ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

OE-KVK
2023.

Bereczki Gergely
T004428/FI12904/K

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Bereczki Gergely
Szakdolgozat száma: SZD2309040923393298[redacted]
Törzskönyvi száma: T004428/FI12904/K **Neptun kódja:** [redacted]

Szak: villamosmérnöki
Specializáció: Műszer-automatika specializáció

A dolgozat címe: Automatizált napelemes magaságyás
A dolgozat címe angolul: Automated solar powered plantbed
A feladat részletezése: Készítsen egy automatizált napelemes magaságyásA projekt, amely IoT technológiát alkalmazva egy szerveren keresztül lehet konfigurálni az termesztés jellemzőit. A szerver oldalon a termesztéssel kapcsoltos statisztikákat lehet követni. Az egész rendszer működése akkumulátorról történjen, amelyet fotovoltaikus cella tölt, amely cella követi a nap mozgását maximális töltési teljesítményt garantálva.

Intézményi konzulens neve: Sándor Tamás

A kiadott téma elévülési határideje:2025. 12. 31.
Beadási határidő: 2023. 12. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2023. 10. 24. P.H

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Kommunikációs Rendszerek Intézet

.....
Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

.....
belső konzulens

.....
külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZETI
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

HALLGATÓI NYILATKOZAT

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

Budapest, 2023.11.24.

.....
hallgató aláírása

TARTALOMJEGYZÉK

1 BEVEZETÉS

1.1 A probléma ismertetése

Napjainkban egyre divatosabb zöldségek otthoni környezetben való termesztése. Számtalan ember nevelget otthon paradicsomot, paprikát, esetleg uborkát. De mi van azokkal az emberekkel, akik lakásban élnek és csak egy erkélyük van, és nincs elég rendelkezésre álló földterület a termesztésre.
Ilyen esetekben megfelelő megoldást kínálhat egy magaságyás kialakítása, amely lehetőséget ad arra, hogy az ember viszonylag kis területen termeszthessen növényeket a helymegtakarítás jegyében, akár felfelé építkezve.

1.1. ábra: Egy átlagos magas ágyás

Ez a fajta termesztési módszer egyre nagyobb népszerűségnek örvend az emberek körében, projektem pedig ennek a módszernek a továbbfejlesztését hivatott elérni, egy napelemes kialakítású rendszerrel.
Az adatok vizualizációján keresztül lehetőség adódik azok kielemzésére, amelyet aztán a termesztési körülmények optimalizálására is fel lehet használni, ezáltal jobb, gazdagabb terméshozamra szert téve.

Az elmúlt években egyre nagyobb figyelem fordítódik a bio élelmiszer termékek vásárlására és fogyasztására. A kizárólag természetes anyagokkal termesztett növények és egyéb termékek ára azonban többszöröse is lehet nem bio versenytársaikéhoz képest. Az olyan emberek, akik nem szeretnének a fogyasztói társadalom részesei lenni, gyakran fordulnak az önálló termesztés eszközeihez. A saját munka gyümölcse mindig édesebb, és az ember pontosan tisztában van vele, hogy mit fogyaszt. Azt azonban a laikus emberek is beláthatják, hogy növényt termeszteni semmilyen esetben sem egyszerű. A gondozás és odafigyelés mellett a szerencse is meghatározó faktorként szerepel az egyenletben, ami azonban bizonyos mértékben kontrollálható abban az esetben, ha lehetőségünk van a termesztett növény környezeti tényezőinek monitorozására és befolyásolására. A megfigyelt értékek alapján lehetőség nyílik a megfelelő beavatkozás végrehajtására, amennyiben azt szeretnénk, hogy a növény megfelelő mennyiségű és minőségű termést hozzon. Ilyen feltételek közé tartozik a gondos öntözés, a növény számára ideális hőmérséklet biztosítása, valamint az optimális páratartalom megléte.

A mért adatok visszamenőleges megtekintésével lehetőség adódik a hozott termés és a körülmények közötti párhuzam megfigyelésére. Megfelelő analitikus szemlélettel olyan következtetések vonhatóak le, amelyek alapján a termesztési tényezők – és ezáltal a terméshozam is – optimalizálhatóak.

1.2 Specifikáció

Az általam tervezett rendszert igyekeztem úgy kialakítani, hogy az előzőekben felsorolt feltételeket többnyire meg tudja teremteni, de legalább visszajelzést adjon a felhasználónak, hogy az szükség esetén még időben meg tudja tenni a kellő lépéseket az esetleges problémák elhárításának érdekében.
A föld optimális nedvességtartalmának megtartásáért automata öntözőrendszer felel. A locsolás időpontja és időtartama felhasználó által beállítható értékek. A konfigurációk létrehozása egy interneten keresztül elérhető felhasználói felületen lehetséges. A konfigurációkat és üzeneteket képező adathalmazok relációs adatbázisban tárolódnak.
A föld nedvességtartalma mellett a levegő hőmérséklete és páratartalma is rögzítésre ke-

rülnek. A felhasználó ezáltal kvázi-valós időben követheti nyomon az egyes környezeti tényezők alakulását.

A maximális teljesítmény elérésének érdekében a rendszer képes lesz a felszerelt fotovoltaikus cella horizontális és vertikális mozgatására, törekedve arra, hogy a napfény beesési szöge minden pillanatban a legoptimálisabb legyen.

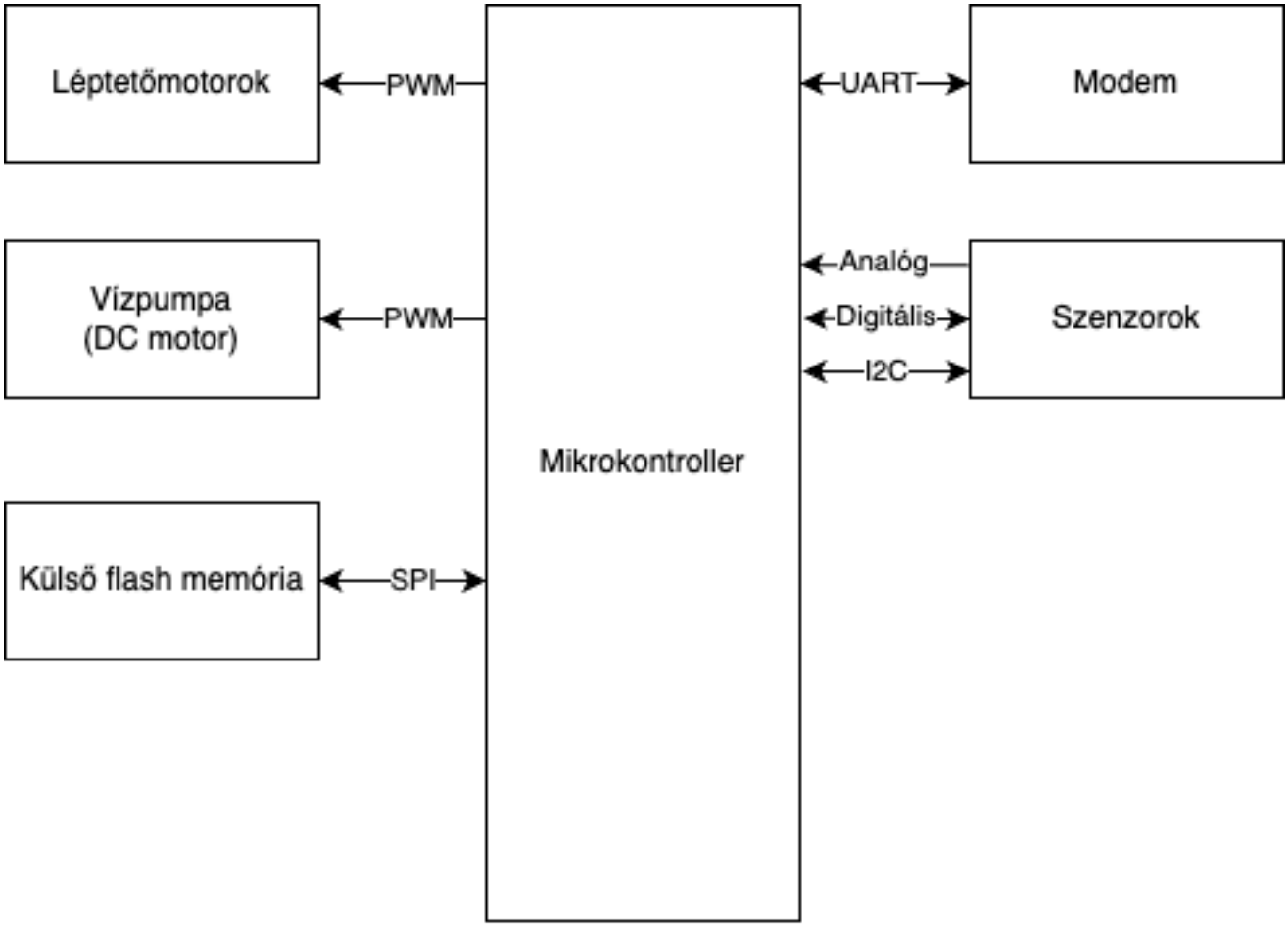
A modul által szolgáltatott adatok webes felületen kerülnek vizualizációra, különböző személyre szabható grafikonokon keresztül.

2 HARDVER

2.1 Logikai rendszerterv

Ebben az alfejezetben a különböző hardverkomponensek logikai funkciói és azok tervezési szempontjai kerülnek ismertetésre. A tervezés során kellő figyelmet kell fordítani a mikrokontroller és az egyéb integrált áramkörök közötti megfelelő vezérlési módszerek és kommunikációs csatornák kialakítására. Alapvetően minden feladat elvégzése a mikrokontrollertől indul, az irányítja a hozzákapcsolódó perifériákat. Ilyen feladatok közé tartozik a szerverrel folytatott kommunikáció, öntözés, illetve bármilyen környezeti változó fizikai mérése.

A 2.1 ábrán látható a logikai rendszerterv magas szintű blokkvázlata, amely felületesen szemlélteti az egyes rendszerelemek közötti kapcsolatot és a kommunikáció módját.



2.1. ábra: A hardver logikai rendszertervének blokkvázlata

2.1.1 Öntözés

Az öntözőrendszer összeállítása olyan módon kell történjen, hogy lehetőleg a földfelület minden pontját víz érje öntözéskor. Az öntözőrendszer "sprinkler" jellegű, lehetőleg állítható kiömlési sebességgel. Az öntözővíz tárolása külön tartályban történik, amelynek aljára beszerelt bináris lebegő szenzor jelzi azt az állapotot, amikor már nincs elegendő víz a tartályban az öntözéshez. A víz emelését vízpumpa végzi, amelynek meghajtása erre a célra tervezett áramkörrel történik. Az áramkör vezérlése impulzus-szélesség modulációs eljárással kerül kivitelezésre.

2.1.2 A nap követése

A nap pozíciójának lekövetése fényellenállásokkal történik. A mikrokontrolleren futó firmware az ellenállások mért értékei szerint határozza meg a megfelelő pozíció beállítását. A mozgatás horizontális és vertikális irányban történik. Mindkét tengelyen való mozgatásért egy-egy léptetőmotor felelős. A motorok működtetésére ebben az esetben is céláramkör használata szükséges, amely szintén impulzus-szélesség modulációval kerül vezérlésre.

A napelem mozgatására használt mechanika megtervezése nem képezi részét a szakdolgozatnak, így a tervezés során fekete doboz modellként kezelendő.

