



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Interaktív hardware, szoftver fejlesztés a megújuló energiaforrások népszerűsítése céljából

Készítette

Sass-Gyarmati Norbert
Programtervező informatikus BSc

Oravecz Zsolt
Programtervező informatikus BSc

Témavezető

Dr. Geda Gábor
Egyetemi docens

EGER, 2021

Tartalomjegyzék

1. A rendszerről összefoglalóan	7
1.1. Jelenlegi probléma	7
1.2. Tudományos Diákköri Konferencia	9
1.3. Terepasztalunk felépítése	10
2. Rendszerünk működése	14
2.1. A mozgatáshoz szükséges számítások	14
2.1.1. Napállás kalkulátor	14
2.1.2. Napállás meghatározása PHP nyelven	21
2.2. Szimulációk	23
2.2.1. Fogyasztó probléma szimulálása	23
2.2.2. Vízerőmű szimulálása	26
2.2.3. Napcellák teljesítményének szimulálása	29
2.2.4. Encoder működése	32
2.3. Optimalizálás	36
2.3.1. Telepített napcellák optimalizálása, tájolása	36
2.4. Termelők és fogyasztók	38
2.4.1. Termelők ismertetése	38
2.4.2. Fogyasztók ismertetése	38
2.5. Modellek	39
2.5.1. Időjárás állomás	39
2.5.2. Fogyasztók modellezése	43
2.5.3. Vízerőmű modellezése	47
2.6. Prototípusok	50
2.6.1. Telepített napelemek prototípusai	50
2.6.2. Fénykövető napelemek prototípusa	51
2.7. Napcellák	56
2.7.1. Telepített napcellák	56
2.7.2. Fénykövető napelem	58
2.7.3. Napcellák integrációja	63
2.8. Vízerőmű	65

2.9. További tervezetek	66
2.10. Összefoglalás	67
3. Weboldal	68
3.1. Bevezetés	68
3.2. CodeIgniter keretrendszer	68
3.3. Adatbázis	69
3.4. Weblapról	69
3.4.1. Az eszközök vezérlése	69
3.4.2. Várólista	71
3.4.3. Bejelentkezés	73
3.4.4. Kezdőlap	74
3.4.5. Rólunk	74
3.4.6. Blog	74
3.4.7. Kategóriák	75
3.4.8. Térkép	76
3.4.9. Jelmagyarázat a térképhez	77
3.4.10. Eszközök	78
3.4.11. Kvíz lehetőség	79
3.4.12. Felhasználók	80
3.4.13. Kapcsolatfelvétel	81
3.4.14. Galéria	82
3.5. További tervezetek	83
3.6. Összefoglaló	83
4. A weboldal és a mikrokontroller közötti információcsere	85
5. Fejlesztői környezetek és publikációi	90
5.1. Git verziókövető rendszer	90
5.2. Trello feladatkövető rendszer	91
5.3. Technológiák	92
5.3.1. Python	92
5.3.2. PHP nyelv	93
5.3.3. C++ nyelv	94
5.4. Mikrokontrollerek, szenzorok és kellékek ismertetése	94
5.4.1. Arduino Uno	94
5.4.2. Arduino Nano	95
5.4.3. LCD kijelzők	95
5.4.4. Dht11	96
5.4.5. Encoder	96

6. Összefoglalás	99
6.1. Megjegyzések	99

Bevezetés

Megújuló energiaforrásnak nevezzük az energiahordozók azon csoportját, amelyek emberi időléptékben képesek megújulni, azaz nem fogynak el, ellentétben a nem megújuló energiaforrásokkal. A mai civilizáció a zöld energiát helyezi előtérbe, és arra törekszik, hogy minél kisebbre csökkentse az ökológiai lábnyomát. Számos gyakorlati felhasználása van, többek között a villanyautók, tisztán elektromos hajtással működő személygépjárművek, illetve egyéb járművek fejlesztése. Napjainkban számos helyen tapasztaljuk, hogy egyre nagyobb szerepet kap a fenntarthatóság és a környezettudatosság nemcsak a vállalatok és cégek, hanem a fogyasztók gondolkodásában is. Egyre több szerepet kap az életünkben a környezettudatos életmód, a szelektív hulladékgyűjtés és a zöldebb életmód.

