MŰEGYETEM 1782

SZAKDOLGOZAT FELADAT

Zombó Balázs
Villamosmérnök hallgató részére

IoT alapú automatizált rendszerek

Az Internet of Things (IoT) alapú automatizált rendszerek a hagyományos automatizált rendszerek képességeit egészítik ki az adatok rendszeregységek közötti interneten keresztül történő intenzív cseréjével és az adatok akár felhőben történő feldolgozásával. Az IoT alapú automatizált rendszerek között elképzelhető olyan alkalmazás, amelyben a mérési adatok alapján különböző döntéseket hoz a rendszer, hogy bizonyos előírt feltételek teljesülése érdekében történjen-e bevatkozás.

Számos területen találkozhatunk automatizált rendszerekkel. Jelen Szakdolgozatban növényi vagy állati életkörülmények felügyeletét ellátó IoT alapú rendszer megvalósítása a feladat. A rendszer különböző szenzorainak mérési adataiból, illetve a beavatkozó egységek státuszának ismeretéből egy megvalósítandó algoritmus segítségével döntések hozhatóak arra vonatkozóan, hogy bizonyos életkörülményeket támogató feltételek biztosítva legyenek.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismertesse az IoT alapú automatizált rendszerek alkalmazhatóságát, kiemelve a saját rendszer specialitásait
- Ismertesse a megvalósított adatgyűjtő és beavatkozó rendszer rendszertervét, hardver és szoftver komponenseit
- Készítsen adatbázist az adatok tárolására és algoritmust azok rendszerműködést befolyásoló feldolgozására
- Tegye lehetővé, hogy az adatok online módon elérhetőek és lekérdezhetőek legyenek
- Ismertesse, hogy milyen bővítési lehetőségeket lát a rendszer szolgáltatásainak továbbfejlesztésére

Tanszéki konzulens: Krébesz Tamás István, tanársegéd
Külső konzulens: -

Budapest, 2020.10.10.

……..
Dr. Dabóczi Tamás
tanszékvezető
egyetemi tanár, DSc

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Zombó Balázs

IOT ALAPÚ AUTOMATIZÁLT RENDSZEREK

KONZULENS

Krébesz Tamás István

BUDAPEST, 2020

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Zombó Balázs**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. 12. 14.

.....

Zombó Balázs

Összefoglaló

A szakdolgozat témája az IoT - alapú automatizált rendszerek. A szakdolgozat egy intelligens és automatizált öntözőrendszer elkészítéséről szól, amelyet egy NodeMCU V2 fejlesztőpanel segítségével készítettem el. A szakdolgozat elején bemutatom, hogy jelenleg hol tart az IoT technológia. Majd az előzetes információk kerülnek bevezetésre, amelyek szükségesek az IoT megértéséhez és az automatizált rendszer elkészítéséhez. A szöveg kitér az IoT architektúrájára és építőelemeire, röviden bemutatásra kerül a big data, cloud és fog computing. A rendszer elkészítéséhez szükséges hardver és szoftver komponensek is bemutatásra kerülnek, mint a felhasznált szenzorok, beavatkozó szervek, az Arduino IDE, ThingSpeak.

Az általam megvalósított rendszerben az egyik ilyen szenzor egy talajnedvesség érzékelő, amelyből kiolvasott analóg adatot a NodeMCU digitalizálja, majd egy általam kidolgozott algoritmus alapján eldönti, hogy szükséges -e a vízpumpa aktiválása. Ha a talaj a száraz tartományba esik, akkor az öntözés beindul. Mivel egy növénynek idő kell a víz felszívásához a talajból, ezért elég naponta 1-2 alkalommal lefuttatni a folyamatot. A projektben power managementet alkalmazok, hogy a rendszer később akár elemről is működhessen. A szakdolgozat végén az elkészült rendszernek a továbbfejlesztéséről olvashatunk.

Abstract

The topic of the BSc thesis work is IoT - based automated systems. The BSc thesis work is about the construction of an intelligent and automated irrigation system, which I designed and implemented applying a NodeMCU V2 development panel. In the introductory part of the BSc thesis work, the state-of the art of IoT market is presented. After that the preliminary information needed to understand the IoT and build the automated system is introduced. The text covers the architecture and building blocks of IoT, with a brief introduction to big data, cloud and fog computing. The hardware and software components needed to build the system is also introduced, such as the applied sensors, the actuators, the Arduino IDE, ThingSpeak. One such a sensor in my own implementation is a soil moisture sensor, by which the analog data provided is digitized by the NodeMCU and then used by an algorithm of my own design to decide whether the activation of a water pump is needed or not. If the soil moisture level falls in the dry range, irrigation is started. Since a plant needs time to absorb water from the soil, it is enough to run the process 1-2 times a day. I use power management, in order to operate the system even on battery. At the end further development directions of the completed system is presented.

1 Bevezetés

Az okosotthon vagy okos eszközök téma napjainkra rendkívül népszerűvé vált. Mindenki okostelefonokkal jár, vezeték nélkül kommunikáló okos eszközökkel vannak tele az üzletek. Ez motivált, hogy a témaválasztásom az Internet of Things (IoT) területre essen. Az IoT, avagy a dolgok internete az egymással kommunikáló okos eszközökről szól. Ez az egyre fejlődő iparág jobbnál-jobb technológiai újításokkal teszi kényelmesebbé az emberek hétköznapi életét. Az 1. ábra azon eszközöket szemlélteti, amelyeken az IoT alkalmazható.

1. ábra: Az IoT alkalmazásának szemléltetése [1]

Az automatizált rendszerekkel emberi beavatkozás nélkül tudunk ipari vagy otthoni tevékenységeket végezni, legyen szó egy gyártósorról vagy akár a lakásunk fűtő rendszeréről. Ha a kettőt kombináljuk, akkor egy telefonról is monitorozható okos rendszert kapunk, amely az élet számos területén megkönnyíti az életünket. Például a növényeinket megöntözi a rendszer, ha kezdenek kiszáradni vagy sötétedéskor a redőnyök automatikusan leengednek, a kerti lámpák bekapcsolnak. Jelen szakdolgozat betekintést nyújt az IoT alapú rendszerek felépítésébe, a ESP8266 WiFi modul és a NodeMCU V2 fejlesztőpanel használatába. A szakdolgozat a téma iránt érdeklődőknek akár gyorstalpalóként is szolgálhat és az első Arduino-s projektjüket is könnyen

elkészíthetik a leírt információk és a saját projektem alapján. A munkám célja, hogy választ adjon a NodeMCU V2 fejlesztőpanel alkalmazhatóságára. A téma választásakor a következő kérdések merültek fel bennem: Mik a NodeMCU határai? Elég -e egy NodeMCU egy lakás, kertes ház, állatkert ellátására?

A dolgozatom felépítése az alábbi fejezetek alapján kerül bemutatásra. A második fejezetben röviden kifejtésre és elemzésre kerülnek a feladatkiírásban szereplő egyes feladatok. A harmadik fejezetben előrejelzésekről és piackutatásról írok, bevezetésre kerül az IoT architektúrája, az IoT elemei és betekintést nyerünk az IoT által használt protokollokba, szabványokba. A negyedik fejezetben megtalálható a feladatok részletes elemzése, a projekt megtervezése és megvalósítása. Ebben a fejezetben kitérek az általam használt IoT -s protokollokra, a fejlesztőkörnyezetre és a cloud platformra. A fejezetben több megoldási lehetőséget is vizsgálok és magyarázatot adok az általam választott irányokra. Az ötödik fejezetben egy rövid összefoglaló olvasható a kutatás eredményeiről és a projekt továbbfejlesztési lehetőségeiről.

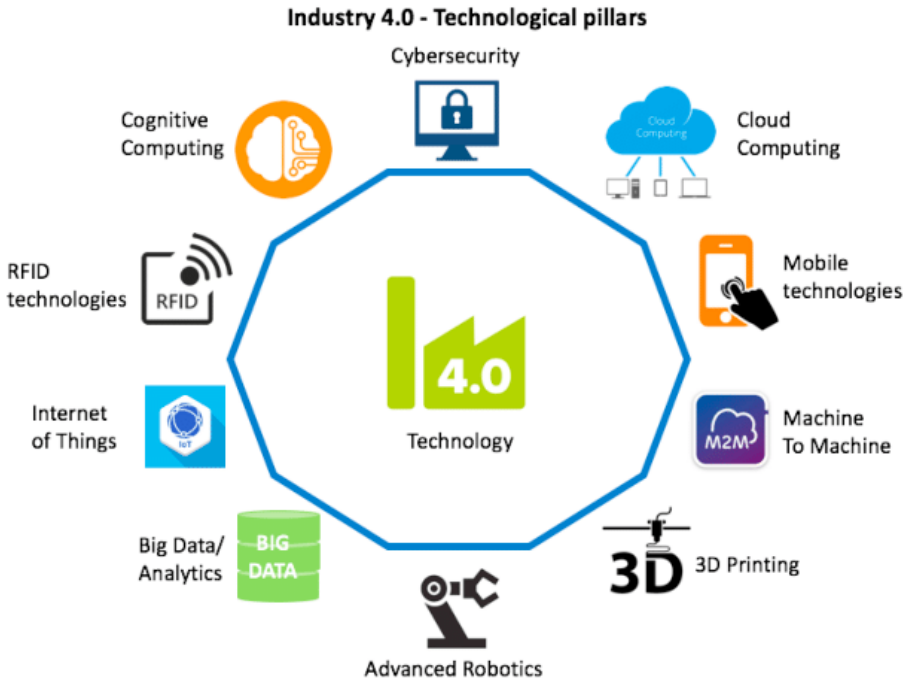
2 A feladatkiírás pontosítása

Ebben a fejezetben a feladatkiírás által megjelölt feladatok lesznek röviden kifejtve és elemezve. Az első pontban meghatározott feladat, hogy a szakdolgozat készítőjének ismertetnie kell az IoT alapú automatizált rendszerek alkalmazhatóságát, kiemelve a saját rendszer specialitásait. Automatizált rendszerre példa lehet az [2] és [3] források alapján egy fűtő, szellőztető és légkondicionáló rendszer (HVAC), egy farm öntözőrendszere, egy irodaház beléptetőrendszere vagy egy gyárépület gyártósora. A 2. ábra a Suzuki esztergomi gyárának automatizált gépkocsi gyártósorát szemlélteti.

2. ábra: Automatizált gépkocsi gyártósor [4]

Az IoT technológiát hozzáadva ehhez a rendszerhez, egy olyan automatizált rendszert kapunk, amelyet képesek vagyunk távolból, hálózaton keresztül monitorozni: A vezeték nélküli kommunikációs modullal felszerelt szenzorok az adataikat egyből egy felhő alapú rendszerbe továbbítják és az adatokat már a felhőben dolgozzák fel. A feldolgozott adatok alapján a rendszer ezután kiadhatja a megfelelő parancsokat a beavatkozó szerveknek. Ez azt jelenti, hogy ha az 1. ábrán bemutatott gyártósoron a szenzorok nem kábelen keresztül továbbítják az adataikat, hanem van egy beépített

vezeték nélküli kommunikációs egységük és azon kommunikálnak az adatokat feldolgozó egységgel. Ilyen IoT alapú automatizált rendszerek gyakorlati alkalmazására példa lehet az imént említett gyártósor. Amikor az IoT és az ipari automatizálás találkozik, akkor eljuthatunk az Ipar 4.0 fogalmához. Az Ipar 4.0 egyesek szerint a negyedik ipari forradalmat jelenti [5], az ipari digitalizációt, más forrásból [6] származó információk alapján az Ipar 4.0 egy szűkebb fogalom, amely a vállalati szférát helyezi középpontba. Az [7] forrás alapján az Ipar 4.0 egyik alapvető technológiája az IoT. Az Ipar 4.0 alkalmazása lehetővé teszi a virtuális és fizikai világ összekapcsolását, virtuális szimulációk futtatását, valós rendszerek monitorozását, a gyártósoron lévő szenzorok kommunikációját, esetleges termékhibák előrejelzését, hogy azt a soron következő beavatkozó szerv elkülönítse a hibamentes daraboktól. A 3. ábra az Ipar 4.0 technológiáit szemlélteti.

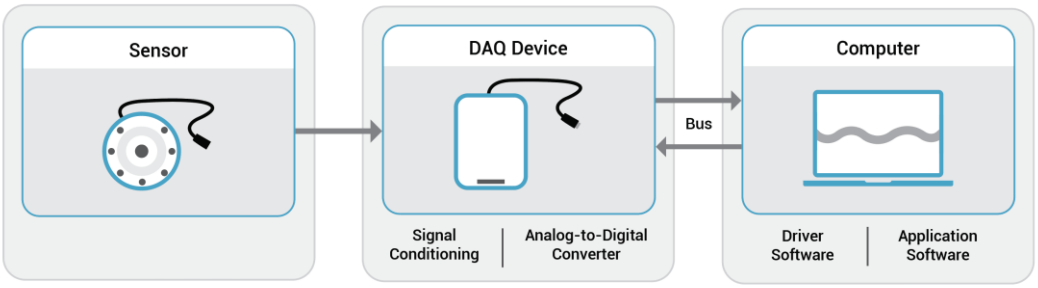


3. ábra: Az IoT szerepe az ipari automatizálásban - Ipar 4.0 technológiái [8]

A projektem egy IoT alapú automatizált öntözőrendszer. A rendszer központi egysége egy NodeMCU V2 fejlesztői panel. A rendszerem a szenzorból jövő adatokat az ESP8266 WiFi modul segítségével küldi fel a cloud platformra, ahol grafikonon megfigyelhető az adatok változása. Én a ThingSpeak ingyenes platformot használtam a projektemhez, mert egyszerű a kezelése és a projekt méretéhez mérten bőven elegendő számomra. Irodalomkutatás során felmértem a hasonló rendszerek képességeit [9], [10], [11] és összevetettem azokat a saját rendszerem képességeivel. Arra megállapításra

jutottam, hogy az általam készített rendszer legfőbb erőssége a hasonlókkal szemben, hogy hangsúlyt helyez az energiagazdálkodásra, hogy a NodeMCU egy kisebb elemről is sokáig működőképes legyen. Erről bővebben a negyedik fejezetben olvashatunk.

A feladatkiírás második pontjában a szakdolgozat készítőjének ismertetnie kell a megvalósított adatgyűjtő és beavatkozó rendszer rendszertervét, hardver és szoftver komponenseit. Először tisztázni szeretném az adatgyűjtő rendszer fogalmát. A [12] és [13] források alapján az adatgyűjtő rendszer egy szenzor vagy szenzorok, egy adatgyűjtő eszköz és egy számítógép együttese. Az adatgyűjtő eszköz a szenzor által mért analóg jelet alakítja át digitális jellé, olyan formába, hogy az a számítógép számára feldolgozható legyen. Az adatgyűjtő eszköz egyfajta interfészként szolgál a szenzorok és a számítógép között. Az adatgyűjtő rendszer angol megnevezése data aquisition system (DAS). A DAS felépítését az előzőekben ismertetettekkel összhangban a 4. ábra szemlélteti.



4. ábra: Az adatgyűjtő rendszer felépítése [13]

Az ESP8266 modul is alkalmazható adatgyűjtő eszközként, hiszen van egy belső analóg-digitális átalakítója (ADC). A megtervezett rendszer hardver komponensei egy ESP8266 ESP-12 NodeMCU V2 típusú fejlesztőpanel, egy SOILCAP-V12 kapacitív talajnedvesség érzékelő szenzor, egy miniatűr merülő vízszivattyú, valamint egy N-csatornás MOSFET.

A feladatkiírás harmadik pontjában a szakdolgozat írójának készítenie kell egy adatbázist az adatok tárolására és algoritmust azok rendszerműködést befolyásoló feldolgozására. Az adatok tárolására egy cloud platformot alkalmazok, amelyről manuálisan is letölthetőek a kívánt információk. A szenzorból érkező adatokat az ESP8266 mikroprocesszora dolgozza fel, még a felhő alapú adatbázisba kerülésük előtt,

így a rendszer hamarabb beavatkozhat a környezetbe. Ez a projekt kivitelezése miatt lehetséges, ugyanis a vízpumpát közvetlenül a NodeMCU -ról irányítjuk.

A feladatkiírás negyedik pontjában a szakdolgozat készítőjének lehetővé kell tennie, hogy az adatok online módon elérhetőek és lekérdezhetőek legyenek. Ennek megoldására a ThingSpeak cloud platform lesz felhasználva, amely segítségével grafikonon ábrázolható a talajnedvesség százalékos aránya. Erről és az előzőekről bővebben a negyedik fejezet szoftveres részében olvashatunk.

