Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai tanszék

Tanyi Szvetlin

Green-IoT keretrendszer okos otthonokhoz

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

KONZULENS

Dr. Varga Pál

BUDAPEST, 2017

Tartalomjegyzék

Kivonat

A Gartner tavalyi és idei, 2017-es jelentésben is az elvárási görbe csúcsán vannak az okos otthonokkal kapcsolatos megoldások [1]. A folyamatos technológiai újításoknak köszönhetően az olyan nagyvállalatok, mint a Google, az Amazon, vagy a Xiaomi is saját okos otthon megoldással lépett be a piacra az utóbbi években. Ezen eszközök felhasználói szemszögből kellemesek, könnyű őket használni és megfizethetőek. Néha hibásan illetve furcsán működnek, de mégis nagy népszerűségnek örvendenek. Ez még korántsem egy kiforrott terület, a felhasználók nincsenek pontosan tisztában az igényeikkel – tehát a vállalatok próbálják őket kitalálni, sok sikeres és sikertelen próbálkozáson keresztül. A felhasználók adatai jellemzően bekerülnek a nagyvállalatok IT felhőibe, ahol a jogviszonytól függően akár adatbányászati tevékenység is folyhat segítségükkel. Az általam tervezett és megvalósított Green-IoT keretrendszer ilyen szempontból próbál kitűnni: valós felhasználói igények lettek figyelembe véve a tervezés során, és az igényeikkel tisztában lévő felhasználónak készülnek.

Az általam készített Green-IoT keretrendszer egy összekapcsolt, de mégis moduláris rendszer, amely kis energiaigényű szenzorokból és beavatkozókból épül fel. A szenzorok megbízhatóak és pontos méréseket végeznek, a beavatkozók ezen eseményekre reagálnak, ezáltal megkönnyítve a felhasználó mindennapjait. Ezek felhasználhatóak az otthonban fellelhető fogyasztók energia-igényének monitorozására, a nem-használt eszközök kikapcsolására, vagy akár energia-igényes folyamatokhoz szükséges áram előteremtésének vezérlésére. A többi platformmal ellentétben a konfigurációs rendszernek köszönhetően bármilyen felhasználói igény kielégíthető, hiszen minden a felhasználó kezében van – legfőképp a saját adatai. Fontos kihangsúlyozni, hogy a jelenleg elterjedt rendszerekkel ellentétben az adatainkat nem osztjuk meg a keretrendszer gyártójával vagy üzemeltetőjével.

E dolgozatban bemutatásra kerülnek ennek a rendszernek a tervezési, implementációs illetve verifikációs lépései. Nagyon fontos a rendszer skálázhatósága és integrálhatósága más rendszerekkel. Az Arrowhead keretrendszerrel [2] való integrálhatóság különösen fontos szerepet kap, mivel ez egy olyan nagy európai innovációs konzorcium által fejlesztett, szabványosítás alatt álló keretrendszer, amelyet várhatóan a mindennapjainkban fogunk használni. A dolgozat a Green-IoT keretrendszer tervezésének, implementációjának és verifikációjának bemutatása mellett annak jövőbeli terveit is taglalja.

Abstract

In Gartner's last year and current 2017 report the solutions related to smart homes are at the top of the expectations curve [1]. Thanks to continuous technological innovations, large companies such as Google, Amazon, or Xiaomi have entered the market with their own smart home solutions in recent years. These devices are user-friendly, easy to use and their price is affordable. Sometimes they work in a wrong or strange way, but they still enjoy great popularity. This is far from being a well-developed area, users are not exactly aware of their needs - so companies are trying to figure them out through many successful and unsuccessful trials. User data is typically included in the IT clouds of the large corporations, where data mining can be done depending on the legal relationship. The Green-IoT framework that I have designed and implemented is trying in this respect to make a difference: real user needs have been taken into account during the design process and the framework is made for users who are aware of their needs.

The Green-IoT framework made by me, is an interconnected, but modular system, which consists of low energy sensors and actuators. They measure reliably and precisely; the actuators react upon the events thus facilitate the user's daily routine. They could be used for monitoring the energy need of the consumer devices, turning off unused appliances, generating the necessary energy for energy intensive processes from renewable energy sources. Unlike other frameworks, thanks to the configuring system every user's need can be satisfied. It is important to emphasize, that this framework does not share personal data with third parties, unlike other frameworks.

This paper deals with the planning, implementation and verification steps of this framework. Scaling and the ability to integrate with other frameworks is very essential. The ability to integrate with the Arrowhead framework is of primary importance, because this is the IoT framework developed by the European Union, which we are to use in our everyday life. The paper also discusses the future perspectives for Green-IoT framework.

1. Bevezetés

Történelmünk során mindig is megvolt az igény arra, hogy kényelmesebben éljünk, a nehéz, illetve unalmas feladatokat állatokkal, majd gépekkel végeztessük el. Napjainkban sincs ez másképp, és a technológiai fejlődésnek köszönhetően már az egyébként fizikai megterhelést nem jelentő feladatokat is könnyebbé tehetjük, vagy automatizálhatjuk. Ez a legfőbb motiváció az okos otthonok fejlesztésénél. Bele se gondolnánk, de többek között a televízió távirányítója is ezen gondolatmenet miatt született meg. Az elmúlt évtizedekben is beszerezhetőek voltak egyes áruházak polcairól a távolról vezérelhető konnektorok.

Ezen egymástól független eszközök működését egyedül a szén alapú intelligencia kötötte össze. Napjainkban jelentősen olcsóbbak lettek a számítógépek, nincs már olyan háztartás, ahol ne lenne belőlük legalább egy. Egy lépést sem teszünk meg a mobil telefonunk nélkül, miközben bele sem gondolunk, milyen erős eszköz lapul a zsebünkben.

Fontos kihangsúlyozni az okos otthonok fejlődésének másik fontos mozgatórugóját, a folyamatos összeköttetést lehetővé tévő internet fejlődését is. Azt is fontos kihangsúlyozni hogy a internet is széles körökben vált elérhetővé. Ezek mind lehetővé tették az aktív összeköttetésben lévő rendszerek létrehozását.

A folyamatos technológiai újításoknak köszönhetően az olyan nagyvállalatok, mint a Google, az Amazon, vagy a Xiaomi is saját okos otthon megoldással lépett be a piacra az utóbbi években. Ez még korántsem egy kiforrott terület, a felhasználók nincsenek pontosan tisztában az igényeikkel – tehát a vállalatok próbálják őket kitalálni, sok sikeres és sikertelen próbálkozáson keresztül.

A Google 2016-ban kiadta a Google Home [3] nevű termékét, amely a telefonunkon futó okos asszisztens tovább gondolt változata. Az eszköz bármely lakásban jól mutat és esztétikai szempontból is könnyedén a berendezés szerves részévé válhat. Az okos asszisztensen túlmutató funkció, hogy képes a Philips Hue bridge-hez kapcsolódni, így rajta keresztül a Philips Hue termékek vezérelhetőek. A lámpák ki- és bekapcsolhatók, fényerejük beállítható, illetve akár színüket is megváltoztathatjuk. Sötét téli reggeleken a természetes fényt pótolva segít könnyebbé tenni az ébredést. Éjjel mozgás érzékelés segítségével éjjeli fénynek is használhatjuk.

A Google által felvásárolt Nest nevű cég termékeivel is kompatibilis a rendszer. Termosztát segítségével otthonunk hőmérsékletét tudjuk beállítani, biztonsági kamerával otthonunkat megfigyelni, ajtónkat zárjukkal kulcs nélkül nyitni.

A Google Home-hoz IFTTT-n (IF This Than That) keresztül lehetőségünk van külső szolgáltatások hozzákapcsolására, melyek akár lehetnek a felhasználó által készítettek is.

A Google Home-unkkal az elsődleges kommunikációs csatorna hang alapú. A hívószóra reagálva az asszisztens felébred és az általunk elmondott kulcsszavak alapján megpróbálja a lehető legjobb választ adni. Nagyon fontos a vizuális visszajelzés, ezért az eszköz villog, mikor velünk kommunikál.

Az Amazon megoldása az Echo [4], amely különféle verziókban is kapható. Ezek között a kisebb méretű és barátságosabb árcédulával ellátott a Dot; a többiek vagy nagyobb hangszóróval, vagy kamerával, LCD kijelzővel rendelkeznek. Legfőbb funkciója az okos asszisztens, amely segítségével könnyedén kaphatunk kérdéseinkre választ. További előnye, hogy nagyon sok, harmadik féltől származó eszközt támogat, melyek így hangunkkal könnyedén vezérelhetővé válnak. Echo termékek esetében is a vizuális visszajelzés fontos, ezért az Echo termékek pereme kéken villog, mikor velük kommunikálunk.

Otthonunk elengedhetetlen kellékei lehetnek az okos asszisztensek, de nem csak a lakás-automatizációs szerepük fontos, hanem, hogy segítsék mindennapjainkat, napi rutinunkat, illetve kikapcsolódási lehetőséget is nyújtsanak számunkra. Időzítő beállítása főzéshez, ébresztés, időjárás jelentés lekérdezése hasznos lehet.

Kikapcsolódást jelenthet számunkra különféle zenei szolgáltatások igénybe vétele, amelyeket az asszisztensek támogatnak [7].

A következő táblázatokban (1., 2., 3., 4., és 5. táblázat) az Amazon Echo és a Google Home képességeit hasonlítottam össze áttekinthető formában.

1. táblázat - Amazon Echo és Google Home képességei – Stream szolgáltatások

Stream szolgáltatások	Amazon Echo	Google Home
Spotify Premium	✓	✓
TuneIn	✓	✓
Pandora	✓	✓
iHeartRadio	✓	✓
Youtube Red	✗	✓
Google Play	✗	✓
Amazon Music	✓	✗
Amazon Prime Unlimited	✓	✗
Audible	✓	✓
Apple Music	✗	✓
Tidal	✗	✓
Soundcloud	✗	✓

2. táblázat - Amazon Echo és Google Home képességei – Lámpák és kapcsolók

Lámpák és kapcsolók	Amazon Echo	Google Home
Philips Hue	✓	✓
LIFX	✓	✓
Lightify	✓	✗
TP-Link bulbs	✓	✓
Sengled bulbs	✓	✗
Litron Caseta	✓	✓
Belkin Wemo	✓	✓
D-Link smart plug	✓	✗
Leviton switches	✓	✗

3. táblázat - Amazon Echo és Google Home képességei – Smart Hubs

Smart Hubs	Amazon Echo	Google Home
SmartThings	✓	✓
Wink	✓	✓
Logitech Harmony	✓	✓
Insteon	✓	✗
Scout	✓	✗
Lowe’s Iris	✓	✗
Control4	✓	✗
Crestron	✓	✗

4. táblázat - Amazon Echo és Google Home képességei – Termosztát

Termosztát	Amazon Echo	Google Home
Nest	✓	✓
Ecobee3	✓	✗
Honeywell Lyric	✓	✗
Sensi	✓	✗
Netatmo	✓	✗

5. táblázat - Amazon Echo és Google Home képességei – Egyéb eszközök

Egyéb eszközök	Amazon Echo	Google Home
Roomba	✓	✗
Neato	✓	✓
Garagio	✓	✗
Gogogate	✓	✗
August lock	✓	✓
Rachio	✓	✗
BAF	✓	✗
IFTTT	✓	✓
Yonomi	✓	✗
Alarm.com	✓	✗

A Xiaomi [5] [6] máshogyan közelítette meg a dolgot. Ők készítettek egy központi egységet, amelyet Gateway-nek neveztek el, ehhez kapcsolódhatnak ZigBee-n keresztül az általuk gyártott eszközök. Letölthető hozzá egy mobil telefonos alkalmazás is, amelyen keresztül vezérelhetjük, beállíthatjuk és visszajelzést kaphatunk eszközeinkről és azok állapotáról. Lehetőség van a hőmérséklet, a levegő páratartalmának megfigyelésére, a mért adatokat az alkalmazásban lehet vizualizált formában megtekinteni. Az ajtók és ablakok állapotának monitorozása hasznos információval szolgálhat, hogy éppen mit felejtettünk el becsukni avagy, ki mikor jött haza. Mozgás érzékelésre is van lehetőség, éjjeli fényről, valamint biztonsági kamerákkal összekötve otthonunk védelméről is gondoskodhatunk. Távolról vezérelhető konnektorok esetén az sem okoz gondot, ha bekapcsolva hagytuk egyik elektronikai eszközünket, mivel könnyedén ki tudjuk azt távolról is kapcsolni. Egyszerűen összekapcsolhatjuk ébresztőóránkkal ha szeretnénk frissen főtt kávé illatára ébredni. A Xiaomi okos háztartási eszközöket is gyárt, robotporszívójuk könnyedén elvégezheti az unalmas és fárasztó házimunka egy részét. Háztartásunkban szerepet kaphat a füstjelzőjük, hűtőgépük, vízforralójuk illetve mosógépük.