Számos előnyökkel rendelkeznek a megújuló energiaforrások, például, hogy ezek hosszú távon rendelkezésre álló készletek, szemben a nem megújuló energiaforrásokkal, melyek fosszilis energiahordozók. A fosszilis energiahordozók nem tartanak örökké, hiszen ezeket a földből kinyerve nem lehet pótolni, ha már véglegesen elfogytak. Ide tartozik az urán, a kőolaj, földgáz, illetve a szén. A zöldenergia rendelkezik egy másik nagy előnyivel, hogy működésük rendkívül környezetkímélő. A fosszilis energiahordozók égetése hatalmas mennyiséggű szén-dioxidot bocsát ki, ezzel mesterségesen növelte az üvegházhatás folyamatát a Földünkön, ezzel szemben a megújuló energiaforrások használatával sokkal kevesebb károsanyag kerül a légkörbe, melyeknek felhasználását egyre több ország helyezi előtérbe, hogy ezzel is mérsékelni tudják a globális felmelegedés problémáját.

Szakdolgozatunkban a megújuló energiaforrások tudatos, és környezetvédő felhasználását szeretnénk modellezni. A modellünkben az energiaforrásokat mikrokontrollerrel ötvözve szeretnénk a leghatékonyabban szabályozni, azaz a rendszer képes önállóan optimalizálni a termelt és a felhasznált energia mennyiségét. Gyakorlati felhasználásban terepasztalon elhelyezett kisméretű modellekben szemléltetjük a különféle erőműveket, zöld energetikai rendszereket, illetve energiatároló technológiákat, fogyasztókat. Fogyasztóinknak gyakorlati felhasználásuk lesz, mely azt jelenti, hogy a való életben megtalálható általános fogyasztókkal fogjuk szimulálni a projektet. Ilyenek lehetnek a családi házak, háztömbök, kórházak, valamint iskolák.

Szakdolgozatunk továbbá egy webalkalmazást is tartalmaz, mely segítségével a fel-

használók a felületen keresztül vezérelhetik terepasztalunk bizonyos eszközeit és funkcióit. Ezen a webes applikáción keresztül tudathatják velünk a felhasználók az egyes észrevételeiket, illetve panaszaikat. Lehetőségük van posztokat készíteni egyes eszközökről, valamint a TDK keretein belül létrejövő Interaktív hardware – software-rendszer fejlesztése természettudományos ismeretek népszerűsítése céljából a Varázstoronyban című projekt létrejöttével lehetőségük van az interneten keresztül bejárni az egri Varázstornyt.

1. fejezet

A rendszerről összefoglalóan

1.1. Jelenlegi probléma

Világunk működéséhez szinte elengedhetetlen az energia felhasználás. Rengeteg áramot használunk el naponta, melyet többek között a nem megújuló energiahordozók segítségével állítunk elő. Több energiaforrás létezik manapság, ezeket csoportosítani is tudjuk. A nem megújuló energiahordozókat szoktuk úgy is hívni, hogy fosszilis energiahordozók. Ilyen energiahordozók lehetnek például:

- Kőszén,
- földgáz,
- kőolaj.
- propán bután gázok.

Ezen fosszilis energiahordozók legnagyobb előnye, hogy gyorsan és sok energiát lehet belőlük előállítani. Azonban hátrányai például, hogy nem megújuló energiaforrások, azaz a készletük véges. További hátránya, hogy környezet szennyező, hiszen az energia termelése a fosszilis anyagok elégetésével kerül előállításra.

A megújuló energiaforrások a fosszilis energiahordozók ellentéte, azaz készlete eleméletben végtelen, hiszen a természeti adottságokat használhatjuk ki az energia termelésére. Legnagyobb előnyük, hogy környezetkímélőek, hátrányként viszont az a jellemzőjük, hogy adott idő alatt kevesebb energiát tudunk előállítani velük, szemben a nem megújuló energiaforrásokkal.

