ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT

Okos otthon megvalósítása NodeRed keretrendszer segítségével

OE-AMK

2024

Hallgató neve: Jancsek József Márk

Hallgató törzskönyvi száma: T001026/FI12904/A-M

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Jancsek József Márk
Szakdolgozat száma: SZD2301230930591482[redacted]
Törzskönyvi száma: T001026/FI12904/A-M **Neptun kódja:** [redacted]

Szak: mérnökinformatikus
Specializáció: Felhő szolgáltatási technológiák és IT biztonság specializáció

A dolgozat címe: Okos otthon megvalósítása NodeRed keretrendszer segítségével
A dolgozat címe angolul: Node-RED Based Smart Home System
A feladat részletezése: A szakdolgozat célja egy okos otthon rendszer összeállítása a Node-RED vizuális programozóeszköz minél hatékonyabb használatával. A hallgató állítsa fel a rendszerrel szemben támasztott követelményeket, amik alapján válassza ki a megfelelő szenzorokat, opcionálisan beavatkozókat és Node-RED komponenseket. Valósítson meg egy működő okos otthon rendszert a kiválasztott komponensek integrálásával, kihasználva a Node-RED rugalmasságát. A rendszer részét képezze egy webes felhasználói felület is, melyen a felhasználó az okos otthonba integrált eszközök működéséről visszajelzést kaphat, illetve amelynek segítségével beavatkozásokat végezhet. A kész rendszert széleskörűen tesztelje.

Intézményi konzulens neve: Dr. Vakulya Gergely

A kiadott téma elévülési határideje:2026. 02. 10.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

ÓBUDAI EGYETEM Alba Regia Műszaki Kar Természettudományi és Szoftvertechnológiai Intézet P.H. 1. Székesfehérvár

Kiadva: Székesfehérvár, 2023. 10. 17.

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom: 2023. 12. 15.

.....
belső konzulens

.....
külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

HALLGATÓI NYILATKOZAT

Alulírott Jancsek József Márk hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Kelt: Sárszentmihály, 2024. május 10.

Jancsek József Márk

.....

hallgató aláírása

Tartalomjegyzék

1. Bevezetés 1
2. Specifikáció 3
3. Rendszerterv 4
3.1 Eszközök kiválasztása 4
3.1.1 Fejlesztőeszköz 4
3.1.2 Mikrokontroller 5
3.1.3 MQTT bróker 6
3.2 Node-RED bemutatása 7
3.2.1 Böngésző alapú flow szerkesztés 7
3.2.2 A Node.js-re épül 7
3.2.3 Közösségi fejlesztés 8
3.2.4 Fontosabb node-ok 8
3.3 Node-RED futtatása 10
3.4 Docker bemutatása 11
3.4.1 Hatékony és kiszámítható 11
3.4.2 Fejlesztés 11
3.4.3 Megosztás 11
3.4.4 Futtatás 12
3.5 MQTT bemutatása 12
3.5.1 MQTT kliens 13
3.5.2 MQTT bróker 14
4. Megvalósítás 15
4.1 HiveMQ MQTT bróker regisztráció 15

4.2 Tasmota bemutatása 17
4.3 Telepítés menete 17
4.4 MQTT csatlakoztatás 19
4.5 Node-RED elindítása 20
4.6 Lámpák integrálása 20
4.7 Hőmérséklet és páratartalom mérő 23
4.8 Kinti hőmérséklet és napszak használata 25
4.9 Mozgásérzékelő 27
4.10 Ajtónyitás érzékelő 30
4.11 Webes kezelőfelület 31
4.12 Szemléltető modell 32
5. Legújabb okosotthon szabvány 34
5.1 Matter szabvány 34
5.1.1 Megalkotása 34
5.1.2 A szabvány előnyei okosotthon viszonylatban 34
6. Összefoglaló 36
7. Summary 38
8. Irodalomjegyzék 40
9. Ábrajegyzék 42

1. BEVEZETÉS

A technológia folyamatosan fejlődő területén az innováció és az automatizálás, olyan rendszerek létrejöttét segítette elő, amelyek nagyban újradefiniálták a környezetünkkel való interakciókat. Ennek az élvonalába tartozik az okosotthon, mely egy olyan rendszer, ami túllép a lakóterek hagyományos határain. Ez a szakdolgozat az otthoni automatizálás világába vezet be minket, különös tekintettel a Node-RED által működtetett okosotthon rendszerekre, melyek számos lehetőséget biztosítanak az ilyen otthonok kialakítására.

A dolgozat központi problémáját a hagyományos okosotthon rendszerek korlátai adják. Ezek a rendszerek gyakran az alkalmazkodóképesség és a személyre szabott intelligencia hiányát mutatják. A korlátok egyre nyilvánvalóbbá válnak a felhasználók számára, akik nem csak kényelemre, hanem a technológia probléma mentes integrálására is törekszenek az otthonukba. A hagyományos rendszerek az előre meghatározott automatizálási folyamatok ellátásában nagyon jók, nem elég kifinomultak ahhoz, hogy dinamikusan reagáljanak a felhasználók és háztartások folyamatosan változó igényeire. Ez a hiányosság különösen az energiahatékonyság és a felhasználóközpontú élmények területén jelentkezik. Probléma lehet még, hogy a népszerű okosotthon eszközök használatához, amely például az Amazon Alexa vagy a Google Home, angol nyelvtudás szükséges, és habár az angol nyelv elterjedt az országunkban, sokan vannak, akik csak a saját anyanyelvüket beszélik, ezt a problémát is meg lehet oldani a Node-RED segítségével.

Az Én okosotthon rendszerem célja, hogy túllépje az otthon hagyományos értelmezését, mégpedig úgy, hogy személyre szabott intelligenciával ruházza fel azt. Egy olyan rendszert szeretnék létrehozni, amely alkalmazkodik a felhasználók egyedi igényeihez, és nem csak a rutinfeladatokat automatizálja.

Többek között a rendszer része lesz egy webes kezelőfelület is, amelyen a felhasználó láthatja az otthonában elhelyezett okos eszközöket, mint például az okos villanykörtéket, amelyeket, ha szeretne ki- és bekapcsolhat távolról. A használt szenzorok által kapott adatokat is itt fogom megjeleníteni.

A szakdolgozat úgy épül fel, hogy kibontakoztatja a Node-RED képességeinek rétegeit, a könnyen kezelhető vizuális programozási felülettől, a csomópontok és integrációk ro-

bosztus, bonyolult ökoszisztémájáig. Ennek az eszköznek a kihasználásával arra törekszem, hogy a lakástulajdonosok számára lehetővé tegyem a saját igényeikre szabott automatizálási folyamatok tervezését és telepítését. Valós alkalmazásokat és felhasználási eseteket fogunk vizsgálni, bemutatva az okosotthon rendszerek hatását az energiahatékonyságra és az általános életminőség javítására.

Végső soron a szakdolgozat azt mutatja be, hogy az okosotthon nem csak egy modern technológiai fejlesztés, hanem egy olyan rendszer, amely az emberek mindennapját könnyebbé teheti. A szakdolgozat áttekintése segít megérteni, hogy hogyan lehet egy ilyen rendszert hatékonyan integrálni az otthonunkba, kibővítve azt egy olyan intelligens környezettel, amely valóban az életminőség javítását célozza meg.

2. SPECIFIKÁCIÓ

A rendszerrel szemben számos igény merült fel felhasználói oldalról. Szeretné egy helyen látni és kezelni az otthona különböző pontjain lévő ledes világításokat, és nyilván ezek adott pillanatbeli állapotát, bekapcsoltságát megtekinteni. Néhány szenzor integrálására is szükség lesz, amelyek segítségével látható lesz a lakás aktuális hőmérséklete és a levegő páratartalma.

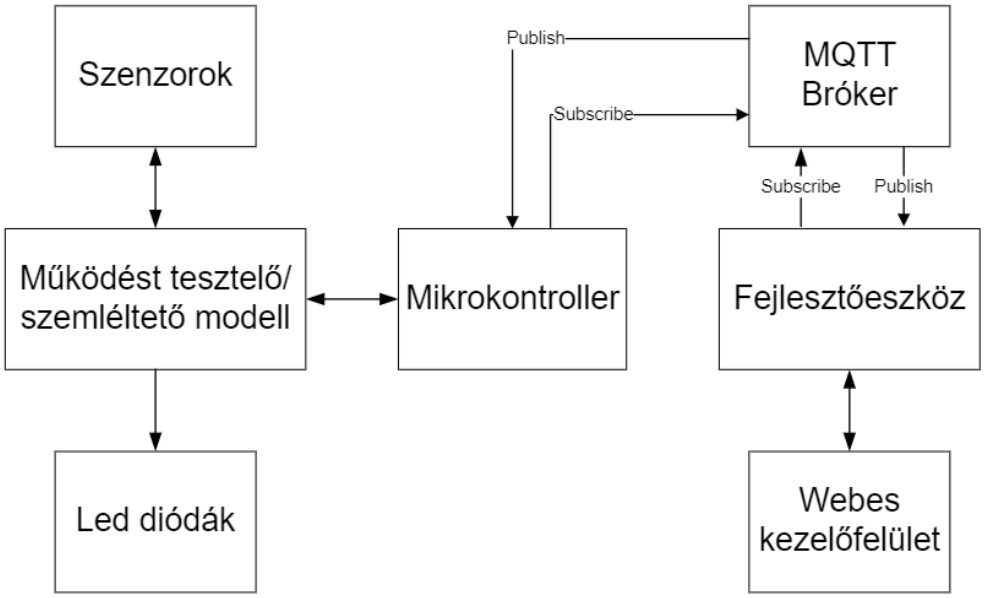
A hőmérséklet adat nagyon fontos lesz a lakás fűtése szempontjából, ami átmeneti időszakban 2 darab klímával működik, ugyanis a saját és a leendő felhasználó tapasztalatai alapján, sok esetben, ezeknek az eszközöknek a belső hőmérője pontatlan értéket mutat a gyártó alkalmazásában, és azt is általában pozitív irányban. Ezzel kiküszöbölhető az a kellemetlen meglepetés, hogy hazaérve több Celsius fokkal kevesebb a hőmérséklet, mint a kívánt, ezzel is növelve a lakók komfortérzetét. A klímák Node-RED-be történő integrálására nem kerül sor, ugyanis ez túlmutat a szakdolgozat határain. Nagyon kevés klíma gyártó néhány eszközének használatára van csak megoldás, és ezekhez is szükséges egy komplett vezérlőmodul gyártása, mert a gyárilag kapott WI-FI-s modullal nem lehet ezt megoldani.

Fontos szempont a költséghatékonyság, hiszen már maguk az okos eszközök sem kis összegbe kerülnek, így a teljes lakás velük történő felszerelése sem lesz olcsó, melléjük pedig jár az adott gyártó által készített szoftver, azért külön fizetni nem kell. Részben emiatt is lett a Node-RED választva, mert ez egy bárki számára ingyenesen elérhető szoftver.

Az energiahatékonyság sem egy utolsó szempont a kialakítás során. Lesznek olyan lámpák melyek mozgásérzékelő segítségével fognak működni, ezeket nyilván csak korlátozott látási viszonyok között szeretnénk használni, a nappal történő bekapcsolásuk energiapazarlás lenne.

3. RENDSZERTERV

A rendszer főbb komponenseinek a szemléltetésére egy blokkvázlatot készítettem, ezek közé tartozik a fejlesztőeszköz, a mikrokontroller, az MQTT bróker, egy webes kezelőfelület és az elmaradhatatlan szemléltető modell a rajta található szenzorokkal és led diódákkal. Ez az alábbi ábrán látható. (1. ábra)



1. ábra Blokkvázlat [Saját kép]

3.1 Eszközök kiválasztása

A rendszerterv következő lépése a megvalósításhoz szükséges eszközök számbavétele és közülük a célnak legmegfelelőbb kiválasztása.

3.1.1 Fejlesztőeszköz

A tervezés első és egyben az egyik legfontosabb lépése a fejlesztőezköz meghatározása. Körbenézve a piacon számos programot találhatunk erre a feladatra, ezek nagyrésze viszonylag kötött feladatok elvégzésére alkalmas, de vannak köztük, amelyek nagyobb szabadságot nyújtanak a felhasználó számára. Kettő programra szűkítettem le a kört, ezek voltak, amelyek alkalmasnak tűntek a feladatra. Az egyik program az OpenHAB, a másik pedig a Node-RED. Mind a kettő nagy szabadságot ad a fejlesztés során, lehetővé teszik teljesen egyedi automatizáció létrehozását is.

