON_UUID ]
    );

    FlutterBluePlus.scanResults.listen((results) {
        onScanResult(results);
    });

    // 2. Schedule periodic timer (e.g., 1 min)
    Timer.periodic(Duration(minutes: 1), (timer) {
        transmitPresence();
    });
END FUNCTION

// 3. BLE Scan Callback (Stream Listener)
FUNCTION onScanResult(List<ScanResult> results):
    // logic to find the closest beacon
    if (results.isNotEmpty) {
        currentZone = results.first.device.name; // e.g.,
              Zone_A
    }
END FUNCTION

// 4. Function that runs every 1 minute
```

```
FUNCTION transmitPresence() async:
    final userID = await loadUserIDLocally();

    IF currentZone == null: // User is not in a zone
        // (Optional: log out event)
        FirebaseFirestore.instance
            .collection( presence_logs )
            .doc(userID).delete();
        RETURN;

    // 5. Update my info in the current zone's log
    // Accessing app state or local storage for prefs
    final currentPrefs = appState.currentUserPreferences;

    final Map<String, dynamic> presenceData = {
         zone : currentZone,
         preferences : currentPrefs,
         lastSeen : Timestamp.now()
    };

    // Overwrite(set) doc with {userID} as the ID
    await FirebaseFirestore.instance
        .collection( presence_logs )
        .doc(userID).set(presenceData);
END FUNCTION
```

F. Data Utilization (AI Core)

Note: This section is Python code running on the Google Cloud backend, processing data uploaded by the Flutter app.

1) Central AI Model (MOP Inference):

a) Platform:: Python (Google Cloud Run / Cloud Function)

b) Description:: Triggered every 5 minutes. Aggregates preferences of users in a zone, 'infers' the optimal setting using a pre-trained model, and sends the command to the HVAC system.

```
FUNCTION calculate_optimal_environment(event, context):
def calculate_optimal_environment(zoneID= Zone_A ) : # 0.
     Load the MOP model trained locally model =
     load_model ( mop_model.pkl )

# 1. Aggregate preferences of users in 'Zone_A'
db = firestore.client()
users_in_zone = db.collection ( presence_logs ) \
                  .where ( zone ,  == ,  zoneID).get()

user_preferences = []
for doc in users_in_zone:
    user_preferences.append(
        doc.to_dict().get ( preferences ) )

IF len(user_preferences) == 0:
    RETURN # No one is in the zone

# 2. 'Infer' the optimal solution
optimal_setting = model.predict(user_preferences)
# e.g., { temp : 23.5,  humidity : 3}

# 3. Send command to the building API
HVAC_API.set_temperature(
    zoneID, optimal_setting[ temp ]
)
LIGHTING_API.set_brightness(
    zoneID, optimal_setting[ illumination ]
)

# 4. (For feedback loop) Record AI's setting in DB
db.collection ( zone_status ) .document(zoneID).update({
     ai_setting : optimal_setting,
     timestamp : NOW()})