Napjainkban ilyen megújuló energiaforrások léteznek:

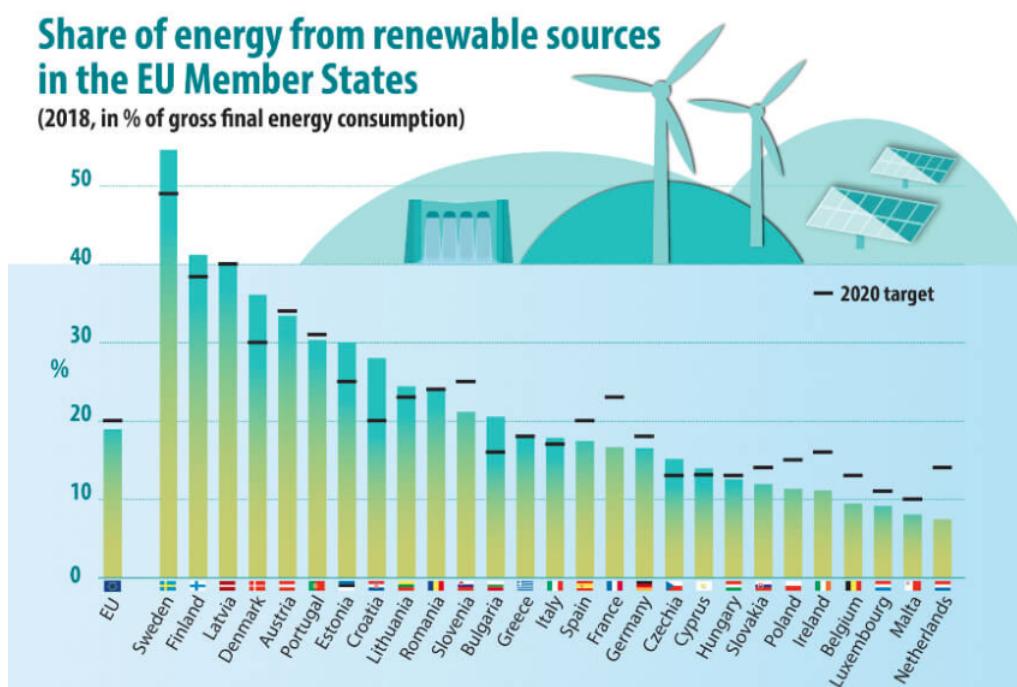
- Napenergia,
- szélenergia,
- vízenergia,

- tengerek hullámzásából kinyerhető energia,
- biomassza

További előnyként szolgál az is, hogy a földet érő 1 órányi napsütés több energiát tud adni, mint amennyit az emberiség 1 év alatt elhasználna.[1]

Modern világunkban nagyon elterjedt a zöld energia hasznosítása, a legtöbbet jelenleg Németország, Nagy-Britannia, Franciaország és Kína költi az alternatív energiára. Az EU-ban 1999-ben még csak 5 százalék volt a megújuló energia aránya a többi energiához képest – mára azonban ez a szám kétjegyűvé vált: 2013-ban már 15 százalék volt, míg 2020-ban elérte a 20 százalékot.[1]

Ezt szemlélteti a következő ábra is:



1.1. ábra. Megújuló energia aránya a többi energiához képest[1]

Terepasztalunk egy szoftver- és hardver elemekkel ötvözött szimulátor lesz, mely egy felhasználó által megadott naptári napból és egy földrajzi helyzetből képes előállítani az aktuális nap tulajdonságait, így a felhasználó átélnéhet egy maketten keresztül, hogy vajon milyen idő lehet az általa választott országban a szintén általa kiválasztott időpontban.

Fontos megjegyezni, hogy a szimulátorunk egy nap leforgását mutatja be kicsiben, napfelkeltétől naplementéig.

A felhasználó végig követheti a terepasztal működését, illetve információkat kaphat az általa kiválasztott országról, hogy vajon elegendő-e abban az országban csak tiszta energiát termelni.

Célunk tehát, hogy egy olyan energetikai rendszert modellezzenk, mely nem fosszilis energiahordozókból áll, csakis megújuló energiaforrások termelnének áramot. A kísérlet, illetve a tanulmány során megtudhatjuk, hogy vajon elegendő-e csak a tiszta energiát hasznosítani, valamint, hogy mennyire függünk a nem megújuló energiaforrásoktól.