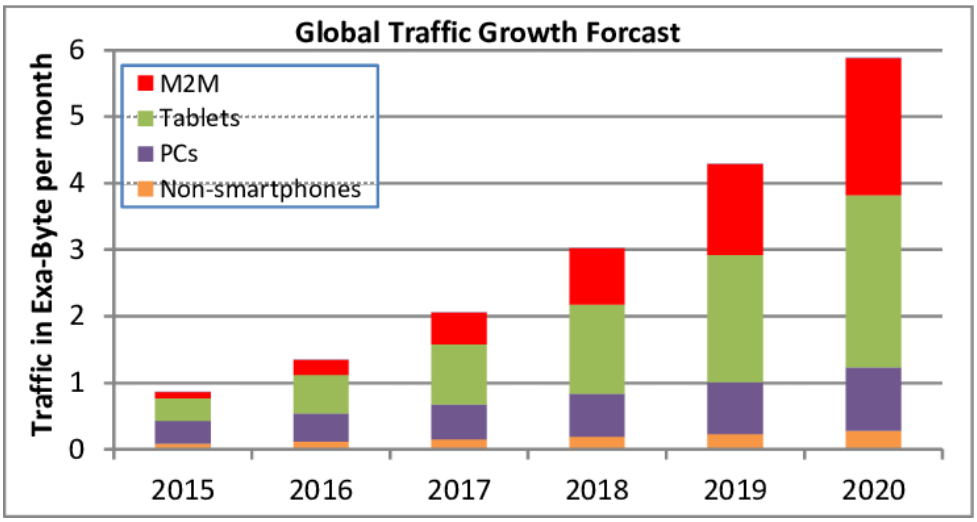
A feladatkiírás ötödik pontjában a szakdolgozat írója ismertesse, hogy milyen bővítési lehetőségeket lát a rendszer szolgáltatásainak továbbfejlesztésére. A rendszer csak egyetlen NodeMCU -t alkalmaz és az adatokat is lokálisan dolgozza fel. A rendszer bővítésére egy lehetőség a NodeMCU -k számának bővítése és a felhőben történő adatfeldolgozás. Ez az ötödik fejezetben kerül bővebb kifejtésre.

3 Előzetes ismeretek

Ebben a fejezetben az IoT piaci helyzetéről és IoT-val kapcsolatos előrejelzésekről értekezek. Bemutatásra kerülnek az IoT architektúrája, elemei, szabványai és az IoT által alkalmazott technológiák. Mindez azért szükséges, hogy egy átfogó képet kapjunk az IoT-ről és eltudjuk helyezni benne az általam megvalósított rendszert.

3.1 Piackutatás és előrejelzések

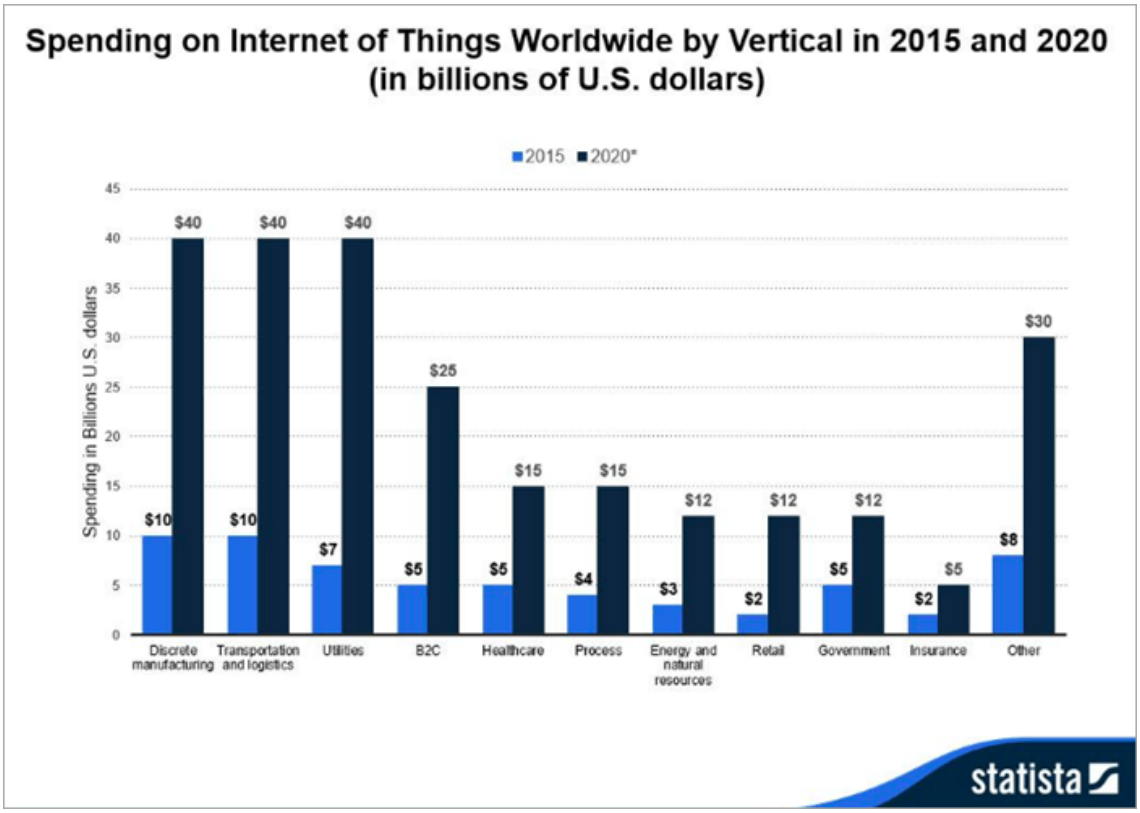
Egy 2015 -ben publikált cikk szerint a telepített IoT alapú okoseszköz egységek száma várhatóan 212 milliárdot is elérheti 2020-ra [2]. A machine-to-machine (M2M) adatforgalom 2022-re a teljes internet adatforgalmának 45% -át teheti ki. 2017-ben azt jósolták, hogy az M2M kapcsolatok száma 2020-ra az összes létező kapcsolat több mint 26% -át fogja kitenni [14]. Az M2M adatforgalom 38%-kal fog növekedni 2015 és 2020 között. Az 5. ábrán a 2017-ben megjósolt M2M adatforgalom növekedésének az előrejelzése látható.



5. ábra: Globális M2M adatforgalom növekedésének előrejelzése egy 2017-es cikkből [14]

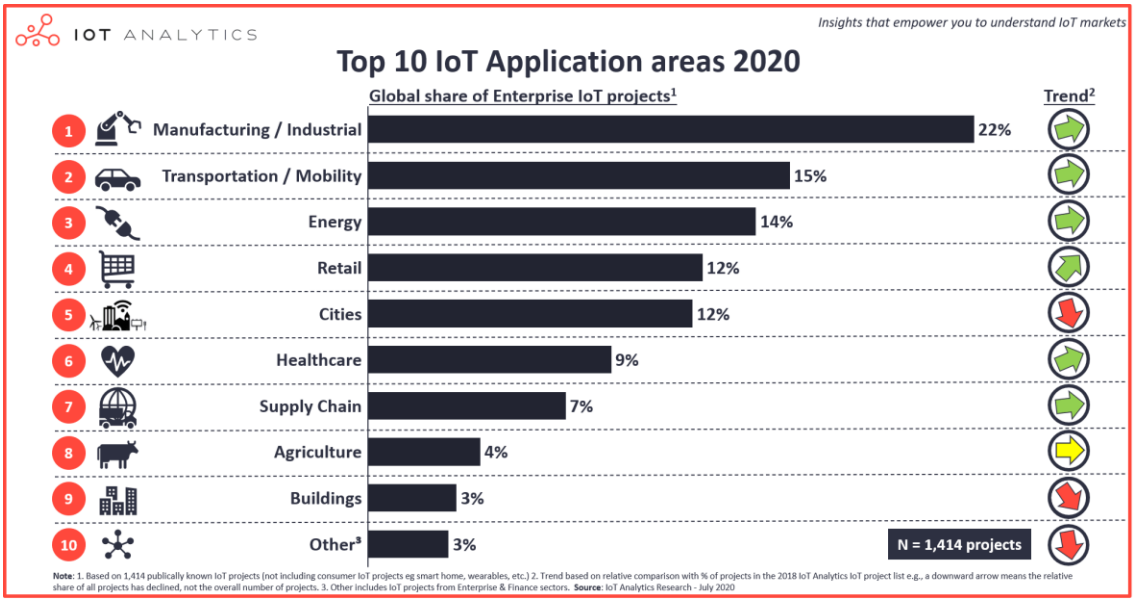
Egy 2017- es cikk alapján a Gartner szerint 2020-ra az aktív eszközök száma 20.4 milliárd lesz [15]. Ezek az adatok természetesen már elavultnak tekinthetők, egyedüli hasznos információ, amely kivehető a régebbi előrejelzésekből, hogy az IoT alapú eszközök száma biztosan növekedni fog. Most viszonylag friss 2020-as cikkekből származó források alapján hasonlítsuk össze, hogy tényleg megfelelőek voltak -e az

előrejelzések. Egy 2020. januári cikk szerint az aktív IoT eszközök száma 2019-ben már 26.66 milliárd volt, 2020 végére ezt a számot 31 milliárdra becsülik, míg 2025-re 75 milliárd aktív eszközt jósolnak [16]. Tehát a számok jóval felülmúlják Gartner jóslatát. 2020 végére az IoT -ból származó jövedelem elérheti az 1.29 billió dollárt. A 6. ábra egy globális IoT jövedelmi előrejelzést mutat vertikális piaci felbontásban.



6. ábra: Globális IoT jövedelem előrejelzés 2020-ra vertikális piaci felbontásban [17]

2024-re az egészségügyi IoT-ból származó jövedelem elérheti a 14 milliárd dollárt. 2020 végére azt jósolják, hogy a vállalatok 93% -a és a gyárak 80% fog IoT technológiát alkalmazni. Az autók 90% fog IoT technológián keresztül a hálózatra csatlakozni. Egy 2020. június 18-án publikált cikk a COVID-19 vírus hatását vizsgálja az IoT piaci növekedésében [18]. A cikk szerint a 2019-ben jósolt 14.9%-os éves jövedelem növekedés IoT alapú eszközökre a vírus miatt 2020-ban csak 8.2%-os lett, de 2021-re újra átlépi a 10%-os határt. A cikk azt is megemlíti, hogy az egészségügynél, biztosításnál és az oktatásnál a legerősebb ez a növekedés 2020-ban. Az IoT-t alkalmazó régiók közül továbbra is az ipar, gyártás vezeti a rangsort, míg a közlekedés csak a második helyen van [19]. A 7. ábra az IoT alkalmazását mutatja régiókra felbontva.



7. ábra: IoT -t alkalmazó régiók rangsora 2020-ban [19]

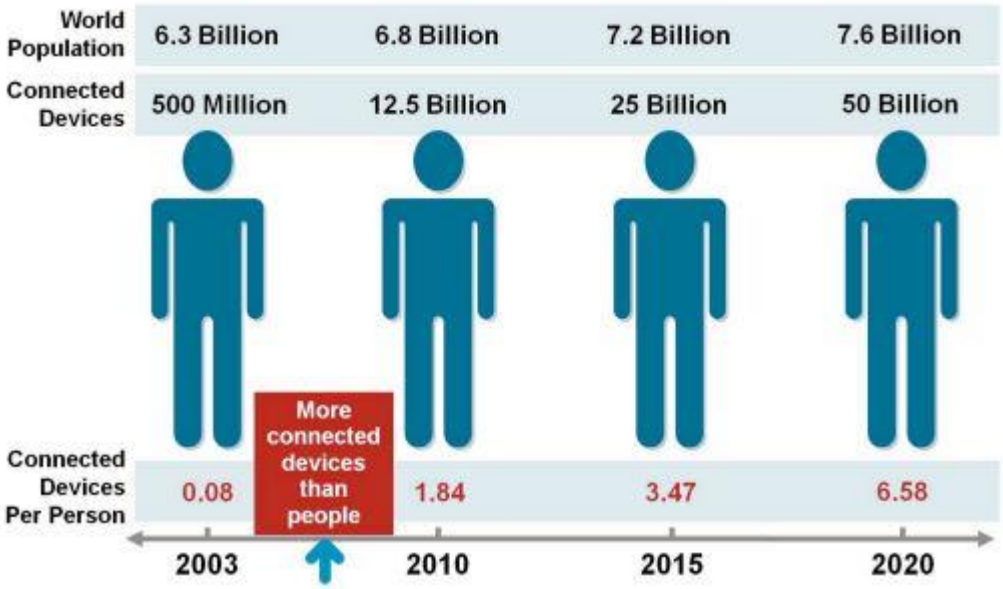
Több IoT -val foglalkozó előrejelzést is olvastam és a becslések sok esetben eltérnek egymástól. Egy következtetést azonban biztosan levonhatunk, hogy az IoT piaca és eszközeinek száma eddig is növekedett és továbbra is biztos növekedést mutat.

3.2 Az IoT

3.2.1 Bevezetés

Mit is rejt magában az Internet of Things elnevezés? Az IoT röviden fogalmazva azokat a fizikai eszközöket jelenti, amelyek egyedi azonosítóval rendelkeznek és ezen eszközök az interneten keresztül képesek külső beavatkozás nélkül egymással kommunikálni [20]. Ez a fizikai eszköz gyakorlatilag bármi lehet, ha tudunk hozzá IP címet rendelni és ha az eszköz képes a hálózaton keresztüli kommunikációra. Néhány jellemző példa lehet a fizikai eszközökre a beágyazott rendszerrel rendelkező eszközök (autó, mosógép, kávéfőző, termosztát), okostelefonok, számítógépek, szenzorok, élőlények, melyekbe chipet ültettek be. Az IoT kialakulása egészen a 90-es évekig vezethető vissza, amikor Kevin Ashton a Massechusettsi Műszaki Egyetem professzoraival és kutatóival együtt elkezdett foglalkozni az RFID technológiával és megalapították az Auto-ID Center kutatócsoportot. Az új technológiát 1999-ben prezentálta a Procter & Gamble vállalatnak, ahol először kerül említésre az Internet of Things elnevezés. Ugyanakkor az IBSG az IoT kialakulását arra az adott időpontra datálja, amikor ezek az eszközök meghaladták az emberiség létszámát [21]. Az IBSG

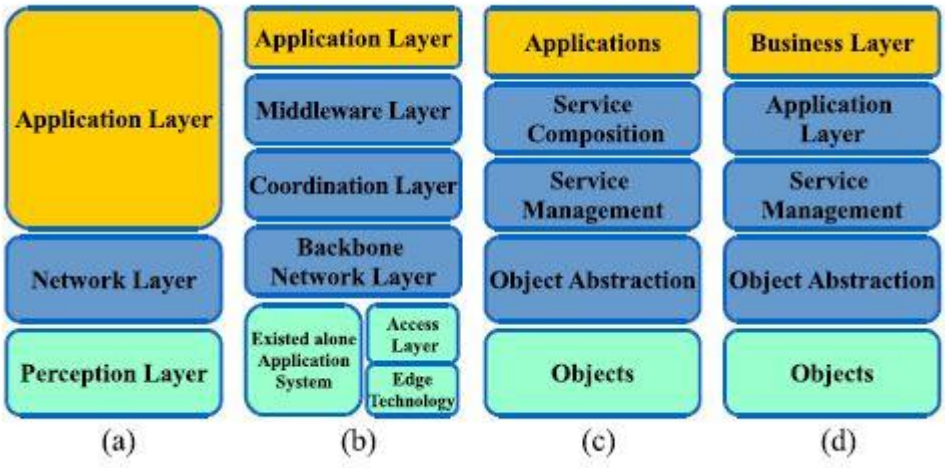
szerint ez az időpont körülbelül 2008-2009 közötti időszakra tehető. A 8. ábra egy összehasonlítást szemléltet az aktív IoT eszközök száma és a világ népessége között. Az ábrán látható, hogy mikor lépte át az aktív IoT eszközök száma a világ népességének számát. Az ábra előrejelzést is tartalmaz, amely a világ populációját és az aktív IoT eszközök számát jósolja meg 2020 -ig. A 2011-ben készült felmérések alapján jóval optimistábban becsülték meg az IoT eszközök várható mennyiségét, hiszen körülbelül 20 milliárd eszközzel túlbecsülték.