END FUNCTION
```

VI. Architecture Design & Implementation

A. Overall Architecture

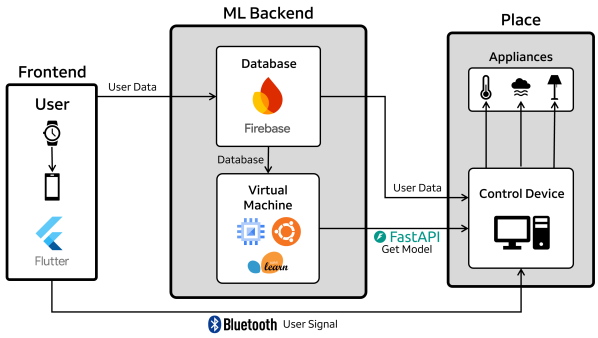

Fig. 10. PocketHome Architecture Diagram

PocketHome system has 3 main modules: Frontend, ML Backend and Place. Frontend is for normal users, ML Backend is for database and virtual machine that makes model in cloud environment, and Place is for certain places like cafe, classroom, or subway.

Frontend contains smart phone and wearable devices such as smart watches. wearable devices send user's health information in case of when user agreed to utilize theirs. The smart phone sends user's data to two directions: database and control device by Flutter. It sends user preferences like which environment the user prefers and user's health information(optional) to database, and signal of user presence to control device using Bluetooth advertising so that control device can get those users' information from database.

In ML Backend, there are database and virtual machine. Database saves entire users' information, and sends all data to virtual machine in regular period. When virtual machine receives the data set from Firebase database, it makes the optimal AI model which can create single best optimized environment. Virtual machine is constructed on Ubuntu of Google Cloud, and AI model is made by scikit learn of Python.

Place module contains control device and appliances. They varies on different places, so basically there are no same places for this system. Once control device get user presence signal, it calls that user's information from database. This control device calculates the single best optimized environment by using AI model. The local AI model is regularly updated through FastAPI every period. After calculation, control device automatically modifies setting values of each appliance.

B. Directory Organization

1) BackendAIModel:

Directory	File Names
/BackendAIModel	virtualMachineConfig.txt weight_model_server.py

2) App:

Directory	File Names
/App/.github	copilot-instructions.md
/App/.vscode	settings.json
/App/android	build.gradle.kts gradle.properties gradlew gradlew.bat local.properties settings.gradle.kts
/App/docs	FIREBASE_SETUP.md
/App/ios/Runner	AppDelegate.swift
/App/lib/models	user_settings.dart
/App/lib/screens	main_screen.dart splash_screen.dart user_id_input_screen.dart
/App/lib/services	bluetooth_id_broadcaster.dart body_metrics_service.dart
/App/lib/widgets	_outlined_text.dart app_name_header.dart brightness_control.dart delete_conformation_dialog.dart humidity_control.dart info_dialog.dart mbti_input_panel.dart temperature_control.dart
/App/lib	firebase_database_config.dart main.dart
/App	.metadata analysis_options.yaml devtools_options.yaml pubspec.lock pubspec.yaml

3) Document:

Directory	File Names
/Document	PocketHome.pdf PocketHome.tex

4) PlaceEndHost:

Directory	File Names
/PlaceEndHost	end_host.py

5) Others:

- **PocketHomeDiagram.png:** This png file shows the architecture of PocketHome system.
- **README.md:** This README file shows how AI-driven optimization model is constructed.

C. Module Description

1) Frontend:

- **Purpose**

  The Frontend Module of PocketHome helps individual users to modify their own preferences. It is delivered as an application form, and users can interact with it by clicking some button on the screen. When each parameter is changed, the application sends that data to database on cloud simultaneously.

- **Functionality**

  It provides user ID initialization, user ID deletion and user-based preferences. User-based preferences consists many parameters such as preferred temperature, preferred humidity, preferred brightness, user's MBTI(optional) and user's real time health information. When user changes any value of them, it also sends its timestamp of that time.

- **Location of Source Code**

  https://github.com/LeeSiwoong/PocketHome/tree/main/App

- **Class Component**

  - build.gradle.kts: It defines build configuration of project.
  - gradle.properties: It controls the performance settings of the Gradle build system.
  - settings.gradle.kts: It makes Flutter project can build Android version.
  - AppDelegate.swift: It is the entry point for the iOS version application.
  - GeneratedPluginRegistrant.h: It is the Objective-C Header file for the plugin registration system.
  - GeneratedPluginRegistrant.m: It is the plugin manifest of iOS application.
  - user_settings.dart: It sends user preferences to database and makes application to remember them.
  - main_screen.dart: It shows users main screen, which enables users can modify their preferences.
  - splash_screen.dart: It is shown for 2 seconds when user starts the application.
  - user_id_input_screen.dart: It is shown when user initially runs the application or after the user deletes ID.
  - bluetooth_id_broadcaster.dart: It enables the smart phone can send signal of user presence by broadcasting Bluetooth advertisement.
  - body_metrics_service.dart: It enables users to give permission to system to utilize their real time health information.
  - app_name_header.dart: It shows buttons for pop-up menus and user ID.
  - brightness_control.dart: It provides user a brightness controller.
  - delete_confirmation_dialog.dart: It shows a pop-up screen of user ID deletion when user clicks the user ID.
  - humidity_control.dart: It provides user a humidity controller.
