Speaker-Adaptive Voice Assistant for Multi-User Homes

Ken Lee
Dept. of Information Systems
Hanyang University
Seoul, Republic of Korea
ceh1502@hanyang.ac.kr

Tae Hee Kim
Dept. of English Language and Literature
Hanyang University
Seoul, Republic of Korea
pieceofmind@hanyang.ac.kr

Patrick Segedi
Dept. of Computer Science
Chalmers Unviersity of Technology
Gothenburg, Sweden
segedi@chalmers.se

Jinseo Hong
Dept. of Information Systems
Hanyang University
Paju, Republic of Korea
h0dduck@hanyang.ac.kr

Nick Ki Gumann
Dept. of Information Systems
Zurich University of Applied
Zurich, Switzerland
gumannic@students.zhaw.ch

Abstract—*In modern smart homes, voice assistants like Alexa and Google Home serve multiple family members but treat everyone the same way. This application combines speaker recognition with speaker-adaptive processing to create a personalized experience for each household member. The system identifies who is speaking, determines their location within the house, and enforces access rules based on whether the user is a parent or a child. To protect user privacy, all voice data is processed locally on the device rather than being transmitted to external servers. When multiple users issue conflicting commands, the system resolves them according to predefined rules and user priority levels. Our goal is to develop a voice assistant that genuinely understands and adapts to the needs of each family member while ensuring the privacy of their conversations.*

Index Terms—*Voice Assistant, Speaker Recognition, Multi-User Personalization, Smart Home, Privacy*

Name	Roles	Task description and etc
Ken Lee	Software Developer (AI)	Responsible for integrating AI features into the voice assistant system. Develops AI models for speaker recognition and intent understanding based on user interaction data, ensuring the system adapts to individual user needs. Focuses on using AI to enhance the product, providing personalized responses and automating voice commands while maintaining a user-friendly experience.
Nick Ki Gumann	Software Developer (Back-end)	Manages the server-side infrastructure, ensuring the backend systems function smoothly. This involves working with databases for user profiles and preferences, optimizing data queries, and ensuring data integrity for the voice assistant application.
Patrick Segedi	Software Developer (Pipeline)	Develops the core voice processing pipeline, including audio stream handling, speech recognition integration, and command execution logic. Implements multi-user management systems, conflict resolution mechanisms, and role-based access control. Ensures efficient communication between voice assistant components and smart home devices while maintaining system security and reliability.
Hong Jinseo	Software Developer (Front-end)	Handles front-end development and user interface design. Focuses on creating a seamless user experience by designing intuitive interfaces for voice assistant configuration and monitoring. Implements the interface using React or similar frameworks, ensuring the application is visually appealing, functional, and easy to navigate across different devices.
Tae Hee Kim	Project Manager, UI/UX Designer	Oversees the development process, managing schedules, ensuring product quality, and coordinating the team to meet user requirements and expectations. Additionally handles UI/UX design, creating voice interaction flows and visual feedback systems that make the multi-user voice assistant intuitive and accessible for all family members.

1 Introduction

1.1 Motivation

In the rapidly evolving world of smart home technology, three key elements – personalization, privacy, and contextual intelligence – are shaping the way users experience their connected environments. As smart devices become more widespread in households around the world, people now expect more than just automation or convenience. They want technologies that can recognize their individual habits, respond to their unique preferences, and adjust seamlessly to different situations within their daily lives. This growing demand highlights the importance of creating smart home systems that not only function efficiently but also feel intuitive, secure, and personally meaningful to each user.

As a global leader in home electronics, LG is driving innovation in smart living through its ever-expanding LG ThinQ platform, which integrates a wide range of connected home devices. Despite this progress, most current voice assistants rely heavily on cloud-based systems and deliver generic, uniform responses. This reliance on generic responses creates a gap between current capabilities and LG's goal of creating AI that personally understands individuals and adapts intelligently to their specific situations.

In multi-user households, several people interact with the same smart devices. Yet, current assistants often fail to distinguish between users or adjust their responses based on the surrounding context, leading to impersonal interactions. To bridge this critical gap and support LG's vision of "Innovation for a Better Life," our team proposes developing the SmartER Speaker. This system will recognize each user, ensure privacy through on-device processing, and ultimately create a more seamless and personalized smart home experience.