8. ábra: Aktív IoT eszközök száma és a világ népességének összehasonlítása [21]

3.2.2 Az IoT architektúrája [2]

A több milliárd IoT eszköz internethez történő kapcsolódásához elengedhetetlenné vált egy olyan rugalmasan és rétegzett architektúra létrehozása, amely képes ekkora mennyiségű, különböző eszközt egységesen kezelni. Az alap architektúra 3 réteget tartalmaz, ezek a Preception, Network, Application régetek. Ebből az alap struktúrából kiindulva az IoT-hoz legalkalmasabb architektúra az 5 rétegű architektúra, amelyet a 9. ábrán is láthatunk. Ezek az Objects, Object Abstraction, Service Managment, Application és Business rétegek. A következőkben ezekről a rétegekről írok röviden.



9. ábra: Az IoT architektúrái. (a) 3-rétegű, (b) middle-ware alapú, (c) SOA alapú, (d) 5-rétegű [2]

Az Objects rétegbe tartoznak maguk a szenzorok, amelyek a környezet információt gyűjtik be. A szenzorok például hőmérsékletet, páratartalmat, gyorsulást vagy mozgást érzékelhetnek. Ebbe a rétegbe tartoznak még a működtető szerkezetek, másnéven aktuátorok, amelyek a feldolgozott információk alapján megfelelő módon avatkoznak be a környezetbe. Az aktuátorok segítségével például bekapcsolható a fűtés a hőmérséklet csökkenés hatására. Ezt a réteget szokták még Perception, azaz Érzékelés rétegnek is nevezni. A réteg feladatai közé tartozik még az adatok digitalizálása és továbbítása biztonságos csatornákon az Object Abstraction réteg felé. A big data adatai, amelyet az IoT szenzorai hoznak létre ebben a rétegben jönnek létre.

Az Object Abstraction rétegben történik az adatok szállítása a Service Management réteg felé. Lényeges, hogy az adatok biztonságos csatornán közlekedjenek, így azokat illetéktelen személyek nem tudják megszerezni. Az adatok többféle technológiával is továbbíthatók. Ilyen technológiák például az RFID, 3G, GSM, UMTS, WiFi, Bluetooth Low Energy, Infrared és a ZigBee. Továbbá egyéb funkciók, mint a cloud computing is ebben a rétegben történnek.

A Service Management réteg felel a szolgáltatás és a szolgáltatást igénylő eszköz párosításáért. A párosítás cím és név alapján történik. Ez a réteg lehetővé teszi az IoT alkalmazást fejlesztők számára, hogy a különböző típusú eszközöket hardver specifikusság figyelembevétele nélkül használják. Ebben a rétegben történik a beérkezett információk feldolgozása, a feldolgozott adatok alapján a döntéshozatal és az igényelt szolgáltatás továbbítása a hálózaton.

Az Application réteg biztosítja az ügyfelek számára az igényelt szolgáltatásokat. Például, ha egy ügyfél lekérdezi a hőmérsékletet vagy a páratartalmat az adott helyiségben, akkor az információ alkalmazás rétegen keresztül jut el az ügyfélhez. Az ügyfél lehet ember, de akár eszköz is, tekintettel az M2M, azaz a gépek közötti kommunikációra és akár automatizált rendszerekre is. Az alkalmazás réteg többféle vertikális piacot is lefed, olyan területeket, mint például az okosotthonok, okos épületek, szállítóipar, ipari automatizáció, egészségügy.

Az Business réteg kezeli az IoT rendszer aktivitásait, szolgáltatásait. A réteg lényege, hogy üzleti modellt, grafikus kimutatásokat és folyamatábrákat készítsen az alkalmazás rétegből beérkezett adatok alapján. Az Business réteg arra is használható, hogy analizálja, monitorozza az IoT alapú rendszereket és hogy hozzájáruljon az IoT rendszerek és szolgáltatások fejlesztéséhez. Az Business réteg lehetővé teszi a big data analizálása alapján történő döntés hozatalt.

A rétegek vizsgálatából és az architektúra egyszerűségéből adódóan az 5-rétegű modellt lehet a legjobban alkalmazni az IoT alkalmazásokban.

3.2.3 Az IoT építő elemei [2]

Az IoT elemeinek vizsgálata segít abban, hogy jobban megértsük az IoT valódi jelentését. A következőkben az IoT hat fő részét vizsgáljuk meg, amelyek az IoT funkcionalitását mutatják be. Ez a hat elem az Identification, Sensing, Communication, Computation, Services és a Semantics, amelyek 10. ábrán is láthatóak.

10. ábra: Az IoT elemei [2]

Az Identification rész alapvető fontosságú, hiszen az IoT szolgáltatásait itt nevezik meg és párosítják a szolgáltatás igénylésekkel. Többféle identifikációs metódus is létezik, ilyenek például az EPC vagy a uCode. Kritikus pontnak számít továbbá az IoT eszközök címzése, ugyanis különbséget kell tenni az eszköz ID-je és címe között. Az eszköz ID alatt, magát az eszköz megnevezését értjük, például egy hőmérő szenzor neve lehet „T1”, míg az eszköz címe, az eszköz kommunikációs hálózatban szereplő

címére utal. Ilyen címzési metódusok például az IPv4 és IPv6. Mivel az eszközök identifikációja globálisan eltérő lehet, ezért van szükség az eszközök címzésére, amely segít egy eszközök egyedi azonosításában. Tehát az identifikációs metódusok biztosítják az eszközeink egyedi azonosítását a hálózaton.

A Sensing azt jelenti, hogy információkat gyűjtünk adott eszközről és ezeket az adatokat egy adatbázisba küldjük. Az összegyűjtött adatokat megvizsgálja a rendszer és ennek megfelelően biztosítja a szolgáltatást. Az IoT érzékelők lehetnek okos szenzorok, aktuátorok, hordozható érzékelő eszközök. Vannak olyan cégek, mint például a Wemo, revolv (2018-ban megszűnt ezen a néven, új neve: nest), SmartThings (anyaszervezet: Samsung Electornics), akik olyan okos felületeket, mobil alkalmazásokat ajánlanak, amikkel képesek vagyunk okos eszközeinket a saját telefonunkról irányítani és megfigyelni. A Single Board Computers (SBCs), olyan apró számítógépek, melyek egyetlen nyomtatott áramkörön elférnek. Ilyen SBC -k például a Raspberry Pi, Arduino UNO, BeagleBone Black. Ezeken az SBC -ken találhatóak beépített szenzorok, beépített TCP/IP protokoll struktúra és egyéb biztonsági funkciók. Az ilyen eszközök általában egy központi ügyintéző portálhoz csatlakoznak, ahonnan biztosítják az információkat az ügyfelek számára. Ezeket az SBC-ket használják tipikusan az IoT termékek megvalósításához. A 11. ábra egy Raspberry Pi 4 SBC -t szemléltet.

11. ábra: Raspberry Pi 4 model B [22]

A Communication rész az IoT kommunikációs technológiáiról szól. Ezek a technológiák kötik össze a különféle eszközöket, hogy megfelelő okos szolgáltatásokat tudjanak biztosítani. Az IoT node-ok RF érzékenysége lehetőleg magas legyen, hogy még a zajos, zavarral és interferenciával terhelt csatornákban is képesek legyenek a kommunikációra. Az IoT-ben használt tipikus kommunikációs protokollok lehetnek WiFi, Bluetooth, IEEE802.15.4, Z-wave és LTE-Advanced. Egyéb speciális környezetben használt protokollok lehetnek RFID, Near Field Communication (NFC) és az Ultra-Wide Bandwidth (UWB). A Bluetooth a 4.1 -es verziójától kezdve támogatja az IoT -t, de a modern IoT eszközök már a Bluetooth 5 technológiát is képesek használni. A 2020. januári CES -en a the Bluetooth SIG által újonnan bemutatott Bluetooth 5.2 technológia a jelenlegi legújabb [23]. A SiliconLabs által megalkotott EFR32BG22 System on Chip (SoC) már támogatja az energia-barát 5.2-es Bluetooth technológiát, amely ideális kommunikációt biztosít az IoT eszközök számára [24].

A Computation rész a feldolgozó egységekről szól, amely lehet mikrokontroller, mikroprocesszor, SOC, Field-programmable gate-array (FPGA) és azokon alkalmazott szoftverekről, amelyek az IoT agyát, számítási képességét képviselik. Többféle különböző hardver platformot fejlesztettek ki az IoT alkalmazásához. Ilyen platformok például az Arduino, Raspberry Pi, BeagleBone, UDOO, FriendlyARM és az Intel Galileo. Továbbá több szoftver platformot is kifejlesztettek az IoT funkciók biztosításáért. Az Operating Systems (OS) operációs rendszerek létfontosságúak a feldolgozó egységek számára, hiszen a bekapcsolásuktól kezdve folyamatosan futnak az eszközön. Többféle valós-idejű operációs rendszer létezik, ezeket angolul real-time operating systems (RTOS) -nak nevezik. Ezeket az operációs rendszereket valós-idejű IoT alkalmazások fejlesztéséhez készítették. Példaképp a Contiki RTOS széles körben elterjedt az IoT fejlesztők körében, mivel a fejlesztők számára a Contiki biztosított egy szimulációs programot a Cooja-t. A Contiki legújabb verziója a Contiki-NG. Egyéb megemlítendő valós-idejű operációs rendszer a TinyOS, LiteOS, RiotOS és az Android. Az RTOS-okról a 12. ábrán láthatunk egy összehasonlítást.

Operating System	Language Support	Minimum Memory (KB)	Event-based Programming	Multi-threading	Dynamic Memory
TinyOS	nesC	1	Yes	Partial	Yes
Contiki	C	2	Yes	Yes	Yes
LiteOS	C	4	Yes	Yes	Yes
Riot OS	C/C++	1.5	No	Yes	Yes
Android	Java	-	Yes	Yes	Yes

12. ábra: Valós-idejű operációs rendszerek összehasonlítása [2]

A 12. ábrán látható, hogy a legtöbb RTOS a C nyelvet támogatja az Android kivételével. A Riot OS kivételével mindegyik operációs rendszer lehetőséget nyújt eseményalapú programozásra. Mindegyik OS legalább részben támogatja a több szálon történő feladatvégrehajtást.

A cloud platforms, avagy felhő platformok az IoT egy másik típusú számítási ága. Lényege, hogy az adatokat felküldjük a felhőbe, amelyeket big data szerverek tárolnak, majd ezután valós időben feldolgozza az adatokat, hogy egyéb végfelhasználók is hasznát vegyék az információknak. Sok ingyenes felhő platform létezik az IoT alkalmazások számára. Ilyenek például az Axonize, Blynk és a Fogwing, de a több szolgáltatást biztosító fizetős platformok is biztosítanak ingyenes próbaidőszakot az érdeklődők számára.

A Services részben az IoT szolgáltatásokat kategorizáljuk. Az IoT szolgáltatásokat 4 osztályra bontjuk, ezek a Identity-related Services, Information Aggregation Services, Collaborative-Aware Services és az Ubiquitous Services. Az Identity-related szolgáltatások a legalapvetőbb és legfontosabb szolgáltatások, amelyeket a többi osztály is felhasználja. Minden alkalmazás, amely egy valós eszközt akar virtualizálni, annak azonosítania kell azt az adott eszközt. Az Information Aggregation szolgáltatás gyűjti be és összesíti a szenzorok által mért nyers adatokat, amelyeket fel kell dolgozni és jelenteni kell az IoT alkalmazásnak. A Collaborative-Aware szolgáltatás az Information Aggregation Services osztály tetején helyezkedik el. A szolgáltatás lényege, hogy a kapott adatok alapján döntést hoz és ennek megfelelően avatkozik be. A legfelső osztály az Ubiquitous Services, amelynek lényege, hogy ilyen Collaborative-Aware szolgáltatást nyújtson bárkinek, bárhol, bármikor. Minden IoT

alkalmazás legfőbb célja, hogy elérje ezt az Ubiquitos Services szintet. Most néhány példát sorolok fel, hogy az egyes területek melyik kategóriába sorolhatóak. Az okos egészségügy és Smart Grid az Information Aggregation kategóriába tartoznak. Az okosotthon, okos épület, ipari automatizáció és az okos szállítóipar a Collaborative-Aware kategóriához van közelebb. Az okos város pedig már az Ubiquitous Services kategóriába sorolható.

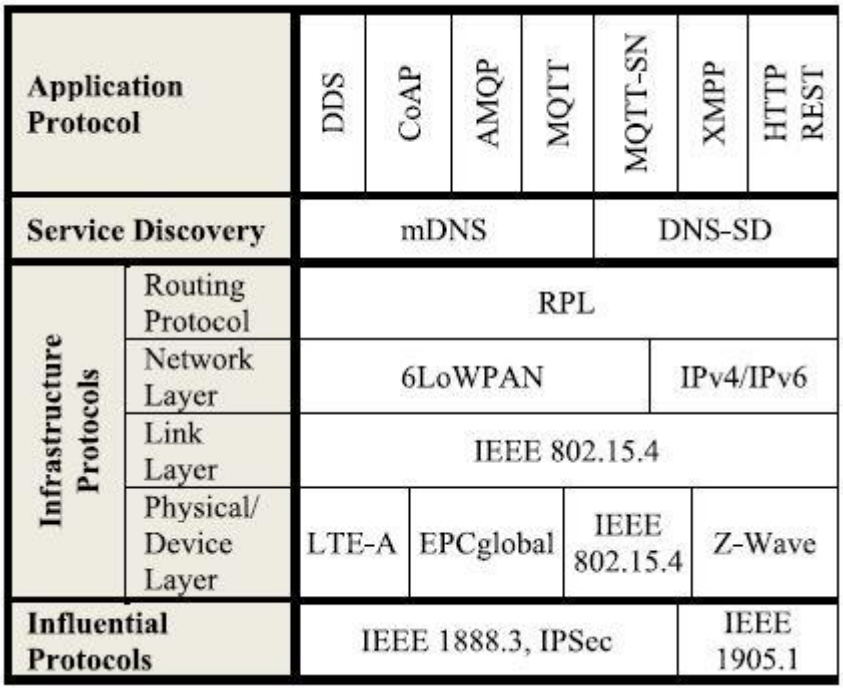
A Semantics rész lényege az IoT-ban, hogy képes legyen lényeges tudást kivonni különböző gépekből, hogy azok vagy más eszközök a tudást felhasználva a lehető legpontosabb szolgáltatást biztosítsák. A tudás kivonásába beletartozik a felismerő képesség, források használata és információk modellezése. Ide tartozik, hogy az IoT eszköz felismeri a feladatot vagy a kialakult problémát és azt analizálva képes optimális szolgáltatást nyújtani. Az Efficient XML Interchange (EXI) egy Semantic Web technológia, amelyet a World Wide Web konzorcium (W3C) adoptált direkt erre a célra. A 13. ábra az előbbiekben tárgyaltak összefoglalása kategóriák szerint.

IoT Elements		Samples
Identification	**Naming**	EPC, uCode
	Addressing	IPv4, IPv6
Sensing		Smart Sensors, Wearable sensing devices, Embedded sensors, Actuators, RFID tag
Communication		RFID, NFC, UWB, Bluetooth, BLE, IEEE 802.15.4, Z-Wave, WiFi, WiFiDirect, , LTE-A
Computation	**Hardware**	SmartThings, Arduino, Phidgets, Intel Galileo, Raspberry Pi, Gadgeteer, BeagleBone, Cubieboard, Smart Phones
	Software	OS (Contiki, TinyOS, LiteOS, Riot OS, Android); Cloud (Nimbits, Hadoop, etc.)
Service		Identity-related (shipping), Information Aggregation (smart grid), Collaborative-Aware (smart home), Ubiquitous (smart city)
Semantic		RDF, OWL, EXI

13. ábra: Az IoT építő elemei és azok technológiái [2]

3.2.4 Az IoT szabványos protokolljai [2]

Az IoT általános szabványai azért születtek meg, hogy segítség és megkönnyítsék az IoT területen dolgozó fejlesztők és szolgáltatók munkáját. Ezen szabványban szereplő protokollokat a W3C, az Internet Engineering Task Force (IETF), az EPCglobal, az Institute of Electrical and Electrics Engineers (IEEE) és az European Telecommunications Standards Institute (ETSI) közösen határozták meg. A 14. ábrán ezekről az általános szabványokról láthatunk egy összesítést a lényegesebb protokollokkal együtt.