Az OpenHAB mellett szól, hogy ezt kifejezetten házautomatizációhoz találták ki, így már rendelkezik számos beépített funkcióval. Nyílt forráskódú és aktív közösséggel rendelkezik, ez hozzájárul a rendszer fejlődéséhez. Ezekkel szemben viszont kisebb hangsúlyt fektet a vizuális programozásra, mint a Node-RED, így a tanulási folyamat némileg hosszabb. [1]

A Node-RED nagy előnye a vizuális programozás, amely megkönnyíti az új felhasználók dolgát, de természetesen itt is szükség van programozási tudásra. Nagyon sok különböző eszközt és rendszert támogat, így könnyen integrálható más platformokkal. Ennek is legalább akkora, ha nem nagyobb az aktív felhasználó bázisa, mint az OpenHAB-nek, akik örömmel segítenek az esetlegesen felmerülő problémák megoldásában különböző platformokon, fórumokon keresztül. [2]

Mindezeket összevetve, a vizuális programozás és az ebből következő gyorsabb és egyszerűbb tanulási folyamat miatt a Node-RED-et választottam az okosotthon fejlesztőeszközenként.

3.1.2 Mikrokontroller

Ahhoz, hogy a fejlesztőkörnyezetben megírt szoftvert alkalmazni tudjuk a gyakorlatban, szükségünk lesz egy mikrokontrollerre. Itt a sokak által már ismert Raspberry Pi és az ESP-32 merült fel, mint lehetséges eszközök.

A Raspberry Pi egy egykártyás számítógép, amely számos célra alkalmazható, mint például programozás tanulásra, alapvető számítógépes alkalmazások futtatására és barkácsprojektek készítésére. Rendelkezik processzorral, memóriával, USB portokkal, és HDMI porttal. Használható önálló számítógépként vagy egy nagyobb projektben például egy okosotthon automatizálási rendszer részeként. Összességében ez egy nagy teljesítményű és rugalmas platform, amelyet úgy terveztek, hogy viszonylag megfizethető legyen, de az újabb és újabb modellek megjelenésével az áruk igencsak megemelkedett. Az általam használni kívánt Node-RED programot is képes lenne futtatni. [3]

Ezzel szemben az ESP32 egy magasan integrált, alacsony fogyasztású, egymagos WI-FI mikrokontroller, amelyet úgy terveztek, hogy biztonságos és költséghatékony legyen, mindemellett nagy teljesítménnyel és gazdag input-output képességekkel rendelkezzen.

Található benne egy beépített antenna, teljesítményerősítő, alacsony zajszintű vételi erősítő, szűrők és energiagazdálkodási modulok. Ezek miatt hihetetlen mértékű funkcionalitást biztosít különböző alkalmazások számára, minimális nyomtatott áramköri lap (PCB - Printed Circuit Board) igény mellett. Képes teljesen önálló rendszerként működni, de akár egy host szolga eszközeként is megállja a helyét, képes más rendszerekkel is összekapcsolódni. [4]

Alaposan tanulmányozva a két eszközt, arra az elhatározásra jutottam, hogy egy Raspberry Pi erőforrásai túl vannak méretezve a feladat igényeihez képest, emellett pedig jóval drágább, mint egy ESP-32. Mivel az okosotthon rendszer kialakításának egyik szempontja a költséghatékonyság, ezért egy ESP-32 használata mellett döntöttem.

3.1.3 MQTT bróker

A rendszerbe integrált eszközöknek valamilyen úton kommunikálniuk kell egymással. Erre nyújt megoldást az MQTT, amelyet kifejezetten erre a célra fejlesztettek ki. A kommunikáció megvalósításához szükséges egy úgynevezett MQTT bróker, amelyre a kliensek feliratkozva megkapják a nekik szükséges üzeneteket, információkat. Egy ilyen bróker megvalósításánál van lehetőségünk egy saját eszközön futtatni a brókert, vagy használhatunk egy ilyen szolgáltató által kifejlesztett felhő alapú megoldást is. Nagyon sok népszerű megoldás van a piacon, amelyek között vannak fizetősek és ingyenesek is. A költséghatékonyság szempontjából kizárólag olyanokat vettem számításba, amelyek ingyenesek, vagy pedig van olyan részük amelyik ingyenesen használható. A két felmerülő szolgáltató az Eclipse Mosquitto és a HiveMQ.

Az Eclipse Mosquitto egy olyan MQTT szerver implementáció melyet mi magunk telepítünk a saját eszközünkre. A használható eszközök skálája széles, a nagy teljesítményű gépektől, az alacsonyabb teljesítményű beágyazott eszközökig terjed, így jó választás lehetne egy alacsony költségvetésű okosotthon megvalósításhoz is. A jelenlegi verziójának futtatható állománya 120kB nagyságrendű, és nagyjából 3MB RAM szükséges a futtatásához. Mindezek mellett akár 1000 darab klienst is csatlakoztathatunk hozzá. A kliensalkalmazásoktól való kapcsolatfelvételen kívül rendelkezik egy híddal, amely lehetővé teszi a felhasználók számára több MQTT szerver összekapcsolását, ezzel szerver hálózatok kiépítését. [5]

A HiveMQ szintén egy olyan MQTT platform, mely gyors, hatékony és megbízható adatmozgást biztosít a csatlakoztatott eszközök számára. Igény szerint nagy mértékben skálázható, emellett vállalati szintű biztonsági koncepciók szem előtt tartásával építették fel. Teljes mértékben megvalósítja az MQTT protokollt, ennek révén világszerte vezető szerepet tölt be a protokollt használó okos eszközök használói között. [6]

A szakdolgozat során a HiveMQ használatára esett a választás. Az Eclipse Mosquitto-val szemben ez futtatható a gyártó saját felhő alapú kiszolgálóján, így ezt nem nekem kell telepítenem és a leendő felhasználónak sem kell folyamatosan felügyelni a megfelelő működését. Költséghatékonyság szempontjából sem utolsó ez a választás, ugyanis így nincs szükség a szerver futtatásához egy külön eszköz vagy számítógép beszerzésére és működtetésére, ezzel elektromos áramot, és végső soron pénzt takarítunk meg.

3.2 Node-RED bemutatása

A Node-RED egy olyan programozóeszköz, amellyel hardvereszközök, API-k, online szolgáltatások új és érdekes módon kapcsolhatók össze. Olyan böngészőalapú szerkesztőt biztosít, amely megkönnyíti a folyamatok összekötését a palettán található csomópontok széles választékának felhasználásával, amelyek egyetlen kattintással telepíthetők futási időben. [2]

3.2.1 Böngésző alapú flow szerkesztés

A Node-RED böngészőalapú flow szerkesztőt biztosít, amely megkönnyíti a flow-k összekapcsolását a paletta csomópontjainak széles választékával. A flowk ezután egyetlen kattintással telepíthetők futásidőben. A szerkesztőn belül JavaScript függvények hozhatók létre részletes szövegszerkesztővel. A beépített könyvtár pedig lehetővé teszi, hogy hasznos függvényeket, sablonokat vagy flowkat mentsünk el újrafelhasználás céljából. [2]

3.2.2 A Node.js-re épül

A light-weight futásidő a Node.js-re épül, teljes mértékben kihasználva annak eseményvezérelt, nem blokkoló modelljét. Ezáltal ideális alacsony költségű hardvereken, például Raspberry Pi-n, valamint felhőben történő futtatásra. A Node-RED csomagtárában található több mint 225 000 modulnak köszönhetően a palettás csomópontok köre könnyen bővíthető új képességekkel. [2]

3.2.3 Közösségi fejlesztés

A Node-RED-ben létrehozott folyamatok JSON segítségével tárolódnak, amelyek könnyen importálhatók és exportálhatók másokkal való megosztás céljából. Egy online flow könyvtár lehetővé teszi, hogy a legjobb flow-kat megosszuk másokkal. [2]

3.2.4 Fontosabb node-ok

A következőkben ismertetem nagyvonalakban a Node-RED-ben található fontosabb csomópontokat, melyek fontos szerepet töltenek be az okosotthon elkészítésében. Ezek név szerint az Inject, a Debug, a Function, a Change, a Switch és végül a Template. [7]

- *Inject*

Az Inject csomópont gombjára kattintva a szerkesztőben, manuálisan indíthatunk el egy üzenetet. Az általa küldött üzenet payload-ja és a téma tulajdonságai is beállíthatók.

A payload többféle típusra is beállítható:

- egy flow vagy globális kontextus mező értéke
- karakterlánc, azaz string, szám, logikai érték, puffer vagy objektum
- időbélyeg, amely egy milliszekundumokban kifejezett érték

A csomópont arra is használható, hogy rendszeres időközönként automatikusan indítsa el a küldeni kívánt üzenetet.

- *Debug*

A Debug csomópont segítségével üzenteket jeleníthetünk meg a szerkesztőn belül lévő Debug oldalsávban. Ez az oldalsáv strukturáltan jeleníti meg az üzeneteket, megkönnyítve így az üzenetek átlátását, ellenőrzését. Az üzenetek mellett információt tartalmaz arról, hogy mikor érkeztek az egyes üzenetek, és melyik debug csomópont küldte azokat. A forráscsomópont azonosítójára kattintva a munkaterületen belül megjelenik az adott csomópont.

A csomóponton található gombbal engedélyezhető vagy letiltható a kimenete. Ajánlott a nem használt debug csomópontokat letiltani vagy eltávolítani.

A csomópont úgy is beállítható, hogy az összes üzenetet a futásidejű naplóba küldje, vagy rövid (32 karakteres) üzeneteket küldjön a debug csomópont alatti státuszszövegbe.

- *Function*

A Function csomópont lehetővé teszi JavaScript kódok futtatását a rajta keresztül küldött üzeneteken. Az üzenet egy msg nevű objektumként kerül átadásra, lesz egy msg.payload mezője, amely az üzenet szövegét fogja tartalmazni. Más csomópontok a saját mezőiket hozzáadhatják az üzenetekhez, ebben az esetben a saját dokumentációjukban rögzíteni kell ezt.

- *Change*

A Change csomópont használható az üzenetek mezőinek módosítására, és a kontextus mezők beállítására anélkül, hogy a Function csomópontot kellene használnunk. Minden csomópont több művelet végrehajtására is konfigurálható, melyek sorrendben kerülnek végrehajtásra.

A rendelkezésre álló műveletek a következők:

- Set: Egy mező beállítása. Az érték különböző típusú lehet, de akár át is vehető egy meglévő üzenet- vagy kontextus mező.
- Change: Az üzenet egyes mezőinek keresése és helyettesítése.
- Move: Egy mező áthelyezése vagy átnevezése.
- Delete: Egy mező törlése.

Egy mező beállításakor az érték lehet egy JSONata kifejezés eredménye is. A JSONata egy deklaratív lekérdezési nyelv a JSON adatokhoz.

- *Switch*

A Switch csomópont lehetővé teszi az üzenetek továbbítását a flow különböző ágaiba azáltal, hogy minden egyes üzenethez egy szabálykészletet értékel ki. A switch elnevezés a sok programozási nyelvben elterjedt switch utasításból származik. Ez nem egy fizikai kapcsolóra való utalás, többszörös elágazást jelent. A csomópontot a tesztelendő mező értékével konfiguráljuk, amely lehet üzenet- vagy kontextus mező.

Négyféle szabály létezik:

- Value: Az értékszabályok a konfigurált érték alapján kerülnek kiértékelésre.
- Sequence: A szekvencia szabályok üzenetsorozatokra használhatók, például a Split csomópont által generáltakra.

- JSONata Expression: Megadható egy JSONata kifejezés, amely az egész üzenetre kiértékelésre kerül, és akkor egyezik meg, ha a kiértékelés igaz értéket ad vissza.
- Otherwise: Ez a szabály használható akkor, ha az előző szabályok egyike sem felelt meg.

A csomópont az üzeneteket az összes olyan kimenetre továbbítja, amelyek megfelelnek az egyező szabályoknak. De úgy is beállítható, hogy leállítsa a szabályok kiértékelését, ha talál egy egyezőt.

- *Template*

A Template csomópont segítségével az üzenetek mezőinek felhasználásával lehet szöveget generálni egy sablon kitöltéséhez. Az eredmény létrehozásához a Mustache nyelvet használja. A csomópont a sablon eredményével beállítja a konfigurált üzenet vagy kontextus mezőjét. Ha a sablon érvényes JSON vagy YAML tartalmat generál, akkor beállítható, hogy az eredményt a megfelelő JavaScript objektumba tegye.

3.3 Node-RED futtatása

Az alkalmazás használatára több lehetőségünk is van. Telepíthetjük magára a mikrokontrollerünkre, de ez a verzió az ESP-32 használata és a költségcsökkentés miatt nem áll rendelkezésünkre. Futtathatjuk akár felhő szolgáltatónál is, ez azonban a folyamatos működés és adatmozgás miatt hamar plusz költséget jelenthet, hiszen ezen szolgáltatók ingyenes használata korlátozott. Marad tehát két lehetőségünk, az egyik a natív futtatás, a másik pedig a Docker használata.

