FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Stahorszki Péter Bence

Otthoni növénygondozási rendszer tervezése

Konzulens

BUDAPEST, 2017

Tartalomjegyzék

[Összefoglaló 5](#_Toc499328991)

[Abstract 6](#_Toc499328992)

[1 Bevezetés 7](#_Toc499328993)

[2 Irodalomkutatás 8](#_Toc499328994)

[2.1 Létező megoldások 8](#_Toc499328995)

[2.1.1 Edyn[1] 8](#_Toc499328996)

[2.1.2 GreenIQ [2] 10](#_Toc499328997)

[2.1.3 PlantLink [3] 11](#_Toc499328998)

[2.1.4 Grove Smart Plant Care Kit 11](#_Toc499328999)

[2.2 Hardver Technológiák 12](#_Toc499329000)

[2.2.1 Raspberry Pi 13](#_Toc499329001)

[2.2.2 Arduino 15](#_Toc499329002)

[2.2.3 Szenzorok 17](#_Toc499329003)

[2.3 Kommunikációs technológiák 17](#_Toc499329004)

[2.3.1 HTTP, HTTP/2 18](#_Toc499329005)

[2.3.2 WebSocket 18](#_Toc499329006)

[2.3.3 MQTT 18](#_Toc499329007)

[3 Tervezés 20](#_Toc499329008)

[3.1 A növények gondozása 20](#_Toc499329009)

[3.1.1 Öntözés 21](#_Toc499329010)

[3.1.2 Világítás 22](#_Toc499329011)

[A telepített egység irányítása 23](#_Toc499329012)

[3.2 23](#_Toc499329013)

[3.2.1 Hardverek kezelése 24](#_Toc499329014)

[3.2.2 Szoftver 25](#_Toc499329015)

[4 Implementáció 27](#_Toc499329017)

[5 Tesztelés 28](#_Toc499329018)

[Irodalomjegyzék 29](#_Toc499329019)

[Függelék 30](#_Toc499329020)

Hallgatói nyilatkozat

Alulírott **Stahorszki Péter Bence**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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, 2017. 12. 04.

...…………………………………………….

Stahorszki Péter Bence

Összefoglaló

Ide jön a ½-1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

* Bevezetés 1-2 oldal
* Irodalomkutatás 10-15 oldal
* Tervezés 10-15 oldal
* Implementáció 5-10 oldal
* Tesztelés 5 oldal
* Összefoglalás 1-2 oldal

# Bevezetés

Az információ technológia fejlődésével egyre mélyebben szivárog bele az emberek életébe a számítástechnika. Ahogy ezek az eszközök egyre több mindenre képesek, kényelmesebb a használatuk, az embereknek is megnövekedett az ilyen irányú igényük. Először a számítógép terjedt el a háztartásokban, amit leginkább szórakozásra használtunk, közben az járműgyártásába is beivódott, ahol rengeteg olyan kényelmi és biztonsági funkciót valósítottak meg vele, amik nélkül ma már igazán el se lehetne adni autót. Az ezredforduló környékén pedig egyre több helyen vált elérhetővé az internet, és az internetre kapcsolódni képes eszközök, egy új fogalom jelent meg, az **Internet of Things (IoT)**,magyarul a dolgok internete.

Az IoT azon eszközök összessége, amelyek a bennük megtalálható szoftverek, szenzorok, elektronikák segítségével képesek adatokat gyűjteni, amelyeket a hálózaton keresztül képesek továbbítani. Az IoT technológia lehetővé teszi a világunk leképzését számítógépes rendszerbe, ezáltal rengeteg folyamat esetén csökkenthető az emberi beavatkozásnak a szükségessége, ezáltal az esetleges hibák száma is redukálható.

Az IoT egy nagyon érdekes területe az okosotthon, amelynek az otthoni folyamatok automatizálásáról szól, legyen szó szűkebb értelemben a fűtés, világítás, takarító-robot, szellőztetés szabályozásáról, tágabb értelemben pedig az adott háztartásra jellemző, nem mindenhol felmerülő feladatokról, akár egy állat rendszeres etetése, vagy mint esetemben, egy mobil növénygondozó rendszer.

Egy olyan rendszer tervezését, és építését választottam feladatul, ami képes lehet több növény egyidejű gondozására, emberi beavatkozás nélkül, emellett a növény környezetének állapotát rendszeresen elmenti, és egy webes felületen is megjeleníti. Természetesen akkor is működnie kell, amikor nincs internetes kapcsolat.

# Irodalomkutatás

Az interneten számos hasonló megoldásról lehet olvasni, nem egy ilyen termék megvásárolható, de rendelkezésre áll számos nyilvános forráskód is, amik segítségével az érdeklődő informatikus hobby kertész könnyebben elindulhat az automatizálás útján. Néhány terméket a közösségi finanszírozás[[1]](#footnote-1) útján akarták piacra dobni, a pénzt sikerült is megszerezni hozzá – szóval a vásárlói igény meg van - viszont ezek közül sok kudarcba fulladt. Ezekről rendszerekről olvashatunk az következő oldalakon.

## Létező megoldások

### Edyn[1]

Az Edyn egy moduláris rendszer, kerti alkalmazáshoz. Három részből áll, a **Garden Sensorból**, ami egy összetett egység, képes a fény, talajnedvesség, páratartalom és a talaj tápanyagtartalmának mérésére. Beépített WiFi modullal rendelkezik, ennek segítségével tud kapcsolatot tartani a rendszer többi részével. A **Water Valve** egy okos szelep, amit rá kell kötni a vízhálózatra, és a **Garden Sensor**, valamint az időjárási adataira alapozva megállapítja, hogy mennyi vizet juttasson a növényhez. A kert állapotát egy telefonos alkalmazáson keresztül lehet figyelemmel kísérni, a növény adatbázisban 5000 fajta jellemzői vannak tárolva, a felhasználó ezek közül választhatja ki a sajátját. Ezen a felületen megjelennek a helyszín időjárási adatai, valamint megtekinthetők a fény és nedvességi viszonyok változása napi, heti, havi és évi lebontásban. Az adatbázisban található információk, és a szenzorból nyert adatok alapján a rendszer megpróbálja meghatározni, hogy az adott növények milyen fejlődési stádiumban van. Ha fontos változás lép fel a kert, vagy egy növény állapotában, a felhasználó erről értesítést kap, ami vonatkozhat az esetleges kedvezőtlen körülményekre, vagy arról, hogy új milyen új növekedési fázisba lépett a növény. A rendszer arra is fel van készítve arra az esetre is, ha elveszítené az internetkapcsolatot. Folyamatosan tölti be előre a következő napok időjárási előrejelzések alapján a várhatóan szükséges öntözések időzítését. Működéséhez nem kell külső tápellátás, a beépített napelemnek és akkumulátornak köszönhetően ideális körülmények között akár teljesen töltés nélkül is képes operálni. Egy szenzorhoz több növényt is be lehet regisztrálni, és 25m2 nagyjából az a terület, amit egy egységgel le lehet fedni. Értelemszerűen, ha a kertünkbe többféle fényviszonnyal rendelkező terület van, vagy esetleg másfajta vízigényű növényeket szeretnénk egyszerre nevelni, akkor több szenzorra van szükség, és ha ezekhez szeretnénk külön öntözést, akkor annyi helyre kell venni egyet-egyet a szelepből is. Ez elég költséges lehet, tekintve hogy a Garden Sensor fogyasztói ára $100 dollár, a Water Valve-ért pedig $60 dollárt kell otthagynia a vásárlónak. Ez nem kevés pénz, viszont ezért egy elég összetett és jól átgondolt megoldást szállítanak. Ami szerintem még hiányzik ahhoz, hogy teljes legyen a rendszer, az a kiegészítő világítás arra az esetre, ha nem lenne elegendő a természetes fény, valamint az alkalmazás oldalán olvasott vélemények alapján a növényi adatbázis növelésére is szükség lenne. /\*Mindent összevetve egy nagyon elegáns megoldás. \*/



1. ábra: Garden Sensor



2. ábra: Water Valve

### GreenIQ [2]

A GreenIQ más irányból közelíti meg a kert automatizálását. Nem annyira a különböző növények specifikus igényeit veszi figyelembe, hanem a vízfelhasználás optimális szinten tartását. Célja az, hogy átvegye a szerepét a régi fajta beépített öntöző berendezéseknek, amik a felhasználók által lettek beprogramozva, valószínűleg nem elég hatékonyan. Egy az egyben kiváltja a már meglévő vezérlőt, és ha már a kert be van építve öntözőrendszerrel, nincs is szükség másra, mint felkonfiguráljuk, és csatlakoztassuk a régi egység helyére. WiFi-n kapcsolódik az internetre, és nem rendelkezik beépített akkumulátorral, így áramkimaradás esetén értelemszerűen szünetel a működés. Több öntözési zónát képes kezelni, valamint az internetről letöltött adatok alapján vezérelheti a kerti világítást is.

Az IoT koncepciót magas szinten valósítja meg. Létezik kliense a nagyobb platformokra, mint IOS, Android, és webböngészőn keresztül is használhatjuk. Nem csak egyszerű szenzorok (talajnedvesség, vízfolyás érzékelő) jeleit használhatja a legkedvezőbb öntözés meghatározásához, a helyi legközelebbi, vagy az esetleges saját időjárás állomás adatait is képes lekérdezni. Ezeken felül irányítható Amazon Echo, és Google Home segítségével is, ehhez az **IFTTT** technológiát használja. Ez a **If This Than That** rövidítése, ami egy web-alapú technológia, ha valami (this) történik, akkor végrehajtja az utasítást (that). Összekapcsolható okosotthon vezérlő alkalmazásokkal, mint a **Gideon[[2]](#footnote-2)** vagy a **Muzzley**, amik célja az, hogy a legtöbb okosotthon rendszernek nyújtsanak egy összpontosított felületet, ezzel könnyítve a felhasználó dolgát. Azon felül, hogy a kerti öntözést sokkal kényelmesebbé teszi, biztonsági funkciót is betölthet a GreenIQ. Ha a **Nest** otthonfigyelő rendszer füst érzékelője túl magas értéket mér, beindít egy öntözési ciklust az összes zónában, ezáltal nedvesen tartja a ház környékét, ezáltal segítve a tűz esetleges terjedésének a megakadályozását.



3. ábra: GreenIQ

### PlantLink [3]

A PlantLink kínálja a legkisebb funkcionalitást, cserébe a kedvezőbb árért, és a használat egyszerűségéért. Két részből áll, a központi egységből és a szenzorból. A központi egységre az otthoni routerre csatlakozik egy Ethernet hálózaton, hozzá pedig 64 szenzor csatlakozhat egyszerre, ezek az alacsony frekvenciás rádiójelnek köszönhetően elég távol, akár 100 méterre is lehetnek az állomástól. Egy szenzor felkonfigurálásához szükség van a talaj és növény típusára. Egyedül a talajnedvesség mérésére képes, ez alapján tesz ajánlást az öntözések időpontjára, viszont mennyiségére nem, ezért inkább tapasztaltabb kertészek számára lehet ideális. A kezdő csomag $79-be kerül, ami tartalmaz egy állomást és egy szenzort, ezen felül minden újabb szenzor plusz $35.



4. ábra: PlantLink

### Grove Smart Plant Care Kit

A következő példa inkább azoknak szól, akik szívesen barkácsolnak. A Grove Smart Plant Care Kit egy kevés részből álló készlet, amit a meglévő Arduino mikrokontrolleres fejlesztőkártyánkhoz vásárolhatunk meg. Azoknak ajánlott, akik most ismerkednek az Arduino világával. Aki megvásárolja, kézhez kap egy csomagot, ami tartalmazza a komponenseket, ami egy egyszerű öntöző és állapotmegfigyelő rendszerhez szükséges.