Jelentős különbség a Xiaomi, a Google és az Amazon között, hogy a Google és az Amazon a személyi asszisztenssel történő integrációt tűzte ki elsődleges céljának, amíg a Xiaomi rendszere mobil illetve webes felületről történő menedzselhetősége kapott hangsúlyt.

Ezen rendszerek biztonsági aggályokat is felvetnek. Gyermekeink vagy esetleg papagájunk [38] könnyedén vásárolhatnak az asszisztensek segítségével, ha külön le nem tiltjuk ezen funkciókat. Több példa is volt rá, hogy már a tévéből kiszűrődő hangokra is aktiválódnak az asszisztensek. Azt sem tudhatjuk biztosan, hogy a cégek, akik termékeit használjuk, mely állami szervekkel illetve cégekkel osztják meg adatainkat, illetve mennyire biztonságosan tárolják azokat. Esetleges támadások esetén otthonunk fölött átveheti a támadó az irányítást.

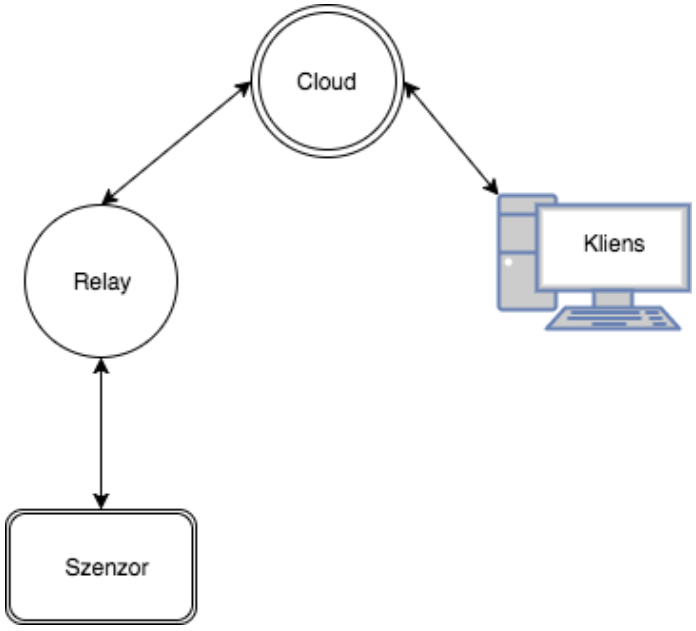
2. Tervezés

2.1 Célkitűzés

Egy olyan IoT keretrendszer megalkotása, amelynek elemei méréseket végeznek és ezek eredményeit továbbítják egy központi feldolgozónak. A központi feldolgozó eltárolja az adatokat, amelyeket aztán lekérdezhetünk, többek között mobiltelefonunk erre a célra kialakított alkalmazásának segítségével.

Követelmény, hogy a keretrendszer elemei, különösen a szenzorokat összekötő központi rendszer legyen könnyen kezelhető és kis-fogyasztású. A felhasználói felület legyen kényelmesen használható, hatékony és ergonomikus.

A rendszer alapvető funkció-halmazait tekintve top-down tervezést követtem, így három komponens-típust határoztam meg; az 1. ábrán is látható módon:



1. ábra - Rendszer modell

- Cloud: felhő alapú adatfeldolgozó rendszer, management felület;
- Relay: összeköttetést biztosít a szenzorok és a Cloud között;
- Szenzorok: ESP8266 [36] mikrokontroller alapú eszközök.

A következő alfejezetekben ezen komponensek tervezési kérdéseit tárgyalja a dolgozat.

2.2 A Cloud-alapú feldolgozó-egység

Központi feldolgozó egység, ahová befutnak a mérési adatok és az feldolgozza őket. Ez egy felhő alapú szolgáltatásként kerül megvalósításra.

Hardware:

Hardware-es megvalósításra nem lesz szükség, mivel egy virtuális gépen fog futni.

Szoftver:

Nagyon sok fajta back-end létezik, mindegyik valamiben jobb mint a másik. Ezek között .NET Windows alapú rendszereken különösen elterjedt a Ruby, vagy a NodeJS. Az én választásom a NodeJS-re [8] esett, mert a back-end teljes egésze Javascript illetve a front-end egyes részei is. Fontos szempont volt a NodeJS-hez tartozó Package Manager [13] (NPM), amellyel könnyedén tudunk programkönyvtárakat letölteni.

Front end részről nem szerettem volna nagy és nehezen elkészíthető HTML fájlokat kézzel készíteni, ezért a Pug [9] templating rendszert választottam. Ez egy nagyon egyszerű templating rendszer, előre kell definiálnunk, hogy az elemek hogyan fognak elhelyezkedni, így a kliens a Pug template-ből generált HTML fájlt kap vissza, amelyet a böngészők meg tudnak jeleníteni.

A virtuális gépen szükség lesz egy adatbázisra is. Ehhez megterveztem az adatbázis sémákat, amelyek segítségével bármilyen adatbázis kapcsolható a rendszerhez. Sokféle adatbázis létezik; a jelenleg legelterjedtebbek: Oracle DB, MySQL, PosrgreSQL, MongoDB [23]. Az első három adatbázis relációs adatbázis, amíg a MongoDB nem az. Azért választottam a MongoDB-t, mert széles körben használt, modern eszközként hivatkoznak rá, és ki akartam próbálni, kíváncsi voltam rá hogyan működik.

Kiemelt jogokkal rendelkező felhasználókat (Admin) csak kézzel tudunk hozzáadni, ez biztonsági szempontból egy megfontolt döntés.

A Cloud rendszer adatfeldolgozójának és kommunikációs képességeinek tekintetben funkcionalitás szerint szükség lesz session kezelésre, adok feldolgozására, adatok megjelenítésére, a többi komponenst beállítani tudjuk és műveleteket tudjunk rajtuk végrehajtani.

Biztonsági szempontból SSL tanúsítványra is szükség lesz, a Relay - Cloud, valamint a böngészők - Cloud közötti kommunikáció titkosítására.

2.3 A Relay csomópont

Ez egy helyi feldolgozó egység, ami a Clouddal és a szenzorokkal fog kommunikálni és összeköti őket. A szenzorokkal vezeték nélküli technológiákkal képes kommunikálni (WiFi, Bluetooth), összegyűjti és elő-feldolgozza az adataikat, majd továbbítja a Cloud felé. A szenzorok felé vezérlő-információt is továbbíthat.

Hardware:

6. táblázat – Relay csomópontként felmerülő hardver-platformok összehasonlítása

Név	Ár	Szállítási idő	Közösség általi támogatottság	Elérhető marad?
Pine64 [26]	~8000 Ft (30$)	~ 1 hónap	Részleges [28]	Talán
Rock64 [27]	~12000Ft (45$)	~ 1 hónap	Részleges [28]	Talán
Raspberry Pi 3 [14]	~11000Ft (32£)	~ 1 hét	Aktív [25]	Biztosan
Rasbperry Pi Zero W [15]	~3200Ft (9.6£)	~ 1 hét	Aktív [25]	Biztosan

Hardware-es oldalról elgondolkodtam a Pine64-en [26] Rock64-en [27], a Raspberry Pi 3-mon [14] illetve a Raspberry Pi Zero W-n [15]. Az alábbi szempontokat vettem figyelembe. Ezeket összefoglaltam a 6. táblázatban.

- Mennyibe kerül?

Fontos számomra, hogy ne legyen túl drága az eszköz, mivel ha tönkre menne akkor kevesebb anyagi befektetéssel lehet újat szerezni, bár látszik, hogy a nagyobb teljesítményű panelek többe kerülnek.

- Mekkora?

Ergonomikus szempontokat is figyelembe kell venni, mivel az otthonunk része lesz, nem mindegy hogy mennyi helyet foglal el. A Zero W 3x6cm-es méretével a legkisebb, míg a Rock64 illetve a Pi 3 bankkártya méretű, végül a legnagyobb a maga tenyérnyi méretével a Pine64.

- Mennyi idő alatt lehet újat rendelni ha elromlik?

A másik szempont a szállítási idő, ha esetleg tönkre menne a számítógép, akkor fontos mihamarabb pótolni, nem megengedhető hogy több mint 1 hónapot álljon a munka azért mert nincs munkaeszköz. Angliai webshopokból nagyjából 1 hét alatt érkezett meg a Pi, amíg a Pine64-re több hónapot kellett várni. Kínai raktárral rendelkező webshop lévén a szállítási idő 1 hónap körül van.

- Van-e a platform mögött közösség?

Köztudott hogy a Raspberry Pi [24] a legnépszerűbb SBC (Single Board Computer). Ez annak is köszönhető, hogy nagyon aktív felhasználói bázissal rendelkezik. A felhasználók élénkek a fórumokon [25] és a közösségi médiában is [32]. A Pine64-nek és Rock64-nek is van fóruma, viszont az közel sem olyan aktív [28], mint a Raspberry Pi-é.

- Mennyire stabil a cég, mi van ha csődbe mennek?

A Pine64 illetve Rock64-et gyártó cég jó terméket gyárt megfizethető áron, bár a szoftveres támogatással el vannak maradva. Másik nagy negatívum hogy a Pine64 nem tartalmaz beépített vezeték nélküli kommunikációt. Emellett a vásárlókkal való kommunikációjuk sem a legjobb. A cég bármikor csődöt jelenthet, tekintve hogy egy Kickstarter-en indult startupról van szó.

Azért gondolkodtam el a Pine64-en illetve a Rock64-en, mivel nagyon jó specifikációkkal rendelkeznek, viszont a cég stabilitása, illetve a hosszú szállítási idő kizárta őket. A Raspberry Pi-k közül végül a Zero W-ra esett a választásom, mivel beépített WiFi és Bluetooth mellett olcsóbb, méretben kisebb, kisebb az áram igénye, ezáltal elegendőnek érzem a Relay feladatainak elvégzésére. Ha tönkremegy, könnyű belőle újat szerezni akár 1 héten belül. Rendelkezik beépített WiFi-vel illetve Bluetooth-al. Vezeték nélküli hálózati kommunikációja által könnyedén megvalósítható a kommunikáció a szenzorok és Cloud között. GPIO portjai és a Bluetooth lehetőséget ad funkcionális bővítésre, illetve változatos kommunikációs protokollt használó eszközzel való kommunikációra.

Szoftver:

Szintén NodeJS-t választottam, megterveztem milyen végpontokra lesz szükség, és hogyan fog kommunikálni a szenzorokkal illetve a Clouddal. Amikor a Relay először működésbe lép, akkor beolvassa a saját konfigurációs fájlját, ha létezik ilyen és kiolvassa

belőle a nevét. Ezzel a névvel regisztrálja be magát a Cloud eszközöknél, ahol MAC címe alapján lesz azonosítva.

2.4 A Szenzorok kommunikációjának vezérlőegysége

Hardware

Vezérlő egységek közül a triviálisan adódó, széles körben használt ESP8266 [36] mellett még a Particle Photon [34] nevű mikrokontrolleren gondolkodtam el, ami bővebb funkcionalitást kínál, viszont 20$-os árcédula tartozik hozzá, az ESP8266 2-5$-os árához képest.

Emiatt vezérlő egységnek az ESP8266-os mikrokontrollerre esett a választás, mivel olcsósága és megbízhatósága mellett 2.4GHz-es WiFi-vel is rendelkezik, amely egyből megoldja a Relay és szenzor közti kommunikációt. A dolgozatban az eredmények demonstrálására a DHT11 [37] típusú hőmérőt használom.

Szoftver

A mikrokontroller kódja az Arduino IDE segítségével készült el, amellyel fel is tudjuk tölteni az eszközre.

2.5 Biztonság

A Cloud –Relay, Cloud – web kommunikáció biztonságát a HTTPS biztosítja, ehhez szükséges az SSL tanúsítvány.

A szenzorok nem kommunikálnak direktben a Clouddal, csak a Relayen keresztül tudják azt elérni.