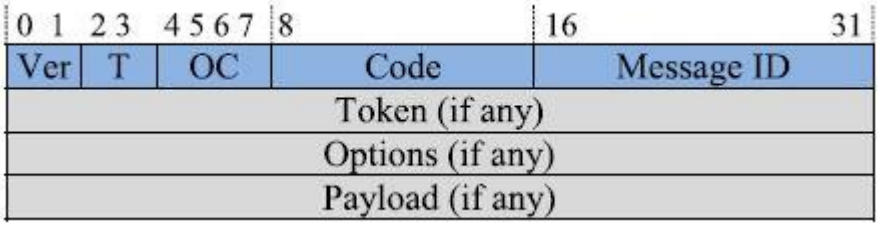


14. ábra: Az IoT szabványos protokolljainak összesítése kategóriák szerint [2]

A protokollokat 4 fő kategóriába osztották be. Ezek a kategóriák az Application Protocols, a Service Discovery, az Infrastructure Protocols és az Influential Protocols. A következő sorok néhány lényeges protokoll funkcionalítását mutatják be.

Az Application Protocols, avagy alkalmazás protokollok kategóriában található a Constrained Application Protocol (CoAP), amelyet IoT alkalmazásokhoz fejlesztettek ki. A CoAP olyan hálózati átviteli protokoll, amely REpresentational State Transfer (REST) alapú. A REST leegyszerűsíti az adatátvitelt kliens és szerver között. A REST egy állapot nélküli szerver-kliens architektúrára épül, amelyet mobil alkalmazások és közösségi hálózatok használnak, mert segít elkerülni a többértelműséget a HTTP féle GET, POST, PUT és DELETE parancsokkal. A REST-el ellentétben a CoAP nem TCP-

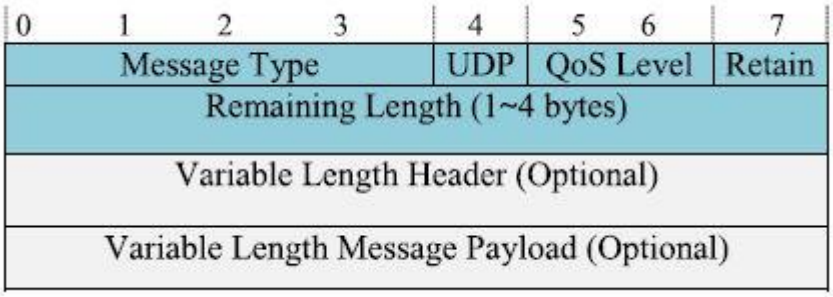
t, hanem UDP -t használ, amely előnyt jelent az IoT alkalmazásoknál. Egy CoAP üzenet tipikusan 10-20 bájtból áll. Az üzenet alakja a 15. ábrán látható.



15. ábra: A CoAP üzenet alakja [2]

A header részei a következők. A „ver” rövidítés a CoAP verzióra utal, a „T” a tranzakció típusa, az „OC” azaz Option Count az opció szám és a „Code” részbe a kérés metódus neve kerül vagy a válasz kódja. Például a GET kérés metódus kódja az 1. A CoAP néhány fontos jellegzetessége a forrás megfigyelés, a blokk-szerű forrás átvitel, a forrás felderítés, a HTTP interakció és a biztonság.

A Message Queue Telemetry Transport (MQTT), olyan üzenő protokoll, amely célja a beágyazott rendszerek és hálózatok összekapcsolása. Az MQTT egy optimális kapcsolatteremtő protokoll az IoT és a Machine-to-Machine (M2M) számára, mivel útvonal választó mechanizmusa lehet egy az egyhez, egy a többhöz és több a többhöz. Az MQTT a publish/subscribe üzenetküldési módszert használja, hogy rugalmas adatátvitelt és egyszerűséget biztosítson. Az üzenetküldési módszernek három komponense van. A subscriber feliratkozik egy témára (topic) és ha a publisher ebben a témában publikál, akkor a bróker értesíti a subscriber-t, hogy új publikáció jelent meg a témában. Ezután a publikáló továbbítja a kívánt információt a feliratkozónak a brókeren keresztül. A brókernek másik fontos feladata, hogy védelmi okokból megvizsgálja a feliratkozó és publikáló jogosultságait. Az MQTT -t gyakran használják egészségügyi, monitorozó és értesítésküldő alkalmazásokban. Az MQTT használható olcsó, alacsony teljesítményű, kevés memóriájú eszközökben és sebezhető, alacsony sávszélességű hálózatokban, emiatt ideális üzenetküldési protokoll az IoT-s alkalmazások számára és M2M kommunikációhoz. Az MQTT üzenet alakja a 16. ábrán látható.

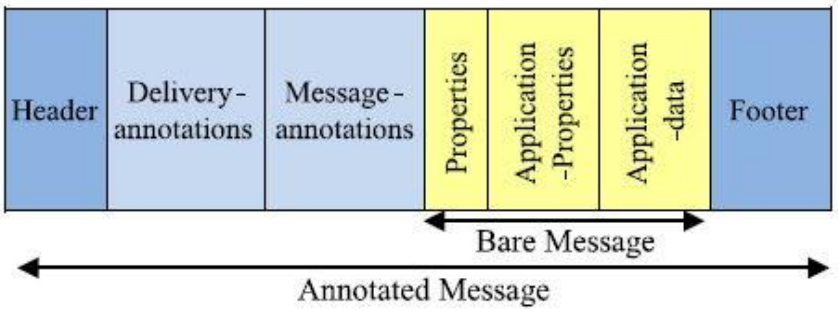


16. ábra: Az MQTT üzenet alakja [2]

Az üzenet első két bájtja fix. A fejlécben található a Message Type rész, amelybe az üzenet típusát kell megadni. Ez lehet például CONNECT (1), CONNACK (2), PUBLISH (3), SUBSCRIBE (8). A DUP bittel jelezhetjük, hogy az üzenet egy duplikáció és a fogadó már lehet, hogy megkapta egyszer. A QoS Level részben megadható, a Quality of Service (QoS) szintje. A QoS-nek három szintje van, amelyekkel jelezhető a fogadó számára, hogy milyen minőségben kívánjuk küldeni az üzenetet. A Retain részben megadható, hogy a szerver megtartsa-e a publisher üzenetét, hogy azt más subscriber-nek is elküldhessük. A 2. bájt a Remaining Field részt tartalmazza, amelyben megadható az üzenet maradék részének a hossza, azaz az opcionális részek hossza.

Az Extensible Messaging and Presence Protocol (XMPP) lehetővé teszi az azonnali üzenetküldést kettő vagy több fél között operációs rendszertől függetlenül. Az XMPP -t főként csoportos csevegésekben, hang- és videóhívásokban használják, így nem releváns választás a rendszerem számára.

Az Advanced Message Queuing Procotol (AMQP) egy üzenet-orientált protokoll, amely a megbízható kommunikációt a QoS szintekkel biztosítja. Az AMQP -nak megbízható szállítási protokoll kell, mint a TCP. Az AMQP kommunikációt két fő komponens kezeli a brókeren belül, ezek az Exchange és Queue komponensek. Az Exchange szabályok és feltételek alapján választja ki, hogy melyik Queue-ba (várakozási sor) kell a továbbítandó üzenetet rakni. Az üzeneteket a Queue -k tárolják, amíg a fogadó meg nem kapja azokat. Az AMQP a pont-pont kapcsolaton kívül támogatja a publish/subscribe kapcsolatot is. Az AMQP -nek a szállító rétegen felül egy üzenő rétege is van, amelynek kommunikációs formája frame alapú. Az AMQP üzenet alakját a 17. ábra szemlélteti.



17. ábra: Az AMQP üzenet alakja [2]

A header tartalmazza az üzenet szállításához szükséges paramétereket.

A Data Distribution Service (DDS) egy publish/subscribe protokoll valós idejű M2M kommunikációhoz. Az MQTT-hez és AMQP-hez képest a DDS-nek bróker nélküli az architektúrája és multicasting-ot használ, hogy kitűnő QoS -t és magas megbízhatóságot biztosítson az alkalmazások számára. A DDS architektúrának két rétege van. A Data-Centric Publish-Subscribe (DCPS) réteg felel az információ átadásáért, míg a Data-Local Reconstuction Layer (DLRL) egyfajta opcionális réteg, amely interfészként szolgál a DCPS funkciók számára.

Összességében elmondható, hogy az egyes alkalmazás rétegbeli protokollok más és más szituációkban kiemelkedőek. Például, ha kis méretű üzenetekkel és 25% alatti csomagveszteséggel dolgozunk, akkor a CoAP kevesebb extra forgalmat eredményez az MQTT -hez képest, de alacsonyabb csomagveszteségnél már az MQTT a kiemelkedő. Az Application réteg protokolljainak összehasonlítását a 18. ábrán láthatjuk.

Application Protocol	RESTful	Transport	Publish/ Subscribe	Request/ Response	Security	QoS	Header Size (Byte)
COAP	✓	UDP	✓	✓	DTLS	✓	4
MQTT	×	TCP	✓	×	SSL	✓	2
MQTT-SN	×	TCP	✓	×	SSL	✓	2
XMPP	×	TCP	✓	✓	SSL	×	-
AMQP	×	TCP	✓	×	SSL	✓	8
DDS	×	TCP UDP	✓	×	SSL DTLS	✓	-
HTTP	✓	TCP	×	✓	SSL	×	-

18. ábra: Az Application kategória protokolljainak összehasonlítása [2]

A 18. ábrán látható, hogy a COAP és DDS kivételével mindegyik protokoll TCP-t használ. A HTTP protokoll az egyetlen, amelyikben a publish/subscribe üzenetküldési módszer nem alkalmazható.

A következő rész a Service Discovery kategória protokolljairól szól. Az IoT magas skálázhatósága érdekében szükség van egy forrás kezelő mechanizmusra, amely felfedezi és regisztrálja a forrásokat és szolgáltatásokat, hatékony és dinamikus módon. Az egyik kiemelkedő protokoll ebben a kategóriában a multicast DNS (mDNS), amely jó választás lehet az IoT-s eszközök számára, hiszen nincs szükség manuális újra konfigurálásra vagy extra adminisztrációra az eszközök kezeléséhez. Az mDNS infrastruktúra nélkül is működőképes és az infrastruktúra meghibásodása esetén is tovább tud dolgozni. Az mDNS kapcsolat létrehozása egy multicast üzenettel kezdődik, az üzenetben szereplő névvel megegyező kliens visszaküldi válaszként az IP címét a feladónak. A DNS Service Discovery (DNS-SD) protokoll képes felismerni a rendelkezésre álló IoT eszközök forrásait és szolgáltatásait.

Összességében elmondható, hogy az IoT-nek egy olyan architektúrára van szüksége, amely biztosítja az IoT eszközök számára a zökkenőmentes felcsatlakozást a rendszerbe és lekapcsolódást a rendszerből anélkül, hogy a rendszer viselkedését megzavarnák.

Az Infrastructure Protocols kategóriába tartozik az RPL protokoll. A Routing Protocol for Low Power and Lossy Networks (RPL) egy IPv6 alapú kapcsolat független útválasztó protokoll. Az RPL képes veszteséges kapcsolatokon minimális útvonalválasztási feltételek mellett nagy méretű topológia kialakítására. Támogatja az egy-egy, egy-több, több-egy pont kapcsolatot. Az RPL a Destination Oriented Directed Acyclic Graph (DODAG) topológiát alkalmazza. Az RTL négy féle kontroll üzenetet alkalmaz. A DODAG Information Object (DIO) üzenet tartalmazza a csomópont aktuális rangját, meghatározza az egyes csomópontok távolságát a gyökér csomóponttól és kiválasztja a kívánt szülőhöz vezető utat. Egy másik üzenet típus a Destination Advertisement Object (DAO), amely a fel és lefele vezető adatforgalmat segíti, hogy szülőkön keresztül az adat elérje célját. A harmadik üzenet típus a DODAG Information Solicitation (DIS), amelyet a csomópont arra használ, hogy DIO üzenetet szerezzen egy elérhető szomszédos csomóponttól. A negyedik üzenet típus a DAO Acknowledgment (DAO-Ack), amelyet a címzett csomópont küld válasz üzenetként a DAO üzenetre.

Az IPv6 over Low Power Wireless Personal Area Networks (6LowPAN) protokollt, azért hozták létre, hogy az IEEE 802.15.4 tulajdonságaihoz az IPv6 internetprotokollt hozzárendelhessék. A szabvány egy fejléc tömörítést biztosít, hogy csökkentsék az átvitel túlcsordulását és a töredezettséget, így teljesítve az IPv6 Maximum Transmission Unit (MTU) feltételeit.

Az IEEE 802.15.4 protokollt azért hozták létre, hogy szabványosítsanak egy alréteget a Medium Access Control (MAC) -nak és egy fizikai réteget (PHY) a low-rate wireless private area networks (LR-WPAN) -nak, amely olyan WPAN -t jelent, ahol alacsony az adatátviteli sebesség. Gyakran alkalmazzák IoT, M2M és wireless sensor network (WSN) alkalmazásokban. Megbízható kommunikációt biztosít, különböző platformokon is használható és nagy mennyiségű csomópont kezelésére alkalmas, ugyanakkor szolgáltatásai nagy biztonságot, titkosítási lehetőséget és hitelesítési lehetőséget is biztosítanak. Negatívumai közé tartozik, hogy nem biztosít QoS garanciát. A Zigbee protokoll alapja is ez. A lehetséges adatforgalmi ütközések elkerülésére érdekében a MAC a CSMA/CA protokollt alkalmazza. Az IEEE 802.15.4 szabvány kétféle hálózati csomópontot támogat. Az egyik a Full Function Devices (FFD) a másik a Reduced Function Devices (RFD). Az IEEE 802.15.4 hálózatok a csillag-, peer-to-peer és a fa topológiát alkalmazzák.

A Bluetooth Low-Energy (BLE) protokoll minimális energiahasználattal működik. A sima Bluetooth verzióhoz képest a BLE, akár tízszer akkora hatótávolságot is lefed (100m), miközben az adatátvitel késleltetése tizenötször rövidebb. Az adatátvitel alatt a BLE fogyasztása 0.01mW és 10mW közé esik. Ezekkel a tulajdonságokkal a BLE jól használható az IoT alkalmazásokban.

Az Electronic Product Code global (EPCglobal) protokollban az EPC egy egyedi azonosító, amit egy RFID tagban tárolnak az eszközök azonosítására. Az architektúra az internet alapú RFID technológiát használja, amely egy bíztató irány az IoT jövőjére nézve a szabad hozzáférhetősége, skálázhatósága, átjárhatósága és megbízhatósága miatt. Az RFID rendszer két fő komponense a radio signal transponder (tag) és a tag reader.

A Long Term Evolution-Advanced (LTE-A) szabvány összefogja azokat a mobil kommunikációs protokollokat, amelyek jól használhatóak a Machine-Type Communications (MTC) -nél és IoT infrastruktúráknál. Ilyen szabványt használnak például okosvárosokban, ahol fontos, hogy az infrastruktúra hosszú távon is tartós

maradjon. Az LTE-A hálózat architektúrája a Core Network (CN) -ből és a Radio Access Network (RAN) -ból áll. A CN kezeli a mobil eszközöket és az IP csomagforgalmat. A RAN kezeli a vezetéknélküli kommunikációt, a rádió elérést.

A Z-Wave egy alacsony teljesítményű, vezetéknélküli kommunikációs protokoll Home Automation Networks (HAN) számára, azaz otthoni automatizálási hálózat számára. Széles körben használják okosotthonokat távolról vezérlő alkalmazásokban. A Z-Wave hatótávolsága körülbelül 30 méter, pont-pont kommunikációt használ és főleg olyan helyeken alkalmazzák, ahol kevés adatforgalom jellemző, mint például világítás, háztartási gépek, HVAC irányítására. Az architektúrája controller és slave csomópontokból áll. A 19. ábrán egy összehasonlítás látható a fent említett PHY protokollokról.

