

Budapesti Műszaki és Gazdaságtudományi Egyetem

Méréstechnika és Információs Rendszerek Tanszék

**IoT alapú otthonautomatizálás megvalósítása ESP8266 használatával**

Diplomaterv

**Murányi Bence**

II. évf mérnökinformatikus szakos hallgató

MSc Intelligens rendszerek szakirány/ágazat

Konzulens:

Dr. Györke Péter

(Méréstechnika és Információs Rendszerek Tanszék)

Tartalomjegyzék

[Abstract 5](#_Toc483054116)

[Összefoglaló 6](#_Toc483054117)

[1 Bevezetés 7](#_Toc483054118)

[1.1 Az IoT-ről 7](#_Toc483054119)

[1.2 Intelligens otthon 7](#_Toc483054120)

[1.3 ESP8266 8](#_Toc483054121)

[1.4 ESP-12 tesztpanel 10](#_Toc483054122)

[2 Irodalomkutatás 11](#_Toc483054123)

[2.1 Otthonautomatizálási rendszerek 11](#_Toc483054124)

[2.1.1 IContrall 11](#_Toc483054125)

[2.1.2 SmartThings 12](#_Toc483054126)

[2.1.3 OpenHAB 13](#_Toc483054127)

[2.1.4 Domoticz 14](#_Toc483054128)

[2.1.5 Amazon Echo 15](#_Toc483054129)

[2.2 Alkalmazott technológiák és keretrendszerek 16](#_Toc483054130)

[2.2.1 JSON 17](#_Toc483054131)

[2.2.2 HighStock 18](#_Toc483054132)

[2.2.3 Bootstrap 19](#_Toc483054133)

[2.2.4 Master Login System 19](#_Toc483054134)

[2.3 Beágyazott fejlesztőkörnyezetek 20](#_Toc483054135)

[2.3.1 AT-Command 20](#_Toc483054136)

[2.3.2 NodeMCU 20](#_Toc483054137)

[2.3.3 ESP SDK 21](#_Toc483054138)

[2.3.4 Arduino 21](#_Toc483054139)

[2.3.5 Egyéb fejlesztőrendszerek 22](#_Toc483054140)

[3 Tervezés és megvalósítás 23](#_Toc483054141)

[3.1 A feladat célja 23](#_Toc483054142)

[3.2 Követelmények 24](#_Toc483054143)

[3.2.1 Funkcionális követelmények 24](#_Toc483054144)

[3.2.2 Nem funkcionális követelmények 25](#_Toc483054145)

[3.3 A rendszer topológiája 26](#_Toc483054146)

[3.4 ESP eszközök működése 27](#_Toc483054147)

[3.4.1 Felhasználói interakciók 27](#_Toc483054148)

[3.5 Hálózati hozzáférés inicializálása 27](#_Toc483054149)

[3.6 Működési módok 28](#_Toc483054150)

[3.6.1 Szenzor 28](#_Toc483054151)

[3.6.2 Beavatkozó 31](#_Toc483054167)

[3.6.3 Menedzser 32](#_Toc483054168)

[3.7 Felhasználói interakciók (use case) 33](#_Toc483054177)

[3.8 Titkosítás 33](#_Toc483054180)

[3.8.1 Eszköz 33](#_Toc483054181)

[3.9 A központi szerver 33](#_Toc483054182)

[4 A projekt állapota 36](#_Toc483054183)

[4.1 tanulságok 36](#_Toc483054184)

[4.2 Kitekintés 36](#_Toc483054185)

[5 Irodalomjegyzék 38](#_Toc483054186)

# Abstract

In the last couple of years, home automation systems appeared in more and more homes. These systems are constatntly evolving, functional and comfort upgrades are coming out year after year. On the other hand, we can see these sytems are still highly expensive, even the cheapest systems are hard to afford for the peoples.

The topic of my thesis is to plan and implement a home automation system, which is much less expensive than the business solutions, and based on preimplemented automation rules by the user, it’s also capable of interfere with its environment.

On the first part of the thesis to get better picture of the expectations and opportunities, i examine some of the business and open systems. These chapter also contains the review of the used technologies.

After that, based on the requirements i plan and implement every part of a home automation system: create the implementation of the sensor and actuator functions, and also the website, which is the graphical interface to register the devices to the system and create automation rules.

On the last part of the thesis, based on test results i review the system, and show some of further possibilities to improve the system.

# Összefoglaló

Az elmúlt néhány évben egyre több otthonban megjelentek az épület működésének automatizálását célzó megoldások. Ezek a rendszerek folyamatosan fejlődnek, évről évre megjelennek felhasználói kényelemben és funkciókban előre lépést mutató fejlesztések. Ugyanakkor az is egyértelműen látszik, hogy a gyors elterjedés egyik legfőbb akadálya a termékek magas ára, a legolcsóbb üzleti megoldások is gyakran túl drágának bizonyulnak a hétköznapi ember számára.

Diplomatervem témája egy olyan okosotthon rendszer megtervezése és elkészítése, mely az üzleti megoldásokhoz viszonyítva töredékébe kerül, és képes kihelyezett mérőeszközök segítségével a felhasználók által megadott szabályokon keresztül automatikusan beavatkozni a környezetének működésébe.

A dolgozat első részében irodalomkutatást végzek, ahol néhány üzleti és nyílt forráskódú okosotthon rendszer bemutatásán keresztül képet kapunk az alapvető elvárásokról és lehetőségekről. A fejezet második részében bemutatom a szakdolgozat elkészítéséhez használt technológiákat és könyvtárakat.

Ezt követően előzetesen megfogalmazott elvárásokat szem előtt tartva megtervezek, majd meg is valósítok egy okosotthon rendszert, annak minden részére kitérve: elkészítem a szenzor és beavatkozó eszközök implementációját, és készítek egy webes felületet, ahol a felhasználó különösebb programozói tudás nélkül, grafikus felületen készíthet egyszerű szabályokat a saját otthona automatizálásához. A dolgozat utolsó részében egy hosszabb távú tesztüzem eredményeit figyelembe véve értékelem az elkészült rendszert, valamint támpontokat adok a továbbfejleszthetőségére vonatkozóan.

# Bevezetés

## Az IoT-ről

Az IoT (Internet of Things), mint fogalom, az elmúlt tíz évben kezdett el megjelenni, amikor is a végfelhasználót tekintve az internetre csatlakozó eszközök mennyisége lassan elkezdte meghaladni az internetezők mennyiségét. Ezt úgy kell érteni, hogy míg a hagyományos értelemben vett interneten a lánc végén valamilyen valós személy generálja a tartalmat (pl. egy email üzenet), addig az IoT lényegében arról szól, hogy egy központi szerverrel kapcsolatot fenntartva, a csomópontokban automatikus módon, emberi közreműködés igénybevétele nélkül gyűjtsünk adatokat (amiket aztán továbbítunk a szervernek), vagy végezzünk beavatkozásokat a környezetünkben. A dolgok internetében az eszközeink kapcsolatban vannak egymással, kommunikációs csatornának az internetet használva.

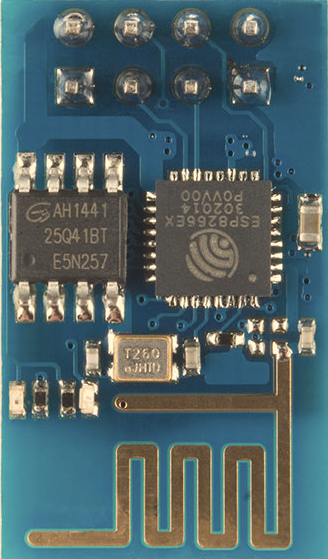
Látható, hogy a környezetünkben külön ráfordítás nélkül is már most rengeteg potenciális IoT eszköz van (okostelefon, autó, stb.), a benne rejlő lehetőségek miatt a téma jelenleg rendkívül aktuális.

## Intelligens otthon

Az intelligens otthon kifejezés a lakás automatizáltságát takarja, a fogalomra sokféle definíció létezik, hiszen rengeteg, önmagukban véve opcionális funkció tartozhat alá. Az otthonunkban megtalálható eszközök automatizáltsága már viszonylag régi jelenség, például az adott időpontban automatikusan bekapcsoló öntözőrendszer vagy az előre programozható mosógép már sok évvel ezelőtt sem voltak elképzelhetetlenek a lakásokban. Az otthon intelligens irányítása egy magasabb szintben gondolkozik, célja, hogy az önmagukban is akár bizonyos mértékben automatizált eszközeinket egy közös rendszer alá rendeljük, és ezen keresztül egységesen kezelhessük őket, hogy aztán a beállításainknak megfelelően automatikus működésre legyenek képesek. Az intelligens otthon célja, hogy a modern infokommunikációs technológiák révén az otthonban végzett hagyományos és újabb tevékenységeket kényelmesebbé, gyorsabbá és nem utolsósorban energiatakarékosabbá tegye. Példának okáért, amikor az utolsó személy is távozik a lakásból, a rendszer a beállításoknak megfelelően automatikusan lekapcsolná az égve felejtett villanyokat, bekapcsolná a riasztót és kikapcsolná vagy lejjebb venné a fűtést. Ugyanígy nem elképzelhetetlen, hogy az okostelefonos applikációnkkal (a telefon GPS[[1]](#footnote-2) helyadatai alapján akár automatikusan) munkából hazaindulás előtt jelezzük az otthonunknak, hogy készüljön fel a hazaérkezésünkre, így otthon már nem a hideg lakás fogad minket. Ha a munkából rendszeres időpontban megyünk haza, ez a működés akár automatizálható is, és ha túlóra vagy bevásárlás miatt kicsit megcsúszunk, ráérünk akkor jelezni ezt a rendszernek, ami alkalmazkodik a későbbi érkezéshez. A lakásban, ha szükség van rá (például ha egy filmet akarunk megnézni a tévénken), automatikus módon leereszthetőek a redőnyök a médialejátszó elindításával egyidejűleg. Az öntözőrendszer beállításainknak megfelelően éjszaka bekapcsol, kivéve, ha nap közben jelentősebb mennyiségű csapadék esett, amit a rendszer például saját érzékelői és internetről letöltött adatok segítségével érzékel. A hűtőszekrény automatikusan értesít, ha elfogyott a tej, és akár a beállításoknak megfelelően maga is megrendelheti a szükséges dolgokat az interneten. Látható, hogy a lehetséges funkcióknak lényegében csak a képzeletünk és az anyagi lehetőségeink szabnak határt.

## ESP8266

Az ESP8266 egy az Espressif Systems által fejlesztett és gyártott chip, mely elsősorban vezeték nélküli (WIFI) internetkapcsolat létesítésére szolgál.



1.1. ábra ESP01

Az eszköz legalapvetőbb tulajdonságai:

|  |  |
| --- | --- |
| Gyártó | Espressif Systems |
| Processzor | Tensilica Xtensa L106 (80 MHz) |
| Támogatott vezetéknélküli szabványok | 802.11 b/g/n[[2]](#footnote-3) |
| Rádiós protokoll | WIFI 2.4 GHz (WPA/WPA2[[3]](#footnote-4)) |
| Data RAM | 96 KB |
| Perifériák | GPIO(16 db), PWM, ADC, UART, I2C,SPI |
| Fogyasztás (normál üzemmódban) | ~170 mA |

1.1. táblázat Az ESP8266 főbb fizikai jellemzői

További pozitív tulajdonsága az ára: a pontos típustól függően már 1.5$ körüli áron hozzájuthatunk, ami azt tekintve, hogy ezt megelőzően egy hasonló igényeket kiszolgálni képes Wifi modul nagyságrendileg 25-40$ körül volt kapható, rendkívül kedvező. Az 1. táblázatban is felfedezhetjük az ESP8266 egy fontos jellemzőjét, miszerint saját ARM processzorral rendelkezik, ennek köszönhetően lehetővé válik a standalone működés is. A fenti tulajdonságok (saját számítókapacitás, kivezetések, kis méret, stb.) alkalmassá teszik arra a feladatra, hogy szenzorhálózat csomópontjaiban, egy központi szerverrel kommunikálva elvégezze a szükséges méréseket és a kinyert adatok továbbítását.

Megvalósítását tekintve az eddigiek során két modullal foglalkoztam. A fenti képen is megtekinthető ESP-01 a gyártó legegyszerűbb és egyben legolcsóbb panele, mely ugyan némileg korlátozza az IC képességeinek kiaknázási lehetőségeit (bizonyos lábak, pl. a GPIO lábak némelyike nincs kivezetve, stb.), de az eszközzel való megismerkedéshez, a fejlesztés megkezdéséhez jó alapot ad.

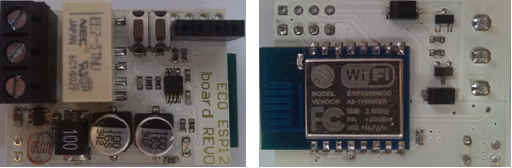
A másik megismert modul az ESP-12, az ESP8266 IC modul változata, mely lényegében az IC összes lehetőségéhez hozzáférést biztosít, így ez a változat a fejlesztési folyamatot követően az éles hálózatban is használható. Konzulensem, Györke Péter készített ehhez a modulhoz egy tesztpanelt, melyet a következő fejezetben röviden be is mutatok.

Bár kipróbálni sajnos nem volt alkalmam, de úgy gondolom, érdemes szót említeni az Espressif egy másik termékéről, az ESP32-ről is, melyet az ESP8266 nagy testvéreként is szoktak emlegetni. A felhasználási lehetőségek jellegét tekintve az ESP32 hasonló tulajdonságokkal rendelkezik, de nagyon sok téren előrelépett a korábbi IC-hez képest. Teljesítmény tekintetében messze túlszárnyalja az ESP8266 képességeit: megduplázott, 160Mhz-es órajellel rendelkezik, a felhasználható RAM 520KB-ra nőtt, ami többszörös növekedés, valamint például a WiFi mellett már a Bluetooth[[4]](#footnote-5) vezeték nélküli szabványt is támogatja. Sajnos a fejlesztés kezdetén még az eszköz nem állt rendelkezésre, és bár a nagyobb teljesítménnyel megnövekedett, 5$ körüli ár is párosult, a tervezés fejezetben láthatjuk majd, hogy lesz a rendszernek olyan pontja, ahol ez ESP eszköz egy helyi központi egység feladatát végzi el, a rendszernek ezen a pontján, ha lett volna rá lehetőség, mindenképp érdemes lett volna megfontolni az ESP32 használatát.

## ESP-12 tesztpanel

A tesztpanel célja, hogy olyan általános célú kiegészítéseket adjon a modul mellé, amely a fejlesztés és tesztelés során a legtöbb felmerülhető funkció használatára lehetőséget ad.

A fejlesztés során szükséges kommunikációs csatornát az ESP8266 és a fejlesztés platformja között USB-UART átalakító biztosítja, ennél fogva adja magát, hogy a tápfeszültséget is innen nyerjük. Ugyanakkor, míg a szabványos USB 5V-ot szolgáltat, az IC-nek a működéshez 3.3V-ra van szüksége. AZ ESP-01 esetén az USB-UART átalakító végezte a tápfeszültség transzformálását is, a tesztpanelnél már erre nincs szükség erre, mivel az már saját maga elvégzi ezt.



1.2. ábra Tesztpanel

Kivezetések tekintetében így értelemszerűen a táp, föld, illetve a kommunikációhoz szükséges UART[[5]](#footnote-6) (RXD és TXD) lábakhoz tudunk csatlakozni, valamint a GPIO0 lábhoz, valamint a GPIO0 lábhoz, melynek állapota induláskor meghatározza, hogy a modul normál vagy boot üzemmódban induljon el. A panelen a megfelelő interfészek teszteléséhez egy I2C[[6]](#footnote-7) hőmérsékletmérő és egy analóg fényerősségmérő szenzor is el lett helyezve. Helyet kapott ezen kívül még néhány további kiegészítés is, mint például nyomógombok, egy bistabil relé[[7]](#footnote-8) és néhány LED[[8]](#footnote-9) is.