A helyi futtatáshoz nem elég pusztán csak egy Node-RED telepítőt leszedni a fejlesztő weboldaláról, ennyire nem egyszerű a menete. Mindenekelőtt a node.js-t kell telepítenünk, amelyre maga a Node-RED épül, ennek a telepítőjével minden más is feltelepül amire szükségünk lehet a Node-RED működéséhez. Ezek után következhet csak az alkalmazásunk telepítése, ezt viszont csak parancssorból végezhetjük el, és később majd szintén parancssorból tudjuk elindítani és leállítani. [8]

A Dockerből történő futtatása ennél egyszerűbb, itt magát a Docker alkalmazást kell feltelepítenünk elsőnek. Ezután a Node-RED oldaláról kimásolt paranccsal egyszerűen feltelepül az alkalmazás legújabb verziója. Az elindításhoz és leállításhoz itt is használhatunk parancssort, de itt van egy dedikált gombunk, amellyel ezt megtehetjük. [9]

3.4 Docker bemutatása

A Docker egy úttörő konténerizációs platform, amely az alkalmazások fejlesztésének, telepítésének és kezelésének forradalmasításában kulcsfontosságú szerepet játszik. Lényege, hogy a szoftvereket és függőségeiket hordozható egységekbe, úgynevezett konténerekbe foglalja, így konzisztens és reprodukálható környezetet biztosít a különböző számítástechnikai környezetekben. Egyszerűvé teszi az alkalmazások skálázását és kezelését, hordozhatóságát. [10]

3.4.1 Hatékony és kiszámítható

A Docker megszünteti az ismétlődő, hétköznapi konfigurációs feladatokat, és a fejlesztés teljes életciklusa során használható a gyors, egyszerű és hordozható alkalmazásfejlesztéshez. Átfogó végponttól végpontig terjedő platformja a felhasználói felületeket, CLI-ket, API-kat és biztonságot tartalmaz, amelyeket úgy terveztek, hogy az alkalmazásfejlesztés teljes életciklusa alatt együttműködjenek. [10]

3.4.2 Fejlesztés

A Docker image-ek kihasználásával előnyt szerezhetünk a kódolásban, hogy hatékonyan fejleszthessük saját alkalmazásainkat Windows és Mac rendszereken. Docker Compose segítségével több konténerből álló alkalmazást hozhatunk létre, integrálhatjuk kedvenc eszközeinket a teljes fejlesztésbe. A Docker az összes használt fejlesztési eszközzel működik, beleértve a VS Code-ot, A CircleCI-t és a GitHub-ot. Csomagolhatjuk az alkalmazásokat hordozható konténerképekbe, hogy bármilyen környezetben következetesen futtathassuk, helyben lévő Kubernetes-től az AWS ECS-ig, Azzure ACI-ig, Google GKE-ig és így tovább. [10]

3.4.3 Megosztás

Használhatjuk a Docker megbízható tartalmait, beleértve a Docker hivatalos image-eit és a Docker ellenőrzött kiadók image-eit a Docker Hub tárolóból. Dolgozhatunk együtt csapattagokkal és más fejlesztőkkel, a Docker Hub-on történő egyszerű image megosztással. Személyre szabhatjuk a fejlesztők hozzáférését az image-ekhez a szerepkörökön alapuló hozzáférés-szabályozással, és betekintést nyerhetünk a tevékenységek előzményeibe a Docker Hub Audit Logs segítségével. [10]

3.4.4 Futtatás

Többféle alkalmazást is gond nélkül futtathatunk az összes környezetünkben, beleértve a tervezési, tesztelési, staging és termelési környezetet. Telepíthetjük alkalmazásainkat külön konténerekbe, egymástól függetlenül és különböző nyelveken. Csökkenthetjük a nyelvek, könyvtárak vagy keretrendszerek közötti konfliktusok kockázatát. Felgyorsíthatjuk a fejlesztést a Docker Compose CLI egyszerűségével és egyetlen paranccsal indíthatjuk el alkalmazásainkat helyben és a felhőben. [10]

3.5 MQTT bemutatása

A következőkben mélyebbre hatóan bemutatásra kerül az okosotthon minden eszköze által kommunikációhoz használt MQTT protokoll. [11]

Az MQTT a Message Queue Telemetry Transport mozaikszava. Az MQTT egy ügyfél-kiszolgálói publish/subscribe (publikáló/feliratkozó) üzenetküldő protokoll. Lightweight (könnyű), nyílt, egyszerű és úgy tervezték, hogy könnyen megvalósítható legyen. Ezek a tulajdonságok számos helyzetben ideálisan használhatók, beleértve a korlátozott környezeteket is, például a gép-gép (M2M) és a dolgok internete (IoT) kontextusokban történő kommunikációhoz, ahol kis kódlábnyomra van szükség és/vagy a hálózati sávszélesség korlátozott.

Amikor az MQTT-t „lightweight” -ként írják le, az azt jelenti, hogy egyszerű, hatékony és nem erőforrás-igényes. Az MQTT-t azzal a céllal hozták létre, hogy kis mennyiségű adatot küldjenek megbízhatatlan, korlátozott sávszélességű és csatlakozási képességű hálózatokon keresztül. Más protokollokhoz képest az MQTT kis kódlábnyommal, alacsony overhead-el és alacsony energiafogyasztással rendelkezik. Minimális csomagterhelése miatt, az olyan protokollokhoz képest, mint a HTTP, kiemelkedik a vezetékes adatátvitel során. Ez is ideálissá teszi a korlátozott feldolgozási teljesítménnyel, memóriával és akkumulátor-élettartammal rendelkező eszközökben, például érzékelőkben és más IoT-eszközökben való használatra.

Az MQTT bináris üzenetformátumot használ az ügyfelek és a brókerek közötti kommunikációhoz. Ez ellentétben áll más, szövegalapú formátumokat használó protokollokkal, mint például a HTTP vagy az SMTP. Fejléc formátuma az alábbi képen látható. (2. ábra)

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

2. ábra MQTT fix fejléc formátum példa [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html]

Az MQTT által használt bináris formátumot úgy tervezték, hogy csökkentse az üzenetek méretét és növelje a kommunikáció hatékonyságát. A bináris formátum használatával a protokoll minimalizálni tudja a továbbítandó adatok mennyiségét, és csökkenti az üzenetek értelmezéséhez szükséges feldolgozási teljesítményt. Ezáltal az MQTT jól alkalmazható alacsony sávszélességű vagy alacsony energiafelhasználású környezetben, például korlátozott erőforrásokkal rendelkező IoT-eszközökön. Vállalati rendszerekben is használják, ahol valós idejű adatkommunikációra van szükség.

A protokoll másik fontos aspektusa, hogy az MQTT rendkívül könnyen megvalósítható az ügyfél oldalon. A fejlesztése során a könnyű kezelhetőség volt az egyik fő szempont. Az MQTT-t széles körben használják az Iot, az ipari IoT (IIoT) és az M2M alkalmazásokban.

Erre egy tökéletes példa az okosotthonoknál történő felhasználása: Az okosotthonba telepített különböző eszközök, mint például intelligens termosztátok, villanykörték, biztonsági kamerák csatlakoztatására használják. Lehetővé teszi a felhasználók számára, hogy az otthoni eszközeiket távolról, egy weboldal vagy mobil alkalmazás segítségével vezéreljék.

3.5.1 MQTT kliens

Mind a kiadók, mind az előfizetők MQTT kliensek. A kiadó és az előfizető címkék arra utalnak, hogy a kliens éppen üzeneteket tesz közzé, vagy feliratkozott-e üzenetek fogadására (a közzétételi és feliratkozási funkciókat ugyanabban az MQTT kliensben is meg lehet valósítani). MQTT kliens lehet bármely eszköz, a mikrovezérlőtől a teljes értékű kiszolgálóig, amely MQTT-könyvtárat futtat, és hálózaton keresztül csatlakozik MQTT brókerhez. Az MQTT kliens lehet például egy nagyon kicsi, erőforrás-korlátozott eszköz, amely vezeték nélküli hálózaton keresztül csatlakozik, és egy csupasz minimális könyvtárral rendelkezik. Lehet akár egy átlagos számítógép is, amelyen tesztelési céllal grafikus MQTT kliens fut. Alapvetően minden olyan eszköz, amely TCP/IP stack-en keresztül MQTT-t beszél, MQTT kliensnek nevezhető. [11]

3.5.2 MQTT bróker

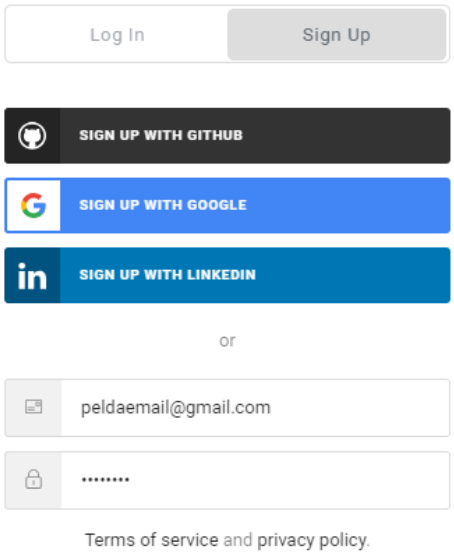
Az MQTT szerver megfelelője az MQTT bróker. A bróker áll minden publish/subscribe protokoll középpontjában. A megvalósítástól függően a bróker akár több millió egyidejűleg csatlakozó MQTT klienst is képes kezelni. A bróker felelős az összes üzenet fogadásáért, az üzenetek szűréséért, az egyes üzenetek feliratkozóinak meghatározásáért és az üzenet elküldéséért ezeknek a feliratkozott klienseknek. A bróker tárolja az összes olyan kliens munkamenet adatait is, amelynek tartós munkamenete van, beleértve a feliratkozásokat és az elmulasztott üzeneteket is. A bróker másik feladata a kliensek hitelesítése és engedélyezése. A bróker általában bővíthető, ami megkönnyíti az egyéni hitelesítést, engedélyezést és a backend-rendszerekbe való integrációt. Az integráció különösen fontos, mivel a bróker gyakran az a komponens, amely közvetlenül az internetre van kitéve, sok klienst kezel, és üzeneteket kell továbbítania a későbbi elemző és feldolgozó rendszereknek. Röviden, a bróker a központi csomópont, amelyen minden üzenetnek át kell haladnia. Ezért fontos, hogy a bróker nagymértékben skálázható, integrálható legyen a backend rendszerekbe, könnyen felügyelhető és (természetesen) hibaálló legyen. [11]

4. MEGVALÓSÍTÁS

4.1 HiveMQ MQTT bróker regisztráció

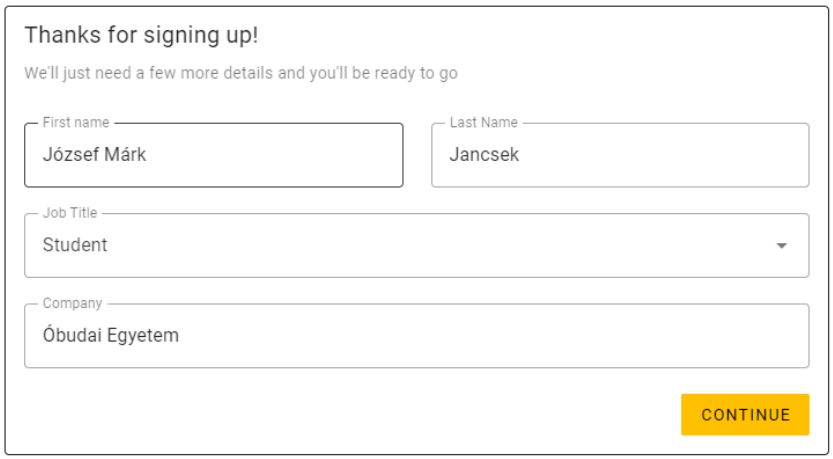
A regisztrációt az alábbi linken elérhető weboldalon lehet elkezdeni: https://auth.hivemq.cloud/login?state=hKFo2SBwaEJ5M3E0S1NOa2hGaXRLblF-KYzYxMjZaRVZvd3BGcqFupWxvZ2luo3RpZNkgNmd3SUROWmlvak-dEb3Z1aU5ZUEhXQTFLNGpEY0RzWmKjY2lk2SBJYWpvNGUzMmp4d1VzOEF-kRnhneFFuMlZQM1l3SVpUSw&client=Iajo4e32jxwUs8Ad-FxgxQn2VP3YwIZTK&protocol=oauth2&audience=hivemq-cloud-api&redi-rect_uri=https%3A%2F%2Fconsole.hivemq.cloud&scope=openid%20pro-file%20email&response_type=code&response_mode=qu-ery&nonce=Mks0NDM0cFpiUDdKdEFpalVIVU5Zc0FhaWlTX3l1Z3F2clZfNk-pkRFY0Mw%3D%3D&code_challenge=SrtzhHkQj52cZwT0H81QS1z7FkXtSaN-MHawEW3lAr5E&code_challenge_method=S256&auth0Client=ey-JuYW1lIjoiYXV0aDAtc3BhLWpzIiwidmVyc2lvbiI6IjEuMTMuNiJ9