A Relay a kapcsolattartó a szenzorok és a Cloud között. A kommunikáció a belső hálózatról kifelé csak a Relay-en keresztül történhet. A szenzorok csak helyi hálózaton működnek.

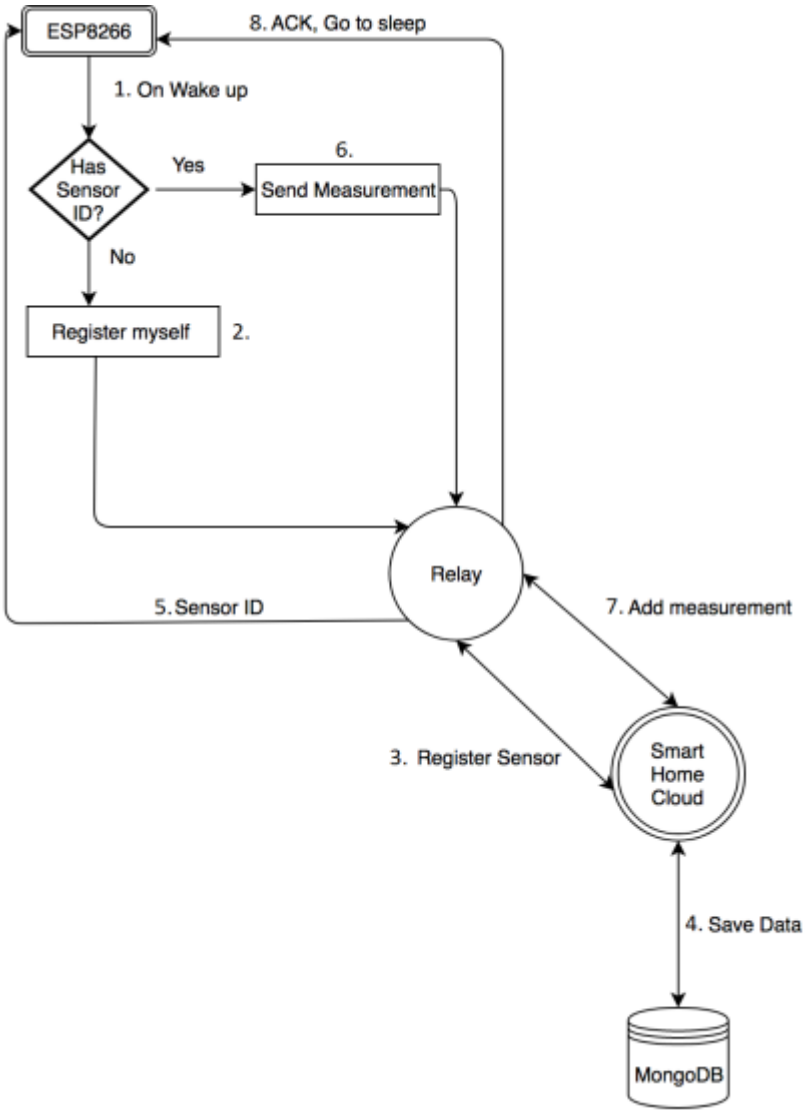
A szenzorok a saját azonosítójuk alapján autentikálják magukat. Ezt az azonosítót első működésbe lépésükkor kapják meg, ez nem változik. A Relay a MAC címe alapján azonosítja magát. Ez azért előnyös, mert SD kártya meghibásodás, adatvesztés esetén csak letölti a már meglévő konfigurációs fájlját és gond nélkül működik tovább. Az eszköz elromlása esetén a Cloud webes management felületén szükséges a MAC címet megváltoztatni. A Cloudhoz

adminisztrátorokat csak kézzel lehet felvenni az adatbázisba. Az adminisztrátorok számára bejelentkezéskor session generálódik, session alapján tudják elérni a Cloud funkcióit, különben 401-es hibával tér vissza az oldal. A jelszavak bcrypt-el [33] vannak hash-elve.

A Clouddal csak Relay és az adminisztrátorok tudnak kapcsolatba lépni. A Relay csak hozzáadni tud adatot. Az adminisztrátornak van jogköre hogy törölni is tudjon.

Cloud naplófájljaiban vannak eltárolva, hogy milyen végpontok lettek meghívva, itt visszamenőleg is megtekinthetőek a hozzáférések.

2.6 Kommunikáció

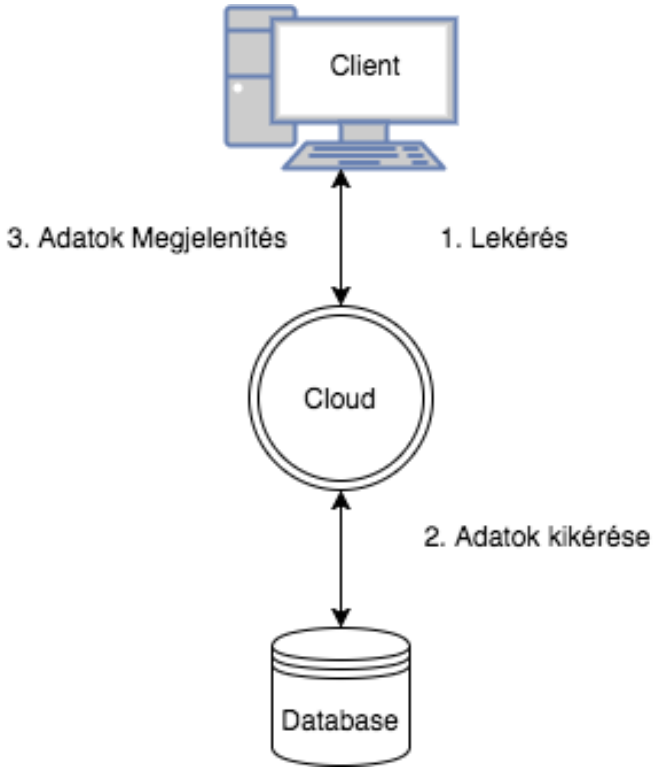


2. ábra - Komponensek közötti kommunikáció

A 2. ábrán látható, hogyan zajlik egy mérési folyamat.

1. A szenzor-modemként alkalmazott WiFi-képes mikrovezérlő, az ESP8266 amikor felébred, akkor megnézi rendelkezik-e azonosítóval.
2. Ha nem rendelkezik azonosítóval, akkor jelzi a Relay-nek hogy szeretne egyet.
3. A Relay továbbítja a kérését a Cloud felé, így tudja a Cloud hogy az adott szenzor melyik Relay-hez tartozik.
4. Adatbázisba eltárolásra kerülnek a mérések illetve a szenzorok adatai.
5. A Cloud regisztrálja mint új szenzor és visszajuttatja az azonosítóját a Relayhez, aki továbbítja az ESP8266-nak.
6. Ha van azonosítója, akkor elvégzi a mérést, eljuttatja a Relayhez.
7. A Relay továbbküldi a mérést a Cloudnak.
8. Ha sikeres volt a mérési adat feltöltése akkor elmegy aludni, egy meghatározott ideig.

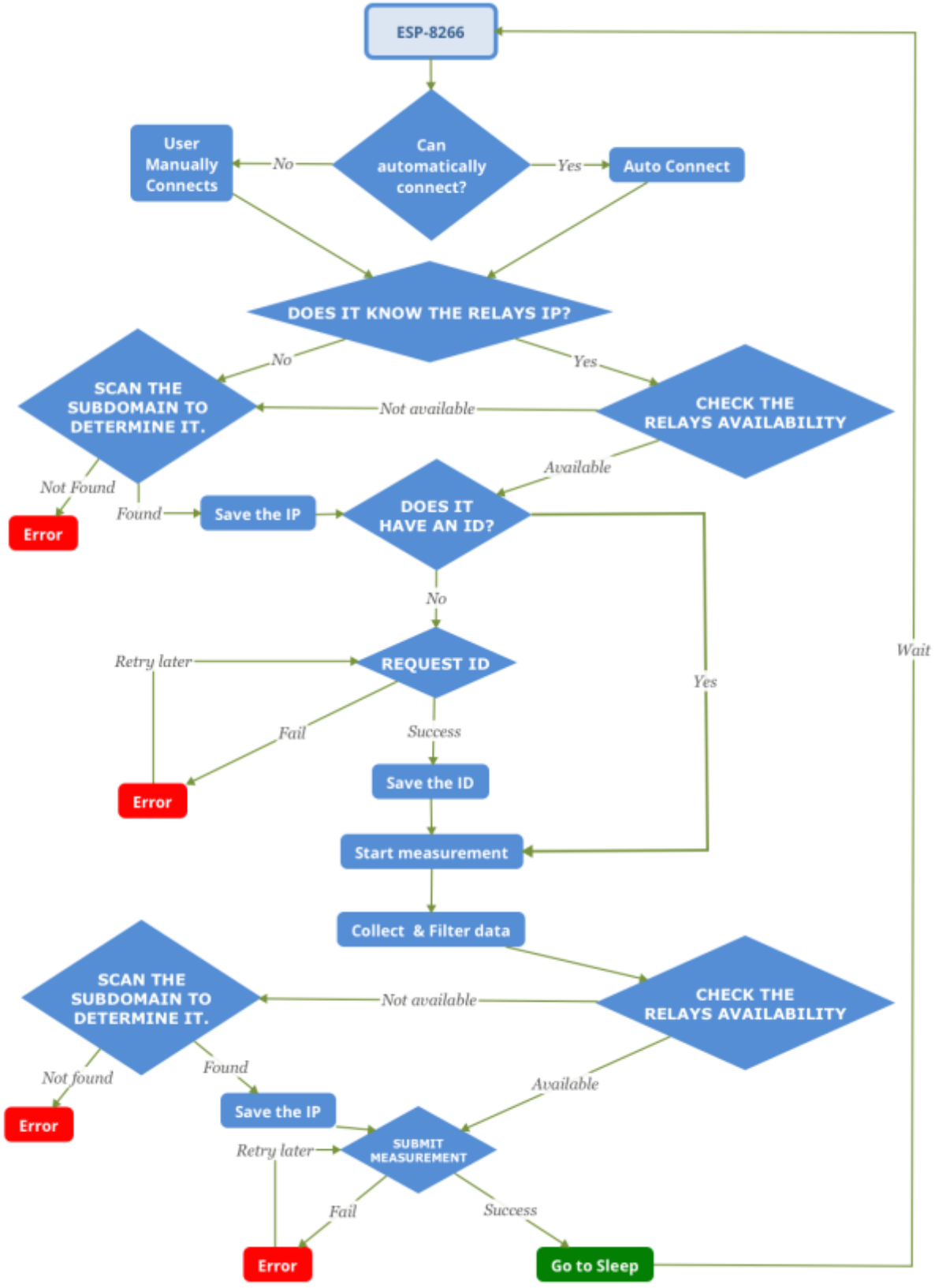
Ennek a kommunikációs folyamatnak az az előnye, hogy a mikrovezérlők nem direktben kommunikálnak a Clouddal. Ezáltal minden adat először a Relayhez fut be, amely segítségével lokális adatfeldolgozásra és konfigurálásra is lehetőség van. Lokális adatfeldolgozás lehet a szenzorok értékének változására a beavatkozókat működésbe hozni, konfiguráció meg lehet a szenzorok paramétereinek állítása. Egy példa erre: mennyi idő legyen az alvási ciklus hossza. Biztonsági szempontból is előnyös, mivel a végberendezések csak helyi hálózatról elérhetőek, így a támadók csak a helyi hálózat feltörésének árán tudhatnának rajtuk keresztül fals adatot beküldeni.



3. ábra – Kliens - Cloud kommunikáció

A 3. ábrán látható a kliens és Cloud között zajló kommunikáció folyamata.

1. A kliens egy lekérést indít a Cloud felé.
2. Ahhoz, hogy ezt a Cloud ki tudja szolgálni, le kell kérdezni az adatokat az adatbázisból. Az adatokat a Cloud visszajuttatja a kliensnek.
3. A kliens megjeleníti a kapott adatokat.