2.1.3 Kommunikáció a szerverrel

A szerverrel való kapcsolatteremtés eszköze a mobilkommunikáció. Napjainkban a legelterjedtebb nagy sávszélességű mobilkommunikációs hálózatok a 2G, 3G, 4G, illetve az 5G. A projekt tervezése és a modem kiválasztása során a 2G és 5G hálózatok nem képeztek szempontot, ugyanis előbbi egyre több országban kerül kivonásra, illetve nagy energiaigényű protokollnak mondható, míg utóbbi egy viszonylag új hálózat, így lefedettsége nem biztos, hogy minden országban optimális, és nehezebb olyan modemet találni amely támogatja ezt a fajta kommunikációt.
A piacon elérhető nyomtatott áramkörre integrálható modemek nagy része több fajta kommunikációs protokoll használatát teszi lehetővé. A projekt során a modem aszinkron soros kommunikációval (UART) kerül irányítás alá. Ez a protokoll viszonylag egyszerű, de mégis megbízható felületet ad a vezérlőegység és a modem közötti kommunikációra. A modem vezérlése szabványosított AT parancsokkal történik. A modem kiválasztásánál fontos szempont, hogy az támogassa a 3GPP TS 27.007 és 27.005 sztenderdeket[1]. Ezek olyan sztenderdek, amelyek folyamatos internetkapcsolatot valamint SMS küldést/-fogadást tesznek lehetővé egy AT parancsokból álló programozási interfészen keresztül. Ez azért fontos szempont, mert számos - az ilyen modemek vezérlésére kialakított - előre megírt könyvtár lelhető fel az interneten, amelyek használata merőben lerövidíti a fejlesz-

tési folyamatok időtartamát.

2.1.4 Memória

A projektben nagyobb volumenű adathalmazok feldolgozására kerül sor. A konfigurációk tárolásához és az egyes üzenetek pufferéléséhez külső, nem-felejtő memória integrálása szükséges[2]. Az ilyen chipek szintén többféle kommunikációs interfésszel kaphatóak. A legelterjedtebb az SPI, illetve QUADSPI és az I^2C. A projekt szempontjából a QUADSPI nem került szóba mert túl komplex, illetve a rendszernek nem szükséges maximalizálnia a memória írás/olvasás sebességét olyan mértékben, hogy az lényeges előnyökkel járjon. A projekt egyszerűségéből kifolyólag a tervezés során nem élvez előnyt egyik kommunikációs protokoll sem, a döntő szempont inkább az áramkör ára, és az implementáció egyszerűsége.

2.1.5 Szenzorok

A felhasznált szenzorok értékei különböző módszerekkel olvashatóak le. A föld nedvességtartalmát mérő szenzor analóg jelet szolgáltat a mikrokontroller felé, míg a víztartályban lévő öntözővíz kritikus szint alá való csökkenését binárisan visszajelző lebegőkapcsoló jelzi. A levegő páratartalmának és hőmérsékletének mérését egy erre a célra kialakított integrált áramkör végzi. A komponensek adatainak kiolvasása valamilyen soros porti kommunikáción keresztül valósul meg, mint az I^2C vagy SPI, ez a kiválasztott áramkörtől függ.

2.2 Fizikai rendszerterv

2.2.1 Külső tápellátás

Az akkumulátor szolgáltatja a rendszer bemeneti tápfeszültségét. Kiválasztásakor lényeges szerepet játszott a kis méret, ugyanakkor a megfelelő kapacitás megléte. Bár a tervezett rendszer nem igényel folyamatos nagy teljesítményű tápellátást, előfordulhat, hogy az időjárás akár több napig is felhős marad, így a termelt energia nagy mértékben lecsök-

ken a napos időszakokhoz képest. Ilyen esetben, amennyiben az akkumulátor kapacitása nem elegendő, előfordulhat, hogy a fotovoltaikus cella nem képes megfelelő mennyiségű áramot termelni és az akkumulátort tölteni, ami a tápellátás megszűnéséhez vezethet.
Az akkumulátor kapacitásának megválasztásakor szem előtt kellett tartani, hogy általános ajánlások szerint az optimális töltőáram az akkumulátor kapacitásának tíz-húsz százaléka. Ezt a töltőáramot mind a fotovoltaikus cellának mind a töltést vezérlő áramkörnek képes kell legyen biztosítani.

A rendszer egyik legmeghatározóbb eleme a tápfeszültség, valamint nagyságának megválasztása. A fő szerepet ebben a napelem és a tápfeszültséget szolgáltató akkumulátor játssza. Ennek a két alkatrésznek a kiválasztására párhuzamosan került sor, ugyanis paramétereik merőben függnek egymástól. Az akkumulátor esetén a kapacitás mellett fontos tényező az ideális töltőáram biztosítása. Általánosságban az ajánlott áramerősség[3] egy ólomsavas akkumulátor esetén 0,1C-0,2C közé esik, ahol C az akkumulátor kapacitása amperórában.Mindezek mellett számos olyan töltési algoritmus létezik amely időszakosan több vagy kevesebb töltőáramot alkalmaz, ez készülékenként változhat.
A tervezés megkezdésekor a financiális korlátok mellett meghatározó szempont volt a fotovoltaikus cellák, valamint a gél, illetve ólomsavas akkumulátorok elérhetősége a piacon. Az említett komponensek nagyrészt 12 és 24 voltos változatban érhetőek el. A 24 voltos kialakítások általában jóval nagyobb teljesítmény megtermelésére és leadására képesek mint amennyire a tervezett rendszernek várhatóan szüksége lesz. Ez a fotovoltaikus cella esetében jelentős méretet is magával hordoz. Ezek alapján egyértelmű volt, hogy a 12V-os tápfeszültség lesz megfelelő a projekthez. Az akkumulátor tekintetében fontos volt, hogy az optimális kapacitás megtartása mellett a lehető legkisebb méretű kerüljön beszerzésre, mivel egy magaságyásnak az is lényege, hogy kis helyet foglal. Rövid keresés után úgy tűnt, hogy ezeknek a feltételeknek leginkább a robogókba szerelhető akkumulátorok felelnek meg, amelyek szerencsére egészen olcsón beszerezhetőek és a méretük is megfelelő. A piacon a legtöbb ilyenfajta akkumulátor sztenderd kapacitás értékekkel érhető el, mint 1.2Ah, 7.2Ah,10 és 12Ah. Végezetül a Green Cell által gyártott 12V 10Ah kapacitással rendelkező akkumulátor került beszerzésre. A kapacitás kiválasztása becslés és tapasztalatok alapján történt a következő gondolatmenet szerint.

Egy robogó kikapcsolt állapotban körülbelül 20-50 mA áramot fogyaszt állandó jelleggel, és még így is képes hetek után akár 190 amper indítóáramot leadni. A tervezett rendszer optimális esetben, alvó állapotba kapcsolva akár mikroamperes nagyságrendbe is képes redukálni az áramfogyasztását, és csak időszakosan ébred fel rövid időre méréseket végezni, illetve viszonylag rövid ideig motorokat vezérelni.
A fent leírt becslések alapján a kiválasztott akkumulátor elegendő kell, hogy legyen a rendszer megtáplálására.

2.2.2 Fotovoltaikus cella

A fotovoltaikus cella kiválasztásakor a legfontosabb szempont - a névleges feszültség mellett - a maximális teljesítmény, amelyet a napelem képes leadni. Mint ahogyan az már az akkumulátor esetében említésre került, a megfelelő töltőáram megléte elengedhetetlen az optimális töltés és az akkumulátor élettartamának maximalizálása érdekében. Mivel a projektnek célja, hogy a cella két tengelyen is forgatható legyen, fontos tényező a napelem tömege. A névleges teljesítménnyel arányosan rohamosan nő a cella mérete, ezáltal a tömege, nem utolsó sorban pedig az ára is. Szerencsére napjainkban egészen egyszerűen vásárolhat az ember kisebb, hobbiszerű felhasználásra szánt napelemeket kedvező áron.
Az előbbiekben már említésre került, hogy az ajánlott töltőáram 0,1C-0,2C, ami ebben az esetben 1-2 amper áramot jelent. Ezek alapján a kiválasztott cella maximális teljesítménye $2A * 12V = 24W$ lehet. Rövid keresés után került a választás az SK20MONO gyártó egyik termékére. A katalógusában 10 wattól egészen akár 350 wattig találhatóak meg napelemek. A 20 watt teljesítményű, mindössze 2 kilogramm tömegű 420*350*25 milliméter méretű cella megfelelő választásnak tűnt. A maximális áram amit a napelem le tud így adni 1.6 amper, amely a töltési áramra vonatkozó ajánlott határértékek között van.

2.2.3 Töltésvezérlő

A napelem mellett egy megfelelő töltésvezérlő áramkör kiválasztása is fontos. A töltésvezérlő feladata, hogy az akkumulátort és napelemet rácsatlakoztatva stabilizált tápellá-

tást biztosítson az áramkör részére, és mindezek mellett az akkumulátor töltését is képes irányítani. A projekt megkezdésekor a töltésvezérlő is egyedileg megtervezett áramkörként volt számba véve, azonban a rendszer komplexitását és a leadási határidőt figyelembe véve inkább egy kereskedelmi forgalomban kapható eszközre esett a választás. Mivel manapság egyre divatosabb napelemes hobbi projekteket készíteni, nagy számban kaphatóak olcsó töltésvezérlő eszközök, amelyek az akkumulátor töltése mellett stabilizált kimenetet nyújtanak a terhelés csatlakoztatására is. A projekthez választott áramkör 120W maximális teljesítmény leadására képes, többféle savas akkumulátort képes tölteni, és mindemellett időzíthető a terheléshez tartozó kimenet engedélyezése/tiltása. Természetesen az eszköz folyamatos visszajelzést ad a töltés és az akkumulátor töltöttségének állapotáról.

2.2.4 Mikrokontroller

A projekt elkészítése során a mikrokontroller kiválasztásának főbb szempontjait képezte a szükséges kommunikációs protokollok – SPI, UART, I^2C – megléte, a kis fogyasztás, valamint elegendő mennyiségű általános felhasználású ki-és bemenet. Fontos volt még, hogy az MCU rendelkezzen RTC modullal, analóg csatornákkal, illetve legyen szélesköről támogatott különböző keretrendszerek esetén, legyen szó fejlesztői környezetről vagy driverekről. Szerencsére a munkahelyi körülmények lehetőséget nyújtottak kétféle mikrokontrollerből való választásra, ezzel jelentősen csökkentve a projekt költségeit.

A választható vezérlők közül az egyik az Espressif által gyártott ESP32-WROOM sorozat egy tagja volt, a másik egy STM32L431. Az ESP32 kezdetben meglehetősen csábítónak bizonyult. A modulcsalád egészen egyedi módon, ugyanakkor nagyon alaposan dokumentált. Az interneten rengeteg példa kód és segítség lelhető fel, ugyanis egy rendkívül közkedvelt mikroprocesszornak számít, főleg a WiFi és Bluetooth applikációk körében. A gyártó saját maga által fejlesztett nyílt forráskódú keretrendszert is biztosít, amelynek segítségével nem szükséges regiszter szinten programozni a mikrokontrollert. Ezek mellett rengeteg előre megírt "driverteket" is tartalmaz a könyvtár, például LED fények és motorok vezérléséhez. A fő processzor mellett helyet foglal egy segédprocesszor

amely a vezeték nélküli kapcsolatok irányításáért felelős, ez rengeteg terhet levesz a fő egységről, ezzel sokkal hatékonyabbá téve a rendszer működését. Az ESP termékek egyik nagy előnye, hogy a fizikai lábak egy úgynevezett GPIO mátrixba vannak kötve, aminek segítségével a chip szinte bármelyik lába konfigurálható bármilyen periféria ki és bemeneteként.
Mindezen előnyök ellenére az ESP32 nem tűnt optimális választásnak, ugyanis a tervezett rendszer mobilkommunikációt alkalmaz, ezáltal a WiFi és Bluetooth funkciók teljesen kihasználatlanok lennének.