A regisztráció nagyon egyszerű, egy email cím jelszó páros begépelése után a regisztráció gombra kattintva, már kapjuk is a megerősítő emailt. (3. ábra)



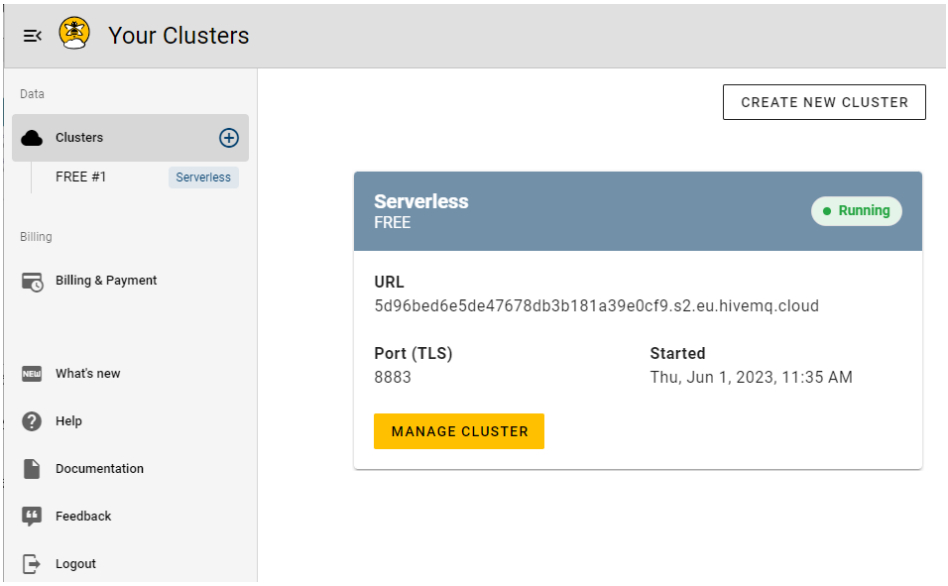
3. ábra Regisztrációs ablak [https://auth.hivemq.cloud/login?state=hKFo2SBwaEJ5M3E0S1NOa2hGaXRLblF-KYzYxMjZaRVZvd3BGcqFupWxvZ2luo3RpZNkgNmd3SUROWmlvakdEb3Z1aU5ZUEhXQTFLNG-pEY0RzWmKjY2lk2SBJYWpvNGUzMmp4d1VzOEFkRnhneFFuMlZQM1l3SVpUSw&client=Iajo4e32jxwUs8Ad-FxgxQn2VP3YwIZTK&protocol=oauth2&audience=hivemq-cloud-api&redi-rect_uri=https%3A%2F%2Fconsole.hivemq.cloud&scope=openid%20profile%20email&response_type=code&res-ponse_mode=query&nonce=Mks0NDM0cFpiUDdKdEFpalVIVU5Zc0FhaWlTX3l1Z3F2clZfNk-pkRFY0Mw%3D%3D&code_challenge=SrtzhHkQj52cZwT0H81QS1z7FkXtSaNMHawEW3lAr5E&code_chal-lenge_method=S256&auth0Client=eyJuYW1lIjoiYXV0aDAtc3BhLWpzIiwidmVyc2lvbiI6IjEuMTMuNiJ9]

A megerősítő emailben a „Confirm my account” gombra kattintás után megnyílik egy weboldal, ahol véglegesíthetjük a regisztrációnkat, nevünk, munkahelyünk, és beosztásunk megadásával. (4. ábra)



4. ábra Regisztráció véglegesítése [Saját kép]

Ezek után elénk tárul a HiveMQ webes kezelőfelülete. (5. ábra)



5. ábra HiveMQ webes kezelőfelülete [https://console.hivemq.cloud/]

Ahogyan látható, egy ingyenes verziót kapunk első körben, ezt természetesen van lehetőségünk módosítani igényeinknek megfelelően. Az ingyenes verziónak persze vannak korlátai. 100 darab IoT eszközt csatlakoztathatunk egyidejűleg, 10 GB adatforgalmunk lehet egy hónapon belül, 3 napos adatmegőrzési időnk van, és az üzeneteink nem lehetnek nagyobbak 5 MB-nál.

4.2 Tasmota bemutatása

Néhány sorban ismertetem a rendszer egyik fő elemét, a mikrokontrolleren futó alkalmazást, mely vezérli majd annak működését. [12]

A Tasmota egy nyílt forráskódú firmware az Espressif ESP8266, ESP32, ESP32S vagy az ESP32-C3 chipset alapú eszközökhöz, amelyet Theo Arends készített. 2016 január 25-én kezdődött a projektje, célja az volt, hogy az előbb említett ESP8266 alapú ITEAD Sonoff eszközöket MQTT-vel és „Over the Air" vagy OTA firmwarrel lássa el. Ami egy egyszerű projektnek indult, hogy egy felhőhöz kötött Sonoff Basic eszközt helyileg vezérelhetővé tegyen, az mára egy teljes értékű ökoszisztémává nőtte ki magát.

A rendszer nagyon hasznos tud lenni számos esetben, például, amikor egy eszközünk eredetileg zárt rendszerrel rendelkezik és korlátozott funkcionalitást nyújt. A Tasmota segítségével ezt az eszközt átalakíthatjuk, hogy szélesebb körű testreszabási lehetőséget biztosítson. Ez a firmware támogatja az eszközök távoli vezérlését, ellenőrzését.

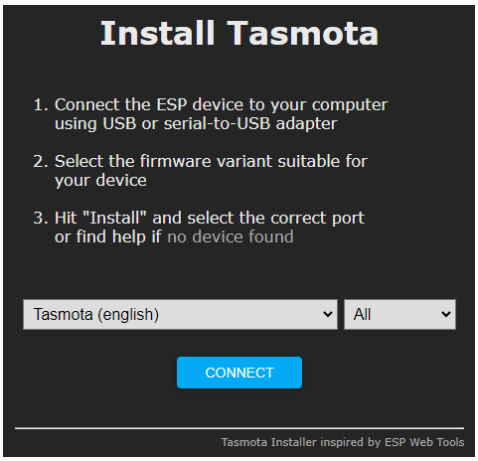
Nem elhanyagolható tény, hogy a rendszer kezelőfelülete könnyen átlátható, így rövid időn belül elsajátítható a használata, nem véletlen, hogy nagy népszerűségnek örvend.

A Tasmota-t egy ESP32S eszközre telepítve fogom használni, a Node-RED-ben megírt programom tesztelésére.

4.3 Telepítés menete

A folyamat megkezdése előtt szükségünk lesz a számítógépünkre egy soros port emulátorra, ami segítségével a számítógépünk fel fogja ismerni az usb-n keresztül csatlakoztatott eszközünket. Ezt követően egy micro usb kábel segítségével csatlakoztatjuk eszközünket a számítógéphez, fontos, hogy az usb kábel ne csak töltésre, hanem adatátvitelre is alkalmas legyen, különben nem fogjuk tudni véghez vinni a telepítést. Ha minden rendben van, az ESP32-n lévő kis piros led világítani kezd, ez mutatja nekünk, hogy az eszköz készen áll a telepítésre.

A telepítéshez Én személy szerint a Tasmota saját böngésző alapú telepítőjét alkalmaztam, számomra ez tűnt a legegyszerűbb megoldásnak. Ezt a telepítőt a Tasmota weboldaláról lehet elérni. Az oldalon nincs túlgondolva, az alábbi képen látható minden eleme. (6. ábra)



6. ábra Telepítő oldal [https://tasmota.github.io/install/]

Itt a legördülő menüben kiválaszthatjuk a számunkra megfelelő nyelvet, a mellette lévőben pedig specifikálhatjuk az eszközünket, de erre nem feltétlen van szükség, a számunkra megfelelő eszközhöz tartozó program fog felkerülni.

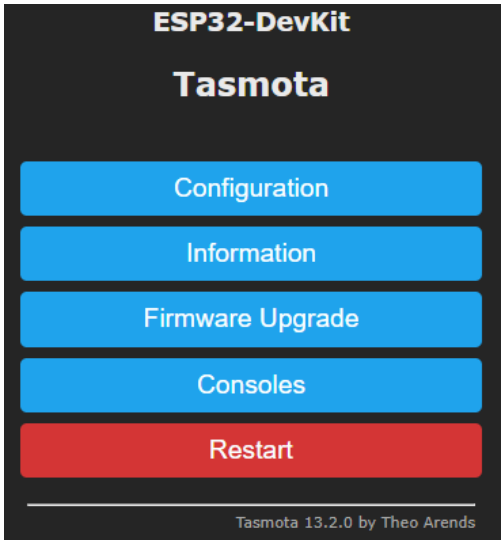
A connect gombra való kattintás után csatlakozunk az eszközhöz, és itt lehetőségünk van kiválasztani, hogy mit is szeretnénk vele csinálni. (7. ábra)

7. ábra Csatlakoztatott eszköz telepítésének megkezdése [Saját kép]

Alap esetben a visit device nem jelenik meg, itt azért látható mert, Én már telepítettem a Tasmota-t az ESP32-re. Az install pontot választva egy újabb ablakot kapunk amelyben eldönthetjük, hogy szeretnénk-e letörölni mindent ami az eszközön van, vagy csak simán

telepítsük fel a programot. A telepítés folyamatát csak úgy tudjuk elkezdeni, ha a weboldalon megjelenő install gomb mellé nyomva tartjuk az ESP32-n található reset gombot.

A telepítés végeztével rögtön lehetőséget is kapunk az eszközünk wifi hálózatunkhoz való csatlakoztatására, amely fontos az eszköz későbbi elérésének szempontjából. Ezek után a webes kezelőfelületét is megtekinthetjük, ezt egyszerűen megtehetjük, csak az eszköz ip címét kell a böngészőnk címsorába begépelni és máris elénk tárul az alábbi ablak. (8. ábra)



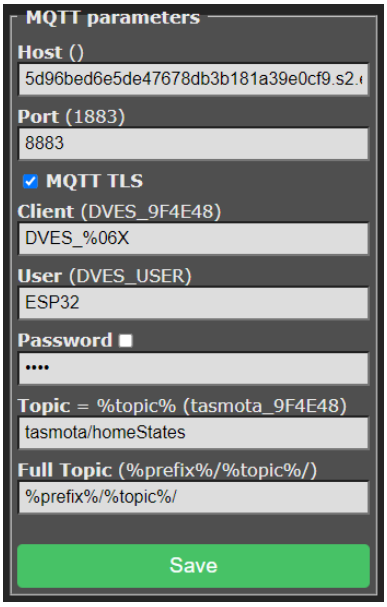
8. ábra ESP32-n futó Tasmota menüje [Saját kép]

Mint látható, valóban nagyon egyszerű és átlátható a kezelőfelület, itt egyből elérjük az eszközünk információit, frissíthetjük a rajta futó szoftvert, de akár újra is indíthatjuk. A Configuration menüpont lesz számunkra a legfontosabb, ezen belül szabhatjuk testre az eszköz működését.

4.4 MQTT csatlakoztatás

Következő lépésben a Configuration menüpont alatt található Configure MQTT pontnál ki kell tölteni a kapcsolathoz szükséges adatokat. (9. ábra) Szükségünk van a szerver URL-jére valamint port számára (lásd: 5-ös ábra). Ezek mellett meg kell adnunk egy felhasználó és jelszó párost, és egy topicot. Használhatjuk az alapértelmezetten létrejövő felhasználót és jelszót, melyet az MQTT brókerünk szolgáltat, vagy adhatunk meg sajátot

is. A topic nevét célszerű személyre szabni, a több rendszer esetén történő eligazodás végett.



9. ábra MQTT konfigurálás [Saját kép]

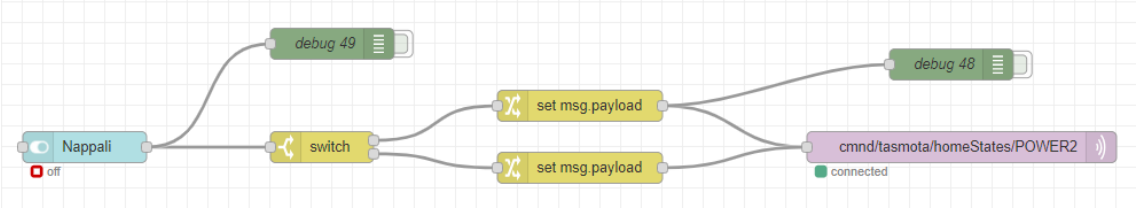
Ez egy nagyon fontos lépés, hiszen ez jelenti az okosotthon rendszer egyik pillérét, enélkül működésképtelen az egész rendszer.