1.2 Problem Statement

Existing voice assistants still lack effective multi-user personalization. Although technologies exist to distinguish between individual users, current voice assistants rarely apply them effectively. This limitation leads to generic responses or actions that fail to cater to each other's unique needs. Additionally, current systems depend heavily on cloud-based processing. This raises privacy concerns because personal data, such as voice patterns and behavioral information, is transmitted to external servers. In households with multiple users, this leads to functional and ethical problems: users may be misidentified, commands may conflict, and sensitive data could be exposed.

Moreover, current assistants have limited abilities to provide personalized content and support. For example, they cannot easily deliver age-appropriate content for children, offer tailored reminders or assistance for seniors, or adjust recommendations based on individual preferences and routines. They also struggle with understanding context over time, maintaining continuity in multi-step interactions, and integrating smoothly with multiple devices or ecosystems. Accessibility challenges further limit their usability for diverse user groups, and reliance on cloud processing can introduce both security and reliability concerns.

Therefore, smart home environments need a more advanced voice assistant. It should be able to recognize each user, understand the context of interactions, and provide personalized content. At the same time, it must protect privacy and operate reliably across multiple de-

vices. Such an assistant would not only make daily life more convenient but also improve the overall intelligence, safety, and accessibility of the connected home. To address these needs, we aim to develop a Speaker-Adaptive Voice Assistant that meets all these requirements.

2 Requirements

A. Mobile Application

The system requires a mobile application that can securely manage user access and permissions through a role-based framework. Users can create an account by registering with their phone number, which is verified through an authentication process, and by setting a strong password that meets the required security standards. Once registered, each user is assigned a specific role that determines their level of access within the system. The administrator can create and manage user profiles, update information, and adjust permissions as needed. For example, a child user can be assigned a "kid" role, ensuring the voice assistant recognizes the user type and provides suitable content or responses. Regular users may have access to general system features, while guests are limited to basic viewing options. This structured approach allows for personalized and secure interaction, ensuring that each user's experience and data protection are properly maintained.

B. Voice Assistant & Services

The system's voice assistant must be capable of recognizing and distinguishing users based on their voices. By analyzing the speaker's voice, it can determine which user profile is currently interacting with the system and respond accordingly. For example, when the assistant detects the mother's voice, it should identify it as belonging to the "mother" profile and automatically adjust the environment to her preferences—such as changing the living room lighting to her preferred setting or playing her favorite genre of TV shows. Similarly, when it detects a child's voice, the system should recognize it as the "child" profile and apply appropriate restrictions, such as denying a request to purchase games or preventing access to adult content. At the same time, it can provide kid-friendly content, ensuring a safe and personalized experience. Through this intelligent voice recognition, the system creates a customized and secure environment for every user.

3 Development Environment

A. Choice of software development platform

a. *Platforms*

Our development environment will consist of Windows and macOS, as we have prior experience and proficiency with these systems. In this prototype, the backend server and AI speaker client are both implemented and tested locally on these machines using FastAPI (Python) and MySQL, before being deployed to a cloud environment in the future.

1) Windows: Our team selected Windows 11 as one of the primary development platforms due to its broad compatibility, stable performance for AI and embedded

development, and robust ecosystem support. It also integrates seamlessly with our main software, including Python, PyTorch, and Fast-Whisper.

2) Mac: We also selected macOS as a primary development platform for several important technical, design, and cross-platform reasons. MacOS meets our project needs by providing a stable, Unix-based foundation, powerful developer tools, and seamless support for our AI and design workflows. It functions well with all our core software, including Python, PyTorch, and Fast-Whisper, as well as Windows.

Our team used devices as follows:

1. Asus A14, AMD Ryzen 7 8845HS w/ Radeon 780M Graphics(3.80 GHz)

2. MacBook Air, M3 chip, 16 GB, macOS Sequoia 15.7.1.

3. ASUS Zenbook 14 Flip OLED, 16 GB, Intel Core i7-1360P, Windows 11 Home 24H2

b. Programming Languages

1) Python: Offering both simplicity and power, Python provides robust libraries and models for high-performance audio processing, speech recognition, and machine learning. Its key speaker verification model, SpeechBrain, is built on PyTorch, leveraging this Python-native framework for the efficient design and operation of deep neural networks.

2) SQL / MySQL: Instead of using SQLite, the current implementation uses MySQL as the relational database management system. SQL is used to define and query relational data such as users, devices, and zones. In the codebase, MySQL is accessed through SQLAlchemy from the FastAPI backend.

3) JavaScript (React): The web-based dashboard and OTT interface are implemented using React and Vite, allowing users and administrators to manage devices, zones, profiles, and to visualize the results of voice-based commands in real time.