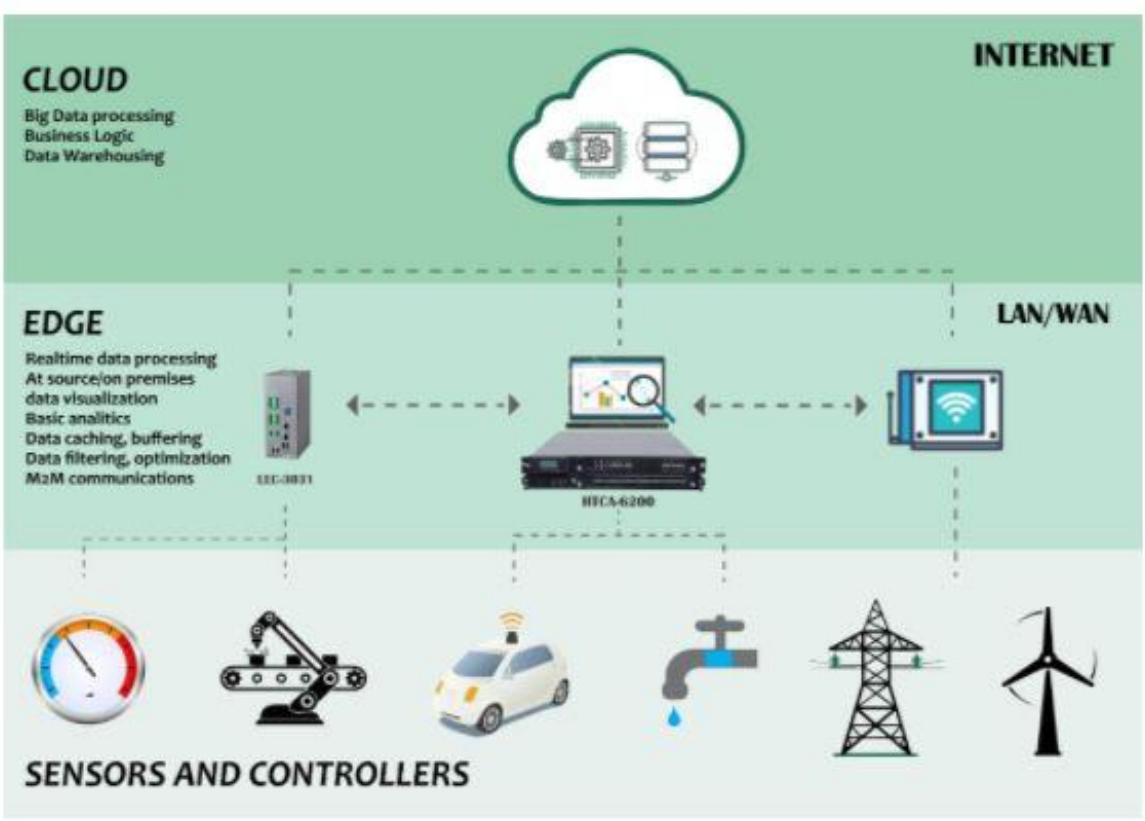
PHY Protocol	Spreading Technique	Radio Band (MHz)	MAC Access	Data Rate (bps)	Scalability
IEEE 802.15.4	DSSS	868/ 915/ 2400	TDMA, CSMA/CA	20/ 40/ 250 K	65K nodes
BLE	FHSS	2400	TDMA	1024 K	5917 slaves
EPCglobal	DS-CDMA	860~960	ALOHA	Varies 5~640 K	-
LTE-A	Multiple CC	varies	OFDMA	1G (up), 500M (down)	-
Z-Wave	-	868/908/ 2400	CSMA/CA	40K	232 nodes

19. ábra: Összehasonlítás a PHY protokollokról [2]

3.2.5 Big data, cloud és fog computing

A big data egyik lehetséges definíciója Gartner szerint: - „Big data is data that contains greater variety arriving in increasing volumes and with ever-higher velocity. This is known as the three Vs.” [25]. Magyarra fordítva körülbelül azt jelenti, hogy a big data egy gyorsan növekvő, hatalmas, komplex adatmennyiség. Ekkora adatmennyiséget az általánosan használt hardverek és szoftverek nem képesek hatékonyan kezelni. Az IoT eszközeinek adatait tárolni kell, fel kell dolgozni és a hasznos adatokat ki kell nyerni az adathalmazból, mindezt valós időben. A cloud

computing (CC), azaz a felhőalapú számítástechnika lehet a megoldás ennek a problémának a kezelésére. A could computing fogalma a National Institute of Standards and Technology (NIST) szerint: - „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” [26]. A CC, tehát egy igény szerint elérhető, felhő alapú szolgáltatás. A szolgáltatások közé tartozik a felhő alapú adattárolás és a felhő alapú erőforrások használata. A CC alkalmazása az IoT -ban nem egyszerű. A cloud platformoknak többféle problémával is meg kell küzdeniük, amelyek a szinkronizáció, szabványosítás, kiegyensúlyozottság, megbízhatóság, kezelés és fejleszthetőség [2]. A cloud platform cégek közül is kiemelkedőek a Thingworx, Microsoft Azure IoT Suite és az Amazon Web Services szolgáltatásai [27]. A fog computing kibővíti a cloud computing szolgáltatásait a hálózat széleire, amelyet edge computingnak is neveznek [28]. A hálózat szélei alatt azt értjük, hogy a CC szolgáltatások a fizikai eszköz közelében kerülnek elvégzésre, ezzel úgymond levéve a terhet az adatközpontokról, alacsony késleltetést biztosítva és nagy mennyiségű szélessávot megtakarítva [29]. A fog computing optimális választás lehet az IoT számára a következő jellemzők miatt. A fog források okos eszköz közeli elhelyezkedése, a fog mikro központok olcsó telepítése és skálázhatósága, a fog szolgáltatások rugalmassága, megismételhetősége és valós idejűsége, a fog források mobilitása, a fog szabványosítása és részbeni adatfeldolgozási képessége miatt. A 20. ábra a fog computing-ot alkalmazó IoT infrastuktúrát szemlélteti.



20. ábra: A fog computing infrastruktúrája [29]

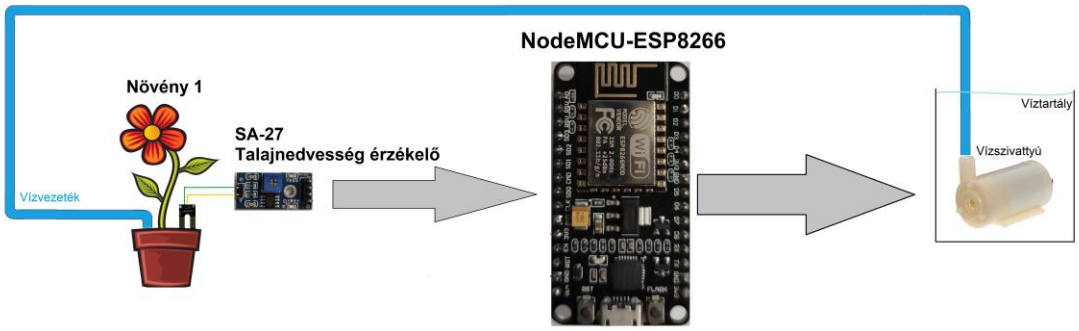
A 20. ábrán látható, hogy az edge computing közelebb helyezkedik el a fizikai környezethez, mint a cloud computing. Az edge computing alkalmazásával a cloud computing csak a big data kezelésével, a business logika felépítésével és az adatok tárolásával foglalkozik.

4 A feladatok részletezése

Ebben a fejezetben az IoT alapú automatizált öntözőrendszer megvalósításáról olvashatunk. A rendszer tervét a 4.1 -es mutatja be. A 4.2 -es részben a rendszer megvalósításáról, hardveres és szoftveres oldaláról lehet részletesen olvasni. A 4.3 -as részben egyéb lehetséges megoldások kerülnek bemutatásra. A 4.4 -es rész a teljes rendszer végső állapotát mutatja be.

4.1 Tervezés

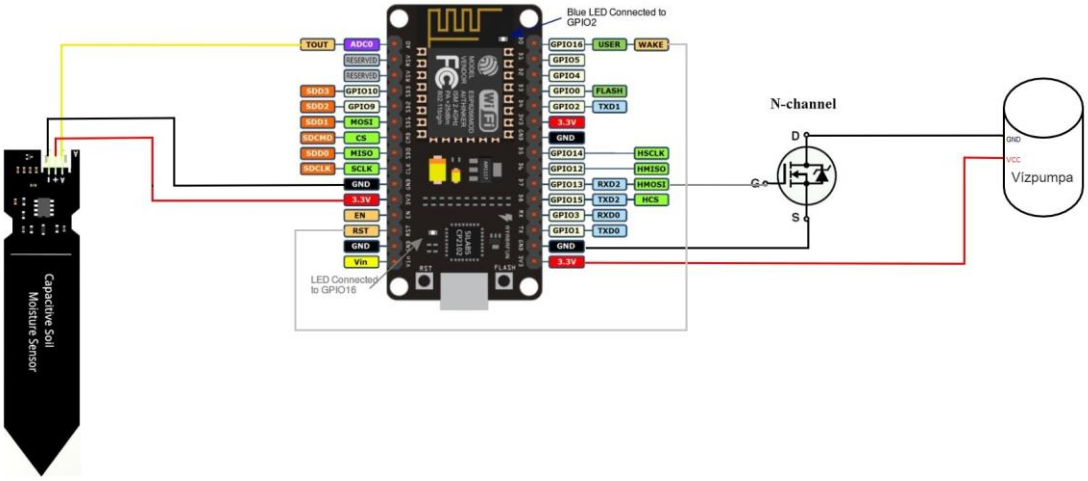
A tervezés fázis arról szólt, hogy az általam elképzelt rendszerhez kitaláljam milyen hardveres és szoftveres eszközöket használjak és azokat, hogyan kapcsoljam össze. Először az általam elképzelt rendszert egy modellben vizualizáltam, hogy egy megfogható képet fessek a terveimről. A 21. ábra a rendszer vizualizációját mutatja



21. ábra: Az öntözőrendszer

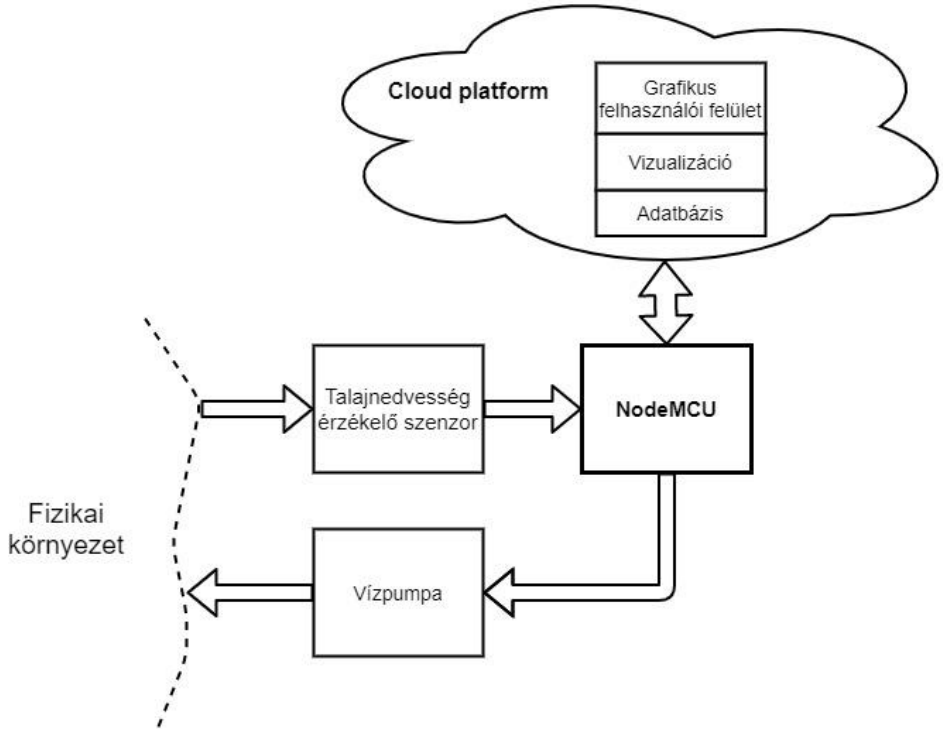
Az ábrán már látható a NodeMCU, egy talajnedvesség érzékelő szenzor és egy vízszivattyú. A szürke nyilak az információ áramlását szemléltetik. Tehát a talajnedvesség érzékelő szenzor mér egyet, ezt az analóg mérési adatot a NodeMCU digitális jellé konvertálja. Az adatot ezután egy algoritmus feldolgozza, amely során eldől, hogy a NodeMCU aktiválja -e a vízszivattyút vagy sem. Ha a talaj száraznak minősül, akkor a vízszivattyút a NodeMCU aktiválja és elindul az öntözés. A talajnedvesség mérő szenzor a következő ciklusban megint mér egyet és ha a talaj már meg lett öntözve, akkor nem aktiválódik a szivattyú. A folyamatot elég, ha naponta 1-2 alkalommal lefuttatjuk, hiszen időt kell adni a növénynek, hogy felszívja a talaj víztartalmát. A rendszerterv megvalósításához a NodeMCU adott volt, de a megfelelő

szenzorokat és beavatkozó szerveket ki kellett választani. Választásom a 21. ábrán is látható SA-27 talajnedvesség érzékelő szenzor és a miniatűr DC vízpumpa volt, amelyek specifikációjuk alapján képesek ellátni feladatukat az öntözőrendszerben. A szenzorokhoz viszonylag olcsón hozzá lehet jutni, így több szenzort is kipróbálhattam, de az egyszerűség kedvéért maradtam a talajnedvességmérő szenzornál. Első lépésként szeparáltan teszteltem az egyes rendszerkomponenseket. Először a NodeMCU -val ismerkedtem meg. A NodeMCU programozásához szükség volt egy fejlesztői platformra, amelyben a programkód megírható. A konzulensem javaslatára a Visual Studio Code fejlesztő környezettel kezdtem, de rövid kutatás után kiderült, hogy az Arduino IDE -t valamivel egyszerűbb használni, mert a sok előre megírt könyvtárat egy kattintásra telepíteni lehet és már használható is. Az interneten fellelhető példák nagy részét is Arduino IDE -ben készítették, így az információ gyűjtés és megvalósítás is egyszerűbbnek bizonyult vele. Ugyancsak szükség volt egy cloud platformra a szenzor adatok megjelenítéséhez. A ThingSpeak platform egy ingyenes cloud platform, amelynek volt többféle integrálható könyvtára is az Arduino IDE -hez. A ThingSpeak -re feltölthető adatoknak van egy limitje, így nagyobb projektekhez nem lenne jó választás, de az én rendszerem esetében bőven elegendő ezt használni. Jövőbeni esetleges bővítési lehetőségek figyelembevétele miatt az Amazon Web Services (AWS) cloud platform szolgáltatásait is megvizsgáltam és remek választásnak tűnik, ha bővíteni szeretném a rendszeremet. Az AWS nem ingyenes, de van egy 12 hónapos ingyenes kipróbálási lehetőség. A szolgáltatásainak sokszínűsége miatt a többi cloud platform óriás közül is kiemelkedik. A rendszer terve végül összeállt, amelyet a 22. ábra szemléltet.



22. ábra: Az öntözőrendszer rendszerterve

A teljes öntözőrendszerről készült blokkdiagramot a 23. ábra mutatja be.



23. ábra: Öntözőrendszer blokkdiagram

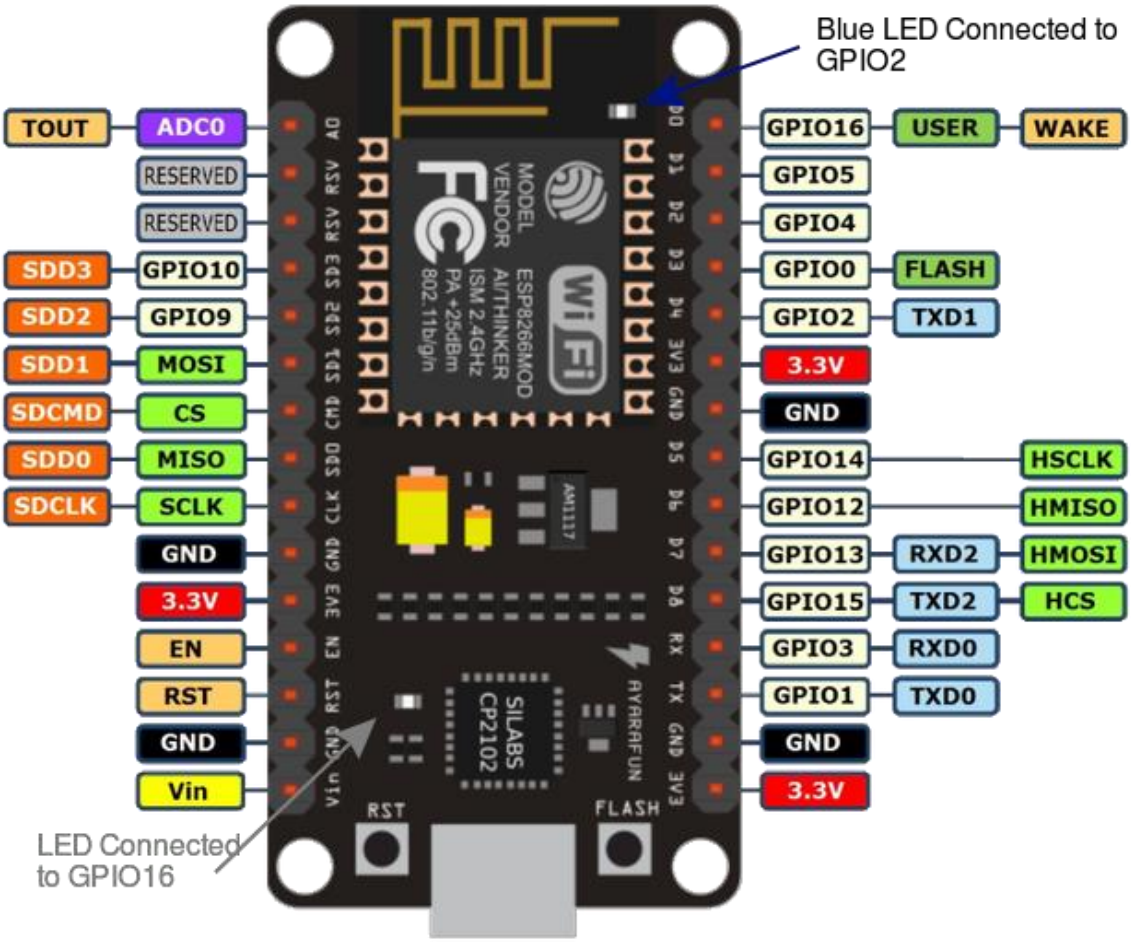
A talajnedvesség szenzor érzékeli a növény talajában történő nedvességtartalom változást. Ezt az adatot a NodeMCU feldolgozza, továbbítja a cloud platformnak és ha szükséges aktiválja a vízpumpát, amely megöntözi a növényt. A ThingSpeak cloud platform biztosítja az öntözőrendszer számára a talajnedvesség adat tárolását és megjelenítését. A ThingSpeak felületén további számításokat végezhetünk a talajnedvesség adatokon MATLAB scriptek segítségével.