# Irodalomkutatás

A fejezet első felében bemutatok néhány piaci és nyílt forráskódú megvalósítást az otthonautomatizálási rendszerek témakörében. Ezt követően röviden ismertetem a saját rendszerem elkészítésénél használt technológiákat és külső könyvtárakat.

## Otthonautomatizálási rendszerek

Otthonautomatizálási rendszerek tekintetében a piacon mára számos üzleti megoldás jelent meg. Ezen rendszerek nagy része csak a gyártó saját eszközeivel működik együtt, de láthatjuk majd, hogy ez alól akadnak kivételek. Léteznek moduláris megoldások is, ezek nemcsak hogy már meglévő házba telepíthetőek, hanem a rendszerbe bármikor könnyen becsatlakoztathatunk egy új eszközt is. Az alfejezetben áttekintünk két nyílt forráskódú megoldást is, a rendszerek tekintetében az alapvető működésükre, funkcionalitására és az alkalmazott technológiákra koncentrálva.

### IContrall

A magyar érdekeltségű IContrall az otthon teljes automatizálását egy az egyben a saját eszközeivel kívánja megoldani. A gyártó a legtöbb háztartási eszköz számára - mint például a riasztó vagy az árnyékoló - speciális vezérlőket kínál, de ezen felül vannak univerzális vezérlő moduljaik is. Az, hogy az IContrall külső eszközöket nem enged be a rendszerébe, az egyértelmű hátrány mellett előnyökkel is szolgál: a kommunikációs csatorna titkos volta miatt a rendszer kevésbé sebezhető külső támadásokkal szemben. A modulok telepíthetőek már meglévő lakásba is: a kommunikáció a vezeték nélküli hálózaton keresztül történik, ez lehetővé teszi az új eszközök gyors csatlakoztatását a már meglévő rendszerhez .

A felhasználói vezérlésre okostelefonos és táblagépes felület szolgál, a gyártó ígérete szerint ez könnyen kezelhető, logikus szerkezetű, így az új felhasználók is hamar beletanulhatnak a rendszer üzemeltetésébe.

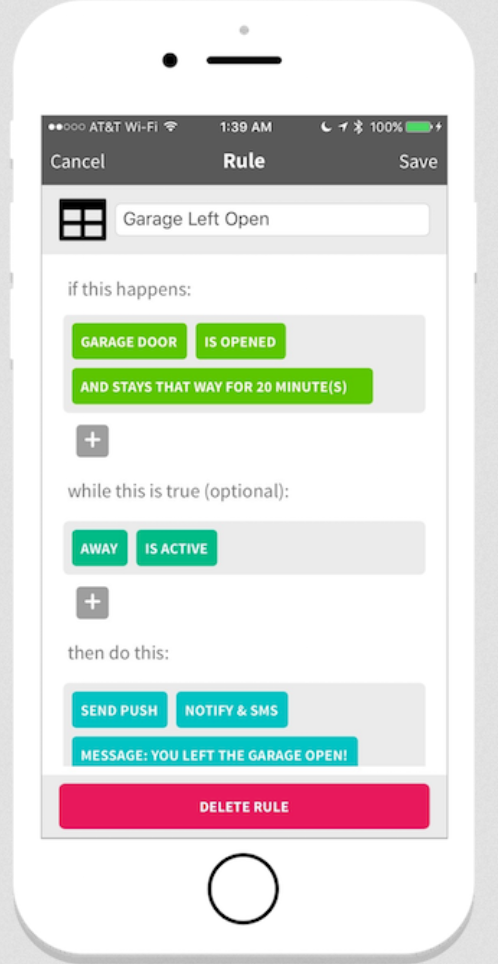


2.1. ábra Az IContrall felhasználói interface tableten

### SmartThings

A Samsung által 2014-ben felvásárolt SmartThing vállalat üzleti stratégiája egy nyílt rendszerre alapoz, melyhez bármely gyártó készíthet kompatibilis eszközöket. A már meglévő, népszerű okoseszközök zömével boldoguló rendszer alapja a Samsung SmartThings HUB, mely otthonunk vezeték nélküli hálózatára csatlakozva rögtön az okosotthon középpontjává válhat.

A rendszer irányítására elsődlegesen a mindhárom okostelefonos operációs rendszerre ingyenesen elérhető SmartThings Mobile alkalmazás szolgál, ezen felül több más eszköz is létezik. Amiben pedig kitűnik a rendszer, az a vezérlési szabályok megalkotása: véleményem szerint a vizsgált, és a dolgozatban nem részletezett, de a kutatás során általam megismert rendszerek közül a SmartThingsnek van a legjobban kitalált és felhasználói szempontból legkönnyebben kezelhető szabályalkotási alapelve. Ehhez egyszerűen annyit tesz, hogy az egész szabályalkotás sokkal kevésbé absztrakttá teszi: az automatizálási szabályok alapján rengeteg mindennapi esemény történésként előre van definiálva, így a felhasználónak elég ezek alapján megfogalmaznia a szabályokat, szinte úgy, mint ha egy mondatot akarnának elmondani arról, pontosan mit szeretnének. Példának okáért a SmartRules, sajnos csak iOS-re elérhető appon definiált szabályt nézhetjük meg a 3. ábrán, ami jól olvasható, és nagyon egyszerűen leír egy viszonylag komplex beavatkozást.



2.2. ábra SmartRules szabályalkotás

### OpenHAB

Az OpenHAB (Open Home Automation Bus) jelenleg a legnépszerűbb nyílt forráskódú megoldás az otthonautomatizálás témakörében, a 2017 januárjában megjelent OpenHAB 2 több mint 130 gyártó eszközéhez vagy technológiához biztosít integrálhatóságot. A Java nyelven írt főrendszerhez – ami az OpenHAB Runtime nevet viseli - számos kliensen keresztül lehet kapcsolódni. A konfigurálásra elsősorban az Eclipse SmartHome Designert ajánlja a fejlesztő – ez az Eclipse IDE egy speciális, OpenHAB-hoz készült verziója - , melynek különlegessége leginkább abban rejlik, hogy olyan fejlesztőkörnyezetektől elvárt támogatásokat nyújt a szabályleíró nyelvhez, mint a kódszinezés vagy a kód szintaktikai ellenőrzése.

A szabályleíró nyelv az Xtend[[9]](#footnote-10) szintaxisát követi, egy szabály az alábbi formában írható le:

rule "rule name"

when

<TRIGGER CONDITION1> or

<TRIGGER\_CONDITION2>

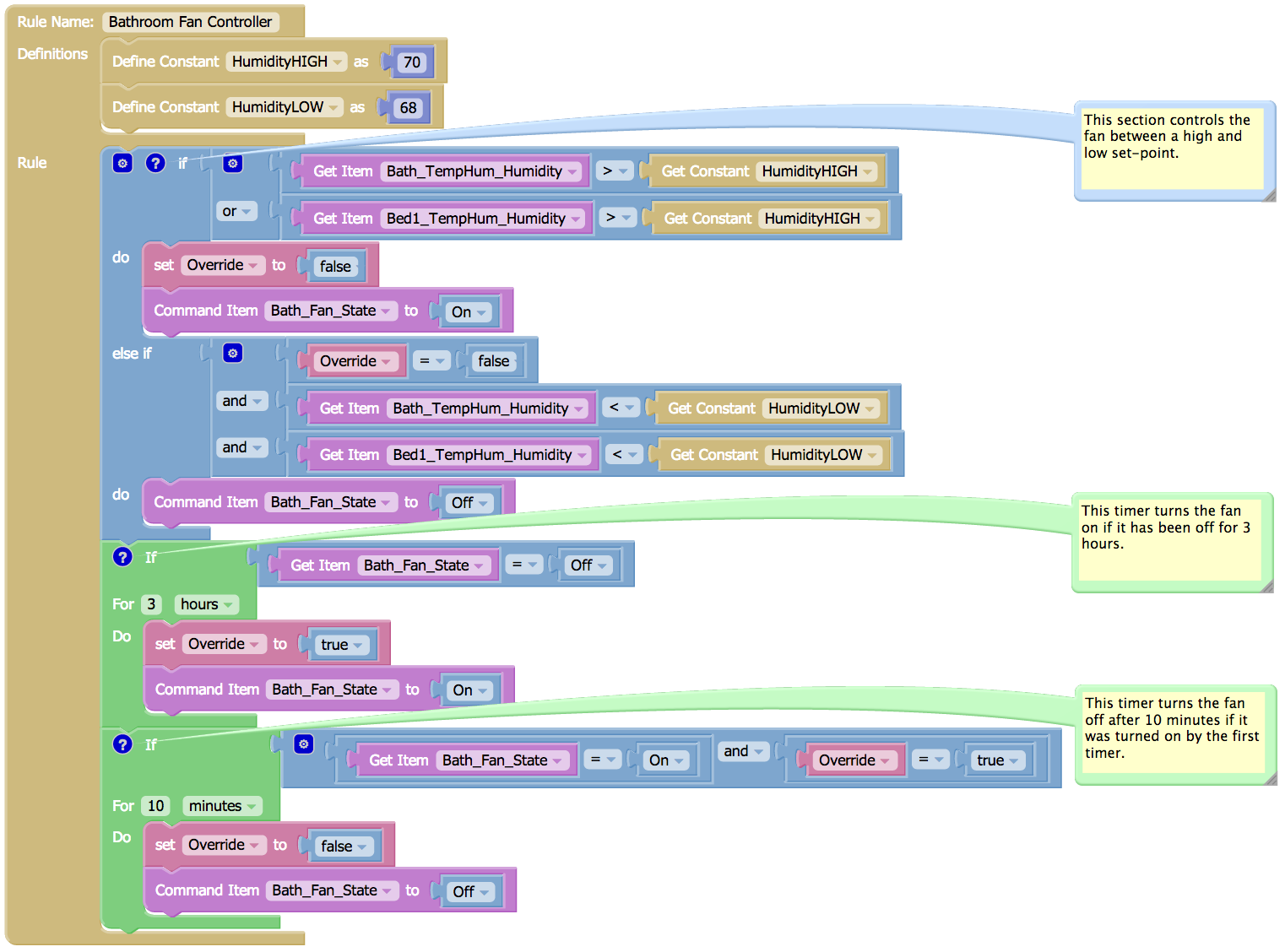
...

then

<SCRIPT\_BLOCK>

end

A számos addon a felhasználók igényeit és kényelmét szolgálja, a szabályleírásnál maradva például a 4. ábrán megtekinthető egy grafikus szabálytervező, ahol a szerkesztett szabályok szintén a fenti struktúrának és szintaxisnak megfelelő formába fordítódnak le, de a drag and drop jellegű folyamat mégis könnyebbséget okozhat a programozói tudással csak alap szinten rendelkezők számára.



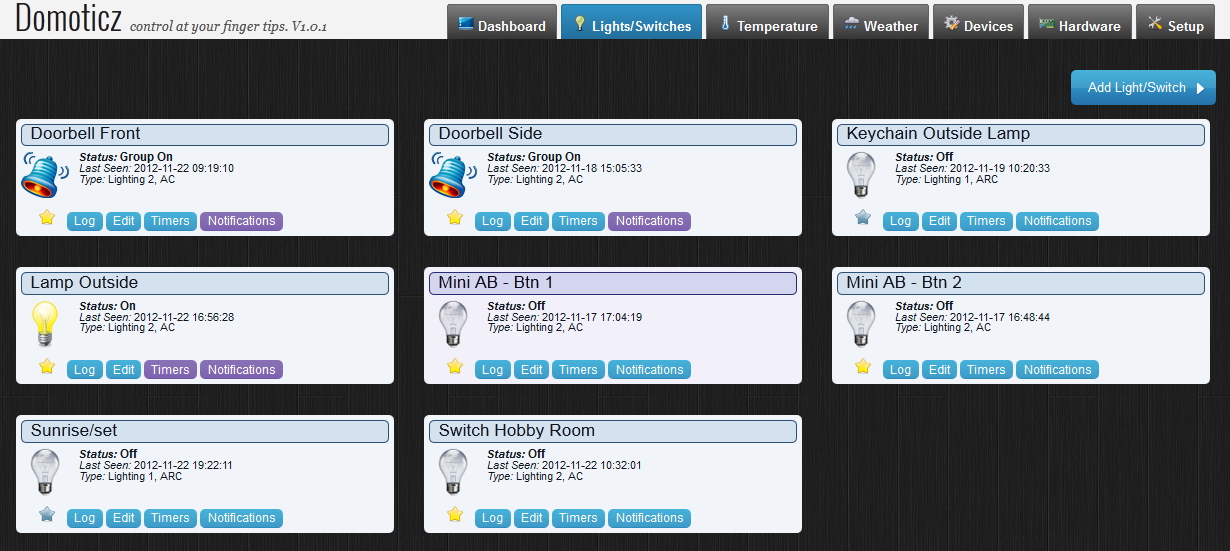
2.3. ábra Grafikus szabályleíró rendszer az OpenHABon

A szabályleírásokra szolgáló felületen túl a fejlesztő a rendszer mindennapi üzemeltetéséhez alkalmazások széles választékát kínálja, többféle webes kliensen felül a három nagy mobil operációs rendszerhez (Android, iOS, Windows Phone) is készített alkalmazást, különösen az utóbbi a rendszer elterjedtsége okán még a nagyobb piaci szereplőknél is gyakran kimarad.

### Domoticz

Az open source megvalósítású, C++ nyelven írt Domoticz többféle platformon is elérhető: telepíthető a három nagy (Windows, Linux, MacOS) asztali operációs rendszerre, de minden további nélkül futtatható például egy Raspberry PI-ról vagy a Synology gyártó NAS[[10]](#footnote-11) készülékeiről is. Az első, 2012-ben kiadott verzió óta folyamatosan fejlesztik, mind a hozzáadható eszközök tekintetében, mind az elérhető funkciók tekintetében. A felhasználók egy HTML5 webes interfészen keresztül férnek hozzá a rendszerhez, mely a saját, beágyazott webszerverén fut. A viselkedés beállítására többféle megoldást kínál a Domoticz: többek között LUA szkript vagy Blocky[[11]](#footnote-12) használatával is automatizálhatjuk az eseményekre történő reakciókat.

Az integrálható eszközök választéka széles, a nagy gyártók megoldásai (pl. Z-Wave eszközök, Phillips Hue) mellett számos más eszköz használata is lehetséges. A rendszer felhasználói értesítéseket is képes küldeni mobil eszközeinkre, legyenek azok Android, iOS vagy Windows Phone rendszerűek.



2.4. ábra A Domoticz webes konfigurációs felülete

### Amazon Echo

Némileg kilóg a sorból az Amazon Echo, hiszen nem, vagy legalábbis nem kizárólag a hagyományos értelemben vett otthonautomatizálási rendszer. Ugyanakkor technológiailag a Google Home-al versenyezve véleményem szerint azt az irányt mutatja, amit a jövő (vagy lassan már jelen, a 2.1.2 fejezetben bemutatott OpenHAB rendszernek például már van az Echohoz interfésze) okosotthon rendszerei követni fognak, így mindenképp érdemes röviden áttekinteni.



2.5. ábra Amazon Echo Dot és Amazon Echo

Az Amazon Echo legfőbb különlegessége a hangvezérlésben rejlik: az eszköz mai viszonylatban fejlett beszédfelismerő rendszere nem csak az összekötött okos eszközeink vezérlését teszi lehetővé, hanem készségesen válaszol az időjárásra vonatkozó kérdésünkre, sőt, egyszerűbb matematikai problémákat is meg tud oldani nekünk. Ehhez pedig csak kérdeznünk kell. Egyelőre leginkább az angol nyelvet támogatja az eszköz, a gyártó ígérete szerint a kínálat a későbbiekben jelentősen bővülni fog.