B. Software in Use

1) SpeechBrain: SpeechBrain is an open-source speech toolkit built on PyTorch, designed specifically for speech processing tasks. Our project incorporates the ECAPTA-TDNN (Emphasized Channel Attention, Propagation and Aggregation in TDNN) model. This model is pre-trained on the VoxCeleb dataset for robust speaker verification. By extracting speaker embeddings from audio, the model enables highly accurate speaker authentication using voice biometrics. The toolkit's modular design and comprehensive documentation make it ideal for implementing robust voice authentication systems.

2) Faster-whisper: Faster Whisper is an optimized implementation of OpenAI's Whisper automatic speech recognition model. It uses CTranslate2 for efficient inference, providing up to 4x speedup compared to the original implementation while maintaining the same accuracy.

In our system, it handles the speech-to-text conversion for wake word detection, accurately transcribing spoken commands to identify activation phrases.

3) NumPy: NumPy serves as the fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. In our audio processing pipeline, NumPy is essential for handling three core tasks: manipulating and transforming audio signals, extracting features from waveforms, and performing mathematical operations for similarity calculations.

4) PyTorch: PyTorch is a deep learning framework that provides tensor computation with strong GPU acceleration and deep neural networks built on a tape-based autograd (automatic differentiation) system. While we do not directly use PyTorch for model training, it serves as the backend for SpeechBrain's speaker recognition models. Its dynamic computational graph and Pythonic nature make it ideal for research and production deployment.

5) Soundfile: Soundfile is a Python library that interfaces with libsndfile to handle reading and writing audio formats, as WAV, FLAC, and OGG. In our project, it handles reading pre-recorded voice samples, writing recorded audio to disk, and converting audio formats.

6) SoundDevice: SoundDevice provides bindings for the PortAudio library, and it allows real-time audio recording and playback. It offers low-latency audio I/O with minimal dependencies. Our system uses it for real-time audio recording from a microphone.

7) Visual Studio Code: Visual Studio Code serves as our primary integrated development environment. Its lightweight architecture, combined with powerful features, makes it ideal for Python development.

8) Notion: Notion functions as our all-in-one workspace for project documentation and knowledge management.

9) Slack: Slack serves as our primary team communication platform, facilitating real-time collaboration

10) FastAPI: FastAPI is a modern, high-performance web framework for building APIs in Python. In our system, it replaces the earlier Node.js/Express backend design and exposes REST endpoints for user management, device control, and movie search. The AI speaker sends HTTP requests to FastAPI (e.g., /voice-search, /devices/{id}), and the React web dashboards consume the same API for visualization and control.

11) SQLAlchemy + MySQL: SQLAlchemy is used as the Object-Relational Mapping (ORM) layer on top of a MySQL database. It defines Python models for users, voice_profiles, zones, and devices, and automatically creates and updates the corresponding tables. This allows the system to persist user profiles, speaker voice samples, and smart-home device states in a relational database.

12) Uvicorn: Uvicorn is an ASGI server used to run the FastAPI application. It enables asynchronous request handling and hot reloading during development.

13) React + Vite: React is used to build the web-based dashboards for both the smart-home device control interface and the OTT movie browsing interface. Vite provides a fast development server and build tool. Together, they allow

administrators and end-users to view and manipulate the state of devices, zones, and age-restricted movie search results in real time.

C. Task Distribution
Will be provided in the next phase.

4   Specifications

A. Requirement 1 – Mobile Application
a. Sign Up: Users can create an account and access features by signing up through the mobile app. Once signed up, their permissions determine access to additional capabilities, such as profile management and device linking.

1) For first-time usage, you must use the provided master key to access the app and create the admin profile.

2) Enter phone number, which will be verified through an authentication system.

3) Enter a password that meets the following criteria:
: Be at least 8 characters long.
   : Include a mix of lower- and upper-case letters.
   : Contain at least 1 number.
   : Contain at least 1 special character.
   Prototype implementation note.
While the original requirement specified a mobile application, the current prototype implements the same functionality as a web-based dashboard using React. The web UI still supports user sign-up, login, role management (admin, user, guest), and device/zone management, and it communicates with the FastAPI backend in the same way that a future mobile app would.

b. Profile Management: User profiles with admin rights have access to the mobile app via their login credentials. They can create new accounts and fully manage existing ones.

c. Profile Role Restrictions: There will be three different roles in our system.

1) Admin: Users assigned the "Admin" role can create and manage other profiles, and they can access every single feature of the system.

2) User: Users assigned the "User" role can access the system's general features, but they cannot create or manage other profiles. This is the only role that can be customized, for example, to accommodate special requirements like an underage user.

