SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR

SmartHome: okosotthon és iroda menedzselő rendszer

MŰSZAKI TUDOMÁNYOS DIÁKKÖRI KONFERENCIA DOLGOZAT

2025

Hallgatók:

Gáspár Tamás, *Sapientia EMTE, Marosvásárhelyi Kar, Számítástechnika IV.*

Gáll Benedek, *Sapientia EMTE, Marosvásárhely Kar, Számítástechnika IV.*

Témavezetők:

dr. Domokos József (docens), *Sapientia EMTE, Marosvásárhely*

dr. Novák Pálma-Rozália (adjunktus), *Sapientia EMTE, Marosvásárhely*

Derzsi Dániel (szoftverfejlesztő), *Codespring, Marosvásárhely*

Tóducz Endre (szoftverfejlesztő), *Codespring, Marosvásárhely*

Kivonat

A SmartHome egy olyan okosotthon és iroda menedzselő rendszer, ami az okoseszköz hálózatok kezelését hivatott segíteni épületek szintjén, felhasználóbarát vizuális felületet nyújtva a felhasználóknak. A rendszer két felhasználói felületből áll: egyik része egy webes adminisztrátori felület, a másik egy mobilos felhasználóknak készült alkalmazás. A rendszer szükséges eleme egy helyileg Raspberry Pi-on futtatott Home Assistant alkalmazás, amelyen keresztül kommunikálunk az okos eszközökkel. Ahhoz, hogy az eszközeinket személyes preferenciák alapján tudjuk szabályozni, a rendszer magába foglal egy szabálytervező és kiértékelő alegységet is, ami előre strukturált termosztát szabályozásra vagy szabad elképzelés alapján történő, Python alapú kódírásra alkalmas.

A webes felületen több minden valósul meg: többek között a felhasználók megrajzolhatják a szobák vagy helyiségek tervrajzait, majd ezekből egész emeleteket lehet kialakítani. Továbbá, itt található az alkalmazás szabálytervező része is. A mobilos applikáció többnyire a rendszer menedzselés részére fókuszál, mivel itt tekinthetőek meg az emeletek és szobák tervrajzai, az ezekhez tartozó okos eszközök, valamint a szabályok listája. A szabálykiértékelő rendszer alapját egy Pyodide nevezetű JavaScript csomag adja, amely képes a rendszeren belül a szabályok Python kódját futtatni, (akár felhasználó által használt új csomagokkal bővíteni, micropip-et használva) majd JavaScriptté alakítani azok kimenetét. Így a szabályok komplexitása szinte teljes mértékben csak a felhasználótól függ. Ezeket a szabályokat a rendszer épületenként összesíti és képes vezérelni az eszközöket.

A Home Assistant program használata azért fontos, mivel rengeteg kommunikációs protokollt támogat, amelyekkel a különböző gyártók különféle eszközei kommunikálnak, ezáltal lehetővé téve számunkra a szabadságot az eszközhasználat terén. Jelenleg Sonoff termékekkel volt tesztelve a rendszer.

Kulcsszavak: smarthome, Sonoff, Pyodide, Home Assistant, Raspberry Pi

Tartalomjegyzék

1. A projekt leírása és bevezető

Az emberi természetünkből adódóan szeretjük a kényelmet és ezért próbáljuk leegyszerűsíteni, felgyorsítani a mindennapi monoton teendőinket, hogy esetlegesen több időnk juthasson más fontosabb tevékenységekre, mint mondjuk a pihenésre. Ilyen teendők például a lámpák kapcsolgatása, hőmérséklet megtekintése a házban, vagy hogyha pont nem vagyunk otthon, de az egyik ablak nyitva maradt, akkor jó lenne kikapcsolni a fűtést, hogy ne működjön feleslegesen a kazán ([1]).

Ezekre a mindennapi problémákra nyújtanak segítséget az okos eszközök és vezérlők (mint például az okos égők, hő- és páratartalom-mérők, esetleg okos kapcsolók), valamint az ezeket irányító rendszerek. Az egyik nagy előnye ezeknek az okosotthon-rendszereknek a különböző eszközök automatizált vezérlése. Ezek az automatizálások lehetővé teszik, hogy az eszközök viselkedése automatikusan változzon, külső beavatkozás nélkül (például a lámpák egy adott sötétségi szint elérése után automatikusan felkapcsoljanak este). Ezeket a megoldásokat nemcsak otthonokban, de akár irodákban is lehet alkalmazni, hogy egy helyről, akár online lehessen több szobának, esetleg emeletnek a különböző eszközeit kezelni (lámpatestek, kazánokat vezérlő termosztátok).

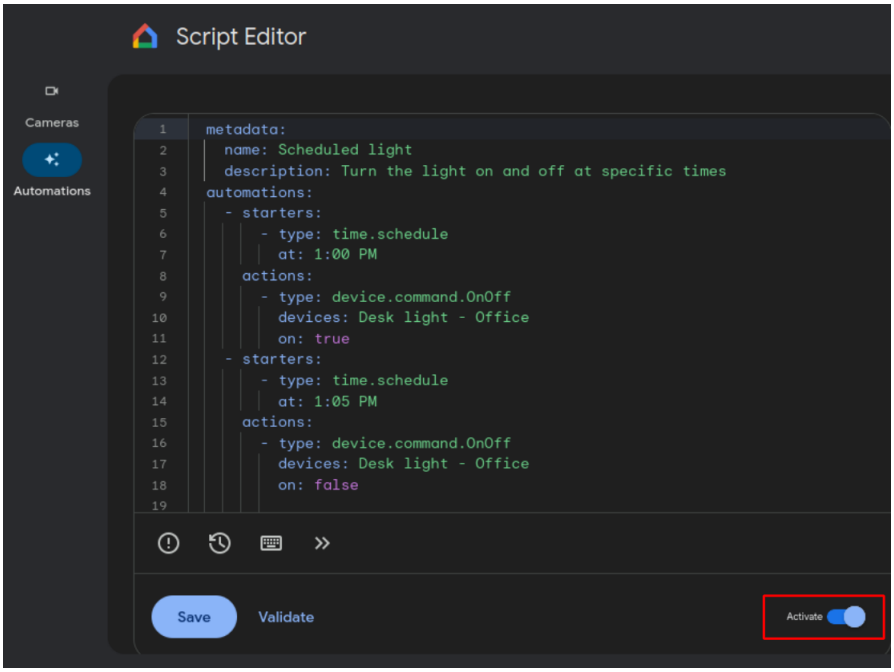
2. A projekt célja

Elsősorban a célunk az volt, hogy egy eszközfüggetlen okosotthon-működtető rendszert fejlesszünk ki, ami képes több eszközt egyszerre kezelni, különböző gyártóktól. Mivel az okoseszközök kommunikációs protokolljai eltérhetnek, ezért szükséges volt számunkra egy olyan platform, ami képes eszközfüggetlenül kommunikálni ezekkel. Emellett egy olyan felületet szerettünk volna kialakítani a felhasználóknak, aminek segítségével jól átláthatóak a nagyobb épületek is.

3. Hasonló alkalmazások

3.1. Google Home

A Google okosotthon rendszere, a Google Home egy elég kifinomult rendszer, ami rengeteg hasznos funkciót tartalmaz a felhasználók számára. Ezek közül megemlítendő a letisztult telefonos alkalmazás, amelyben elérhetőek az okoseszközök adatai, valamint egy szabályozástervező is, ami csak scriptek formájában ad lehetőséget a felhasználóknak különféle szabályok megírására. A rendszer hátránya pedig az, hogy csak a Google Home által jóváhagyott eszközöket támogatja, amelyek javarészt többe kerülnek, mint a hasonló tulajdonságú, de Google Home támogatással nem rendelkező társaik.

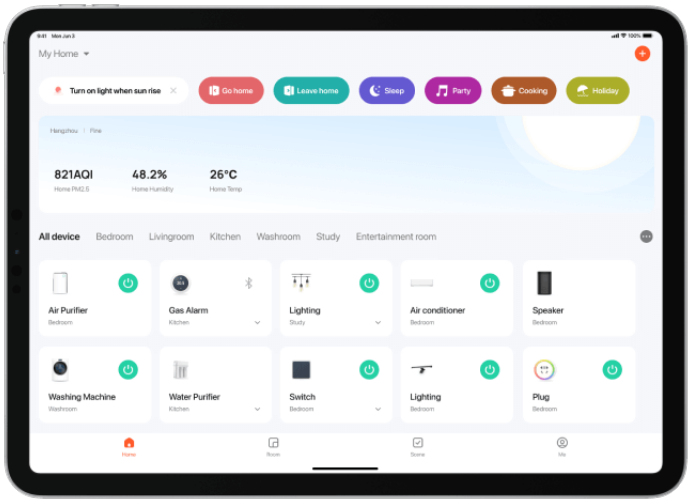


1. ábra. Google Home Script editor (kép forrása)

3.2. Tuya Smart

A Tuya Smart egy kínai fejlesztésű platform okosotthon-menedzselésre (2. ábra). Saját ökoszisztémát nyújt, lehetőséget adva a felhasználóknak, hogy akár saját maguk bővítsék a termékkínálatot, valamint újabb szolgáltatásokkal egészítsék ki a platformot. Ez a lehetőség nemcsak a magánszemélyek, de akár vállalatok számára is adott, megadva nekik a lehetőséget, hogy a platformon keresztül nemcsak a szolgáltatásaikat vagy bővítményeiket tegyék elérhetőbbé, de egyúttal saját okoseszközeiket is forgalmazhatják. Lehetőséget adnak automatizálások tervezéséhez is, telefonos applikáción belül, megadott folyamat szerint, viszont nem script

alapon. Ez egy egyszerű felhasználónak könnyítés, de a tapasztaltabb, kódírásban is jártasabb felhasználóknak hátrány is lehet.