4. ábra - ESP8266 működésének folyamatábrája

2.7 Szenzor mikrovezérlőjének működése

A 4. ábrán az ESP8266 mikrovezérlő működésének folyamata látható. Az ESP8266 bekapcsolást vagy alvásból való felébredést követően bekapcsolja a WiFi modulját., Ilyenkor megvizsgálja, hogy van-e-e számára eltárolt SSID + jelszó páros az EEPROM-ban. Ha talál egy SSID + jelszó párost, akkor megpróbál kapcsolódni a hálózathoz. Ha sikertelen a kapcsolódás, ha helytelen a jelszó, vagy nincs ilyen SSID a közelben, akkor átvált AP (access point) üzemmódba. Ilyenkor a felhasználó feladata az, hogy kapcsolódjon a szenzorhoz, annak menedzsment felületén válassza ki a kapcsolódni kívánt hálózatot, adja meg a jelszavát. Ezután az eszköz újraindul.

Sikeres hálózatra kapcsolódás után az EEPROM-ból megpróbálja kiolvasni a Relay IP címét. Ha nem ismert a számára az IP címe, akkor megkeresi a hálózaton. Ez jelenleg úgy zajlik hogy az alhálózaton lévő IP címeket végigpróbálja. Megkérdezi tőlük hogy: Te vagy a Relay? Erre a célra készült a Relayen egy végpont. Ha megtalálta, akkor lementésre kerül az EEPROM-ba ez az IP cím. Ha azonban már ismert számára a Relay IP címe, akkor megnézi, hogy valóban elérhető-e azon a címen, itt is az elérhetőség ellenőrző végpont lesz meghívva.

Ezek után a szenzor kiolvassa az EEPROM-ból a saját azonosítóját, ha nem rendelkezik azonosítóval, akkor beregisztrálja magát, azaz kér egy azonosítót a Relay-től, aki a Cloud-tól kér egyet és ez lesz visszajuttatva a szenzor számára. Ez az azonosító lementésre kerül az EEPROM-ba. Később minden egyes mérésnél ezzel az azonosítóval azonosítja magát a szenzor, mind a Relay, mind a Cloud felé.

Ha a szenzor már be van regisztrálva van, akkor elkezdi a mérést. A mérés az általam meghatározott mérési módszertan alapján történik. A szenzor n darab mérést hajt végre, mérések között k időnyi szünetet hagyva. Miután befejeződött a mérési adatok gyűjtése, átlagot számolunk az n adatból. Van egy általunk meghatározott mérési tolerancia. Ha mért értékek közül van olyan amely az átlag ± tolerancia értéken kívül esik, akkor ezt a mérési eredményt hibásnak tekintjük, és eldobásra kerül. A helyesnek ítélt mérésekből újra átlagot számolunk. Az így megkapott érték lesz a mérés eredménye. Ez beküldésre kerül a Relay felé, majd ő továbbítja a Cloud felé.

Ha sikeres volt a mérési adat beküldése, akkor a szenzor kikapcsolja a WiFi modulját, ezáltal lecsökkentve fogyasztását és elmegy aludni, amiből t idő után fog felébredni. Ha sikertelen volt a mérés beküldése, akkor a hiba helyétől függően, vagy a Relay vagy a szenzor fogja ismét megkísérelni a mérés beküldését.

2.8 Relay - Cloud kommunikáció

A Relay és a Cloud közötti kommunikációt az alábbi módon tervezem megvalósítani:

1. **Relay regisztráció**

   Request: POST /api/v1/location/add

   Body: name: String (optional), mac: String, latitude: Number, longitude: Number

   - name: A relay / location beszédes neve
   - mac: A relay mac címe
   - latitude: Latitude
   - longitude: Longitude

   Response: {success: bool, name: String/null}

Amikor a Relay először működésbe lép, akkor megvizsgálja, hogy rendelkezik-e konfigurációs fájllal. Ha nem akkor be kell regisztrálnia magát. Ilyenkor beküldi a saját MAC címét, GEO IP [16] alapján a szélességi és hosszúsági körét. A hibát a válasz success flag-je jelzi, ebben az esetben később újra próbálkozik.

2. **Szenzor regisztráció**

   Request: POST /api/v1/sensor/add

   Body: name: String, type: String, position: String, location: String

   - name: Szenzor beszédes neve
   - type: Szenzor típusa
   - position: Szenzor beszédes elhelyezkedése
   - location: Relay MAC címe

   Response: {success: bool, id: number/null}

Ha a Relayhez befut egy szenzor regisztrációs kérés, akkor azt a feldolgozás után továbbítja a Cloudnak. Ilyenkor a szenzortól származik a neve, típusa, pozíciója. A Relay kiegészíti a saját MAC címével, így tudja magához kapcsolni a Relay a szenzorokat. A hibát a válasz success flag-je jelzi, ebben az esetben később újra próbálkozik.

3. **Mérés hozzáadása**

   Request: POST /api/v1/measurement/add

   Body: measurement_data: {sensor_id: String, value: Number, date: Date}

A szenzor a saját mérését elküldi a Relay-nek. A kérésben a szenzor az azonosítója alapján lesz azonosítva. Emellett a kérés tartalmazza a mérés értékét, illetve dátumát. A hibát a válasz success flag-je jelzi, ebben az esetben később újra próbálkozik.

Response: {success: bool}

4. Relay konfiguráció

Request: GET: /api/v1/config/relay

Query: mac: String

Response: {success: bool, name: String, latitude: Number, longitude: Number, position: String}

A Relay időnként megvizsgálja, hogy frissült-e a konfigurációja, ilyenkor a Cloudtól visszakapja azokat az adatokat amelyek megváltoztak. Az új adatokat elmenti magának. A hibát a válasz success flag-je jelzi, ebben az esetben később újra próbálkozik.

5. Szenzor konfiguráció letöltése

Request: GET /api/v1/config/sensor

Query: id: String

Response: {success: bool, sleep_time: Number, time_between_measurements: Number, measurement_count: Number, tolerance: Number}

A Szenzor képes arra, hogy megvizsgálja, frissült-e a konfigurációja. Ilyenkor a Cloudtól visszakapja a Relay azokat az adatokat amelyek megváltoztak és továbbítja a szenzornak. A sleep_time paraméter azt jelenti, hogy két mérés között mennyi idő teljen el. Alapértelmezett beállítás az 1 óra. A measurement_count paraméter azt jelenti, hogy hányszor történjen mérés a mérési időszak alatt. Alapértelmezett értéke 10. A time_between_measurements paraméter azt jelenti, hogy a mérés alatt mennyi időt várakozzon két mérés között. Alapértelmezetten ez 1 másodperc. Az utolsó érték a tolerance, amely azt adja meg, hogy mekkora az elfogadott differencia a mért érték és az átlag között. DHT11-es szenzor esetében ez 3 fokot jelent. Ezt úgy kapjuk meg, hogy a mérési értékekből átlagot számolunk (szum mérési értékek / measurement_count), azután azokat az mérési értékekek eldobjuk, amelyek tolerancia határon kívül esnek. A hibát a válasz success flag-je jelzi, ebben az esetben később újra próbálkozik.

3. Implementáció

3.1 A Cloud-alapú feldolgozó egység

A Cloud-alapú feldolgozó-egység kódját NodeJS-ben készítettem el EScript6-al. Azért döntöttem a NodeJS mellett, mert rendelkezem már tapasztalattal benne és kiforrott technológia, amelyhez közösség által készített programkönyvtárak letölthetőek. Implementáció során környezeti változókat is használtam, ezek olyan változót amelyek a szerver működéséhez elengedhetetlen információkat tartalmaznak. Itt kerültek eltárolásra az adatbázis hozzáférés adatai, melyik porton figyeljen a szerver, lapozásnál hány elemet jelenítsünk meg oldalanként. Környezeti változóknál két adatbázis adatait adtam meg. Egyik a teszt adatbázis, amit a unit, integration illetve load tesztek használnak, a másikba kerülnek beírásra a valódi adatok. Front end részről Pug template rendszert használtam, ami az általunk írt Pug template kódból dinamikusan HTML fájlokat generál.

3.2 Digital Ocean

Sok hosting megoldás létezik, a közismertebbek közé tartozik például az Amazon Web Services, és a Microsoft Azure. Amikor hosting-ra kerestem megoldást, akkor láttam, hogy egyetemi tanulmányaim révén jár a Github Student Pack [17], ahol megláttam, hogy Digital Ocean promóciós jelleggel 50$-t ad egyetemistáknak. Meggyőztek a weboldalunkon feltüntetett információk, és az emiatt is döntöttem úgy, hogy a Digital Ocean hosting megoldását választom. A szerver hostolási költsége havi 5$.

A Szerveren Ubuntu 16.04-fut, a Digital Ocean leírása [18] alapján állítottam be.

3.3 NGINX

Az NGINX [29] egy HTTP szerver, amelyet én routingra használok. A kéréseket irányítja végpont alapján a szervereim között. Azt is beállítottam, hogy a HTTP kéréseket irányítsa át HTTPS-re. Először is felkonfiguráltam, hogy az weboldalamra egy HTML fájlt adjon vissza, a Green-IoT keretrendszernek irányuló kéréseket irányítsa át a NodeJS-es backendhez. A helyes beállításokhoz Digital Ocean leírásait [10] [12] használtam.

3.4 UFW

Minden rendszer elengedhetetlen része a tűzfal. Az UFW-t (Uncompliated FireWall, [19]) nevű tűzfalat használtam. Beállítottam hogy az SSH (22)-es portja engedélyezve legyen. Mind IPv4-en illetve IPv6-on. Enélkül nem tudnám távolról elérni a szerveremet, karbantartani, keretrendszeremet fejleszteni. Szükséges kinyitni a HTTP (80) illetve HTTPS (443)-as portokat, e nélkül nehezebben tudnánk csak elérni a keretrendszert. Ezek mellett az NGINX-nek is szükséges kiengedni a tűzfalon, különben nem tudna működni. Az aktív beállítások a 5. ábrán láthatóak.

```
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
Nginx HTTPS                ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
Nginx HTTPS (v6)           ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
```

5. ábra – Tűzfalbeállítások UFW segítéségével

3.5 Let’s Encrypt

A Let’s Encrypt [20] egy Certificate Authority, amelynek az a célja, hogy az egész internetes forgalom HTTPS-en keresztül történjen. Ennek a célnak az élérése érdekében ingyenes SSL tanúsítványt nyújtanak bárkinek. Viszont a tanúsítványok 90 naponta lejárnak, ezzel ösztönzik a felhasználókat, hogy ne felejtsék el megújítani őket. Erre készítettek egy Certbot nevű programot, amely segítségével könnyedén lehet SSL tanúsítványt készíteni. Egy CRON job segítségével beállítottam, hogy hetente fusson le a tanúsítvány megújítás, ezáltal nem fog lejárni.

3.6 Process Manager

A PM2 [21] nevű program egy process manager. Segítségével könnyedén figyelhetjük a NodeJS szerverünk állapotát: milyen kérések futnak be és milyen válasszal tért vissza. Továbbá esetleges hibák, crash esetén újraindítja a szerverünket. Ez azért fontos, hogy ne kelljen kézzel újraindítani, ha valami hiba történik, hiszen ennek során mérési adatok maradnának ki. A hibaüzeneteket az alkalmazás a naplózó könyvtárába menti, ezekből a logokból visszamenőleg megtekinthetőek milyen kérések tértek vissza, milyen válasszal,

illetve az esetleges hibákról is találunk naplóbejegyzést. Működés közben a 6. ábrán tekinthetjük meg. Konfigurálásához a Digital Ocean leírását használtam [11].

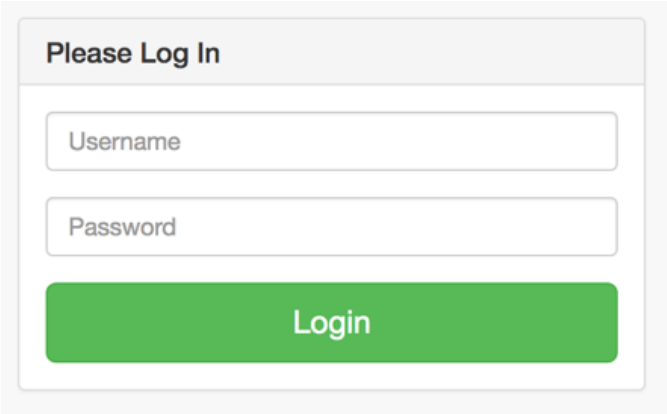
6. ábra – Process Manager

3.7 Megvalósított kommunikáció

A 2.8-as pontban meghatároztam, hogyan fogom megvalósítani a kommunikációt a komponensek között. Ezeket a végpontokat sikeresen implementáltam és a komponensek azóta is használják.