4.2 Megvalósítás

Ebben a részben először a hardverekről és azok specifikációiról írok, majd a választott szoftverekre és a kódrészletekre térek ki. A teljes programkód a függelékben látható. A megvalósítás és tervezés fázis sokszor nem különülnek el teljesen egymástól, hiszen iteratív módon zajlik a fejlesztés. Ha valamit nem tudunk úgy elkészíteni, mint ahogy azt előre megterveztük, mert az adott eszköz esetleg nem pont úgy működik vagy nem állt rendelkezésre olyan eszköz, amilyet szerettünk volna alkalmazni, akkor újra át kell gondolnunk a tervünket és mérnöki szemlélettel más megoldást kell keresni a problémára.

4.2.1 Hardver

Az egyetlen adott hardver a rendszerben a NodeMCU V2 fejlesztőpanel volt. A következő információkat a [30] forrásból szereztem, amelyek a NodeMCU tulajdonságairól szólnak. A NodeMCU pin térképét a 24. ábra személteti.



24. ábra: A NodeMCU V2 fejlesztőpanel pin térképe [31]

A NodeMCU -n található egységekből szeretnék kiemelni néhányat. Az egyik az ESP8266 WiFi modul, amellyel a WiFi alapú kommunikáció könnyen megvalósítható. A másik a CP2102 soros-USB illesztőchip, amely biztosítja a számítógép és az ESP8266 modul közti soros kommunikációt. A modul egy 10 bites ADC -t is tartalmaz, amellyel a szenzorok analóg jeleit digitálissá konvertálhatjuk. Az öntözőrendszeremhez az A0 analóg pint a talajnedvesség szenzor adatainak olvasására használom. Használom továbbá az RST és D0 pint a power management megvalósításához. A D7 pint a MOSFET vezérlésére használom. A GND és 3.3V pinek a szenzort és a beavatkozót látják el energiával. Az ESP8266 rengeteg hasznos tulajdonsággal rendelkezik, amelyek közü a rendszerhez használtakat említem meg. Az ESP8266 az IEEE 802.11 b/g/n

vezeték nélküli adatátviteli protokollt használja, így képes a 2.4 GHz frekvenciasávon történő adatátvitelre is. TCP/IP protokoll struktúrával is rendelkezik, ami azt jelenti, hogy tudunk hozzá IP címet rendelni, így kijelenthetjük, hogy a NodeMCU egy IoT eszköz. Az ESP8266 képes ultra low power technológiát alkalmazni, emiatt gyakran alkalmazzák hordozható IoT termékekben [32]. Az ESP8266 energiatakarékos architektúrája három móddal rendelkezik. Az aktív módban a modul minden funkciót ellát energiával. Alvó módban a modul kevesebb, mint 12 uA áramot fogyaszt és kevesebb, mint 1mW teljesítménnyel képes stabil kapcsolatban maradni a vezeték nélküli összeköttetésben. Mélyalvás üzemmódban a WiFi nem üzemel, csak a real-time clock és a watchdog működik. A real-time clock -ot beállíthatjuk, hogy egy adott időintervallum lejárta után felébressze az ESP8266 modult. Ezt a funkciót a rendszeremnél én is kihasználom, hiszen későbbi terveim alapján szeretném, ha egy elemről is sokáig működne a rendszer. A modul IO pinjei ESP és túlfeszültség védelemmel lettek felszerelve és a NodeMCU is rendelkezik további védelmekel, így nem kell félnünk a lehetséges elektromos kisülésektől, amelyeket a DC pumpa motorja kelthet. A 3-6V DC feszültségen operáló mini vízpumpa a specifikációja alapján megfelelő választás volt az öntözőrendszerhez. A vízpumpa a 25. ábrán is látható.

25. ábra: Az öntözőrendszerhez használt 3-6V mini DC vízszivattyú [33]

A projekt méretét nézve elegendő volt egyetlen növény öntözését demonstrálni a pumpa segítségével. A DC pumpának 125mA áramra van szüksége, emiatt nyílvánvalóan a NodeMCU GPIO pinjei nem alkalmasak közvetlen meghajtásra, hiszen azok maximum 12mA áramerősséget képesek biztosítani. Rendelkezésemre állt egy MOSFET, amelyet kapcsolóként használtam [34]. A [35] és [36] források alapján a DC pumpa beüzemelése sikeresen megtörtént. A talajnedvesség méréséhez először az SA-27 típusú talajnedvességmérő szenzort próbáltam alkalmazni, de a NodeMCU analóg portján beolvasott adatokat akkor még nem tudtam értelmezni, így a SOLICAP-V12 kapacitív talajnedvesség érzékelőt alkalmaztam, amelyet a 26. ábra szemléltet.

26. ábra: Az öntözőrendszerhez használt SOLICAP-V12 talajnedvességmérő szenzor [37]

Ehhez az érzékelőhöz biztosítottak olyan szükséges információkat, amelyeket a másiknál nem találtam, de szükséges lett volna a használatához. Először le kellett jegyezni a levegőn, majd a vízben mért nedvességtartalom adatot, amelyek szélsőértékként szolgáltak a további számításokhoz. A földbe helyezett szenzor adatai és a szélsőértékek alapján már ki lehetett számolni a talajnedvesség százalékos arányát. A NodeMCU hátrányai közé tartozik, hogy csak egyetlen ADC bemenete van. Ha több szenzort is szeretnénk használni akkor érdemes multiplexert alkalmaznunk. A hasonló NodeMCU -s projektekhez leggyakrabban alkalmazott multiplexer a CD4051 -es típus.

Ez egy 8 csatornás analóg multiplexer. A NodeMCU digital IO pinjei használhatóak a multiplexer kiválasztási mechanizmusának kezeléséhez. A multiplexer használatához szükséges információk a CD4051 specifikációjában megtalálhatók [38]. A specifikációból kiemelném a Pin Functions CD4051B táblázatot, amely a 27. ábrán is látható.

Pin Functions CD4051B

PIN		I/O	DESCRIPTION
NO.	NAME		
1	CH 4 IN/OUT	I/O	Channel 4 in/out
2	CH 6 IN/OUT	I/O	Channel 6 in/out
3	COM OUT/IN	I/O	Common out/in
4	CH 7 IN/OUT	I/O	Channel 7 in/out
5	CH 5 IN/OUT	I/O	Channel 5 in/out
6	INH	I	Disables all channels. See Table 1.
7	V_{EE}	—	Negative power input
8	V_{SS}	—	Ground
9	C	I	Channel select C. See Table 1.
10	B	I	Channel select B. See Table 1.
11	A	I	Channel select A. See Table 1.
12	CH 3 IN/OUT	I/O	Channel 3 in/out
13	CH 0 IN/OUT	I/O	Channel 0 in/out
14	CH 1 IN/OUT	I/O	Channel 1 in/out
15	CH 2 IN/OUT	I/O	Channel 2 in/out
16	V_{DD}	—	Positive power input

27. ábra: A CD4051 multiplexer pinjeinek funkcióit szemléltető táblázat [38]

Én a táblázatból kiolvasható információk és a specifikációban megtalálható egyéb adatok alapján alkalmaznám a multiplexert. A multiplexer egy lehetséges alkalmazásához ötlet meríthető a [39] -es forrásból.

A power management használatához a NodeMCU RST és D0 pinjét össze kell kötnünk. A power managementről a következő szoftveres részben írok bővebben.

4.2.2 Szoftver

A NodeMCU programozásához először az Arduino IDE fejlesztő környezetet használtam, mert ehhez sokkal több információ állt rendelkezésre, mint a Visual Studio Code -hoz. A WiFi kapcsolat létrehozásához az ESP8266WiFi könyvtárból a WiFi.begin() funkciót használtam. A kapcsolódáshoz szükséges kód a [40] alapján készült. Maga a kód a következőképpen néz ki:

```
//Connecting to WIFI
WiFi.begin("test1_ssid", "test1_pw");
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED)
{
delay(500);
```

```
Serial.print(".");
}
Serial.println();
Serial.print("Connected, IP address: ");
Serial.println(WiFi.localIP());
```

A WiFi.begin() funkció első paramétere a WiFi hálózatunk neve, második paramétere a WiFi hálózatunk jelszava. A kódrészletben látszik, hogy sikeres csatlakozás esetén a WiFi.localIP() funkcióval kiiratom a hálózat IP címét.

Ezután a talajnedvesség szenzor által küldött adatok beolvasásához szükséges kódot készítettem el. A kód elkészítéséhez segítséget nyújtott a következő kódrészlet [41]. A függvények használatához szükség volt néhány információt előre definiálni. A kódrészlet a következő:

```
//Moisture Sensor
const int air_moisture_value = 710;   //sensor value in air
const int water_moisture_value = 310;  //sensor value in water
int soil_moisture_value = 0;
int soil_moisture_percent=0;

//Reading sensor data
soil_moisture_value = analogRead(A0);
Serial.println(soil_moisture_value);

//Raw data to Percent
soil_moisture_percent   =   map(soil_moisture_value,   air_moisture_value,
water_moisture_value, 0, 100);
```

A szenzor által küldött adatokat az analogRead(A0) függvénnyel olvashatom be. Az air_moisture_value és a water_moisture_value értékét a szenzor első használatakor kell beállítani. Az air_moisture_value konstans változóba a szenzor által a levegőn mért nyers adat kerül, míg a water_moisture_value konstans változóba a vízben mért adat. Ezek lesznek a szenzor szélső értékei, amelyeket a map() függvényben használok. A map() függvény segítségével számolható ki a talajnedvesség százalékos aránya.

A következő lépés a DC pumpát működtető algoritmus megírása volt. A MOSFET vezérléséhez a NodeMCU D7 pinjét a MOSFET Gate -jére kötöttem, ahogyan ezt a hardveres részben is kifejtettem. A D7 pint a pinMode() függvénnyel OUTPUT módba kellett állítani. A talajnedvesség százalékos értéke alapján egyszerű if-else ágakkal eldönthető, hogy melyik esetben állítsuk magas állapotban a D7 pint, ezzel aktiválva a vízpumpát. Az öntözés akkor indul, ha a talajnedvesség 5% felett és 30% alatt van. Az 5% -os határ a szenzor esetleges földből való kifordulásának hibáját

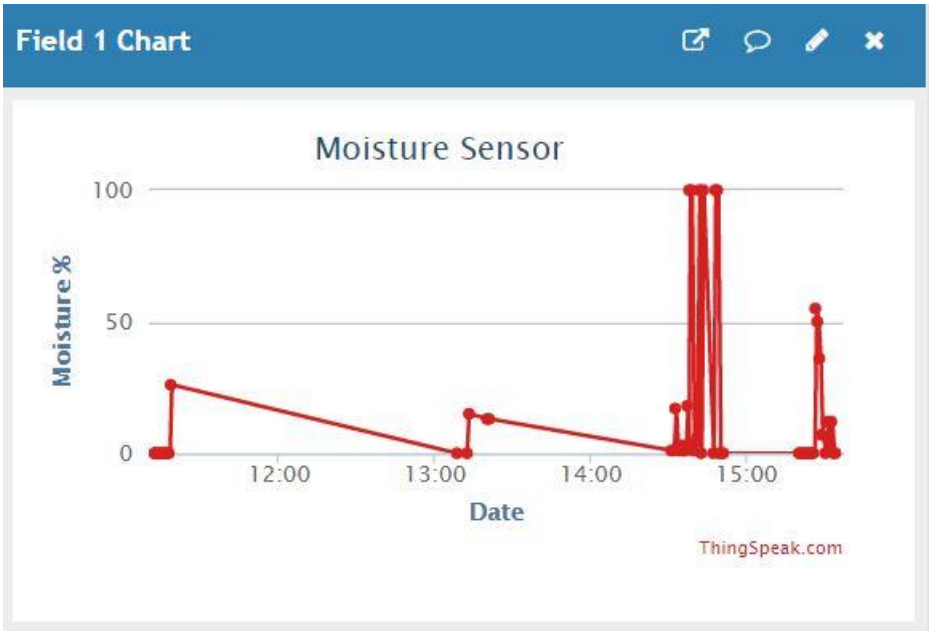
próbálja lekezelni a szabad levegővel érintkezve ugyanis előfordulhat, hogy 0% közeli értéket mér. Az algoritmust a következő kódrészlet tartalmazza:

```
if(soil_moisture_percent >= 100)
{
  Serial.println("Moisture: 100 %");

}
else if(soil_moisture_percent <=0)
{
  Serial.println("Moisture: 0 %");
}
else if(soil_moisture_percent >0 && soil_moisture_percent < 100)
{
  Serial.print("Moisture: ");
  Serial.print(soil_moisture_percent);
  Serial.println(" %");

  //Irrigation
  if(soil_moisture_percent >5 && soil_moisture_percent < 30){
      Serial.println("Start irrigation...");
      digitalWrite(13, HIGH);
      Serial.print("Irrigating for ");
      Serial.print(irrigation_time);
      Serial.println(" seconds");
      delay(irrigation_time*1000);
      digitalWrite(13, LOW);
      Serial.println("Stop irrigation...");
  }
}
```

A D7 pin kimenetét a digitalWrite() függvénnyel változtathatjuk. A D7 HIGH állapotba állításával a MOSFET rákapcsolja a vízpumpára a 3.3V feszültséget és ezzel elindítja azt. A D7 LOW állapotba állításával a pumpát kikapcsolhatjuk. Az öntözés időtartamát a pumpa aktiválása és leállítása közötti delay() funkcióval állíthatom. A delay funkció paraméterének mértékegysége ezredmásodperc. Az előre definiált irrigation_time változóban megadhatom hány másodpercig szeretném a vízpumpát működtetni.

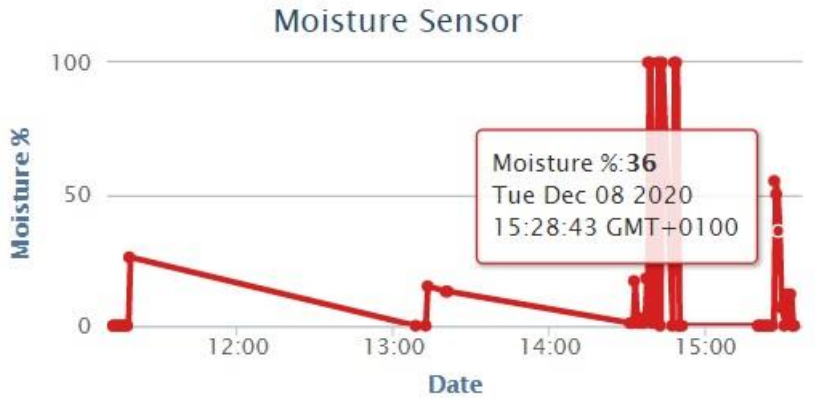
A szenzor adatait a ThingSpeak cloud platformon jelenítettem meg. Az Arduino IDE -ben telepíteni kellett a ThingSpeak használatához szükséges könyvtárat. Ezután a könyvtár függvényei segítségével már könnyű volt a megjelenítést elvégezni. A függvények kommunikációja REST alapú [42]. A megjelenítéshez szükséges kódot [43] alapján készítettem el. A ThingsSpeak inicializálását a ThingsSpeak.begin() függvény végzi. A ThingSpeak.writeField() függvénnyel a ThingSpeak -en létrehozott csatornába lehet adatot felvinni. Meg kell adnunk a CHANNEL ID -t, azt, hogy a csatorna melyik FIELD -jére szeretnénk küldeni az információt a 8 lehetséges FIELD közül, a felvinni kívánt adatot és az íráshoz szükséges API KEY -t, amelyet a ThingSpeak biztosít

számunkra. Az paraméterek megadása után a kívánt információ megjelenik a ThingSpeak -es csatornánkon. A grafikont testre lehet szabni, hogy hogyan jelenítse meg az információkat. Én talajnedvesség százalék és dátum alapján jelenítettem meg a szenzor által küldött információkat. A ThingSpeak -en megjelenített grafikon a 28. ábrán látható.



