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

Nagy Borbála

Fenntartható mezőgazdaságot támogató okos üvegház fejlesztése IoT technológiákkal

Konzulens

Gódor Győző

BUDAPEST, 2023

Tartalomjegyzék

[Összefoglaló 6](#_Toc151914731)

[Abstract 7](#_Toc151914732)

[1 Bevezetés 8](#_Toc151914733)

[2 Használt eszközök 9](#_Toc151914734)

[2.1 Érzékelők 9](#_Toc151914735)

[2.1.1 Hőmérséklet és páratartalom 9](#_Toc151914736)

[2.1.2 Fényerősség 10](#_Toc151914737)

[2.1.3 Talajnedvesség 10](#_Toc151914738)

[2.1.4 Szélerősség 11](#_Toc151914739)

[2.1.5 Vízszint 11](#_Toc151914740)

[2.2 Beavatkozók 12](#_Toc151914741)

[2.2.1 Vízpumpa 12](#_Toc151914742)

[2.2.2 Ablaknyitó motor 13](#_Toc151914743)

[2.2.3 Ventilátor 13](#_Toc151914744)

[2.2.4 LED szalag 14](#_Toc151914745)

[2.3 Egyéb eszközök 14](#_Toc151914746)

[2.3.1 Relé 14](#_Toc151914747)

[2.3.2 Analóg-digitális átalakító 15](#_Toc151914748)

[2.4 Vezérlés 15](#_Toc151914749)

[2.4.1 Raspberry Pi Zero W 15](#_Toc151914750)

[2.4.2 Raspberry Pi 4 Model B 16](#_Toc151914751)

[3 A megvalósított rendszer 17](#_Toc151914752)

[4 Szoftverarchitektúra és implementáció 18](#_Toc151914753)

[4.1 Operációs rendszer 18](#_Toc151914754)

[4.2 Python 19](#_Toc151914755)

[5 Kommunikáció 20](#_Toc151914756)

[5.1 I2C 20](#_Toc151914757)

[5.2 MQTT 21](#_Toc151914758)

[5.2.1 Topicok 21](#_Toc151914759)

[5.2.2 Résztvevők feladatai 21](#_Toc151914760)

[5.2.3 Szolgáltatási szintek 21](#_Toc151914761)

[5.2.4 Kliensek lehetőségei 22](#_Toc151914762)

[6 Android alkalmazás 23](#_Toc151914763)

[6.1 Architektúra és technológiai háttér 23](#_Toc151914764)

[6.1.1 Android Room 23](#_Toc151914765)

[6.1.2 MVVM architektúra 24](#_Toc151914766)

[6.1.3 Paho MQTT kliens 24](#_Toc151914767)

[6.1.4 Navigáció 25](#_Toc151914768)

[6.1.5 Jetpack compose 25](#_Toc151914769)

[6.2 Felhasználói felület 26](#_Toc151914770)

[7 Utolsó simítások 28](#_Toc151914771)

[Irodalomjegyzék 29](#_Toc151914772)

[Függelék 30](#_Toc151914773)

Hallgatói nyilatkozat

Alulírott **Nagy Borbála**, 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, 2023. 11. 25.

...…………………………………………….

bála

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

Nagyon izgi lesz

# Használt eszközök

## Érzékelők

Elsőkörben megállapítható, hogy egy okos üvegháznak szüksége van egy szenzorrendszerre, amivel valós időben tudjuk monitorozni a benne uralkodó aktuális állapotokat. A növények számára fontos paraméterek szerencsére már köztudottak, így könnyen találhatunk olyan érzékelőket a piacon, amiket érdemes beépítenünk a rendszerbe.

### Hőmérséklet és páratartalom

Sajnos az utóbbi évek tavaszán szélsőségesen változó időjárással, hőmérséklettel kellett szembenézniük a növényeknek, ami sokszor a termelést törtrészeire tudja csökkenteni. Egy ilyen zárt rendszer nagy előnye az is, hogy a hőmérsékletet könnyen tudjuk állandósítani, így nem kell tartani az esetleges hirtelen lehűlésektől, melegedésektől.

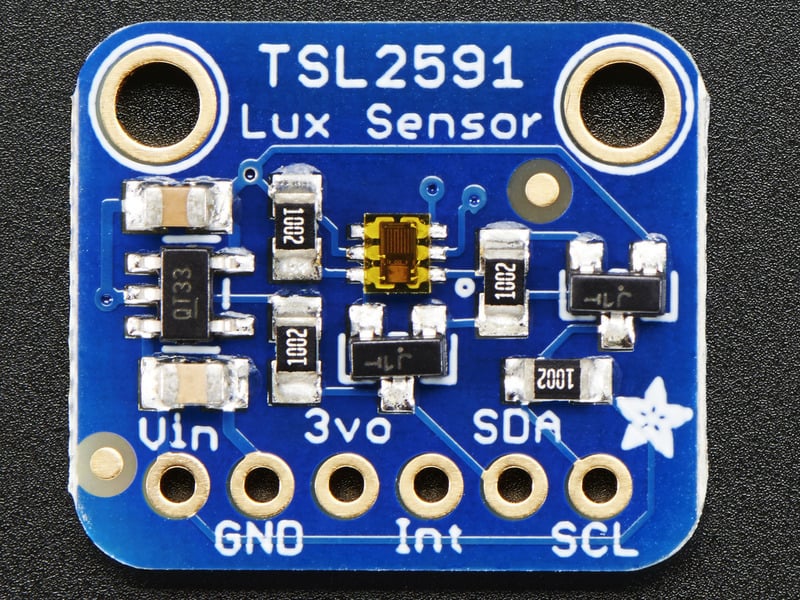
A rendszerben az Adafruit Si7021 hőmérséklet- és páratartalom-mérő szenzort használtam, ami -10 és +85 C° között képes mérni hőmérsékletet, tehát széles tartományú, és emellett nagy pontosságú is. A relatív páratartalmat 0 – 80% között tudja precízen jelezni, de felette is használható, valamivel kisebb pontossággal. A szenzor I2C kommunikációt használ.