3) Guest: Users with the "Guest" role can only use limited "read" functions.

B. Requirement 2 – Speaker
a. Wake word detection with voice recognition: One of the main features of the system is Wake Word Detection with Voice Recognition. This function allows the assistant to activate when a specific keyword is spoken and then identify the speaker through voice recognition. Once identified, the system grants access to personalized features and delivers responses tailored to the individual user.

b. Voice commands

The system allows users to interact naturally through voice commands, enabling hands-free control of smart home devices and personalized services.

For the prototype, the system will demonstrate three main functions.

1. Opening and locking the door
2. Turning on and off the TV
3. Changing the light in the room

In the future, the system can be easily expanded to include more features such as alarm setting, media playback, environmental control, and child-specific modes. This flexibility allows the assistant to grow and adapt to different household needs.

c. Full Access / Limited Access: It allows users to set different access levels for each account. Parents can set their accounts to access all the commands and contents. They can also set up their children's accounts to prevent access to adult content or inappropriate content.

d. Speaker Adaptive & Location Awareness: This function allows the assistant to understand the environment and location in which it operates. The system will include multiple speakers placed in different rooms. This enables detection of where the user is speaking from and responds accordingly, creating a more seamless and speaker-adaptive experience.

5 Architecture Design & Implementation

A. Overall Architecture
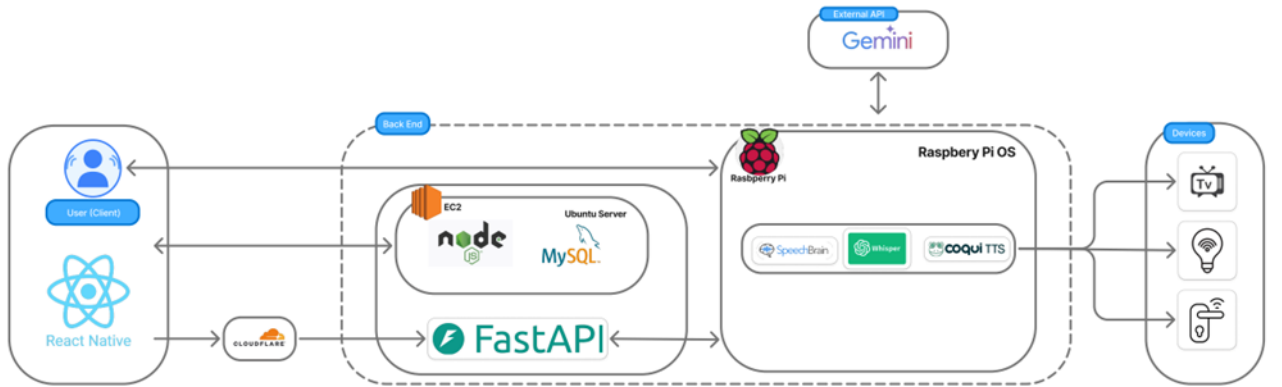
This smart speaker system features a distributed architecture with clear separation of concerns. The Raspberry Pi serves as the local processing unit, running SpeechBrain for speaker verification, faster-whisper for speech-to-text, and Coqui TTS for text-to-speech synthesis. All voice processing happens locally to ensure privacy and low latency.

The AWS EC2 instance hosts the backend infrastructure with Node.js server and MySQL database managed through Aiven cloud platform. The FastAPI framework handles API endpoints and communication between components. Cloudflare acts as the security and CDN layer, protecting and routing traffic to the system.

Google Gemini provides external AI capabilities for natural language understanding and response generation, integrating with the local voice processing pipeline. The React Native

serves as the user interface, allowing users to interact with the system and monitor device status.

The system workflow processes voice locally on Raspberry Pi, authenticates users through SpeechBrain, sends verified requests to the EC2 backend via FastAPI, queries Gemini for AI responses, and delivers audio feedback through the local TTS system. This architecture ensures both privacy (local voice processing) and powerful AI capabilities (cloud-based Gemini integration).