28. ábra: A talajnedvesség százalékot és a mérés időpontját tartalmazó grafikon

A grafikon pontjaira kattintva látható a szenzor által mért talajnedvesség százalékos értéke és a mérés időpontja, amit a 29. ábra szemléltet.



29. ábra: A december 8-án mért 36% -os talajnedvesség adat a grafikonon

A grafikonon megjelenített adatokat manuálisan is le lehet kérdezni egy XML formátumban, de a programban is le lehet kérdezni a ThingSpeak biztosította funkciók segítségével. Ilyen lekérdező funkció például a ThingSpeak.readField() függvény,

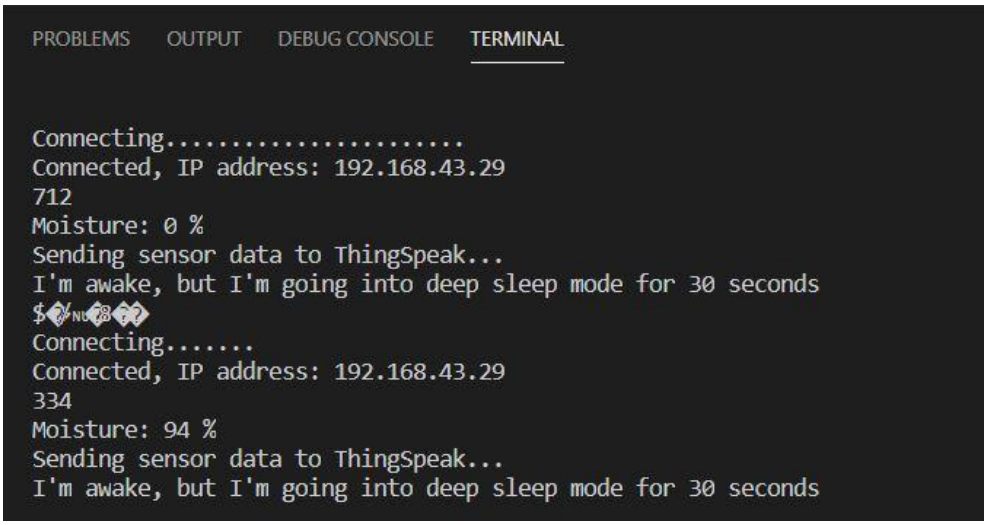
amely alkalmazása nagyban hasonlít a writeField változathoz. Ez a letölthető XML fájl adatbázisként is szolgál, hiszen az itt megjelenő adatok rendszerezve vannak.

Az öntözőrendszerhez készült power managementet az ESP.deepSleep() függvény segítségével végzem. A power management [44] alapján készült. A függvény használatához szükség van a NodeMCU RST és D0 pinjének összekötésére. Az ESP.deepSleep() paraméterében megadott időtartamig az ESP8266 mélyalvás üzemmódban marad. A függvény a D0 pint magasba állítja és ezzel újraindítja az ESP-t. Az újraindítás hatására az ESP felébred és a programkód az elejéről kezdi a lefutást. Mivel a talajnedvességet elegendő egy szobanövénynél naponta kétszer megmérni, így a mélyalvás időtartamát is ehhez viszonyítva kell meghatározni. A folyamat szemléltetése érdekében ezt az időtartamot 30 másodpercre állítottam.

4.3 Egyéb megoldási lehetőségek

Ebben a részben az általam vizsgált további megoldási lehetőségek lesznek kifejtve. A fejezetben olvashatunk a Visual Studio Code -ban megírt algoritmusról, ThingSpeak -es további lehetőségekről és az Amazon Web Services cloud platform lehetőségeiről.

A VS Code az INO kiterjesztést nem tudja lefordítani, de a kód átmásolásával és az Arduino.h könyvtár hozzárendelésével már sikeresen fordítható. A ThingSpeak használatához hozzá kell adni a projekthez a ThingSpeak könyvtárat. Ezután már fordítható és feltölthető a program a NodeMCU -ra. Működés alatt a soros monitoron hasonló kiírások láthatóak, amelyet a 30. ábra szemléltet.



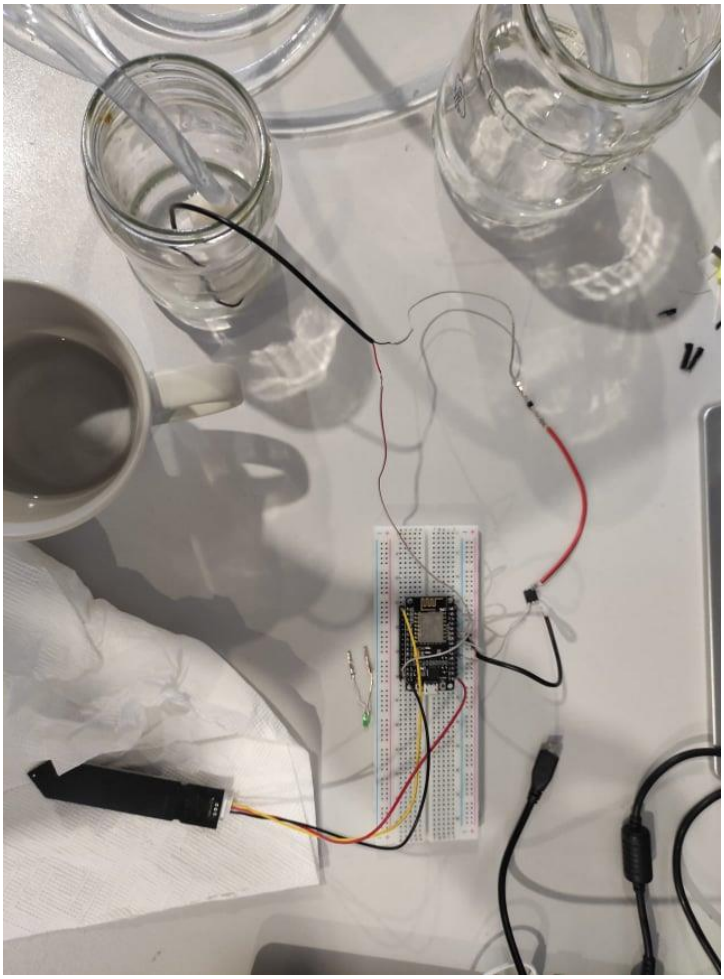
30. ábra: A VS Code soros monitorján megjelenő információk két ciklus alatt

A ThingSpeak könyvtár által használt függvények REST alapon kommunikálnak, de lehetőségünk van másik ThingSpeak -es könyvtár használatára is, amely már MQTT kommunikációt használ. A ThingSpeak cloud platform az MQTT kommunikációt is támogatja [45].

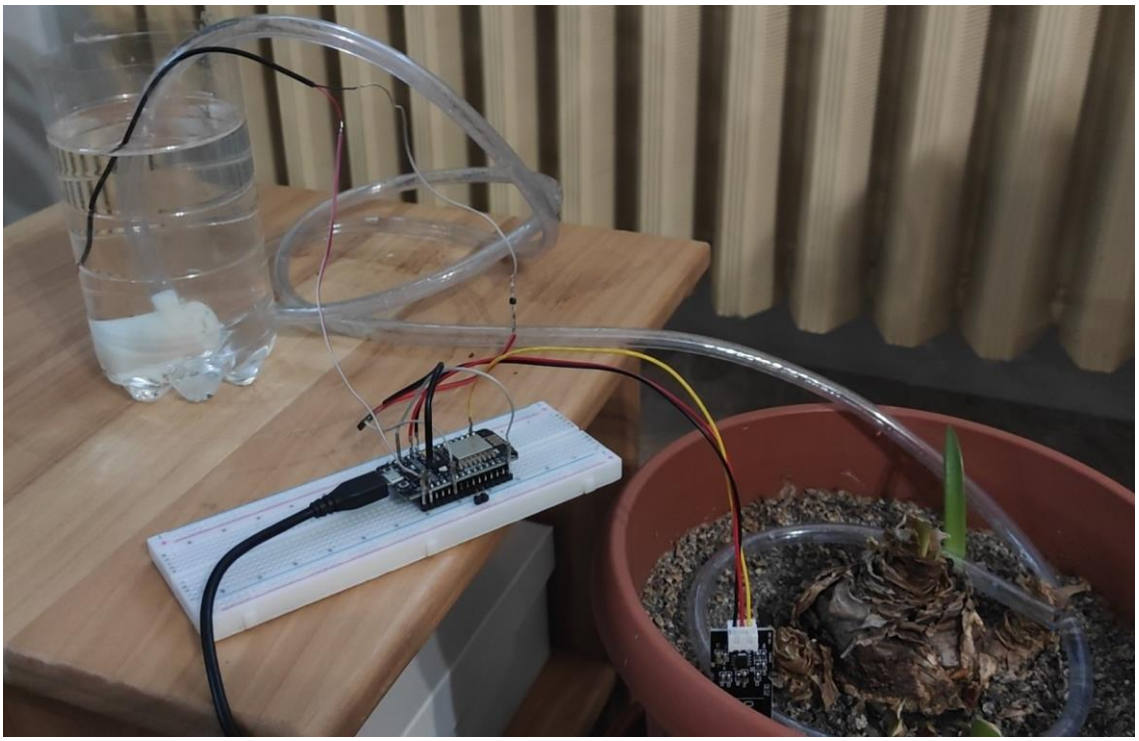
Az Amazon Web Services (AWS) cloud platform az egyik legjobb a sok közül. Több cloud platformokat rangsoroló weboldalon is a legjobbak között helyezkedett el a Thingworx, Microsoft Azure, Google Cloud és IBM Watson melett. A skálázhatósága és sokszínű szolgáltatásai miatt mégis az AWS uralja a piacot [46]. Az AWS IoT szolgáltatásai közül kiemelném az AWS IoT Core -t és az AWS IoT Greengrass -t. Az IoT Core használata biztosítja az eszközök és a cloud platform közötti kommunikációt, amely MQTT alapú is lehet. Az AWS IoT Greengrass szolgáltatás az edge computingért felel. Eszközeink így internet kimaradás ellenére megszakítás nélkül is tudják végezni a dolgukat. Az internetre történő újra csatlakozás esetén az adatok szinkronizálódnak a cloud platformmal. Az AWS a szolgáltatások könnyű használata érdekében rövid gyakorlati videókkal segíti a felhasználókat, így keresgélés nélkül akár azonnal is elkezdhetjük az adott szolgáltatás használatát.

4.4 Összefoglalás

A rendszer az elképzelésemnek megfelelően működik és a feladatkiírásban szereplő feladatait teljesíti. A rendszer használja az ESP8266 több tulajdonságát is, mint például a WiFi kapcsolat létrehozását és power managementet. A talajnedvességmérő szenzorból sikeresen olvassa ki a rendszer az adatokat és azokat az algoritmus megfelelően dolgozza fel. Az algoritmus megfelelően működteti a vízszivattyút. A rendszer beindítás után magára lehet hagyni, az automatikusan öntözni fogja a növényt, mielőtt a növény talaja kiszáradna. Az egyedüli emberi beavatkozás csak a víztartály újra töltésénél szükséges. A rendszer működése monitorozható a ThingSpeak -en megjelenített grafikon alapján. Az adatbázisban tárolt adatok vizsgálata alapján a rendszer később finomítható. A rendszer felépítése egyszerű és könnyen áttekinthető. A szakdolgozat tartalmaz olyan alap információkat és megoldási lehetőségeket, amelyek segítségével hasonló vagy akár bonyolultabb NodeMCU -s projektek is elkészíthetőek. A teljes rendszer tesztelését és a gyakorlati alkalmazását a 31. és 32. ábra szemlélteti.

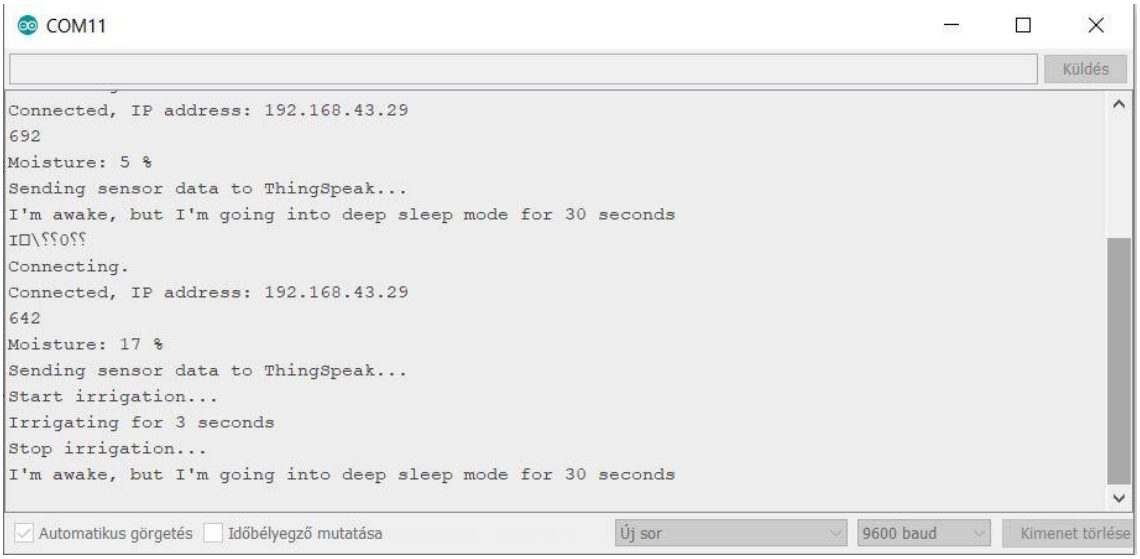


31. ábra: Az öntözőrendszer tesztelése



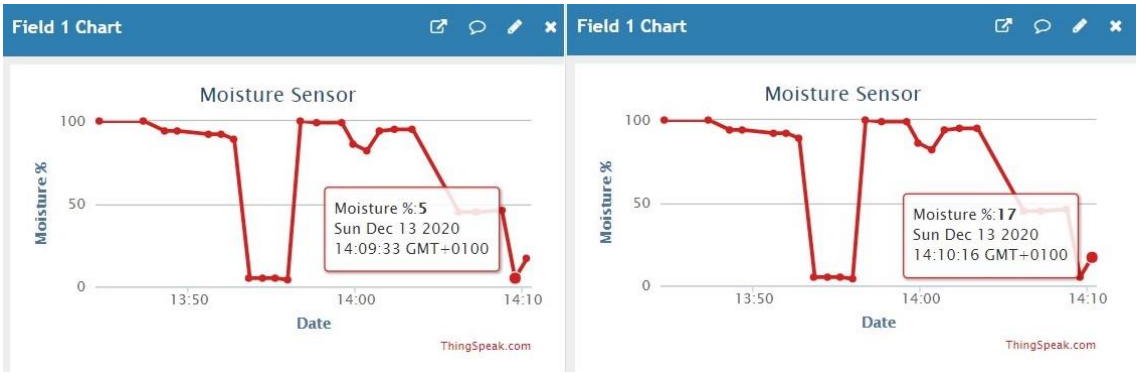
32. ábra: Az öntözőrendszer a gyakorlatban

A rendszer által kiírt információk a rendszer két ébren töltött ciklusa alatt a 33. ábrán láthatóak.