A következő helyen szereplő kandidátus egy STM32L431 processzor volt[4], az STMicroelectronics gyártótól. Az STM családba tartozó mikrokontrollerek használata széles körben terjedt el ipari és hobbi alkalmazásokban egyaránt. A kínálatban megtalálható összes termék rendkívül részletes és alapos dokumentációval rendelkezik, mindemellett a gyártó több saját fejlesztésű segédprogramot is szolgáltat kódírásra, programozásra és debuggolásra is. A fejlesztői környezetnek több hasznos funkciója is van amelyek egy későbbi fejezetben tárgyalásra is kerülnek. Az L széria különlegessége, hogy extra alacsony fogyasztással rendelkeznek. Katalógusadatok szerint a processzor olyan standby alvómóddal rendelkezik, amelyben 280 nA fogyasztással is képes üzemelni, a bekapcsolva hagyott perifériáktól függően. Mindezek mellett a munkahelyen forgalmazott eszköz is STM mikrokontrollert használ, így a cég által fejlesztett kód jó alapot szolgáltathat ötletek merítéséhez.
A fent leírt szempontokat figyelembe véve a kiválasztott mikrokontroller az STM32L431, 48 lábas LQFP kialakításban.

2.2.5 Szenzorok

A tervezett rendszer által mérendő fizikai tényezők a következőek:

- hőmérséklet
- páratartalom
- termőföld nedvességtartalma
- öntözővíz tartályszint
- fénymennyiség

A felsoroltak mérésére a megfelelő szenzor kiválasztása leginkább a beszerzési időtől és az érzékelő árától függött.

A termőföld nedvességének mérésére egy kimondottan erre a célra gyártott áramkör felhasználása volt tervben. Az áramköri alkatrészek megrendelésére használt weboldalon raktárról kapható nedvesség szenzor a DFRobot kínai gyártó által kínált SEN0308. Ez egy rendkívül kompakt és az árához képest minőségi szenzornak tűnő alkatrész. Azzal a nagy előnnyel rendelkezett más gyártók termékeihez képest, hogy a szenzor nyáklapjának és a vezetékezés találkozásánál betokozták a készüléket, így védelmet nyújtva víz és egyéb fizikai behatások ellen. Sok gyártó teljesen csupaszon hagyja ezt a részt, csak simán ráforrasztják a kábelt a nyáklapra, ami egy nedves környezetben nem feltétlenül egy előnyös megoldás. A megfelelő tokozás ellensúlyozta azt a tényt, hogy ez a szenzor valamivel drágább volt versenytársaihoz képest.

A hőmérséklet és páratartalom mérésére korábbi munkák során szerzett tapasztalat alapján egyértelmű választás volt a Texas Instruments által kínált szenzorok egyikének felhasználása. A HDC1080 integrált áramkör hőmérséklet és relatív páratartalom mérésére képes[5], mindezt $\pm$ 2% a páratartalom és $\pm$0,2 °C eltérési hibával a hőmérséklet mérésének esetében. A mért értékek leolvasása I^2C protokollon keresztül történik. Előnyös tulajdonságai közé sorolható még a kiemelten alacsony energiafogyasztása is, amely $710nA - 1,3\mu A$ nagyságrendbe esik.

A víztartályban lévő vízmennyiség mérése két-állapotú kapcsoló szenzorral került megoldásra. A szenzor nem konkrét űrtartalom mennyiséget mér, csupán a tartály aljára beszerelve ad bináris visszajelzést arról, hogy az ott lévő vízmennyiség a kritikus szinthez képest több vagy kevesebb. Az angol szakirodalomban "float sensor"[6] néven ismert érzékelő két végpontja a felhajtó erő hatására rövidzárat képez abban az esetben, ha a víz ellepi a szenzort.

A fénymennyiség mérésére egyszerű fényellenállások használata lett tervezve. A dokumentáció írásakor még nem volt lehetőség annak meghatározására, hogy ez mennyire hatékony megoldás a fénymennyiség mérésére. Elképzelhető, hogy a megvalósításkor a fotoellenállások értékei nem tükrözik hűen a valóságot a megvilágítottság tekintetében, ezért azt csak mérésekkel lehet biztosítani. Amennyiben a mért értékek nem bizonyulnak megfelelőnek, egy alternatív megoldáshoz kell fordulni. A fénymennyiség mérésére megfelelő opció lehet egy fotodióda, vagy lux szenzor használata, azonban szem előtt kell tartani, hogy ez jelentősen megnövelheti a projekt költségeit.

2.2.6 Modem

A modem kiválasztása hasonlóképpen történt a mikrokontrollerhez. A munkahelyi projektek során használt uBlox SARA modemek már bizonyították megfelelőségüket. Az ettől a gyártótól származó alkatrészek megfelelnek a korábban leírtak szerint támasztott követelményeknek,így egy remek megoldást kínálnak széleskörű támogatottság mellett.

A munkahelyen régóta használt uBlox SARA modemek bizonyítottan jó eszközök, amelyek számos IoT projekt során kerültek már felhasználásra. A konkrét modell egy uBlox SARA R412.[7] Ez a modem megfelel a logikai rendszertervben támasztott követelményeknek, ezen felül pedig rendkívül jó dokumentációval rendelkezik mind szoftver, mind hardver tekintetében. A programozási utasítások mellett a gyártó egy tervezési útmutatót is készített, amely rendkívüli részletességgel vezet végig a különböző lehetőségeken, amely nyáktervezéskor rengeteg segítséget nyújtott.[8]
A modemhez kapcsolódó alkatrész, a mobilkommunikációs kapcsolat létrehozásához elengedhetetlen SIM kártya. Ahhoz, hogy a modem regisztrálni tudjon a hálózatra, min-

denképpen szükség van egy fizikai vagy e-SIM-re. A munkahelyi körülmények lehetővé tették a fizikai SIM kártya használatát, így felesleges lett volna költségvetési és programozási erőforrásokat költeni egy e-SIM beszerzésére és implementálására. Ezután csupán egy áramkörre integrálható SIM kártya tartót/olvasót volt szükséges beszerezni, ugyanakkor az áramkör tervezésekor tudatosult csak, hogy egy ilyen alkatrész mennyire sok helyet tud elfoglalni a nyákon. Amennyiben több idő jutott volna a projekt kivitelezésére, mindenképpen az e-SIM lett volna az optimálisabb választás a helytakarékosság szempontjából.

2.2.7 Öntözés

Mivel a magaságyás által felhasznált földterület nem kimondottan nagy, kisebb méretű berendezés használata is elegendő az öntözéshez. Ez természetesen addig igaz, amíg a magaságyás csak egy szinttel rendelkezik. Amennyiben több szintet szeretnénk egymásra építeni két alternatíva lehetséges. Az egyik, hogy nagyobb teljesítményű motort alkalmazunk a növények öntözésére, vagy lehetséges több kisebb motort is használni. Amennyiben a motort meghajtó IC[9] még képes elegendő teljesítményt leadni, az előbbi egy jobb választás lehet, mivel nem jár a kapcsolási rajz módosításával.

A jelenlegi helyzetben elegendő egy kisebb teljesítményű vízpumpa, amit akár egy hobbi akvarista szaküzletben is be lehet szerezni. A projektben felhasznált motor az AD20P-1230A, amely 240L/h kiömlési sebességet képes előállítani akár három méter magasban is, mindezt négy wattos fogyasztással.
Az interneten kaphatóak olyan állítható öntözőfejek, amelyek egy T elágazást képezve csatlakoznak az öntözőcsövekhez. Az öntözőfejen található szelep állításával könnyedén lehet variálni a rajta kiömlő víz mennyiségét, illetve annak formáját, ezáltal lehetőség van nagy területet lefedő, vagy koncentrált vízsugár előállítására is.

2.2.8 Léptetőmotorok

A napelem mozgatására használt léptetőmotorok optimalizált vezérlésének érdekében egy erre a célra kifejlesztett integrált áramkör került felhasználásra[10]. Bár ez a projekt költ-

ségvetésének megnövekedését jelentette, sok programozásba fektetett órát spórol meg a firmware fejlesztése során, ami egyúttal költségmegtakarításként is felfogható. Az áramkör használatával lehetőség adódik a motor – megfelelő lábainak feszültségszintbe állítása mellett – PWM-el való vezérlésére, egyetlen bemenet felhasználásával. Az IC különböző algoritmusok alkalmazásával képes optimalizálni a motor indulási és leállási körülményeit, ezáltal kímélve azt az idő előtti elavulástól.
A különböző léptetési módok között három láb beállításával lehet választani. Mivel a napelemek mozgatása nem kíván nagy felbontású aprólékos léptetést, a vezetékezés úgy lett megtervezve, hogy az MCU egyik kimenetével lehessen váltani teljes és fél-lépéses üzemmódok között. Az áramkör "SLEEP" bemenetének vezérlésével alvó módba lehet állítani az áramkört, amely jelentősen lecsökkenti az áramfelhasználást. A mikrokontroller és a vezérlőáramkör közötti összeköttetéseket a 2.1 táblázat szemlélteti az MCU szemszögéből tekintve.

Funkció	Típus
motor léptetése	PWM kimenet
léptetés irányának kiválasztása	általános kimenet
alvó üzemmód	általános kimenet
áramkör alaphelyzetbe állítása (reset)	általános kimenet
léptetési mód	általános kimenet

2.1. táblázat: Az MCU és léptetőmotor vezérlőáramkör közötti összeköttetések

A táblázat alapján egyetlen vezérlő használatához öt darab kimenet szükséges a mikrokontroller részéről, ami két vezérlő esetén tízet jelent. A 48 lábas kiosztásnak, illetve annak köszönhetően, hogy a többi ellátni kívánt funkció nem igényelte sok GPIO felhasználását, ez könnyedén megvalósítható volt.

A vezérlőáramkör egyik fontos és hasznos tulajdonsága, hogy áramkorlátozást is be lehet állítani referenciafeszültség segítségével. A kiválasztott léptetőmotorok katalógusadatai szerint egy fázis 400mA áramot fogyaszt meghajtáskor. Általánosságban egy léptetőmotor tekercseinek vezérlésekor egyszerre kettő kerül meghajtásra, így kerekítéssel

ezt az értéket vehetjük 1 ampernek. Ekkor abban az –egyébként nem valószínű – esetben, amikor négy fázis kerül meghajtásra közel egy időben, a fellépő áramerősség 2A körüli. Az áramkorlátozás beállítása a következő képlet szerint alakul.

$$I_{MAX} = \frac{U_{REF}}{(8 * R_S)} = 2A$$

ahol $R_S = 0.1\Omega$, az interneten talált fejlesztő modulokon lévő ellenállásértékek alapján[11]. Az egyenletet átrendezve megkapjuk, hogy 2,5 amperes áramkorlát esetén $U_{REF} = 1.6V$. Ez a referenciafeszültség az alkatrész logikai tápellátásáról egy feszültségosztóval lett beállítva.

2.3 Teljesítményfelvétel

A felhasznált elemek ismeretében meghatározható az egyes rendszerek maximális áramfelvétele. Ennek alapján került megtervezésre a túláram ellen védelmet szolgáltató áramkör, valamint a rendszer tápfeszültségének megválasztása. A 2.2 táblázat mutatja a rendszer nagyobb részét képző elemeinek várható maximális áramfogyasztását. Fontos megjegyezni, hogy a rendszer olyan feltétel kiszabása mellett került megtervezésre, hogy a nagy áramfelvételű fő komponensek – mint a modem vagy a motorok – közül egy időben csak egy működhet, ezzel minimalizálva a pillanatnyi teljesítményt.