  - info_dialog.dart: It enables user to permit for their real time health information utilization.

- mbti_input_panel.dart: It provides user MBTI entering side bar.
- temperature_control.dart: It provides user a temperature controller.
- firebase_database_config.dart: It gives app the information of database that should be connected.
- main.dart: It controls the main flow of the PocketHome application.

2) ML Backend:

- **Purpose**
  The ML Backend Module of PocketHome stores users' preferences to database on cloud. Database is managed in a cloud environment, and the virtual machine brings the database to make the best AI Model. After building an AI model, it spreads the model to end hosts of places. Also it gives end hosts the user's information based on user presence from places.

- **Functionality**
  It stores user's preferences in form of NoSQL on Firebase. Firebase provides the database to virtual machine in Google Cloud. Virtual machine works on Ubuntu, and it periodically gets database to build a best model for mediation of user preferences. Model is trained by scikit learn of Python. After building a model, the model is delivered to end host of each place by FastAPI.

- **Location of Source Code**
  https://github.com/LeeSiwoong/PocketHome/tree/main/BackendAIModel

- **Class Component**
  - virtualMachineConfig.txt: It helps configurating virtual machine.
  - weight_model_server.py: It is the code for training the model.

3) Place:

- **Purpose**
  The Place Module of PocketHome is for actual performance for each place. Every end host for every place works separately, but the main logic is the same due to the model downloaded from the main single server. It collects the user presence and calls their information from database.

- **Functionality**
  It defines if a particular user is in that place by using Bluetooth advertisement every 5 minutes, and get their user preferences from database. Device that performs this mechanism is called control device, and it must be connected to network to get the information from cloud. Also, it computes the best single optimized environment by running a model, which is downloaded from virtual machine.

- **Location of Source Code**
  https://github.com/LeeSiwoong/PocketHome/tree/main/PlaceEndHost

- **Class Component**
  - end_host.py: It is the code for downloading json file on control device.

VII. Use Case

A. Loading

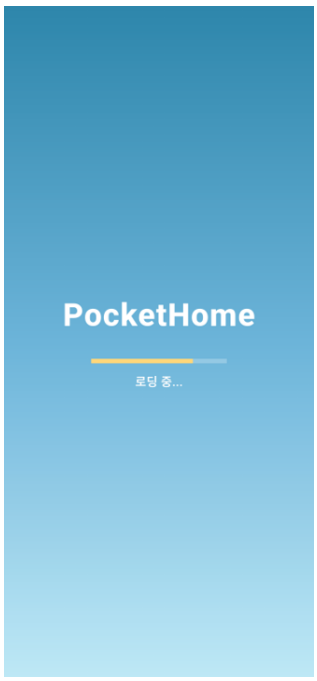

Fig. 11. Splash Screen

When the application is launched, a splash screen is shown for 2 seconds. This screen helps to wait the application to be fully loaded. Also, the application asks users to enable Bluetooth usage function to interact with place.

B. ID Initialization & User Agreement

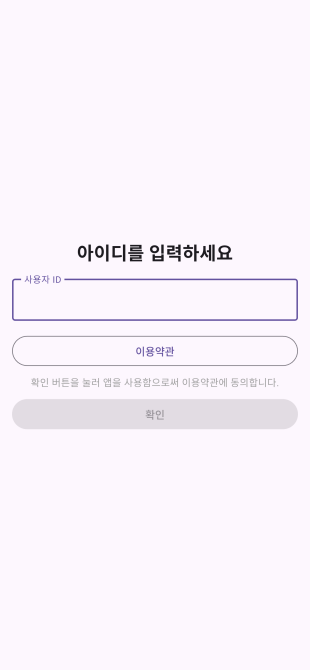

Fig. 12. ID Initialization Screen

This screen is shown only when user launches the application for the first time, or when the user deletes the ID which triggers the deletion of user information. The user enters the own unique ID. If there exists the same ID in database, user cannot continue to the main page. If the user clicks the button of user agreement, a text box appears that shows the requirements for the users to use this application. After entering the main screen, this screen does not appear until the ID is deleted.

C. Main Screen

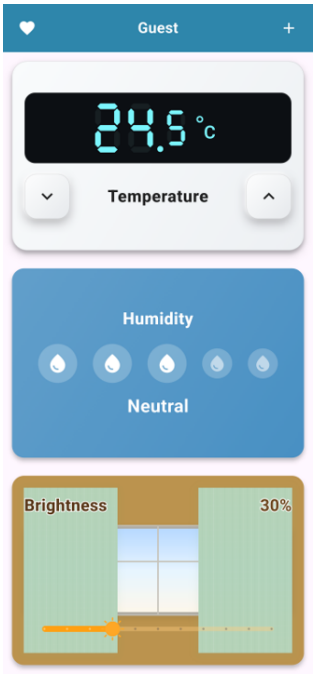

Fig. 13. Main Screen

This screen shows the parameters that user can control. When the user changes one of them, the modified data is sent to the database so that the place can utilize that user's preferences information. By clicking heart-shaped icon and plus-shaped icon, user can agree to send their health information and MBTI information for each.

D. User ID Deletion

Fig. 14. ID Section

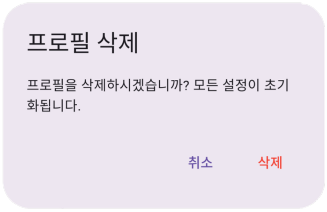

Fig. 15. ID Deletion Pop-up

This screen is for user ID deletion. If users do not want to use this application anymore, users can delete their information by clicking the user ID in ID section. When the user clicks the user ID, an ID Deletion Pop-up is shown, and ther user can delete their information from database and application by clicking delete button.

VIII. Discussion

This project was the first experience to make an application for all teammates. It was difficult to decide how to separate frontend and backend, and somehow we made it. However, there were still some complicated parts, so we tried to find easy answers on the Internet. In addition, constructing a cloud environment for machine learning was hard to achieve because the configuration was completely different from what we have done. Communication issues were also an issue of making this system in terms of understanding, simple miscommunications, and time to mail to each other.

Especially the making of an ML model for the best single environment and the connection of the application to the database were the two major problems of this project. The goal of this project is to find the best single environment for many people, and it is an unusual situation compared to other common situations. Therefore, how to make an algorithm for this multi-value data set using supervised learning was a most important quest.

Connecting the application and the database was done in Flutter to simplify the code. That was simple, but managing Firebase database was another problem. This was our first time using the NoSQL database, so we searched various documents to get used to it.