2.1. ábra: Adafruit si7021

### Fényerősség

A fény mennyisége szintén létfontosságú, ehhez az Adafruit TSL2591 fényérzékelő szenzort használtam. Látható és infravörös tartományban is képes mérni, 188 uLux és 88000 Lux között. Összehasonlításképpen a holdtalan, borús éjszakai égbolt fényintenzitása 10-4 lux, egy lakószoba átlag megvilágítása 500 lux, a közvetlen napfény pedig 32000-130000 lux között van. Ez a szenzor is I2C-n keresztül kommunikál.

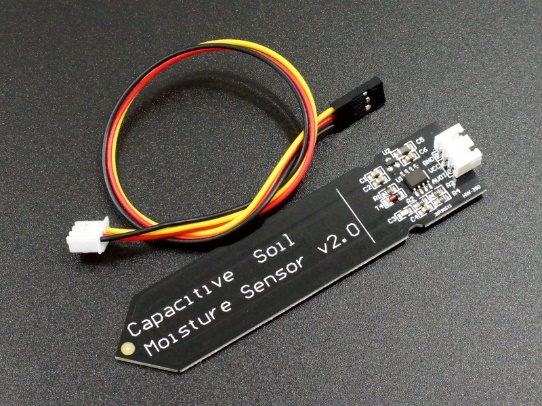


2.2. ábra: Adafruit tsl2591

### Talajnedvesség

Az egyik legalapvetőbb paraméter, amit mérnünk kell, a talajnedvesség. Erre a célra egy kapacitív talajnedvesség-mérő szenzort használtam, aminek előnyös tulajdonsága a hosszú élettartam, ami kapacitív mivoltának köszönhető.

A szenzor kimenete analóg, és a kimeneti feszültség fordítottan arányos a talajnedvesség szintjével. A nedves talaj jobb elektromos vezető, nagyobb a kapacitása, így a kimeneti feszültség csökken. Száraz talaj esetén ennek az ellenkezője történik, a vezetőképesség csökken, a kimeneti feszültség pedig megnő. Az általam mért két szélsőérték teljesen száraz szenzor esetén – 45%-os relatív páratartalommal a szobában – 1650 volt, vízbe mártva pedig 620.



2.3. ábra: Kapacitív talajnedvesség-mérő szenzor

### Szélerősség

A szélerősséget természetesen nem bent, hanem az üvegház mellett, azon kívül mérjük, amire azért van szükség, hogy tudjuk, mikor nyithatjuk ki az ablakot és mikor nem ajánlott. A méréshez az A1733 kanalas anemométert használtam, ami maximum 70 m/s szélerősségig mér, 1 m/s pontossággal. Az előzőhöz hasonlóan ez az érzékelő is analóg kimenettel rendelkezik, viszont ennek üzemi feszültsége 12V.



2.4. ábra: A1733 kanalas anemométer

### Vízszint

Az üvegház alatt helyezkedik el egy víztartály, amiből pumpák segítségével locsolunk, így azok megfelelő működéséhez tudnunk kell, mennyi víz van még a tartályban. Erre a célra a WLD-75 vízszintmérő szenzort használtam, aminek kimenete a vízszinttel arányos feszültség. Maximum értéke 100, ha 2 mm-re belelóg a vízbe, már legalább 500-as értéket olvashatunk ki róla, hogyha pedig teljesen száraz, az értéke 0, tehát az utolsó két mm-en változik annyit, mint szinte az egész hosszán.



2.5. ábra: WLD-75 vízszintmérő szenzor

## Beavatkozók

Az érzékelők mellett fontos pont az automatizálás is, így szükségünk van különböző beavatkozókra, hogy a könnyen változtatható paraméterek esetén elkerülhető legyen az emberi beavatkozás. Ezeknél elsődleges szempont, hogy a céljuk az, hogy minél kevesebb erőforrást fogyasszunk, minél optimálisabb legyen az energiafelhasználás, ezzel is javítva a hatékonyságot.

### Vízpumpa

A locsolás a legalapvetőbb eleme az üvegháznak, ehhez vízpumpákat volt szükséges beiktatni a rendszerbe. A használt vízpumpa a merülő vízszivattyúkra hasonlít, tehát víz alá kell helyezni, és onnan fogja kipumpálni a vizet. Működéséhez elengedhetetlen, hogy valóban víz alatt legyen használat közben, emiatt van szükség a fentebb említett vízszintmérő alkalmazására. Mivel On/Off alapú, így a rendszerhez egy relén keresztül kapcsoljuk, amivel ki/bekapcsolni tudjuk.

A képen kábel, dugó, füldugó látható

Automatikusan generált leírás

2.6. ábra: Vízpumpa

Itt még pár szót a csövekről, amiken megy a víz majd, vagy bármi hogy ne legyen ilyen üres a lap alja.