Áramköri elem	Maximális áram [A]
Modem	2,5
Léptetőmotor	1
Vízpumpa	0,5
Mikrokontroller	0,14

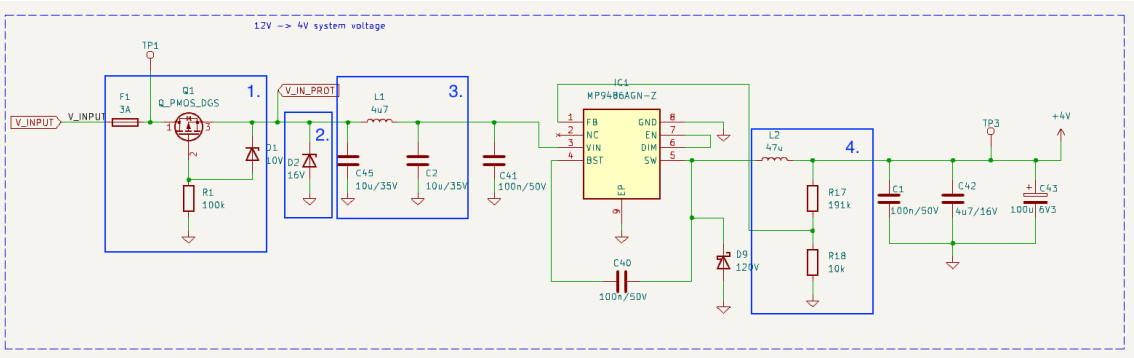
2.2. táblázat: A fő komponensek maximális áramfelvétele

A felsorolt áramkörök közül a legnagyobb pillanatnyi teljesítménnyel a mobilkommunikációs modem rendelkezik. Katalógusadatok alapján[7] a maximális áramfelvétel 2G hálózat használatakor léphet fel. Annak ellenére, hogy a kivitelezés során ez a hálózati protokoll nem kerül felhasználásra, a rendszer úgy lett méretezve, hogy a későbbiekben

még is lehetséges legyen használni. Mivel az ilyenkor fellépő 2,5 amperes áramerősség a rendszer egészét tekintve a legnagyobb várható érték, a túláram elleni védelem egy 3 amperes biztosítékkal lett megoldva.

2.4 Nyákterv

A kapcsolás megrajzolása a KiCad 7.0.6-0 verziójával történt. A rajzok elkészítéséhez az adatlapokban leírt ajánlások rengeteg segítséget nyújtottak.[12]
Az első lépés a tápellátás megtervezése volt, amely két részre bomlott. Első lépésben egy kapcsoló üzemű buck converter[13] a tizenkét voltos bemeneti feszültséget öt voltra konvertálja. Ez a kapcsolás tartalmazza az áramkört védő és stabilizáló elemeket is.

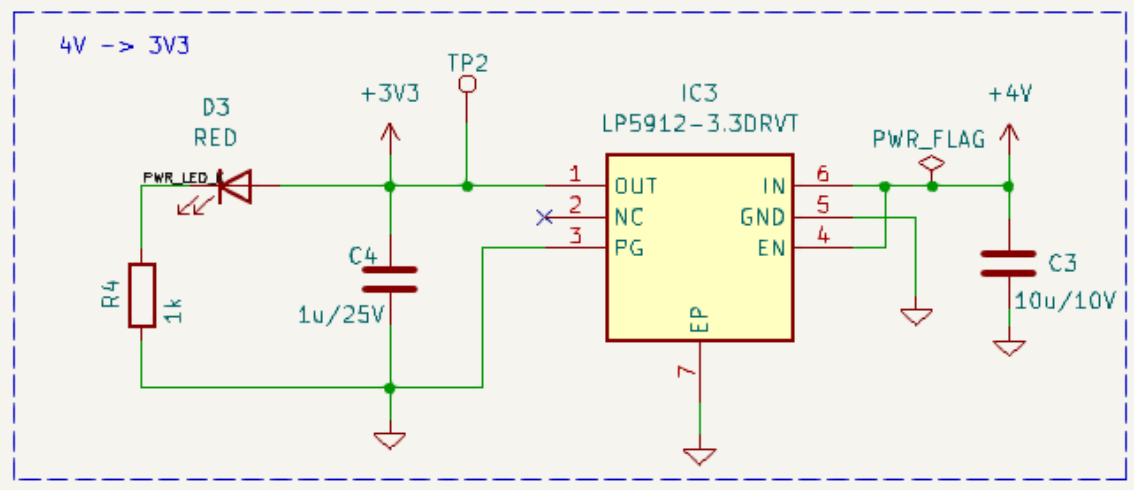


2.2. ábra: A tápellátást szolgáltató áramkör és elemei

A 2.2 ábrán látható a tápellátást szolgáltató áramkör első szintje. Az 1. számmal jelzett bekeretezett részben látható a három amperes biztosíték, valamint a negatív polaritás elleni védelem egy P csatornás MOSFET felhasználásával. A 2. a túlfeszültség ellen védelmet nyújtó dióda. Az ezután következő rész a védett tizenkét voltos tápfeszültség. A motorok is innen kapják a tápfeszültséget a vezérlőáramkörökön keresztül. A 3. egy PI szűrő, amely az esetlegesen fellépő zavarjeleket hivatott szűrni a feszültségátalakító bemenetéről. A 4. bekeretezett rész tartalmazza a kimeneten lévő folytótekercset, illetve a kimeneti feszültségszint beállítására szolgáló feszültségosztót, amelynek számítása az alábbi képlet alapján történik.

$$R_1 = R_2 * \frac{V_{OUT} - V_{FB}}{V_{FB}}$$

ahol $V_{FB} = 0.2V$ katalógusadatok szerint[12]. A rajz és a feliratozások alapján megfigyelhető, hogy az előzőekben leírtakkal szemben a kimeneti feszültség négy voltra lett beállítva. Ennek az oka, hogy kezdetben a mobilkommunikációs modem tápellátása 3.3 voltnak lett megválasztva, azonban az adatlap szerint a megfelelő hálózati kommunikációhoz a minimum feszültségnek minden pillanatban legalább 3.1 voltnak kell lennie. A modem azonban időnként kimondottan nagy áramfelvétel mellett üzemel, ami feszültségesést eredményezhet. Ez kiküszöbölhető megfelelő szűrőáramkörökkel, ugyanakkor egy bizonytalan tényezőt jelent a komponens működését tekintve, ami a költségek és a szűkös határidőt figyelembe véve nem engedhető meg. Az öt voltos feszültség az áramkör többi részén nem került felhasználásra, innen jött az ötlet, hogy öt volt helyett inkább négy volt kerül előállításra, és ez szolgáltatja a tápfeszültséget a modem számára.



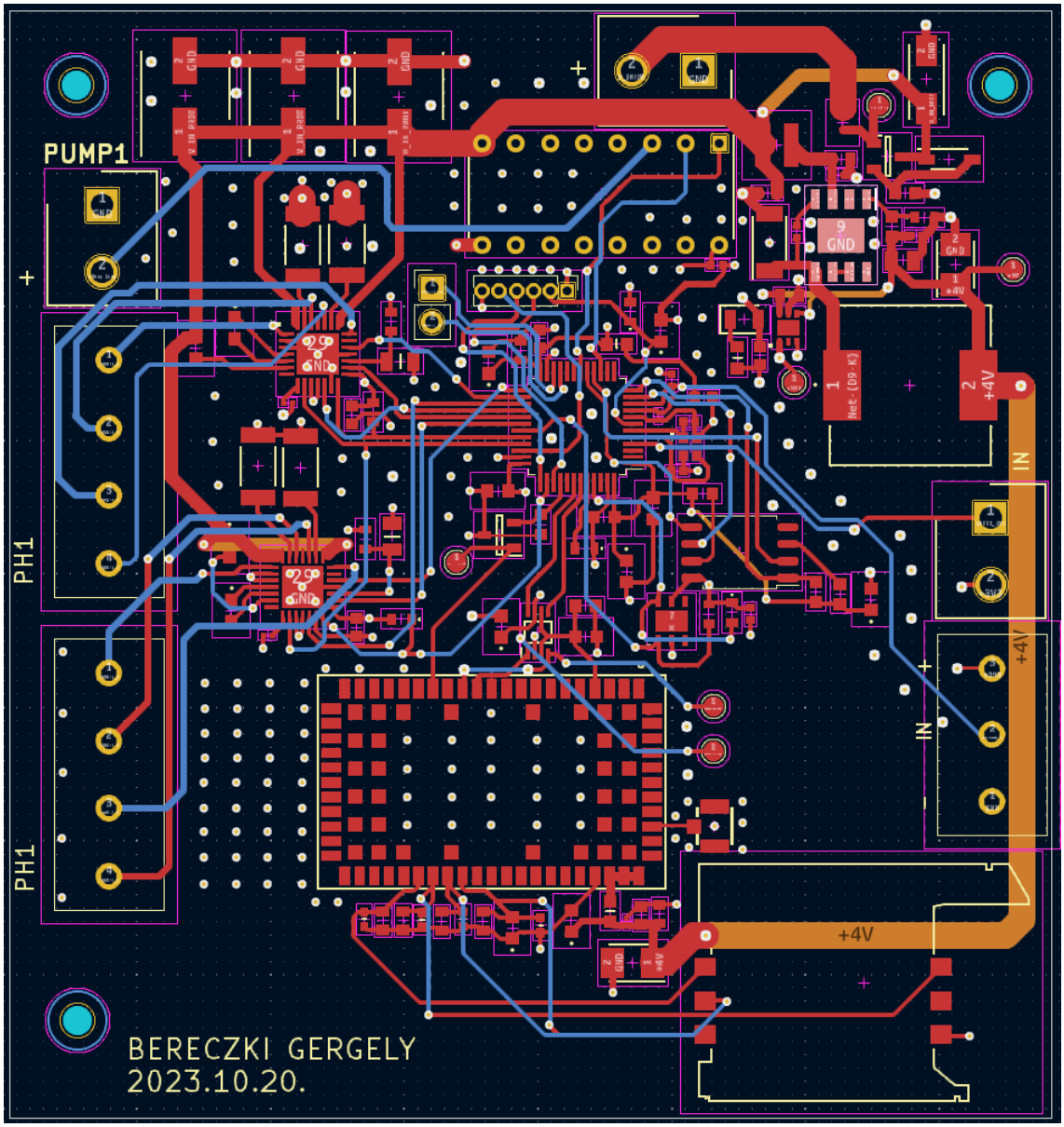
2.3. ábra: A logikai tápellátást szolgáltató áramkör

A 2.3 ábrán látható a 3.3 voltos logikai tápfeszültséget előállító áramkör.[14] Azon felül, hogy ez a komponens is rendelkezik szűrőkondenzátorokkal a ki és bemenetén, egy LED ad visszajelzést a megfelelő tápfeszültség meglétéről.

Az összes kapcsolás közül a tápellátás bizonyult a legnehezebbnek. A többi áramkör könnyen megrajzolható volt a gyártók által szolgáltatott referenciakapcsolások alapján. Kihívást jelentett ugyanakkor a mikrokontroller megfelelő vezetékezése. Nagy segítséget nyújtott az STM32CubeIde fejlesztőkörnyezetbe integrált hardver konfigurációs modul.

Miután a követelményeknek megfelelően az összes periféria beállításra került a programban, egyszerűen csak le kellett másolni azt a rajzba, a megfelelő lábak felcímkézésével.
A kapcsolás elkészítése után, ellenőrzés során ötlött csak szembe egy igen kellemetlen hiba, amire az ember nem feltétlenül számítana. A korábbiakban már leírásra került, hogy a modem és az MCU közötti kommunikáció soros porton keresztül történik, azonban ehhez egy feszültségszint konvertáló áramkör szükséges, ugyanis a modem 1.8 voltot használ. Ennek a konvertáló áramkörnek a tervezésekor először nem tűnt fel, hogy az uBlox a mikrokontroller szemszögéből címkézte fel az RX/TX portokat. Ez azt jelenti, hogy TX-TX, RX-RX összeköttetéseket kell létrehozni, ami nem megszokott. Amennyiben az áramkör felcserélt portokkal került volna legyártásra, ellehetetlenítette volna a modem és az MCU közötti kommunikációt.