2. ábra. Tuya Smart app (kép forrása)

4. Rendszer specifikáció

4.1. Felhasználói követelmények

A SmartHome rendszere lehetőséget kell adjon a felhasználóknak egy webes felületen való regisztrálás után bejelentkezni akár egy mobilos, akár egy webes felületre a regisztráláskor megadott e-mail cím és jelszó használatával. Bejelentkezés után már eltérnek a felületek által nyújtott lehetőségek, mivel két különálló felhasználói felülete kell legyen a rendszernek.

A **webes felületen** bejelentkezés után a felhasználónak el kell érnie az összes regisztrált irodát. Minden irodának megnyitható az emelet- vagy a szabályszerkesztője.

Az emelettervező oldalon létrehozhatóak kell legyenek az új emeletek, illetve szerkeszthetőek kell legyenek a meglévők adatai. Szobákat ugyanezen az oldalon lehet létrehozni. Kiválasztva egy emeletet, listázhatóak kell legyenek az emelethez adott szobák és tervrajzaik, amennyiben már léteznek. Amennyiben nem, akkor a szobához tartozó szerkesztő-oldalt megnyitva létrehozható kell legyen az is. Az emeletszerkesztő oldalon lehetőséget kell biztosítani szobák törlésére.

Egy szabályszerkesztő oldal kell biztosítsa a szabályok menedzselését. Ezen az oldalon láthatóak kell legyenek a már meglévő szabályok, jelezve, melyek aktívak és melyek nem aktívak.Lehetőséget kell biztosítani egyszerű (termosztát szerű) szabályok, illetve bonyolultabb nem termosztát jellegű szabályok szerkesztésére. A bonyolultabb szabályok esetében Python scripteket kell írnia a felhasználónak. A termosztát szerű szabályok egy külön oldalon kell szerkeszthetők legyenek, erre előre strukturált szerkesztési opciókkal.

A **mobilalkalmazás** elsődleges célja, hogy a felhasználók számára kényelmes és intuitív módon biztosítson hozzáférést az iroda szenzorhálózatához és annak vezérléséhez.

4.1.1. Főoldal (Térképnézet)

Bejelentkezés után a felhasználó egy központi dobozt kell lásson a főoldalon, amelyre rákattintva megtekintheti az iroda alaprajzát. Az alaprajz mindig ahhoz az irodához tartozik, amelyhez a felhasználó hozzá van rendelve. A felületen lehetőséget kell biztosítani az emeletek közötti váltásra, valamint az egyes szobák interaktív megtekintésére. Amennyiben a felhasználó egy adott szobára kattint, meg kell jelenjen a szobában elhelyezett szenzorok listája.

4.1.2. Navigáció

Az alkalmazás alsó részén elhelyezkedő navigációs sáv segítségével a felhasználónak lehetősége kell legyen egyszerűen váltani az egyes oldalak között. A navigáció három fő oldalt fog tartalmazni:

1. Főoldal (Térképnézet)
2. Eszközlista
3. Szabálykezelés

4.1.3. Eszközlista

Az Eszközlista oldalon az iroda összes regisztrált eszköze megjelenítésre kell kerüljön. A felhasználó számára többféle szűrési lehetőséget kell biztosítani:

- Eszköz- vagy szenzortípus alapján
- Tag-ek (címkék) alapján
- Szöveges keresés eszköznév szerint

Továbbá a rendszer lehetőséget kell nyújtson új tag-ek létrehozására. Amennyiben a felhasználó hosszabban lenyom egy eszközt, az ki kell jelölődjön. Ezután több eszköz is kijelölhető egyszerű érintésekkel. Amint legalább egy kijelölés történik, a jobb alsó sarokban meg kell jelenjen egy „Add Tag” gomb. A gombra kattintva egy felugró ablak jelenik meg, ahol megadható a címke neve, amely ezt követően elmentésre kerül.

4.1.4. Szabálykezelés

A Szabályok oldalon az összes létrehozott szabály kilistázásra kerül. Az oldal kell rendelkezzen egy keresőfunkcióval, amely a szabályok nevében történő szöveges keresést kell lehetővé tegye. A szabályok állapottól függően vizuálisan megkülönböztethetők: aktív szabály esetén zöld, inaktív szabály esetén piros háttér kell megjelenjen. Egy szabályra kattintva egy felugró ablakban kell láthatóvá váljon. Az ablak kell tartalmazza a szabály nevét, leírását, aktuális állapotát, valamint egy kapcsolót, amellyel a szabály aktiválható vagy inaktiválható.

4.2. Rendszerkövetelmények

4.2.1. Funkcionális követelmények

Jelenleg csupán egyféle felhasználója létezik a rendszernek. A két felület viszont különböző funkciókat lát el a rendszeren belül. Míg a webes felület többnyire az

adatok bevitelére szolgál, vagy azok szerkesztésére, addig a mobil felület azok megjelenítésére és vezérlésére fókuszál. A **web** felület funkciói:

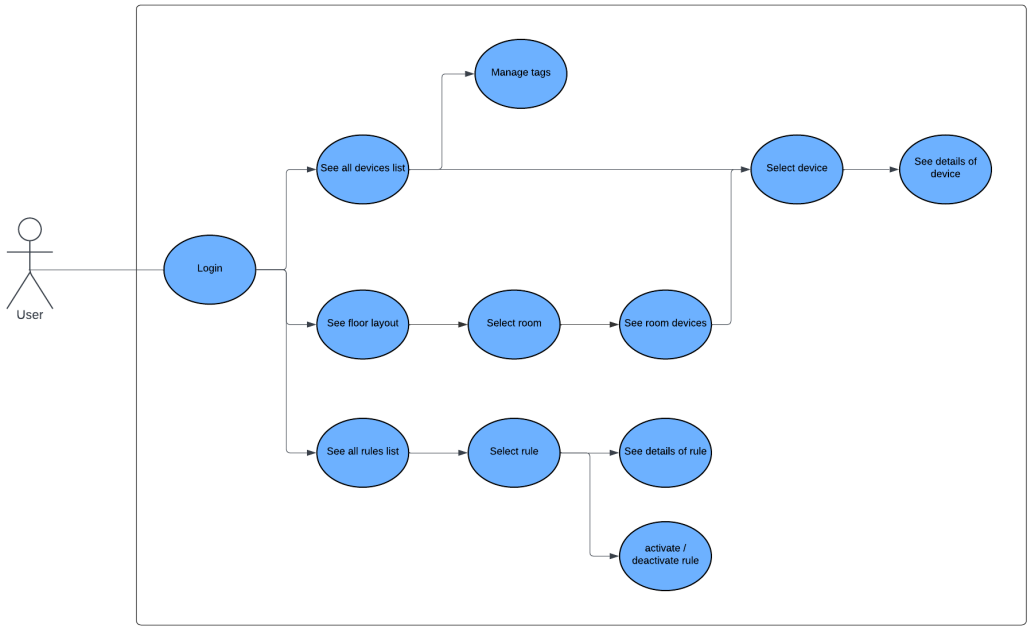
- ki- és bejelentkezés;
- irodák megtekintése;
- emeletszerkesztők megnyitása;
- új emeletek létrehozása;
- emeletek tervrajzának szerkesztése és megtekintése;
- szobák létrehozása;
- szobák tervrajzának és egyéb adatainak szerkesztése;
- szabályok megtekintése, szerkesztése.

A **mobil** felület funkciói:

- bejelentkezés;
- térkép nézet megtekintése;
- interakció a térkép nézettel;
- szenzorok adatainak megtekintése;
- szenzor-csoportok kezelése (létrehozás, törlés);
- szenzorok szenzorcsoporthoz rendelése és eltávolítása;
- szabályok megtekintése;
- szabály aktiválás és deaktiválás.

4.2.2. Nem funkcionális követelmények

A rendszer megfelelő működéséhez szükséges egy olyan eszköz, amely megfelel a Home Assistant OS minimális hardverkövetelményeinek. A terveink szerint ez egy Raspberry Pi modell lesz.



3. ábra. Mobil alkalmazás használati eset diagramja (saját ábra)

5. Felhasznált technológiák

5.1. IoT rendszerek

Az **IoT (Internet of Things)** olyan hálózatok összessége, amelyekben az eszközök adatokat osztanak meg egymással és együttműködnek a folyamatok optimalizálása érdekében. Az IoT rendszerek legfőbb kihívásai a következők:

- **Biztonság és adatvédelem:** Az IoT eszközök hálózata sérülékeny lehet, hiszen nagy mennyiségű adat kerül tárolásra és továbbításra online környezetben.
- **Kompatibilitás:** A különböző gyártók által fejlesztett eszközök szabványosításának hiánya akadályozza az együttműködést.
- **Skálázhatóság:** Az IoT-hálózatok bővítése és kezelése egyre komplexebb problémát jelent a növekvő eszközszám miatt.

E kihívások mellett az IoT rendszerek számos lehetőséget kínálnak, például az automatizálási folyamatok létrehozására. Az **otthonautomatizálás** célja, hogy a különböző eszközök és rendszerek egyetlen platformon keresztül vezérelhetők legyenek. Jelenlegi trendek:

- **Hangvezérlés és mesterséges intelligencia:** Az olyan rendszerek, mint a Google Assistant és az Amazon Alexa, népszerűvé tették az automatizálási megoldásokat, mivel képesek adott mondatokra valamilyen tevékenységet végrehajtani az általuk is elérhető okoseszközökkel.
- **Energiahatékonyság:** Az okos termosztátok és világítási rendszerek csökkentik az energiafogyasztást, miközben növelik a kényelmet.
- **Távoli vezérlés:** A különböző okosotthon vezérlő platformok általában lehetőséget adnak távoli vezérlésre cloud technológiák segítségével, ezzel nagyban megkönnyítve a felhasználók mindennapjait.