4.5 Node-RED elindítása

Mint ahogyan azt a rendszertervben említettem ez egy egyszerű feladat, a Node-RED hivatalos oldaláról lemásolt kód Dockerbe illesztését követően, automatikusan használhatóvá válik a legújabb verzió. Az ezt tartalmazó konténer elindítása után az első és nagyon fontos lépés a Node-RED Dashboard bővítmény hozzáadása, mert ezt alapvetően nem tartalmazza az alkalmazás, ezen keresztül érjük el az okosotthon rendszer webes kezelőfelületét. Bővítmények hozzáadását a beépített paletta kezelővel tehetjük meg, melyben megjelennek a hivatalos és a közösség által fejlesztett bővítmények is.

4.6 Lámpák integrálása

Egy egyszerű lépést választottam elsőnek, mely nem más, mint a helyiségekben található lámpák okosotthonba való integrálása. Mind a 6 helyiséghez tartozik egy az alábbiak szerint kinéző flow részlet. (10. ábra)



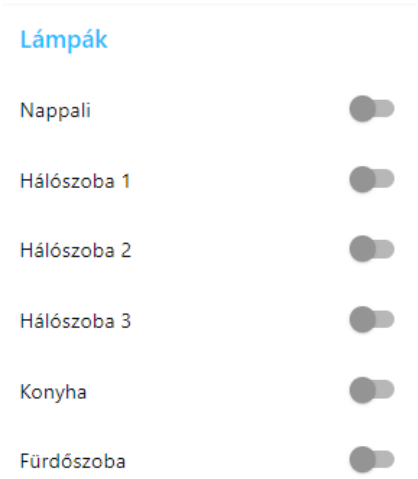
10. ábra Lámpakapcsoló flow részlet [Saját kép]

A Nappali nevezetű switch csomópont egy kapcsolót helyez el a Dashboard felületére, ezt kattintva tudjuk ki-be kapcsolni az adott lámpát. A switch feliratú csomópont a következő lépést hivatott eldönteni az alapján, hogy bekapcsoltuk, vagy pedig kikapcsoltuk a lámpát. Mivel ez a pont igaz/hamis érték alapján dönt jelen esetben és ezt az értéket küldi tovább, így szükség van ennek megváltoztatására. A set msg.payload igaz érték esetén ON, hamis érték esetén OFF üzenetet továbbít, amit MQTT-n keresztül parancsként elküldünk a Tasmota felé.

A képen látható lilás csomópont az úgynevezett MQTT out csomópont. Ennek a megfelelő konfigurálásával tudunk parancsokat küldeni a Tasmota számára. Itt hozzá kell adnunk egy MQTT szervert, amely adatait már említettem a Tasmota konfigurációjában. A szerver hozzáadása után már csak a megfelelő topic beírásával kell foglalkozni, ami jelen esetben ez lesz: cmnd/tasmota/homeStates/POWER2. A cmnd jelöli, hogy ebbe a topicba parancsokat küldhetünk, a tasmota/homeStates a Tasmota MQTT konfigurációjánál megadott egyedi topic kell, hogy legyen, a POWER2 pedig a kettes számú lámpát jelöli.

A 6 darab helyiségben található lámpa mindegyikéhez tartozik egy így kinéző rész, ezek annyiban térnek el, hogy az elől található kapcsoló neve egyedi a megkülönböztethetőség végett, valamint a végén a POWER2 helyett az adott lámpához tartozó azonosító található.

Az ebben a flowban létrehozott 6 lámpa kapcsolója a Dashboard felületén az alábbiak szerint néz ki. (11. ábra)



11. ábra Dashboard lámpa kapcsolók
[Saját kép]

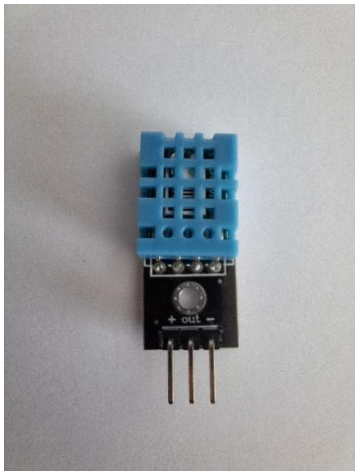
Ahhoz, hogy ezt a modellen szemléltetni lehessen, a Tasmota-t is konfigurálni kell. Ezt a Configure Module menüpont alatt tehetjük meg, ahol az ESP32-es eszközünk összes portját lehet állítani. (12. ábra)

12. ábra Lámpa portok konfigurációja
[Saját kép]

A 6 szobához keresnünk kell 6 darab szabad portot, ezek a szabad portok a képen láthatóak alapján reléként konfigurálandók, egyenként egyedi számmal. A számozás azért indul 2-ről, mert az 1es relére szükség lesz majd a későbbiekben a mozgásérzékelő használatához.

4.7 Hőmérséklet és páratartalom mérő

Következő lépésként egy szenzor integrálását választottam, ami nem más, mint egy hőmérséklet és páratartalom mérő, pontosabban egy DHT11-es szenzor. (13. ábra) Ez az érzékelő a nappaliban kap majd helyet.

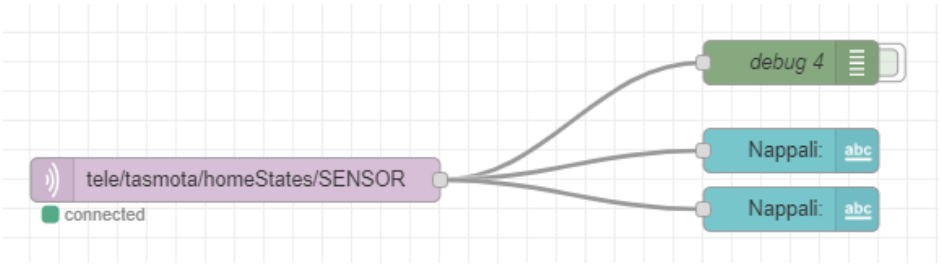


13. ábra DHT11-es szenzor
[Saját kép]

Azért erre az eszközre esett a választásom, mert a Tasmota-val nagyon egyszerűen használható, néhány kattintás után automatikusan kezeli a program. A 12-es ábrán látható konfigurációs beállítások egészülnek ki egy sorral, a 21-es porton keresztül fog csatlakozni az eszközünk, a portszám melletti legördülő listában egész egyszerűen csak ki kell választanunk a DHT11-es opciót és már használható is a szenzor. (14. ábra)

14. ábra Szenzor beállítása [Saját kép]

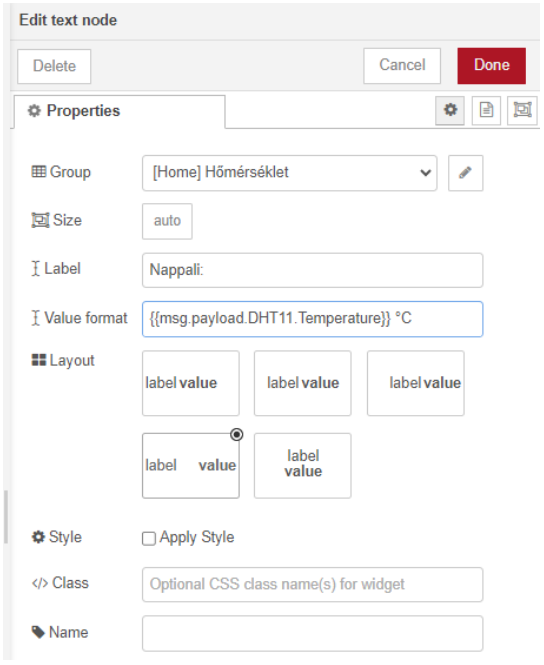
Mivel az MQTT kapcsolatunk már működik, így a Tasmota automatikusan küldi is tovább a szenzortól kapott adatokat az MQTT brókerünk felé, innen a Node-RED-en belül egy MQTT in csomóponttal meg is kapjuk a mért értékeket. (15. ábra) Ez a csomópont az MQTT out-hoz hasonlóan konfigurálandó, annyi eltéréssel, hogy itt nem a cmnd, hanem a tele szóval kell kezdődnie, amely a telemetria(telemetry) rövidítése. Valamint a beállított topic-on belül a Tasmota automatikusan létrehoz egy topic-ot SENSOR néven és ide küldi a mért értékeket.



15. ábra Hőmérséklet szenzor adatainak lekérése [Saját kép]

A szenzor által mért adatok már eleve a Node-RED számára értelmezhető formátumban érkeznek, így nincs szükség ezek átalakítására, azonnal kiírásra kerülnek a Dashboard felületére, a kék színű szöveg csomópontok segítségével.

Ezeket a csomópontokat azonban konfigurálni kell, hogy ne az egész üzenet tartalmát, hanem csak a nekünk szükséges adatokat jelenítse meg. (16. ábra)



16. ábra Szöveg csomópont konfigurációja [Saját kép]

Innen a Value format mező az, amivel ezt a személyre szabást megtudjuk tenni. Minden Node-RED üzenet, úgynevezett message-ként érkezik, így az ott látható szöveg msg.payload része adott, az utána lévő részek a lényegesebbek. A szenzorunk típusa DHT11, ami egyben automatikusan a nevét is jelenti, és mivel jelen esetben a hőmérséklet adatra van szükség, ezért a neve után a hőmérséklet angol megfelelője szerepel, a Temperature.

Ha ez alapbeállításon maradna, akkor az összes érkező adatot kiírná a hőmérséklet mellé, mint például a páratartalmat, a mérés idejét és így tovább.

A páratartalom kiírása ugyan ezen az elven történik, csak ott az üzenetnek a páratartalom része van használatban.

A Dashboard-on ezek az adatok megjelenítésre kerülnek, két különálló oszlopban. (17. ábra)

17. ábra Hőmérséklet és páratartalom megjelenítése a Dashboard felületén [Saját kép]

4.8 Kinti hőmérséklet és napszak használata

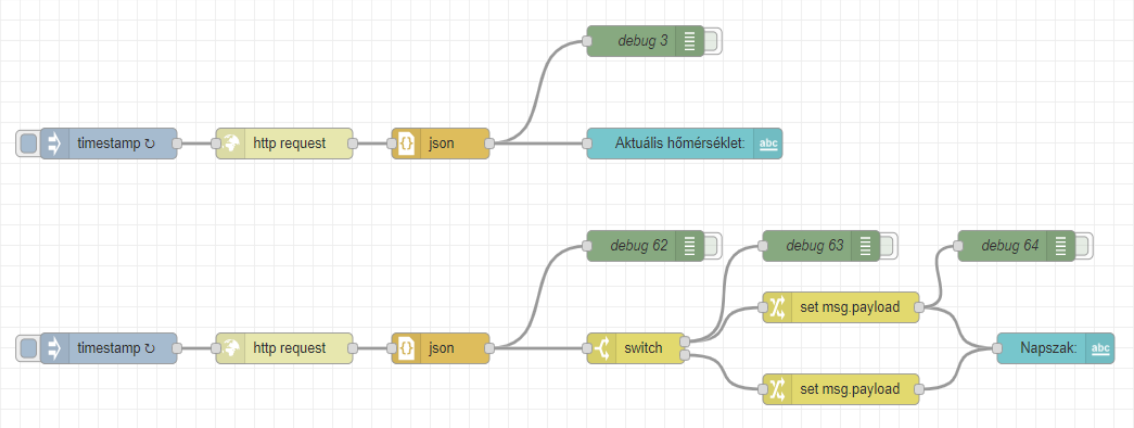
Az időjárás és napszak adatokat egy web-API szolgáltató oldaláról kapom meg, ez az oldal nem más, mint az Open-Meteo. Az oldal ingyenes, gyorsan frissülő és naprakész adatokkal szolgál a felhasználóinak, legyen szó múltbéli, aktuális vagy jövőbeni időjárási adatokról. [13]

Komplett webes kezelőfelületet szolgál, amelyen személyre szabottan választhatjuk ki a számunkra fontos adatokat. A beállított preferenciáink alapján generál nekünk egy linket, ami segítségével elérhetjük a kívánt adatokat. [14]

A számos lehetőség közül, az okosotthonba én csak a kinti hőmérsékletet és a napszakot használom. A hőmérséklet reggel jöhet jól, amikor az aznap viselni kívánt ruhákat dönti el a felhasználó. A napszak már egy sokkal hasznosabb szerepet tölt be. A mozgásérzékelő és az ajtónyitás szenzorokhoz fog segítséget nyújtani, hiszen senki sem szeretné, hogy fényes nappal világítsanak a lámpái, ezzel értékes elektromos áramot fogyasztva. Ennek

az adatnak a segítségével fogja a Node-RED eldönteni, hogy éppen nappal van vagy éjszaka, és ez alapján fogja vezérelni az adott lámpákat.