### Ablaknyitó motor

Amennyiben a kinti környezeti változók elősegítik, hogy az üvegházon belül elérjük a kívánt állapotokat, egy léptetőmotor segítségével ki tudjuk nyitni az ablakot. A motor üzemi feszültsége 12V, és egy mikrokontroller használata is szükséges működéséhez. Még pár dolog jöhet ide, miután kipróbáltam.

A képen Elektronikus alkatrész, Áramköri elem, Passzív áramköri elem, Elektrontechnika látható

Automatikusan generált leírás

2.7. ábra: Ablaknyitó motor

### Ventilátor

A hőmérséklet, és főként a páratartalom változtatására egy OEM PC ventilátort használunk. Alapvetően az üzemi feszültsége ennek is 12V, de fordulatszáma arányos a rá adott feszültséggel, tehát kisebb feszültségen is működik lassabb fordulatszámmal. Még pár dolog jöhet ide, miután kipróbáltam.

A képen ventilátor, eszköz, Mechanikus ventilátor, elektromos ventilátor látható

Automatikusan generált leírás

2.8. ábra: OEM PC ventilátor

### LED szalag

A fény növelésére és paramétereinek állítására RGB LED szalagot alkalmazunk, ami által kielégíthetjük a növények különböző hullámhosszok iránti igényüket. A növények fotoszintéziséhez szükséges hullámhosszok a 440 nm (ibolya) és 660 nm (vörös). A fejlődéshez, növekedéshez 660 nm és 735 nm (távoli vörös) szükséges, a levélképződést pedig a 435 nm-es (ibolya) fény segíti elő. A 440 nm a fotoszintézis mellett azért is felel, hogy a növények a fény irányába mozogjanak.



2.9. ábra: RGB LED szalag

## Egyéb eszközök

### Relé

Ahogyan fentebb már említésre került, néhány beavatkozó vezérlése relén keresztül történik. Erre a célra a Robofun 5V egycsatornás relé modult használtam.

A képen Áramköri elem, Elektronikus alkatrész, elektronika, Passzív áramköri elem látható

Automatikusan generált leírás

2.10. ábra: Robofun 5V relé modul

### Analóg-digitális átalakító

Mivel az általam használt vezérlő nem képes az analóg bemenet kezelésére, a szenzorok egy része viszont analóg kimenetű, így szükséges volt az AD konverter használata. A rendszerben az Adafruit ADS1115 16 bites, 4 csatornás átalakítót használtam, aminek előnye a magas felbontás, valamint az I2C interfész, ami könnyű interfészt biztosít. Ahogy a többi Adafruit termékhez, ehhez is pontos dokumentáció található, sok példakóddal, amik megkönnyítik a fejlesztés menetét.



2.11. ábra: Adafruit ADS1116 16-bit ADC

## Vezérlés

Az érzékelők és beavatkozók összekötéséhez szükségünk van vezérlő elemekre, amiben a rendszer logikáját tudjuk elkészíteni. Erre az alábbi eszközöket használtam.

### Raspberry Pi Zero W

A Raspberry Pi Zero egy kisméretű, alacsony költségű számítógép, amit kifejezetten olyan alkalmazásokhoz készítettek, ahol a kis méret, alacsony ár és alacsony energiafogyasztás fontos szempont.



2.12. ábra: Raspberry Pi Zero

Egy egymagos, 1 GHz-es processzort és 512 MB RAM-ot tartalmaz, ami bőven elegendő annak az egyszerű programnak, amit futtatni szeretnénk rajta. Memóriáját a többi raspberry-hez hasonlóan egy behelyezett SD kártya adja, ez a jelenlegi rendszerben egy Kingston 16 GB-os memóriakártya. Rendelkezik egy mini HDMI porttal, ami által monitort tudunk hozzá csatlakoztatni, két micro USB porttal az áramellátáshoz és a külső perifériákhoz – egér, billentyűzet –, valamint GPIO lábakkal. Beépített Wi-Fi és Bluetooth támogatást is tartalmaz, ami lehetővé teszi a vezeték nélküli kapcsolatot.

### Raspberry Pi 4 Model B

A Raspberry Pi 4 egy erősebb modell, szélesebb körben használható, akár nagyobb projektekre is.