5.2. React

A **React** egy nyílt forráskódú JavaScript könyvtár, amelyet a Meta (korábban Facebook) fejlesztett ki, és amelyet elsősorban felhasználói felületek (UI) építésére használnak. A React egyik legfontosabb jellemzője a komponens-alapú architektúra, amely lehetővé teszi, hogy az alkalmazás különálló, újrahasznosítható egységekből (komponensekből) épüljön fel. Ez a megközelítés jelentősen megkönnyíti a kód karbantartását, tesztelését és skálázhatóságát.

A React emellett a *virtual DOM* használatával optimalizálja a böngészőben történő megjelenítést, így a felhasználói felület gyorsabban és hatékonyabban frissül.

A felhasználók hitelesítése során a rendszer a bejelentkezéskor JWT (JSON Web Token) tokent generál, amelyet a kliens a további kommunikáció során elküld minden kéréshez. A szerver ez alapján tudja hitelesíteni a felhasználót, és hozzáférést biztosítani a megfelelő erőforrásokhoz.

5.3. React Native

A **React Native** szintén a Meta által fejlesztett, JavaScript-alapú keretrendszer, amely lehetővé teszi natív mobilalkalmazások készítését Android és iOS platformokra egyaránt. A technológia egyik legnagyobb előnye, hogy a React-ben megszokott komponensalapú szemléletet átülteti mobilalkalmazásokba is, ezáltal lehetővé téve a kód újrahasznosítását különböző platformok között.

A React Native nem webes felületet jelenít meg, hanem natív UI komponenseket használ, így a felhasználók egy valódi natív alkalmazás élményét kapják, miközben a fejlesztőnek nem szükséges külön kódot írnia minden platformra. A jelen dolgozatban bemutatott mobilalkalmazás – amely a térképes irodai navigációt, eszközkezelést és szabálykonfigurációt valósítja meg – React Native technológiával készült.

5.4. Backend szolgáltatások Honoval

A **Hono** egy kisméretű, egyszerű és villámgyors webes keretrendszer, amelyet a Web Standards alapjaira építettek. Számos JavaScript futtatókörnyezetben használható, például Cloudflare Workers, Fastly Compute, Deno, Bun, Vercel, Netlify, AWS Lambda és Node.js alatt. Főbb jellemzői:

- **Ultragyors:** A beépített RegExpRouter segítségével gyors útválasztást biztosít, elkerülve a lineáris ciklusokat.
- **Könnyűsúlyú:** A `hono/tiny` előre beállított verziója kevesebb, mint 14 kB méretű.
- **Több futtatókörnyezet:** Ugyanaz a kód különböző környezetekben, például Cloudflare Workers vagy AWS Lambda alatt is működik.
- **Bővíthető:** Beépített middleware-ek, valamint testreszabható és harmadik féltől származó middleware-ek állnak rendelkezésre.

A Hono-t széleskörűen használják különböző alkalmazási területeken, mint például:

- Webes API-k létrehozása
- Backend szerverek proxyzása
- Full-stack alkalmazások alapja

A projektünk során a Hono előnyeit kihasználva építettük fel a backend rendszert, amely gyors és skálázható megoldást nyújt az IoT eszközök vezérléséhez és az adatok feldolgozásához.

5.5. MongoDB

A **MongoDB** egy nyílt forráskódú, dokumentum-orientált NoSQL adatbáziskezelő rendszer, amely rugalmas séma-struktúrát biztosít adatok tárolására és lekérdezésére.
A MongoDB kulcsfontosságú jellemzői:

- **Dokumentum-orientált tárolás:** Az adatokat JSON-szerű BSON (Binary JSON) formátumú dokumentumokban tárolja, amely lehetővé teszi a komplex adatstruktúrák egyszerű kezelését.
- **Horizontális skálázhatóság:** A sharding technika segítségével több szerverre is eloszthtató az adatbázis, így támogatja a nagy adatmennyiséget kezelő rendszereket.
- **Flexibilis adatséma:** Nincs szükség előre definiált sémára, az adatmodell könnyen módosítható a projekt fejlődése során.
- **Magas rendelkezésre állás:** A replikációs mechanizmus biztosítja az adatok redundanciáját és a folyamatos működést.

5.6. Home Assistant

Mivel célunk az volt, hogy minél szélesebb körben képes legyen az alkalmazásunk vezérelni a különböző kommunikációs protokollokkal szerelt eszközöket, ezért szükségünk volt egy olyan platformra, ami képes erre. Mivel szükségleteinket teljes mértékben képes volt lefedni, ezért a Home Assistant alkalmazását választottuk.

[2] A **Home Assistant** egy nyílt forráskódú, Python-alapú platform, amely különböző IoT (Internet of Things) eszközök és szolgáltatások integrációjára és automatizálására nyújt lehetőséget bárki számára. Népszerűsége több tényezőnek köszönhető:

- **Eszközfüggetlen működés:** Támogatja több, mint 1000 különböző eszköz és szolgáltató integrációját.
- **Nyílt forráskód, aktív fejlesztői közösség:** Az okosotthon felhasználók adatainak védelme szempontjából nagyon előnyös.
- **Modularitás:** A közösség által fejlesztett modulokkal bővíthető a rendszer, ezáltal még személyre szabottabb élmény vár a felhasználókra.
- **Külső kommunikáció:** A Home Assistant applikációja biztosít a felhasználók számára egy RESTful API réteget, aminek segítségével lekérhetőek az eszközök adatai, valamint vezérelhetőek is.

A dolgozat közvetlenül kapcsolódik ezekhez az IoT alapelvekhez, mivel a Home Assistant szolgáltatásait használja fel az adatok feldolgozására és vezérlési kérések küldésére. Ez lehetővé teszi, hogy a rendszer hatékonyabbá, automatizáltabbá és személyre szabhatóbbá váljon.

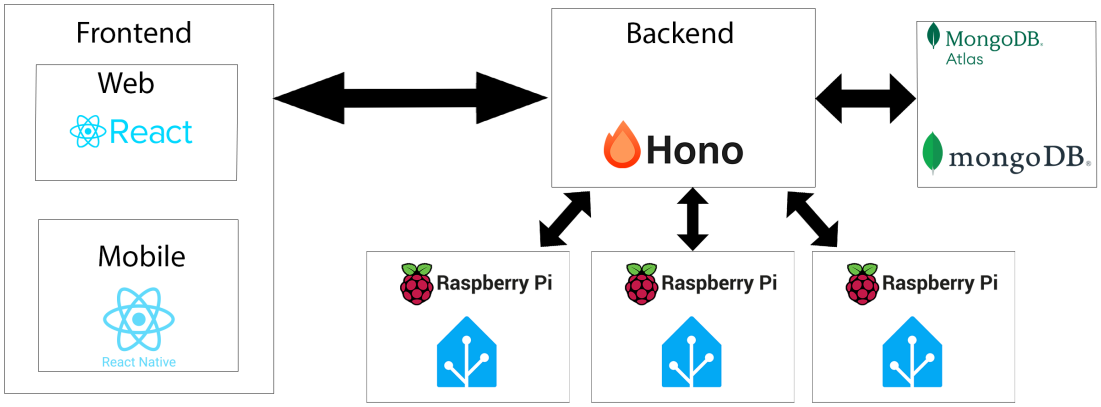
4. ábra. Home Assistant logo (kép forrása)

6. A rendszer bemutatása

6.1. A rendszer felépítése

A SmartHome rendszerét 4 különálló komponens alkotja: a Raspberry Pi-on futó Home Assistant alkalmazás (minden lokációban szükség van egy Raspberry Pi-ra), a rendszer backend szervere, az adatbázis (MongoDB), valamint a webes felület és a mobil applikáció, mint különálló frontendek.

Az alkalmazáshoz használt MongoDB adatbázis egy Mongo Cluster felhő tárolóban fut, ami elősegíti az adatok elérhetőségét interneten keresztül bármilyen eszközön, így biztosítva az állandó adatforgalmat a felhasználóknak, és biztosítva a fejlesztőknek is a hozzáférést, ezáltal az adatbázison esett változások mindenkinél azonosan jelen vannak a fejlesztés során.