Nyílt forráskódú projekt, így tökéletes lehet olyan emberek számára, akik szeretnék megérteni egy ilyen rendszer működését, hogyan működik együtt az Arduino a szenzorokkal és beavatkozókkal, hogyan kell rá programot írni.



5. ábra: Grove Smart Plant Care Kit

## Hardver Technológiák

A fentebb felsorolt termékek már olyan célhardvert használtak, amit kifejezetten ezekre az alkalmazásokra fejlesztett a gyártó. Ha nagy mennyiségű termelés a cél, ennek a specifikus hardvernek a megalkotása fontos, mivel azzal, hogy csak a legkisebb szükséges halmazra redukálja a képességeit, nagyban csökkenti az előállítási költséget. Egy egységen a féldollárnyi költség megtakarítása elsőre nem tűnik olyan soknak, viszont ha ebből sikerül eladni mondjuk egy millió darabot, akkor máris fél millió dollárt spórolt a gyártó, amit sok fontos dologra költhet, fejlesztésre, terjeszkedésre, reklámra.

Számomra és a többi hobbista, építgetni szerető emberek számára más szempontok az érdekesek. Fontos általában, hogy az irányító egység sokoldalú legyen, mivel a projekteknél sok a próbálkozás, sokszor ki kell próbálni különböző megoldásokat. Emiatt, a hobbiépítők egyik legjobb barátai a nem régen megjelent kicsi, egyetlen áramköri lapra integrált számítógépek. Rendelkeznek processzorral, memóriával, háttértárral, szóval mindennel, amit egy teljesen funkcionális számítógépnek tartalmaznia kell. Ezek általában nagyjából bankkártya méretű kis teljesítményű gépek, amelyeken teljesértékű számítógépes operációs rendszerek futnak, ami általában Linuxnak valamely változata, valamint változatos IO-val szerelik őket. A funkcionalitásukhoz képest olcsók, a kis méret miatt könnyen kezelhetők, sok helyre beépíthetők, a számítógépes OS miatt pedig a felhasználási lehetőségeinek csak a programozójának képességei szabnak határt.

### Raspberry Pi

A legelterjedtebb ilyen eszköz a Raspberry Pi. Ezt a kis komputert eredetileg oktatási célból fejlesztették ki, hogy bevezessék az iskolásokat a számítástechnika világába és, hogy olcsóságával a fejlődő országokban is alternatíva legyen az egyszerűbb, számítógépet igénylő feladatok megoldásához. A termék sokkal sikeresebb lett, mint várták, mivel képességei nem csak oktatási célra teszik alkalmassá, hanem akár robotok, beágyazott okos rendszerek alapjának is. 2012-ben adták ki az első Pi-t, és öt éves pályafutása alatt több mint 12 milliót adtak el belőle, ezzel elnyerve a legtöbbet eladott brit számítógép, és a világ harmadik legtöbbet eladott általános célú számítógép címet. Jellemzően a Linuxnak valamelyik variánsa fut rajta, de az újabbakra már a Windows 10 IoT Core operációs rendszert is feltelepíthetjük.

Többfajta kiadásban találkozhatunk a Raspberry Pivel. Az általánosabb felhasználásra szánt típus a kezdetekben Model A, majd Model B ma a harmadik iterációnál jár. Nagyjából bankkártya méretű, és rendelkezik a legtöbb olyan input/output lehetőséggel, amit megszoktunk egy PC vagy laptop esetében.

Az első generációs modellt egy magos, 700 MHz-es processzorral, 256 MB RAM-mal, USB, video és audio csatlakozókkal szerelték, kezdetben hálózati port nélkül. Emellett rendelkezett GPIO, azaz általános célú input/output tüskékkel, amelyeknek a viselkedését a felhasználó futási időben vezérelheti, megfigyelheti. Ezen tüskék között találhatóak tápellátáshoz szükséges csatlakozók, így alacsonyabb energiaigényű eszközöket könnyen irányíthatunk segítségükkel. Az operációs rendszer, és háttértár egy cserélhető SD kártyán foglal helyet.



6. ábra: Raspberry Pi 3

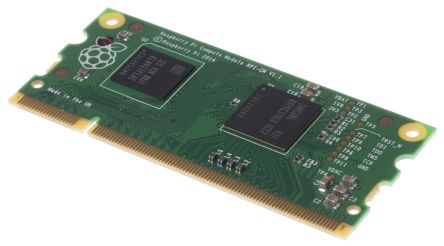
A harmadik generációs Raspberry Pi már négy magos, 1.2 GHz-es processzorral, 1GB RAM-mal, beépített WiFi modullal rendelkezik. Fogyasztói ára $35, ennyi pénzért egy olyan számítógépet kapunk, melynek hardveres képességei lehetővé teszik, hogy akár fullHD médiakiszolgálóként, dedikált játékgépként használjuk, de volt, aki felhő szervert csinált belőle.

Létezik egy **Zero** nevű típus is, ami még kisebb, mint a Model A/B család, ezáltal kisebb teljesítményű, kevesebb porttal rendelkezik, kis mérete és súlya viszont alkalmassá teszi arra, hogy olyan helyeken alkalmazzuk, ahol szűkek a méret vagy súly korlátok, például okos drónok irányítására. Alapára $5 dollár, $10 dollár a beépített WiFi modullal rendelkező verzióé, szóval az okos projektjeink vezérlését olcsón megoldhatjuk.



7. ábra: Raspberry Pi Zero

Ezek mellett még forgalmaznak egy **Compute Module** nevű típust, ami formára megegyezik a SO-DIMM memóriákkal, viszont a lábkiosztása értelemszerűen teljesen más. 4GB beépített tárhellyel rendelkezik, és megvásárlása esetén kapunk mellé egy IO Board eszközt, aminek segítségével fel tudjuk programozni. Ezt azok számára fejlesztették ki, akik saját nyomtatott áramkört terveznek, így biztosan a lehetséges legkisebb helyet foglalja el belőle a Pi.

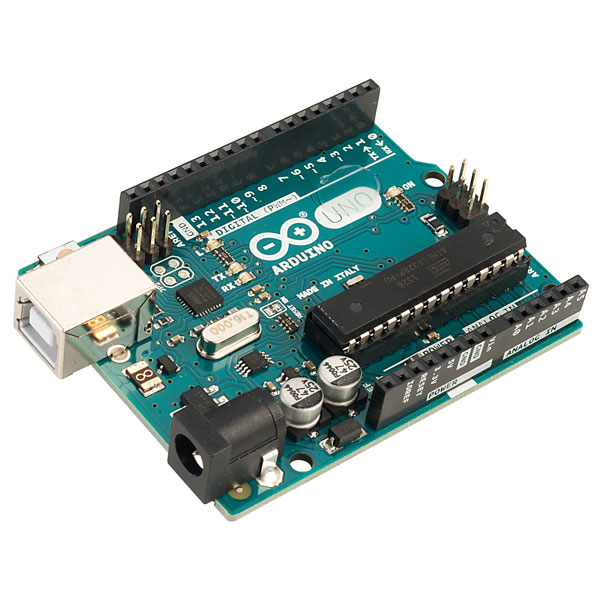


8. ábra: Raspberry Pi Compute Module

### Arduino

Sok olyan projekt van, amikor nincs szükség akkora számítási kapacitásra, mint amivel például a Raspberry Pi rendelkezik, szeretnénk egy egyszerűbben programozhatóbb vezérlő egységet, kevesebb a hely a vezérlőnek, valós időben beavatkozásra képes alkalmazást szeretnénk írni, vagy nagyobb mennyiségű GPIO tűre van szükségünk, esetleges analóg bemenetekre, amivel a Pi nem rendelkezik. Ilyenkor kerülnek képbe a mikrokontrollerek.

Általában olyan helyeken alkalmazzák, amikor a környezetben belépő változásra gyorsan kell reagálni. Általában nem operációs rendszer, hanem az adott feladatra készített program fut rajtuk. Programozásuk történhet gépi kódon, saját programnyelv, vagy magas szintű programnyelven keresztül, ez attól függ, hogy milyen fejlesztőrendszert építettek az adott mikrokontroller köré.



9. ábra: Arduino Uno

A ma leginkább elterjedt mikrokontrolleres fejlesztőkártya az Arduino. Ezek az Atmel 8-bites mikrokontrollerei köré épülnek. Egyszerűen programozható az Arduino IDE segítségével, ami felismeri a számítógéphez csatlakoztatott Arduino eszközöket, a kód validálása után pedig fel is tölthetjük a programunkat rajta keresztül.

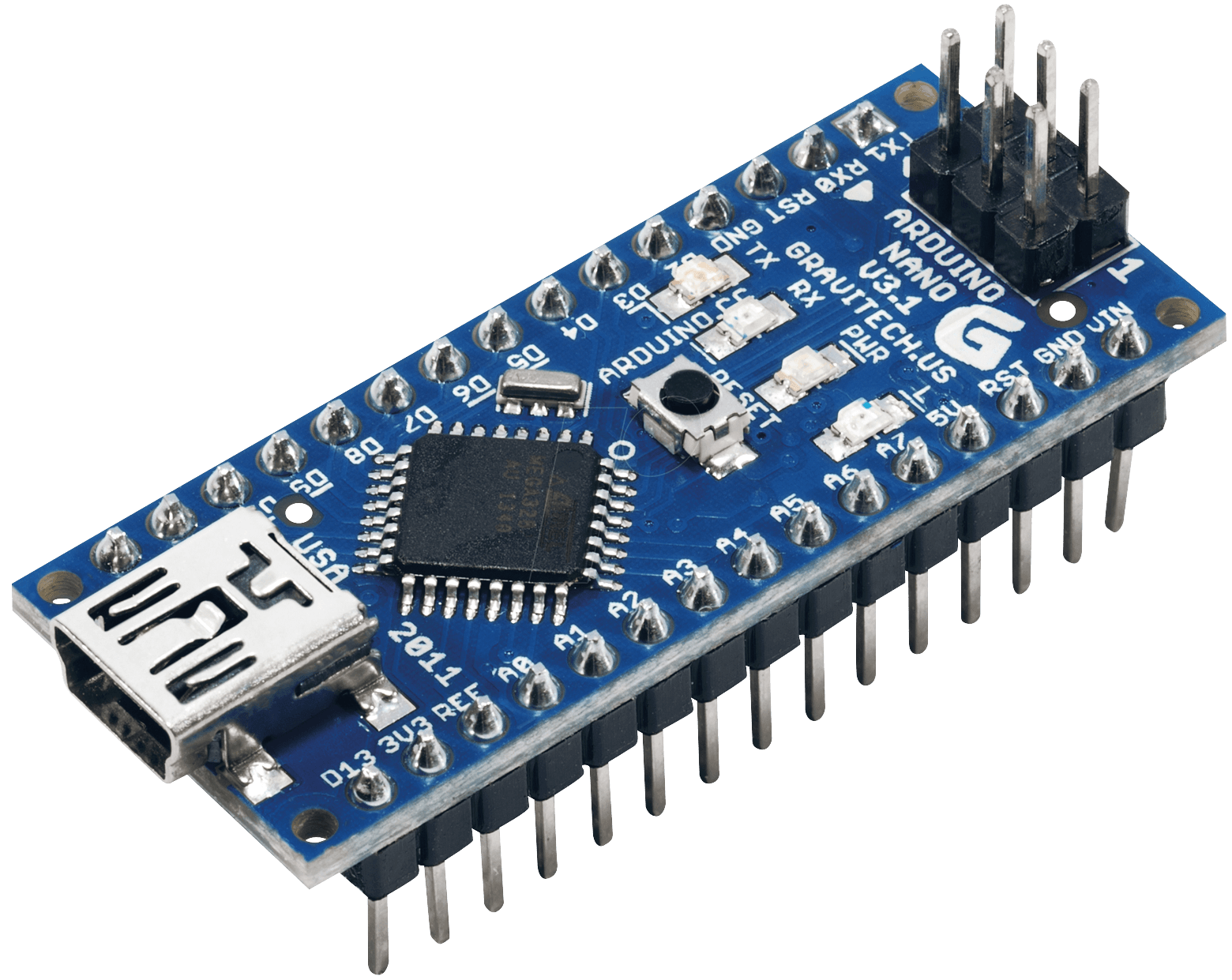
Népszerűségét több dolognak is köszönhet. Egyik ilyen, hogy nem kell hozzá külön programozó áramkör. A bootloader egy olyan program, ami minden bekapcsolás után lefut, egy bizonyos byteszekvenciát vár a soros porton. Ha megérkezik ez a bytesorozat, akkor programozza a mikrokontrollert, ha ezt nem érkezik meg, akkor betölti a már rátelepített programot. Ezáltal könnyebb a fejlesztés, viszont érdemes figyelembe venni, hogy ez minden indításnál késleltetést jelent, ami valós idejű alkalmazások esetén fontos lehet.