A kapcsolási rajz elkészültével, a nyákterv megkezdése előtt mindenképpen szükséges volt kiválasztani, hogy melyik gyártótól lesz az megrendelve. Ez azért volt fontos, mert nem minden gyártó rendelkezik ugyanolyan technológiákkal, így például a minimum via és vezetősávok mérete eltérhet. Hosszabb keresés után a választás a JLCPCB néven ismert gyártóra esett. Kétrétegű nyákokat 100x100 milliméteres kivitelezésben 2€ értékben is van lehetőség rendelni. Amennyiben a nyák mérete nem haladja meg az 50x50 millimétert, a négy rétegű áramkörök legyártása nem jár extra költségekkel.
A fentebb felsoroltak fényében a tervezett nyák 4 rétegű kivitelezésben került megtervezésre. A felső és alsó, valamint az egyik közbenső réteg a nulla referenciaszinttel vannak kitöltve. A két felszíni rétegen haladnak a különböző jeleket vezető sávok, míg a negyedik, közbenső réteg a logikai tápfeszültséggel van kitöltve.
Miután a gyártó által megadott adatok alapján a megfelelő gyártási korlátozások beállítása készen volt, először a főbb elemek kerültek elrendezésre, például a tápellátást szolgáltató áramkörök. Minden esetben az első lépés a szűrőelemek elhelyezése volt a fő komponens körül, utána az áramkör többi része. Az elkészült nyáktervet a 2.4 ábra mutatja.



2.4. ábra: Az elkészült nyákterv

A nagyobb áramfelvételű vezetősávok szélessége maximalizálva lett, azonban ennek gyakran gátat szabtak a komponensek csatlakozási pontjai. Sajnálatos módon a nyáklap méretének csökkentésére tett kísérletek ellenére a végtermék dimenziói meghaladták az 50x50 millimétert, így a 2€ helyett kb. 6€-ba került az öt darab nyáklap. A nyáklapok mellé sablont is mellékeltek, így az egész csomag ára kb. 30€ volt szállítással együtt.

A tervezés után levont következtetések alapján a legnagyobb tanulság az volt, hogy a komponensek kiválasztásakor nagyon fontos azoknak méreteit ellenőrizni. Sajnos három

olyan kondenzátor is belekerült a tervbe, amit a valóságban szabad szemmel szinte nem is látni.

3 SZOFTVER

3.1 Adatok és adatformátumok

A vezérlőegység és szerver közötti kommunikáció az egyes konfigurációk letöltését, valamint a mikrokontroller által generált üzenetek szerver felé továbbítását foglalja magába. Mindkettő esetben célszerű az adatok strukturált egységbe való szervezése a könnyebb kezelhetőség és áttekinthetőség érdekében. Ennek alapján az üzenetek és konfigurációk adatformátumaként a JSON standard került kiválasztásra. Ez a fájlformátum nagy népszerűségnek örvend minden felhasználási területen egyszerű, ugyanakkor könnyen olvasható és kezelhető felépítésének köszönhetően. Szinte minden programozási nyelven található hozzá olyan könyvtár, amely megkönnyíti az adatstruktúrák kezelését.

3.2 Firmware

Ebben az alfejezetben az egyénileg tervezett áramkörben lévő mikrokontrolleren futó program leírására kerül sor. A firmware megírására kiválasztott nyelv a C és C++ együttes kombinációja.

A dokumentáció készítésekor az egyénileg tervezett áramkör még nem állt rendelkezésre, így a fejezetben tárgyalásra kerülő koncepciók csupán tervezési szinten kerülnek leírásra, konkrét példák és implementációk nem állnak rendelkezésre.

3.2.1 Valós idejű operációs rendszer

A tervezett firmware számára kiszabott feladatok ciklikusan végrehajthatóak, így alapvetően nem feltétlenül lenne szükség valós idejű operációs rendszer (RTOS) használatára. A projekt komplexitását figyelembe véve azonban egy RTOS számos olyan funkcionális elemet nyújthat, amelyek merőben megkönnyíthetik a fejlesztést.

Az *mbedos* nevű operációs rendszer egy C++ nyelven leprogramozott keretrendszert nyújt a felhasználónak, amely az iparban megtalálható gyártók mikrovezérlőinek szé-

les skáláját támogatja. Nagy mértékben hasonlít az Arduino keretrendszerhez, azonban egy annál sokkal kifinomultabb eszköztárat bocsájt a felhasználó számára. Az mbedos könnyen használható drivereket tartalmaz külső SPI flash memória és a projekt során alkalmazott Sara uBlox modemhez is, ami alapján egyértelmű választás lett volna. Sajnálatos módon a támogatott mikrokontrollerek listáját böngészve a projektben felhasznált STM32L431CC6TR nem volt megtalálható. Egy hasonló kialakítású vezérlő, az STM32L433RC mind lábkiosztásban mind memóriát tekintve megegyezik és támogatott is, azonban az internetet böngészve hamar kiderült, hogy nagyon komplikált és megbízhatatlan implementációt eredményezhet.
Mindezek fényében a választás a *FreeRTOS*-ra került. A nyílt forráskódú operációs rendszer szintén széles körben felhasznált, azonban eszköztára nem annyira kiterjedt mint az mbedos által kínált. Mindezek ellenére enyhítő körülményként szolgált a tény, hogy ez a keretrendszer szintén rendelkezik a modemhez előre megírt driverrel.

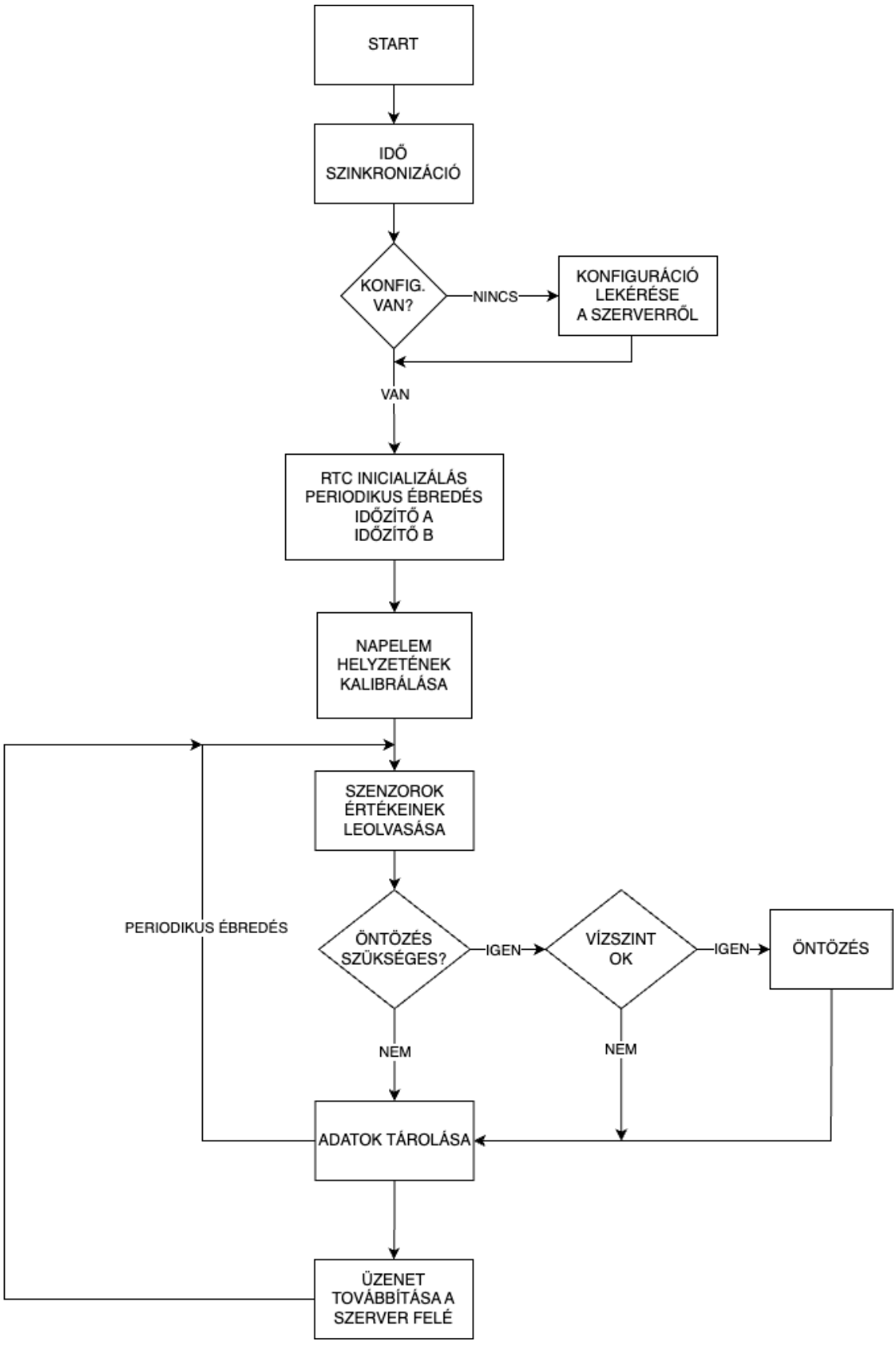
3.2.2 Fejlesztői környezet

A fejlesztői környezet megválasztása azon alapult, hogy milyen valósidejű operációs rendszer kerül megválasztásra, ugyanis annak támogatottsága rendszerről rendszerre eltérhet. A FreeRTOS esetében legkézenfekvőbb az STM32CubeIDE, amely beépített modulként képes használni ezt az operációs rendszert. Ez a modul egy úgynevezett *CMSIS wrappert* használ, amely egy absztrakt réteget képez az alap FreeRTOS keretrendszer felett, ezáltal lehetőséget adva egy másik operációs rendszerre való áttérésre. A környezet grafikus felületet kínál órajelek és perifériák konfigurálására, amelyhez aztán programkódot is generál. A kódgenerálás után részletes információt kapunk az egyes memóriaelemek (FLASH, RAM) állapotáról, amely hasznos információként szolgálhat memóriagazdálkodás tekintetében.
Amennyiben a céláramkör rendelkezik a megfelelő interfésszel, lehetőség van a programkód debugolására is.

3.2.3 Logikai rendszerterv

A firmware egyik legfontosabb alapját a szerverről letöltött konfiguráció jelenti, ezért szükséges, hogy ezt hatékonyan legyen képes feldolgozni és eltárolni. A konfigurációk tárolása mindenképpen egy nem-felejtő memóriában kell, hogy történjen. A céláramkörön futó program periodikus és előre meghatározott időpillanatokban is hajt végre feladatokat. Annak érdekében, hogy az időzítés megfelelő legyen, minden esetben szükséges a pontos idő tárolása, ezért a programnak rendelkeznie kell egy időszinkronizációs folyamattal. A pontos idő és konfiguráció birtokában, a firmware beállítja a felébredés periódusidejét, illetve az öntözés időpontjai alapján meghatározott időzítőket, amelyből összesen kettő lehet.

A periodikus ébredés alkalmával a program rögzíti a szenzorok által mért adatokat, valamint kalibrálja a napelem helyzetét, amennyiben az szükséges. Ezt az aktuális és előzőleg mért fénymennyiség értékek közötti differencia határozza meg. A mért szenzorértékek egy külön adattípusban tárolódnak, amelyet aztán a megfelelő szoftverkomponens dolgoz fel és illeszt be az egyes üzenetekbe. A mért adatok mellett minden esetben az aktuális időpont rögzítése is szükséges. A generált adatokat a program összefűzi, majd a szerver felé továbbítja. A modulüzenetek küldése minden esetben óránként történik. A program futásának magas szintű vázlata a 3.1 ábrán látható.



3.1. ábra: A firmware működésének folyamatábrája

Az egyes feladatok kiszolgálását biztosító folyamatok külön taszkokba rendezve kerülnek futtatásra. Az eseménysorozatok időzítése és a taszkok közötti szinkronizáció és kommunikáció a valósidejű operációs rendszer által nyújtott funkciók segítségével történik. Ilyen elemek például a Queue, amelyek FIFO jellegű tárolóelemek, vagy az eseményeket bitenként jelző WaitFlag-ek.

A firmwarenek rendelkeznie kell olyan naplózási funkcióval, amely folyamatosan információt szolgáltat az éppen lezajló folyamatokról és műveletekről, amely soros porton keresztül olvasható.