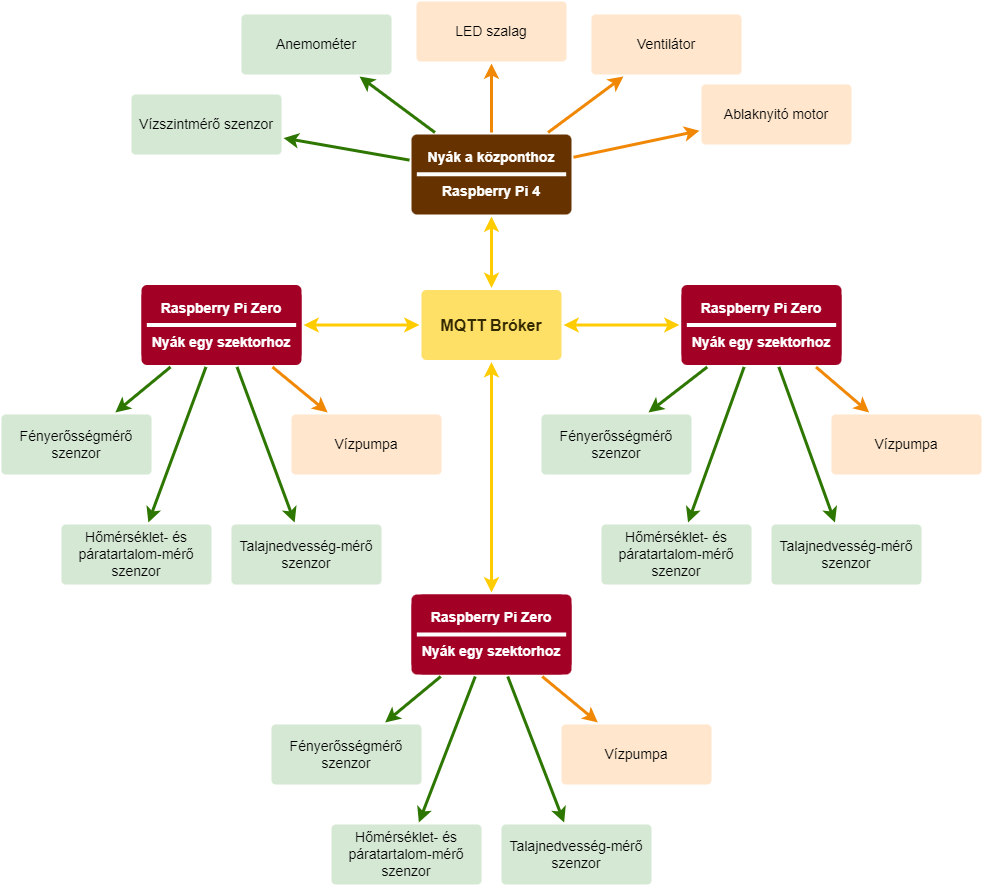


2.13. ábra: Raspberry Pi 4 Model B

Processzora egy négymagos ARM Cortex-A72, ami a korábbi modellekhez képest jelentős teljesítményjavulást hozott, X GB RAM-ot tartalmaz. Számos csatlakozóval rendelkezik, ideértve két 3.0 és két 2.0 USB portot, egy Gigabit Ethernet csatlakozót, két micro HDMI kimenetet, egy 3,5 mm-es hangkimenetet, egy USB C-s csatlakozót az áramellátás biztosítására, valamint GPIO lábakat. Memóriáját szintén egy SD kártya adja, jelen rendszerben egy SanDisk 16 GB-os memóriakártya. Szintén tartalmaz beépített Bluetooth és Wi-Fi támogatást.

# A megvalósított rendszer

A jelenlegi üvegházat három szektorra osztottuk.



3.1. ábra: A rendszer logikai felépítése

# Szoftverarchitektúra és implementáció

A Raspberry Pi által vezérelt rendszer megteremti a szenzorok és beavatkozók közötti összeköttetést, azonban ehhez elengedhetetlen egy ehhez megfelelő kód implementálása. A következőkben bemutatom az üvegház szoftveres oldalát, használt programozási nyelveket, esetleges problémákat, amibe az implementálás során belefutottam.

## Operációs rendszer

A használt Raspberry-ket a Raspberry Pi Imager használatával setupoltam fel, ami elsősorban a debian 12 használatát javasolja, aminek nagyon örültem, mert többször is dolgoztam már debianos rendszeren, illetve számos előnnyel rendelkezik.

Ez a legújabb debian, 2023 júniusában dobták piacra, így még hosszú ideig támogatott. Rengeteg dokumentációt lehet találni hozzá, ami segít a fejlesztésben és a problémák megoldásában. Emellett a Linux hatékonyan működik kevés erőforrással is, ami kifejezetten hasznos esetünkben, hiszen a Raspberry Pi Zero nem bővelkedik ezekben. Nagy mértékben és egyszerűen testreszabható, amire itt is szükség volt a különböző interfészek használatához. Stabil és biztonságos, ami különösen fontos egy ilyen projekt esetén, ahol valós környezetben a kihelyezés után ritkán lehet és kell hozzáférni az eszközökhöz.

Egyetlen apró buktató volt, a raspberry pi imager a 64 bites linuxot javasolja, viszont a zero csak 32 bites rendszeren működik, így végül azt kellett rá feltelepíteni. Ettől eltekintve viszont nem adódtak problémák a telepítés során, az imager megengedi, hogy már ekkor beállítsunk bizonyos paramétereket, az engedélyezett interfészeket, ssh kapcsolatot, hálózatokat, amire automatikusan csatlakozik az eszköz.

Az egyik zero-t még előző félévben, önálló labor keretein belül üzemeltem be, így azon debian 11 fut, hiszen akkor még az volt az aktuális legújabb verzió. Végül nem cseréltem le, mivel még ez az operációs rendszer is sokáig támogatott, használatában nincs különbség a 12-től.

## Python

Egy ilyen projekt esetén, valamint raspberry használata mellett valószínűleg mindenkinek az lenne a logikus lépés, hogy pythonban kódoljon, így természetesen én is erre a következtetésre jutottam.

Bár egyetemi keretek között nem tanultunk python-ban programozni, könnyű olvashatósága és egyszerűsége miatt nagyon gyorsan tanulható. Magas szintű nyelvi elemeket tartalmaz, ami meggyorsítja a fejlesztés menetét és megkönnyíti a karbantartást. Platformfüggetlen, tehát ugyanazt a kódot futtathatjuk linux és windows rendszereken is. Nagy és aktív fejlesztői közösséggel rendelkezik, ami rengeteg támogatást jelent, ha most kezdünk belemerülni a nyelvbe jobban.