Ár tekintetében abszolút versenyképesnek érzem az Amazon Echo-t: a gyártó oldaláról rendelve 150$-ért már miénk lehet, de aki megelégszik a kisebb változattal, már 50$-ért beszerezheti. Az Amazon Echo önmagában még nem alkot egy okos otthon rendszert, de a rendszer magjaként képes működni, ha további eszközöket csatolunk hozzá.

## Alkalmazott technológiák és keretrendszerek

A fejezetben egyenként ismertetem a dolgozatban felhasznált technológiákat, terjedelmi okokból elsősorban a lényegesebb, szakmai döntést igénylőkre kitérve. A fejlesztés során az ESP8266 üzleti logikájának megírása során a saját döntésemből kifolyólag többszöri API váltás történt, indokolt így ezekről a platformokról is rövid áttekintést adni, kiemelve a tapasztalt előnyöket és hátrányokat. A szerver oldal teljes mértékben a LAMP szoftver csomagként is ismert megoldásgyűjteményre épít, mely betűszó a mögötte rejlő technológiákat takarja: (esetünkben Debian) Linux operációs rendszerre telepített Apache HTTP szerver, melyhez tartozik egy MySQL adatbázis, az oldalak megvalósítása pedig elsődlegesen PHP szkriptnyelven történik. Az alfejezet végén további technológiákról esik szó, mint a platformok közötti kommunikáció megkönnyítésére használt JSON formátum, a szenzoradatok grafikus megjelenítésére szolgáló HighStock API, illetve a felhasználó autentiákációs keretrendszerként használt Master Login System.

### JSON

A JSON (JavaScript Object Notation) egy szöveges adatcsere-formátum, mely eredetileg a JavaScript programozási nyelv részeként jelent meg 1999-ben. Mára a nyelvtől lényegében függetlenné vált, szinte minden jelentős programozási nyelvhez készült JSON generáló és feldolgozó programcsomag. A JSON-t számos helyen alkalmazzák az XML alternatívájaként, mivel amellett, hogy egyes feladatokra az ember számára sokkal áttekinthetőbb struktúrát kínál, egyszerűbb is, ami gyorsabb feldolgozást tesz lehetővé. Létezik sémaleíró nyelve is, amivel a JSON struktúrák egyszerűen validálhatóak.

A JSON gyakorlatilag két féle struktúrára épít:

* Név-érték párok gyűjteménye. Ez nyelvtől függően reprezentálhat például egy objektumot, struktúrát, hash táblát vagy asszociatív tömböt[[12]](#footnote-13)
* Értékek listájaként reprezentálja például a tömböt, vektort, listát

Példaként megnézhetünk a dolgozatban használt, egy szabályt leíró struktúrát:

**{**

"user\_id"**:** 1**,**

"name"**:** "lampa1"**,**

"description"**:** "Ez a szabály felkapcsolja a ledet, ha a fényerősség két érték közé kerül egy szenzoron"**,**

"triggers"**:** **[**

**{**

"id"**:** 1**,**

"rule\_id"**:** 1**,**

"sensor\_id"**:** 1**,**

"operation"**:** "<"**,**

"value"**:** "300"

**},**

**{**

"id"**:** 2**,**

"rule\_id"**:** 1**,**

"sensor\_id"**:** 1**,**

"operation"**:** ">"**,**

"value"**:** "200"

**}**

**],**

"actions"**:** **[**

**{**

"action\_id"**:** 1**,**

"actuatorid"**:** 1**,**

"rule\_id"**:** 1**,**

"value"**:** "1"

**}**

**]**

**}**

A hozzá tartozó séma pedig:

**{**

"title"**:** "Rule"**,**

"type"**:** "object"**,**

"properties"**:** **{**

"user\_id"**:** **{**

"type"**:** "integer"**,**

"title"**:** "User id"

**},**

"name"**:** **{**

"type"**:** "string"**,**

"title"**:** "Name"

**},**

"id"**:** **{**

"title"**:** "Id"**,**

"type"**:** "integer"

**},**

"description"**:** **{**

"title"**:** "Description"**,**

"type"**:** "string"

**},**

"triggers"**:** **{**

"type"**:** "array"**,**

"format"**:** "table"**,**

"title"**:** "Triggers"**,**

"uniqueItems"**:** **true,**

"items"**:** **{**

"type"**:** "object"**,**

"title"**:** "Trigger"**,**

"properties"**:** **{**

"id"**:** **{**

"type"**:** "integer"

**},**

"rule\_id"**:** **{**

"type"**:** "integer"

**},**

"sensor\_id"**:** **{**

"type"**:** "integer"

**},**

"operation"**:** **{**

"type"**:** "string"

**},**

"value"**:** **{**

"type"**:** "string"

**}**

**}**

**}**

**},**

"actions"**:** **{**

"type"**:** "array"**,**

"format"**:** "table"**,**

"title"**:** "Actions"**,**

"uniqueItems"**:** **true,**

"items"**:** **{**

"type"**:** "object"**,**

"title"**:** "Action"**,**

"properties"**:** **{**

"action\_id"**:** **{**

"type"**:** "integer"

**},**

"actuatorid"**:** **{**

"type"**:** "integer"

**},**

"rule\_id"**:** **{**

"type"**:** "integer"

**},**

"value"**:** **{**

"type"**:** "string"

**}**

**}**

**}**

**}**

**}**

**}**

### HighStock

A HighCharts egy tisztán JavaScript nyelven írt könyvtár, melynek segítségével webes környezetben készíthetünk grafikonokat.

A programcsomag része az eredetileg tőzsdei adatsorok megjelenítésére készült HighStock, de rugalmasságánál fogva szenzoradatok megjelenítésére is alkalmas. Az oldalon részletes API, számos tutorial és demo diagram segíti a megértést és a könnyű integrálhatóságot. Az eredetileg meglehetősen drága API-nak nem kereskedelmi célú felhasználásra létezik ingyenes licensze is, ami lehetővé tette a dolgozatban történő felhasználást is.



2.6. ábra A Highstock dolgozatban is használt grafikonja

### Bootstrap

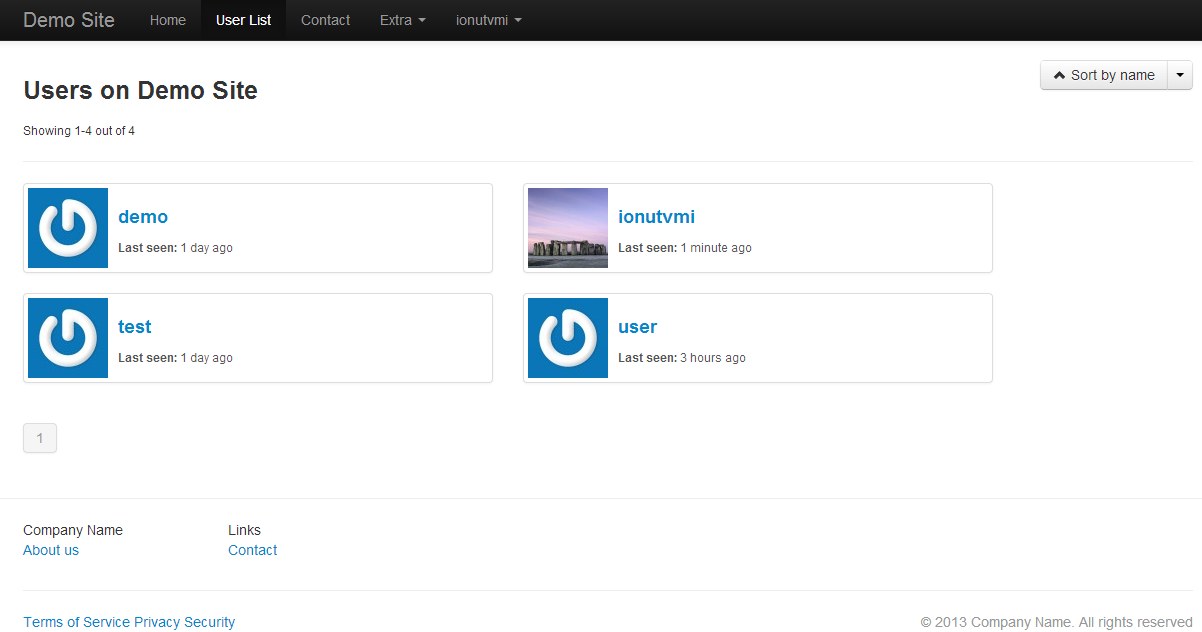
A Bootstrap manapság a világon legtöbbet használt CSS[[13]](#footnote-14) keretrendszer, melynek első verzióját a Twitter fejlesztői adták ki 2011-ben, de mára a reszponzív oldalfejlesztés egyik legfőbb eszközévé vált. Az open source megvalósítás, illetve hogy 2017-ben már a negyedik fő verzió megjelenésére számíthatunk, együttesen járulnak hozzá az egyre növekvő népszerűségéhez. Rendkívül sokféle, az oldalba beépíthető elemet tartalmaz, kezdve az egyszerű szövegformázási lehetőségektől egészen a táblázatokig vagy a tartalom oldalon belüli fülekbe szervezéséig.

A Bootstrap használata egyszerű, segítségével minimális erőfeszítéssel készíthetünk reszponzív oldalakat. Ez kiemelt jelentőséggel bír, hiszen abban a korban, amikor a weboldalakat az okostelefonok kijelzőjétől kezdve a PC monitorokig egyaránt kényelmesen szeretnénk használni, erre már nem plusz szolgáltatásként, hanem valós elvárásként tekinthetünk.

### Master Login System

A Master Login System egy bootstrap alapú user management keretrendszer, mely a szerver oldal alapjául szolgált. A php nyelven íródott rendszer a felhasználók kezelésére biztosít felületet, az ilyen rendszerektől elvárt alapvető igényeket kiszolgálva. A rendszer funkciói:

* Felhasználói alapműveletek (be -és kijelentkezés, regisztrálás, profil szerkesztése)
* Jogosultságkezelés (csoport alapú, admin, user, custom felhasználói csoportok)
* Admin panel (userek listázása, admin funkciók)
* Kapcsolattartási űrlapok



2.7. ábra Master Login System

## Beágyazott fejlesztőkörnyezetek

Az ESP8266 megjelenése (2014 második fele) óta komoly népszerűségre tett szert, így érdemes időt szánni arra, hogy megvizsgáljuk a fejlesztés platformja tekintetében a fejlesztők számára milyen alternatív fejlesztőrendszereket képes nyújtani. Ennek már csak azért is jelentősége van, mert a diplomaterv készítése során magam is többször váltottam platformot.

### AT-Command

Az IC-n gyárilag is feltöltött AT command nyelvet használó firmware használata indokolt lehet, ha az eszközt csak a wifi hálózathoz csatlakozáshoz miatt használjuk, de komplexebb alkalmazásokhoz (pl. szenzorhálózatban történő felhasználásnál) érdemes valami magasabb szintű nyelvet keresni a fejlesztéshez.

### NodeMCU

A fejlesztés kezdetén használt firmware a NodeMCU nevet viseli, mely egy C nyelven írt Lua interpreter. A választás mellett elsődlegesen az szólt, hogy az ESP-t használó közösség körében a korai szakaszban az egyik legnépszerűbb platform volt az ESP8266-on belül, így biztosítva a jól dokumentáltságot, a sok példakódot, valamint azt is, hogy a firmware-t a fejlesztők folyamatosan fejlesztik és karban is tartják. A NodeMCU kezdeti lendülete mára kissé erejét vesztette, de továbbra is a nagyobb támogatottságú környezetek közé tartozik. Egyszerűbb üzleti logikájú működés megvalósítására minden további nélkül ajánlott a használata. A rendszer fejlesztése során azonban nálam stabilitási gondok léptek fel: a NodeMCU-nál sokszor okozott gondot a memóriakorlátok betartása, egy egyszerű webes felület létrehozása is már komoly megszorításokat igényelt, ezek sokszor valószínűleg a firmware apróbb hibáiból eredő pazarló memóriahasználatra voltak visszavezethetőek.

### ESP SDK

A szakdolgozat készítésének kezdetén a firmware váltás mellett döntöttem, az egyik kipróbált új rendszer az ESP8266-ot gyártó Espressif által fejlesztett, így lényegében hivatalosnak tekinthető SDK lett. Ennek két alapvető változata az OS nélküli, és az RTOS[[14]](#footnote-15) alapú megvalósítás, a fő különbség, hogy míg előbbiben az eseménykezelés elsősorban timer és call-back hívások segítségével valósul meg, utóbbi a FreeRTOS[[15]](#footnote-16) ütemezőjére épít. Én a feladat jellegéhez való jobb illeszkedés miatt a NONOS verziót használtam. A platform mellett egyértelmű előny volt a C nyelv, magamhoz közelebbinek éreztem és felhasználható függvénykönyvtárak széles választékának köszönhetően több funkciót is jelentősen kisebb munkával sikerült újraimplementálnom. Sajnos a működés implementálása során többször rá kellett jönnöm, hogy a kiadott SDK bár folyamatos fejlesztés alatt van, sajnos még mindig sok hibát tartalmaz, ami nehézkessé tette az eszközben rejlő lehetőségek kiaknázását. Például bár hivatalosan az API tartalmazta az SSDP[[16]](#footnote-17) implementációját, a valóságban ezt nem sikerült életre keltenem, az interneten az SDK fórumjait böngészve pedig azt tapasztaltam, hogy ez egy javításra váró hiba.

### Arduino

Az Arduino egy az Atmel AVR mikrovezérlő családra épülő, szintén C nyelvű szabad szoftveres elektronikai fejlesztőplatform, mely alapvetően a gyártó saját eszközeihez készült és „easy to use” szemlélete révén hamar elterjedt főleg a hobbi felhasználók körében. Ez a folyamat azzal járt, hogy hamarosan más több más eszközre is megjelent a platform implementációja, a kereskedelmi forgalomban való megjelenését követően erre az ESP8266 esetében sem kellett sokat várni. Előnyei közé sorolható az egyszerű használhatósága, széles körű elterjedtsége, a funkciók jól dokumentáltsága, rengeteg online példakód. Ezen felül az ESP SDK-val tapasztaltakkal szemben az Arduino esetében azt tapasztaltam, hogy a leírt funkciók valóban működnek is, használat közben sokkal megbízhatóbbnak éreztem, így az ESP modulokon végzett fejlesztést teljes egészében erre a platformra építettem fel.

### Egyéb fejlesztőrendszerek

Fentieken felül természetesen léteznek más alternatívák is, például a Python fejlesztési felületet nyújtó MicroPython keretrendszernek is van implementációja ESP8266-ra, de ezekkel nem foglalkoztam részletesebben a dolgozat elkészítése során.

# Tervezés és megvalósítás

Az előző fejezetben az üzleti és nyílt forráskódú megoldások megismerése során egyaránt azt láttam, hogy az újabb rendszerek már szinte kivétel nélkül előnyben részesítik a vezeték nélküli megoldásokat, hiszen ez biztosítja a könnyebb beépíthetőséget (már meglévő lakásba is), valamint a rendszer elemeinek cserélhetőségét és bővíthetőségét is. Kritikus tényező azonban a rendszerek ára, mely sok potenciális felhasználó számára jelenleg elérhetetlenné teszi az okosotthon rendszerek használatát. A tanulságok figyelembevételével ez a fejezet a korábban ismertetett technológiákra alapozva egy teljes okosotthon rendszer tervezési és elkészítési folyamatát mutatja meg.