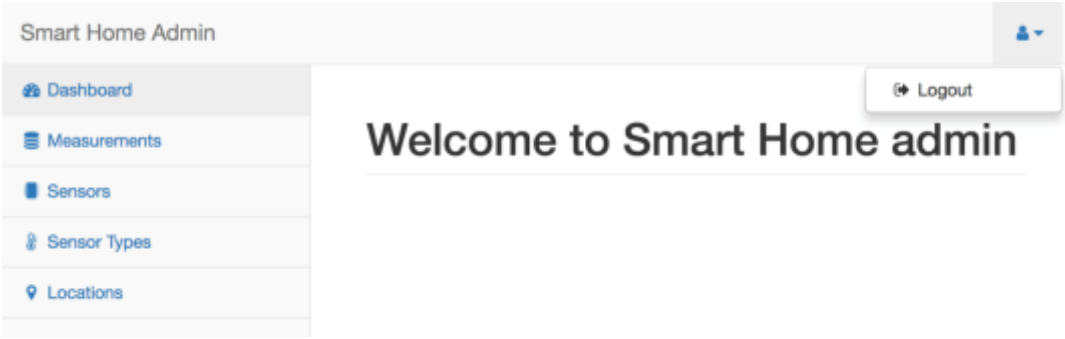
3.8 Adminisztrátor funkciók

A Green-IoT keretrendszer csak bejelentkezés után elérhető, mivel nem szeretnénk, ha bárki hozzáférhetne az otthonunkról gyűjtött adatokhoz, vagy működésbe hozná az eszközeinket.



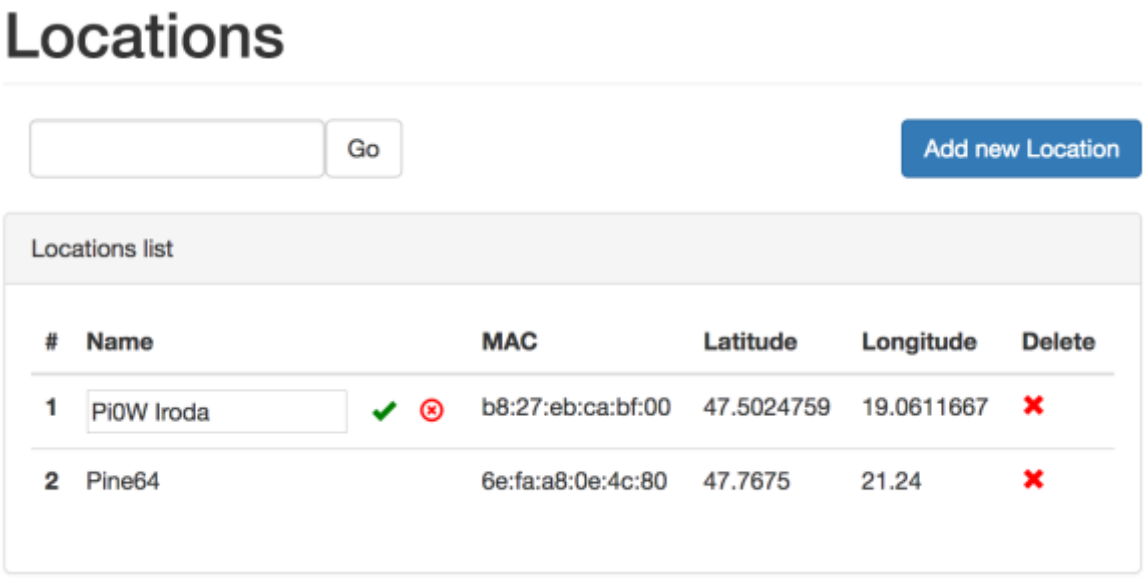
7. ábra – Bejelentkezési képernyő

Az 7. ábrán látható módon megvalósítottam a bejelentkezési képernyőt, ahol az adminisztrátornak meg kell adnia a felhasználónevét és jelszavát, ezután a Login gombra kattintva tud belépni.



8. ábra – Dashboard

Bejelentkezés után a 8. ábrán látható Dashboard köszönti az adminisztrátort, amelyen később grafikonok és táblázatok lesznek elhelyezve, amelyek segítségével átfogó képet kaphatunk. A oldalsó menüből könnyedén elérhetőek a további funkciók.

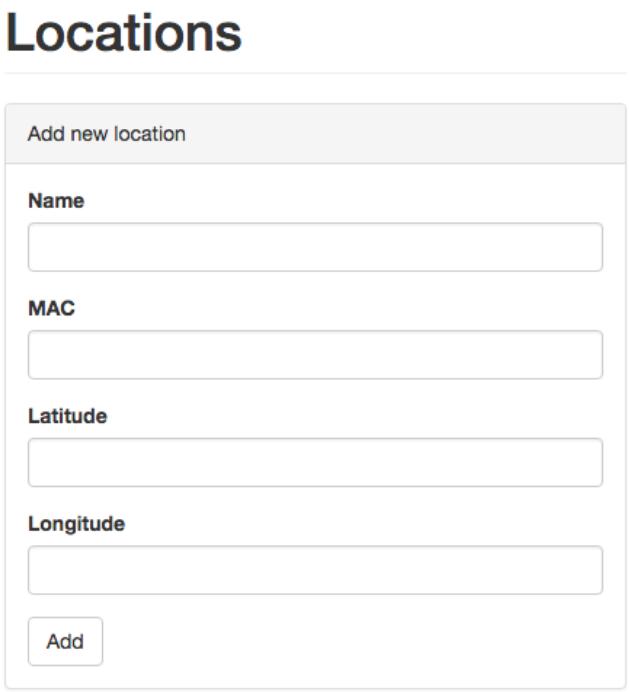


9. ábra – Relay-ek

Lehetőségünk van a Relayek listázására is, ezt a 9. ábrán láthatjuk.

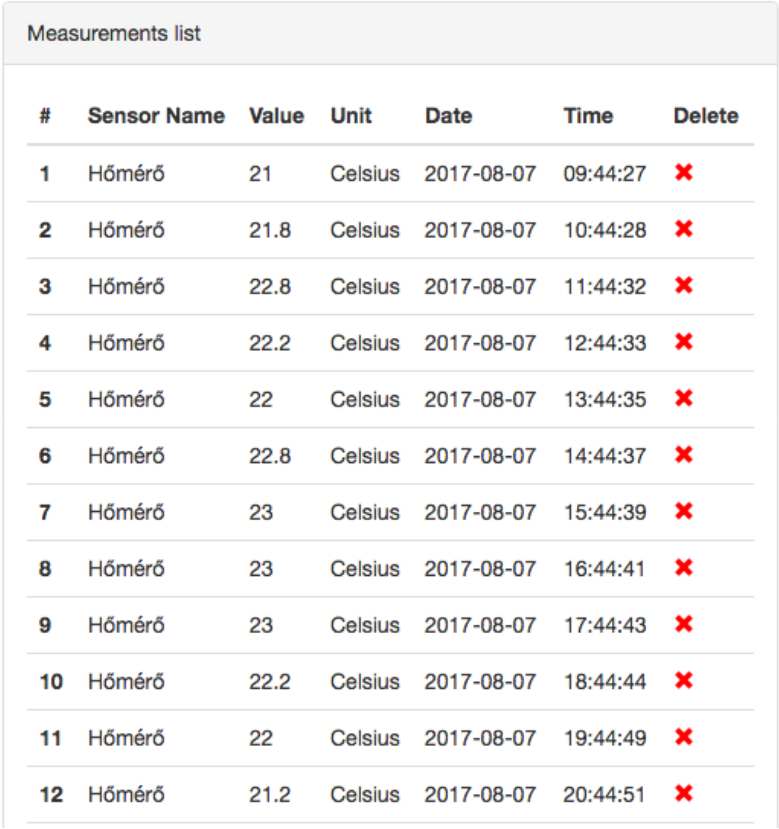
Ezen a képernyőn egy attribútumra kattintva lehetőségünk van módosítani. Jelenleg lehetőségünk van megváltoztatni a nevét, MAC címét, az eszköz szélességi kör illetve hosszúsági kör paraméterét. Ez később a Dashboard-on fog nagyobb szerepet kapni. Ott tudjuk majd térképen markerek segítségével megtekinteni a pontos helyzetüket.

Ha szeretnénk eltávolítani, akkor az X-re kattintva ezt is megtehetjük.



10. ábra – Relay hozzáadása

A 10. ábrán látható módon lehetőségünk van egy Relay-t kézzel is felvenni, ha szeretnénk pontos paramétereket megadni. Ha kihagynánk ezt a lépést, akkor ezt a Relay magától elvégzi.

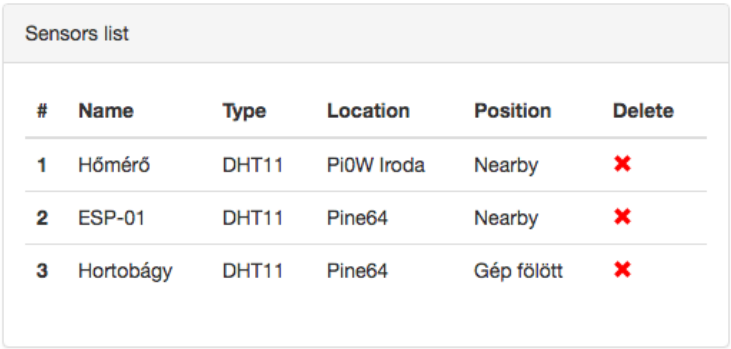


Measurements list

#	Sensor Name	Value	Unit	Date	Time	Delete
1	Hőmérő	21	Celsius	2017-08-07	09:44:27	✖
2	Hőmérő	21.8	Celsius	2017-08-07	10:44:28	✖
3	Hőmérő	22.8	Celsius	2017-08-07	11:44:32	✖
4	Hőmérő	22.2	Celsius	2017-08-07	12:44:33	✖
5	Hőmérő	22	Celsius	2017-08-07	13:44:35	✖
6	Hőmérő	22.8	Celsius	2017-08-07	14:44:37	✖
7	Hőmérő	23	Celsius	2017-08-07	15:44:39	✖
8	Hőmérő	23	Celsius	2017-08-07	16:44:41	✖
9	Hőmérő	23	Celsius	2017-08-07	17:44:43	✖
10	Hőmérő	22.2	Celsius	2017-08-07	18:44:44	✖
11	Hőmérő	22	Celsius	2017-08-07	19:44:49	✖
12	Hőmérő	21.2	Celsius	2017-08-07	20:44:51	✖

11. ábra – Mérések listázása

A Measurements menüpont alatt lehetőségünk van az eddigi összes mérést listázni, ez a 11. ábrán látható. Emellett egyes méréseket törölni is tudunk, ha nem találnánk megfelelőnek. Szintén implementáltam a lapozást is: oldalanként 30 eredmény jelenik meg, de ez a környezeti változó formájában testre szabható.