5. ábra. Architektúra diagram (saját ábra)

6.1.1. Home Assistant

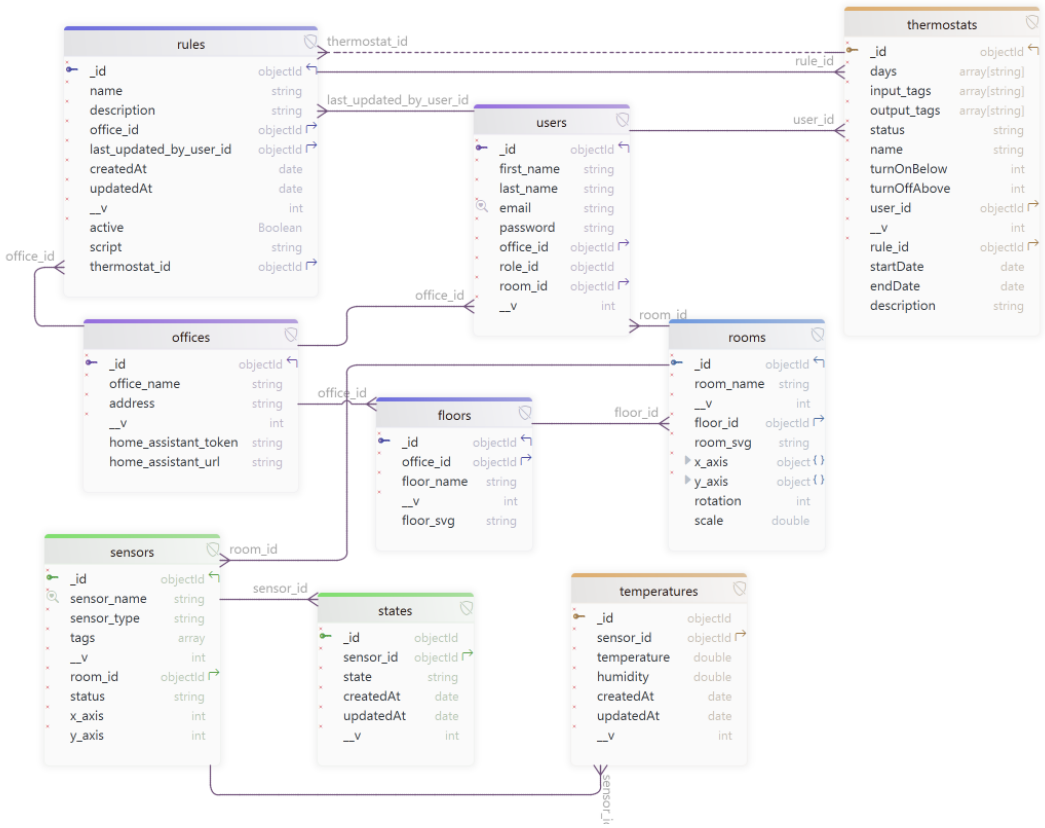
Esetünkben egy 4 GB memóriával rendelkező Raspberry Pi 4B szolgált a rendszer futtatására. Emellett szükség van egy SD kártyára (ajánlott: legalább 32 GB kapacitású), valamint egy számítógépre vagy laptopra, amely képes SD kártya olvasására és írására.

Az operációs rendszer telepítéséhez a *Raspberry Pi Imager* programot használtuk, amellyel a Home Assistant OS került az SD kártyára. A kiválasztott operációs rendszer az *Other specific-purpose OS* kategória *Home assistants and home automation* szekciójából lett kiválasztva, azon belül is a *Home Assistant* lehetőség.

6.1.2. Adatbázis

A backend alapjául egy MongoDB adatbázis szolgált, amelyet a MongoDB Atlas felhőszolgáltatás segítségével hosztoltunk. Az Atlas lehetőséget biztosított egy

dedikált MongoDB cluster létrehozására, amelyet a felhőben futtattunk, így biztosítva a valós idejű szinkronizációt, az adatok egységességét és a megbízható elérhetőséget.



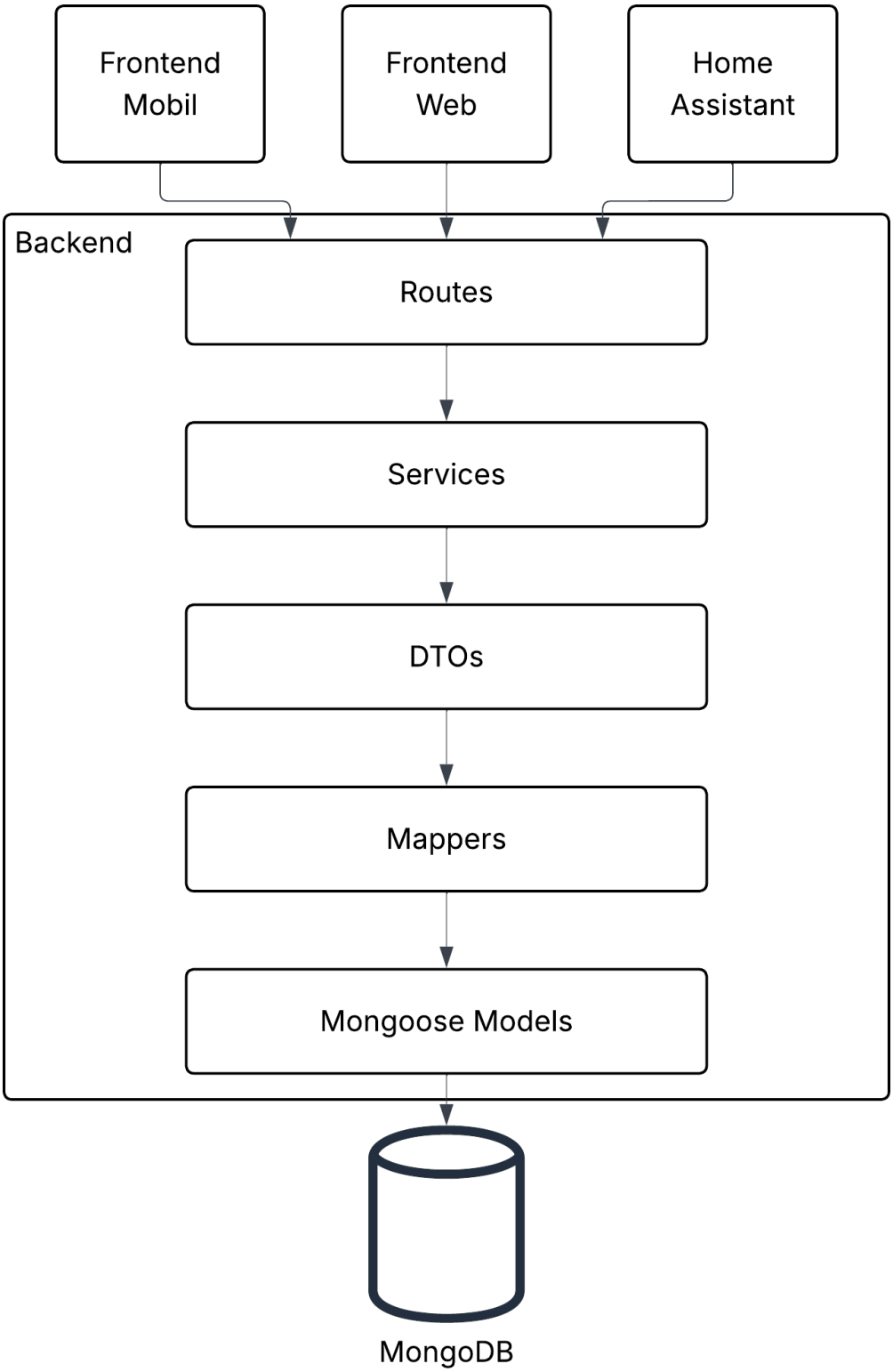
6. ábra. Egyed-kapcsolat diagram

A MongoDB Atlas egy felhőalapú, adatbázis-szolgáltatás, amelyet a MongoDB Inc. fejlesztett és üzemeltet. Lehetővé teszi MongoDB adatbázisok egyszerű létrehozását, kezelését és skálázását különböző felhőszolgáltatók – például az Amazon Web Services (AWS), a Google Cloud Platform (GCP) vagy a Microsoft Azure – infrastruktúráján.

Az Atlas segítségével a fejlesztők könnyedén hozhatnak létre úgynevezett *clustereket*, amelyek magas rendelkezésre állást, automatikus biztonsági mentéseket, skálázhatóságot és beépített monitoring lehetőségeket kínálnak. A MongoDB Atlas megszabadít a hagyományos adatbázis-adminisztrációs feladatok (például telepítés, frissítés, biztonsági mentés, monitorozás) jelentős részétől, így a fejlesztők a tényleges alkalmazáslogikára és az üzleti értékre koncentrálhatnak.

6.1.3. Backend

A rendszer backendjének fejlesztéséhez a Hono nevű, lightweight és rendkívül gyors webes keretrendszert alkalmaztuk. A Hono a modern webes szabványokra épül, és kompatibilis több JavaScript futtatókörnyezettel, mint például a Cloudflare Workers, Deno, Bun vagy Node.js. Projektünkben a Bun JavaScript futtató-



7. ábra. Backend rétegei (saját ábra)

környezetet választottuk, amely integrálja a futtatómotort, a csomagkezelőt és a bundlert, így jelentős mértékben leegyszerűsíti a fejlesztési és futtatási folyamatot.

A teljes backend fejlesztés *TypeScript* nyelven történt, amely lehetőséget ad a statikus típusellenőrzésre. Ez különösen hasznos a nagyobb kódbázisokban, mivel javítja a hibadetektálást már fordítási időben, ezáltal növelve a kód megbízhatóságát és karbantarthatóságát.

A rendszer architektúrája többrétegű és moduláris felépítésű (lásd 7. ábra). Az egyes rétegek jól elkülönített felelősségi körrel rendelkeznek:

- **Kontrollerek:** HTTP végpontokat biztosítanak, és felelősek az adott entitások CRUD műveleteinek (Create, Read, Update, Delete) kezeléséért.
- **Szerviz réteg:** Az üzleti logika itt kerül implementálásra. A kontrollerek delegálják a feldolgozást ide, ahol validáció, adatfeldolgozás és adatbáziskezelés történik.
- **DTO-k (Data Transfer Object):** Segítségükkel validált, típusos adatcsere történik a frontend és a backend között. Külön DTO-k szolgálnak a bemenő és kimenő adatokra.
- **Mapperek:** Átalakítják a DTO-kat és entitásokat egymás között. Ez az absztrakció elősegíti a rendszer rugalmasságát és könnyű módosíthatóságát.
- **Mongoose modellek:** Ezek biztosítják az adatbázissal való kapcsolatot. A MongoDB sémák határozzák meg, hogyan épül fel egy entitás tárolása az adatbázisban.
- **Entitások:** Az alkalmazás alapvető adatmodelljeit reprezentálják (pl. irodák, szabályok, szenzorok). Ezek meghatározzák a rendszer logikai struktúráját.