Egy Arduino program minimum két függvényből áll, ez a setup() és a loop(). A setup() egyszer fut le, a program indulásakor, itt konfiguráljuk a mikrokontroller lábainak viselkedését. A loop() függvény tartalmazza a végrehajtandó parancsokat, folyamatosan fut.

Rengetegen használják, emiatt nagyon sok open-source library található hozzá mindenféle részfeladatra, amik segítségével néhány sor kód segítségével egészen komplex rendszereket lehet megvalósítani.

Ha kevésnek találnánk az Arduino képességeit, könnyen bővíthetjük az úgynevezett Shieldek segítségével. A Shieldek egyszerű interfacet adnak az Arduino számára komplex hardveres technológiák használatához, mint például Bluetooth, WiFi, LCD kijelző. Olyan áramköri lapok, amik méretre és lábkiosztásra megegyeznek az Arduinoval, így csak rá kell helyezni, és akár több ilyen Shieldet is használhatunk egyszerre.

A Raspberryhez hasonlóan az Arduino is megkapható több formában, eltérő specifikációkkal. Az alap Uno mellett létezik például a Nano, ami sokkal kisebb helyet foglal, a Mega, ami nagyobb, és sokkal több lábkivezetést tartalmaz, vagy az Ethernet, ami beépített internetes interfésszel rendelkezik. Az eredeti Arduino is elég olcsó, €20 körül kapható, viszont mivel nyilvánosak a hardveres tervrajzai, sokan gyártanak belőle másolatokat, amik ennek akár az ötödébe is kerülhetnek.



10. ábra: Arduino Nano

## Kommunikációs technológiák

Amikor olyan eszközöket készítünk, amivel interneten keresztül szeretnénk kommunikálni gondolnunk kell arra, hogy mi a legmegfelelőbb technológia erre a feladatra.

### HTTP, HTTP/2 REST API

A HyperText Transfer Protocol egy kérés-válasz alapú kommunikációs protokoll kliens és szerver között. Első verzióját 1991-ben alkották meg, a legfrissebb a 2015-ben kiadott HTTP/2. A kliens egy kérést küld a szerver felé, a végrehajtani kívánt metódussal és paraméterekkel. Ezt a kérést a szerver feldolgozza, majd az eredményt válaszüzenetben visszaküldi a kliensnek. Ez az üzenet tartalmazza a státuszt a kérés teljesítéséről, valamint a kért tartalmat, és egyéb információkat. Egy megbízható szállítási protokollt igényel, ez a legtöbb esetben a TCP, így a QoS (Quality of Service) felelősséget nem vállal.

A régebbi HTTP protokollok szöveges üzenetekkel, és soros üzenetfeldolgozással működtek, viszont az új HTTP/2 már bináris adatot használ, és a kéréseket multiplexálással dolgozza fel. Ezáltal csökken az üzenet mérete, és a feldolgozás ideje. Újdonság még a HTTP/2-ben a szerver oldali „push” művelet, aminek segítségével lehetőség van arra, hogy akkor is küldjünk adatot a kliensnek, amikor nem érkezett kérés.

Bár a HTTP/1 is használható volt az IoT eszközök esetében, az új HTTP/2 verzió alkalmasabb a feladatra, tekintve, hogy sokkal kisebb az erőforrásigénye, ami fontos, ha egy kis teljesítményű hardverrel, vagy akkumulátorral szerelt eszközről van szó.

### WebSocket

A WebSocket protokoll a kliens és a szerver között létesít kétirányú kapcsolatot, ezáltal ideálissá teszi, ha egy rendszer állapotát szeretnénk folyamatosan megfigyelni, vagy ha egy kliensnek rendszeresen, és gyors frissítésekre van szüksége. Ahhoz hogy létrejöjjön egy ilyen kapcsolat, a kliens egy „kézfogás” üzenetet küld a szervernek HTTP-n keresztül, a szerver válaszával pedig felépül a közvetlen kapcsolat a kliensalkalmazás és a szerver között.

2008-ban kezdték a használatát, a fő motiváció a létrejöttére az volt, hogy, a HTTP/1-ből hiányzott a kétirányú kapcsolat. Bár a HTTP/2-ben ez bevezetésre került, a WebSocket így sem válik szükségtelenné, mivel bár a HTTP/2 push művelete eljuttatja a kliensgépre az üzenetet, de nem a kliensalkalmazáshoz, a WebSockettel ellentétben, így még egy lépésre szükség van.

### MQTT

A Message Queue Telemetry Transport egy kis erőforrásigényű protokoll, amit az IBM fejlesztett ki a gép-gép közötti kommunikáció megkönnyítésére, azokra az esetekre, amikor fontos, hogy a kapcsolat fenntartásához a lehető legkisebb kódot használjuk, vagy a sávszélesség erősen korlátozott. Működéséhez szükség van egy harmadik, köztes szereplőre, az üzenetközvetítőre. Az ő szerepe az, hogy a küldő által alkalmazott üzenetküldési protokollról átfordítsa az üzenetet a fogadó protokolljára.

Háromszintű QoS-t definiál, arra vonatkozóan, hogy az üzenetközvetítő/kliens mennyi felelősséget vállal az üzenet kézbesítésére.

1. A közvetítő/kliens egyszer küldi el az üzenetet, nem vár megerősítésre.
2. A közvetítő/kliens legalább egyszer elküldi, és megerősítésre vár.
3. A közvetítő/kliens egyszer küldi el az üzenetet, négy-utas[[3]](#footnote-3) kézfogást alkalmazva a biztos átvitel érdekében.

Az üzeneteket bármilyen szinten lehet küldeni, és a kliensek akármilyen szintű „témára” feliratkozhatnak. Ez azt jelenti, hogy ha a kliens QoS 0 szinten iratkozott fel, de QoS 2 szintű az üzenet, azt QoS 0 szinten fogja megkapni.

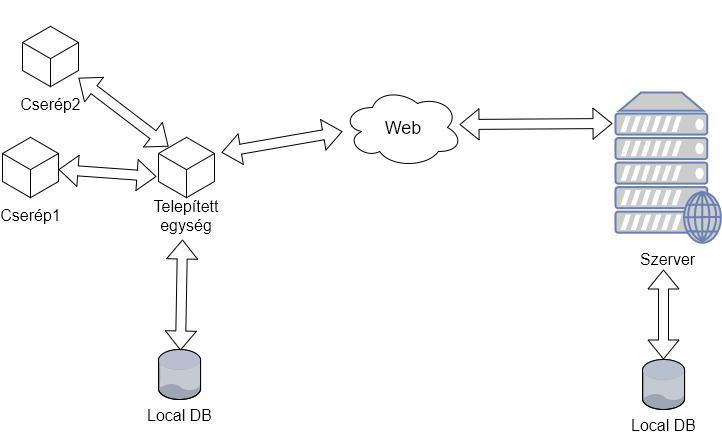
Jelenleg ez a legkisebb erőforrásigényű a széles körben elterjedtek protokollok között, így ha erősen korlátoltak az eszköz hardveres képességei, esetleg akkumulátorról működik, vagy gyenge, megbízhatatlan az adatkapcsolat érdemes számításba venni.

# Tervezés

A célom ezzel a projekttel az volt, hogy megismerkedjek az IoT világával, és kipróbáljak pár olyan technológiát, amivel eddig még nem volt szerencsém foglalkozni, ezért lehet, hogy a megoldásom nem lesz optimális.

A feladat egy olyan rendszer megtervezése, ami önállóan képes több növény felügyelésére és gondozására. Ezen felül legyen lehetőség a növények állapotát távolról megfigyelni, beavatkozni.

Ezek szerint két részből kell állni a rendszernek, egy egységből, ami a növények mellett van, és egy webszerverből. A növény mellé telepített egység kötelességei a növény állapotának folyamatos rögzítése, ezek alapján a gondozási teendők elvégzése, az adatok továbbítása a webszerver felé. A webszerver feladata az adatok tárolása, megjelenítése könnyen értelmezhető formában, és felhasználói tevékenységek kezelése.



11. ábra: Kommunikációs diagram

## Telepített egység

### A növények gondozása, szoftver

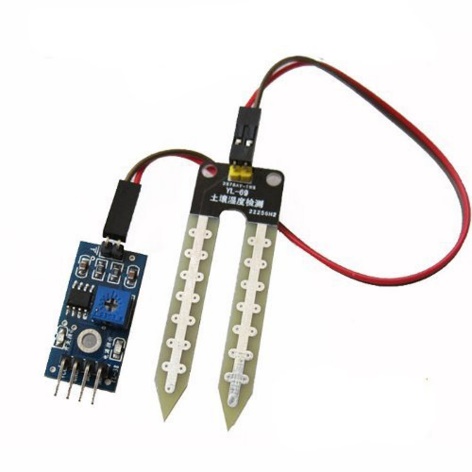
A növények számára fontos környezeti tényezők a fény, hőmérséklet, páratartalom, talajnedvesség, ezeket kell egy optimális szinten tartani, ahhoz, hogy megfelelő legyen a növekedés. Ahhoz, hogy megállapítsuk a környezet alapján a gondozási teendőket, szükség lesz érzékelőkre, egy talajnedvességmérőre, és egy fényszenzorra növényenként. A páratartalmat valamint a hőmérséklet egyelőre csak megfigyelem, nem veszem számításba a gondozás során, tekintve, hogy egy-két növény gondozása miatt nem lenne praktikus az egész szoba páratartalmát és hőmérsékletét befolyásolni.

A telepített egységen található szoftver gondozási szempontból releváns részei a következők:

* adatgyűjtés, ami meghatározott időközönként történik,
* adatok feldolgozása, és a gondozási teendők végrehajtása egy adott időpontban.

#### Adatgyűjtés

A talajnedvesség nem változik hirtelen, ezért úgy gondolom, hogy ha tíz percenként mérem meg, az elegendő.



12. ábra: Talajnedvesség érzékelő

Az öntözéshez a vízellátást először úgy terveztem, hogy egy keret tetején legyen egy tartály, amiből egy elektromos szelep segítségével engedem ki a vizet, viszont rájöttem, hogy ez túlságosan körülményes, bekorlátozza a növény magasságát, és az se mindig egyértelmű, hogy mennyi vizet ereszt át. Ezért inkább a növény mellé helyezem a tartályt, és egy kisméretű szivattyúval szívom fel belőle a vizet. Ahhoz, hogy a felhasználó tudja, mikor esedékes a tartály utántöltése, egy ultrahangos távolságmérőt helyezek a tetejére, ami a kibocsát egy hullámot, és a visszaverődő hullám beérkezési ideje alapján kiszámítja a távolságot, emiatt ügyelni kell arra, hogy a vízfelszínre merőlegesen legyen elhelyezve.