## A feladat célja

A feladatban egy otthonautomatizálási projekt részeként szeretnék a kommunikáció támogatására létrehozni egy hálózatot, melynek csomópontjaiban az internetre kapcsolódni képes eszközök egy központi szerverrel kommunikálnának.

Az eszközök két funkciót valósítanak meg. Egyrészt képesnek kell lennie arra, hogy a környezetükben található szenzoroktól előre meghatározott rendszerességgel adatokat gyűjtsenek és továbbítsák a szerver számára, amely aztán képes feldolgozni és megjeleníteni ezeket. Az eszközök másik felelőssége, hogy a szerverrel kommunikációs csatornát legyenek képesek felépíteni, és a központtól kapott utasításokat fogadva beavatkozzanak a környezetükben lévő eszközök működésébe.

Ugyancsak megvalósítandó feladat a központi szerver implementációja. Ennek több, a feladat szempontjából lényeges funkciója is van. Egyrészt képesnek kell lennie a szenzorok számára lehetőséget biztosítani a méréseik feltöltésére. Ezeket természetesen a rendszer más nélkülözhetetlen adataival együtt tárolni is szeretnénk, így szükséges, hogy valamilyen adatbázisunk is legyen. A központi szerveren szükséges egy webes felület kialakítása, ahol a felhasználók elvégezhetik a rendszer konfigurálását. Természetesen egy weboldal tetszőleges számú otthont kiszolgálhat, a felhasználói jogosultságkezelésnek köszönhetően az egyes rendszerek kezelőfelülete el van választva egymástól, a felhasználók csak a saját rendszereikhez kapnak hozzáférést.

Már a tervezési szakaszban érdemes foglalkozni az esetlegesen felmerülő problémákkal, amelyek a rendszer működését befolyásolják. Az interneten keresztül adatokat továbbító eszközök esetében például fontos kérdés az, hogy mi történik akkor, ha az internetkapcsolatunk valamilyen oknál fogva rövidebb, vagy akár hosszabb időre megszakad. Egy egyszerű architektúrában, ahol a szenzorok és beavatkozók a központi szerverrel kapcsolatot tartva csak üzeneteket küldenek és fogadnak, ez a probléma megszűnéséig a rendszer teljes időleges működésképtelenségét okozná, hiszen egyrészt a szenzoradatok nem érkeznek meg a szerverhez, másrészt a beavatkozónak küldött utasítások sem érnének célba. A probléma áthidalására érdemes olyan megoldást találni, ahol az internetkapcsolat megszakadása esetén is képesek a lakáson belüli eszközök működni, legfeljebb időlegesen a szerverről történő konfigurálási lehetőségeinket veszítjük el.

## Követelmények

A 3.1 fejezetben leírtakat követően fontos lépés a rendszerre vonatkozó legfontosabb elvárások áttekinthető pontokba szedett megfogalmazása, hiszen a későbbiekben ezen követelmények alapján lesz ellenőrizhető, hogy a rendszer teljesíti-e az előzetes elvárásokat.

### Funkcionális követelmények

A teljes rendszerrel szemben támasztott legfontosabb követelmények:

1. Az okosotthon rendszerben szerepelnie kell
   1. egy központi szervernek
   2. szenzor eszközöknek, melyek képesek a környezetük érzékelésére, és a mért eredményeiket képesek továbbítani a megfelelő irányba
   3. beavatkozó eszközöknek, melyek kapott utasításoknak megfelelően képesek a környezetükbe történő beavatkozásra
2. A központi szerver feladata
   1. fő adatbázis biztosítása
   2. felhasználói felület biztosítása
      1. alapvető user adminisztrációs feladatok
      2. eszköz kezelési feladatok (eszközök, szenzorok, beavatkozók felvétele a rendszerbe, vezérlési szabályok megalkotására)
      3. szenzorhoz tartozó mérési adatok megtekinthetősége grafikon formájában
   3. Forgalom kezelése
      1. bejövő szenzor mérési eredmények letárolása
      2. HTTP kérésekre válaszként adatokat továbbítani a kérelmező számára (például egy adott területhez tartozó szabályadatbázis)
   4. egy szerver több otthont is képes kell legyen kiszolgálni
   5. jogosultságkezelés, adott felhasználó csak a saját rendszerébe tartozó eszközökhöz kap szerkesztési és megtekinthetőségi jogokat
3. A szenzor feladata
   1. a beállításoknak megfelelően mérési eredmények előállítása és továbbküldése a közös hálózaton
4. A beavatkozó feladata
   1. a közös hálózaton keresztül kapott utasításokra való reagálás, a környezetbe való beavatkozás (például egy relé átkapcsolása)

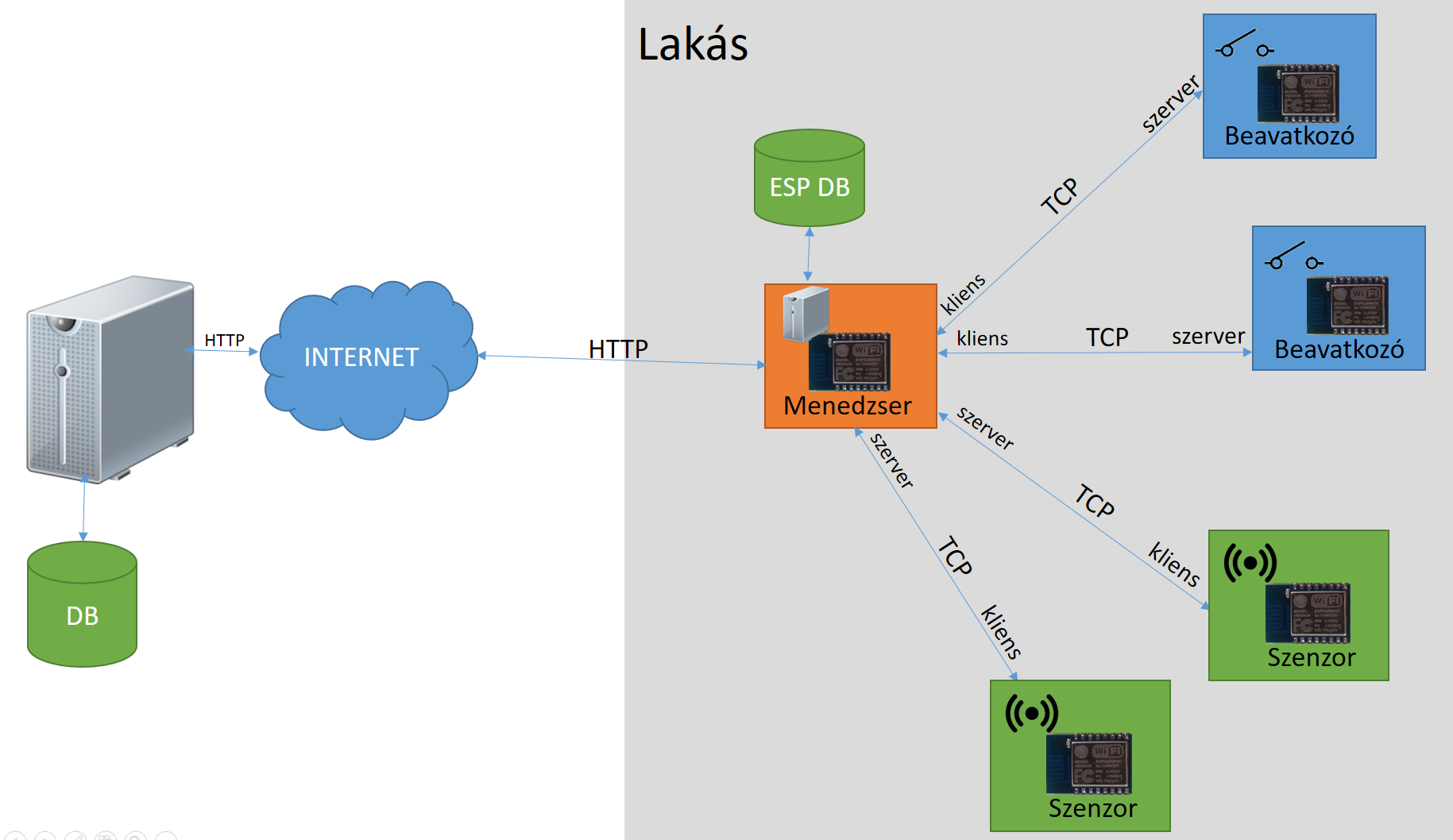
### Nem funkcionális követelmények

A funkcionális követelmények megfogalmazása mellett néhány nem funkcionális elvárást is támasztottam a készítendő rendszerrel szemben:

1. lehetőségekhez mérten a webes felület legyen felhasználóbarát, a használatához ne legyen szükség mély informatikai ismeretekre vagy előzetes betanulásra
2. a rendszer a piaci rendszerekhez képest legyen olcsón előállítható
3. az internetes kapcsolat megszakadása esetén az otthoni rendszer ne bénuljon teljesen meg, a felhasználói konfigurálást és a mérési eredmények valós idejű megjelenítését, tehát a szerver oldali beavatkozási lehetőségeket kivéve a működés ilyen esetben is legyen folyamatos
4. lehetőség szerint törekedni kell a rendszer alacsony energiafogyasztására

## A rendszer topológiája

Az ismertetett követelmények figyelembe vételével a tervezés első lépéseként ismertetem, hogy milyen egységeket tervezek megvalósítani, és ezek hogyan kapcsolódnak egymáshoz.

Ennek áttekintéséhez a 8. ábra nyújt segítséget:

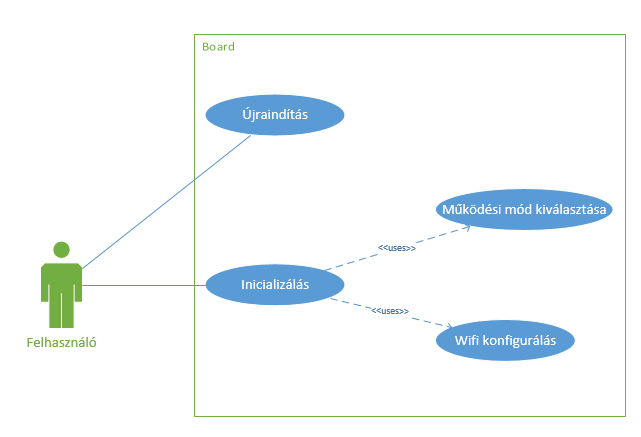
3.1. ábra A rendszer topológiája

A központi szerver az ábra bal oldalán látható. A szürke négyzet egy, fizikailag és logikailag is egy helyen lévő egységeket takar, nevezzük lakásnak. A lakás alapvetően három féle elemet tartalmaz, a már ismert szenzorokat és beavatkozókat, illetve kötelező elemként mindenhova tartozik egy helyi menedzser is. Az természetesen nincs kizárva, hogy utóbbinak fizikailag valamelyik szenzor vagy beavatkozó eszköz lássa el a feladatát, ahogy egy modul is lehet beavatkozó és szenzor is egyszerre, de a logikai szétválasztás miatt érdemes külön is gondolni rájuk. A központi szerverrel a helyi menedzserünk kommunikál, a többi modul rajta keresztül tartja a kapcsolatot, azaz a szenzorok ide küldik az adataikat, a beavatkozókat pedig közvetlen módon a lokális menedzser utasítja, a saját központi szerverről frissített adatbázisa alapján. A kommunikáció módja két IC (amiből az egyik a menedzser) között TCP kapcsolat segítségével valósul meg, míg a menedzser és szerver között csak a szerver által biztosított HTTP lekérdezeseken keresztül történhet a kommunikáció. Ez egyúttal azt is jelenti, hogy a szerver nem tud közvetlen módon kommunikációt kezdeményezni, de a rendszer megtervezése során ezzel a megkötéssel számoltam.

## ESP eszközök működése

### Felhasználói interakciók

A rendszer tervezése során figyelemmel kell lenni arra, hogy a legfeljebb átlagos informatikai ismeretekkel rendelkező felhasználók számára is minél könnyebben kezelhető legyen. Ennek jegyében a felhasználó rendszerbe való beavatkozási lehetőségeit az ESP boardon igyekeztem minimalizálni. A 8. ábrán láthatjuk, hogy a felhasználók feladata lényegében kimerül egy egyszeri inicializálásban, amit majd egy eszköz által szolgáltatott webes felületen tehetnek meg. Univerzális eszközök esetén a felhasználó ugyanezen a felületen a működési módot is kiválaszthatja, de ez a lépés előre meghatározott feladattal rendelkező ESP-k használata esetén kihagyható. A boardon található gomb segítségével probléma esetén bármikor újra lehet indítani, áramkimaradás (vagy lemerült elem) esetén a probléma megoldását követően ez automatikusan megtörténik. Minden más műveletet a felhasználó a szerver később bemutatott webes felületén tud végrehajtani.



3.2. ábra Eszköz oldali felhasználói use casek

## Hálózati hozzáférés inicializálása

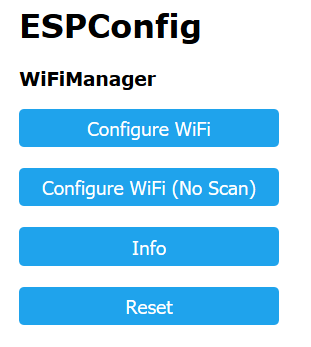
Bármilyen ESP eszköz esetén a kommunikáció egyetlen csatornája a vezeték nélküli hálózat, így az eszköz indulását követően első feladatként szükséges, hogy a modul felcsatlakozzon valamilyen előre meghatározott vezeték nélküli hálózatra, ez például lehet a lakás otthoni WiFi hálózata.

Ennek jegyében az eszköz első indításának úgy kell megtörténnie, hogy a felhasználó megadhassa a WiFi-re történő csatlakozásához szükséges SSID és password adatokat. Ennek biztosítására az első alkalommal Acces Pointként induló eszköz által szolgáltatott weboldal fog szolgálni. Ha ez megtörtént, a modul automatikusan station módba vált, majd a saját működési módjának megfelelően már bármikor tud csatlakozni a hálózathoz, megnyitva az utat a többi eszközzel és a szerverrel történő kommunikációhoz.

Természetesen sokféle okból előfordulhat, hogy a megadott vezeték nélküli csatlakozási adatokat módosítani kell, így a felhasználó számára bármikor lehetővé kell tenni, hogy újrainicializálja az eszközt.

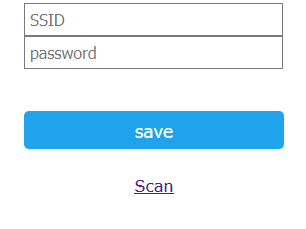
A feladat megoldásához az Arduinohoz készült külső WifiManager könyvtárat használtam fel, ez lehetővé tette, hogy az első indításkor a felhasználó beállíthassa a hálózati hiteletesítési adatokat, amelyet az eszköz a saját EEPROM-jába ment le, így a konfiguráció egyszeri szükséges művelet, minden további indításnál rendelkezésre állnak az adatok.

A felhasználói folyamat lépéseit néhány ábrán mutatom be. Az inicializálandó eszköz indulását követően *ESPConfig* SSID-vel indul AP-ként, a csatlakozáshoz használt jelszó alapesetben *password*. A csatlakozást követően böngészőből a 192.168.4.1 címen elérhetjük a konfigurációs felületet, ahogy azt a 3.3. ábra is mutatja.