B. Directory Organization

Directory	File Names	Module Names in Use
/.vscode	c_cpp_properties.json, launch.json, settings.json	N/A (Configuration files)
/Document	Document.aux, Document.fdb_latexmk, Document.fls, Document.pdf, Document.synctex.gz, Document.tex, SmartER_Speaker.docx	N/A (Documentation files)
/src/	main.py, smarterspeaker.app	smarterspeaker.app
/src/backend/utility/model	RestrictionDbo.py, UserDbo.py	dataclasses, typing
/src/backend/utility/repositories	RestrictionDboRepository.py, UserDboRepository.py	json, pathlib, typing, model.RestrictionDbo, model.UserDbo
/src/backend/utility/repositories/test	test_user_repository.py	unittest, sys, pathlib, repositories.UserDboRepository
/src/smarterspeaker	__init__.py, app.py, config.py, users.JSON	pathlib, json
/src/smarterspeaker/Users	__init__.py, admin_user.py, guest_user.py, restricted_user.py, user.py	abc, typing
/src/smarterspeaker/speaker	__init__.py, audio_to_text.py, speaker_active.py, speaker_verification.py, tts.py, voice_recorder.py, wake_word_activation.py	speechbrain, sounddevice, soundfile, torch, numpy, os, pathlib, playsound, pyttsx3
/src/smarterspeaker/voice_samples	jiseong/voices (voice files), kun/voices (voice files), patrick/voices (voice files), taehee/voices (voice files)	N/A (Audio data files)
/src/smarterspeaker/voices	invalid.mp3	N/A (Audio files)
/ (root)	.gitignore, Introduction.md, README.md, SETUP.md	N/A (Project documentation)

C. Module 1: Voice Recorder (voice_recorder.py)
1) Purpose

The Voice Recorder module captures real-time audio input from a microphone

and produces a clean, standardized waveform for subsequent wake-word and speaker-verification modules. Low-latency acquisition is necessary for the system's responsiveness when users issue voice commands.

2) Functionality

Function	Description
Real-time audio capture	Streams audio from microphone input
Fixed-length sampling	Records three second segments per activation attempt
Standardization	Converts input into mono, 16 kHz sample rate
Export	Store audio as .wav file for downstream modules

3) Source Code Location
/project/src/audio voice_recorder.py
4) Implementation Components

| Component	Description
sounddevice	Handles live microphone stream
soundfile	Writes audio into .wav format
numpy	Buffer manipulation and preprocessing
Internal methods	record_audio(), save_audio(), normalize_audio()

5) External References

The recorder was implemented using the sounddevice official documentation as a reference. No pre-built library for full recording logic was used, ensuring full control over latency and buffer size.

6) Rationale for Use

Most cloud-based voice assistants depend on continuous streaming, which increases latency and privacy risk. For SAVA, collecting short, strictly local audio segments provides the following:

- Low-latency activation

- Minimal RAM and storage requirements

- Strict privacy (raw audio remains local only)

7) Graphical Representation
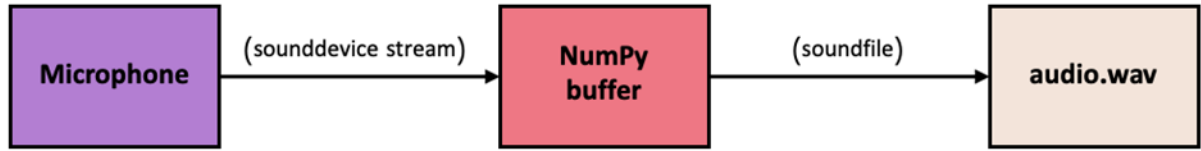
D. Module 2: Wake Word Detection

1) Purpose

The Wake Word Detection module continuously analyzes recorded audio and determines whether the activation keyword "Hello" is present. It ensures that speaker verification and smart home actions occur only after explicit user intent.

2) Functionality

Function	Description
Speech-to-Text conversion	Converts input waveform to text
Keyword spotting	Checks for presence of "Hello" in output transcription
Noise tolerance	Designed to operate in home-noise conditions
Trigger logic	Activates subsequent modules upon detection

3) Source Code Location
/project/src/audio audio_to_text.py
/project/src/audio wake_word_activation.py
4) Implementation Components

Component	Description
faster-whisper	Primary STT inference engine
soundfile	Lightweight model optimized for latency
Internal methods	transcribe_audio(), detect_awake_word()

5) External References

The module uses the Faster-Whisper implementation of OpenAI's Whisper model for transcription, sourced from the HuggingFace Hub. This version provides up to 4× faster inference compared to the default Whisper. This is critical for real-time interaction in resource-constrained environments.