A Node-RED-ben az ehhez tartozó flow az alábbiak szerint néz ki. (18. ábra)



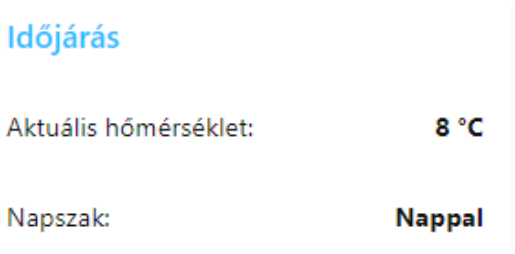
18. ábra Időjárási adatokhoz tartozó flow [Saját kép]

Az egész flow szíve a http request csomópont. Az Open-Meteo által generált linket ez fogja lekérdezni, és erre a requestre adott válaszként fogja a Node-RED megkapni a kívánt értékeket. Ahhoz, hogy ez végbe tudjon menni, kell az úgynevezett timestamp csomópont. Ennek a feladata bizonyos időközönként jelet küldeni a http request csomópontnak és ezzel működésbe hozni azt. Jelen beállítás alapján mind a kettő ilyen csomópont Node-RED indulás után 0.1 másodperccel működésbe lép, majd a későbbiekben 1 percenként automatikus elvégzi a feladatát.

A http requestre kapott válasz egy szöveges adat (string), ezzel önmagában nehéz lenne dolgozni, így átalakításra kerül JSON objektummá. Átalakítás után könnyen kezelhetővé válik és a hőmérséklet érték a benti szenzornál taglalt szöveg csomópont megfelelő konfigurálásával kiírásra kerül a Dashboard felületére.

A napszaknál annyiban bonyolultabb a feladat, hogy itt nem konkrétan a nappal vagy éjjel szót adja vissza az Open-Meteo, hanem bináris értéket, ahol az 1-es a nappalt, a 0 pedig az éjszakát jelöli. Emiatt itt szükség van egy switch csomopóntra, amely ezen értékek alapján kétfelé ágazik, nappal esetén a képen látható set msg.payload csomópont az 1-es értéket Nappal szöveggé, éjszaka esetén pedig a 0-s értéket Éjszaka szöveggé változtatja.

Végül a szöveg csomópont ezt az átalakított üzenetet kapja meg és írja a Dashboard-ra, ahol a kinti hőmérséklettel egy oszlopban jelenik majd meg. (19. ábra)

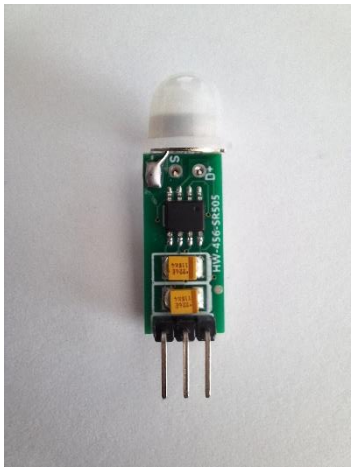


19. ábra Dashboard időjárás része [Saját kép]

A kinti hőmérséklet és a napszak külön szedve lett végül megoldva, nem pedig egyben, azért, hogy a jövőben külön-külön lehessen konfigurálni a lekérdezési gyakoriságukat.

4.9 Mozgásérzékelő

A mozgásérzékelő szenzor egy PIRMINI-SR505 típusú eszköz lett. (20. ábra) Azért erre esett a választásom mert széles körben elterjedt, megbízható működésű és nem utolsó sorban kis méretű eszköz, tökéletesen illeszkedik az okosotthont szemléltető modellbe.



20. ábra Mozgásérzékelő szenzor [Saját kép]

Működését tekintve igen egyszerű, mozgás érzékelése esetén a kimenetét magasra állítja, majd az utolsó mozgás érzékelése után 8 másodperccel alacsonyra vált. Ennek a jelnek a felhasználásával küld a Tasmota üzenetet a Node-RED számára.

Azonban ennek a szenzornak a használata nem annyira egyszerű, mint például a hőmérséklet szenzoré, hogy pár kattintás után automatikusan kezeli a Tasmota. Ezt egy kapcsolóként (switch-ként) kell hozzá társítani az ESP32 egy szabad portjához. Jelen esetben a

19-es portra esett a választásom. A 12-es ábrán látható konfigurációs beállításokat kell kiegészíteni, itt a 19-es port melletti listából ki kell választani a Switch opciót, majd egy egyedi azonosító számot. (21. ábra) A későbbiekben Switch1-ként lehet majd a szenzorra hivatkozni.

21. ábra Mozgásérzékelő beállítása [Saját kép]

Ahhoz, hogy ténylegesen működésbe lépjen a szenzor, egy vak relét (dummy relay) kell létrehozni, ezt szintén a konfigurációs beállításoknál kell megtenni, itt bármely szabad és későbbiekben sem használandó portra kell ezt beállítani.

Ezek után egy szabályt kell létrehoznunk, amely alapján a Tasmota az MQTT üzeneteket fogja küldeni. Ez a szabály a következőképpen épül fel. (22. ábra)

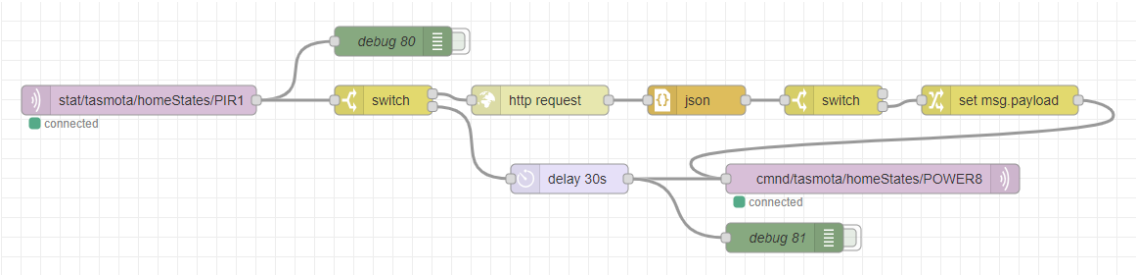
```
Rule1 on Switch1#state=1 do publish stat/%topic%/PIR1 ON endon on Switch1#state=0 do Publish stat/%topic%/PIR1 OFF endon
```

22. ábra Mozgásérzékelő Tasmota szabály [Saját kép]

Minden esetben kell, hogy legyen egy egyedi száma a szabálynak, ez jelen esetben az 1-es. Látható, hogy Switch1-ként lehet hivatkozni a szenzorra. Két állapota lehet az érzékelőnek, az 1-es és 0-s (state=1, state=0). Az 1-es a mozgás érzékelésekor bekövetkező állapot, itt vált magasra a kimenet, ebben az esetben a Tasmota egy ON MQTT üzenetet küld a PIR1 nevezetű topic-ba. Ha már nincs mozgás érzékelve akkor a 0-s állapot lép életbe, a kimenet alacsonyra vált, ilyenkor elküldésre kerül az OFF MQTT üzenet.

Ezt a szabályt a Tasmota parancssorába kell begépelni, majd ahhoz, hogy működésbe lépjen, ki kell adni egy Rule1 1 parancsot is. Ezek után már használatba is vehető a kis eszköz, a Node-RED hozzá tud férni az MQTT üzenetekhez.

Az MQTT üzenet lekérdezés ugyanúgy történik, mint az eddigiekben, az mqtt in csomópont segítségével. (23. ábra) Az első switch csomópont egy döntést végez el, az alapján, hogy mozgás érzékelése történt vagy pedig az érzékelés utáni kikapcsolás, az ide beérkező üzenet a Tasmota által küldött ON vagy OFF szöveg lesz.



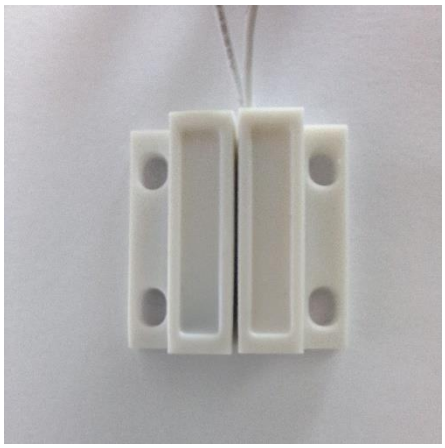
23. ábra Mozgásérzékelő Node-RED flow [Saját kép]

Mozgás érzékelése esetén történik egy http request, amely szintén az Open-Meteo oldaláról kérdezi le az adott napszakot, majd a visszakapott üzenet szintén JSON objektummá kerül átalakításra. A második switch csomópont csak abban az esetben engedi tovább az üzenetet, amennyiben éjszaka van, mivel nappal nem szükséges lámpát kapcsolni. Mielőtt felkapcsolódna a lámpa a továbbítandó üzenet szövegét meg kell változtatni, ezt végzi el a set msg.payload pont, ahol az üzenet szövege ON-ra változik. Azért van szükség üzenetváltoztatásra, mert a kezdetekben beérkező ON üzenet, a http request után megváltozik. Végül a jobb oldalon látható mqtt out csomópont küldi el ezt a Tasmota irányába, ami értelmezi az üzenetet, és felkapcsolja a megfelelő lámpát.

Mozgás érzékelése után a szenzor kimenete megváltozik, ebben az esetben lámpa lekapcsolásra van szükség. Itt nem kell üzenetet változtatni, hiszen a beérkező üzenet eleve az OFF szót tartalmazza, ezt is küldi tovább az első switch csomópont. Mivel nem szeretném, hogy egyből lekapcsolódjon a lámpa, így egy késleltetés van az üzenetre téve, ami így 30 másodperccel az utolsó mozgás érzékelése után kerül elküldésre MQTT-n keresztül. Ez a késleltetés természetesen az igényeknek megfelelően változtatható. Ehhez a részhez Dashboard elem nem tartozik.

4.10 Ajtónyitás érzékelő

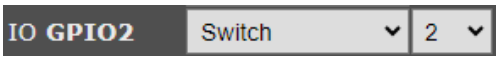
Az ajtónyitás érzékelés egy egyszerű reed csöves nyitásérzékelővel került megvalósításra. (24. ábra)



24. ábra Ajtónyitás érzékelő [Saját kép]

Működése nagyon egyszerű, van benne egy mágnes és egy reed cső. Mágneses tér hatására a csőben található lágy vasak összeérnek így zárva az áramkört. Tehát amennyiben a két fele közel van egymáshoz a kimenete magas lesz, amennyiben eltávolodnak a kimenet alacsonyra vált.

A mozgásérzékelőhöz hasonlóan kell integrálni a rendszerbe, ez is egy kapcsolóként (switch) fog működni. Szintén kap a konfigurációs beállításoknál egy szabad portot, aminél ebben az esetben is Switch opciót kell választani, majd pedig egy egyedi számot. (25. ábra) Továbbiakban Switch2-ként lehet rá hivatkozni.



25. ábra Ajtónyitás érzékelő beállítása [Saját kép]

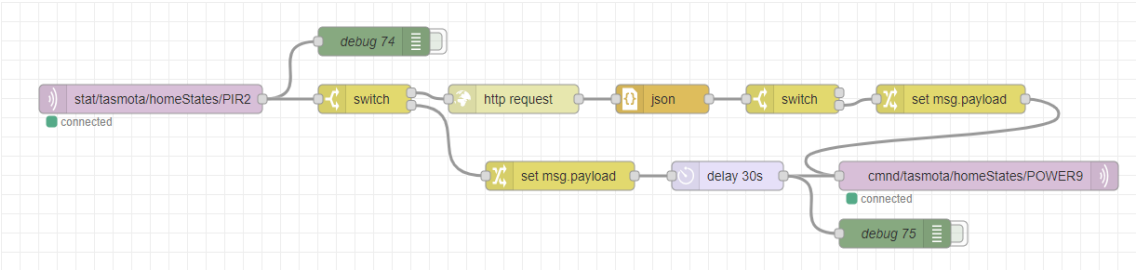
Ahhoz, hogy a Tasmota MQTT üzenetet tudjon küldeni, ebben az esetben is definiálni kell egy szabályt. (26. ábra)

```
Rule2 on Switch2#state=1 do publish stat/%topic%/PIR2 ON endon on Switch2#state=0 do Publish stat/%topic%/PIR2 OFF endon
```

26. ábra Ajtónyitás érzékelő Tasmota szabály [Saját kép]

Felépítése teljesen megegyezik a mozgásérzékelő szabályával, annyiban különbözik, hogy az azonosító adatok változnak, mint például a szabály száma, az érzékelő azonosítója, valamint a topic neve. Ezt leszámítva működése ugyanúgy történik. Természetesen egy Rule2 1 paranccsal ezt is érvénybe kell léptetni.