3.3. ábra ESP konfigurációs felület nyitóoldala

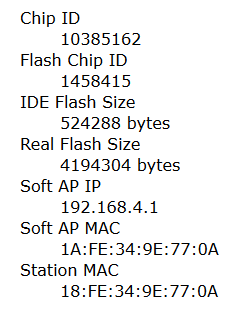
A felső két opció használható a konfiguráláshoz, a különbség, hogy az elsőben a nyilvánosan látható SSID-vel rendelkező hálózatokat az ESP kilistázza, így mi ezeket listából is kiválaszthatjuk. A másodikban egyszerűen megadjuk a hálózat két paraméterét a felbukkanó ablakban. (3.4. ábra)



3.4. ábra Hálózati adatok megadása a konfigurációs felületen

Az követően válaszként értesülünk a művelet sikerességéről, az ESP újraindul, ezúttal már a saját működési módjában.

Az Info menüpontban megtekinthetjük az eszközünk alapadatait, ezek közül a MAC címre szükség is lehet majd az eszköz szerver oldali regisztrációjánál. (3.3. ábra)



3.5. ábra Info oldal a konfigurációs felületen

A Reset opciót választva az eszköz törli az esetlegesen mentett adatokat és újraindul.

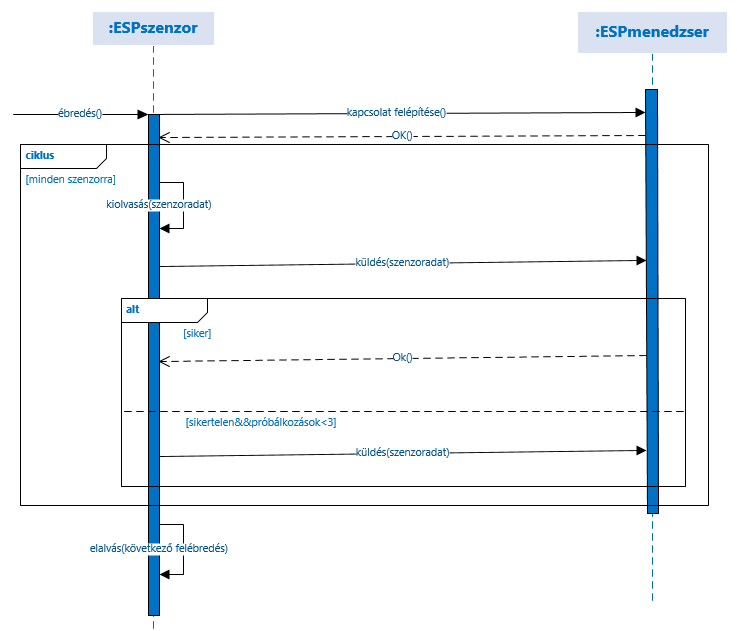
## Működési módok

A 3.3 fejezet megmutatta, hogy az ESP eszközök több, felelősségét tekintve jól elkülöníthető funkciókat biztosítanak. A szenzor, beavatkozó és menedzser eszközöket a lakás különböző pontjain lehet elhelyezni igény szerint. Az elkülönített működési módok nem jelentik azt, hogy egy eszköz csak egy feladatot képes ellátni: a lakásnak lehetnek olyan pontjai, ahol egy helyen van szükségünk szenzorokra és beavatkozókra is. Ilyenkor cél lehet, hogy ne kelljen két modult telepítenünk, de figyelembe kell venni, hogy a működések és megszorításaik eltérhetnek, ezekre figyelni kell az implementációnál. Például az olyan beavatkozó eszköz, mely szenzor kiolvasási és továbbítási műveleteket végez, nem aludhat el a szenzoradatok kiolvasását követően, hiszen a beavatkozó feladatai folyamatos kapcsolatot feltételeznek a menedzser eszközzel. Helyette a szenzor életciklus egyszerűen egy ütemezett feladat, ami bizonyos időközönként lefut.

### Szenzor

A szenzorhálózat csomópontjaiban az eszköz fő feladata, hogy előre meghatározott időközönként érzékelőivel méréseket végezzen, majd a kiolvasott adatokat továbbítsa a menedzser eszköznek. Ebben az üzemmódban az eszköz az idő legnagyobb részében valójában nem csinál semmit, hisz két mérés között az elküldést követően a modulnak semmilyen feladata sincs. Emiatt a fogyasztás optimalizálása érdekében ebben az időszakban célszerű alvó módban üzemeltetni az eszközt megmondva, hogy mennyi idő elteltével ébredjen fel újra az új mérés elvégzéséhez. Ha ilyen módon készítjük el a programunkat, akkor optimális esetben az eszköz akár elemről is működtethetővé válik (hosszú ideig, hónapokig, vagy akár évekig működhet elemcsere nélkül), ami kihelyezett szenzorhálózati csomópontoknál egy fontos szempont lehet.

A fejlesztés eredményeként a beállított szenzor eszköz egy egyszerűsített ciklusa a következőképp vázolható.



3.6. ábra Egy szenzor életciklusa

A folyamat mindig úgy kezdődik, hogy a szenzor eszköz elindul, ami legtöbb esetben a deep sleep állapotból való felébredést jelenti, de a már beállított eszköz tápfeszültséget kapva (például elemcserénél) szintén ezt a folyamatot indítja el.

Kliensként a TCP hálózaton keresztül kapcsolódik a menedzser eszközhöz, a kapcsolat felépítésének sikeressége esetén pedig végigmegy a saját, regisztrált szenzorain, egyenként elvégzi az értékek kiolvasását és rögtön továbbítja is a menedzser eszközöknek. Ha az adatok továbbítása nem sikerül, akkor meghatározott számban újra próbálkozik a művelettel. Az összes szenzoradat átküldését követően az eszköz visszatér alvó üzemmódba, a következő mérési ciklus kezdetéig nincs feladata.

A beküldésnél a szenzornak az alábbiakat kötelezően meg kell adnia:

(TODO, a táblázatot átírni, ez nem pontos már)

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#### Alvó üzemmód

Volt szó már többször arról, hogy a szenzor modulok két mérési ciklusuk között, az idő nagy részében nem végeznek semmilyen műveletet, így ilyenkor alvó állapotba kerülnek energiatakarékossági szempontok miatt. Ezt az állapotot érdemes kicsit közelebbről is áttekinteni. Az ESP8266 alapvetően három különböző szintjét támogatja az alvó üzemmódnak, ezekről ad áttekintést a 3.1. táblázat, ahonnan kiolvasható, hogy az egyes állapotok mely komponensekre vannak hatással, és milyen fogyasztás csökkenést várhatunk a használatukkal.

|  |  |  |  |
| --- | --- | --- | --- |
| **Jellemző** | **Modem-sleep** | **Light-sleep** | **Deep-sleep** |
| WiFi | KI | KI | KI |
| Rendszer óra | BE | KI | KI |
| RTC[[17]](#footnote-18) | BE | BE | BE |
| CPU | BE | Felfüggesztve | KI |
| Fogyasztás | 15 mA | 0.4 mA | ~ 20 µA |

3.1. táblázat Alvó üzemmódok az ESP8266-on

Látható, hogy a három alvó üzemmód fokozatosan a modul egyre több komponensének kikapcsolásával éri el a fogyasztás csökkentését, az, hogy melyiket érdemes használni, az mindig a használat körülményeitől függ:

1. Modem-sleep módba kerüléshez feltétel, hogy az eszköz stationként csatlakozzon egy meglévő hálózathoz. A WiFi modem kikapcsolásra kerül, de az AP-től DTIM[[18]](#footnote-19) érkezése esetén automatikusan felébred az eszköz. Modem-sleep módba az eszköz kerülhet automatikusan és programból történő hívás által is.
2. A Light-sleep üzemmód hasonlóan működik, mint a Modem-sleep, a különbség, hogy itt az eszköz rendszerórája is kikapcsolásra kerül, és a CPU működése felfüggesztődik. Ilyenkor az eszköz felébresztése a külső GPIO lábon keresztül történhet meg. Tipikus használati esete, mikor azt szeretnénk, hogy az eszközünk kapcsolódva maradjon a routerhez, és a kapott üzenetekre tudjon válaszolni azonnal, de a köztes időben nincs szükség műveletek elvégzésére. AZ ESP eszközök egy hálózathoz történő csatlakozás után automatikusan Light-Sleep üzemmódba kerülnek.
3. Deep-sleep üzemmódba az eszközök a korábbi két móddal ellentétben nem kerülhetnek automatikus módon. A programból történő hívást követően az eszköz azonnal mélyalvó üzemmódba kerül, a fentieken felül ez a CPU teljes kikapcsolásával jár, egyedül az RTC marad bekapcsolva, mivel ennek a modulnak a felelőssége, hogy ha az üzemmódba történő váltáskor a hívás során a felhasználó megadott időtartamot is, ennek leteltét követően megtörténjen a felébredés. Ehhez az eszköz GPIO16 lába az EXT\_RSTB gombra kell legyen kötve, másképp nem fog megtörténni az automatikus felébredés. A GPIO16 láb külső triggerelésével az eszköz manuálisan is felébreszthető.

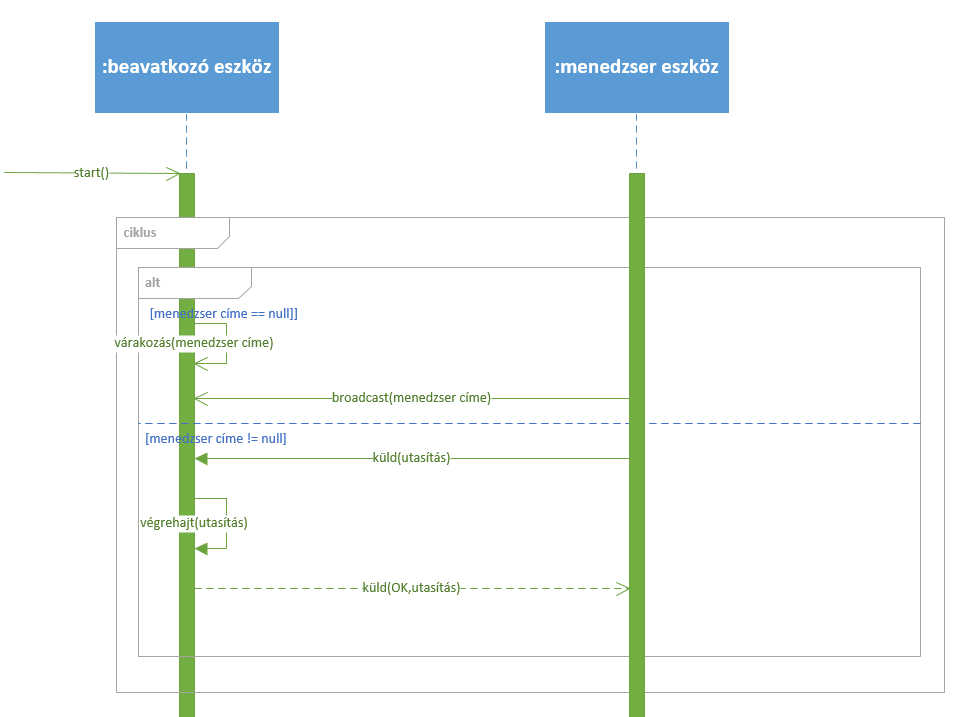
Mivel a szenzor eszközök elsődlegesen maguk kezdeményezik a kommunikációt, a működési ciklus két lefutása között akkor járok a legjobban, ha a deep-sleep üzemmódot használom. A menedzsertől általában a visszaigazoláson kívül nincs szüksége az eszköznek más információra, amikor mégis, a visszaigazolással egyidőben elküldött üzenettel ez bármikor jelezhető.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Beavatkozó

Beavatkozó üzemmódban az eszköz célja, hogy a környezetének működésében változásokat eszközöljön a kapott utasítások alapján. Ennek biztosítására folyamatos kapcsolatot kell fenntartania a menedzserrel, a beérkező üzenetekre pedig a lehető leghamarabb reagálnia kell-

A beavatkozó működési folyamatának főbb lépéseit a 3.4. ábra mutatja be.



3.7. ábra Beavatkozó működése

A folyamat néhány inicializációs feladatot követő első lépése a menedzser által szolgáltatott TCP szerverhez való csatlakozás. Amennyiben az eszköz nem ismeri a szerver IP címét, akkor várakozik, a menedzser feladata ugyanis, hogy bizonyos időközönként broadcast UDP üzenetben elküldje azt. A cím segítségével már felépíthető a kapcsolat, ezt követően a beavatkozó feladata

### Menedzser

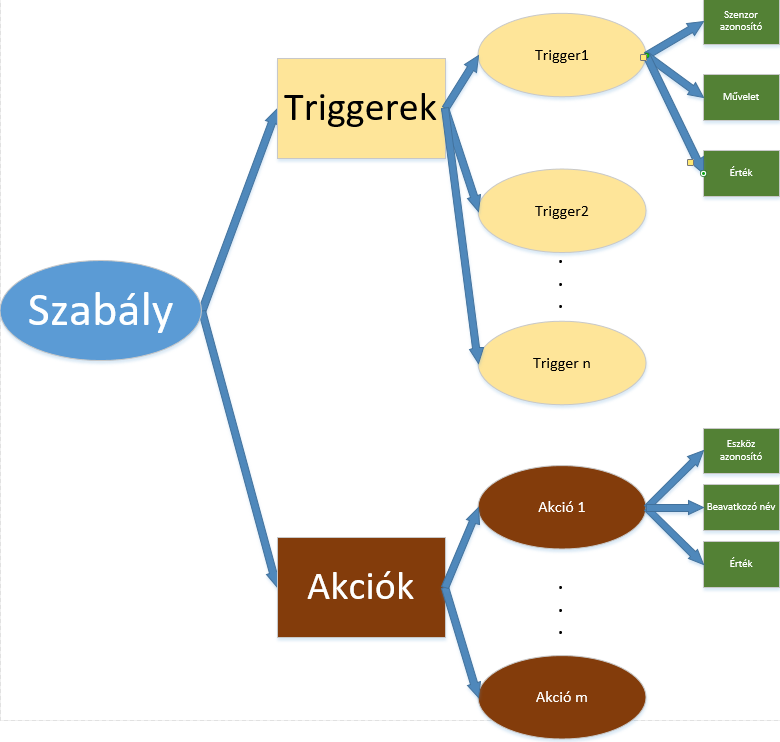
A menedzser működése lényegesen bonyolultabb a szenzor és beavatkozó típusú eszközökénél, hiszen a feladata sokrétűbb és a rendszer minden másik elemével kapcsolatban van. A menedzser által végzett főbb feladatok:

1. Periodikusan közli UDP broadcast üzenetek küldésével az elérhetőségét a többi komponens számára, hogy a rendszerben esetlegesen újonnan megjelenő eszközök is csatlakozhassanak hozzá.
2. Periodikusan frissíti a saját adatbázisát a szervertől lekért szabályokkal, hogy a felhasználók által a szerveren készített új automatizálásokról is értesítést szerezzen.
3. A lakás automatizálásának vezérlése

A lista 3. pontjában említett feladat ellátása egy összetett funkció, és mint az egész rendszer egyik legfontosabb része, érdemes részletesebben is kifejteni. Ehhez első lépésben bemutatom a vezérlési szabályok mögötti logikai elgondolást, utána pedig a menedzser kiértékelési folyamatát.

#### Vezérlési szabályok felépítése

A szabályok szerkezetének kigondolását a feladat egyik kulcsfontosságú részének érzem: olyan megoldást szeretnék, ahol a felhasználók részéről nem igényel mélyebb programozási ismereteket szabályok létrehozása, ugyanakkor az egyszerűség nem korlátozza túlságosan a kevésbé triviális működések automatizálását sem. Sajnos a két szempont kicsit szembemegy egymással, így kompromisszumokat kellett vállalnom a megalkotás során. A kitalált szerkezetet a 3.5. ábra mutatja be.