Ezen kívül rengeteg külső könyvtárat és modult kínál a széleskörű funkciók eléréséhez, ami magában foglalja az IoT szenzorok kezelését is. Ezen könyvtárak közé tartozik többek között az RPi.GPIO is, amivel a raspberry GPIO lábait tudjuk könnyedén beállítani, valamint az Adafruit-Blinka is, ami az adafruitos szenzorok olvasásához elengedhetetlen.

### Csatlakoztatott eszközök elérése

Az első feladat az volt, hogy a kódból le tudjuk kérni a különböző szenzorok akuális értékeit. Szerencsére minden szenzorhoz találtam dokumentációt példakóddal, beimportálható könyvtárral, ideértve az AD átalakítót is. Ennek következtében ezen eszközök inicializálásához csupán ez a pár sor kellett:

import adafruit\_tsl2591

import adafruit\_si7021

import Adafruit\_ADS1x15

i2c = board.I2C()

lightsensor = adafruit\_tsl2591.TSL2591(i2c)

thsensor = adafruit\_si7021.SI7021(i2c)

adc = Adafruit\_ADS1x15.ADS1015(address=0x48, busnum=1)

Okozott egy kis problémát, hogy eleinte nem találta az I2C eszközt, mivel nem tudtam, hogy nem a default, 0x48-as címen volt, hanem a 0x49-esen. Az i2cdetect paranccsal végül könnyen fel tudtam térképezni, melyik eszköz milyen címen van, így már az ADC által küldött értékeket is ki tudtam olvasni, ami az analóg szenzoroktól származott. Végül kiderült, hogy a nyákon az address jumper összeköttetésben maradt az ADC-vel, és az okozta ezt a problémát, így azóta eltávolítottam, és már a 0x48-as címet használom.

A fő egységnél jött elő az a probléma, hogy eleinte nem találtam, a relé melyik pinre van kötve. Ennek kiderítésére multiméterrel megmértem a relé bemenetén található feszültséget, valamint az egyes lábakét is, így hamar megtaláltam a szükséges lábat.

Emellett a LED vezérléséhez is külön módszer kellett, mert a GPIO lábak csak két értéket tudnak felvenni, vagy küldünk rájuk áramot, vagy nem, ez azonban nem megfelelő arra, hogy bármilyen rgb színkombinációt elő tudjunk állítani. Ennek a megoldására használtam a PWM, azaz Pulse Width Modulation technikát, ami lehetővé teszi, hogy egy digitális jellel közelítőleg utánozzunk egy analóg jelet. Segítségével megadhatjuk, hogy milyen frekvenciával és milyen kitöltési tényezővel küldünk áramot az adott pinre. Így tehát hogyha megadjuk, hogy a piros színre kötött láb 100%-ban legyen aktív, a zöld 60%-ban, a kék pedig 10%-ban, akkor az RGB(255, 153, 25) narancssárga színt kapjuk a led szalagon. A vezérlésből egy kódrészlet:

red = 16

GPIO.setup(red, GPIO.OUT)

r = GPIO.PWM(red, 100)

…

def on\_message(client, userdata, msg):

    if (msg.topic).find("light") != -1:

        if (msg.payload.decode()).find("on") != -1:

            print("turn on light")

            rgb = msg.payload.decode().split(" ")

            r.start(int(rgb[1]))

…

        if (msg.payload.decode()).find("off") != -1:

            print("turn off light")

            r.stop()

…

### MQTT kliens

A kódban a Paho MQTT klienst használtam, ami egyszerű interfésszel rendelkezik, és tökéletesen megfelel a célnak. Az újrafelhasználhatóság érdekében külön fájlba került a kliens inicializálása, a feliratkozás és a publikálás is, így ezek mind importálva vannak a fő kódba.

Az eszközöket jelenleg 3 másodpercenként olvassa a kód, és ugyanennyi időként publikálja ezeket az értékeket mqtt üzenetben a brókernek. A feliratkozás a szükséges topicokra a futás elején történik, ahol az eredeti üzenetkezelő függvényt felül is írjuk egy sajáttal. Ennek a feladata az, hogy bizonyos üzenetekre helyesen tudjon reagálni a program, amire egyik példa a szektortokban a locsolás vezérlése. Ennek során feliratkozunk a FIM3VE/general/waterlevel topicra, amire a fő egység 500-as vízszint alatt „danger” üzenetet küld, felette pedig „enough”-t. Amennyiben a legutolsó üzenet danger volt, akkor, ha akarjuk sem tudjuk bekapcsolni a locsolást, ugyanis leégne a locsoló motorja elegendő víz hiányában. Amennyiben megérkezik az enough üzenet, ismét engedélyezve lesz a locsolás, amire az utasítást a FIM3VE/sector<szektor\_száma>/water topicon kapjuk.

global KEVES\_VIZ

KEVES\_VIZ = True

def on\_message(client, userdata, msg):

    global KEVES\_VIZ

    print(f"Recieved `{msg.payload.decode()}` from `{msg.topic}` topic")

    if (msg.topic).find("sector3/water") != -1:

        if not KEVES\_VIZ:

            print("locsol")

            GPIO.output(relay, True)

            time.sleep(5.0)