13. ábra: Ultrahangos távolságérzékelő

Arra az esetre, ha nem volt elegendő a napi fénymennyiség, a hiányzó részt egy növénynevelő lámpával biztosítanám. A növényhez tárolva lesz a kívánt a napsütéses órák száma, így az adatokból könnyen kiszámítható, mennyi fényre van még szüksége.

A növény állapotát, azaz a talajnedvességet és fénymennyiséget, hőmérsékletet egy táblázatba tervezem menteni, tíz percenként, amikor a talajnedvesség mérése történik, viszont ha a fényt csupán tíz percenként mérem meg, akkor könnyen lehet, hogy az adatok fals eredményhez vezetnének. A nap általi megvilágítás a felhők vonulása miatt gyakrabban változhat, mint amilyen gyakran a talajnedvességet mérjük, azaz a mért jel frekvenciája nagyobb, így nagyobb mintavételi frekvenciára van szükség. Ezért úgy terveztem, hogy a fényt sokkal sűrűbben mérem, mint a talajnedvességet, tíz másodpercenként, majd tíz percenként megnézem hány százaléka van az elmúlt tíz percben vett mintáknak azon érték felett, ami már megfelelő a növény számára. Ez a százalék elég pontosan visszaadja, hogy a tíz perc alatt hány percig sütött a nap megfelelő mértékben, mivel tíz másodperc alatt ritkán változnak drasztikusan a fényviszonyok. Ehhez a tíz másodperces mérésekhez egy külön tervezek létrehozni, amiből tíz percenként törlöm az értékeket.



14. ábra: Fényérzékelő

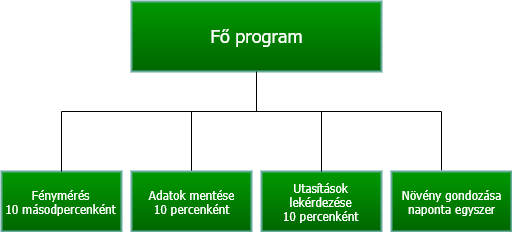
#### Adatok feldolgozása, gondozási teendők végrehajtása

Tekintettel arra, hogy napi többszöri öntözésre csak rendkívüli esetekben van szükség, a gondozási teendők számítását napi egyszer tervezem elvégezni, aminek időpontjául a naplemente időpontját választom.

A fotoszintézisnek két szakasza van, a fényszakasz és a sötétszakasz. A sötétszakasz a fényszakaszban megtermelt anyagok felhasználásával állítja elő a glükózt, ami a növény tápanyaga. Ahhoz, hogy ez a leghatékonyabban működjön, arra kell törekedni, hogy az egyes szakaszok ne váltogassák egymást túl sűrűn, azaz, ha a növényről elkerül a fény, mondjuk 19:00-kor, ne 23:00-kor kapcsoljuk be a világítást, ezzel kirángatva a növényt a sötét szakaszból. Ezt úgy oldom meg, hogy minden növényhez be lehet állítani, mikor van a napfelkelte, mikor a naplemente, és ezek közötti számolja az aznapi fénymennyiséget. A naplemente időpontjában összegzem a fénymennyiséget, és ha kell, bekapcsolom a lámpát, így nagyobb az esély, hogy a növény folytatni tudja a fényszakaszt. Azért a napfelkelte időpontjától, és nem éjféltől kezdem az összegzést, mivel ha mondjuk egyik nap egyáltalán nem sütött a nap, és este 19:00-kor elindít egy nyolc órás világítást, akkor az már átcsúszik a következő napra, így az összegbe is beleszámítana.

### Kommunikáció a szerverrel

A telepített egység további feladata, hogy továbbítsa az adatokat a szerver felé, valamint kérdezze le az esetleges utasításokat. Ezt a két műveletet egyszerre tervezem végrehajtani az adatmentéssel. Egy utasítás végrehajtása így maximum tíz percet késik az utasítás kiadása után, de az, hogy tíz perccel később öntözünk, vagy kapcsoljuk fel a lámpát nem sokat számít. Ezek alapján a szoftver felépítése a következőképpen alakul:



15. ábra: A telepített egység szoftverének felépítése.

### Hardver

Az telepített vezérlőegység elé támasztott követelmények a következők:

* Képesnek kell lennie kapcsolódni az internetre.
* Legyen képes adatbáziskezelő szoftver futtatására.[[4]](#footnote-4)
* Tárhely mérete lehet viszonylag csekély.[[5]](#footnote-5)
* Legyen kisméretű, hogy ne növelje túlzottan az egész telepített egység méretét.

A Raspberry Pi-t választottam erre a feladatra. Ahogy korábban olvashattuk nagyjából bankkártya méretű. Rendelkezik Ethernet csatlakozóval, így egyszerű az internetre való kapcsolódás. Linux operációs rendszer fut rajta, így a legtöbb ismert és széles körben alkalmazott szoftveres technológiát tudom rajta használni. A háttértár méretét én választom meg azzal, hogy mekkora SD-kártyát teszek bele, így ez sem probléma. Rendelkezik számos olyan ki és bemeneti csatlakozóval, mint például a hangkimenet, kamera és kijelző kapcsolódási lehetőségek, amelyeket bár ebben a projektben nem fogok kihasználni, a jövőre nézve érdekesek lehetnek. Ha egy kamerás megfigyelést szeretnénk implementálni, esetleg helyi hangjelzéseket kiadni bizonyos események bekövetkeztekor, vagy lehetőséget biztosítani egy kijelző segítségével az állapot helyi megtekintésére, az előbb említett portok nagyban megkönnyítenék a továbbfejlesztést.

#### Hardverek kezelése

A talajnedvesség-, és fényérzékelőből van, ami csak bináris kimenettel rendelkezik, és amelynek analóg kimenete is van. Nekem az utóbbira lesz szükségem, a bináris ugyanis csak 0 és 1 értékeket ad vissza. A rajta található potenciométerrel meg lehet adni azt a szintet, ami felett 1-es jelet ad, így jó lehet olyan alkalmazások esetén, amikor ez a szint nem változik. Az én rendszeremet úgy tervezem, hogy többféle növény gondozására is képes legyen, amelyeknél ez a szint eltérő lehet, ezért az analóg szenzort választom. Az általa szolgáltatott jelet a fogadó eszköz ADC-je alakít át digitális jellé. Minél több bites az ADC, annál pontosabb a mérés, 10 bit esetén 0-1023 az értékkészlet, 16 bit esetén 0-65535.

A Pi-n nem található analóg bemenet, így kell egy jelátalakító az analóg kimenetű szenzorok után. Erre a problémára megoldás lenne egy különálló ADC, azaz Analog-Digital Converter. Ezek az eszközök könnyen csatlakoztathatók a Raspberry Pi-hez, csupán pár kábelt kell rákötni a GPIO tűkre. Az ADC-k általában több csatornásak, így egy ADC-n keresztül több szenzor adatait is tudnánk fogadni. Amint viszont el kezdenénk bővíteni a rendszert több növénnyel, újból kábelezésre lenne szükség, esetleg több ADC-re is. Az, hogy az egyes szenzorokat melyik ADC melyik lábára kötjük tetszőleges, azaz minden ilyen bővítés kódváltoztatással járna, hogy a rendszerrel felismertessük az új inputokat. Ezen felül a növényekhez nem csak érzékelők tartoznak, hanem beavatkozók is, így érdemes lenne egy olyan növényenkénti gondozási egységet megalkotni, ami rendelkezik ezekkel a szolgáltatásokkal, és egyszerűen csatlakoztatható a Raspberryhez.

Erre a feladatra tökéletes az Arduino Nano. USB-n keresztül csatlakoztatható a Raspberryhez. Ha minden hardver irányítását az Arduinon keresztül végzem, egy egyszerűen használható interfészt nyújthatok a Raspberry számára az adatok lekérdezéséhez, öntözés, világítás irányításához, ezzel csökkentve a Raspberry felelősségét, és sokkal könnyebbé teszem a további növények hozzáadását a rendszerhez. Így a Raspberry feladata annyi lesz, hogy lekérdezi az adatokat az Arduinorol, feldolgozza ezeket, meghatározza a teendőket, amik végrehajtását kiadja az Arduino számára.

A Nano rendelkezik nyolc darab analóg lábbal, a benne található az ADC felbontása 10-bit, azaz 1024 különböző értéket vehet fel. A talajnedvesség és fényintenzitást nem szükséges túl nagy precizitással mérni, így ez elegendő. Ezen felül rendelkezik digitális portokkal, valamint tápellátást biztosító lábakkal is, így a szenzorok tápellátását meg lehet oldani róla.

A szenzorokon kívül a pumpa és a lámpa irányítását is az Arduino fogja végezni. Ezeket egy relépanelen keresztül tervezem ki-be kapcsolni. Egy ilyen panelen több relé[[6]](#footnote-6) található, és mindegyik relé rendelkezik egy lábkivezetéssel, amivel a relé vezérelhető. A vezérlőjelet a Nano digitális lábai szolgáltathatják, 5 V-os relépanelt választva pedig megoldható, hogy a tápellátást is az Arduinoról szolgáltassuk, tekintve hogy az is 5 V-os tápot kap az USB porton keresztül.

Az Arduino Nano a Raspberryvel virtuális soros porton[[7]](#footnote-7) keresztül tud kommunikálni. Amint USB-vel rácsatlakoztatjuk az Arduinot a Raspberryre, a megfelelően konfigurált csatornán küldhetünk számára üzeneteket, és fogadni is tudjuk az általa küldötteket.

Az Arduinot szoftverét úgy terveztem, hogy ha kap valamilyen üzenetet, hajtson végre utasításokat. A legegyszerűbb, ha számkódokat kap. Például az 1-es üzenetre küldje vissza a szenzorok adatait, 2-est tartalmazó üzenet hatására kapcsolja be a lámpát.

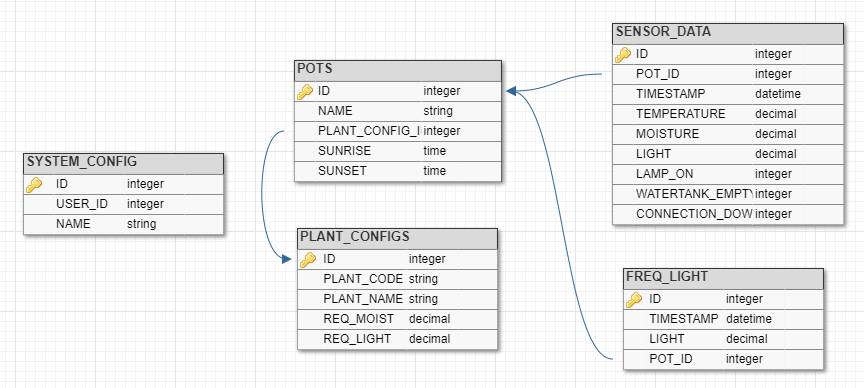
#### Az adatbázis struktúra tervezése

Milyen táblákra van szükség a Raspberryn?

A telepített egységre vonatkozó adatoknak fent fogok tartani egy táblát, amiben tárolom a nevét, valamint az azonosítóját, és a felhasználó azonosítóját.