3.2.4 Fizikai rendszerterv

A szoftverelemek meghatározása előtt kitűzésre kerültek a rendszer által elvégzendő fő feladatok, amelyek a következőek:

- időszinkronizáció
- öntözés
- konfiguráció menedzselése
- szenzorértékek leolvasása
- adatmenedzsment
- mobil kommunikáció

Ezeknek a feladatoknak a végrehajtásáról erre a célra programozott osztály gondoskodik minden esetben. Az osztályok létrehozása a *singleton* elemeken alapuló programtervezési minta[15] alapján történik. A singleton elemek lényege, hogy a kód lefutása során egyetlen példány létezhet belőlük, amelyek statikus módon tárolódnak a memóriában. Egy taszk az így létrehozott entitások felhasználásával hajtja végre feladatait.

A rendszer meghatározó részét képezi a konfigurációs adatokat tároló *SystemConfiguration* osztály. Ez az entitás tartalmaz minden olyan beállítást, amelyek a programkód futását befolyásolhatják. Az egyes szoftverkomponensek innen képesek kinyerni a számukra szükséges információkat.

A különböző lezajló folyamatok során az adatok manipulációjáért külön entitás felelős. A *DataManager* osztály egy közbenső szereplőként van jelen az egyes szoftverkomponensek közötti adatmenedzselés céljából. A program lefutása során számos alkalommal lesz szükség például JSON objektumok elemzésére. A DataManager az adatok feldolgozása után a megfelelő elemek rendelkezésére bocsájtja azokat. Annak érdekében, hogy az egyes entitások ne közvetlenül használják a DataManager erőforrásait, a megfelelő

adattípusok queue-ba való küldésével tudnak adatfeldolgozást kezdeményezni. Az adatok tárolásának érdekében a modul fontos részét képezi a külső FLASH memóriát kezelő driver elem. Az ismertetett osztály által generált üzenetek az alábbi JSON objektumhoz hasonlóan néznek majd ki.

```
{
    "ts":1700212080,
    "cfid":4,
    "tmp": 2350,
    "ahmd": 45,
    "shmd": 75,
    "irg": [1700914778,1700914838],
    "wlvl": 1
}
```

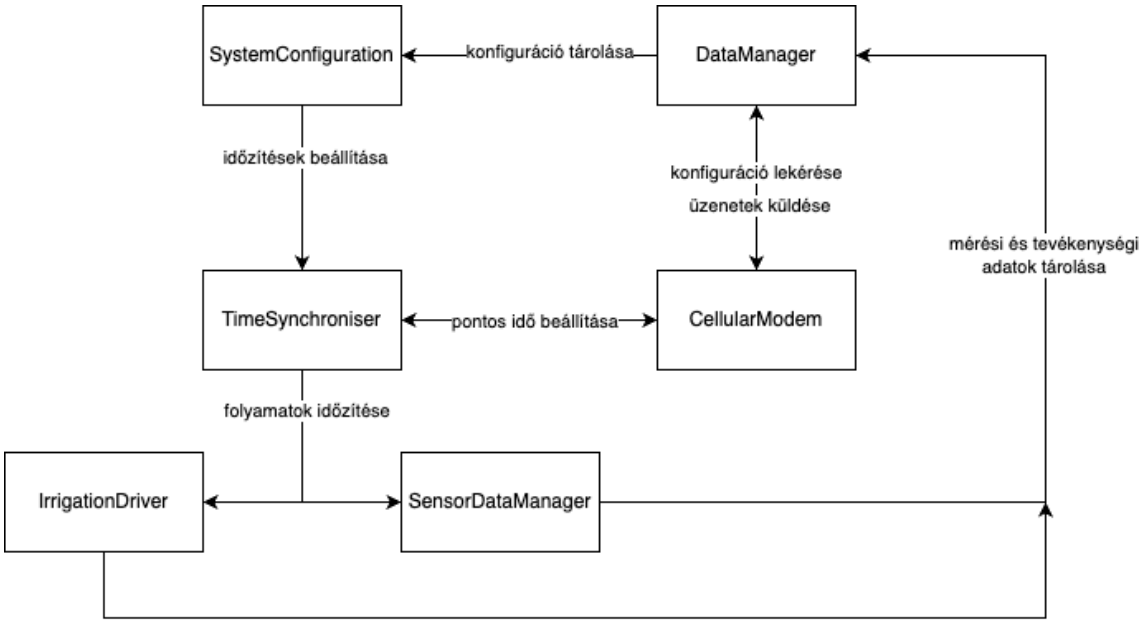
Az internetre való csatlakozásért, illetve az internettel kapcsolatos feladatok elvégzéséért a *CellularModem* osztály felelős. A modem alacsony szintű implementációját a valós-idejű operációs rendszer szolgáltatja, így csak az projekthez kötődő feladatok leprogramozása szükséges.

Az időszinkronizáció a mikrokontrollerbe integrált RTC modul és internetkapcsolat segítségével lehetséges. A weben számos olyan programozási interfész megtalálható, amelyen keresztül egy GET request segítségével megkaphatjuk a pontos időt. A projektben felhasznált API a *worldtimeapi.org*.
A hívás végpontja http://worldtimeapi.org/api/timezone/Europe/Budapest. Az implementáció nem terjed ki a pontos geo-lokáció meghatározására, így az a meghívott végpontban statikus módon szerepel.

A szenzorok értékeinek rögzítéséért felelős szoftverkomponens tartalmaz minden olyan szoftverelemet, amely az érzékelők leolvasásához szükséges. Egy periodikusan futtatott taszk az értékeket egy specifikus adattípusban queue-n keresztül továbbítja az adatmenedzselésért felelős osztály felé feldolgozásra.

Az öntözés az időzítők jelzése alapján veszi kezdetét. A szükséges időtartam és az öntözés időpontja a konfigurációt menedzselő modultól kérdezhető le. Az öntözést végző driver PWM jel előállítása után a megfelelő láb vezérlésével képes elindítani a folyamatot. A program az RTC modul segítségével rögzíti az öntözés kezdeti és végső időpontját, ezáltal egyfajta visszajelzést adva arról, hogy az tényleg végbe ment.

Az előzőekben felsorolt szoftverkomponensek közötti relációk könnyebb szemléltetését a 3.2 ábra segíti.



3.2. ábra: A firmware szoftverelemeinek relációja

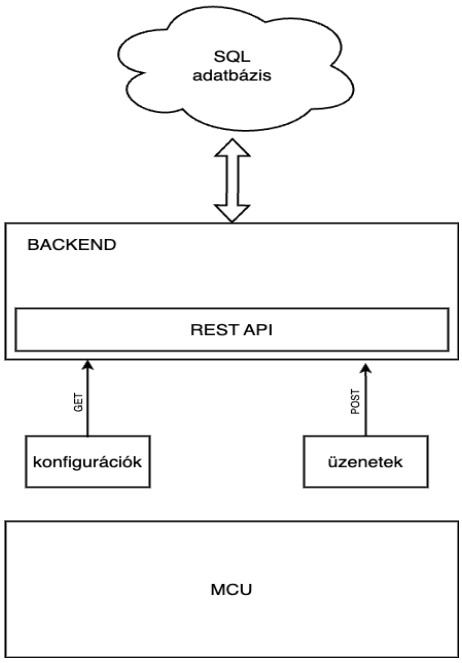
3.3 Szerver

Az angol szakirodalomban "backend"-ként nevezett szerver a felelős az adatok háttérben történő feldolgozásáért. Ennek implementációja fogja kiszolgálni a mikrokontrollert interneten keresztül, az adatbázisában található adatok alapján.

3.3.1 Logikai rendszerterv

A tervezés megkezdésekor a szervert képző entitásnak a feladata konfigurációk és modulüzenetek tárolása, rendelkezésre állítása volt. Mindkét esetben az adatok sorosítására a

JSON struktúra kerül alkalmazásra. A feladatokhoz tartozó adatokat SQL adatbázis tárolja két külön táblában. A szerver hosztolása olcsó, sokoldalú funkcionalitásokat nyújtó szolgáltató által kerül sor.



3.3. ábra: A szerver logikai rendszertervének magas szintű ábrája

A 3.3 ábra szemlélteti a rendszer magas szintű vázlatát. A feltüntetett elemek mellett helyet kap még egy grafikus felület, ahol a felhasználó személyre szabott konfigurációkat hozhat létre, amelyet később a modul fog használni.
A konfigurációk és üzenetek küldése, valamint fogadása REST programozási interfészen keresztül történik. Ezzel a protokollal viszonylag egyszerűen implementálható a szerver és a kérő közötti adatcsere HTTP kérésekkel. A kérő fél a megfelelő URL linken képes adatot lekérni, vagy szolgáltatni a szerver számára, amelyre a szerver egy státuszkóddal, illetve adott esetben egy JSON objektummal válaszol.

Az üzenetek szerver felé történő továbbítása JSON formátumban történik egy POST request által. A szerverrel való autentikáció token alapú, amelyet a request fejlécében kell beállítani, máskülönben a kérés visszautasításra kerül. Amikor a szerver egy kérést kap, mindenképpen ellenőriznie kell az objektum integritását mielőtt beillesztené az adatbázisba.

A konfigurációk adatbázisban történő tárolása során a legújabb konfiguráció a fő kulcsként használt "id" oszlop alapján határozható meg. Minden konfigurációs objektumnak és üzenetnek tartalmaznia kell egy "cfid" mezőt, amely alapján meghatározható, hogy pontosan melyiket használja a modul. A legfrissebb konfiguráció lekérdezéséhez egy olyan végpont került kialakításra, amely bemenő paraméterként veszi a konfiguráció azonosítóját, majd amennyiben van újabb, – tehát van olyan azonosító amelyik nagyobb számú – a szerver válaszként egy JSON objektumként küldi azt vissza. Amennyiben a modulon a legújabb konfiguráció található meg, a szerver egy üres választ küld vissza 200-as HTTP kóddal.

3.3.2 Fizikai rendszerterv

A szerver fizikai rendszerterve áttekintést ad az implementációhoz kiválasztott eszközökről illetve az implementáció módjáról. A webfejlesztés megkezdése előtt fontos szempont volt egy olyan keretrendszer kiválasztása amely lehetőleg egy intuitív könnyen kezelhető felületet ad mind a háttérben, mind a felhasználó szeme előtt lejátszódó folyamatok programozására. Ezeknek a feltételeknek eleget tesz a Django, Python nyelven programozható keretrendszere. Segítségével csak minimális HTML programozás szükséges, amennyiben nem szükséges igényesebb kinézetű weboldalak szerkesztésére. A jelen implementációban az esztétikai tulajdonságok nem képeztek szempontot.

3.3.3 Django keretrendszer

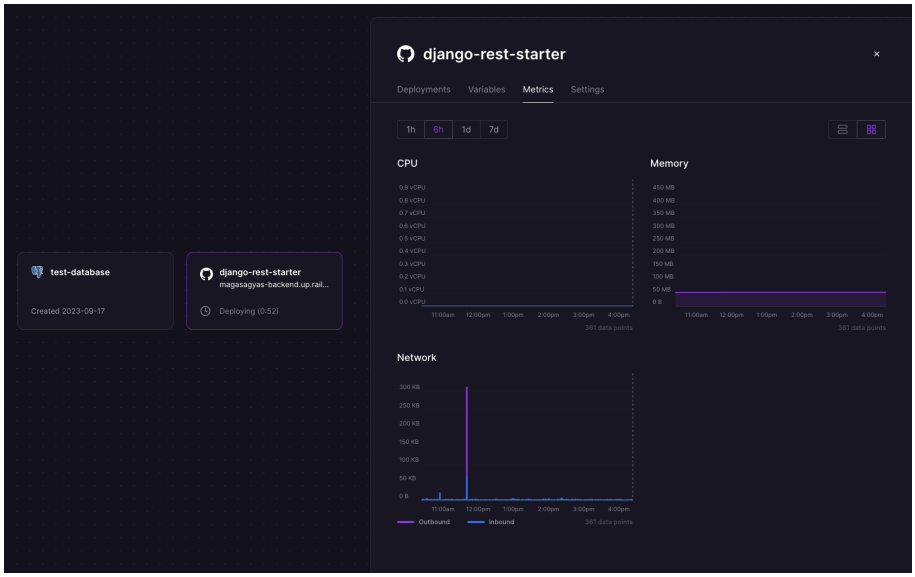
A Django egy rendkívül jól kezelhető, fejlesztői eszközökkel széleskörűen ellátott keretrendszer, amely képes absztrakt réteget képezni a Python programozási nyelv valamint az adatbázis kelezéskor használt SQL parancsok között, úgynevezett modellek segítségével.[16] Ez tulajdonképpen azt jelenti, hogy az adatbázisban szereplő táblákat egy-egy osztály reprezentálja. Az adatok manipulációja Python függvényeken keresztül történik, amit aztán a keretrendszer SQL utasításokká alakít.[17] A Django lehetővé teszi az admin felület használatát is. Amennyiben egy felhasználó megfelelő jogokkal rendelkezik, bejelentkezve erre a felületre manuális módosításokat hajthat végre az adatbázisokon, il-

letve hozzáadhat más felhasználókat is az admin csoporthoz. Természetesen ez a felület is teljes mértékben személyre szabható ezáltal lehetővé téve az alkalmazás teljes mértékű "kézi" manipulációját.