3.8. ábra Szabály felépítése

Láthatjuk, hogy egy szabály két részből tevődik össze:

1. **Triggerek:** A szabály feltétel része, melyből egy szabályhoz tetszőlegesen sok tartozhat. A triggerhez tartozó mezők jelentését a **3.2**. táblázat írja le.

|  |  |  |  |
| --- | --- | --- | --- |
| **Paraméter** | **Jelentés** | **Lehetséges értékek** | |
| Szenzor azonosító | Annak a szenzornak a sorszáma, melynek az utolsó mért eredményére való vizsgálatot el szeretnénk végezni | egész szám |
| Művelet | Az elvégzendő matematikai összehasonlító művelet, amit vizsgálni szeretnénk | összehasonlító operátorok (== ; != ; <; <=; >; >=, !=) |
| Érték | Az a számérték, amihez a szenzor által mért értéket hasonlítani szeretnénk | szám |

3.2. táblázat Trigger paraméterei

Egy trigger kiértékelése során azt a szenzoradatot vesszük alapul, amelynek szenzor azonosítója megegyezik a triggerben lévő szenzor azonosítójával. A trigger kiértékelése IGAZ vagy HAMIS értéket adhat aszerint, hogy a [mért szenzoradat] [operátor] [érték] behelyettesítésekkel kapott összehasonlítás milyen logikai értéket ad vissza. Például, ha a mért érték 30.2, az operátor „<=”, a trigger érték 50, akkor a (30.2 <= 50) összehasonlítás IGAZ lesz. A szabályon belül a triggerek teljesülésének ÉS kapcsolatát vesszük a teljes kiértékelésénél, azaz a szabály következtében a menedzser akkor küld el üzenetet a beavatkozó(k)nak, ha minden trigger feltétel teljesül.

1. **Akciók:** Azokat a beavatkozásokat gyűjtjük itt, amit a feltételek teljesülése esetén el szeretnénk végezni. Az akcióhoz tartozó tulajdonságokat a 3.3. táblázat

|  |  |  |  |
| --- | --- | --- | --- |
| **Paraméter** | **Jelentés** | **Lehetséges értékek** | |
| Eszköz | Annak a modulnak az azonosítója, amelyen a beavatkozó található | egész szám |
| Beavatkozó név | Az eszközön található beavatkozó egyedi neve | szöveg |
| Érték | Az az érték, amire a beavatkozó állapotát módosítani szeretnénk | beavatkozótól függ, de alapvetően szöveges |

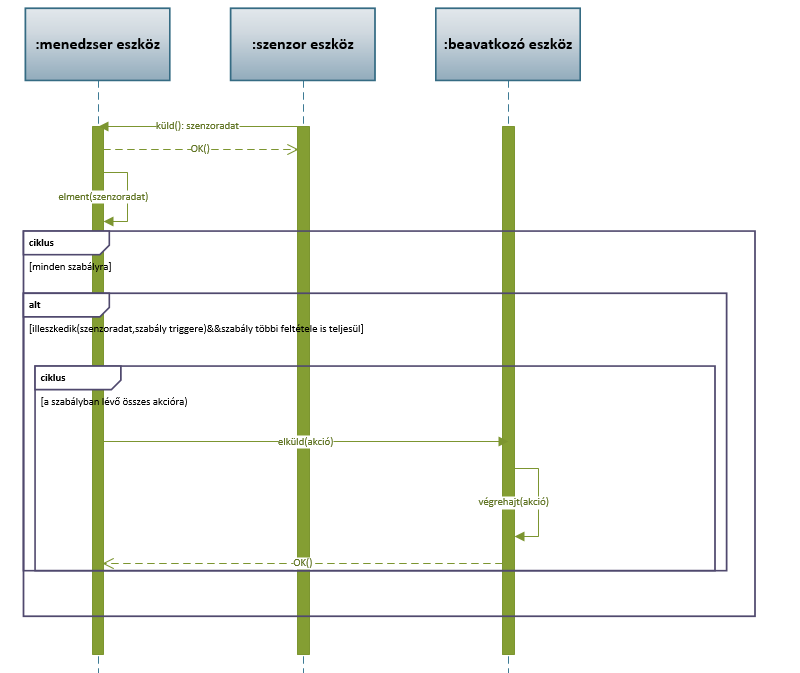
3.3. táblázat Akció paraméterei

A menedzser eszköz a szabály teljesülése esetén minden, a szabályhoz tartozó akción végigmegy, és az eszköz azonosítójához tartozó csatornán kiküldi az üzenetet a megfelelő eszköznek, aki elvégzi az üzenet alapján a szükséges beavatkozásokat.

A fentiek szerint megalkotott szabályok a feltételek minden logikai kapcsolatát képesek reprezentálni, hiszen ha a feltételek között ÉS kapcsolatot szeretnénk alkotni, akkor több triggert hozunk létre egy szabály alatt. VAGY kapcsolat reprezentáláshoz pedig egyszerűen több szabályt kell létrehozni ugyanazzal az akcióval. Egy szabály tetszőleges mennyiségű akciót válthat ki, így a struktúra megalkotásával lényegében sikerült megoldani a vezérlési szabályok kezelését, csak a fentiekben megfogalmazott, nem túl bonyolult működésű végrehajtást kell implementálni a menedzser kódjában.

#### Szabályok kiértékelése

A folyamat alapvetően úgy működik, hogy egy beérkező szenzoradatra a menedzser reagál a 3.5. ábra által leírt módon:



3.9. ábra A menedzser reagálása egy érkező szenzoradatra

Az érkező szenzoradatra a menedzser választ küld a szenzor eszköznek, majd végigmegy az összes szabályán. Ha ebben a listában valamelyik szabály triggerjének szenzorazonosítójával egyezőt talál, akkor ha vannak, a szabály többi feltételének teljesülését is megvizsgálja, azaz a többi trigger teljesülését is validálja. Ehhez minden rendelkezésére áll, hiszen a saját adatbázisában tárolja minden szenzornak az utolsó elküldött értékét is. Egyetlen hátrány, hogy így a tárolt adatok adott esetben már nem biztos, hogy tükrözik az aktuális állapotot, ezt vállaljuk annak érdekében, hogy a szenzorok a korábban bemutatott saját mérési ciklusuk szerint működhessenek, valamint a szenzorok igény szerinti gyakorisággal mérnek, így a problémákat ki tudjuk küszöbölni.

Ahogy a 3.1. ábrán is láthattuk korábban, a menedzser további feladata még a központi szerver irányába történő kommunikáció is: egyrészt rendszeresen továbbítania kell a szenzorok által beküldött, általa letárolt adatokat, valamint frissen kell tartania a saját környezetéhez tartozó szabályadatbázisát. Ehhez a szerver által szolgáltatott HTTP interfészt fogja használni, a lehetséges kéréseket a szerver bemutatása során ismertetem.

#### Kommunikáció az eszközökkel

A menedzser a beavatkozókkal és szenzorokkal egyaránt TCP kapcsolaton keresztül kommunikál, ahol szerverként funkcionál, folyamatosan lehetővé téve a kliensek csatlakozását.

A rendszerek könnyű egymáshoz illeszthetősége érdekében az üzeneteket a menedzser egyaránt JSON formátumban küldi és fogadja. Ez nem csak könnyebben kezelhetővé teszi az üzeneteket, hanem felkészíti a rendszerünket arra is, hogy esetlegesen teljes komponenseket cseréljünk ki nem ESP-s eszközökre is, amíg azok az üzenetek formátumára szabott elvárásokat teljesítik. Bár kényelmes lenne, az adatokat az eszköz nem JSON formátumba tárolja, mivel az fölösleges memóriapazarlást jelentene, memóriából pedig főleg a menedzser esetében így is szűkösek az erőforrások. A szabályok formátumát a 2.2.1 fejezetben már példaként láthattuk, ezen felül a menedzsernek csak a szenzor és beavatkozó eszközök listájának lekérésekor, a szenzor mérések küldésekor és fogadásakor, illetve beavatkozók utasításakor van szüksége üzenetben kapott vagy küldött adatokra, ezekben az esetekben is JSON formátumot használ.

Mivel az összes, egy lakásban lévő eszközünknél feltétel, hogy a közös belső hálózaton legyenek, a rendszer külső beavatkozás elleni védelmét úgy oldhatjuk meg, hogy nem engedünk be külső szereplőket, ami lényegében a vezeték nélküli hálózatunk megfelelő védelmével biztosítható. Egy erős, WPA2 jelszó használatával a rendszerbe csak a kívánt elemek léphetnek be, akiktől megbízható voltuk miatt nem kell külön védenünk az ESP modulok hálózaton belüli kommunikációját. Az egyetlen problémát a kifelé történő kommunikáció okozza, azaz amikor a menedzser kommunikál a szerverrel, ennek a problémának a megoldására az eszköz a kérésekben query paraméterként elküldi a jelszava SHA256-al titkosított formáját, amit a szerver ellenőrizhet. Ennél jobb módszer lenne az, ha a mindkettejük által ismert jelszóhoz egy időbélyeget rakva (mondjuk perces pontosságra „kerekítve”, hogy a nem pontosan szinkronban lévő órák ne okozzanak gondot) kódolná a jelszót mindkét oldal, ilyen esetben a kívülről megszerzett jelszó is csak rövid ideig lenne érvényes. Vagy akár minden kérésnél az eszköz a kapna egy szerver által generált kulcsot, amit a következő kérésnél használhat fel, szintén a jelszó+kulcs hashét képezve. Az előbbi megoldással próbálkoztam, de sajnos az összeszinkronizált órák ellenére akadtak problémák, így végül idő hiányában az egyszerű, de kevésbé biztonságos megoldást választottam.

## Központi szerver

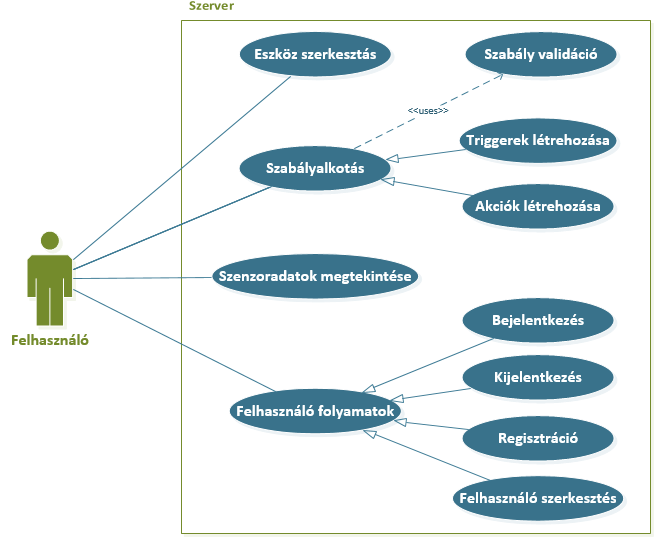
A központi szerver működése az egész rendszer szempontjából kritikus, hiszen a felhasználók számára itt van lehetőség az eszközök menedzselésére és a vezérlési szabályok megalkotására. A szervernek ESP eszközök közül közvetlenül csak a menedzserekkel kell kommunikálnia, azt is csak a menedzserek által kezdeményezett HTTP lekérdezéseken keresztül. Ennek megfelelően alapvetően két fő feladatkört tudhatunk a központi szervernek.

hogy a felhasználók felügyelhessék és üzemeltethessék a saját lakásukban található eszközök működését. Ez jelenti az adatsorok követhetőségét grafikonos formában, illetve lehetőséget a rendszer teljes konfigurálására.

Ezen felül pedig a szükséges adatsorokat hozzáférhetővé kell tenni a menedzser eszközök számára, hiszen az automatizálás üzleti logikájában a szerver közvetlen módon már nem vesz részt, a vezérlés mindig a menedzserek feladata.

### Használati esetek

A lehetséges használati eseteket a 3.6. ábra szemlélteti, a későbbiekben ezeket egyenként is meg fogom vizsgálni, elemzem a szerepüket és megmutatom, milyen módon nyújt lehetőséget a szerver az elvégzésükre. Az adatbázis terv bemutatásához viszont szükség van, hogy képet kapjunk a szerver alapvető képességeiről.



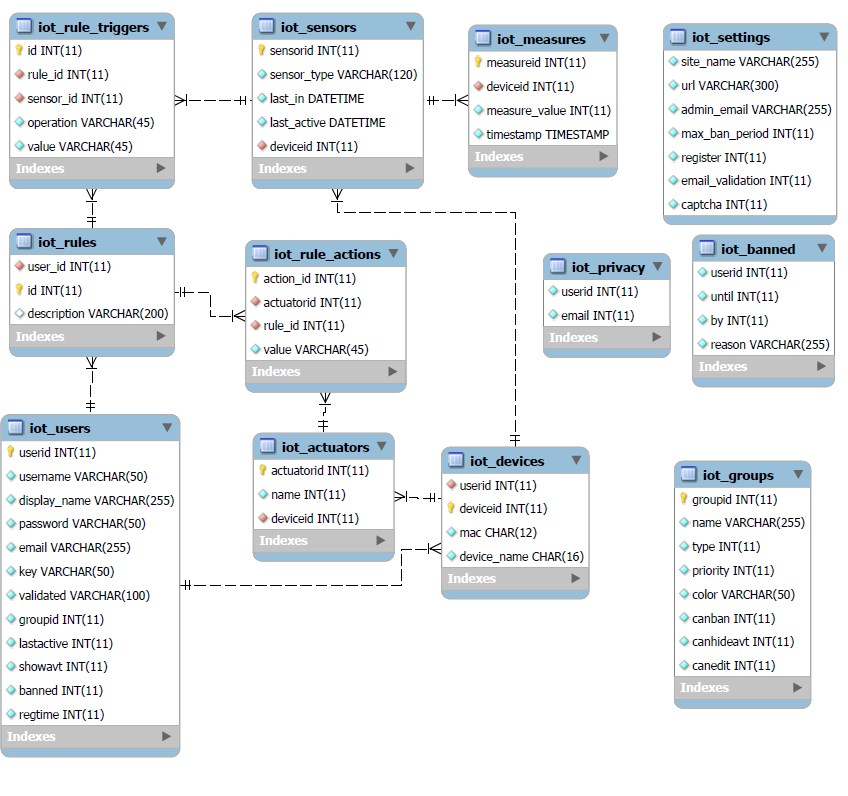
3.10. ábra Felhasználói interakciók a szerveren

Az alapvető felhasználói műveletekről már volt szó, itt csak a minden rendszerben lévő általános feladatokra kell gondolni, mint a regisztráció, ki-és bejelentkezés, illetve az adatok változtatásának lehetősége.

A korábban definiált követelményeknek megfelelően a regisztrált felhasználók hozzáférhetnek a saját eszközeikhez: ezeket megjeleníthetik és szerkeszthetik, hozzáférhetnek a szenzoradatokhoz, továbbá szabályokat készíthetnek a hozzájuk tartozó triggerek és akciók megvalósításával.

### Adatbázis terv

A fejezetben a rendszer eddig ismertetett tervei alapján elkészíthető a szerver adatbázisa, melynek tervét a 3.8. ábra mutatja be. A rendszeren belüli táblák nevei egységesen „iot\_” prefixet kaptak, ezt követően igyekeztem beszédes elnevezést adni nekik.