Tárolni kell, milyen növényeink vannak. A növényeket külön gondozási logikai egységbe rendezem, ez a cserép. Minden cserépnek lehet neve, ami alapján a felhasználó megkülönböztetheti őket, a rendszerben való megkülönböztetésre pedig az azonosítója szolgál majd. Ezen felül tárolni kell minden cseréphez a tartalmazott növény jellemzőit, ami alapján a rendszer ki tudja számolni a szükséges teendőket. Ezt külön táblában tárolom, ami tartalmazza, hogy milyen erősségű fény számít kielégítőnek a növény számára, és ebből hány percnyi fényre van szüksége, valamint, hogy milyen talajnedvesség alatt szükséges a locsolás. A Raspberryn ezt a konfigurációt igazából tárolhatnám egy táblában a cseréppel, viszont a szerveren már mindenképpen külön táblában tervezem tárolni őket, így ezt a telepített egységen is így teszem.

A megfigyeléshez és a gondozási feladatok megállapításához tárolni kell a növények állapotait. Erre összesen két táblát szánok. Egy, amiből közvetlenül számolom a gondozási teendőket, ebben tárolom a cserép azonosítóját, talajnedvességet, fényt, hőmérsékletet, az aktuális időt, valamint egyéb információkat, fel van-e kapcsolva a lámpa, üres-e a víztartály, vagy van-e internet kapcsolat. A másik táblába a sűrű fénymérés során mentem a szenzor adatait, amit ürítek 10 percenként, amikor kiszámítom a napos idő mennyiségét.



14. ábra: Adatbázis modell a Raspberry Pi-n

## Szerver, webalkalmazás

A webalkalmazás feladata, hogy egy szép, könnyen kezelhető felületet szolgáltasson a felhasználónak, ahol regisztrálhat, megtekintheti a cserepeinek az állapotát, elvégezheti a szükséges beállításokat, kiadhatja a kívánt utasításokat.

A felhasználói felület három részből fog állni, a főoldalból, a bejelentkezési felületből, és az egyes cserép részletes adatait megjelenítő felületből.

A főoldalon a felhasználó egy áttekintő felületet kap, amin megnézheti a cserepeinek legaktuálisabb adatait, és innen tud tovább navigálni az egyes cserepekhez.

A cserép oldalán a felhasználó grafikonos nézetben megtekintheti a fény és talajnedvesség viszonyokat, valamint táblázatos nézetben az összes további adatot. Ugyanitt változtathatja a konfigurációs beállításokat, a cserépre valamint a növény konfigurációra vonatkozóan. A rendszer alapból tartalmazni fog növény konfigurációkat, amik közül a felhasználó választhat, de ezeket később módosíthatja

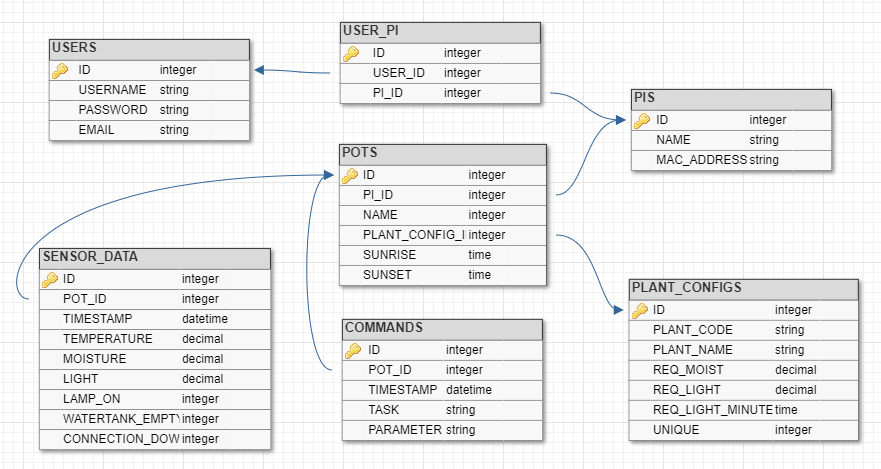
### Adatbázis

A szerver adatbázis modellje szinte megegyezik a telepített egységével, kiegészítve a többfelhasználós rendszerhez szükséges táblákkal.

Szükség van egy táblára a felhasználók számára, ahol az azonosításhoz és bejelentkezéshez szükséges adataikat tárolom, felhasználónév, jelszó, emailcím, és az egyedi azonosítójuk.

Az adatbázist felkészítem arra az eshetőségre, hogy később akár egy felhasználóhoz több telepített egység is tartozzon. Ehhez elég lenne a telepített egység táblájában fenntartani egy oszlopot a felhasználó azonosítójának, viszont ha egy kapcsolótáblát veszek fel a kapcsolat leírására, az a lehetőség is meg lesz, hogy egy egységet több felhasználó is megfigyelhessen, irányíthasson.

A fentieken kívül még négy fontos különbség van a telepített egység és a szerver adatbázisa között. A telepített egységen nincs szükség, hogy a cserepeket tartalmazó táblában tároljuk az egység azonosítóját, itt viszont muszáj, hogy a cserepeket hozzá tudjuk rendelni az egységekhez. A felhasználó által kiadott utasításokat tárolni kell, amíg azokat le nem kérdezi a telepített egység, így erre létrehozok egy táblát, amiben tárolom melyik cseréphez tartozik az utasítás, az utasítás kódját és az esetleges paramétert. Az utolsó, hogy a növénykonfigurációk között tárolni fogok előre definiált konfigurációkat, amiket a felhasználó, ha szeretné, megváltoztathat. Ahhoz, hogy az előre definiált és a megváltoztatott konfigurációkat meg lehessen különböztetni, felveszek egy új mezőt UNIQUE névvel, ami jelzi, hogy egyedi vagy sem az adott bejegyzés.



15. ábra: Szerver adatbázis modell

# Implementáció

A felhasználó szempontjából a következő képpen néz ki a rendszer használata. Első lépésként beregisztrálja magát a rendszerbe. Ezután bejelentkezik, de még nincs bekonfigurálva a telepített egység, tehát a cserepek sem, így a következő lépés számára a telepített egység beállítása lesz. Itt meg kell adni a telepített egység azonosítóját, amit a felhasználó ismer, valamint itt tudja beállítani a telepített egység nevét is. A Raspberry úgy érkezne a felhasználóhoz, hogy azon már be van állítva az azonosító. Miután beállította a telepített egység adatait, felveheti hozzá a cserepeket. Ezen a felületen meg kell adni a cserép azonosítóját, a nevét, gondozott növény fajtáját, valamint a napfelkelte és napnyugta időpontját. Minden cserép rendelkezik saját egyedi azonosítóval, amit a felhasználó a telepített egységhez hasonlóan ismerne. Eközben a felhasználó rácsatlakoztatja a Raspberryre az Arduinot. A cserép felvétele egy utasítást hoz létre a Raspberry részére, amiben az megkapja a beállítani kívánt cserép adatait. Miután a Raspberryn lefutott a cserép konfigurálása, elkezdi küldeni az adatokat a szerverre, ami hamarosan megjelenik a felületen. A főoldalon megjelenik a cserép aktuális állapota, valamint egy gomb, amivel navigálhatunk a cserép felületre. Ezen az oldalon a felhasználó megtekintheti a fényviszony és talajnedvesség változását grafikonon ábrázolva, valamint a többi adatot táblázatos formában. Továbbá itt adhat ki utasításokat öntözésre és világításra, megváltoztathatja a növény konfigurációját, valamint a cserép beállításait.

## Szerver

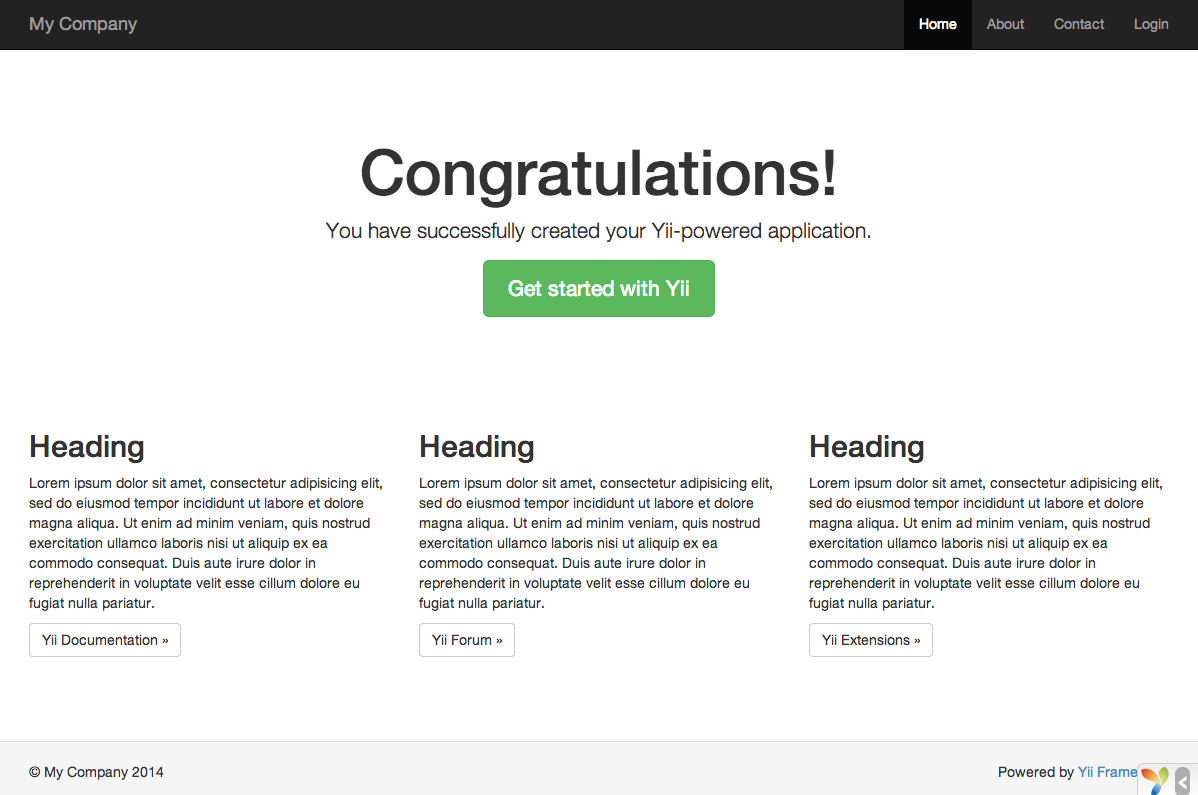
A webalkalmazás megvalósításához a PHP nyelvet választottam. Ez egy szerver oldali szkriptnyelv[[8]](#footnote-10), amit először webfejlesztésre terveztek, de később általános programozási feladatokra is elkezdték használni. A PHP kód beágyazható HTML-be, de önállóan is állhat. Amikor egy kérés érkezik a szerver számítógépre, az azon futó webszerver - jelen esetben Apache – ezt feldolgozza, és válaszként elküldi a kért erőforrást, vagy weboldalt, amiben kicserélte a PHP kódokat a futtatásuk eredményére.

A weboldal megalkotásához egy PHP keretrendszert, a Yii2-t használom. MVC[[9]](#footnote-11) tervezési mintát használ, hogy egymástól függetlenné tegye az adatok megjelenítését a háttérben történő folyamatoktól. Könnyű elindítani egy alkalmazást fejlesztését a segítségével. Egy generáló szkript lefuttatásával létrehoz egy működő alkalmazás vázat, egy főoldallal, bejelentkezési lehetőséggel, amit már meg is tekinthetünk a szerveren. Ezután megadhatjuk az adatbázis kapcsolat paramétereit, és a keretrendszer összeköti az alkalmazásunkat az adatbázissal, és a fejlesztő felületen keresztül generálhatunk minden táblához az adatbázisból modell és controller osztályt, valamint felületet a tartalmazott adatok megjelenítéséhez, beszúráshoz, módosításhoz, törléshez. A Yii keretrendszer lehetőséget nyújt külső PHP modulok használatára, ezen függőségek kezelésére a Composer nevű függőségkezelő alkalmazást használja.