6) Rationale for Use

Alternative wake word systems require custom dataset collection and offline keyword training. Whisper-based recognition eliminates dataset preparation and supports the following:

- Multilingual interaction

- Robust performance with accents/dialects

- High-speed compatible with Raspberry Pi deployment targets

7) Graphical Representation
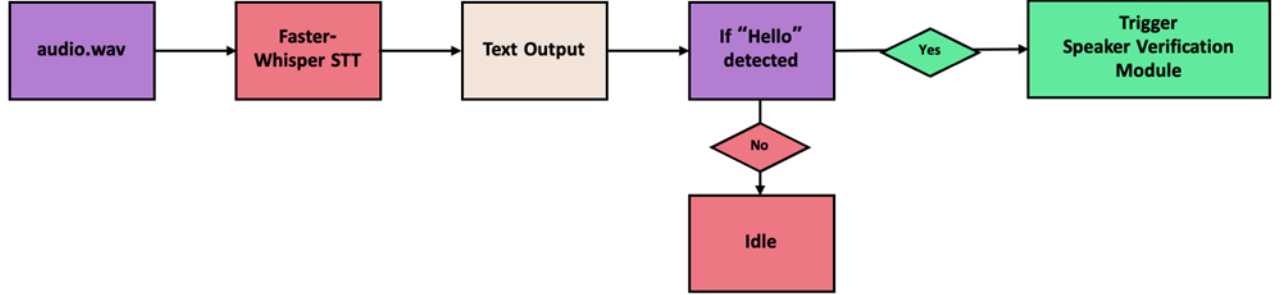
E. Module 3: Speaker Verification
1) Purpose

The Speaker Verification module identifies which family member is speaking by extracting a speaker embedding from the user's voice and comparing it to the stored embeddings in the database. This enables role-based access control (RBAC) in multi-user households.

2) Functionality

Function	Description
Embedding extraction	Converts 3-second audio to 192-dimensional ECAPA-TDNN speaker embedding
Similarity matching	Computes cosine similarity against registered embeddings
Identify inference	Returns user ID and confidence score
Security threshold	Grants access only when similarity ≥ 0.30

3) Source Code Location
/project/src/speaker speaker_verification.py
4) Implementation Components

Component	Description
speechbrain	Speaker embedding generation
torch	Deep learning backend
Model	speechbrain/spkrec-ecapa-voxceleb (HuggingFace)

| Component	Description
Internal methods	extract_embedding(), match_speaker(), compute_similarity()

5) External References

The ECAPA-TDNN model used in this module was downloaded from the HuggingFace Hub. It is trained on VoxCeleb, which contains over 7,000 speakers, leading to state-of-the-art performance with 0.80% EER (Equal Error Rate).

6) Rationale for Use

Rule-based or MFCC classification approaches perform poorly across accents and family environments. ECAPA-TDNN is selected because it offers the following:

- High robustness to age, noise, and recording conditions

- Small model size suitable for home devices

- Real-time processing capability

7) Graphical Representation
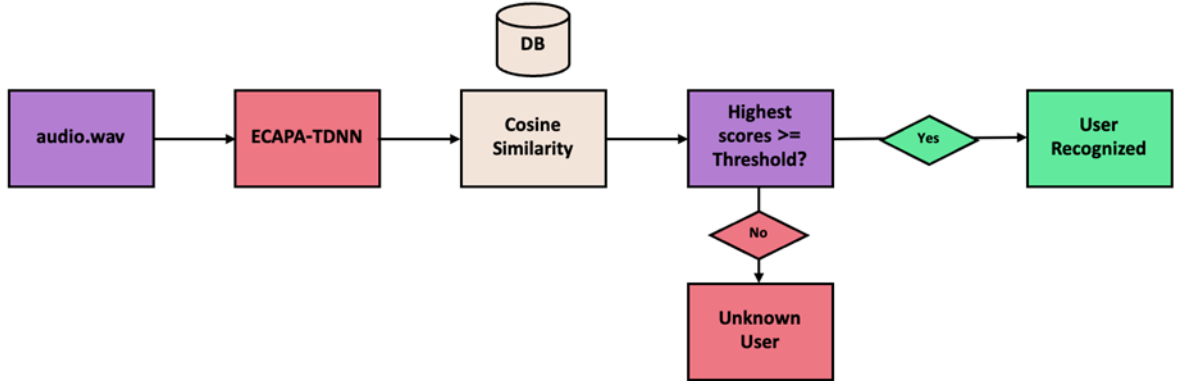
F. Module 4:FastAPI Backend & Database

1) Purpose
The FastAPI backend provides a central API layer that connects the AI speaker client, the web frontends, and the MySQL database. It implements endpoints for movie search, age restriction checking, user and device management, and smart-home control.

2) Functionality

- Exposes REST endpoints such as /voice-search, /movies, /devices, /zones, and /users.