1.2. Tudományos Diákköri Konferencia

Terepasztalunk az Interaktív hardware – software-rendszer fejlesztése természettudományos ismeretek népszerűsítése céljából a Varázstoronyban című projektnek egy része, amely a helyi TDK (Tudományos Diákköri Konferencia) projektjeként, illetve az OTDK (Országos Tudományos Diákköri Konferencia) projektjeként valósult meg.

A komplett rendszert egy webalkalmazás zárja egysége, melynek köszönhetően mikrokontrollerek segítségével képesek vagyunk irányítani az eszközöket webes felületen keresztül. A Tudományos Diákköri Konferencián az alábbi eszközök kerültek megvalósításra:

- Cartesius búvár
- Robi a robot
- Terepasztal
- Hell-harang
- Lézer show
- Lejtő

Szakdolgozatunk tehát tartalmazza a terepasztalhoz szükséges dokumentációkat, kódokat és egyéb információkat, illetve a weblap ezen részéhez kapcsolódó tartalmakat, információkat, melyek a TDK -és OTDK keretein belül valósultak meg.

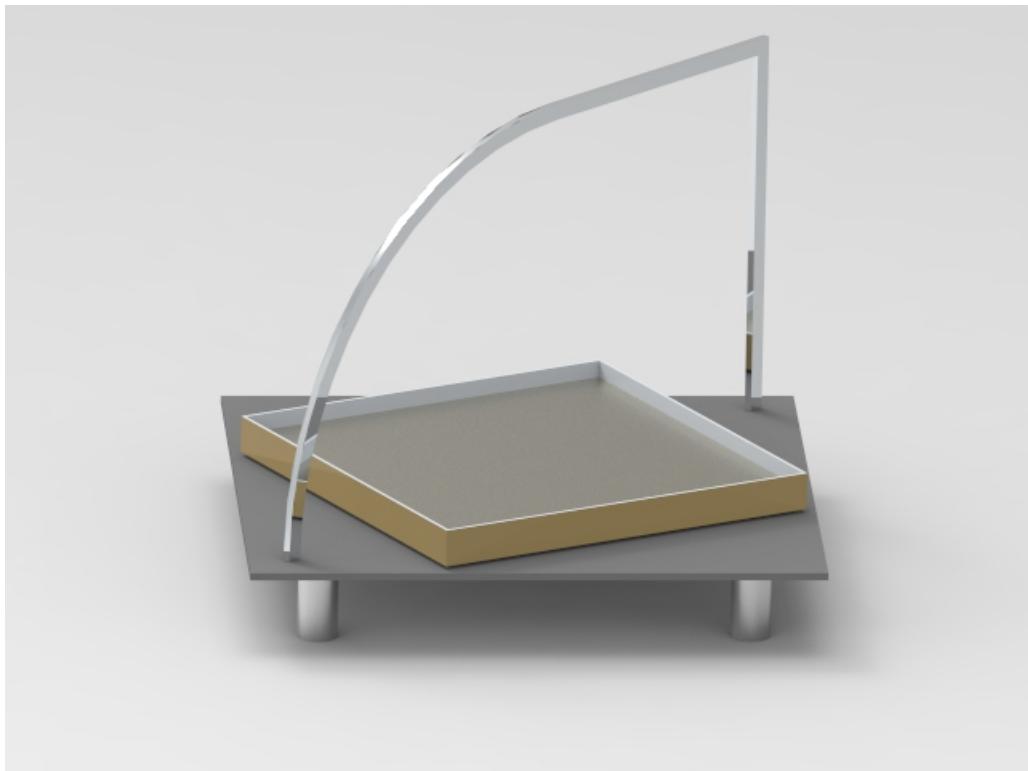
Weblapunk további funkciókkal is el van látva, melyről bővebben a 3-ik fejezetben olvashatunk, illetve a <https://varazstorony.robucation.com>. weboldalon tekinthető meg.