6.1.4. Frontend - Mobil alkalmazás

A mobilalkalmazás fejlesztéséhez a React Native keretrendszert választottuk, mivel lehetővé teszi natív élményű mobilalkalmazások fejlesztését JavaScript és TypeScript nyelven, egyetlen kódbázisból, Android és iOS platformokra egyaránt. A projekt az Expo keretrendszerre épül, amely egyszerűsíti a fejlesztést, telepítést és hibakeresést – különösen a cross-platform fejlesztések során.

A felület kialakításához React komponenseket használtunk. Minden képernyő és UI-elem önálló komponensként működik, így biztosítva a kód modularitását és újrahasznosíthatóságát. A komponensek stílusozása inline StyleSheet objektumokon keresztül történt, valamint használtunk külső ikonokat az expo/vector-icons csomagból, amely az Ant Design és más népszerű ikoncsaládokat tartalmazza.

6.1.5. Frontend – Webalkalmazás

A rendszer webes felületének fejlesztéséhez a React könyvtárat használtuk, amely lehetővé teszi a felhasználói felület hatékony és moduláris fejlesztését. A Reactet **TypeScript**-tel kombináltuk, amely statikus típusellenőrzést biztosít, ezáltal növelve a fejlesztés megbízhatóságát és a kód olvashatóságát.

Az oldal navigációját a `react-router-dom` könyvtár segítségével valósítottuk meg. Ez lehetővé teszi, hogy minden nézet (oldal) egyedi URL-elérhetőséget kapjon, és a felhasználói útvonalak jól strukturáltan, egyértelműen legyenek kezelhetők.

A frontend rétegei logikusan tagoltak, a következő struktúrában épülnek fel:

- **Service réteg**:

  Ez a réteg tartalmazza azokat a függvényeket, amelyek HTTP-kéréseket küldenek a backend API felé. Ezek felelősek az adatok lekéréséért, küldéséért, frissítéséért vagy törléséért.

- **Oldalak (Pages)**:

  Minden oldal egy-egy különálló nézetet képvisel az alkalmazásban. Az oldalak közvetlenül a service réteg függvényeit hívják meg az adatok betöltéséhez és kezeléséhez.

- **Komponensek**:

  Az oldalak újrafelhasználható UI elemekből, azaz komponensekből épülnek fel. Ezek lehetnek formok, gombok, kártyák, listák stb., amelyeket egységes stílusban jelenítünk meg, az **Ant Design** (antd) UI könyvtár segítségével.

Ez a rétegzett struktúra lehetővé teszi a kód újrafelhasználhatóságát, karbantarthatóságát és átláthatóságát. A frontend célja egy responsive, könnyen kezelhető és esztétikus felhasználói felület biztosítása, amely gördülékenyen kommunikál a háttérrendszerrel.

7. Megvalósítás

7.1. Home Assistant

A telepítés folyamata az alábbi lépésekből állt:

1. A Raspberry Pi Imager program letöltése és telepítése a hivatalos weboldalról.
2. Az SD kártya behelyezése a számítógép kártyaolvasójába.
3. Az Imager alkalmazás elindítása után a *Choose OS* menüpont kiválasztása.
4. A megjelenő listában az *Other specific-purpose OS* kategórián belül a *Home assistants and home automation* szekció kiválasztása, majd a *Home Assistant* opció kijelölése.
5. Az *SD kártya* kiválasztása a *Choose Storage* menüpontban, majd a *Write* gombra kattintva az operációs rendszer telepítése az adathordozóra.

A telepítés után az SD kártyát behelyeztük a Raspberry Pi-be, majd az eszközt csatlakoztattuk a hálózathoz és a tápegységhez. A rendszer automatikusan elindította a Home Assistant környezetet, amely néhány perc múlva elérhetővé vált a `http://homeassistant.local:8123/` címen keresztül. Itt beregisztráltunk, majd bejelentkeztünk és elkezdtük a szenzorok és eszközök csatlakoztatását.

Szenzorok és eszközök csatlakoztatása

A Home Assistant rendszer lehetővé teszi különféle szenzorok és eszközök integrálását. A projekt során az alábbi típusú eszközöket csatlakoztattuk:

- **Sonoff SNZB-02** – Zigbee hőmérséklet- és páratartalom szenzorok, amelyek alapvető bemeneti adatokat szolgáltattak a termosztát szabályokhoz.
- **Sonoff SNZB-03 és SNZB-03P** – Zigbee mozgásérzékelők, amelyekkel jelenlétet lehetett detektálni egyes helyiségekben.
- **Sonoff SNZB-04** – Zigbee ajtó- és ablaknyitás-érzékelők, amelyek a nyitott/zárt állapot figyelését teszik lehetővé.
- **Sonoff S26R2 WiFi Smart Plug** – Intelligens konnektor, amely WiFi-n keresztül vezérelhető. Ehhez a Home Assistant *eWeLink* integrációjára volt szükség a megfelelő kommunikációhoz és eszközfelismeréshez.

- **Zigbee 3.0 USB Dongle Plus** – A Zigbee protokollon kommunikáló eszközök számára biztosított kapcsolatot a Raspberry Pi és a Home Assistant között.

A Zigbee alapú eszközök integrálása a *ZHA (Zigbee Home Automation)* modul segítségével történt. A dongle segítségével egy helyi Zigbee hálózat jött létre, amelyhez az eszközök automatikusan csatlakoztak párosítás után.

A WiFi-alapú Sonoff S26R2 okoskonnektor működéséhez külön konfigurációra volt szükség. A Home Assistant-ban az *eWeLink Smart Home* integráció lett telepítve, amely lehetővé tette az eszköz vezérlését és állapotának lekérdezését a rendszerből.

Az eszközök azonosítása és elnevezése után a Home Assistant automatikusan elérhetővé tette azokat a backend számára, így azok a szabályok kiértékelése során felhasználhatóak lettek. A backend a Home Assistant által biztosított végpontok által lettek elérhetőek, amelyhez egy Bearer token volt szükséges, de ezt biztosította a Home Assistant. Az eszközökhöz társított nevek és típusok alapján lehetőség nyílt scriptek és automatizálási szabályok létrehozására mind grafikus felületen, mind a script editor segítségével.

7.2. Fejlesztői környezet

A fejlesztés során a backend és a frontend fejlesztéséhez egyaránt Visual Studio Code-ot használtunk. Az adatbázist MongoDB Compassban kezeltük, valamint a kész endpointokat Postmanben teszteltük. Azért választottuk ezeket a környezeteket, mert segítőkész, könnyen kezelhető a felületük, és sok aprósággal segítik elő a fejlesztést komplexebb, több komponensből összeálló applikációk fejlesztése során. Emellett mindegyik általunk használt fejlesztői környezet támogat valamilyen formájú mesterséges intelligencia alapú segédeszközt, amit használtunk a fejlesztés során (esetünkben a GitHub Copilotot), mivel sokszor felgyorsította a hasonló szerkezetű kódrészletek megírását, valamint a fellépő hibák megoldásában is használható segítség volt.

7.3. Mobil frontend - React Native

7.3.1. Projektstruktúra

Az alkalmazás frontend projekt rétegekre van bontva:

- **(auth)/** – Ez a mappa tartalmazza a hitelesítési (autentikációs) folyamatokhoz kapcsolódó oldalkomponenseket, például a bejelentkezés képernyőt. Itt történik a felhasználó azonosítása, a JWT token kezelése.

- **(tabs)**/ – Ebben a mappában találhatóak az alkalmazás oldalai, amelyeket a tab navigáció segítségével lehet elérni. Ilyenek például a kezdőoldal (Home), az eszközlista (Devices) vagy a szabálykezelő (Rules) oldal. A tab navigáció lehetővé teszi a gyors és intuitív váltást a fő nézetek között.
- **services**/ – Backend API-hívásokat kezelő modulok, amelyek REST végpontokon keresztül kommunikálnak.
- **dtos**/ – Az adatátviteli objektumokat (*Data Transfer Object*) tartalmazó típusdefiníciók.
- **components**/– Újrahasznosítható felhasználói felület elemek.
- **assets**/- Az alkalmazásban használt statikus fájlokat, például képeket, ikonokat, SVG-ket, tartalmazza.

7.3.2. Navigáció és routing

Az alkalmazás navigációs logikáját az `expo-router` csomag segítségével valósítottuk meg, amely a React Navigation könyvtárra épül, de egyszerűbb, fájlrendszer-alapú útvonalkezelést biztosít. Ez lehetővé teszi, hogy a fájl- és mappaszerkezet közvetlenül tükrözze az alkalmazás oldalait és útvonalait, így leegyszerűsítve a navigáció implementálását.

A projekt struktúrában ennek megfelelően speciális mappanév-konvenciókat alkalmaztunk:

A **(tabs)**/ mappa például a fő navigációs felületet biztosító alsó menüsáv oldalait tartalmazza. Az expo-router felismeri, hogy ezek a fájlok tabként funkcionálnak, és automatikusan létrehozza a szükséges tab navigációs struktúrát.

Ez a megközelítés lehetővé teszi, hogy minimális konfigurációval alakítsuk ki az alkalmazás navigációját, miközben a könyvtárstruktúra intuitív módon vezeti a fejlesztőt a különböző képernyők és nézetek közötti kapcsolatban. Az expo-router támogatja a dinamikus útvonalakat, layout komponenseket, és middleware-funkcionalitást is, amelyet igény szerint ki tudtunk használni.