- Handles age-based access control by comparing the authenticated user's age with the ageRating of requested movies.

- Persists and retrieves user, device, and zone information in MySQL via SQLAlchemy.

- Provides a single source of truth for both the AI speaker and the React dashboards.

3) Source Code Location
/app.py, /src/smarterspeaker/api.py, /src/smarterspeaker/db.py, /src/smarterspeaker/models.py, /src/smarterspeaker/schemas.py

4) Implementation Components

- FastAPI: Defines API routes and request/response models.

- SQLAlchemy: Maps Python classes to MySQL tables and handles queries.

- Uvicorn: Runs the ASGI application during development.

5) Rationale for Use
FastAPI provides type-safe, high-performance APIs in Python, which integrates naturally with the existing audio and AI modules. Using SQLAlchemy and MySQL offers a scalable and cloud-ready storage layer for multi-user smart-home deployments.

G. Module 5: Web Dashboards for Smart Home and OTT Integration

1) Purpose
The web dashboards allow users and administrators to visualize and control the system through a browser. One dashboard focuses on smart-home device management, and the other focuses on OTT-style movie browsing with age-aware search.

2) Functionality

- Admin dashboard (speaker-web): log in, view and edit users, assign roles, add/remove zones, register devices, and toggle device states.

- OTT frontend (ott-frontend): search movies, display search results, and show a banner when a voice-triggered movie request is blocked due to age restrictions.

- Periodically polls the /devices and /voice-search endpoints to keep the UI synchronized with the backend and the AI speaker activity.

3) Source Code Location

- /speaker-web/src/. . .

- /ott-frontend/src/. . .

4) Implementation Components

- React + Vite for SPA rendering

- Fetch/axios for HTTP communication with FastAPI

- Custom CSS modules for consistent theming across pages

5) Rationale for Use
A web-based interface is faster to prototype and easier to demo in a classroom setting than a native mobile app, while still satisfying the original functional requirements (account management, device control, and monitoring).

6 Use Cases

A. Use Case 1: Web Application – Basic Features

Use Case	Functionality	Initial state	Input	Output
Register system	Creating a new user account	Logged out: Register page	Master key (for admin access), email, password	Registration successful
Login	Reading login information	Logged out: Login page	Email, Password	Move to dashboard page.
Dashboard	Navigate to Dashboard	Logged in	Click on the "Dashboard" button when on another page.	Move to dashboard page.
Profile	Navigate to Profile	Logged in	Click on the "Profile" button when on another page.	Move to profile page.
Turn on/off lights	Turn on/off lights in a room.	Logged in: Dashboard page	Click on the "Turn on/off" button under specific room light.	Changes light status displayed on app. Turns on/off light in the specific room.
Turn on/off TV	Turn on/off TV in a room.	Logged in: Dashboard page	Turn on/off button under specific room TV.	Changes TV status displayed on app. Turns on/off light in the specific room.
Unlock/lock the main door.	Unlocks/locks the main door.	Logged in: Dashboard page	Click on the "Unlock/Lock" button under main door.	Changes door status displayed on app. Turns on/off light in the specific room.