1.3. Terepasztalunk felépítése

Az alkalmazásban lehetőségünk van kiválasztani egy naptári napot, illetve egy szélességi- és egy hosszúsági fokot. Az alkalmazás kiszámolja, hogy napfelkelte és napnyugta idejében a Nap milyen szöget zár be a Föld felszínével, illetve azt is, hogy e két időpont tartományában milyen mozgás történik a két égitest között.

Az alkalmazás a kellő információkat továbbítja a mikrokontrollernek, mely képes elemezni az adatokat, és olyan helyzetbe mozgatja az asztalt, amelyben a két említett égitest is elhelyezkedik.

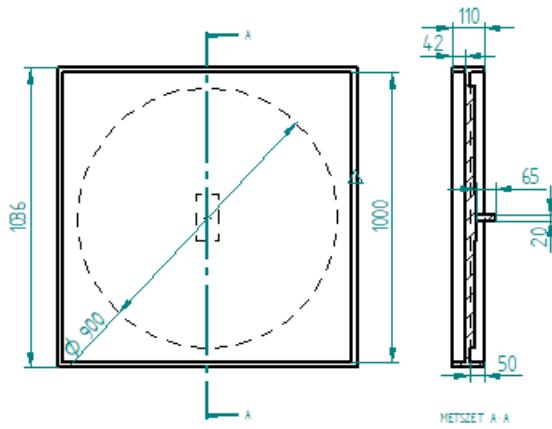
A következő képen a terepasztalunkról tekinthetünk meg egy illusztrációt.



1.2. ábra. Ábra a terepasztalunk felépítéséről

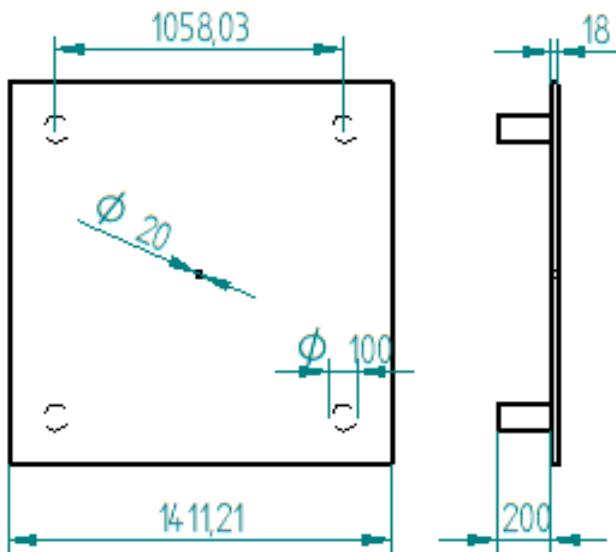
A következő sorokban kitérünk az eszköz részletes felépítésére és rajzokkal szemléltetjük azt.

Az eszköz felépítésénél 18mm falvastagságú bútorlapot használtunk.