Adatbázis technológiának a MySQL-t választottam. Ez ma a leginkább elterjedt adatbáziskezelő rendszer, azért rá esett a választásom, mert egyszerűen telepíthető, SQL nyelven írhatunk benne utasításokat, aminek alapjait már ismerem, így ez remek lehetőség a tudás elmélyítésére.

A fejlesztés első lépése az adatbázis felépítése volt, ehhez egy dbdesigner.net[8] címen megtalálható internetes alkalmazást használtam segítségül, ahol grafikus felületen lehet megtervezni az adatbázis sémát, amit exportálhatunk SQL szkriptként. Ezt a fájlt a MySQL felületén lefuttatva felépül az adatbázis.

A következő lépésben legeneráltam a webalkalmazás vázát, és a generált alkalmazást összekapcsoltam az adatbázissal, amihez mindössze egy konfigurációs fájlban kellett megadni az adatbázis kapcsolat paramétereit (adatbázisszerver elérési útja, adatbázis neve, felhasználónév, jelszó). A generáló szkript lefuttatása után az alábbi weboldal tekinthető meg a szerveren:

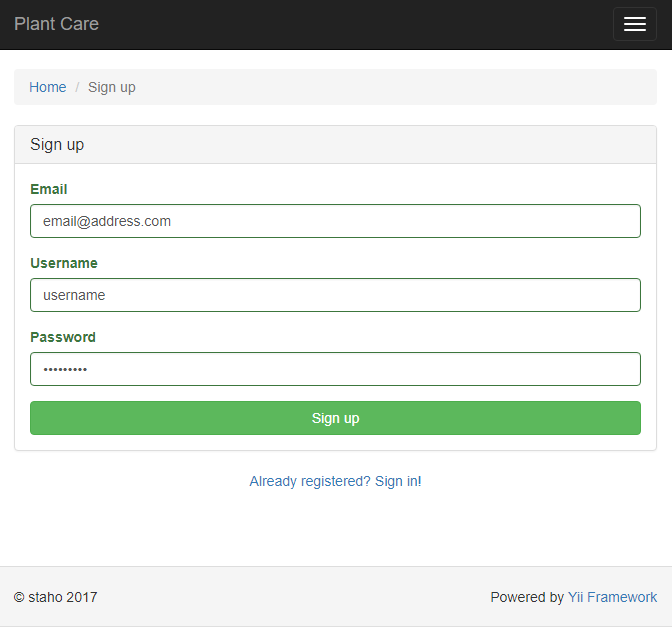


16. ábra: Generált webalkalmazás

### Felhasználók kezelése

Az új felhasználónak tudnia kell regisztrálni, erről a regisztrációról értesítést kapnia, bejelentkezni, ha esetleg elfelejtené a jelszavát, akkor ezt biztonságosan megváltoztatni. E funkcionalitás megvalósításához a Dektrium/Yii2-user[9] külső modult használtam. A Composer segítségével egy parancssori utasítás kiadásával egyszerűen hozzáadható a projekthez, amint ez megtörtént, be kell állítani az alkalmazást, hogy felhasználó osztályként az előbb hozzáadott modul osztályát használja. A Yii2-user modul saját adatmodellt használ, így az általam létrehozott felhasználó tábla feleslegessé vált, azt töröltem az adatbázisból. A modul számára szükséges táblákat egy parancssori utasítás hatására létrehozza a keretrendszer.

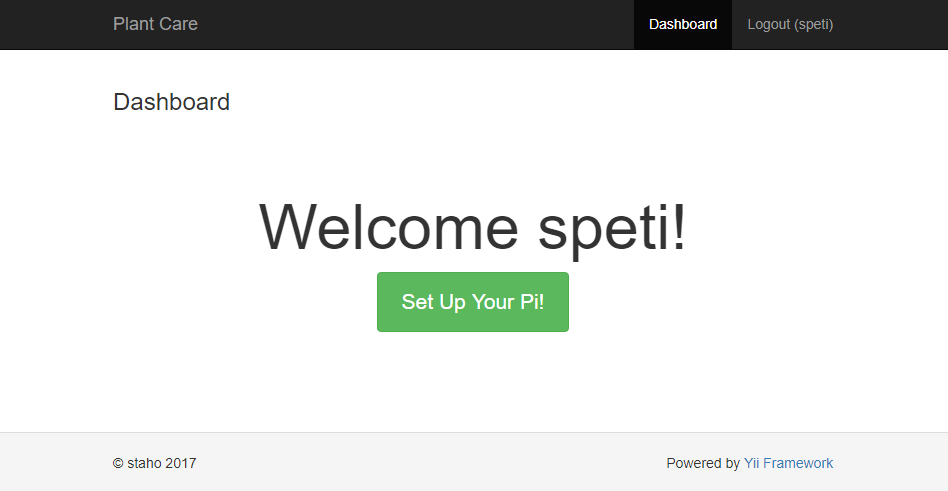
A Dektrium a regisztráció során üzenetet küld a megadott email címre a regisztráció véglegesítésére. Ahhoz, hogy az alkalmazásból leveleket tudjunk küldeni, szükség van egy levelező szolgáltatásra, aminek használatához szükséges adatokat egy konfigurációs fájlban kell beállítani. Minden Google felhasználó használhatja a Gmail SMTP szolgáltatását, így a fejlesztés idejére nem kellett levelező szervert bérelnem. A Google fiókom belépési adatait megadva már tudtam levelet küldeni az alkalmazásból.



17. ábra: Regisztrációs felület

### Fő oldal

Amíg a felhasználó nem jelentkezik be, a főoldalon egy üdvözlő üzenet fogadja, ami bejelentkezésre utasítja. Első bejelentkezés után még nincsenek beállítva sem a cserepek, sem a telepített egység, így a fő oldalon egy gombot jelenítek meg, amivel arra a felületre navigálja a felhasználót, ahol ezt megteheti:



18. ábra: Fő oldal első bejelentkezés után

Azt, hogy a felhasználó beállította-e már a telepített egységét, onnan lehet tudni, hogy létezik-e a **USER\_PI** táblában bejegyzés, ami tartalmazza a felhasználó és a telepített egység azonosítóját. A Yii alkalmazásokban az adatbázis tábláihoz generálhatunk modell osztályokat, amiken keresztül az adott táblára vonatkozó lekérdezéseket végezhetünk. Egy ilyen generált modell osztály kezdetben csak egy lekérdező függvényt tartalmaz, ami a paraméterül kapott azonosítóhoz tartozó rekordot kérdezi le az adatbázisból:

public static function findIdentity($id) {

return self::findOne($id);

}

Én a felhasználó azonosítója alapján keresem a rekordot ezért egy új függvényt hozok létre. Ehhez egy feltételes lekérdezést kell végrehajtani, amihez az fent látható **findOne** függvényt kell az alábbi módon paraméterezni:

public static function findByUserIdentity($user\_id) {

return self::findOne(['user\_id'=>$user\_id]);

}

Ezek után a fő oldal felületleíró fájljában annyit kell csupán tenni, hogy beimportáljuk a **UserPi** modell osztályt, meghívjuk a lekérdező függvényét, és az eredmény alapján kirajzoljuk a kívánt felületet:

<?php

use app\models\UserPi;

$usr\_pi = UserPi::findByUserIdentity(Yii::$app->user->getId());

if ($usr\_pi) {

if (Pots::findPotsByUserId(Yii::$app->user->getId())) {

echo '<p class="lead">Your pots:</p>';

} else {

echo Html::button('Add Your first Pot!', [

'value' => Url::to('index.php?r=pots/create'),

'class' => 'btn btn-success', 'id' => 'modalButtonPots'

]);

}

} else {

echo Html::button('Set Up Your Pi!', [

'value' => Url::to('index.php?r=pis/create'),

'class' => 'btn btn-success', 'id' => 'modalButtonPots'

]);

}

?>

## Telepített egység szoftvere

### Választott technológia

A háttérlogika megvalósításához a Python nyelvet választottam, azért, mert ez egy olyan nyelv, amivel korábban még nem volt dolgom, és szerettem volna kipróbálni. Hatalmas közösség áll mögötte, nagy az irodalmi háttere, rengeteg ingyenes könyvtárral rendelkezik megoldásként gyakori programozási problémákra, így legtöbb esetben csak az aktuális projekt sajátosságait kell implementálni. Nagyon kedvelt nyelv az IoT világában, olyan eszközök esetében, amikor erősen limitált a rendelkezésre álló számítási kapacitás, memória. Könnyű a különböző technológiák használata, adatbázis hozzáférés, HTTP kérések adása mind csak pár sor kódot kíván. Használják tudományos számításokhoz, oktatási célra, mivel egyszerű a szintaxisa, könnyen olvasható, könnyen tanulható. Adatbáziskezelő technológiának a szerverhez hasonlóan MySQL-t választottam.

### Megvalósítás

#### Rendszer indítása

A telepített egység indítása során két helyzet állhat fenn, vagy nincs bekonfigurálva egy cserép sem, vagy pedig be van állítva egy vagy több cserép. Előbbi esetben várni kell az utasításra a beállítandó cserép adataival, majd ezeket a beállításokat el kell végezni. Ha már vannak beállítva cserepek, akkor hozzájuk el lehet indítani a fő programot.

Ahhoz, hogy a felhasználónak ne kelljen ügyelnie a programok elindítására, az operációs rendszer indulására időzítek egy szkriptet, ami elindítja az programot. Ehhez a Cron[[10]](#footnote-12) nevű szoftvert használom, amit a következőképpen konfiguráltam:

@reboot /usr/bin/python /.., elérési út .../startupScript.py

A cron-feladatot a *@reboot* idejére ütemeztem, így minden induláskor le fog futni az utána következő utasítás. Ez az utasítás elindítja a **startupScript.py** nevű Python programot.   
 Ez a program felelős azért, hogy megfelelően induljon el a rendszer. Ha nincs cserép bejegyzés az adatbázisban, vár az utasításra, ha pedig már van cserép, akkor elindítja hozzájuk a fő programot.

Az adatbázishoz való csatlakozáshoz mindössze néhány sor kód szükséges:

db\_config = {

'user': 'root',

'password': '\*\*\*\*',

'host': 'localhost',

'database': 'PLANT\_CARE'

}

db\_cnx = mysql.connector.connect(\*\*db\_config)

db\_cursor = db\_cnx.cursor()

Importálom a **mysql.connector** könyvtárat, és megadom az adatbáziskapcsolat adatait. A **db\_cnx** változó az adatbázis kapcsolatot reprezentálja, a **db\_cursor**-ra pedig azért van szükség, mert azon keresztül tudjuk a lekérdezéseket végrehajtani. Egy ilyen lekérdezés a következőképpen történik:

get\_pots = ("select COUNT(\*) from POTS;")

db\_cursor.execute(get\_pots)

count\_of\_pots = db\_cursor.fetchall()[0][0]

A db\_cursor **execute** függvényének SQL utasítást adunk meg szöveges paraméterként, majd a lekérdezés eredményét a **fetchall** függvénnyel kapjuk meg. Ez egy tömböt ad vissza, amiben az eredmény sorok találhatóak. A **COUNT(\*)** lekérdezés során egy darab szám az eredmény - az adott táblában lévő elemek száma -, szóval az első sor első mezőjének értékére van szükségem, ezért a **[0][0]** indexelés.