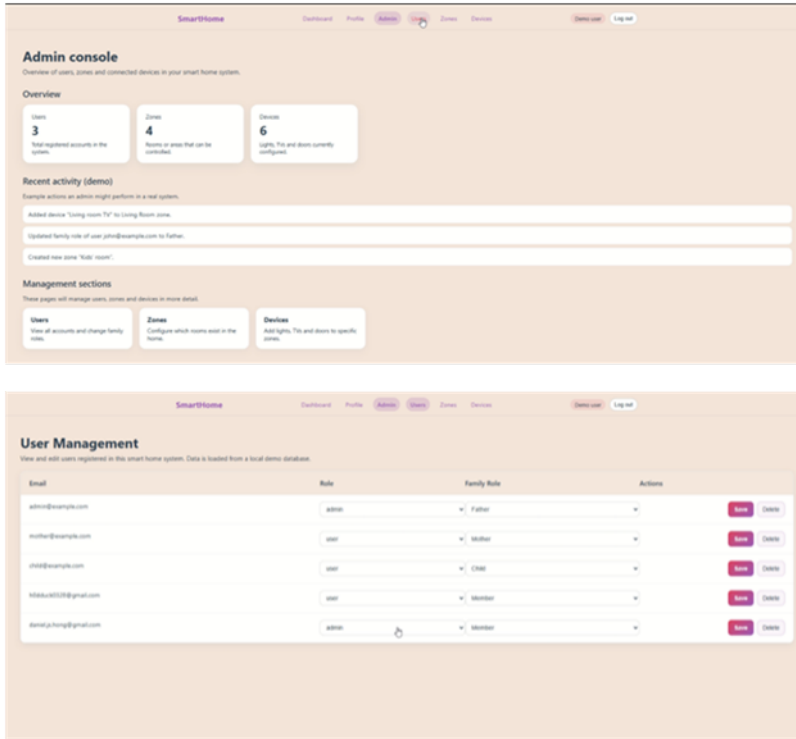| Use Case	Functionality	Initial state	Input	Output
Change personalized assistant voice.	Change the personalized assistant voice for a specific user.	Logged in: Profile page	Click on the chosen Speaker name and then click "Save Changes" button.	Changes the voice of the speaker assistant for that specific user.
Train voice	Trains the model on a specific user's voice.	Logged in: Profile page – Train voice	Click on the "Start training" button	Displays the first sentence to be recorded.
Train voice	Trains the model on a specific user's voice.	Logged in: Profile page – Train voice	Click on the "Next sentence" button	Displays the next sentence to be recorded.
Train voice	Trains the model on a specific user's voice.	Logged in: Profile page – Train voice	Click on the "Train again" button	Displays the first sentence to be recorded.
Train voice	Cancel the voice training	Logged in: Profile page – Train voice	Click on the "cancel" button	Displays the start training button and cancels voice training.
Log out	Logs out the user from the system app.	Logged in: Login page	Click on the "log out" button.	Move to login page.
Age-restricted movie search banner	Display a warning when a movie request is blocked due to the user's age	Logged in: OTT search page	User or AI speaker triggers a search for an age-restricted movie	A banner appears at the top of the page explaining that the movie is blocked based on the user's age and the movie's age rating.
Sync device states with AI speaker commands	Reflect device status changes triggered by voice commands	Logged in: Dashboard page	AI speaker sends /devices/{id} requests after recognizing a command	The device tiles on the dashboard update (e.g., lights from "off" to "on") without manual refresh.

B. Use Case 2: Web Application – Admin Exclusive Features
1) Table for Web Application User Case

Use Case	Functionality	Initial state	Input	Output
Admin	Navigate to Admin	Logged in	Click on the "Admin" button when on another page.	Move to admin page.
User	Navigate to users	Logged in	Click on the "Users" button when on another page.	Move to user's page.
Zones	Navigate to zones	Logged in	Click on the "User" button when on another page.	Move to user page.
Devices	Navigate to devices	Logged in	Click on the "Devices" button when on another page.	Move to devices page.
Change role	Change the user role for any user.	Logged in: User page	Select role, Family role and click on the "Save" button on a specific user.	Changes a user's role.
Delete role	Delete users	Logged in: User page	Click on the "Delete" button on a specific user's row.	Removes a user from the system.
Add zones	Add rooms for IoT-devices to be placed at.	Logged in: Zones page	Enter zone name, click on "Add zone" button.	Adds a new room to the system that the user can add IoT devices to.
Remove zones	Delete rooms from the system.	Logged in: Zones page	Click on the "Remove" button on a specific room.	Deletes a room from the system.
Add devices	Add IoT devices to a specific room	Logged in: Devices page	Enter device name, choose device and room from list and click on "Add device" button.	Adds a specific IoT device to a room.

Use Case	Functionality	Initial state	Input	Output
Remove devices	Remove IoT devices from a specific room	Logged in: Devices page	Click on the "Remove" button on a specific device.	Deletes a specific IoT device from the room.

2) Screenshots for Web Application User Case

B: Speaker Interaction

Use Case	Functionality	Initial System State	Input	Output
Wake word	Listen for wake word	Listening state	Speaking input	No output
Wake word	Rejecting wake word	Listening state	Speaking input	No output
Wake word	Accepting wake word	Listening state	Speaking input	No output
Wake word	Compare voice to prerecorded voices	Listening state	Speaking input	No output
Wake word	No user found	Listening state	Speaking input	Reject audio response
Wake word	User found, accepts user commands.	Listening state	Speaking input	Personalized audio greeting.

Use Case	Functionality	Initial System State	Input	Output
Voice-controlled device action	Control lights, TV, or door via voice command	System in listening state after wake word and speaker verification	Spoken command such as "Turn on the living room lights"	The AI speaker sends a REST request to the FastAPI backend, which updates the device state in MySQL; the web dashboard reflects the new state and, in a real deployment, the corresponding IoT device would be switched on.