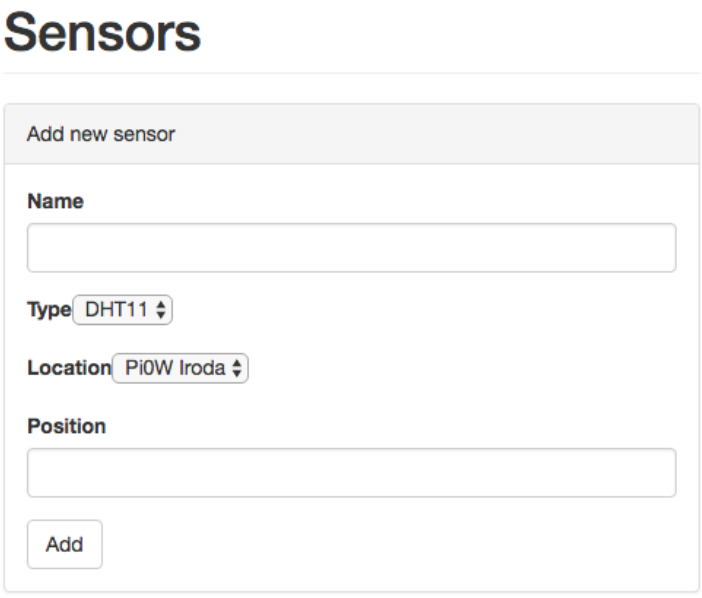


Sensors list

#	Name	Type	Location	Position	Delete
1	Hőmérő	DHT11	Pi0W Iroda	Nearby	✖
2	ESP-01	DHT11	Pine64	Nearby	✖
3	Hortobágy	DHT11	Pine64	Gép fölött	✖

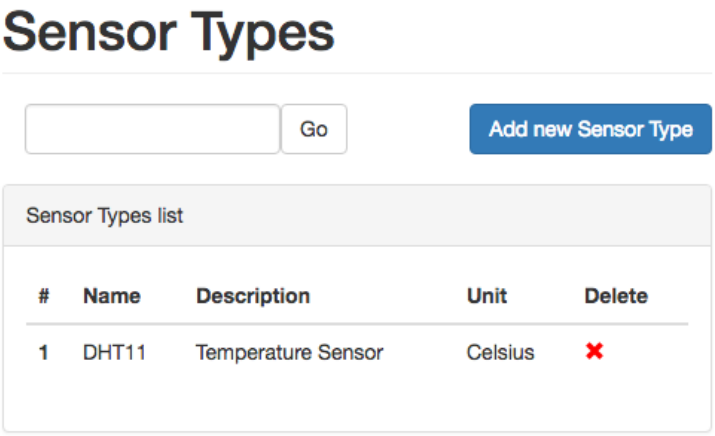
12. ábra – Szenzorok listázása

A Sensors menüpont alatt ki tudjuk listázni a szenzorokat, itt lehetőségünk van nevük, típusuk, helyik és pozíciójuk megváltoztatására. Ahogyan látható, a szenzor is törölhető a listából. Ez a 12. ábrán látható.



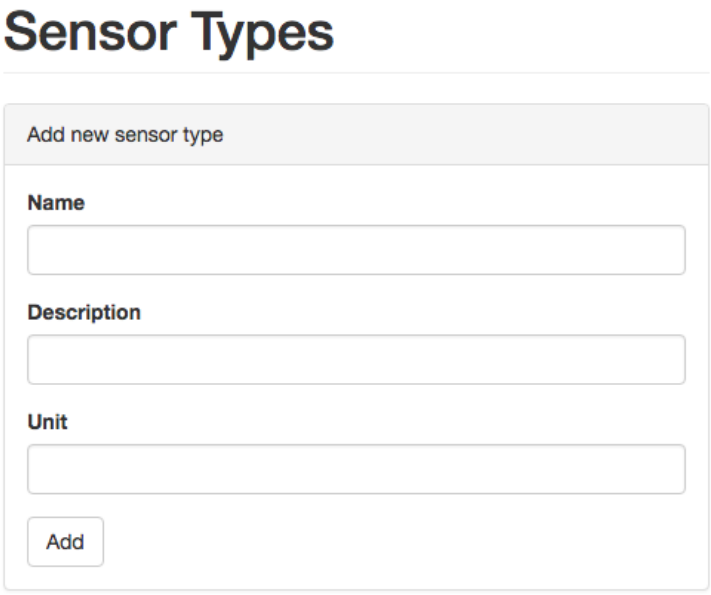
13. ábra – Szenzorok hozzáadása

A 13. ábrán fel tudunk venni új szenzort is, itt meg kell adnunk a nevét, típusát, helyét illetve pozícióját. Típusnál illetve helynél csak valós értéket adhatunk meg, ezeket a legördülő menüből tehetjük meg.



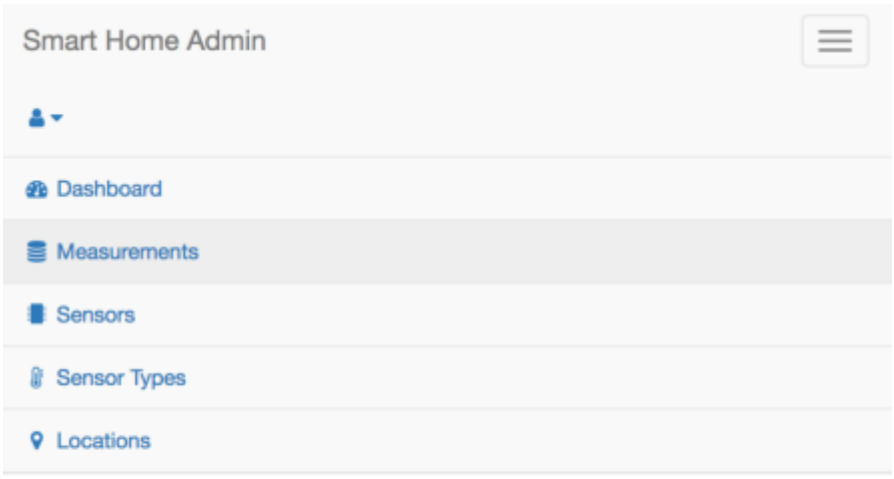
14. ábra – Szenzor típusok listázása

A 14. ábrán látható menüpontban megtekinthetjük az összes Szenzor típusunkat. Lehetőségünk van a nevük, leírásuk, illetve az általuk mért adat mértékegységének megváltoztatására. Ha nincs szükségünk a szenzorra, akkor ki is tudjuk törölni.



15. ábra – Szenzor típus hozzáadása

Új szenzor típus felvételénél kötelező megadnunk annak a nevét, és mértékegységét. Opcionálisan leírást is adhatunk neki. Ezt a 15. ábra mutatja be.



16. ábra – Mobilbarát weboldal

A 16. ábrán látható módon elkészítettem a Cloud felület mobilbarát verzióját is, ha kisebb kijelzőjű készülékről nézzük az oldalt akkor is megfelelően jelennek meg az adatok.

3.9 Relay Funkcionalitás

17. ábra – A Relay eszköz fizikai kinézete

A 17. ábrán megtekinthetjük hogyan néz ki a valóságban az eszköz. A Relay indulás után leellenőrzi, hogy be van-e regisztrálva. Ezt úgy tudja megtenni, hogy megvizsgálja a konfigurációs fájljában lévő adatokat. Ha nincsen, akkor egy kéréssel fordul a Cloud felé, regisztrálja magát. Ha a webes felületen már fel van véve mint Relay, akkor letölti a hozzá tartozó konfigurációs beállításokat.

A Relay átjáróként szolgál a beavatkozók, a szenzorok és a Cloud között. A Relay felelőssége beregisztrálni a szenzorait, méréseiket elküldeni. Később a beavatkozókat vezérelni. Fontos szerepe lesz az áramszámlánk csökkentésében: helyi adatfeldolgozó és vezérlő egység révén ki tudja majd kapcsolni a nem használt fogyasztóinkat.

3.10 Relay API

1. Relay elérhetőség ellenőrzése, megkeresése a hálózaton

   Request GET /api/v1/check/

   Response: {success: bool, ip: String }

   Annak ellenőrzésére hogy elérhető-e a Relay ezt a végpontot készítettem el. A válaszban visszatér a saját IP címével.

2. Mérés hozzáadása

Request POST /api/v1/measurement/

Body: sensor_id: String, value: Number, date: Number (optional)

- sensor_id: A szenzor egyedi azonosítója
- value: A mért érték
- date: Mérés időpontja

Response {success: bool}

Egy szenzor úgy tud mérést beküldeni, hogy a Relaynek elküldi a saját azonosítóját, a mért értéket. A dátum opcionális, ha nincs dátum akkor a Relay kitölti ezt a mezőt.

3. Szenzor regisztrálása

Request POST /api/v1/register/

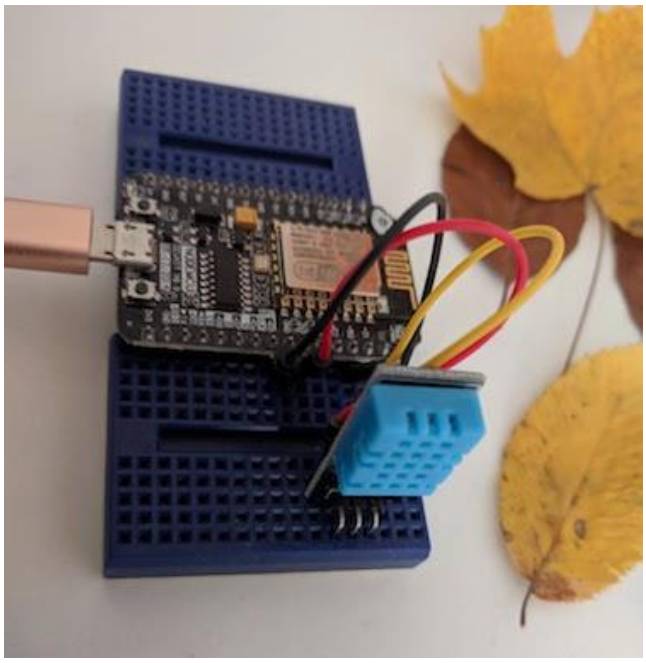
Body: type: String

- type: Szenzor típusa

Response: { error: bool/null, id: String }

Ha egy szenzornak nincs azonosítója, akkor a Relay felé jelzi, hogy szeretne egyet. Itt a szenzornak meg kell mondania, hogy milyen típusú. A Relay a Cloudtól kap egy azonosítót, amit továbbít a szenzornak, amelyet a szenzor elment.

3.11 Szenzorok bemutatása



18. ábra – ESP8266 és DHT11

A tervezés során bemutatott, a 4. ábrán megtervezett folyamatábra alapján kerültek a funkciók implementálásra. Illusztrációként jelen dolgozatban egy ESP82266 és DHT11 típusú hőmérő kerül bemutatása, amely a 18. ábrán látható.

A mérés folyamata ebben a demonstrációban a következő. A szenzor felébredés után másodpercenként mér egyszer, ezt 10-szer ismételi meg. Az így összegyűjtött mérési adatokból átlagot számolunk. Van egy megadott tolerancia, amely DHT11 típusú hőmérőnél alapértelmezett esetben ±3 fok. Ha a mért érték ezen a tolerancia határon kívül esik, akkor az a mérési adat eldobásra kerül, mivel hibás. Az ezen lépés után maradt mérési adatokat újra kiátlagoljuk, és ez lesz a mérésünk eredménye, amelyet a Relaynek továbbítunk.

Mérés beküldése után és sikeres válasz esetén 1 órára elmegy aludni az eszköz.

4. Vizsgálatok és eredmények

Az ellenőrző mérésekhez a hőmérő és a Relay 28 napig folyamatosan használatban voltak, ez az idő alatt 682 darab mérési eredményt sikerült végrehajtani. Áramszünet után újraindult a mérési folyamat. A mérési folyamatban használt átlagolás és attól való differencián alapuló hibás mérési érték szűrés effektívnek bizonyult, mivel kiugró értékek nem tapasztalhatóak.

A szenzornak 10 másodpercbe telik elvégezni a mérést, amit óránként hajt végre. Az alvás időtartamából levonásra kerül ez az érték. Ezek alapján megbecsülhetjük a hálózat késleltetését a Relay és a Cloud között. Az így megállapított hálózati késleltetés jellegzetesen 1 és 3 másodperc közé esik, ami ezen az alkalmazási területen teljesen megfelelő.

A Cloud VM a Digital Ocean szolgáltatásai által nyújtott keretein belül felfelé tetszőlegesen skálázható. Digital Ocean-ön lehetőségünk van processzort, memóriát, tárhelyet növelni, ha esetleg kevés lenne az eddigi mennyiség.

16	Hőmérő	22	Celsius	2017-08-09	06:45:56	✖
17	Hőmérő	22	Celsius	2017-08-09	07:45:58	✖
18	Hőmérő	21.2	Celsius	2017-08-09	08:46:00	✖
19	Hőmérő	22.6	Celsius	2017-08-09	09:46:03	✖
20	Hőmérő	23	Celsius	2017-08-09	10:46:06	✖
21	Hőmérő	23.4	Celsius	2017-08-09	11:46:08	✖
22	Hőmérő	23.2	Celsius	2017-08-09	12:46:10	✖
23	Hőmérő	23.8	Celsius	2017-08-09	13:46:12	✖
24	Hőmérő	24	Celsius	2017-08-09	14:46:14	✖
25	Hőmérő	24	Celsius	2017-08-09	15:46:17	✖
26	Hőmérő	24.8	Celsius	2017-08-09	16:46:19	✖
27	Hőmérő	25	Celsius	2017-08-09	17:46:21	✖
28	Hőmérő	23.4	Celsius	2017-08-09	18:46:22	✖
29	Hőmérő	23	Celsius	2017-08-09	19:46:24	✖
30	Hőmérő	22.2	Celsius	2017-08-09	20:46:26	✖

19. ábra - Irodai mérés

A 19. ábra azt ábrázolja, hogy a termosztáttal 22ºC fokra beállított irodában hogyan változott a levegő hőmérséklete a nap folyamán.