Ha az eredmény nulla, akkor a következő utasításokat hajtom végre:

if( count\_of\_pots == 0 ):

schedule.every(1).minutes.do(checkNewPots.work())

while True:

schedule.run\_pending()

time.sleep(1)

A **schedule** egy külső könyvtár, amit időzítési feladatokra használhatunk. Először be kell állítani, hogy mit és milyen időközönként szeretnénk végrehajtani – jelen esetben percenként -, majd egy ciklusban másodpercenként „felébresztjünk”, hogy hajtsa végre az esedékes parancsokat. A **checkNewPots.py** egy saját modul, ami a **work** függvényét meghívva lekérdezi a szerverről, hogy van-e új cserép beállítására utasító parancs, ha igen, ezt végrehajtja, és újraindítja a rendszert.

A checkNewPots modul két függvényből és egy osztálydefinícióból áll. A work függvény lekérdezi az adatbázisból a telepített egység azonosítóját, majd lekérdezi a szerverről ehhez az azonosítóhoz tartozó utasításokat. Ehhez a **requests** könyvtárat használom, egy ilyen kérés kiadása a következőképpen történik.

get\_commands\_address = 'http://152.66.254.93/speti/myapi.php/COMMANDS/PIS'

payload = { 'PI\_ID' : pi\_id }

headers = {'content-type': 'application/json'}

commands = requests.get(

get\_commands\_address,

data = json.dumps(payload),

headers = headers)

A **payload** változóban megadjuk a kéréshez csatolni kívánt adatot, a **headers**-ben a beállításokat, jelen esetben, hogy json[[11]](#footnote-13) formátumban kérjük az adatokat. A **get** függvénynek megadjuk a lekérdezés helyét, valamint az előbb említett változókat.

Az utasításokat a szerver json formátumban bocsájtja rendelkezésemre. A könnyebb kezelhetőség érdekében létrehoztam egy osztályt, ami reprezentál egy ilyen utasítást. Egy új cserép utasítás a következőképpen fest:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | PI\_ID | POT\_ID | TIMESTAMP | TASK | PARAMETER |
| Parancs azonosító | Pi azonosító | 0 | Létrehozás ideje | P\_ADDED | A cserép azonosítója |

Amire nekem jelen esetben szükségem van, az az utasítás azonosítója, az utasítás típusa, valamint a paraméter, amiben a cserép azonosítója van tárolva, így az utasítás osztály az alábbi módon alkottam meg:

class Command:

def \_\_init\_\_(self, ID, task, parameter):

self.ID

self.task

self.parameter

Miután lekérdeztem az utasításokat, végigiterálok a kapott json objektumon, és a benne szereplő **P\_ADDED** típusú utasításokat hozzáadom egy tömbhöz. Ha ez a tömb nem üres, akkor a benne található elemekkel meghívom az **executeCommand** függvényt.

try:

json\_response = commands.json()

new\_pot\_commands = []

for rec in json\_response:

cmmnd = Command(rec["id"], rec["task"], rec["parameter"])

if (cmmnd.task == 'P\_ADDED'):

new\_pot\_commands.append(cmmnd)

if new\_pot\_commands:

for t in new\_pot\_commands:

executeCommand(t)

os.system("sudo reboot")

except ValueError, e:

print e

print "There aren't new pots"

Az executeCommand első lépésben lekérdezi az új cserép adatait a szerverről. Ezután lekérdezi a hozzá tartozó növény-konfigurációt, és beszúrja az adatbázisba. Ezt a cserép beszúrása előtt kell megtenni, mivel a cserép hivatkozik a növény-konfigurációra, így annak bent kell lennie az adatbázisban, hogy lehessen rá hivatkozni. Ha ez megtörtént, akkor a cserepet is felveszem, és kiadok egy törlési kérést a szervernek, a végrehajtott utasítás azonosítójával.

def executeCommand( commandToExecute ):

try:

#get new pot

payload = { 'ID' : commandToExecute.parameter }

headers = {'content-type-\*': 'application/json'}

new\_pot = requests.get(get\_pot, data = json.dumps(payload), headers = headers)

json\_pot = new\_pot.json()

db\_cnx = mysql.connector.connect(\*\*db\_config)

db\_cursor = db\_cnx.cursor()

#get plant\_config for new pot

payload = { 'ID' : json\_pot[0]['plant\_config\_id'] }

new\_config = requests.get(get\_plant\_configs\_address, data = json.dumps(payload), headers = headers)

json\_config = new\_config.json()

#put plant\_config in database

config\_to\_insert = (json\_config[0]['plant\_code'], json\_config[0]['plant\_name'],json\_config[0]['req\_light'],json\_config[0]['req\_moist'],json\_config[0]['req\_temp'], 480, json\_pot[0]['id'])

db\_cursor.execute(insert\_plant\_config, config\_to\_insert)

#find the id of the plant\_config put in previously

db\_cursor.execute(get\_plant\_config\_id, [json\_pot[0]['id']])

new\_plant\_config\_id = db\_cursor.fetchall()[0][0]

db\_cnx.commit()

#insert pot into database, with the plant\_config

pot\_to\_insert = (json\_pot[0]['id'], json\_pot[0]['name'], new\_plant\_config\_id,json\_pot[0]['sunrise'],json\_pot[0]['sunset'], None)

db\_cursor.execute(insert\_pot, pot\_to\_insert)

db\_cnx.commit()

db\_cursor.close()

db\_cnx.close()

payload = { 'ID' : commandToExecute.ID }

requests.delete(delete\_commands\_address, data = json.dumps(payload), headers = headers)

except (mysql.connector.Error) as e:

print e

Amint a work függvény végigjárta az utasítások tömbjét, és minden utasítással meghívta az executeCommand metódust, újraindítja a rendszert. Miután ez megtörtént, újra lefut a **startupScript.py**, ez alkalommal már létezik az adatbázisban cserép bejegyzés, így a másik ága hajtódik végre a programnak.

Amikor a csatlakoztatunk egy USB eszközt a Linux rendszerhez, ez az eszköz elérhetővé válik a /dev/ttyUSB\* útvonalon, ahol a csillag nullától kezdődő egész számokat jelent. Azt, hogy melyik eszköz melyik úton érhető el, tekintve, hogy tetszőleges számú Arduinot csatlakoztathatunk, nem lehet előre tudni, így kell egy módszer ezek meghatározására.

Az eszközök feltérképezéséhez a **serial.tools.list.ports** könyvtárat használom. Ahhoz, hogy egy eszközt cserép-Arduinoként azonosítsak, először megpróbálom felépíteni a soros kapcsolatot vele. Ha ez nem sikerült, az azt jelenti, hogy nem lehet Arduino, ha pedig igen, akkor elküldök neki egy kódot, amire a cserép-Arduino az azonosítójával válaszol. Ha érkezik válasz, akkor az adatbázisban frissítem a cserép bejegyzést az új elérési úttal.

Ahogy a program végzett az cserép bejegyzések frissítésével, lekérdezi az összes cserepet az adatbázisból, és mindegyik azonosítójával elindít egy fő programot. A gondozási programokat külön folyamatokban indítom el, így a párhuzamos futásukat az operációs rendszer biztosítja. Ennek megvalósításához a **subprocess** könyvtárat választottam, mely **Popen** függvénye szöveges paraméterben kapja meg, a végrehajtandó utasítást, ami egy parancssori utasítás. A fő program egy parancssori argumentumot vár indításakor, amiben meg van határozva, hogy melyik cserép gondozását látja el, így ezt még hozzáfúzöm az utasítás végéhez.

Tekintve, hogy később tetszőleges időpontban kaphat a telepített egység új cserép felvételére utasítást, a program végén megadok egy időzítést, ami tíz percenként meghívja a checkNewPots modul work függvényét ezen utasítások lekérésére és végrehajtására.

else:

#check the connected serial\_ports

list = serial.tools.list\_ports.comports()

for element in list:

device = element.device

try:

ser = serial.Serial(

port=device,

baudrate=9600,

parity=serial.PARITY\_ODD,

stopbits=serial.STOPBITS\_TWO,

bytesize=serial.SEVENBITS)

ser.isOpen()

time.sleep(3)

input = 'C'

out = ''

ser.write(input)

time.sleep(1)

while ser.inWaiting() > 0:

out += ser.read(1)

if (out != ''):

raise ValueError('')

db\_cursor.execute(update\_pots, (device, out))

db\_cnx.commit()

ser.flush()

except SerialException, ValueError as ex:

print device + ' isn\'t an arduino'

except mysql.connector.Error as ex:

print ex

#getting pots, and starting a process for each

get\_pots\_ID = ('select ID from POTS;')

db\_cursor.execute(get\_pots\_ID)

pot\_id\_list = db\_cursor.fetchall()

db\_cursor.close()

db\_cnx.close()

for element in pot\_id\_list:

id\_str = str(element[0])

subprocess.Popen("python /home/pi/plantcare\_raspberry/PLANTCARE\_RASPBERRY/baseprog.py "

+ id\_str, shell=True)

schedule.every(10).minutes.do(checkNewPots.work())

while True:

schedule.run\_pending()

time.sleep(1)

#### Gondozási program

A tervezés során a fő program felelősségeként, az adott cserépre vonatkozó adatgyűjtési és gondozási teendők vezénylését állapítottam meg. Ezeket a funciókat külön modulokba szerveztem, így a fő program feladata tényleg annyi, hogy a megfelelő időközönként elindítsa az alfeladatokat.

Négy ilyen művelet van, a gyakori fénymérés, az adatok mentése (adatbázisba és szerverre), cserépre vonatkozó utasítások lekérdezése és végrehajtása, valamint a napi gondozási teendők elvégzése. Ezek időzített indításához újból a schedule könyvtárat használom.

A első lépésben a program lekérdezi az adatbázisból a cseréphez elérési útját a soros adatkapcsolat felépítéséhez, valamint a naplemente idejét a gondozás időzítéséhez. Ezután bekonfigurálom az ütemezést, és elindítom az ütemezés végrehajtó ciklust. Amikor a schedule végrehajt egy utasítást, addig blokkolja az alkalmazást, amig az utasítás végrehajtása véget nem ér. Ez gyors lefutású kódok esetén nem jelent problémát, viszont a potenciálisan sokáig tartó folyamatok esetén már foglalkozni kell vele. A fénymérés és az adatok mentése hamar végrehajtódik, így ezeket nem lenne muszáj háttérfolyamatként elindítani, viszont az utasításokat végrehajtó és a gondozó modul adott esetben sokáig futhat, így ezek mindenképpen külön folyamatokat igényelnek, hogy ne akasszák meg a program működését. A konzisztencia érdekében, a fénymérést és adatmentést is alfolyamatként indítom. Ezeket a programokat úgy implementáltam, hogy parancssori argumentumként várják a cserép azonosítóját, valamint a cserép elérési útját a soros adatkapcsolathoz, amit ez alapján építenek fel. Úgy is meg lehetne oldani, hogy csak a cserép azonosítót adom át nekik, és az adatbázisból lekérdezik az elérési utat, viszont így elég a fő programban egy adatbázis művelet, a négy helyett.

A fénymérést a **sampleLight.py**  fájlban implementáltam, egy számkódot küldök az Arduinonak, amire az válaszol a fényszenzorból vett adattal. Ezután ezt az adatot elmentem az adatbázisba.