A továbbiakban a Node-RED már hozzá is fér a Tasmota elküldött üzeneteihez. (27. ábra)

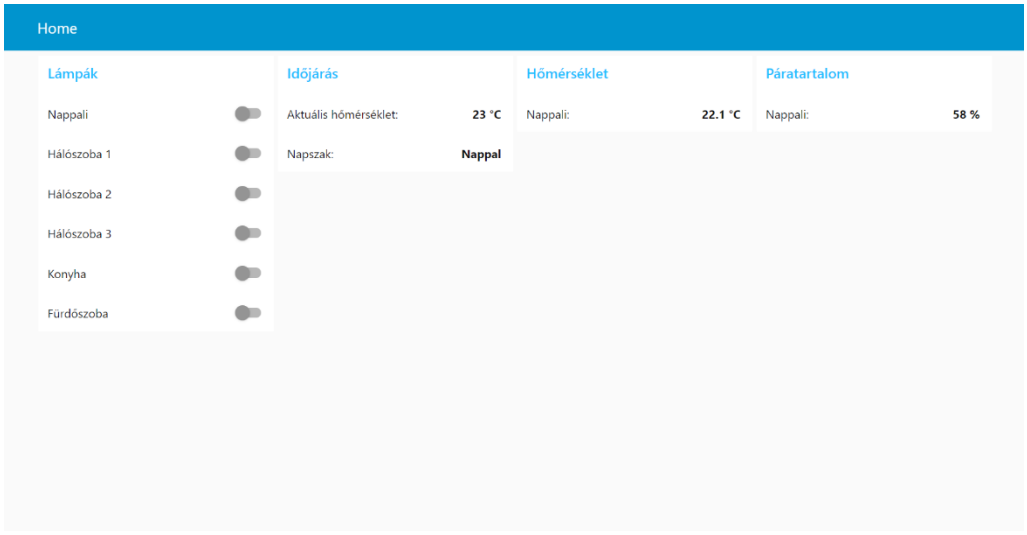


27. ábra Ajtónyitás érzékelő Node-RED flow [Saját kép]

A csomópontok gyakorlatilag ugyanazt a feladatot látják el, mint a mozgásérzékelőnél. Annyi különbséggel, hogy mivel becsukott ajtónál lesz magas kimenetünk, ezért a switch csomópont OFF üzenet esetén fogja felkapcsolni a lámpát. ON üzenet esetén fogja kikapcsolni azt, ezért ide szükség van ismételten a set msg.payload csomópontra, hogy az üzenet szövege ON-ról OFF-ra változzon, ez az üzenet utána szintén 30 másodperc késleltetéssel fog a Tasmota-hoz eljutni és végrehajtódni. Erre a kis csavarra azért van szükség, mert pont fordítva működik ez az érzékelő, becsukott ajtónál van bekapcsolt állapotban nyitott ajtónál pedig kikapcsoltban, és ezen állapotok alapján érkezik az üzenet a Tasmota felől. Ebben az esetben is változtatható a késleltetés időtartama, és ehhez sem tartozik Dashboard rész.

4.11 Webes kezelőfelület

Az egész okosotthon rendszer kezelőfelületét a Node-RED által nyújtott, már említett Dashboard szolgáltatja. Egy böngésző segítségével lehet elérni az oldalt, amely természetesen teljes mértékben, igényeknek megfelelően testre szabható. (28. ábra)

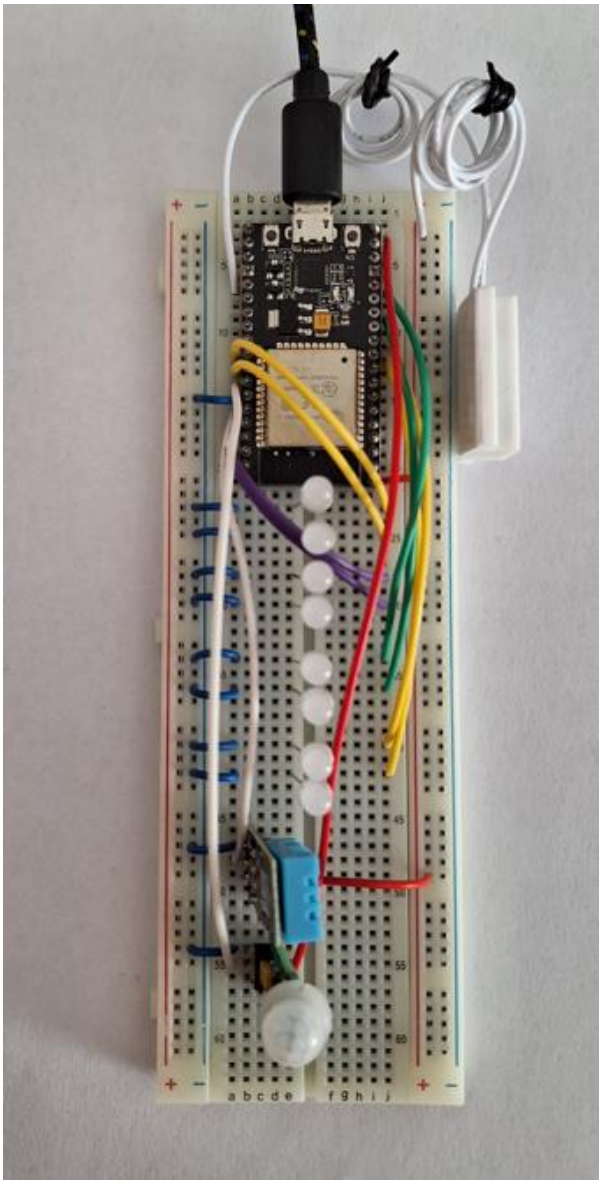


28. ábra Webes kezelőfelület [Saját kép]

Dizájn terén nem mentem bele nagyon a lehetőségekbe, maradtam az alapértelmezett beállításoknál, csak a megjelenített oszlopok elrendezésével foglalkoztam. A végeredmény egy letisztult, átlátható és könnyen kezelhető felület lett.

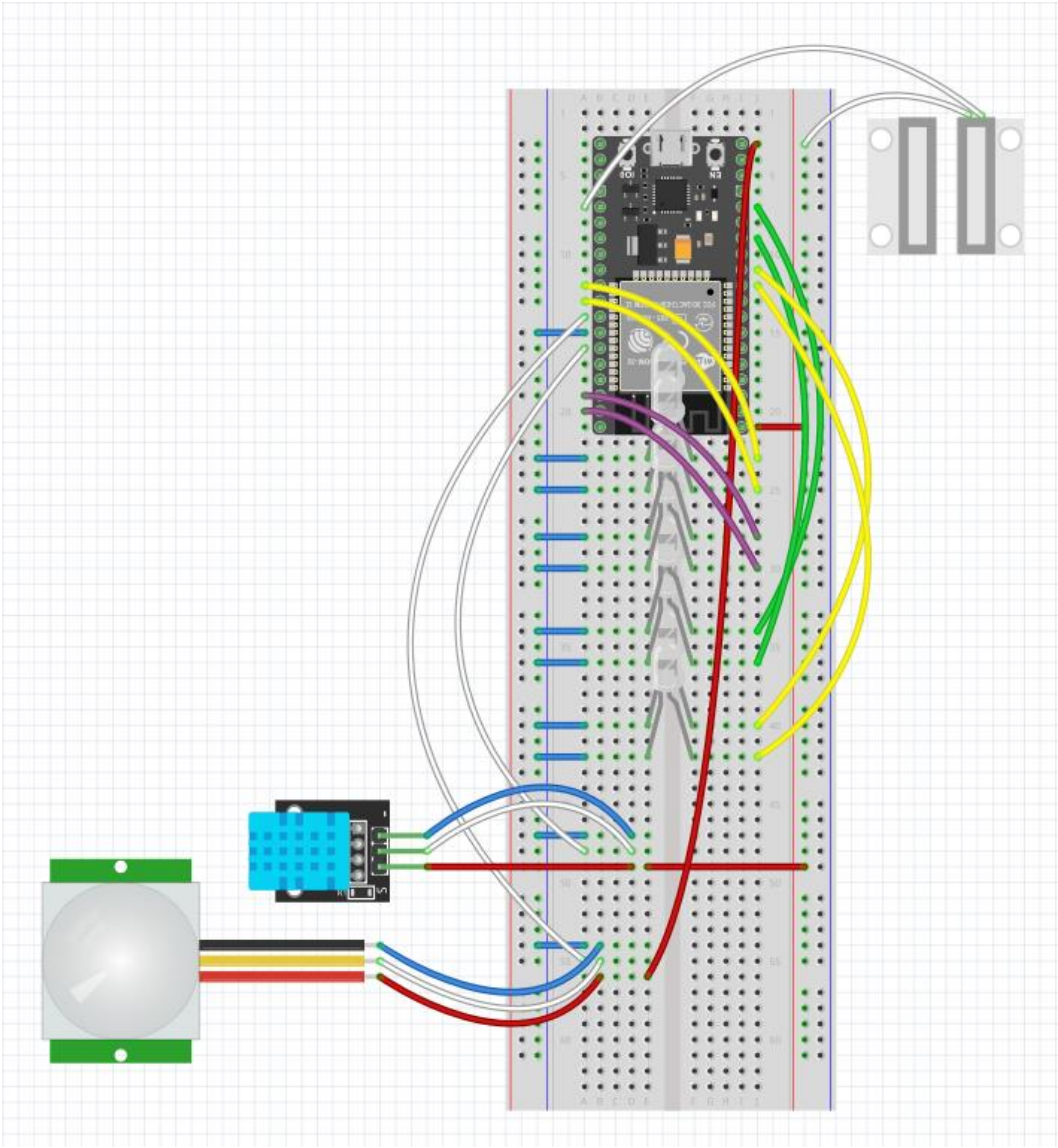
4.12 Szemléltető modell

Az elkészült rendszer szemléltetésére egy deszkamodellt készítettem, mely tartalmazza az eddig említett komponenseket, mint az ESP32, a ledek, a hőmérséklet érzékelő, a mozgásérzékelő, valamint az ajtónyitás érzékelő. Ezek az elemek egy breadboard-ra vannak helyezve és vezetékekkel csatlakoznak az ESP megfelelő lábára. (29. ábra)



29. ábra Szemléltető modell [Saját kép]

A vezetékek jobb átláthatósága érdekében a Fritzing nevű programmal elkészítettem ennek a modellnek a digitális változatát, azon jobban lehet látni a bekötések pontos helyét, valamint a rajzon a mozgás- és hőmérséklet érzékelő szintén ezen okból nem került fel az eredeti helyére. (30. ábra)



30. ábra Modell digitális rajza [Saját kép]

A képen látható mozgásérzékelő nem a valóságban használt konkrét darab, hanem annak egy nagyobbik változata, mert ehhez a programhoz még nem készült el az általam használt érzékelő modellje, ezen kívül megegyeznek, mind működésben, mind bekötésben.

5. LEGÚJABB OKOSOTTHON SZABVÁNY

5.1 Matter szabvány

A Matter a legújabb okosotthon kommunikációs szabvány. Az ezen szabvány szerint készített okos eszközök egy közös kommunikációs interfészt fognak használni a különböző adatok átvitelére. Segítségével megoldható a különböző gyártók eszközei közötti kommunikáció is, amely eddig meglehetősen bonyolult volt. A jövőben nem kell foglalkoznunk a kompatibilitás ellenőrzésén, ha például egy okos izzót szeretnénk integrálni otthonunkba, a Matter-képes izzók képesek lesznek együttműködni többféle okosotthon megoldással is, mint például az Apple HomeKit, Amazon Alexa, Google Home, és ezeken felül az összes többi Matter-képes eszközzel is hibátlanul fognak kommunikálni. [15]

5.1.1 Megalkotása

A Matter 1.0-t 2022. október 4-én jelentette be a Connectivity Standards Alliance, amely egy 550 nemzetközi vállalatot magába foglaló egyesülés. Az itt tagként szereplő nagyvállalatok közösen együttműködve dolgoznak, egy márka- és platformfüggetlen, felhasználók számára megfelelően nagy adatvédelmet, biztonságot és egyszerűséget nyújtó új termékgeneráció megalkotásán. Ez az 1.0-s kiadás több mindent is magába foglal, mint például a tesztkészletek, valamint eszközök elérhetőségét, a tanúsítási engedéllyel rendelkező laborok megnyitását, és egy teljes szoftverfejlesztő készletet is. Ezek együttes célja, hogy a szabványt támogató innovatív eszközöket minél hamarabb a piacra lehessen dobni. Ebbe az egyesülésbe olyan nagy nevek is tartoznak, mint az IKEA, az Apple, az Amazon, az LG, a Samsung Electronics, és még hosszasan lehetne sorolni. [15]

5.1.2 A szabvány előnyei okosotthon viszonylatban

A szabvány alapján készült eszközök szélesebb körben valamikor 2023 folyamán fognak elterjedni, ezek használata nagy előnnyel fog járni a végfelhasználók, és az okosotthon ipar számára. A felhasználók számára egy új eszköz installálása az otthonukba sokszor nehézségekbe ütközik. A Matter többek között ezt is hivatott megoldani, mégpedig úgy, hogy egységes beállítási lépéseket eszközöl különböző gyártók és eszközök között, a biztonság és adatvédelem szem előtt tartásával. Az összes eszköznek lesz egy üzembehelye-

zési kódja az eszközön vagy a dobozán. Az eszköz beállítását elkezdhetjük a gyártó alkalmazásában is, de akár a kedvenc okosotthon platformunk applikációjában is. Ezt a kódot beolvashatjuk QR kódként a telefonunk kamerájával, vagy akár manuálisan begépelhetjük. Ezen felül lehetséges ezen okosotthon applikációk automatizálása is, így az áram alá helyezett új eszközt automatikusan felismerik és végigvezetnek minket a beállítási lépéseken. Az első beállítás után már egyszerűen hozzáadhatjuk az új eszközünket más, általunk használt alkalmazásokhoz, így több helyről is elérve azt. Könnyen belátható, hogy a szabvány hatalmas előnyt fog jelenteni, főleg azon felhasználók számára, akik kevés tapasztalattal rendelkeznek ezen téren. [15]

6. ÖSSZEFOGLALÓ

A dolgozat elején fontos szerepet kapott az okosotthon rendszer felé támasztott elvárások meghatározása, a megvalósítani kívánt funkciók tervezése, összegzése.