33. ábra: A soros monitoron megjelenő információk két öntözési ciklus alatt

Az első ciklusban a talajnedvesség szenzor 5% -ot mutat és az öntözés mégsem indult el. Ez azért van, mert a szenzor kimozdult a földből, így a szabad levegőt is belemérte az értékbe. Az algoritmus csak 5% felett indítja el az öntözést, ezzel kiküszöbölve ennek a hibának a lehetőségét. A második ciklusban a talajnedvesség 17% volt. Ez már egy érvényes szenzor adatnak számít, így az öntözést a rendszer elindítja, amelyet a rendszer a soros monitoron is naplóz. A ThingSpeak -re közben az adatok feltöltésre kerültek. A 34. ábra az utolsó két ciklusban feltöltött adatokat szemlélteti.



34. ábra: Utolsó két ciklus talajnedvesség adatai

Az öntözőrendszer működéséről készült videó a függelékben megtalálható linken elérhető.

5 Értékelés

Ebben a fejezetben olvashatunk az általam levont következtetésekről és a rendszer továbbfejlesztési lehetőségeiről.

5.1 Konklúzió

A rendszer, ahogyan azt az 4.4 -es összefoglalás részben is megemlítettem az elképzeléseknek megfelelően működik. Az öntözőrendszer jelenleg csak egy szenzort tud kezelni, de a 4.2.1 -es hardverekről szóló részben kifejtettem az ennek megoldására lehetséges opciót, amely egy multiplexer alkalmazását jelenti. Az öntözőrendszer tartalmaz power management -et, de amíg a NodeMCU látja el árammal a vízpumpát addig ennek nem lesz jelentősége, hiszen a vízpumpa motorja egy elemet hamar lemeríthet. A későbbiekben, ha elemre szeretném kötni a NodeMCU -t, akkor a vízpumpának mindenképpen másik energiaforrást kell keresni. A talajnedvességmérő szenzorból érkező adatot a NodeMCU lokálisan dolgozza fel és dönti el, hogy aktiválja -e a vízpumpát vagy sem, emiatt a ThingSpeak -es adatbázis nincs hatással a rendszer működésére, az csak az adatok megjelenítéséhez szükséges. Több NodeMCU hálózatba kapcsolása esetén már lényegesen több feladata lenne a cloud platformnak, hiszen onnantól kezdve a NodeMCU -k adatátvitele azon keresztül történne. A NodeMCU határaira visszatérve azt mondhatom, hogy az ESP8266 belső memóriája hatalmas terjedelmű kódokat is képes eltárolni, viszont adatbázisként és a megjelenítés kivitelezésére már nem érdemes használni. A NodeMCU -nak csak egy analóg bemenete van, de multiplexerek kaszkádosításával több szenzort is képes lenne kezelni. Egy darab NodeMCU sok mindenre képes, de nem érdemes egy egész lakást, illetve több állatkerti terráriumot is egyetlen NodeMCU -val kezelni, hiszen a szenzorok és NodeMCU között felhasznált kábelmennyiség és azok lefektetésének költsége jóval megdrágítaná az applikációt, sokkal olcsóbban kijövünk, ha több NodeMCU -t alkalmazunk vezeték nélküli kommunikációval.

5.2 Továbbfejlesztési lehetőségek

Ahogyan azt az 5.1 -es fejezetrészben is említettem a rendszer csak lokálisan működik, nem különül el az érzékelő és a beavatkozó rész. Továbbfejlesztésként

javasolnám az érzékelő és beavatkozó részeket elkülöníteni, így az érzékelőket tartalmazó rendszer elemről is működhet az ESP8266 power management lehetőségeit kihasználva. A beavatkozó rendszer ugyanúgy működhet egy másik NodeMCU -ról, de annak energiaellátását mindenképpen hálózatról kell biztosítani a vízpumpa és egyéb beavatkozó szervek teljesítményfelvétele miatt. A NodeMCU -val összekapcsolt szenzorok egy külön egységet alkothatnának és ezek elemről akár hónapokig is működhetnének. Ilyen NodeMCU -s érzékelő egységeket viszonylag olcsón elő lehetne állítani és a power management miatt ritkán kellene karbantartani. Ilyen egységeket lehetne telepíteni a növényeink környezetének érzékelésére és az adataikat a felhőn keresztül továbbítanák a beavatkozó egységnek, amelyen található algoritmus eldönti, hogy melyik növényt kell meglocsolni. Vagy akár az adatfeldolgozás a felhőben is történhet és csak egy egyszerű jelet küldenénk a beavatkozó egységnek, hogy elinduljon -e. A szenzorok változtatásával az érzékelő egység alkalmazható lenne akár állatkertek kisebb terráriumaiban vagy akváriumaiban, így azok akár egyszerre is monitorozhatóak lennének. A szenzorok által mért adatok alapján a beavatkozást is lehetne automatizálni, mint például a terrárium fűtése beindul, ha a hőmérsékletmérő szenzor által mért érték alapján. A skálázhatóság érdekében mindenképpen számításba kellene venni egy fizetős cloud platformot, mint például az AWS. Több szenzor és beavatkozószerv esetén a programkódot is érdemes lenne felbontani és függvényekben elvégezni az egyes műveleteket.

Irodalomjegyzék

[1] Threatpost, Lindsey O'Donnell, „*Top 10 IoT Disasters of 2019"*, https://threatpost.com/top-10-iot-disasters-of-2019/151235/ (2020.12.08.)

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, „*Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications*", IEEE Comm. Surv. & Tut., vol. 17, no. 4, 2015.

[3] LTECOVENT: „*HVAC alapok"*, https://hovisszanyero-szelloztetes.hu/legkondicionalo-futes-szelloztetes/ (2020.12.08.)

[4] Gyártástrend, *„Innovációs projekt egy hazai autógyártónál"*, http://gyartastrend.hu/autoipar/cikk/innovacios_projekt_egy_hazai_autogyartonal (2020.12.08.)

[5] Ipar 4.0 Technológiai Központ: „*Negyedik Ipari forradalom"*, http://www.ipar4.bme.hu/ipar-4-0/#page-content (2020.12.08.)

[6] Nagy Judit, „*Az Ipar 4.0 fogalma és kritikus kérdései – Vállalati interjúk alapján"*, Budapest, oldalszám 14-15, 2019, http://unipub.lib.uni-corvinus.hu/3869/1/VT_2019n1p14.pdf (2020.12.08.)

[7] Ipar 4.0 Technológiai Központ: „*Ipar 4.0 technológiák"*, http://www.ipar4.bme.hu/technologiak/#page-content (2020.12.08.)

[8] AIE Internship, E. Laureano, *„"Where is the Industry 4.0 in our life?" by Busra Guler"*, https://aie-internship.com/where-is-the-industry-4-0-in-our-life-by-busra-guler/ (2020.12.08.)

[9] Instructables, EasyIoT, *„ESP8266 Smart Plant Irrigation System"*, https://www.instructables.com/ESP8266-Smart-Plant-Irrigation-System/ (2020.12.08.)

[10] IoT Design Pro, *„IoT based Smart Irrigation System using NodeMCU ESP8266 & Adafruit IO",* https://iotdesignpro.com/projects/smart-irrigation-system-using-iot (2020.12.08.)

[11] Circuit Digest, A. Pandit, *„IoT based Smart Irrigation System using Soil Moisture Sensor and ESP8266 NodeMCU",* https://circuitdigest.com/microcontroller-projects/iot-based-smart-irrigation-system-using-esp8266-and-soil-moisture-sensor (2020.12.08.)

[12] NI, *„Data Acquisition (DAQ)"*, https://www.ni.com/hu-hu/shop/data-acquisition.html (2020.12.08.)

[13] Omega, *„A Complete Guide to Data Acquisition (DAQ) Systems",* https://www.omega.com/en-us/resources/daq-systems (2020.12.08.)

[14] N. Al-Falahy, O. Alani, „*Supporting massive M2M traffic in the Internet of Things using millimetre wave 5G network*”, University of Salford, Manchester, UK, University of Anbar, Iraq, IEEE, pp. 83-88, 2017.

[15] Gartner, Rob van der Meulen, „*Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016*”, Egham, UK, 2017, https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 (2020.12.08.)

[16] Security Today, G. D. Maayan, „*The IoT Rundown For 2020: Stats, Risks, and Solutions*”, 2020.01.13, https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx?Page=2 (2020.12.08.)

[17] Software Testing Help, „*TOP 11 Best Internet Of Things (IoT) Companies To Watch In 2020*”, https://www.softwaretestinghelp.com/top-iot-companies/ (2020.12.08.)

[18] M. Shirer, M. Torchia, „*Worldwide Spending on the Internet of Things Will Slow in 2020 Then Return to Double-Digit Growth, According to a New IDC Spending Guide*”, Framingham, Mass., 2020.06.18., https://www.idc.com/getdoc.jsp?containerId=prUS46609320 (2020.12.08.)

[19] IoT Analitics, P. Scully, „*Top 10 IoT applications in 2020”,* 2020.07.08, https://iot-analytics.com/top-10-iot-applications-in-2020/ (2020.12.08.)

[20] IoT Agenda, M. Rouse, „*What is IoT (Internet of Things) and How Does it Work?”,* https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (2020.12.08.)

[21] Cisco IBSG, D. Evans, „*The Internet of Things - How the Next Evolution of the Internet Is Changing Everything*”, 2011, https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (2020.12.08.)

[22] Wikipedia, „*Raspberry Pi*”, https://hu.wikipedia.org/wiki/Raspberry_Pi (2020.12.08.)

[23] Novel Bits, M. Afaneh, „*The Ultimate Guide to What's New in Bluetooth version 5.2*”, 2020.03.09., https://www.novelbits.io/bluetooth-version-5-2-le-audio/ (2020.12.08.)

[24] Silicon Labs, „*EFR32BG22 Wireless Gecko SoC Family Data Sheet*”, https://www.silabs.com/documents/public/data-sheets/efr32bg22-datasheet.pdf (2020.12.08.)

[25] Oracle, „*What Is Big Data?*”, https://www.oracle.com/big-data/what-is-big-data.html (2020.12.08.)

[26] NIST, P. Mell, T. Grance, „*The NIST Definition of Cloud Computing*”, Gaithersburg, Spec. Pub. 800-145, 2011,

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (2020.12.08.)

[27] DZone, D. Rana, „*Top 11 Cloud Platforms for Internet of Things (IoT)*”, 2019.08.05., https://dzone.com/articles/10-cloud-platforms-for-internet-of-things-iot (2020.12.08.)

[28] Cisco Blogs, M. Abdelshkour, „*IoT, from Cloud to Fog Computing*”, 2015.03.15., https://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing (2020.12.08.)

[29] OmniSci, „*Edge Network*”, https://www.omnisci.com/technical-glossary/edge-network (2020.12.08.)

[30] TávIR, „ *NodeMCU (ESP-12E/ESP8266; V2 (keskeny modul); CP2102)*”, https://shop.tavir.hu/product_info.php/kapcsolat-bluetooth-wifi-nodemcu-esp-12eesp8266-keskeny-modul-cp2102-p-552 (2020.12.08.)

[31] Circuits4you, „*Nodemcu pinout*”, https://circuits4you.com/2017/12/31/nodemcu-pinout/ (2020.12.08.)

[32] Espressif Systems, „*ESPRESSIF SMART CONNECTIVITY PLATFORM: ESP8266*”, 2013.10.12.

[33] PotentialLabs, „*Submersible Pump Mini*”, https://potentiallabs.com/cart/buy-micro-dc-3-6v-submersible-pump-mini-water-pump-online-hyderabad-india (2020.12.08.)

[34] ON Semiconductor, „*MOSFET – Power, N-Channel, Logic Level, DPAK/IPAK 12 A, 60 V*”, https://www.onsemi.com/pub/Collateral/NTD3055L104-D.PDF (2020.12.08.)

[35] TávIR, „*Egyszerű elektronika: MOSFET I. rész*”, http://www.tavir.hu/egyszeru-elektronika-mosfet (2020.12.08.)

[36] Arduino Project Hub, ejshea, „*Connecting an N-Channel MOSFET*”, https://create.arduino.cc/projecthub/ejshea/connecting-an-n-channel-mosfet-7e0242 (2020.12.08.)

[37] Indiamart, „*Capacitive Soil Moisture Sensor V1.2*”, https://www.indiamart.com/proddetail/capacitive-soil-moisture-sensor-v1-2-21672599291.html (2020.12.08.)

[38] Texas Instruments, „*CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer with Logic-Level Conversion*”, 2017, https://www.ti.com/lit/ds/symlink/cd4051b.pdf?ts=1607770246308&ref_url=https%253A%252F%252Fwww.google.com%252F (2020.12.08.)

[39] Instructables, witnessmenow, „*How to Use Multiple Analog Sensors on Your ESP8266*”, https://www.instructables.com/How-to-Use-Multiple-Analog-Sensors-on-Your-ESP8266/ (2020.12.08.)

[40] ESP8266 Arduino Core, „*ESP8266WiFi library*”, https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html (2020.12.08.)

[41] Hestore, „*SOILCAP-V12*”, https://www.hestore.hu/prod_10040575.html# (2020.12.08.)

[42] ThingSpeak Community, „*Send Data to ThingSpeak with Arduino*”, https://community.thingspeak.com/tutorials/arduino/send-data-to-thingspeak-with-arduino/ (2020.12.08.)

[43] FactoryForward, Sarath, „*Upload Sensor Data to ThingSpeak using NodeMCU*”, https://www.factoryforward.com/upload-sensor-data-thingspeak-using-nodemcu/ (2020.12.08.)

[44] Random Nerd Tutorials, „*ESP8266 Deep Sleep with Arduino IDE (NodeMCU)*”, https://randomnerdtutorials.com/esp8266-deep-sleep-with-arduino-ide/ (2020.12.08.)

[45] MathWorks, „*API Reference*”, https://www.mathworks.com/help/thingspeak/channels-and-charts-api.html (2020.12.08.)

[46] C# Corner, M. Chand, „*Top 10 Cloud Service Providers In 2020*”, https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/ (2020.12.08.)

Függelék

Az öntözőrendszer teljes programkódja:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ThingSpeak.h>

//ThingSpeak
WiFiClient client;
unsigned long channel_id=1248959;
const char* write_api_key="08I5FCTYFRV6TYRH";

//Power Management
unsigned long deep_sleep_time=30e6;

//Irrigation
unsigned int irrigation_time=3;

//Moisture Sensor
const int air_moisture_value = 710;   //sensor value in air
const int water_moisture_value = 310;  //sensor value in water
int soil_moisture_value = 0;
int soil_moisture_percent=0;


void setup()
{
//Serial port
Serial.begin(9600);
Serial.println();

//Connecting to WIFI
WiFi.begin("test1_ssid", "test1_pw");
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println();
Serial.print("Connected, IP address: ");
Serial.println(WiFi.localIP());

//Creating ThingSpeak client
ThingSpeak.begin(client);

//D7 digital output pin used for MOSFET switch
pinMode(13, OUTPUT);
}
void loop() {
//Reading sensor data
soil_moisture_value = analogRead(A0);
Serial.println(soil_moisture_value);
//Raw data to Percent
```

```
soil_moisture_percent   =   map(soil_moisture_value,   air_moisture_value,
water_moisture_value, 0, 100);
delay(1000);
//Sending sensor data to ThingSpeak Channel
if(soil_moisture_percent >= 100)
{
  Serial.println("Moisture: 100 %");
  ThingSpeak.writeField(channel_id,1,100,write_api_key);           //update
ThingSpeak channel field
  Serial.println("Sending sensor data to ThingSpeak...");
}
else if(soil_moisture_percent <=0)
{
  Serial.println("Moisture: 0 %");
  ThingSpeak.writeField(channel_id,1,0,write_api_key); //update ThingSpeak
channel field
  Serial.println("Sending sensor data to ThingSpeak...");
}
else if(soil_moisture_percent >0 && soil_moisture_percent < 100)
{
  Serial.print("Moisture: ");
  Serial.print(soil_moisture_percent);
  Serial.println(" %");
  ThingSpeak.writeField(channel_id,1,soil_moisture_percent,write_api_key);
//update ThingSpeak channel field
  Serial.println("Sending sensor data to ThingSpeak...");
  //Irrigation
  if(soil_moisture_percent >5 && soil_moisture_percent < 30){
      Serial.println("Start irrigation...");
      digitalWrite(13, HIGH);
      Serial.print("Irrigating for ");
      Serial.print(irrigation_time);
      Serial.println(" seconds");
      delay(irrigation_time*1000);
      digitalWrite(13, LOW);
      Serial.println("Stop irrigation...");
  }
}
  delay(10000); //10s delay
  //Power Management
  Serial.println("I'm  awake,  but  I'm  going  into  deep  sleep  mode  for  30
seconds");
  ESP.deepSleep(deep_sleep_time);

}
```