1.3. ábra. Rajz a mozgatható elemről

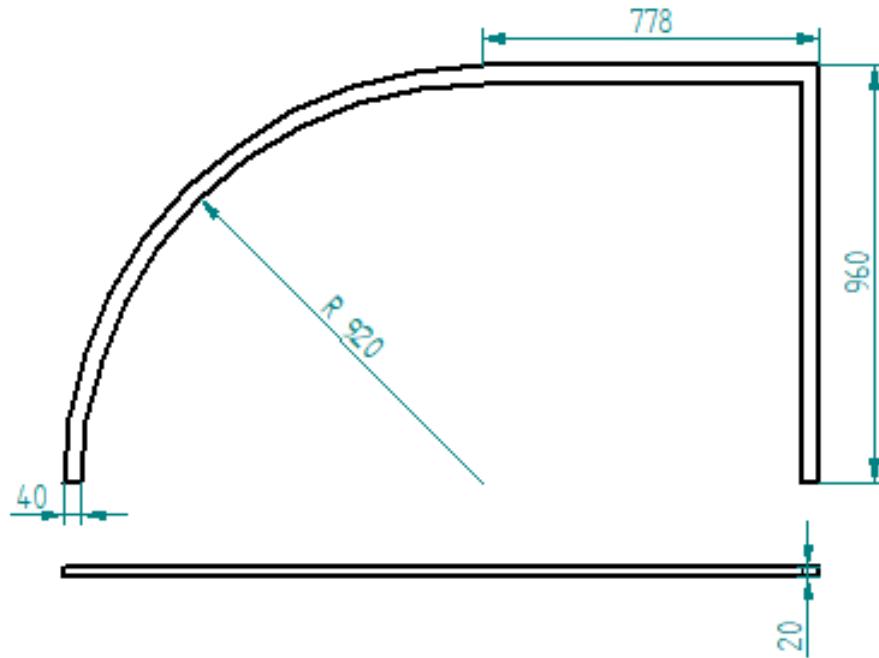
Az 1.3 számú ábrán a mozgatható elem műszaki rajzát tekinthetjük meg. Az alapjául egy 1000mm oldalhosszúságú és 18mm magasságú téglatest szolgál, mely bútorlapból készült el. Annak érdekében, hogy az elkészült domborzat megfelelően illeszkedjen az asztalba, továbbá, hogy az eszköz csatlakozási pontjait és a működtető mechanikát eltakarjuk a felhasználók elől, egy a rajzon feltüntetett méretű peremmel öleltük körbe a téglatestet. Az aljára egy 900mm átmérőjű 18mm falmagasságú hengert rögzítettünk. A henger palástjára bicikli láncot rögzítettünk. Így tudjuk a későbbiekben a forgatható elemet mozgatni. A henger középpontján egy 20mm átmérőjű és 65mm hosszú csap található, mely a későbbi elforduláshoz szükséges központosítást teszi lehetővé.



1.4. ábra. Rajz az alapként szolgáló asztalról

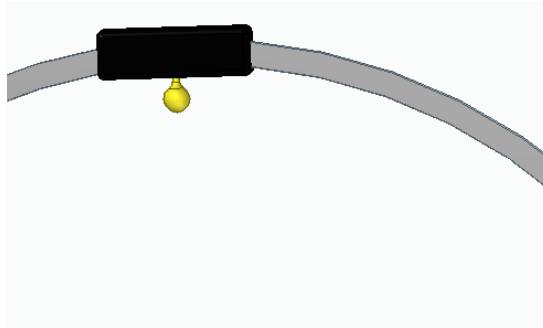
Az 1.4 számú ábrán azt az asztalt tekinthetjük meg, melyhez a forgatható rész, illetve az az állvány csatlakozik, mellyel a nap vertikális helyzetét tudjuk bemutatni.

Az asztal szélessége a forgatható elem átlójának hosszával egyenlő, így a forgatható elem sosem fog lelógni az alapul szolgáló asztalról. Középen találhatunk egy furatot, amelybe perselyt helyezünk. Ebbe a perselybe fog csatlakozni a forgatható lap, csap alkatrésze. Az asztal alátámasztásáért négy darab láb felel, melyek méretei a rajzról leolvashatók.



1.5. ábra. Rajz az állványról, melyen a fényforrás mozog

Az 1.5 számú ábrán egy rajzot láthatunk a fényforrás mozgatásához szükséges állványról. Az állvány egy 40mmX20mm-es zártszelvényből készült. Erre az állványra egy kocsit rögzítünk, mely betudja járni az állvány körívét. A kocsira egy fényforrást helyezünk, mely a napfényt imitálja.



1.6. ábra. Illusztráció a fényforrásról

Terepasztalunk mozgatásához szükséges számításokat kétféle megoldással közelítettük meg. Először matematikai úton kezdtük terepasztalunk mozgatásának problémáját megközelíteni, melyben matematikai képletek, illetve azok implementációjuk segítségével írtuk le a problémát. Ehhez Python programozási nyelvet használtunk, mely

Sass-Gyarmati Norbert feladata volt és a 2.1.1 alfejezetben olvashatunk róla bővebben.

A második megoldás egy más személy által PHP programozási nyelven készített könyvtár felhasználásán alapul. A feladat elkészítése Oraveczi Zsolt feladata volt. A feladatról a 2.1.2 alfejezetben olvashatunk.