A unit és integration teszteken felül terhelés tesztet is futtattam a Cloudon és a Relayen is. Ehhez az Artillery.io [22]-t választottam.

4.1 Artillery.io

Az Artillery.io [22] egy terhelés teszteket végrehajtó program. NodeJS-ben íródott. Parancssoros alkalmazásként lehet használni. Lehetőség van saját tesztelési forgatókönyvet írni, ezt egy yaml fájllal tehetjük meg, amiben meg kell adnunk, mi az alap URL, amit tesztelni fogunk, milyen fázisok vannak. Egy fázis rendelkezik időtartammal, illetve meg kell adni hogy hány kérést küldjön másodpercenként. Azt is meg lehet adni, hogy folyamatosan növekedjen a kérések száma. Szcenáriókat is definiálhatunk, ami ez esetben egy végpont tesztelését jelenti. Ha több szcenáriót adunk meg, akkor véletlenszerűen sorsol a program hogy melyik végpontot hívja meg. Lehetőség van súlyozni a szcenáriókat, ilyenkor a nagyobb súllyal rendelkező szcenáriók gyakrabban hívódnak meg.

A teszt során 3 fázist állapítottam meg. Az első fázis során 10 másodpercig másodpercenként 5 kérést küldött a szerver felé. Második fázis során 120 másodpercig, 5-tel kezdődően és 50-el bezárólag nőtt a kérések száma másodpercenként. A harmadik fázis 600 másodpercig tartott és másodpercenként 50 kérést küldött a szervernek. Egy olyan szcenáriót határoztam meg, ami során egy új mérést küld be az eszköz. Azért egy szcenáriót határoztam meg, mert mérés hozzáadásánál mindig kell az adatbázisba írni ezáltal jelentős a terhelés.

4.2 Cloud Load teszt

```
All virtual users finished
Summary report @ 16:13:23(+0200) 2017-10-21
  Scenarios launched:  33377
  Scenarios completed: 33377
  Requests completed:  33377
  RPS sent: 45.12
  Request latency:
    min: 96.1
    max: 2743.4
    median: 114
    p95: 145.2
    p99: 380.2
  Scenario duration:
    min: 97.2
    max: 2744.9
    median: 115.7
    p95: 149.4
    p99: 382.9
  Scenario counts:
    0: 33377 (100%)
  Codes:
    200: 33377
```

20. ábra - Cloud Load teszt

A 20. ábrán látható jelentésből kiolvasható, hogy összesen 33377 kérés lett elküldve a Cloud felé. Az eredményből leolvasható, hogy minimum 97.2ms, maximum 2744.9ms, átlagosan 115.7ms a válaszidő. A kérések 95%-a 149.4ms alatt lett kiszolgálva, a kérések 99%-a 382.9ms alatt lett kiszolgálva. Az összes elküldött kérést sikeresen feldolgozta a szerver.

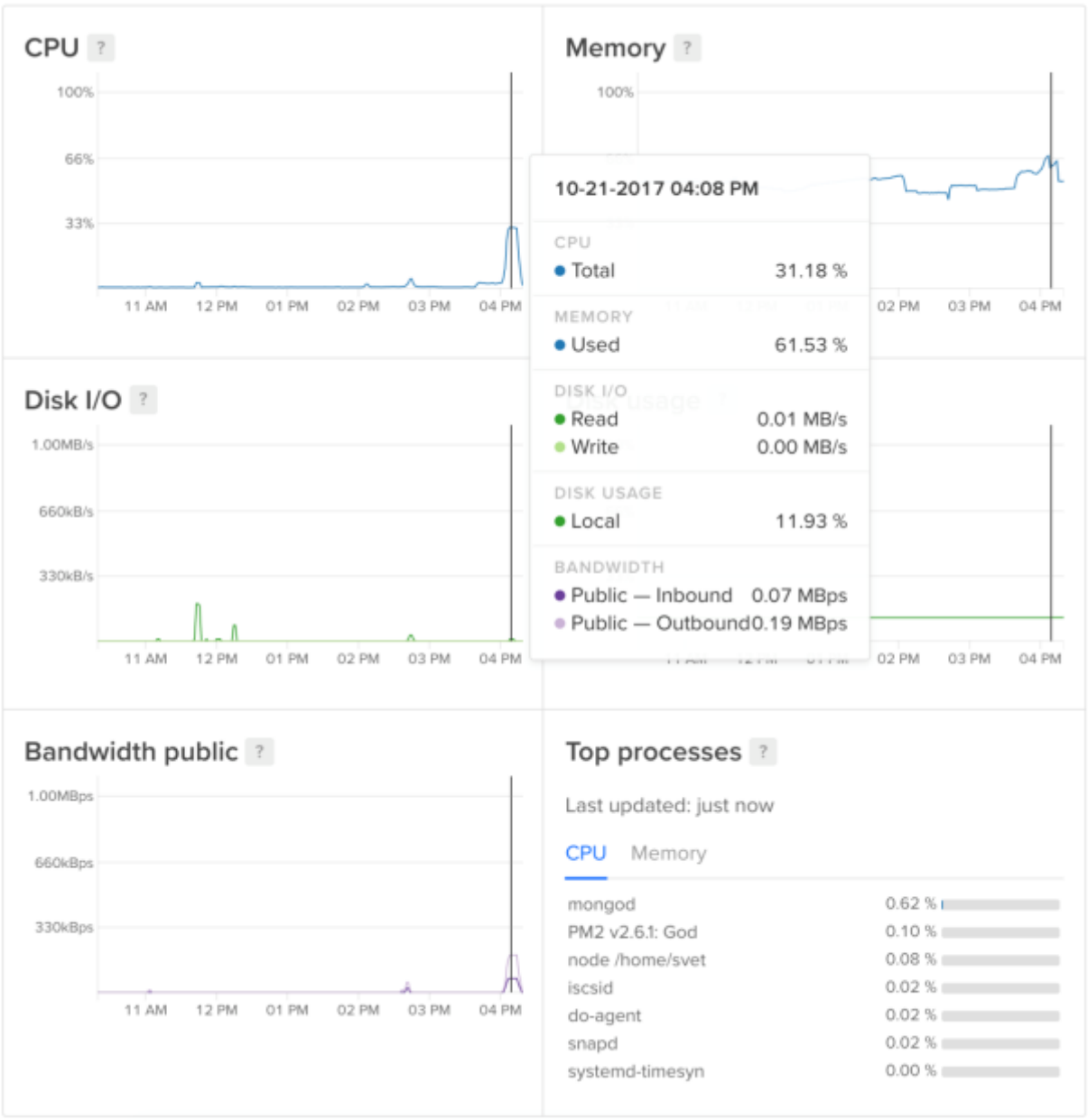
A NodeJS szerver memóriaigénye nyugalmi állapotban 63.9MB. Ez a tesztelés során felment 106.2MB-ra, a 21. és 22. ábrán látható ez az információ a process manager által. A 23. ábrán látható ahogy a CPU felhasználás 0%-ról 31.18%-ra nőtt. A szerver újabb erőforrásokat foglalt le hogy ki tudja szolgálni a beérkező kéréseket. Azt az eredményt vártuk, hogy az összes kérést szolgálja ki, és ezt sikeresen el is értük.

App name	id	mode	pid	status	restart	uptime	cpu	mem	user	watching
production	0	fork	27476	online	0	74m	0%	63.9 MB	svetlin	disabled

21. ábra – Cloud mérés előtt

App name	id	mode	pid	status	restart	uptime	cpu	mem	user	watching
production	0	fork	27476	online	0	85m	0%	106.2 MB	svetlin	disabled

22. ábra – Cloud mérés közben



23. ábra - Szerver teljesítmény-metrikák

4.3 Relay Load Teszt

Ugyanezt a tesztet a Relay csomóponton megismételve a következő eredményeket tapasztaltam.

```
All virtual users finished
Summary report @ 19:43:12(+0200) 2017-10-21
  Scenarios launched:  33363
  Scenarios completed: 2579
  Requests completed:  2579
  RPS sent: 35.88
  Request latency:
    min: 123.2
    max: 193722.2
    median: 15206.9
    p95: 87547
    p99: 91166.1
  Scenario duration:
    min: 124.4
    max: 193725.5
    median: 15208.3
    p95: 87548.1
    p99: 91167.2
  Scenario counts:
    0: 33363 (100%)
  Codes:
    200: 2578
    400: 1
  Errors:
    ECONNRESET: 848
    ETIMEDOUT: 15811
    ESOCKETTIMEDOUT: 14125
```

24. ábra - Relay Load teszt

A 24. ábrán látható jelentésből kiolvasható, hogy összesen 33363 kérés lett elküldve a Relay felé. Az eredményből leolvasható, hogy minimum 124.4ms, maximum 193725.5ms, átlagosan 15208.3ms a válaszidő. A kérések 95%-a 87548.1ms alatt lett kiszolgálva, a kérések 99%-a 91167.1ms alatt lett kiszolgálva. Az összes elküldött kérésből csak 2578-at sikerült feldolgoznia a Relaynek és továbbítani a Cloudnak. Azt vártam, hogy az összes kérést sikerül majd feldolgozni, ezzel ellentétes eredményt kaptam.

A 25. ábrán látható hogy a NodeJS szerver memóriaigénye nyugalmi állapotban 38.5MB. Tesztelés során ez felment 193.5MB-ra, a 26. ábrán látható. A CPU felhasználás 1.6%-ról 69.5%-ra, ezeket az értékeket a 27. illetve 28. ábrán tekinthetjük meg.. A Raspberry Pi Zero W szűkös memóriája miatt nem tudott tovább skálázódni, feltorlódtak a csomagok. Ezáltal nem tudta kiszolgálni a kéréseket.

App name	id	mode	pid	status	restart	uptime	cpu	mem	user	watching
production	0	fork	645	online	0	26s	0%	38.5 MB	pi	disabled

25. ábra - Relay mérés előtt

App name	id	mode	pid	status	restart	uptime	cpu	mem	user	watching
production	0	fork	642	online	0	7m	40%	193.5 MB	pi	disabled

26. ábra - Relay mérés közben

```
top - 18:34:52 up 7 min,  1 user,  load average: 0.06, 0.11, 0.08
Tasks:  94 total,   1 running,  93 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.6 us,  1.9 sy,  0.0 ni, 96.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

27. Relay CPU használat mérés előtt

```
top - 19:30:55 up 4 min,  1 user,  load average: 1.69, 0.91, 0.39
Tasks:  98 total,   2 running,  96 sleeping,   0 stopped,   0 zombie
%Cpu(s): 69.5 us, 21.9 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  8.7 si,  0.0 st
```

28. Relay CPU használat mérés közben

A Raspberry PI 0 W Relay körülbelül 10 kérést tud kiszolgálni másodpercenként. Mivel a teszt egészen 50 kérés / másodpercig felment, a feldolgozatlan kérések összetorlódtak és timeoutoltak. Jelenleg a néhány hozzákapcsolt szenzort rendben ki tudja szolgálni, de intő jel a jövőre nézve.

4.4 Árak

A Cloud jelenleg a Digital Ocean legolcsóbb havi 5$os virtuális gépén fut, amely 1 Intel(R) Xeon(R) CPU E5-2650L v3 @ 1.80GHz-es processzor maggal, 512MB RAM-mal és 20GB SSD-vel rendelkezik, ez a csomag a 29. ábrán látható.

29. ábra Havi 5$-ért cserébe ezt kapjuk a Digital Ocean-nél

Jelenleg csak WiFi-n keresztül zajlik a kommunikáció, de a Raspberry Pi Zero rendelkezik beépített 4.0-s Bluetooth modullal, ezáltal könnyedén kapcsolódhatnak majd hozzá további eszközök. Továbbá bármilyen más kommunikáció csatorna megvalósítható a GPIO pinek segítségével.

A Relay jelenleg egy Raspberry Pi Zero W-n fut amely 9.6£-ba kerül, vásároltam hozzá egy tokot is 5£-ért a postaköltség 4£-ba került.

Az ESP8266 3.2$-ba került. A DHT11-es hőmérő 1.11$-ba került

Összesen: körülbelül 7300 forint.

4.5 Külső rendszer kapcsolat

Fontos szempont az Arrowhead [35] keretrendszerrel való integrálhatóság. Mivel ez egy európai innovációs konzorcium által fejlesztett, szabványosítás alatt álló keretrendszer, amelyet várhatóan a mindennapjainkban fogunk használni. Az Arrowhead által okos házunk hozzákapcsolható a városi infrastruktúrához.

5. Kiberbiztonság

Egy ilyen keretrendszer esetében, amely az otthonunkat irányítja, nagyon fontos, hogy illetéktelen hozzáférések ne történjenek. Azon felül, hogy szenzitív adatokat nyerhetnek ki, sok kellemetlenség érhet minket.

Már csak a saját szerverem naplófájljait vizsgálva is észrevettem, hogy naponta több ezer illetéktelen próbálkozás van.

Az UFW tűzfalon csak az 22 (SSH), 80 (HTTP), 443 (HTTPS)-as port van kiengedve. A 80 (HTTP)-as port a 443 (HTTPS)-ra irányít át. A 30. ábrán látható, hogy a támadók folyamatosan nézik, mely portok vannak nyitva. Megoldást jelenthet, hogy ha egyszer rossz portot hívnak, akkor kitiltjuk az IP címüket.