            GPIO.output(relay, False)

    if (msg.topic).find("general/waterlevel") != -1:

        if (msg.payload.decode()).find("danger") != -1:

            KEVES\_VIZ = True

        elif (msg.payload.decode()).find("enough") != -1:

            KEVES\_VIZ = False

subscriber.subscribe(client, "FIM3VE/general/waterlevel", on\_message)

subscriber.subscribe(client, "FIM3VE/sector3/water", on\_message)

subscriber.run(client)

## Automatikus indítás

Fontos, hogy amint áram alá kerül a raspberry, elinduljon rajta magától a program, ne kelljen semmilyen emberi beavatkozás hozzá. Erre a célra először a crontabot használtam, azonban az nem hozta a kívánt eredményt, így más eszközt kerestem. A használt operációs rendszer systemd-t használ, ami átfogó és hatékony kezelést biztosít a rendszernek indulásától a leállításáig, valamint a futó szolgáltatásoknak. Konfigurációs fájlokban tárolja az indítási és szolgáltatási beállításokat, elkerülve ezzel a hagyományos parancsfájlok használatát. Az általa vezérelt rendszerek közös naplófájlrendszert használnak, ami lehetővé teszi a különböző szolgáltatások naplóbejegyzéseinek csoportosítását és könnyű kezelését. Célja az indítási folyamatok párhuzamosítása, hogy ezzel meggyorsítsa a rendszerindítást.

Szerettem volna, hogy az általam írt python script is elinduljon a rendszerrel együtt, így létrehoztam az alábbi systemd service konfigurációs fájlt.

[Unit]

Description=Start Monitoring Service

After=network.target

[Service]

ExecStart=/usr/bin/python3 /home/uveghaz/kod/main.py

WorkingDirectory=/home/uveghaz/kod/

Restart=always

User=uveghaz

Group=users

Environment=PATH=/usr/bin:/usr/local/bin

Environment=NODE\_ENV=production

RestartSec=3

StandardOutput=syslog

StandardError=syslog

SyslogIdentifier=start\_monitoring

[Install]

WantedBy=multi-user.target

A szolgáltatás neve Start Monitoring lett, és mivel a kód helyes futásához szükség van arra, hogy hálózatra kapcsolódjon az eszköz, így az After paraméterben megadtam, hogy csak a hálózat beállítása után futtassa ezt a szolgáltatást. A Restart=always sor is egyértelműen indokolt, hogyha valami esetleg hibát okozna a programban, nem tudna wifire csatlakozni egyből, minél kevésbé legyen ez a felhasználó számára észrevehető, próbálkozzon újra a program.

Ezek után már csak arra volt szükség, hogy a main.py fájlnak helyesen állítsam be a jogosultságait, a users group tudja futtatni, illetve az uveghaz felhasználó birtokolja azt. A szolgáltatás indítása és engedélyezése után elindult a kód, és minden újraindításnál ezt sikeresen meg is teszi.

# Kommunikáció

A rendszer fontos pontja a kommunikáció a szenzorok, beavatkozók és a vezérlés között, amire az eszközöktől függően több különböző technológiákat ismerhettem meg, amik közül az egyik legfontosabb az I2C kommunikáció. Ezek mellett lényeges pont a szektorok és a központi egység közötti, valamint az ezek és a felhasználó közötti adatátvitel, aki számára szükséges, hogy lássa a mért adatokat. Olyan technológiára van szükségünk, ami hatékonyan tudja kezelni a több forrásból érkező és több címzett számára küldött üzeneteket, erre a célra pedig ideális megoldást nyújt az MQTT (Message Queuing Telemetry Transport).

## I2C

Az Inter-Integrated Circuit protokoll olyan kommunikációs forma, amely lehetővé teszi, hogy több slave IC kommunikáljon egy vagy több masterrel. A master felel a kommunikációért, ő a főnök, a slave pedig – jelentése szolga – az, akitől lekérjük, vagy neki küldjük az információt. A többszereplős kommunikációt úgy valósítja meg, hogy minden eszköz egyedi azonosítóval rendelkezik, így nem keverednek össze az adataik.

A képen szöveg, sor, diagram, Betűtípus látható

Automatikusan generált leírás

5.1. ábra: I2C kommunikáció

Kétvezetékes soros adatátvitelt tesz lehetővé, ahol a két vezeték az SDA – Serial Data Line –, amin az adatok továbbítódnak, és az SCL – Serial Clock Line –, amin az órajel pulzusokat küldi az aktuális master, aki generálta. Általában alacsony adatátviteli sebességgel működik, 100 kHz-en vagy 400 kHz-en.

## MQTT

Az MQTT egy nyílt, ingyenes protokoll, amit arra terveztek, hogy megbízható üzenetküldést valósítson meg a lehető legegyszerűbb üzenetformátumban. A kommunikációban három típusú fél vesz részt, bróker, publisher és subscriber.