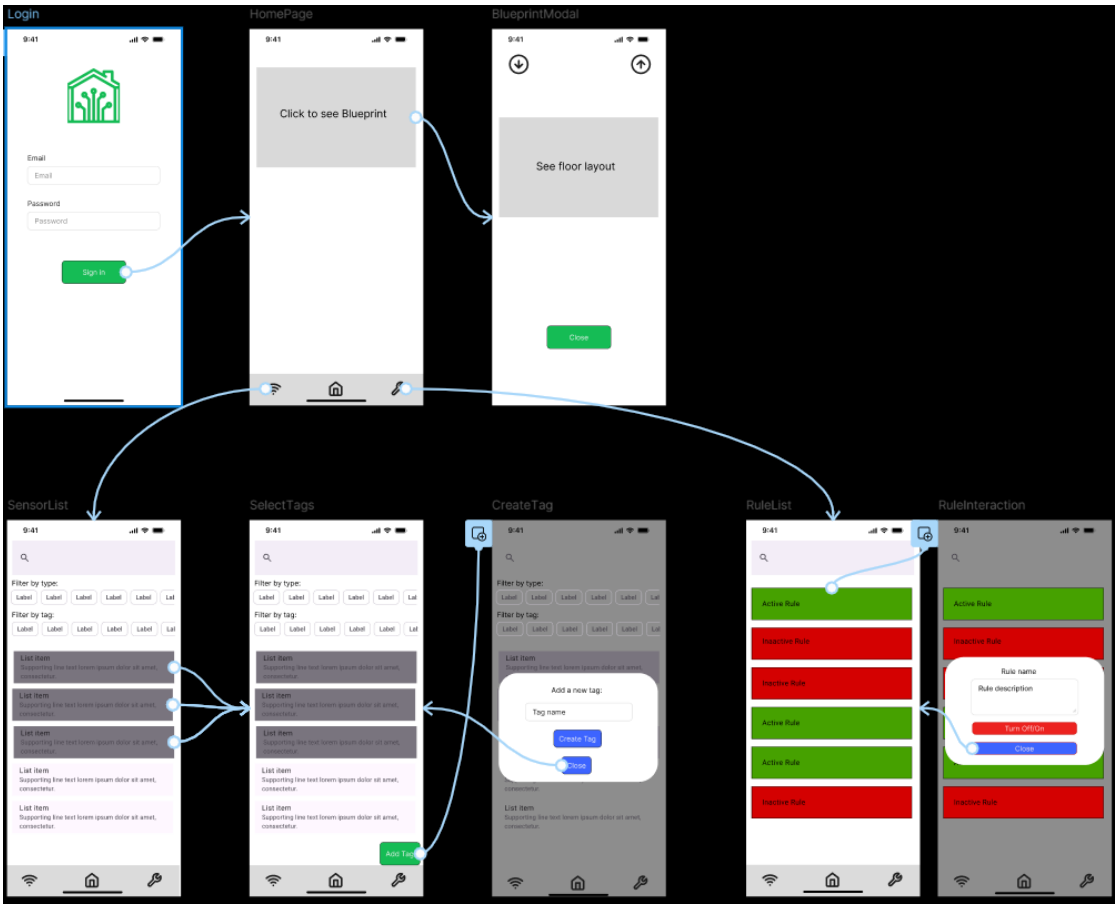
7.3.3. Hitelesítés

A bejelentkezést követően a backend által visszaküldött JWT tokent az `expo-secure-store` segítségével tároljuk el a felhasználó eszközén. Ez a csomag biztonságos, titkosított módon képes adatokat menteni, így különösen alkalmas érzékeny információk, például tokenek kezelésére mobil környezetben.

A token az alkalmazás teljes működése során elérhető marad, és minden olyan védett (autentikációt igénylő) API-hívás során automatikusan csatolásra kerül a `services/` mappában található, REST-végpontokat kezelő modulokon keresztül.

Ez a megközelítés lehetővé teszi, hogy a felhasználónak ne kelljen minden alkalmazásindításkor újra bejelentkeznie, amennyiben a token még érvényes. Amikor a token lejár, a rendszer automatikusan kijelentkezteti a felhasználót, vagy hiba esetén lehetőséget biztosít új bejelentkezésre.

7.3.4. Felhasználói felület



8. ábra. Mobil alkalmazás wireframe (saját ábra)

A felhasználó az alkalmazás megnyitásakor a bejelentkező oldallal találja szemben magát (lásd 8. ábra). Innen bejelentkezés után a "Home Page" fogadja. Ha az elmúlt 24 órában már be volt jelentkezve, akkor bejelentkezés nélkül az alkalmazás megnyitásakor is erre az oldalra kerül. Itt az oldalon található téglalapra nyomva megtekintheti az emelet tervrajzát, az ott levő szobákat. Ha egy szobára nyom, akkor egy felugró ablakban láthatja a szobában levő eszközöket. Emellett a képernyő tetején található nyilakkal navigálhat az emeletek tervrajzaik között.

Az alsó navigációs sáv segítségével lépkedhet a további oldalak között. Ha a hálózat ikonra nyom, akkor az eszközök listája oldalt tekintheti meg. Itt a felhasználó számára elérhető egy keresési funkció, illetve kétféle szűrési lehetőség, amelyek kombinálhatóak. Lehetséges eszköztípus szerint és csoportosított címkék szerint szűrni. Egy eszközre nyomva megjelenik az eszköz adatlapja, ahol megtekintheti az általa mért adatokat. Ha új címkét szeretne létrehozni, akkor hosszasan nyomva tartva egy eszközt az kijelölődik, ezután kiválaszthatja, mely eszközöket szeretné még csoportosítani. Ezt követően megjelenik a jobb alsó sarokban az "Add Tag" gomb, amelyre rányomva egy felugró ablakot láthat. Itt szükséges megadni a címke nevét, majd a "Create Tag" gombra kattintva létrehozhatja a címkét. Ha meggondolta volna magát, akkor a "Close" gombbal bezárhatja az ablakot.

A kulcs ikonra nyomva a felhasználó számára megjelenik egy lista a szabályokkal. Az aktív szabályok zöld háttérszínnel, míg az inaktívak piros háttérrel. Ezen az oldalon is rendelkezésre áll egy keresési lehetőség. Egy szabályra nyomva egy felugró ablakban megjelenik a szabály neve, leírása és aktuális állapota. Ha szeretné változtatni, akkor az állapottól függően a "Turn On/Off" gombra nyomva aktiválhatja vagy deaktiválhatja a szabályt. A "Close" gombbal léphet vissza a szabálylista oldalra.

7.4. Webes frontend

A webes felhasználói felület megvalósításához a React könyvtárat használtuk, amely lehetővé teszi a komponens-alapú, dinamikus fejlesztést. A fejlesztés során a React-et *TypeScript*-tel kombináltuk, hogy típusbiztos, skálázható és könnyebben karbantartható kódot készíthessünk.

Projektstruktúra

A frontend projekt rétegekre van bontva:

- **pages/** – Az alkalmazás oldalai (pl. kezdőlap, iroda kezelőfelület, szabálykezelő) külön fájlokban kaptak helyet, melyekhez a routing alapján egyértelmű útvonal tartozik.
- **components/** – Újrahasznosítható felhasználói felület elemek, mint például űrlapok, kártyák, táblázatok.
- **services/** – Backend API-hívásokat kezelő modulok, amelyek REST végpontokon keresztül kommunikálnak.

- **dtos**/ – Az adatátviteli objektumokat (*Data Transfer Object*) tartalmazó típusdefiníciók.
- **utils**/ és **constants**/ – Segédfüggvények és állandó értékek tárolására szolgáló könyvtárak.

Navigáció és routing

Az útvonalkezelést a `react-router-dom` csomag segítségével valósítottuk meg. Az alkalmazás többboldalas SPA (Single Page Application) struktúrát követ, az útvonalak pedig egyértelműen irányítanak az oldalakhoz. Példa egy route definícióra:

```
<Route
    path="/manage_rules/:officeId"
    element={<ManageRulesPage />}
/>
```

Backend kommunikáció

A `services/` könyvtárban található modulok aszinkron függvényeket tartalmaznak, amelyek a backenddel való kapcsolatot biztosítják `fetch` hívásokon keresztül. Például egy iroda lekérdezése az alábbi módon történik:

```
export const getOffices = async (): Promise<OfficeDto[]> => {
  const response = await fetch(`${BACKEND_API_URL}/offices`);
  return response.json();
};
```

A hibakezelést az `antd` könyvtár `message` komponensével végeztük, amely felhasználóbarát módon jelez vissza.

Felhasználói felület megvalósítása

A dizájn kialakításához az **Ant Design** UI könyvtárat használtuk, amely egységes stílust és előre elkészített komponenseket biztosít, mint például `Card`, `Button`, `Form` stb.

A reszponzív elrendezést a `flexbox` és `grid` rendszer biztosítja. A komponensek illeszkednek a különböző képernyőméretekhez, így az alkalmazás mobilbarát.

Állapotkezelés

Az állapotkezelésre a `useState` és `useEffect` React hookokat használtuk. A projekt mérete nem indokolta komplex állapotkezelő könyvtár (pl. Redux) használatát.

Hitelesítés

A bejelentkezés után a kapott tokent a `localStorage`-ben tároljuk, és minden védett API-hívás ezt használja hitelesítésre. A kijelentkezés során a token törlődik, és a felhasználó visszairányításra kerül a bejelentkező oldalra.