```
SRC=197.248.194.234 DST=207.154.231.45 LEN=44 TOS=0x00 PREC=0x00 TTL=242 ID=3425 PROTO=TCP SPT=58763 DPT=143
SRC=77.72.82.80 DST=207.154.231.45 LEN=40 TOS=0x00 PREC=0x00 TTL=251 ID=44019 PROTO=TCP SPT=40709 DPT=4804 WI
SRC=74.82.225.31 DST=207.154.231.45 LEN=40 TOS=0x00 PREC=0xC0 TTL=52 ID=53272 PROTO=TCP SPT=61044 DPT=23 WIND
SRC=104.236.216.235 DST=207.154.231.45 LEN=57 TOS=0x00 PREC=0x00 TTL=249 ID=54321 PROTO=UDP SPT=59642 DPT=534
SRC=24.179.244.59 DST=207.154.231.45 LEN=40 TOS=0x00 PREC=0x00 TTL=51 ID=16059 PROTO=TCP SPT=64679 DPT=23 WIN
SRC=31.163.100.182 DST=207.154.231.45 LEN=40 TOS=0x00 PREC=0x00 TTL=250 ID=15516 PROTO=TCP SPT=38324 DPT=23 W
SRC=186.46.84.36 DST=207.154.231.45 LEN=60 TOS=0x00 PREC=0x00 TTL=50 ID=23581 DF PROTO=TCP SPT=38660 DPT=8086
SRC=128.199.69.55 DST=207.154.231.45 LEN=40 TOS=0x00 PREC=0x00 TTL=246 ID=30359 PROTO=TCP SPT=53809 DPT=4066
```

30. ábra – UFW Napló

31. ábrán látható ahogy SSH-n keresztül is folyamatosan próbálkoznak belépni a szerverre. Viszont a jelszóval történő azonosítást kikapcsoltam, csak kulccsal lehet bejelentkezni.

```
fatal: Unable to negotiate with 103.89.88.66 port 50345: no matching key exchange method found. Their off
error: maximum authentication attempts exceeded for root from 113.215.198.100 port 60804 ssh2 [preauth]
Disconnecting: Too many authentication failures [preauth]
error: maximum authentication attempts exceeded for root from 122.116.75.252 port 47786 ssh2 [preauth]
Disconnecting: Too many authentication failures [preauth]
Invalid user biology from 61.90.150.242
input_userauth_request: invalid user biology [preauth]
Connection closed by 61.90.150.242 port 61449 [preauth]
fatal: Unable to negotiate with 103.89.89.160 port 56772: no matching key exchange method found. Their of
Did not receive identification string from 103.89.88.63
Received disconnect from 121.18.238.125 port 47447:11:  [preauth]
Disconnected from 121.18.238.125 port 47447 [preauth]
error: maximum authentication attempts exceeded for root from 181.211.92.212 port 42479 ssh2 [preauth]
Disconnecting: Too many authentication failures [preauth]
```

31. ábra - SSH Napló

Sok végpont feltérképező támadás is szerepel az NGINX hozzáférési naplójában. A naplóról egy képernyőkép látható a 32. ábrán.

```
http://207.154.231.45:80/php-myadmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/phpmy-admin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysqladmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysql-admin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/phpmyadmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/phpMyAdmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/sysadmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/sqladmin/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/db/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/web/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/admin/pMA/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysql/pma/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysql/db/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysql/web/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
http://207.154.231.45:80/mysql/pMA/ HTTP/1.1" 301 0 "-" "Mozilla/5.0 Jorgee"
```

32. ábra - NGINX Napló

A naplófájlok megtekintése során nem tapasztaltam illetéktelen hozzáférésre utaló jelet.

6. Jövőbeli tervek

6.1 Relay továbbfejlesztése

A Relay a közeljövőben jelentősen ki lesz bővítve funkcionálisan. Rendelkezni fog saját szenzorokkal, hogy akár magában is használható legyen. Ezen túl kerül majd bele PIR presence szenzor [31], illetve ws2812b [30] LEDenként vezérelhető LED szalag, amely visszajelzést tud majd adni a felhasználónak, és éjjeli fényként is használható. Mikrofon és hangszóró is kerül a Relay mellé, hogy tudjunk kommunikálni az általam fejlesztett személyi asszisztenssel. Ezek egy általam tervezett 3D nyomtatott tokba fognak kerülni. Emellett készül ehhez is webes management felület, amin szövegesen, illetve hangunkkal tudjuk majd konfigurálni illetve vezérelni az eszközeinket. Az funkcionalitás által beavatkozók is hozzá kapcsolhatók a rendszerhez.

6.2 Cloud továbbfejlesztése

A Dashboardon különféle grafikonokon lehet majd megtekinteni az adatokat. A távoli konfigurációs paraméterek bővítése által részletesebb konfigurációs lehetőségünk lesz az egyes eszközökön, ezáltal nem kell majd firmware-t frissíteni rajtuk, ha meg akarunk valamit változtatni.

6.3 További eszközök

A Green-IoT keretrendszer a környezet-tudatosság jegyében készül, jövőbeli célkitűzésem, hogy ESP8266-os WiFi modulokkal WiFis konnektorok fognak készülni, amelyen azon felül, hogy távvezérelhetőek és monitorozzák az áramfogyasztásunkat, a jelenlét (presence) szenzorok segítségével kikapcsolhatóak a nem használt eszközök. Ezzel sok áramot meg lehet spórolni. Napelemek integrálása a rendszerbe izgalmas feladat lesz, de az általuk termelt áramot akkumulátorokban lehet tárolni és azt felhasználni áramszünet esetén, illetve számláink csökkentésére. Mindezek mellett általam tervezett áramkör is fog készülni, amely kisebb formában fogja tartalmazni ugyanezt a funkcionalitást, kompakt 3D nyomtatott tokkal. Új szenzorok mellett beavatkozók is elkészítésre kerülnek.

7. Összefoglalás

Az okos otthon megoldásokat kínáló nagyvállalatok megoldásait összevetve megállapítottam azok előnyeit és hátrányait. Ezeket is figyelembe véve egy több előnnyel rendelkező irányt jelöltem ki a saját rendszeremnek.

- Hang alapú vezérelhetőség;
- Relay mint éjjeli fény;
- Beavatkozók működésének irányelvei;
- Megújuló energiaforrások használata;
- Biztonságra vonatkozó irányelvek.

Külön kiemelendő az a biztonsági szempontból jelentős előny, hogy a rendszer által gyűjtött adatokkal a felhasználó rendelkezik; ezek nem kerülnek mások által felhasználásra.

A dolgozatban bemutattam a keretrendszerem tervezésének jellemzőit, a rendszer felépítését. Itt kitértem a Cloud, Relay illetve a szenzorok kommunikációjára, bemutattam a saját API-mat. Részletesen leírtam egy tipikus szenzorhoz kötődő jellegzetes működési folyamatot. Implementáltam és dokumentáltam a Cloud, Relay és szenzorok felépítését illetve működését.

Az általam tervezett rendszert funkcionális és terheléses tesztekkel is verifikáltam. A méréseim alapján azt találtam, hogy a Cloud a jövőbeli igényeket is ki tudja elégíteni, viszont a Raspberry Pi Zero W alapú Relay nem biztos, hogy megfelelő választás marad a jövőbeli tervekre nézve.

Dolgozatomban dokumentáltam a keretrendszer verifikációjának főbb elemeit, kitértem a kiberbiztonságra is, mivel ez is elengedhetetlen része a keretrendszernek. Megvizsgáltam a szerverem naplóbejegyzéseit: a biztonsági beállításaim megfelelőnek bizonyultak és illetéktelen hozzáférés nem történt.

A bemutatott, saját tervezésű és fejlesztésű keretrendszer egy, a követelményeknek megfelelő, alaposan átgondolt, jól felépített projekt, amelyet a jövőben könnyen tudok majd igényeimnek megfelelően bővíteni. Jelenlegi állapotában még nem versenyképes a nagy cégek megoldásával, amelyekbe több ezer mérnökévnyi időt és több milliárd dollárnyi pénzt

invesztálnak. A projekt jövőbeli fejlesztéseit is részletesen taglaltam a dolgozatban, ezeknek a megvalósításával fogok az elkövetkezendő időben foglalkozni.

Irodalomjegyzék

Minden webes hivatkozás 2017. 10. 26.-án elérhető volt.

[1] Gartner Inc. – Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017 http://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/

[2] J. Delsing (ed.) – IoT Automation: Arrowhead Framework, CRC Press, 2017.

[3] Google Home hivatalos oldala, https://store.google.com/us/product/google_home?hl=en-US

[4] Amazon Echo hivatalos oldala, https://www.amazon.com/dp/B06XCM9LJ4/ref=fs_ods_fs_ha_dr

[5] Xiaomi Okos otthon hivatalos oldala, https://xiaomi-mi.com/mi-smart-home/

[6] Xiaomi Okos otthon teszt, https://rendeljkinait.hu/xiaomi-smart-home-teszt/

[7] Google Home, Amazon Echo összehasonlítás, https://thewirecutter.com/reviews/amazon-echo-vs-google-home/#what-to-look-forward-to

[8] NodeJS, https://nodejs.org/en/

[9] Pug, https://pugjs.org/api/getting-started.html

[10] Digital Ocean Nginx, SSL, https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-16-04

[11] Digital Ocean Process Manager, https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-14-04

[12] Digital Ocean NGINX telepítés és konfiguráció, https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04

[13] Node Package Manager, https://www.npmjs.com

[14] The Pi Hut Rasberry Pi 3, https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-3-model-b

[15] The Pi Hut Raspberry Pi Zero W, https://thepihut.com/collections/raspberry-pi-store/products/raspberry-pi-zero-w

[16] Geo IP, http://freegeoip.net

[17] Github Student Pack, https://education.github.com/pack

[18] Digital Ocean Ubuntu beállítás, https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04

[19] UFW, https://help.ubuntu.com/community/UFW

[20] Let's Encrypt, https://letsencrypt.org

[21] PM2, http://pm2.keymetrics.io

[22] Artillery.io, https://artillery.io

[23] MongoDB, https://www.mongodb.com

[24] Raspberry Pi, https://www.raspberrypi.org

[25] Raspberry Pi Fórum, https://www.raspberrypi.org/forums/

[26] Pine64, https://www.pine64.org

[27] Rock64, https://www.pine64.org/?page_id=7147

[28] Pine64 és Rock64 fórum, https://forum.pine64.org

[29] NGINX, https://www.nginx.com/resources/wiki/

[30] WS2816B Led szalag, https://www.pololu.com/product/2547

[31] PIR Presence Szenzor, https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor?view=all

[32] Raspberry Pi Twitter, https://twitter.com/Raspberry_Pi

[33] Bcrypt, https://www.npmjs.com/package/bcrypt

[34] Particle Photon, https://store.particle.io/products/photon

[35] Arrowhead, http://www.arrowhead.eu

[36] ESP8266, https://en.wikipedia.org/wiki/ESP8266

[37] DHT11, https://www.adafruit.com/product/386

[38] Papagáj Amazonon vásárol, http://www.foxnews.com/tech/2017/09/21/parrot-goes-viral-after-placing-amazon-order.html