5.2. ábra: Az MQTT működése

### Topicok

Az üzenetek megkülönböztetésére, osztályzására az úgynevezett topicok szolgálnak, ezek definiálják az üzenet tartalmát. Általában hierarchikusan szervezettek, a „/” karakter használatával tudunk létrehozni al-topicokat. Egy „/” karakter utáni „#” jellel tudunk feliratkozni az adott topic összes al-topicjára. Erre példa a jelenlegi rendszerből:

FIM3VE/sector1/#

FIM3VE/sector1/lightness

FIM3VE/sector1/temperature

### Résztvevők feladatai

A subscriberek, nevükből adódóan, feliratkozhatnak a különböző topicokra, ami után az összes, arra a topicra publikált üzenetet megkapják. Nem ismerik a publishereket, csak a brókert. A publisherek feladata, hogy az általuk meghatározott topicra publikáljanak. Nekik nem szükséges ismerni, hányan és kik iratkoztak fel a témáikra, csak a bróker kilétéről van tudomásuk. Egy kliens lehet egyszerre publisher és subscriber is, ez a kettő nem zárja ki egymást, sőt egy kliens több témára is nyugodtan felirakozhat. A bróker feladata, hogy menedzselje a kliensek közti üzenetküldést, hogy minden subscriber megkapja az általa rendelt üzeneteket. A jelenlegi rendszerben egy online elérhető publikus brókert használtam a HiveMQ oldaláról.

### Szolgáltatási szintek

Három szolgáltatási szintet definiáltak az MQTT-ben. A magasabb szolgáltatásminőség nagyobb megbízhatóságú üzenet célba juttatást valósít meg, de természetesen ennek ára is van, ami a nagyobb sávszélesség és/vagy késleltetés.

A szerver megtartja az utoljára elküldött üzenetet, és egy új feliratkozó esetén egyből elküldi ezt is a kliensnek. A szolgáltatási szintek abban különböznek egymástól, hogy QoS = 0 esetén a szerver legfeljebb egyszer, QoS = 1 esetén legalább egyszer, QoS = 2 esetén pedig pontosan egyszer küldi el a megőrzött üzeneteket.

### Kliensek lehetőségei

A klienseket egy 23 bájtos egyedi string azonosítja. Amikor csatlakozik egy kliens a szerverhez, beállíthat egy clean-session flaget, aminek 1-es értéke esetén a kliens összes feliratkozása törlődni fog, ha az eszköz lekapcsolódik a szerverről. Nulla érték esetén a kliens előfizetése egészen addig élő marad, amíg vissza nem kapcsolódik, és ekkor az összes addigi üzenet elküldésre kerül neki.

Ezek mellett egy végrendeletet (will) is megadhatnak, ami által, ha a kliens váratlanul lecsatlakozik, akkor a szerver egy üzenetet küld a kliens által előre meghatározott topicra. Ilyen lehet akár egy riasztás, ha egy érzékelő lecsatlakozott.

# Android alkalmazás

Felhasználói oldalról természetesen akkor van igazán haszna egy okos üvegháznak, hogyha távolról is tudjuk figyelni az aktuális állapotokat és esetleg be is tudunk avatkozni. Erre a célra két lehetőség fordult meg a fejemben, egy telefonos alkalmazás vagy egy weboldal. Mindkettőnek megvannak az előnyei és a hátrányai, viszont az androidalapú szoftverfejlesztés tárgy keretein belül jobban meg tudtam ismerni az androidra való fejlesztést, így végül azt választottam.



6.1. ábra: A GreenHouse alkalmazás ikonja

## Architektúra és technológiai háttér

A fejlesztés az android hivatalos nyelvén, kotlinban történt, ami egy modern, statikusan típusos programozási nyelv. Teljes mértékben kompatibilis a Java-val, illetve elég hasonlóak is, így nem nehéz megtanulni a szintaktikáját a Java után. Támogatja a funkcionális és az objektumorientált programozást is, sok funkcionális programozási elemet tartalmaz.

### Android Room

Az adatbázis kezelésére az Android Roomot használtam, ami a háttérben SQLite adatbázist használ, és ahhoz biztosít absztrakt réteget, így az alkalmazásban könnyen és hatékonyan kezelhetjük a lokális adatokat. Együttműködik az ORM, Object-Relational Mapping koncepcióval, ami alapján minden osztály egy tábla, minden objektum egy sor a táblában és minden attribútum egy oszlop.

Bár a legtöbb adatot az MQTT brókertől kapja az alkalmazás, a használt topicokat érdemes elmenteni, hogy ne kelljen minden alkalommal újra beírni a felhasználónak. A mentett adatok tehát minden szektorhoz egy név, egy mqtt topic, illetve a benne termesztett növények, amiket megadhat a felhasználó. Ezeket természetesen szerkeszteni és törölni is lehet.

### MVVM architektúra

Az alkalmazást az MVVM, azaz Model–View–ViewModel architektúra alapján építettem fel. A Model a legalsó réteg, ami az alkalmazás üzleti logikáját és adatmodelljét tartalmazza, független a felhasználói interfésztől. A View a legfelső réteg, ami a felhasználói interfészt reprezentálja, passzív, és csak a ViewModel-től kapott adatokat jeleníti meg. Köztük áll a ViewModel, ami a Model által szolgáltatott adatokat fordítja olyan formába, ami könnyen megjeleníthető a View számára. Ő felelős a felhasználói interakciók kezeléséért és a megfelelő műveletek elindításáért. A kommunikáció tehát a szomszédos rétegek között kétirányú.