A Django keretrendszerben létrehozhatóak úgynevezett "form"-ok, amelyek adatbevitelre használhatóak, ezek a felhasználó által is látható elemek. A segítségével bekért adatokkal gazdálkodva újabb beszúrásokat készíthetünk az adatbázisba. Az így létrehozott objektumokat HTML-ként kell reprezentálni a felhasználó felé. Ez egyszerűen megoldható egy HTML sablon segítségével, ahova csak néhány sort kell beilleszteni, a többit a keretrendszer elintézi.

3.3.4 Hoszt

A backend hosztolását a Railway nevű szolgáltató végzi, amelynek főoldala a *railway.app* linken érhető el. Az alap csomag havi 5€ fix díjjal rendelkezik. Ebbe az összegbe X mennyiségű processzor és memóriahasználat tartozik. Amennyiben az alkalmazás ennél többet fogyaszt, az arányosan kiszámlázásra kerül. Az oldal kiválasztásakor előnyös tulajdonságnak bizonyult, hogy rengeteg úgynevezett "template" áll a felhasználók rendelkezésére. Ezek által a projekt alapjai egy gombnyomással elő lettek készítve, úgy mint GitHub projekt Django keretrendszerrel REST API alkalmazásra konfigurálva, és egy PostgreSQL adatbázis a szükséges táblákkal és beállításokkal[18]. Az oldal tulajdonképpen ezt a GitHub projektet hosztolja, így minden változás azonnal megjelenik az élő környezetben is.



3.4. ábra: A Railway projekt és a projekt által felhasznált erőforrások

A 3.4 ábrán látható a hoszt által nyújtott kezelői felület egy része. A GitHub projekt és adatbázis két külön elemként kezelhetőek. A jobb oldalon a projekt által felhasznált erőforrások látszódnak.

3.3.5 Konfigurációk

A JSON formátumban tárolt adatoknak köszönhetően a konfigurációs elemek bővítése vagy szűkítése könnyűszerrel megtehető, a fontos, hogy a változások mindig szinkronban legyenek a konfigurációt készítő és az azt megkapó eszközök között. A projekt megkezdésekor fix konfigurációs elemek kerültek meghatározásra, amelyeket a 3.3 táblázatban felsorolt elemek mutatnak.

Megnevezés	Mértékegység/Formátum
locsolás 1	"óó:pp"
locsolás 2	"óó:pp"
locsolás időtartama 1	másodperc
locsolás időtartama 2	másodperc
szenzorok beolvasásának frekvenciája	másodperc

3.3. táblázat: Konfigurációs elemek

A konfigurációk az adatbázis "configs" táblájában tárolódnak, amelynek felépítését a 3.4 táblázat mutatja.

Név	Típus	Alapbeállítás
id	integer	automatikusan inkrementált
config	json	-
created_at	időbélyeg időzóna nélkül	automatikus

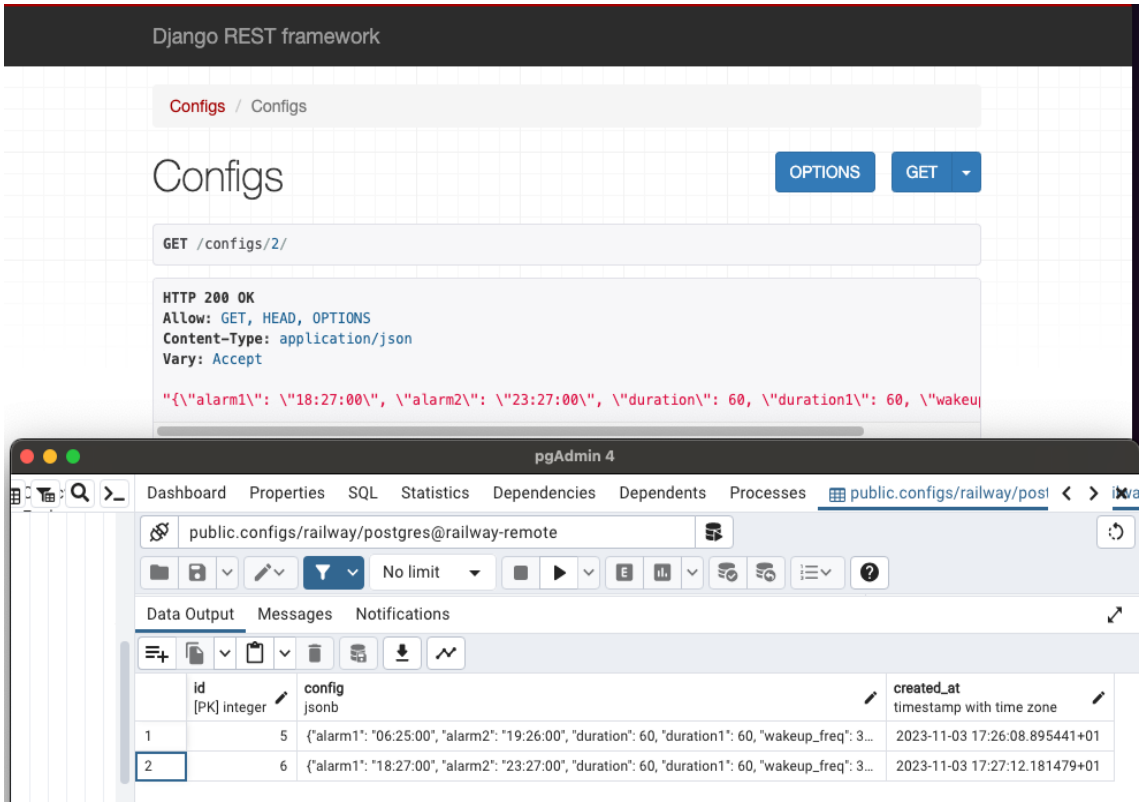
3.4. táblázat: Az adatbázis configs táblájának oszlopai

A logikai rendszertervben leírtak alapján új konfiguráció meglétének lekérése GET request segítségével történik. A Django keretrendszer az egyes URL végpontokat úgynevezett "View"-ként implementálja, jelen esetben az APIView amely egy változata a View osztálynak REST API felhasználásra módosítva. A GET request során a "get" függvény kerül futtatásra.

```
class ConfigsView(APIView):
    def get(self, request, id):
        try:
            latest_config = Config.objects.latest("id")
            if latest_config.id > id:
                return Response(latest_config.config)
            elif latest_config.id == id:
```

```
            return Response(status=200)
        else:
            return Response(status=404)
    except Config.DoesNotExist:
        return Response(status=404)
```

A `latest_config = Config.objects.latest("id")` kód a táblába utoljára beszúrt sor azonosítóját adja vissza. Amennyiben ez az azonosító nagyobb mint ami a kérésben szerepel, a szerver a válaszban küldi vissza a konfigurációt. Erre az esetre mutat példát a 3.5 ábra, ahol bemenő paraméterként a 2-es azonosítójú konfiguráció került megadásra.

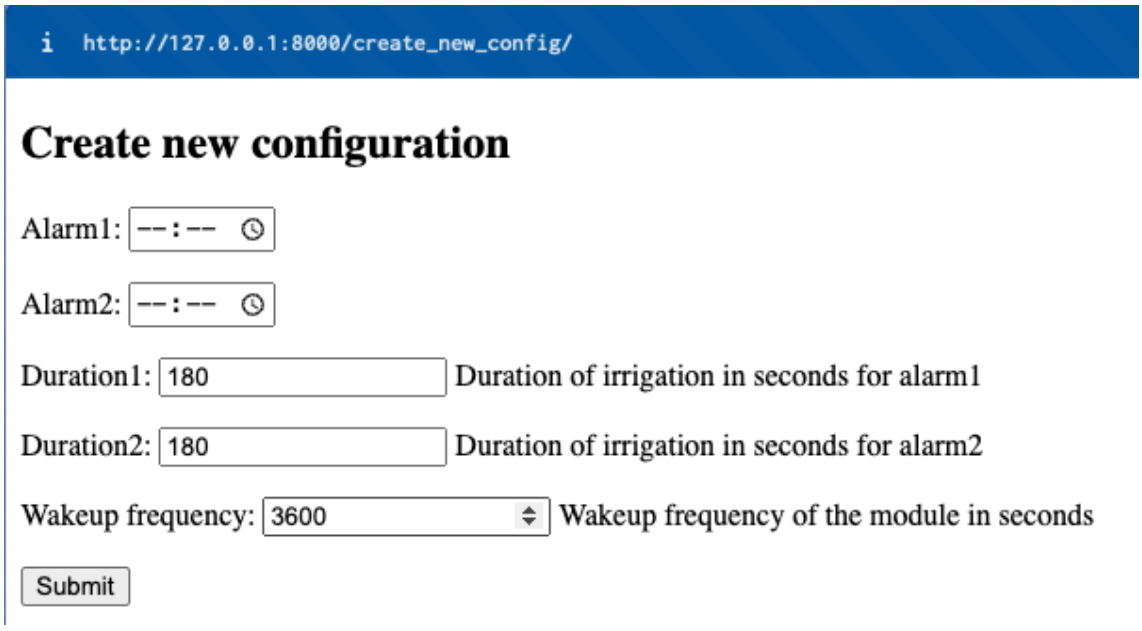


3.5. ábra: Új konfiguráció lekérése

Az új konfigurációk létrehozása egy külön végpont meghívásával történik. A felhasználótól való adatok bekérésére a "Form" osztályon keresztül van lehetőség. A formot létrehozó osztály kódja alább olvasható.

```
class ConfigCreationForm(Form):
    alarm1 = forms.TimeField(widget=forms.TimeInput(attrs={
        "type":"time"}))
    alarm2 = forms.TimeField(widget=forms.TimeInput(attrs={
        "type":"time"}))
    duration1 = forms.IntegerField(help_text="Duration␣of␣
        irrigation␣in␣seconds␣for␣alarm1",initial=180,
        min_value=60)
    duration2 = forms.IntegerField(help_text="Duration␣of␣
        irrigation␣in␣seconds␣for␣alarm2",initial=180,
        min_value=60)
    data_send_frequency = forms.IntegerField(help_text="
        Wakeup␣frequency␣of␣the␣module␣in␣seconds",initial
        =3600,min_value=1200)
```

A formot regisztrálva az egyes bemeneti elemek a megfelelő URL végpontot meghívva jelennek meg, amit a 3.6 ábra mutat.