3.11. ábra Szerver adatbázis felépítése

Az adatbázis terv a már korábban megtervezett elemekhez tartozó táblákat írja le. Az egyes táblákhoz tartozó tartalmakat a 3.4. táblázat mutatja be.

|  |  |
| --- | --- |
| **Tábla** | **Tartalom** |
| iot\_settings | A rendszer fő konfigurációs adattáblája, itt találhatóak meg az olyan rendszerszintű paraméterek, mint például az oldal neve, az url (a belső hivatkozásokhoz) vagy a regisztrációt letiltható kapcsoló |
| iot\_banned | A keretrendszer funkciója, hogy a felhasználókat időlegesen bannolhatják az adminisztrátorok, a tábla ezeket tartja nyilván. A dolgozatban ezt a funkciót csak kipróbáltam, a jelenlegi rendszerben valószínűleg nincs nagy szükség rá. |
| iot\_groups | Felhasználói csoportokat tart nyilván, alapesetben normál felhasználókat és adminisztrátorokat, de igény szerint fel lehet venni a rendszerbe új csoportokat is, a tábla mezőinek állításával beállítva a jogaikat. |
| iot\_privacy | Felhasználói beállításokat tartalmaz, jelenleg csak azt, hogy a felhasznáűló email címét más (nem admin) felhasználók láthatják-e |
| iot\_users | A felhasználókat tárolja. A rendszer jelen formájában a felhasználóhoz közvetlenül vannak rendelve az eszközök is, így ez a tábla reprezentálja a rendszerünkben a lakást is. Itt tároljuk a főbb felhasználói adatokat, például a profilhoz tartozó jelszót (illetve annak SHA-256-al kódolt változatát), email címet stb |
| iot\_devices | A rendszerben fizikailag jelen lévő (ESP) modulok, melyek egyedi MAC[[19]](#footnote-20) címmel rendelkeznek (ahogy minden hálózati eszköz a világon) |
| iot\_sensors | Az iot\_devices táblában definiált eszközökön lévő szenzorok. A típusa mellett azt is nyilván tartjuk, mikor jött be például utoljára adat a szenzortól |
| iot\_measures | Mérési eredmények, idegen kulcson keresztül a megfelelő szenzorokhoz kötve. Az értéke mellett a mérés (nem a szerverre beküldés) időpontját is tároljuk. |
| iot\_actuators | Az iot\_devices táblában definiált eszközökön lévő beavatkozók. A beavatkozó állapotát is itt tarjuk nyilván. |
| iot\_rules | A vezérlési szabályok, minden szabály egy felhasználóhoz tartozik. A szabályokhoz tárolunk egy leírást is és a nevét is a felhasználói követhetőség miatt |
| iot\_rule\_triggers | A szabályokhoz tartozó trigger feltételek, a korábban kifejtett struktúrának megfelelően egy összehasonlító operátort és az értéket is tartalmazzák a sorok |
| iot\_rule\_actions | A szabályokhoz tartozó cselekvések, tárolva, mely beavatkozó értékét mire kell módosítani |

3.4. táblázat Az adatbázis táblái

### Felhasználói felület

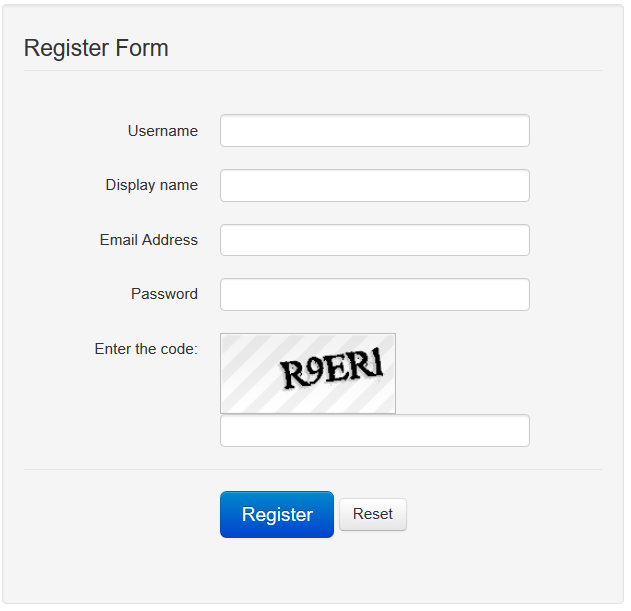
Az alfejezetben röviden bemutatom a szerver oldali felhasználói felület fő oldalait. A főbb elérési pontokat a 3.5. táblázat foglalja össze, ahol a cím oszlop az oldalon belüli relatív hivatkozást adja meg. A grafikus felülettel (azaz közvetlenül a felhasználónak szánt) oldalak közül a fontosabbak rövid bemutatására a táblázatot követően kerül sor.

|  |  |  |
| --- | --- | --- |
| **Típus** | **Cím** | **Funkció** |
| GET | /init/init.php | A rendszer telepítésénél használt konfigurációs felület |
| GET | /register.php | Regisztrációs felület |
| GET | /users\_list.php | Felhasználók listázása |
| GET | /contact.php | Kapcsolat form, adminisztrátornak küldhet itt üzenetet a felhasználó |
| GET | /my\_devices.php | Felhasználó eszközeinek táblázatos megjelenítése |
| GET | /add\_device.php | Eszköz felvétele felület |
| GET | /add\_sensor.php | Szenzor felvétele felület |
| GET | /add\_actuator.php | Beavatkozó felvétele felület |
| GET | /viewdata.php?sensorid=[id] | Szenzor adatsorok grafikus megtekintése |
| POST | /add\_data.php?sensorid=[id]&&  measure\_time=[time]&&value=[value]  &&password=[password] | Szenzor adatok beküldése |
| GET | /add\_rule.php?rule\_id=[id] | Szabályszerkesztő felület |

3.5. táblázat Elérési pontok a szerveren

#### Regisztráció

Egy otthoni rendszer felépítése minden esetben egy felhasználó létrehozásával kell kezdődjön. A rendszer jelenlegi struktúrája szerint a felhasználó egyben reprezentálja magát a lakást, külön otthonok létrehozásánál ezért célszerű minden otthonhoz új felhasználót létrehozni. A regisztrációs felület erre ad lehetőséget, ami teljesen szokványos, néhány alapvető felhasználói adat megadásával létrehozható a felhasználói profil, ahogy azt a 3.9. ábra is szemlélteti.

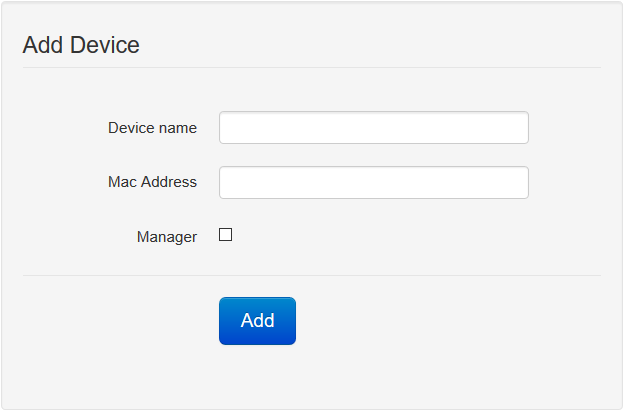


3.12. ábra Felhasználói profil létrehozása

Az oldal alapbeállítások mellett nem kéri az email cím megerősítését, de igény esetén erre is lehetőség van.

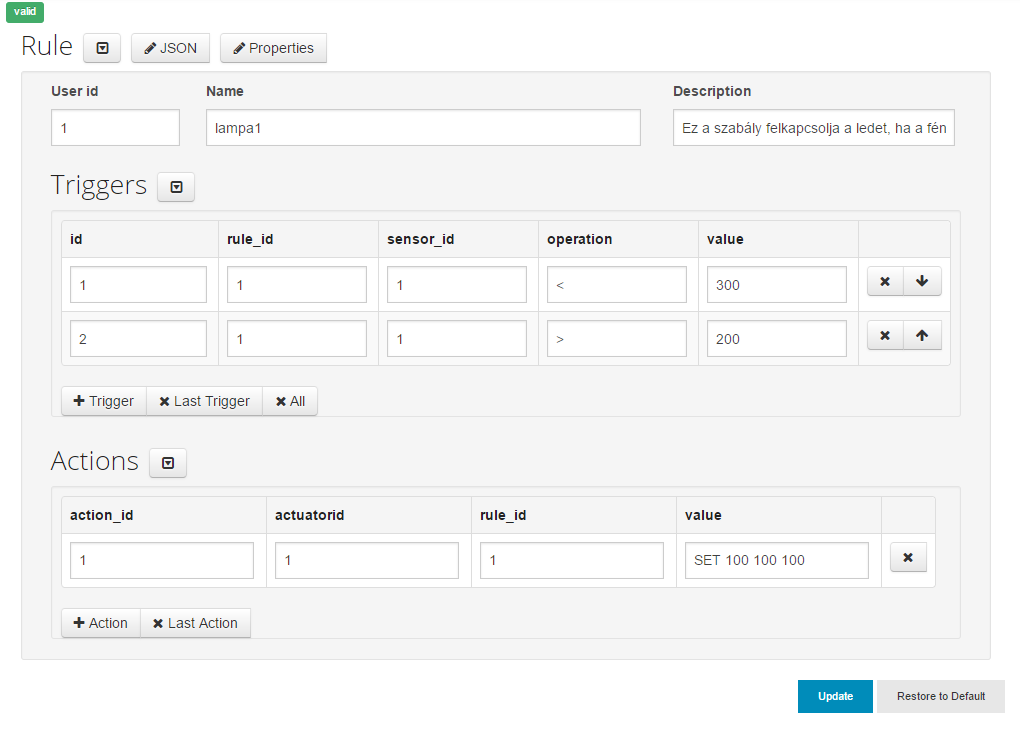
#### Vezérlési elemek felvétele a rendszerbe

Az eszközök, illetve rajtuk található szenzorok és beavatkozók vezérlésére a szerver egyszerű formokat biztosít, itt felvehetjük, illetve szükség esetén módosíthatjuk a szükséges fizikai eszközöket. Eszköz felvételére mutat példát a 3.13. ábra, ahol az eszköz nevén és MAC címién túl megadhatjuk, hogy az eszköz managerként funkcionál-e. Ezekre a mezőkre alapvető validálásokat is végez a rendszer.



3.13. ábra Eszköz felvétele

A szabályokat létrehozó felület összetettebb, itt fontosnak tartottam, hogy a könnyű szerkeszthetőséghez is minél több támogatást adjak. Mivel a szabályok JSON reprezentációja egyébként is rendelkezésre állt, úgy döntöttem, a szerkesztéshez egy JSON editor felületet fogok használni. A felületet egy egyszerű példaszabállyal a 3.14. ábra mutatja be. A JSON sémaleírójának köszönhetően az érvényes szabályok követelményeinek mindig ellenőrizhetőek és könnyen betarthatóak, az id mezőkbe például nem tudunk felvenni szöveges adatokat, csak számokat.



3.14. ábra Szabályszerkesztő felület

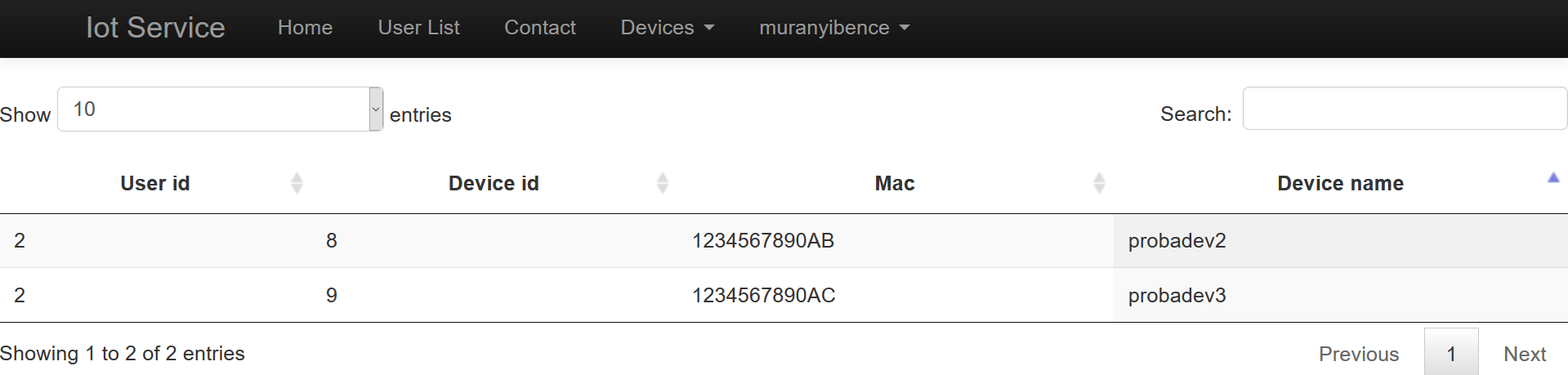
A JSON gombra kattintva előhozható a szabály aktuális JSON megfelelője, és természetesen itt is szerkeszthető szövegesen a szabály, a Save gombra kattintva a változtatások meg fognak jelenni a szerkesztőfelületen is . (3.15. ábra, az egész szabály nincs rajta a helyigény miatt, de talán nincs is rá szükség)



3.15. ábra Az aktuálisan szerkesztett szabály JSON reprezentációja

#### Eszközök listázása

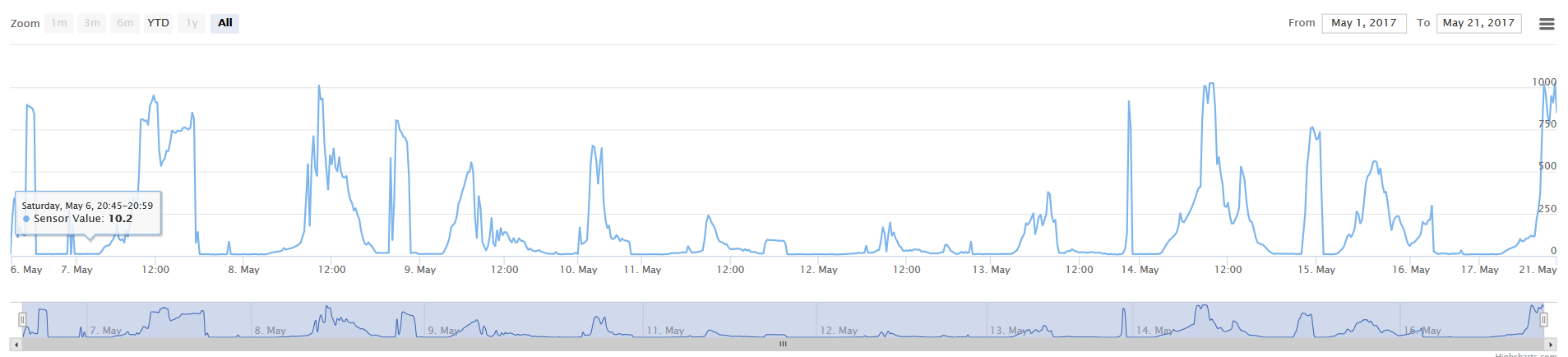
A felület lehetőséget nyújt az elemek táblázatos listázására is. A felhasználóhoz tartozó eszközöket felsoroló táblázatban (3.16. ábra) például kereshetünk a felhasználó neve szerint és a táblázat egyszerre megjelenő sorait szabályozhatjuk.

******

3.16. ábra Eszközök böngészése táblázatban

#### Szenzor adatsorok megjelenítése

A saját hálózaton belüli adatsorok megjelenítése a szerver egy másik fontos funkciója (3.17. ábra) . A HighCharts API biztosítja, hogy az adatsor egy tetszőleges időszeletét tekintsük meg, a grafikon felett látható szűrő mezőkkel és gombokkal, illetve az alatta található „csúszkákkal” bármely részre rá lehet nagyítani.