Az adatmentést a **persistData.py** fájlban valósítottam meg. Ehhez ki kell számítani a legutóbbi adatmentés óta eltelt idő alatti fényes percek számát. Ahelyett, hogy lekérdezném az összes adatot a fénymérés táblázatából, és a Python programban dolgoznám fel ezeket, egy összetett SQL utasítás segítségével kérdezem le, hány százaléka volt a mintáknak a kívánt szintérték felett.

select ((select COUNT(\*) from FREQ\_LIGHT

where LIGHT <= (select REQ\_LIGHT from PLANT\_CONFIGS

where ID in (select PLANT\_CONFIG\_ID from POTS

where ID = %s)) and POT\_ID = %s)\*100 / COUNT(\*))

as PERCENTAGEOFSUFFICIENT from FREQ\_LIGHT;"

Ezt az eredményt tízzel elosztva azt kapom meg, hogy az elmúlt tíz percben hány percen keresztül voltak kedvezőek a fényviszonyok (hetvenöt százalék osztva tízzel hét és fél perc). Ezután lekérdezem a többi adatot az Arduinorol, és ezeket kiegészítve az előbbi számítás eredményével elküldöm a szerverre, majd mentem az adatbázisba. Végezetül törlöm a gyakori fénymérés táblázatából az aktuális cserép azonosítójához tartozó adatokat, így az egymást követő adatmentések során nem kell azzal foglalkozni, hogy szűrjük az eredményt a legutóbbi mentés óta eltelt időszakra, mert csak olyan adatok lesznek az adatbázisban, amik azután lettek beszúrva.

A cserépre vonatkozó utasítások feldolgozása a **checkPendingCommand.py** fájlban található, és hasonlóan történik, mint a telepített egységre vonatkozó utasítások kezelése. A cserép azonosító alapján lekérem ezeket szerverről, átalakítom **Command** típusúvá, és egyenként végrehajtom, majd törlöm őket a szerverről. A jelenleg támogatott cserépszintű parancsok a következők:

* W: öntözés, paramétere az öntözés mennyisége deciliterben,
* L\_SPEC: világítás, paramétere az világítás időtartama percben,
* C\_CHNG: növény konfiguráció megváltoztatása, paramétere a szerveren található új növény konfiguráció azonosítója.

Az Arduinot úgy programoztam be, hogy ha öntözésre való utasítást kap, akkor egy deciliternyi vizet szolgáltat a növénynek. A szerverről érkező parancs paramétere tartalmazza, hány deciliter a kívánt öntözés mennyisége, így ennyiszer kell kiadni az Arduino számára az öntözés utasítást.

A világítás parancs végrehajtása során először utasítom az Arduinot, hogy kapcsolja fel a lámpát, majd a **time** könyvtár **sleep** metódusának segítségével altatom a programot a paraméterben kapott ideig, ezután utasítom az Arduinot, hogy kapcsolja le a lámpát.

A növény konfiguráció változtatása során először lekérdezem a szerverről az új növénykonfigurációt a paraméterben kapott azonosító alapján, majd ez alapján frissítem az adatbázisban a cseréphez tartozó bejegyzést.

Az utasításokat végrehajtásuk előtt két részre osztom, a várható végrehajtási idejük alapján. A világítás várható ideje több óra, és ha sorban érkezik egy világítás, öntözés, konfiguráció változtatása parancs, akkor a világítás időtartamáig a többi utasítás nem hajtódna végre. Így először a gyorsan lefutó utasításokat hajtom végre, és csak utánuk a világítást.

A növény napi gondozását a **daily.py** programban valósítottam meg. Első lépésben lekérdezem a fényes percek számát, az elvárt napfényes percek számát, az öntözési szintet és az utolsó talajnedvességet az adatbázisból, majd ezek alapján végrehajtom a teendőket. Ha az utolsó talajnedvesség kisebb, mint a kívánt talajnedvesség, fixen két deciliternyi vízzel öntözi meg a növényt. Ezután, ha nem volt elegendő mennyiségű napfény, a felkapcsolja a lámpát, majd a hiányzó idő leteltével lekapcsolja azt.

#### Az Arduino szoftvere

Az Arduino programozására a platform saját nyelvét használtam, ami lényegében C/C++, kiegészítve olyan könyvtárakkal, amik megkönnyítik a hardverek kezelését. Például amikor ki akarunk adni egy jelet a mondjuk a mikrokontroller hármas számú digitális tűjén, ahelyett, hogy a tűhöz tartozó regiszterén direkt bitműveleteket végeznénk, csak annyit kell tennünk, hogy beállítjuk a tűt mint kimenetet: ***pinMode(3, OUTPUT)*** és kiadjuk a jelet: ***digitalWrite(3, 1)***.

Az Arduino feladata, hogy a soros kapcsolaton keresztül kapott utasításkódokra reagálva adatokat gyűjtsön be a csatlakoztatott szenzoroktól, vezérelje a lámpát valamint a pumpát a relére küldött jel segítségével. A soros kommunikációhoz az Arduino platform **SPI** könyvtárát használom, ezen felül még egy külső könyvtárat importálok be, ami a hőmérő használatához szükséges. A mikrokontroller tűivel végzett műveletek megkönnyítésére és a program olvashatósága érdekében ezekhez konstansokat definiálok. Ezen felül globális változóban tárolom a cserép azonosítóját, és egy bool típusú változóban a lámpa állapotát, hogy éppen fel van-e kapcsolva, vagy sem.

Egy Arduino program minimum két függvényből, a **setup** és **loop** függvényből áll. A setupfüggvény egyszer fut le az Arduino indításakor, itt állíthatjuk be az eszköz kezdeti állapotát, például a tűk irányát, valamint itt érdemes felállítani a kapcsolatot a csatlakoztatott eszközökkel. Esetemben itt építem fel a soros kapcsolatot, valamint itt csatlakozok a hőmérőhöz a reléhez kapcsolódó tűk irányát kimenetre.

A setup függvény befejeztével a végrehajtás a loop függvénnyel folytatódik ami ciklikusan hajtja végre a benne található utasításokat. Minden ciklus elején megvizsgálom, érkezett-e üzenet a soros csatornán, és ennek megfelelően eldöntöm, mi a teendő. Amikor egy karaktert küldök az Arduinonak a soros porton, a mikrokontroller oldalán a karakter ASCII[[12]](#footnote-14) kódja jelenik meg. A fejlesztés során az Arduino működését a számítógépemhez csatlakoztatva teszteltem, és ott minden kiküldött kód elérte célját, ezért megírtam úgy a szoftvert, hogy nullától (ASCII 48) hatig minden számhoz rendeltem egy-egy funkcionalitást. A nullás kódra visszaadta a cserép azonosítóját, az egyes kódra visszaadta JSON formában a szenzorok adatait, hármas kódra felkapcsolta a lámpát, és így tovább. Viszont amikor a Raspberryhez csatlakoztattam, a küldött karakterek csupán töredékét kapta meg az Arduino. Ezt próbáltam orvosolni jobb minőségű kábellel, de nem javult a helyzet, valószínűleg a Raspberry gyengébb minőségű hardvere miatt nem olyan stabil a soros adatátvitel. Így végül kísérletezéssel sikerült megtalálni azokat a karaktereket, amiket minden alkalommal sikeresen átmennek a csatornán. Az Arduino interfésze így a következőképpen alakul:

|  |  |  |
| --- | --- | --- |
| Kód | ASCII | Művelet |
| 1 | 49 | Szenzoradatok küldése JSON formátumban. |
| 2 | 50 | Fényszenzor minta küldése. |
| 7 | 55 | Lámpa felkapcsolása. |
| 8 | 56 | Lámpa lekapcsolása. |
| C | 67 | Cserép azonosító küldése. |
| E | 69 | Öntözés (1dl). |

# Tesztelés

Tettelés

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
* Kinézet ellenőrzése PDF-ben: a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

Irodalomjegyzék

1. Edyn: <https://www.edyn.com/>
2. GreenIQ: <http://greeniq.co/>
3. Plantlink: <http://myplantlink.com/>
4. Gideon: <http://www.gideon.ai/>
5. Muzzley: <https://smarthome.muzzley.com/>
6. Grove Smart Plant Care Kit: <https://www.seeedstudio.com/Grove-Smart-Plant-Care-Kit-for-Arduino-p-2528.html>
7. JSON: <https://hu.wikipedia.org/wiki/JSON>
8. DbDesigner.net: <https://dbdesigner.net>
9. Yii2-user: <https://yii2-user.dektrium.com/>

Függelék

1. Közösségi finanszírozásról beszélünk, amikor egy projektre úgy gyűjtik össze a szükséges pénzt, hogy sok ember járul hozzá kisebb összeggel, tipikusan interneten keresztül. [↑](#footnote-ref-1)
2. Gideon [4] és Muzzley [5] mindkettő olyan alkalmazás, amin keresztül az otthonunkban található okos eszközöket irányíthatjuk. [↑](#footnote-ref-2)
3. A négy-utas kézfogás során egy üzenet küldése, amikor a fogadó egy PUBLISH üzenetet kap, visszaküld egy PUBREC (publish redeived) üzenetet, és eltárolja az üzenetben kapott csomag azonosítóját, hogy elkerülje az esetleges dupla feldolgozást, ha többször kapná meg az üzenetet. A küldő a PUBREC üzenetre egy PUBREL (publish release) üzenettel válaszol, amint ezt a fogadó megkapja, törölheti a tárolt állapotokat, és visszaküldi a PUBCOMP (publish completed) üzenetet. Amikor a küldő megkapja, ezt az üzenetet, ő is törölheti az üzenet adatait. [↑](#footnote-ref-3)
4. Az adatokat célszerű adatbázisban tárolni. Adatbáziskezelő szoftver segítségével könnyű átlátható struktúrát adni az adatainknak, ezzel megkönnyítve a velük való műveleteket. [↑](#footnote-ref-4)
5. Az adatbázis legtöbb adatot tároló táblájának, az állapottároló táblának a sor mérete is maximum pár tíz byte, tekintve, hogy egyszerű számadatokat és dátuminformációt tárolok majd benne. Még ha száz byte sormérettel is számolunk, akkor is 144\*365\*100 byte adatmennyiség jön ki egy évre egy növényhez, ami felfelé kerekítve 5.3 Mbyte, így több növény gondozása esetén is sok év, mire akár a 100 Mbyte méretet eléri az adatbázis. [↑](#footnote-ref-5)
6. Külső vezérlő jel hatására összeköt vagy megszakít egy nagyobb teljesítményű áramkört. [↑](#footnote-ref-6)
7. Kommunikációs csatorna, amin olyan adatátvitel, amely során egy időben egy bit adat továbbítódik. [↑](#footnote-ref-7)
8. Közvetlenül a forráskódú változatban, futásidőben értelmezett és gépi kódú utasításokká fordított nyelv. [↑](#footnote-ref-10)
9. Model-view-controller egy szoftvertervezési minta, amit felhasználói felületek implementálásához használnak. A model osztály reprezentálja az adatszerkezetet, a view a felületet definiálja, a controller pedig kezeli a felhasználói cselekvéseket. Ez a felosztás segít függetleníteni az alkalmazáslogikát a megjelenítéstől, hogy a kettőt anélkül lehessen megváltoztatni, hogy a másikra hatással legyen. [↑](#footnote-ref-11)
10. Unix alapú rendszereken megtalálható ütemező szoftver. Egy szöveges fájlban adhatunk meg cron-feladatokat, egy ilyen feladat tartalmazza a végrehajtandó utasítást, valamint a végrehajtás időpontját. [↑](#footnote-ref-12)
11. A JSON (JavaScript Object Notation) egy kis méretű, szöveg alapú szabvány ember által olvasható adatcserére. A JavaScript szkriptnyelvből alakult ki egyszerű adatstruktúrák és asszociatív tömbök reprezentálására.[7] [↑](#footnote-ref-13)
12. Az ASCII kódolás 128 szövegkaraktert képez le 0-tól 127-ig egész számokra. [↑](#footnote-ref-14)