7.5. Backend

A backend logikáját REST API végpontokon keresztül érhetjük el, melyeket a routerek kezelnek. Minden entitás saját routert kapott, amely az adott erőforrás CRUD műveleteit biztosítja. Például az `/offices` útvonal alatt kezelhetjük az irodákkal kapcsolatos műveleteket: létrehozás, lekérdezés, módosítás, törlés.

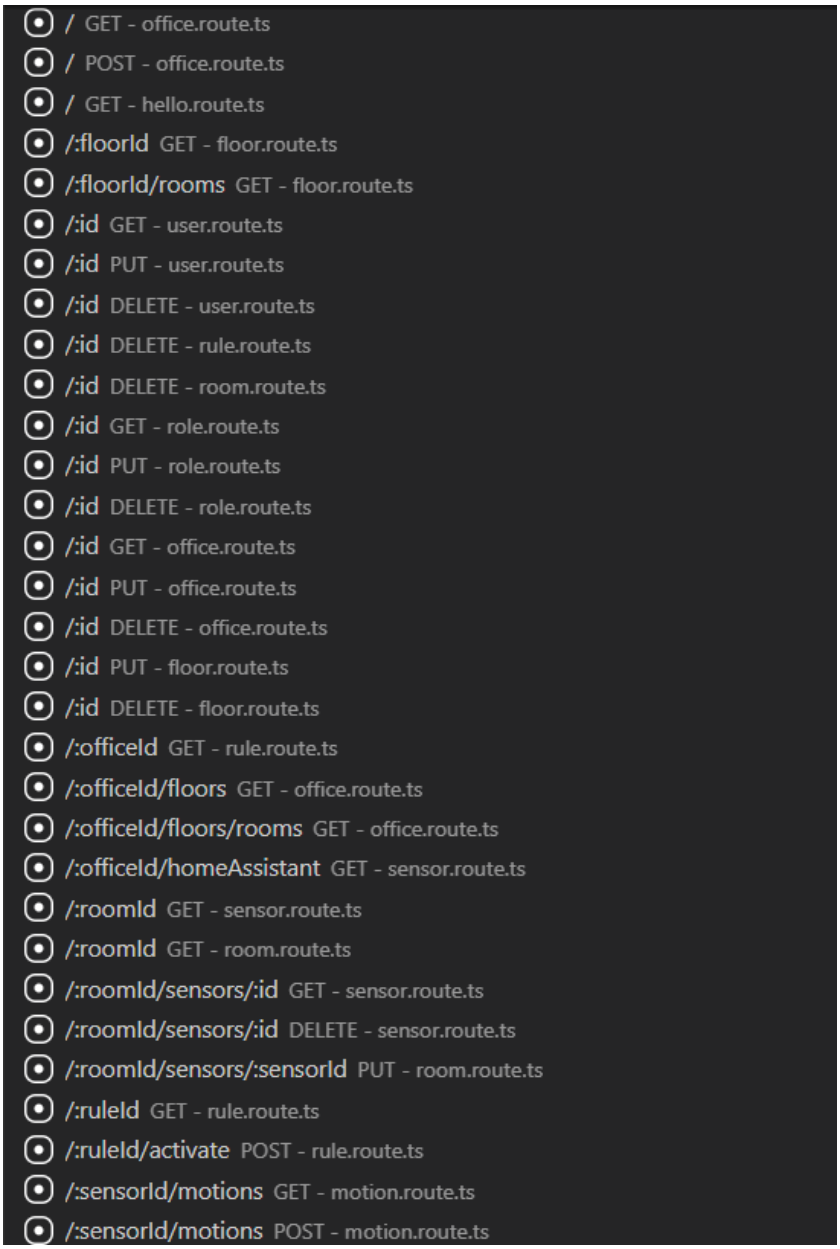
A végpontok a Hono útvonalkezelőjén keresztül kerültek regisztrálásra, és a routerek kizárólag a kérések irányításáért felelnek. A tényleges üzleti logika a szervizrétegben kapott helyet. A 9. ábrán látható néhány implementált végpont.

A szerviz réteg a következő funkciókat látja el:

- Üzleti szabályok és validációk ellenőrzése
- Adatok betöltése és módosítása az adatbázisban Mongoose modelleken keresztül
- DTO-k előállítása a válaszhoz

A biztonságos kommunikáció érdekében az API hitelesítést is tartalmaz. A felhasználó bejelentkezéskor egy tokent kap, amelyet a frontend minden kérésnél továbbít. Ez a token kerül ellenőrzésre middleware segítségével.

A mapperek biztosítják, hogy a nyers adatok konzisztens módon kerüljenek átalakításra a rendszer belső entitásaira, valamint fordítva. Ez különösen fontos a kód újrafelhasználhatósága és a fejlesztés során történő egyszerű bővíthetőség miatt.



9. ábra. Példák implementált API végpontokra

7.5.1. Termosztát szabály-végrehajtó szkript és végpont működése

Az egyik legfontosabb része a backendnek a termosztát szabályok szkriptjeinek kigenerálása, ezért külön részletezve tárgyaljuk a működését.

Backend végpont

A backend egy `POST` típusú végpontot biztosít az alábbi útvonalon, amely lehetővé teszi egy egyszerű termosztát szabály végrehajtását egy adott irodában:

`POST /execute/:officeId/simple`

A végpont megvalósítása:

```
ruleRoutes.post(
  "/execute/:officeId/simple",
  zValidator("json", ThermostatFormInputSchema),
  async (c) => {
    const { officeId } = c.req.param();
    const thermostatData = c.req.valid("json");
    try {
      const sensorOutputs = await createRuleFromThermostat(
        officeId,
        thermostatData
      );
      return c.json(sensorOutputs);
    } catch (error) {
      console.error("Failed to create thermostat", error);
    }
  }
);
```

Paraméterek:

- `officeId` – Az iroda azonosítója.
- `Request body` – Egy `ThermostatFormInputDTO` típusú JSON objektum, amely tartalmazza a szabály feltételeit (napok, időintervallum, bemeneti/kimeneti címkék, küszöbértékek stb.).

Folyamat:

1. Az érkező adatokat a rendszer validálja egy `zod` séma segítségével.
2. A `createRuleFromThermostat` függvény legenerálja a szkriptet a megadott adatok alapján.
3. A szkript kiértékeli az állapotokat, és visszatér az `output` eszközök új állapotaival.

A szkript működése

A `generateThermostatScript` függvény Python nyelven generál egy szkriptet, amely a következő logika mentén működik:

1. Konfiguráció: A bemeneti adatok Python változókba kerülnek, például:

```
days = ["Monday", "Tuesday"]
input_tags = ["temperature"]
output_tags = ["heater"]
turnOnBelow = 21
turnOffAbove = 24
startDate = "2025-04-17 07:00"
endDate = "2025-04-17 20:00"
status = ""
```

2. Idő- és napellenőrzés:

- Ellenőrzi, hogy az aktuális időpont az érvényes időintervallumba esik-e.
- Megvizsgálja, hogy a jelenlegi nap szerepel-e az engedélyezett napok között.

3. Bemeneti és kimeneti eszközök kiválasztása:

- Az `inputs` objektum alapján a címkék (`tags`) segítségével kiválasztja a bemeneti (pl. hőmérők) és kimeneti (pl. fűtőtestek) eszközöket.

4. Állapotkiértékelés:

```
if sensor['type'] == "temperature":
  if turnOnBelow and sensor['lastState'] <= turnOnBelow:
    status = "on"
  elif turnOffAbove and sensor['lastState'] >= turnOffAbove:
    status = "off"
```

5. Kimeneti eszközök módosítása:

```
for device in output_devices:
    outputs[device] = {"status": status}
```

Szkript bemenetek és kimenetek

- **Bemenet:**
  - `inputs`: A szenzoradatokat tartalmazó dictionary.
  - `output_tags, input_tags, days, threshold` értékek stb.
- **Kimenet:**

```
{
  "heater_1": { "status": "on" },
  "heater_2": { "status": "on" }
}

```

Összegzés

A szkript segítségével dinamikusan alkalmazhatóak hőmérséklet-alapú automatizációs szabályok különböző irodákra. A Pyodide-alapú szkriptek futtatása platformfüggetlen módon biztosítja a szabályok értékelését.

7.5.2. Rule Engine megvalósítása

A backend rendszer egyik fontos komponense a szabályok kiértékeléséért felelős modul (rule engine), amely lehetővé teszi, hogy a szenzoradatok alapján dinamikusan határozzunk meg kimeneti akciókat. A megoldás lényege, hogy az admin felület által definiált szabályokat futásidőben tudjuk végrehajtani Python nyelven, a JavaScript környezeten belül.

A szabályok futtatásához a `pyodide` könyvtárat használjuk, amely lehetővé teszi Python kód végrehajtását böngészőben vagy Node.js környezetben WebAssembly segítségével. A következő lépések mentén történik a szabályok kiértékelése (lásd 10. ábra):

0. Backend szerver indítása A backend szerver elindításakor automatikusan aktiválódik a `node-cron` könyvtár ütemezője, amely percenként futtatja a szabályok kiértékelését. Az ütemező minden irodára külön-külön végzi el az értékelést.

```
cron.schedule("* * * * *", async () => {
    //irodák lekérdezése
    //szabályfuttatás irodánként
});
```

1. Pyodide inicializálása A rendszer első lépésben betölti a Pyodide Python környezetet:

```
let pyodide = await loadPyodide();
```

2. Bemeneti adatok strukturálása A `SensorInputDTO` és `GeneralInputDTO` típusok alapján a rendszer összegyűjti a szenzoradatokat. Ezekből egy `Map` típusú strukturált objektum (`SerializedSensorsDTO`) készül, amely a szenzorokat és a rendszerállapotot Python-kompatibilis formátumban tárolja.