Ezek után következhetett a rendszer elméleti megtervezése. Kulcsfontosságú lépés volt ez, hiszen a célnak legmegfelelőbb komponenseket itt kellett kiválasztani. Utánanéztem a rendelkezésre álló fejlesztői eszközöknek, mikrokontrollereknek és MQTT brókereknek. Ezek mind-mind nagy halmazok, ezeket szűkítettem le a dolgozat szempontjából legmegfelelőbbekre, majd kiválasztottam a véglegesen használandókat.

Ezek után következett, a kiválasztott komponensek részletesebb ismertetése, bemutatása, kiemelve a dolgozat szempontjából lényegesebb előnyeiket, funkcióikat.

Elsőként a fejlesztőeszközzel kezdtem, ami nem más, mint a Node-RED. Bemutattam, hogy mégis mi ez a program, hogyan működik, mik az alapjai, majd ismertettem az okosotthon elkészítéséhez szükséges fontosabb funkcióit.

Ezt szorosan követte a program futtatásához szükséges alkalmazás, a Docker, itt szintén bemutattam magát az alkalmazást, valamint a Node-RED Dockerrel történő használatának menetét.

Még mindig a rendszerterv részeként ismertettem az eszközök mindegyike által használt MQTT protokollt, meglehetősen részletesen.

Innen tértem át a gyakorlati megvalósításhoz, melynek elején rögtön a kiválasztott MQTT brókerhez történő regisztrációt írtam le részletesebben, végig vezetve a folyamat összes lépésén keresztül.

Amint ezzel megvoltam, következett a mikrokontrolleren használt Tasmota rendszer rövid bemutatása, majd a mikrokontrollerre történő feltelepítése, természetesen itt is minden lépést számba véve. A Tasmota-t elindítása után összecsatlakoztattam az MQTT kiszolgálóval és innentől kezdődött a tényleges okosotthon rendszer megvalósítása.

A dolgozat folytatásában taglaltam a használt eszközöket és azok integráláshoz szükséges Node-RED programot és Tasmota beállítást. Minden használt eszközt röviden ismertettem is.

Zárásként írtam a legújabb, nemsokára megjelenő okosotthon szabványról, ami kiküszöböli majd az egyes gyártók közötti réseket, lehetővé téve az okos eszközök kommunikációját gyártótól független. Ez megoldást nyújt majd a dolgozat megszületésének egyik okára, a sok különböző gyártóhoz tartozó alkalmazás sokaság leváltására.

Végezetül írok pár sort a lehetséges fejlesztésekről. Ilyen lehet a már meglévő elemek sokszorosítása, több hőmérséklet érzékelő beszerelése, még több lámpa okosítása, nyitásérzékelők felszerelése az ablakokra. További lehetőség a még nem használt, új funkciókkal rendelkező eszközök integrálása, mint például az okoskonnektorok, wi-fi-s biztonsági kamerák, vagy egy időjárás állomás.

7. SUMMARY

At the beginning of the thesis, an important role was given to the definition of the expectations of the smart home system, the design and the summary of the functions to be implemented.

This was followed by the theoretical design of the system. This was a key step, since the components most suitable for the purpose had to be selected here. I looked into the available development tools, microcontrollers and MQTT brokers. These are all large sets, and I narrowed them down to the ones most relevant to the thesis, and then selected the ones I would use.

After that, the selected components were described in detail, highlighting their most important advantages and functions.

I started with the development tool, which is Node-RED. I explained what this program is, how it works, what are its basics, and then I described the most important functions needed to create a smart home.

This was closely followed by the application needed to run the program, Docker, where I also introduced the application itself and the process of using Node-RED with Docker.

Still as part of the system design, I described the MQTT protocol in detail used by each of the tools.

From there, I moved on to the practical implementation, starting right away by describing the registration process for the selected MQTT broker in more detail, taking you through all the steps of the process.

Once I had done that, I went on to give a brief introduction to the Tasmota system used on the microcontroller and then to install it on the microcontroller, again of course, covering all the steps. Once Tasmota was up and running, I connected it to the MQTT server and from there the actual implementation of the smart home system began.

In the continuation of the thesis, I described the devices used and their integration with Node-RED and Tasmota. I also briefly described all the tools used.

In the end, I wrote about the latest, soon to be released smart home standard, which will eliminate the gaps between vendors, allowing smart devices to communicate independently of vendors. This will address one of the reasons for writing this thesis, namely the multitude of applications from many different vendors.

Finally, a few lines on possible improvements. These could include duplicating existing elements, installing more temperature sensors, adding more lights, installing opening sensors on windows. Another possibility is to integrate devices with new features that are not yet in use, such as smart connectors, security cameras with wi-fi, or a weather station.

8. IRODALOMJEGYZÉK

[1] openHAB Foundation e.V., „Introduction | openHAB," openHAB Foundation e.V., 2023. [Online]. Available: https://www.openhab.org/docs/. [Hozzáférés dátuma: 8. március 2023].

[2] OpenJS Foundation, „Node-RED," OpenJS Foundation, [Online]. Available: https://nodered.org/about/. [Hozzáférés dátuma: 8. március 2023].

[3] Simply IoT Sensors, „Difference between ESP 32 vs Raspberry pi," Simply IoT Sensors, 2021. [Online]. Available: https://www.simplyiotsensors.com/2023/05/ESP-32-vs-Raspberry-pi.html. [Hozzáférés dátuma: 24. március 2023].

[4] Espressif Systems, „ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems," Espressif Systems, 2023. [Online]. Available: https://www.espressif.com/en/products/socs/esp32. [Hozzáférés dátuma: 24. március 2023].

[5] Eclipse Foundation, „Eclipse Mosquitto™ | projects.eclipse.org," Eclipse Foundation, [Online]. Available: https://projects.eclipse.org/projects/iot.mosquitto. [Hozzáférés dátuma: 12. április 2023].

[6] HiveMQ GmbH, „HiveMQ MQTT Broker - Enterprise ready to move IoT data," HiveMQ GmbH, 2023. [Online]. Available: https://www.hivemq.com/products/mqtt-broker/. [Hozzáférés dátuma: 12. április 2023].

[7] OpenJS Foundation, „The Core Nodes : Node-RED," OpenJS Foundation, [Online]. Available: https://nodered.org/docs/user-guide/nodes. [Hozzáférés dátuma: 06. november 2023].

[8] OpenJS Foundation, „Running on Windows : Node-RED,” OpenJS Foundation, [Online]. Available: https://nodered.org/docs/getting-started/windows. [Hozzáférés dátuma: 20. november 2023].

[9] OpenJS Foundation, „Running under Docker : Node-RED,” OpenJS Foundation, [Online]. Available: https://nodered.org/docs/getting-started/docker. [Hozzáférés dátuma: 20. november 2023].

[10] Docker Inc., „Docker: Accelerated Container Application Development,” Docker Inc., 2023. [Online]. Available: https://www.docker.com/. [Hozzáférés dátuma: 10. október 2023].

[11] HiveMQ GmbH, „MQTT Essentials - All Core Concepts Explained,” HiveMQ GmbH, 2023. [Online]. Available: https://www.hivemq.com/mqtt-essentials/. [Hozzáférés dátuma: 10. október 2023].

[12] T. Arends, „About - Tasmota,” [Online]. Available: https://tasmota.github.io/docs/About/. [Hozzáférés dátuma: 07. november 2023].

[13] Open-Meteo.com, „Free Open-Source Weather API | Open-Meteo.com,” Open-Meteo, [Online]. Available: https://open-meteo.com/. [Hozzáférés dátuma: 16. február 2024].

[14] Open-Meteo.com, „Docs | Open-Meteo.com,” Open-Meteo, [Online]. Available: https://open-meteo.com/en/docs/. [Hozzáférés dátuma: 16. február 2024].

[15] Alza.cz a.s., „Mi az a Matter szabvány? | Alza.hu,” 08. november 2022. [Online]. Available: https://www.alza.hu/matter-szabvany. [Hozzáférés dátuma: 10. október 2023].

9. ÁBRAJEGYZÉK

1. ábra Blokkvázlat [Saját kép] 4
2. ábra MQTT fix fejléc formátum példa [http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html] 13
3. ábra Regisztrációs ablak [https://auth.hivemq.cloud/login?state=hKFo2SBwaEJ5M3E0S1NOa2hGaXRLblFKYzYxMjZaRVZvd3BGcqFupWxvZ2luo3RpZNkgNmd3SUROWmlvakdEb3Z1aU5ZUEhXQTFLNGpEY0RzWmKjY2lk2SBJYWpvNGUzMmp4d1VzOEFkRnhneFFuMlZQM1l3SVpUSw&client=Iajo4e32jxwUs8AdFxgxQn2VP3YwIZTK&protocol=oauth2&audience=hivemq-cloud-api&redirect_uri=https%3A%2F%2Fconsole.hivemq.cloud&scope=openid%20profile%20email&response_type=code&response_mode=query&nonce=Mks0NDM0cFpiUDdKdEFpalVIVU5Zc0FhaWlTX3l1Z3F2clZfNkpkRFY0Mw%3D%3D&code_challenge=SrtzhHkQj52cZwT0H81QS1z7FkXtSaNMHawEW3lAr5E&code_challenge_method=S256&auth0Client=eyJuYW1lIjoiYXV0aDAtc3BhLWpzIiwidmVyc2lvbiI6IjEuMTMuNiJ9] 15
4. ábra Regisztráció véglegesítése [Saját kép] 16
5. ábra HiveMQ webes kezelőfelülete [https://console.hivemq.cloud/] 16
6. ábra Telepítő oldal [https://tasmota.github.io/install/] 18
7. ábra Csatlakoztatott eszköz telepítésének megkezdése [Saját kép] 18
8. ábra ESP32-n futó Tasmota menüje [Saját kép] 19
9. ábra MQTT konfigurálás [Saját kép] 20
10. ábra Lámpakapcsoló flow részlet [Saját kép] 21
11. ábra Dashboard lámpa kapcsolók [Saját kép] 22
12. ábra Lámpa portok konfigurációja [Saját kép] 22
13. ábra DHT11-es szenzor [Saját kép] 23
14. ábra Szenzor beállítása [Saját kép] 23
15. ábra Hőmérséklet szenzor adatainak lekérése [Saját kép] 24
16. ábra Szöveg csomópont konfigurációja [Saját kép] 24
17. ábra Hőmérséklet és páratartalom megjelenítése a Dashboard felületén [Saját kép] 25
18. ábra Időjárási adatokhoz tartozó flow [Saját kép] 26

19. ábra Dashboard időjárás része [Saját kép] 27
20. ábra Mozgásérzékelő szenzor [Saját kép] 27
21. ábra Mozgásérzékelő beállítása [Saját kép] 28
22. ábra Mozgásérzékelő Tasmota szabály [Saját kép] 28
23. ábra Mozgásérzékelő Node-RED flow [Saját kép] 29
24. ábra Ajtónyitás érzékelő [Saját kép] 30
25. ábra Ajtónyitás érzékelő beállítása [Saját kép] 30
26. ábra Ajtónyitás érzékelő Tasmota szabály [Saját kép] 30
27. ábra Ajtónyitás érzékelő Node-RED flow [Saját kép] 31
28. ábra Webes kezelőfelület [Saját kép] 31
29. ábra Szemléltető modell [Saját kép] 32
30. ábra Modell digitális rajza [Saját kép] 33