3.6. ábra: Új konfiguráció készítése

Az adatok kitöltése után a *Submit* gombra kattintva a `create_new_config()` függvény hívódik meg, amely a bevitt adatok alapján egy új sort hoz létre az adatbázisban.

```
def create_new_config(request):
    if request.method == "POST":
        form = ConfigCreationForm(request.POST)
        if form.is_valid():
            config_data = {
                "alarm1": form.cleaned_data["alarm1"].
                    isoformat(),
                "alarm2": form.cleaned_data["alarm2"].
                    isoformat(),
                "duration1": form.cleaned_data["duration1"
                    ],
                "duration": form.cleaned_data["duration2"],
                "wakeup_freq": form.cleaned_data["
                    wakeup_freq"],
                }
            Config.objects.create(config=json.dumps(
                config_data))
            return redirect("create_new_config")
        else:
            form = ConfigCreationForm()
        return render(request, "config_change.html", {"form
            ": form})
```

3.3.6 Üzenetek

A modulüzenetek tárolása szintén JSON formátumban történik. Az üzenetek a megfelelő URL végpont POST kérés meghívásán keresztül kerülnek regisztrálásra. Az adatbázisba

való beillesztést a végpontra kialakított osztály *post* függvénye végzi el, amely az alábbi kódsorban látható.

```
def post(self,request):
    serializer = ModuleMessageSerializer(data=request.data.
       get("data"))
    if serializer.is_valid():
        new_entry = serializer.save()
    read_serializer = ModuleMessageSerializer(new_entry)
        return Response(read_serializer.data,status=201)
    return Response(serializer.errors,status=400)
```

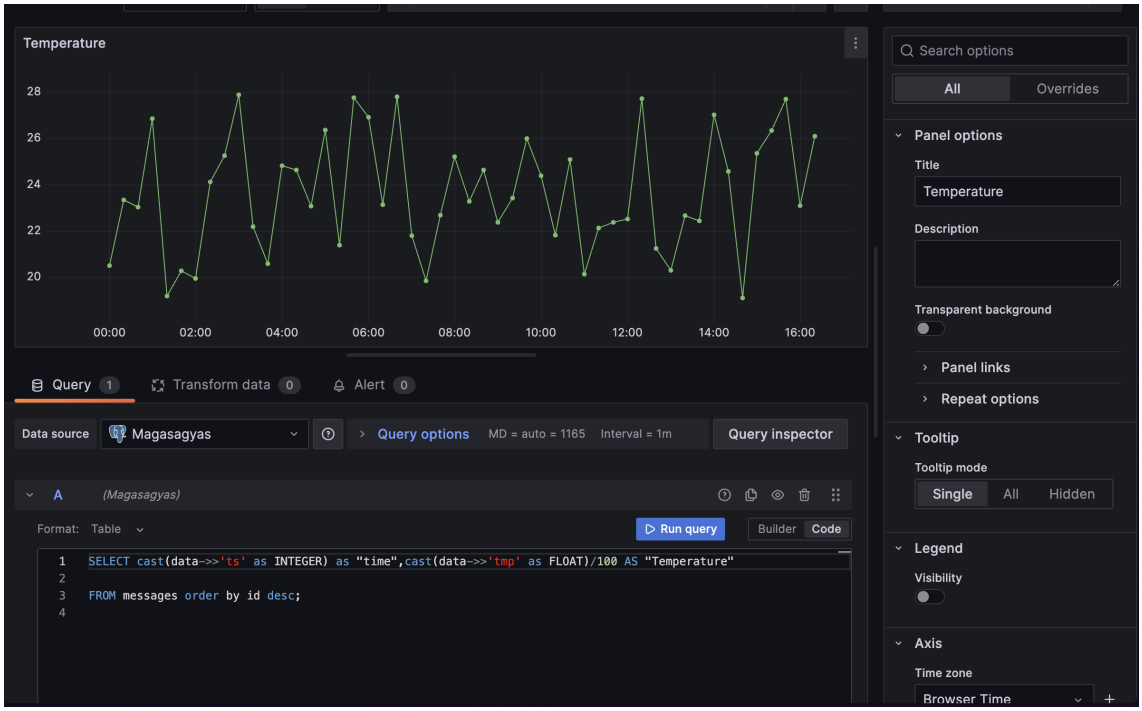
A fogadott adatok további feldolgozására, módosítására nincsen szükség, a szerver csupán a formátum megfelelőségét ellenőrzi, amelyet a *serializer* objektum végez el. Amennyiben az nem megfelelő, a szerver egy hibakódot küld vissza válaszként.

3.4 Adatvizualizáció

3.7. ábra: A személyreszabott dashboard végleges kinézete

Az adatok megjelenítése a modulüzenetek JSON objektumaiból kinyert paraméterek alapján történik. A bizonyos mértékig ingyenesen használható *Grafana Dashboard* remek lehetőséget kínál erre. Segítségével dashboardokat hozhatunk létre[19], amelyeken tetszőleges számú paneleket jeleníthetünk meg. Egy panel egy vagy több SQL kérés által szolgáltatott adatok vizualizációjának felel meg. A webapplikáció lehetőséget ad különböző típusú diagramok használatára, illetve az egyes grafikonelemek teljes személyre szabására. [20]

A vizualizációk teszteléséhez egy Python script segítségével 50 darab üzenet került beszúrásra az adatbázis "messages" táblájába véletlenszerű hőmérséklet és nedvességtartalom értékekkel. A 3.8 ábrán látható a Grafana kezelőfelülete egy panel elkészítésének alkalmával.



3.8. ábra: A Grafana Dashboard kezelőfelülete

4 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A tervezett rendszer eleget tesz az olyan igényeknek, amelyeket egy hasonló alkalmazásban egy átlagos felhasználó támasztana. Ennek ellenére a projektnek számos területen van még lehetősége fejlődni.

A nyáklap méretének csökkentése céljából lehetőség van e-SIM használatára. Egy ilyen áramköri alkatrész körülbelül harmad akkora mint egy SIM kártya tartó. Minden bizonnyal volna lehetőség az áramkör méretének csökkentésére az alkatrészek optimálisabb elhelyezésével is.
A jelenleg választott csatlakozók a kínálatban kapható legolcsóbb fajtából valóak. Az összeállítás színvonalát nagyban megemelné szabványosított, zárt tokozású csatlakozók használata.
Az áramkör jelenleg nem rendelkezik semmilyen por vagy víz elleni védelemmel. Mivel a rendszer szabad ég alatti felhasználásra lett tervezve, mindenképpen szükségeltetik egy erre a célra megfelelő ház. Ez lehet 3D nyomtatott, vagy fröccsöntött műanyag, előbbinél azonban a vízállóság megléte erősen kérdéses tud lenni.

Az időszinkronizáció a jelenlegi implementáció szerint statikus módon történik olyan tekintetben, hogy a pontos idő lekérésekor a Budapesti időzóna van megadva paraméterként. Annak érdekében, hogy a projekt bármilyen időzónában használható legyen előzetes beállítások nélkül, helymeghatározási funkciók szükségesek. Kiváló megoldást nyújthat erre a Google által nyújtott *Cellocate* szolgáltatás, amely a környező rádiótornyok és jelerősség függvényében képes egy becsült geolokáció meghatározására.

5 ÖSSZEFOGLALÓ

A dokumentáció megírásának idején valós tesztelésre nem volt lehetőség. A mesterségesen generált modulüzenetek segítségével az adatok vizualizációja ugyanakkor tesztelhető volt, amely a specifikációban támasztott feltételeknek hiánytalanul megfelelt.
A projekt során rengeteg tapasztalatot sikerült gyűjtenem mind az azt alkotó elemekről, mind arról, hogy hogyan is érdemes egy ilyet felépíteni, megtervezni. Ennek folyamán számos olyan hibát vétettem, amelyekből rengeteget tudtam tanulni. Tapasztalataim alapján kijelenthetem, hogy egy ehhez hasonló projekt legfontosabb alkotóeleme a hardver. A hardver tervezésekor elkövetett hibák tudnak a legköltségesebb tényezőknek bizonyulni. Természetesen amennyiben az ember rendelkezik a megfelelő technológiával és házon belül tud paneleket gyártani, ez nem akkora gond. A statikus hibakeresés általánosságban nehezebb egy kapcsolási rajz esetén programkódhoz képest, főleg a tapasztalatlan szemnek mint az enyém is. A következő revízióban mindenképpen jobban fogok ügyelni arra, hogy az aktív áramköri elemek és ellenállások mérete nagyjából megegyező legyen. Az alkatrészek szimbólumainak és lábnyomainak megfelelő összhangjára is nagyobb súlyt kell fektetnem, ugyanis az első prototípus tesztelésekor derült, ki hogy a fordított polaritás elleni védelmet szolgáltató MOSFET lábkiosztása nem megfelelő így az áramkör egyáltalán nem működött.

5.1 English summary

No real testing was possible at the time of writing. However, the data visualisation could be tested using the artificially generated module messages, which fully met the requirements of the specification.
During the project, I gained a lot of experience both about the elements that make up the system and about how to build and design it. In the process, I made a number of mistakes from which I learned a lot. From my experience, I can say that the most important component of a project like this is the hardware. Mistakes made when designing the hardware can prove to be the most costly factor. Of course, if you have the right technology and can manufacture panels in-house, this is not a problem. Static debugging is generally

more difficult for a schematic compared to program code, especially for inexperienced eyes like mine. In the next revision I will definitely take more care to make sure that the active circuit elements and resistors are roughly the same size. I will also have to pay more attention to the correct matching of the symbols and footprints of the components, as the first prototype testing revealed that the footprint of the MOSFET providing reverse polarity protection was not correct and the circuit did not work at all.

6 IRODALOMJEGYZÉK

[1] F. Firmin, *AT command set for User Equipment (UE)*, 27.007, 3GPP, 2015.

[2] winbond, *3V 32M-BIT SERIAL FLASH MEMORY WITH DUAL, QUAD SPI*, Rev G, winbond, 2018.

[3] C.-C. Hua és M.-Y. Lin, „A study of charging control of lead-acid battery for electric vehicles”, *ISIE'2000. Proceedings of the 2000 IEEE International Symposium on Industrial Electronics (Cat. No.00TH8543)*, 1. köt., 2000, 135–140 vol.1. DOI: 10.1109/ISIE.2000.930500.

[4] STMicroelectronics, *Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 256KB Flash, 64KB SRAM, analog, audio*, Rev 3, STMicroelectronics, 2018.

[5] T. Instruments, *HDC1080 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor*, Texas Instruments, 2015.

[6] Cím: https://www.liquidlevel.com/float-switches/.

[7] ublox, *SARA-R4 series*, UBX-16024152-R34, ublox, 2023.

[8] ublox, *SARA-R4 series System integration manual*, UBX-16029218 - R30, ublox, 2023.

[9] T. Instruments, *L293x Quadruple Half-H Drivers*, Texas Instruments, 2016.

[10] *A4988 DMOS Microstepping Driver with Translator DMOS Microstepping Driver with Translator And Overcurrent Protection*, Rev 4, Allegro Microsystems LLC, 2012.

[11] *A4988 Stepper Motor Driver Carrier*. cím: https://www.pololu.com/product/1182.

[12] mps, *100V Input, 3.5A, Switching Current Limit Step-Down Converter 100V Input, 3.5A, Switching Current Limit Step-Down Converter*, Rev 1.1, mps, 2017.

[13] R. Semiconductor, *PCB Layout Techniques of Buck Converter*, 2017.

[14] T. Instruments, *LP5912 500-mA Low-Noise, Low-Iq LDO*, Texas Instruments, 2016.

[15] Cím: https://sourcemaking.com/design_patterns/singleton.

[16] Cím: https://docs.djangoproject.com/en/4.2/intro/tutorial03/.

[17] Cím: https://docs.djangoproject.com/en/4.2/intro/tutorial01/.

[18] Cím: https://www.postgresql.org/docs/current/.

[19] Cím: https://grafana.com/docs/grafana/latest/dashboards/.

[20] Cím: https://grafana.com/docs/grafana/latest/datasources/.

[21] Cím: https://docs.djangoproject.com/en/4.2/intro/tutorial02/.

7 ÁBRAJEGYZÉK

8 TÁBLAJEGYZÉK