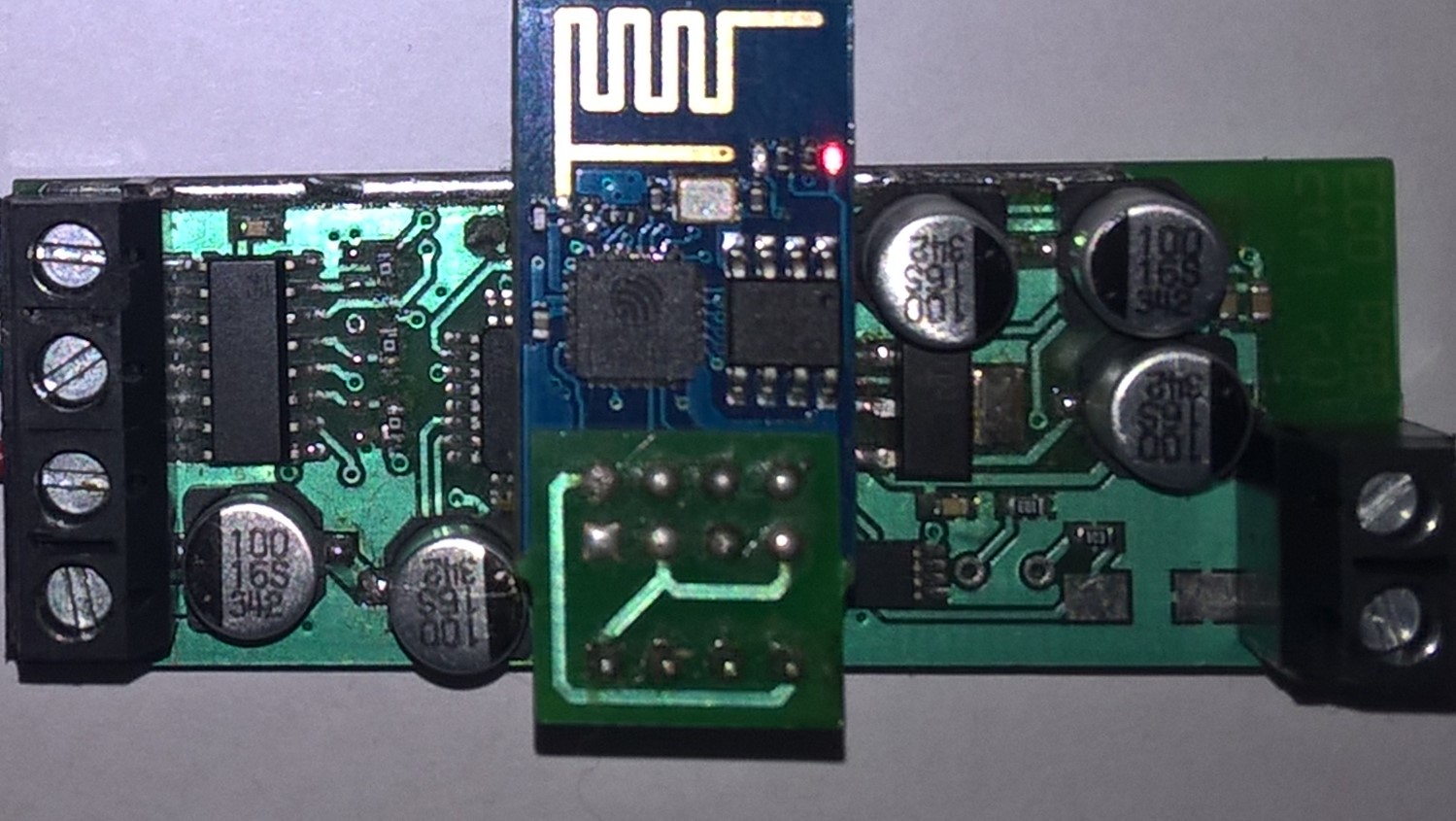
3.17. ábra Egy fényerősségmérő szenzor adatainak megjelenése a szerveren

# Tesztelés

A dolgozat utolsó fejezetében egy példa összeállításon keresztül szeretném demonstrálni a működést. A bemutatást követően értékelem az elkészült rendszert, melyet néhány hetes futtatás eredményei alapoznak meg. Az értékelésen felül a rendszer fejlesztési lehetőségeire vonatkozóan adok áttekintést, valamint összefoglalom a diplomaterv készítése során szerzett tapasztalataimat.

## Tesztelendő rendszer létrehozása

A teszteléshez egy olyan rendszert állítottam össze, melyben a szerveren felül a lakásban három eszközt helyeztem el, minden szerepkörhöz tartozóan egyet. A szenzor és menedzser szerepekre az 1.4 fejezetben bemutatott tesztpanelt használtam. Beavatkozóként egy ESP01-es eszközt használtam, melyhez egy RGB modult csatlakoztattam az eszköz UART interfészén keresztül, a modulhoz pedig egy LED szalagot illesztettem. (4.1. ábra)



4.1. ábra ESP01 RGB modullal

Az UART interfészen keresztül az RGB modul a SET [RED] [GREEN] [BLUE] formátumban érkező üzenetekre a színkódnak megfelelően állította be a szalagon a fényt. Az ESP így beavatkozó eszköz módban futva bármikor felkapcsolhatta a fényeket. A tápellátás is biztosítva van a működéshez az RGB modulon keresztül.

A szerver oldalt egy saját VPS[[20]](#footnote-21)-re telepítettem, a telepítés pontos környezetét a 4.1. táblázat írja le.

|  |  |  |
| --- | --- | --- |
| **Típus** | **Szoftver neve** | **Verzió** |
| Operációs rendszer | Debian Linux | 7 |
| Operációs rendszer kernel | Linux | 3.2.0-4-686-pae |
| Adatbázis | MySQL | 5.5.47 |
| Webszerver | Apache | 2.2.22 (Debian) |
| PHP | PHP | 5.4.45 |

4.1. táblázat A tesztrendszeren használt szoftveres környezet

Természetesen a fentiketől eltérő verziójú környezetben is jól kell futnia a rendszernek, de a tesztelést ezen az összeállításon végeztem el.

A szerver telepítéséhez a szoftveres környezet előállítását követően egy adatbázis létrehozása a feladat. Biztonsági okokból célszerű a rendszergazda felhasználó helyett az adatbázishoz külön felhasználót is létrehozni, aki csak erre az adatbázisra kap jogokat.

A szerver webes elérési pontjára be kell másolni a szerver könyvtár teljes tartalmát. Ezt követően legegyszerűbben a szerveren belül */init/init.php* elérési ponton található konfigurációs oldalon keresztül állítható be a szerver alkalmazás, megadva az alapadatokat (oldal neve, stb), valamint az adatbázist és az adatbázishoz tartozó felhasználót.

Az inicializálási lépést követően elviekben a rendszerünk készen áll az eszközök felvételére, ezt a korábbiakban ismertetett módon tettem meg.

## Funkcionalitási teszt

Első körben megvizsgáltam, hogy teljesít a rendszer egy szabály triggerelése esetén. Ehhez a 3.14. ábra szerinti szabályt vettem fel, illetve ideiglenesen annyi módosítást végeztem a menedzseren, hogy az UART interfészen keresztül írja ki az aktuális történéseket. Tapasztalatból úgy állítottam be a fényerősségmérő szenzor triggert, hogy kézzel kissé letakarva 200 és 300 közötti értékeket mérjen. A menedzser által visszaadott eredmények:

GOT sensordata: {sensor\_id:1, measure\_time: 1495305035, measure\_value: 221}

CHECK RULE(rule\_id:1) TRIGGERS (sensor\_id:1,measure\_time:1495305035)

TRIGGER(trigger\_id:1, sensor\_id:1) MATCH ON sensordata(sensor\_id:1, measure\_time: 1495305035)

TRIGGER(trigger\_id:1) calculate(measure\_value:221,operation:„<”,value:300) result:TRUE

TRIGGER(trigger\_id:2, sensor\_id:1) LAST value:221

TRIGGER(trigger\_id:1) calculate(measure\_value:221,operation:„>”,value:200) result:TRUE

RULE(rule\_id:1) GET ACTIONS

ACTION(action\_id:1) SENT TO ACTUATOR(actuator\_id:1), value: „SET 100 100 100”

A menedzser a szabály kiértékelését követően elküldte a bekapcsolásra vonatkozó üzenetet a végrehajtóhoz, aki azt végrehajtotta.

szerver backend újraimplementálása

A Diplomaterv keretén belül az új fejlesztőrendszer megismerésével és kipróbálásával töltöttem, miközben elkezdtem a saját, webes szerver megoldásom implementálását.

Utóbbi már használható, a legtöbb, tőle kívánt funkció működőképes. A szerveren képesek vagyunk a szükséges adminisztrációs feladatok elvégzésére, mint amilyen például az új eszközök, vagy azokhoz tartozó szabályok felvétele a rendszerbe. Működik továbbá a vizualizáció is, a szenzoradatokat a szerver felületén grafikus formában is megtekinthetjük. Az eddigiek nem jelentik azt, hogy a központi szerver fejlesztése befejezettnek tekinthető: a felület sok ponton esetlen és nem túl felhasználóbarát, a kényelmesebbé tételre még szeretnék időt fordítani a jövőben.

Az eszközök működési módjaik közül a szenzor áll a legközelebb a kész változathoz: ez funkcionálisát nézve közel teljesnek nevezhető, néhány apróbb javításon, illetve a hosszú tesztelést követő esetlegesen felmerülő feladatokon kívül tulajdonképpen komolyabb munka már remélhetőleg nem adódik vele.

A beavatkozók és lokális szerver működésének implementációja egyelőre készülő fázisban van: eddig elsősorban a rendszer elemei közötti kommunikáció megvalósításán dolgoztam, továbbá olyan megvalósítandó feladatokon, mint a lokális szerver hitelesítése a központi szerver felé.

## Kitekintés

A jövőre nézve az elsődleges feladat azon részfeladatok befejezése, mely még nem, vagy nem teljes mértékben készült el. Ez elsősorban a menedzser és a beavatkozók megvalósítását és az egész rendszer alapos, több eszközre kiterjedő tesztelését jelenti.

A szenzor üzemmódnál az időszakos működési igény miatt érdekes, és lényeges kérdés az, hogy megoldható-e (gazdaságosan és ésszerű karbantartási igénnyel) a csomópont elemmel való működtetése. Ennek kiderítéséhez szükségessé válik majd az eszköz fogyasztásának mérése (mind alvó, mind aktív üzemmódban), és az eredmények kiértékelése.

Ha a szoftver fejlesztési folyamatának a végére érünk, a következő lépés az lehet, hogy a már éles használathoz specializált hardvert tervezzünk az eszközhöz, mely(ek) már a valódi, nem tesztelési célú csomópontokban is működhetnek.

Távlatibb, de nem elhanyagolható lehetőség a szenzorhálózat csomópontjaiban heterogén hálózatok összekapcsolása, így esetleg nem csak IoT, hanem pl. Bluetooth vagy rádiófrekvenciás eszközöket is bevonhatnánk a rendszer működésébe. A lehetőségek vizsgálata önmagában is egy elég jelentős méretű feladat lehet a jövőben.

Az IoT projekt témájánál maradva ugyancsak érdekes feladat lehetne valamilyen okostelefonos alkalmazást tervezni a rendszerhez, amelyben további beavatkozásokat is végezhetnénk.

# Irodalomjegyzék

*Domoticz*. (dátum nélk.). Forrás: http://www.domoticz.com/

*ESP8266 Low Power Solutions.* (dátum nélk.). Forrás: https://espressif.com/sites/default/files/documentation/9b-esp8266-low\_power\_solutions\_en.pdf

1. Global Positioning System, műhold alapú navigációs rendszer, a szolgáltatás a világ minden pontjáról a nap 24 órájában ingyenesen elérhető. [↑](#footnote-ref-2)
2. Az IEEE 802.11 egy vezeték nélküli adatátviteli protokoll, mely a fizikai és az adatkapcsolati réteget definiálja. A b/g/n változatok a sávszélesség és a használt rádiós frekvenciában térnek el. [↑](#footnote-ref-3)
3. Wi-Fi Protected Access, protokollok a vezeték nélküli hálózatok biztonságossá tételére [↑](#footnote-ref-4)
4. Kis hatótávolságú, adatcseréhez használt, nyílt, vezetéknélküli szabvány [↑](#footnote-ref-5)
5. Universal asynchronous receiver/transmitter, aszinkron soros átvitelre használt eszköz [↑](#footnote-ref-6)
6. Inter-Integrated Circuit, kétvezetékes szinkron adatátviteli rendszer. [↑](#footnote-ref-7)
7. A bistabil relé egy olyan kapcsoló, mely csak a nyitott és zárt állapot közötti váltáshoz igényel energiát, az aktuális állapotot képes korlátlan ideig megőrizni [↑](#footnote-ref-8)
8. Magyarul ritkán szokták fénykibocsátó diódának is nevezni [↑](#footnote-ref-9)
9. Az Xtend egy általános célú magas szintű programozási nyelv, mely a Java nyelvből indul ki, kiegészítve azt extra funkciókkal, mint az opcionális típushasználat, objektumok dinamikus kiterjesztése vagy az operátor túlterhelés. [↑](#footnote-ref-10)
10. Network Attached Storage, azaz a hálózatra csatolt tároló. A NAS egy fájl szintű adattároló eszköz, amely a számítógépes hálózathoz csatlakoztatva lehetővé teszi távoli felhasználók között adatok menedzselését, hozzáférési jogosultságok kezelését. Internetkapcsolat segítségével bárhonnan lehet rá csatlakozni. [↑](#footnote-ref-11)
11. A Blocky egy vizuális fejlesztőeszköz, melyet a Google készített. Használói vezérlési szerkezeteket reprezentáló blokkok egymáshoz illesztésével tudnak algoritmusokat, és végső soron programokat írni. [↑](#footnote-ref-12)
12. Az asszociatív tömb (angolul map vagy dictionary) adatszerkezet kulcs-érték párokat tartalmaz, ahol egy kulcshoz pontosan egy érték tartozik. [↑](#footnote-ref-13)
13. Cascading Style Sheets, a számítástechnikában egy [stílusleíró nyelv](https://hu.wikipedia.org/w/index.php?title=St%C3%ADlusle%C3%ADr%C3%B3_nyelv&action=edit&redlink=1), mely a [HTML](https://hu.wikipedia.org/wiki/HTML) vagy [XHTML](https://hu.wikipedia.org/wiki/XHTML) típusú strukturált dokumentumok megjelenését írja le. [↑](#footnote-ref-14)
14. Real Time Operation System, olyan operációs rendszer architektúra, ahol az egyes eseménykezelőknek meghatározott időn belül kötelezően le kell futnia. [↑](#footnote-ref-15)
15. A FreeRTOS egy népszerű RTOS kernel beágyazott eszközökre írva [↑](#footnote-ref-16)
16. Simple Service Discovery Protocoll, egy hálózati protokoll hálózati szolgáltatások felderítésére és jelenlétének közzétételére [↑](#footnote-ref-17)
17. Real time clock, az eszköz belső órája, mely nem egyezik meg a rendszer órával, utóbbi egy szoftveres óra, mely az indítás óta eltelt időt jelzi [↑](#footnote-ref-18)
18. Delivery Traffic Indication Message, a kliens felé az AP-tól érkező üzenet, jelezve, hogy a kliens számára egy hálózati csomag kézbesítésre vár [↑](#footnote-ref-19)
19. Media Access Control, 12 hexadecimális számjegyből álló egyedi azonosító [↑](#footnote-ref-20)
20. Virtual private server, jellemzően nagy teljesítményű fizikai gépen létrehozott virtuális gép. AVPS egy nagyon népszerű szolgáltatás, mivel saját fizikai szerver üzemeltetéséhez képest viszonylag olcsón lehet saját, egyedi IP címmel rendelkező szerverhez jutni, melyen a fizikai géphez hasonlóan teljesen saját környezetet alakíthatnak ki a felhasználók [↑](#footnote-ref-21)