3. Python kód generálása A `mapSerializedSensorsDTOToPythonCode` függvény a JavaScript oldalon elkészített `Map` alapú struktúrából egy Python szótárat generál, ami a bemeneti adatokat jelenti a scriptnek. Ez például a következőképp nézhet ki:

```
inputs = {
  'sensor2': {
    'type': 'temperature',
    'lastTemperature': 37.3,
    ...
  },
  ...
}
```

4. Szabályok definiálása és kiértékelésre való előkészítése A szabályokat szöveges Python kódként definiáljuk egy tömbben:

```
const rules = [evalautionCode1, evalautionCode2];
```

Minden szabály egy külön Python kódrészlet, amely az `inputs` szótárral dolgozik, és egy `outputs` nevű szótárat állít elő. Például:

```
for sensor in inputs.values():
    if "tag3" not in sensor['tags']:
        continue
    if sensor['type'] == 'temperature' and
    sensor['lastTemperature'] > 37:
        outputs['room1SensorId'] = {'active': True}
```

5. Python kód végrehajtása és validálása A végső futtatandó kód a következőképpen épül fel:

```
inputs = { ... }
outputs = {}
<rule evaluation code>
outputs
```

Ez a kód átadásra kerül a `pyodide.runPython(finalCodeToExecute).toJs()` függvénynek, amely lefuttatja azt, és visszaadja az `outputs` szótár értékét JavaScript objektumként.

6. Kimenet validálása A Python oldali kimenet szintén egy szótár, amelyet a `SerializedOutputSensorsSchema` alapján validálunk:

```
const {
    success,
```

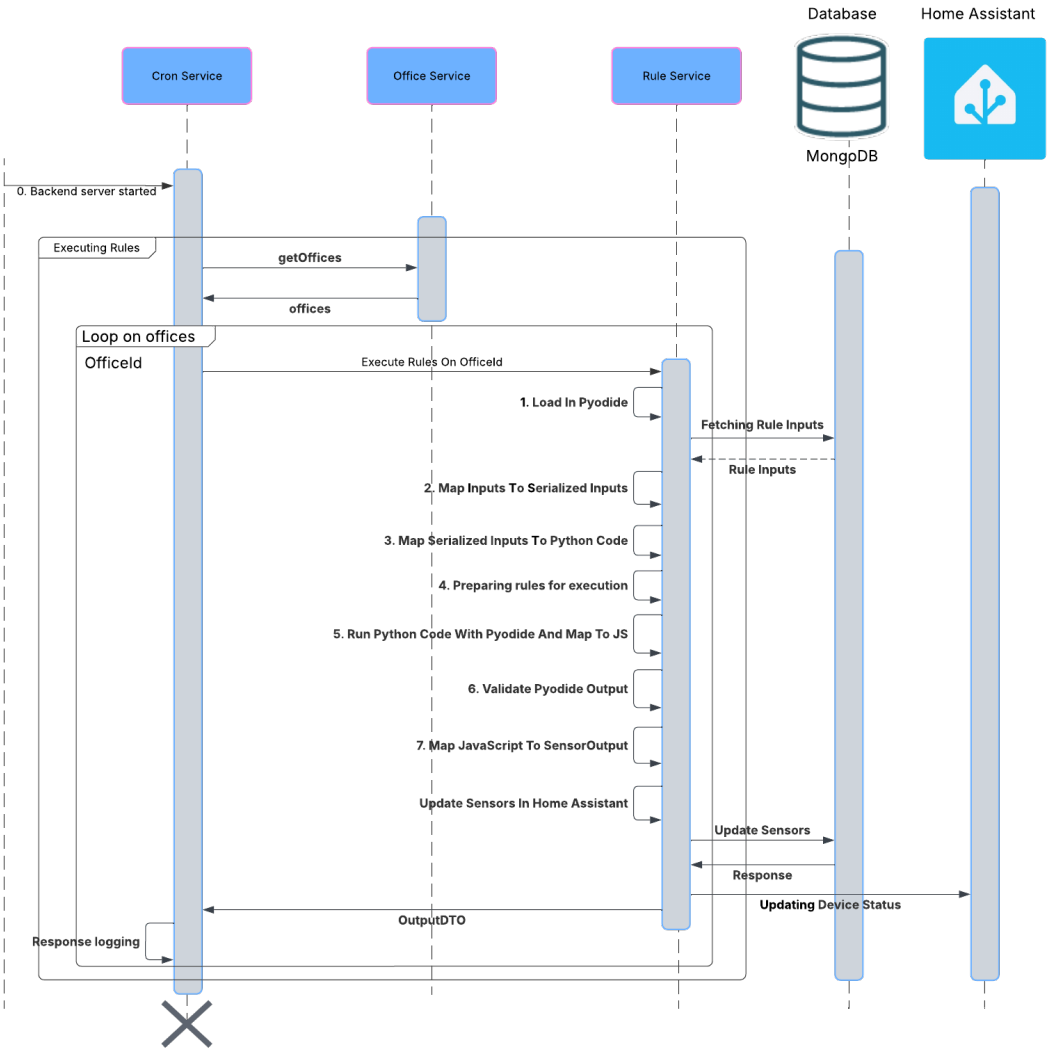
```
    error,
    data: ruleOutputData,
} = SerializedOutputSensorsSchema.safeParse(ruleOutput);
```

Amennyiben az adatok helyesek, azok bekerülnek a végleges `outputs` objektumba.

7. Kimenet átalakítása DTO-vá A `mapSerializedOutputSensorsDTOToSensorOutputDTOs` függvény segítségével a Pythonból visszatérő strukturált adatokat átalakítjuk `SensorOutputDTO` típusú JavaScript objektumokká, amelyeket a backend továbbít az adatbázis és a vezérlőegységek felé.

Összefoglalás A fenti rendszer lehetővé teszi, hogy a backend Python szabályokat definiáljon, amelyeket dinamikusan futtatunk a JavaScript környezetből, és az alapján meghatározzuk az aktuális kimeneti állapotot. A megoldás rugalmas, mivel:

- A szabályokat futásidőben lehet módosítani.
- A bemenet és kimenet validálása típusbiztos módon történik `zod` segítségével.
- Python nyelvű logika futtatása könnyen integrálható a rendszerbe.



10. ábra. Szabály rendszer szekvencia diagramm (saját ábra)

8. Összefoglaló

A dolgozat egy intelligens épületautomatizálási rendszer tervezését és megvalósítását mutatja be, amely különböző szenzoradatok alapján képes automatizált vezérlési döntéseket hozni, elsősorban hőmérséklet-szabályozás céljából. A rendszer célja, hogy irodai környezetben energiahatékony módon irányítsa a fűtési eszközöket, figyelembe véve az aktuális hőmérsékletet, a hét napjait, valamint meghatározott időintervallumokat.

A megvalósított rendszer négy fő komponensből áll:

- **Backend szolgáltatás és adatbázis**, amely TypeScript nyelven, Hono keretrendszer segítségével készült. A backend biztosítja a REST API-kat a kliensek számára, valamint kapcsolódik a MongoDB adatbázishoz, amely a szenzorok, eszközök, irodák, helyiségek, valamint a szabályok és naplóbejegyzések tárolásáért felelős.
- **Webes adminisztrációs felület**, amely React segítségével készült. Az adminfelület lehetővé teszi:
  - Irodák és helyiségek tervrajzának létrehozását és kezelését, ahol a szenzorok és eszközök vizuálisan elhelyezhetők.
  - A termosztát szabályok létrehozását grafikus űrlapkitöltéssel.
  - Egy **Script Editor** felület használatát, ahol a haladó felhasználók saját, egyedi Python-alapú automatizálási szkripteket készíthetnek, melyek ugyanúgy kiértékelésre kerülnek a rendszer által.
- **Mobilalkalmazás**, amely React Native alapokon készült. Lehetővé teszi a szenzoradatok valós idejű megtekintését, valamint a szabályok vezérlését.
- **RaspberryPI**, a rendszer fizikai alapját egy Raspberry Pi 4B képezi, amelyre a Home Assistant operációs rendszer került telepítésre a Raspberry Pi Imager segítségével. A rendszer SD kártyára történő telepítése és az eszköz hálózatra csatlakoztatása után a Home Assistant automatikusan elindul, és elérhetővé válik a helyi hálózaton keresztül.

A Home Assistant szoftverben több eszköz is integrálásra került:

- **Sonoff S26R2 WiFi Smart Plug**, amely az eWeLink integráció segítségével került be a rendszerbe.
- **Sonoff SNZB-02** hőmérséklet- és páratartalom érzékelő
- **Sonoff SNZB-03P és SNZB-03 mozgásérzékelők**

- **Sonoff SNZB-04 ajtó- és ablaknyitás-érzékelő**.
- **Zigbee 3.0 USB Dongle Plus**, amely a Zigbee eszközök integrálásáért felel.

Összességében a projekt egy moduláris és bővíthető keretrendszert kínál intelligens automatizálási feladatok megvalósítására, amely technológiailag korszerű, felhasználóbarát, és jól illeszkedik a modern okosépületek környezeti igényeihez.

Könyvészet.

Hivatkozások

[1] Andrei Decebal ENGHEȘ, Adelina Rodica BORDIANU, and Octavian Mihai GHIȚĂ. The economic benefits of smart homes. *UPB Sci. Bull., Series C*, 85(1):315–326, 2023.

[2] CrossTalk Solutions. Home assistant: The ultimate setup guide – a step-by-step tutorial. `https://www.crosstalksolutions.com/home-assistant-the-ultimate-setup-guide-a-step-by-step-tutorial/`, 2022.

Ábrák jegyzéke