6.2. ábra: MVVM architektúra

### Paho MQTT kliens

Több olyan csomag is elérhető az interneten, aminek segítségével androidos alkalmazásban tudunk mqtt üzeneteket kezelni, ezek közül a paho mqtt klienst választottam, mivel ehhez találtam a legtöbb dokumentációt, példakódot, segédletet. A példakódok miatt a rendszerbe való integrálása eleinte nagyon könnyen ment, viszont hamar abba a problémába ütköztem, hogy az egyik fájlban egy olyan flaget használnak, ami android 12-től már nem támogatott. Mivel ezt a kódot nem én írtam, hanem az importált csomag használja, így nem tudtam csak egyszerűen átírni. Szerencsére azonban nyílt forráskódú, githubon megtalálható, és már több pull request is érkezett ennek a problémának a kijavítására, így tudtam találni egy olyan változatot, amit már egy felhasználó kijavított, és a jitpack segítségével az ő kódját tudtam használni.

### Navigáció

A programon belüli navigációhoz az android navigation component könyvtárát használtam, ami tartalmaz egy NavHost-ot és egy NavController-t. Első lépésként screen-eket, képernyőket kell definiálnunk, amiket odaadhatunk a NavHost-nak. A NavHost egy compose komponens, amiben megadhatjuk, melyik screennel akarjuk indítani az alkalmazást, illetve az adott screenekben melyik navigációs gombra nyomva melyik képernyőre navigáljunk. Összességében elég egyszerű és átlátható navigációt tudunk készíteni ezzel a technológiával, ami könnyen módosítható és megbízható. A képernyők között így még információt is tudunk továbbítani, ami jelen programban is hasznos, amikor kiválasztunk a szektorok listájából egyet, hiszen akkor a következő képernyőnek szüksége van arra az információra, hogy melyik konkrét szektor adatait töltse be.

### Jetpack compose

A Jetpack Compose egy modern toolkit UI készítéséhez, ami által a kotlin kódban írhatjuk le a UI paramétereit, ami alapján a compose motor generálja a felületet. Előnyei, hogy kevesebb kóddal tudjuk ugyanazt a nézetet elérni, nincs szükség hozzá a korábban megszokott XML layoutra. Könnyebben újrafelhasználható, hatékonyabb, illetve sokkal egyszerűbb így eljuttatni a megjelenítendő adatokat a felülethez. Készíthetünk hozzá preview-kat, aminek köszönhetően nem kell lefuttatni a programot ahhoz, hogy egy adott nézet elrendezését megtekinthessük, ami különösen a fejlesztés korai szakaszában nagyon hasznos. A fentebb említett androidalapú szoftverfejlesztés tárgy keretein belül jobban megismerhettem ezt a technológiát, így emiatt és sok előnyös tulajdonsága miatt természetes volt, hogy most is ezt fogom választani.

## Felhasználói felület

Az alkalmazás három fő képernyővel rendelkezik. Megnyitáskor a Settings képernyőn találjuk magunkat, ahol beállíthatjuk, melyik mqtt brókerhez akarunk csatlakozni, illetve megadhatjuk az üvegház fő egységének az MQTT topicját. Amennyiben sikeres volt a kapcsolódás, egy Toast üzenetet láthatunk, ami ezt megerősíti.

Átlépve a General fülre találhatjuk meg a központi egységhez csatolt érzékelőket és beavatkozókat. Amennyiben 500 alatt van a water level érték, az alkalmazás egy figyelmeztető üzenettel jelzi, hogy kevés víz van a tartályban, töltsük fel azt. Hogyha elegendő a víz, az üzenet eltűnik. A Red, Green, Blue mezőkben 0-255-ig adhatjuk meg, milyen rgb értéket vegyen fel a LED világítás, ami akkor kerül beállításra, ha a Change light gombra rányomunk, aminek feladata elküldeni az mqtt üzenetet a megfelelő topicra. A Start ventilator gombbal a ventilátort indíthatjuk el, az Open window gombbal az ablakot kinyittathatjuk, a Close window gombbal pedig bezárhatjuk.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

6.3. ábra: Settings és General képernyők

A Sectors fülre kattintva láthatjuk a szektorokat felsorolva a listában. Hogyha szeretnénk újat hozzáadni, azt a jobb alsó sarokban található plusz gombbal tehetjük meg, ekkor egy hasonló képernyő jelenik meg, mint ami a középső képernyőképen látható. Mentés után a szektorok listájához kerülünk, ahol egy adott szektorra kattintva megjelenik a jobb oldali képernyőképen látható ablak, ahol az adott szektorhoz tartozó szenzorok értékeit láthatjuk. A Water it gombra kattintva elindíthatjuk a locsolót, ami adott mennyiségű vizet pumpál a szektorba. A szektort a jobb felső sarokban lévő ceruza ikonra kattintva tudjuk szerkeszteni, a kuka ikonnal pedig törölni. Az összes szektor együttes törlésére is lehetőség van a szektorok listájának a képernyőjén, a bal alsó sarokban elhelyezkedő kuka lebegő gombbal.

A képen szöveg, képernyőkép, szám, tervezés látható

Automatikusan generált leírás

6.4. ábra: Sectors képernyő változatai

# Utolsó simítások

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. Levendovszky, J., Jereb, L., Elek, Zs., Vesztergombi, Gy.: Adaptive statistical algorithms in network reliability analysis, Performance Evaluation - Elsevier, Vol. 48, 2002, pp. 225-236
2. National Istruments: LabVIEW grafikus fejlesztői környezet leírása, <http://www.ni.com/> (2010. nov.)
3. Fowler, M.: UML Distilled, 3rd edition, ISBN 0-321-19368-7, Addison-Wesley, 2004
4. Wikipedia: Evaluation strategy, <http://en.wikipedia.org/wiki/Evaluation_strategy> (revision 18:11, 31 July 2012)

Függelék