2. fejezet

Rendszerünk működése

2.1. A mozgatáshoz szükséges számítások

2.1.1. Napállás kalkulátor

A feladat elkészítéséért **Sass-Gyarmati Norbert** a felelős.

Feladatom ebben a cikkben az volt, hogy matematikai úton megvalósítsak egy szoftvert, mely egy adott napból és egy szélességi, hosszúsági fokból képes legyen kiszámolni az adott nap deklinációs szögét, azaz a Nap szöghelyzetét a szoláris délben. Ezen kívül kitudja számolni a rendszer az adott nap naphosszát, a napfelkelte és naplemente megközelítő értékeit, zenit szöget, illetve abból a napmagassági szöget. Továbbá képes megmondani, hogy a terepasztalunk hány fokos szögben kell elforduljon ahhoz, hogy az arányokat megtartva szimulálhassuk egy ország éghajlati és napszaki adatait. További feladata, hogy képes megmondani, hogy egy fixen telepített déli 40 fokos tájolással rendelkező naprendszer különböző országokban, eltérő napokban milyen szögbe esik, illetve az ebbe eső szögnek milyen termelői hatékonysága van.

Feladatomat Python nyelven írtam, melyhez a PyCharm Community Edition keretrendszer használtam.

Kezdetben meg kellett határoznom egy évszámból, illetve szélességi és hosszúsági fokból a deklinációs, zenit, napmagassági szöget, illetve a napfelkelte, naplementét, ezek kettő adatok alapján pedig a teljes nappalhosszt.

```
import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np
import constant

year = int(input("year:"))
```

```

while year < 0 or year > int(date.today().year):
    year = int(input("year:"))
month = int(input("month:"))
while month < 1 or month > 12:
    month = int(input("month:"))
day = int(input("day:"))
while day < 1 or day > 31:
    day = int(input("day:"))
latitude = int(input("latitude(f) :"))
while latitude < -90 or latitude > 90:
    print("Try again!")
    latitude = int(input("latitude(f) :"))

longitude = int(input("longitude:"))

today = date.today()
dn = date.replace(today,
                  int(year),
                  int(month),
                  int(day)).timetuple().tm_yday

def to_radians(val):
    return (math.pi/180) * val
def Format(value):
    return round(value * 1000000) / 1000000

declaration = 23.45 * math.sin(to_radians(360 / 365 * (dn + 284)))
declaration = Format(declaration)
print("a Nap szogohelyzete a szolaris delben:"
      + str(declaration))

def daylength(dayOfYear, lat):
    latInRad = np.deg2rad(lat)
    declinationOfEarth = 23.45
    *np.sin(np.deg2rad(360.0

```

```

        *(283.0+dayOfYear)
        /365.0))

if -np.tan(latInRad)
    * np.tan(np.deg2rad(declinationOfEarth)) <= -1.0:
return 24.0
elif -np.tan(latInRad)
    * np.tan(np.deg2rad(declinationOfEarth)) >= 1.0:
return 0.0
else:
    hourAngle = np.rad2deg(np.arccos(-np.tan(latInRad)
        * np.tan(np.deg2rad(declinationOfEarth))))
return 2.0*hourAngle/15.0

length_of_the_day = daylength(dn, latitude)
print("nappal\u00f3hossza:" + str(int(length_of_the_day)) + "h")

zenith_angle = -math.acos(math.sin(latitude)
    *math.sin(declination)
    +math.cos(latitude)
    *math.cos(declination)
    *math.cos(length_of_the_day))+2
    *math.pi

print("zenit\u00e1szog:" + str(round(zenith_angle, 2)))
print("Napmagassagszoge:" +
    + str(round(90 - zenith_angle, 2)))
print("napfelkelte:" + str(hour_value) + ":" +
    + str(abs(minute_value)))
print("naplemente:" + str(set.hour-1) + ":" +
    + str(set.minute))

```

A program megértése érdekében szükségünk van a definíciókra. Az alábbi definíciók teljes egészében megtalálhatók Dr. Barótfi István: Környezettechnika című könyvében.

2.1. Definíció. Deklináció: a Nap szöghelyzete a szoláris délben (azaz ha a Nap a helyi délkörön van) az egyenlítő síkjához viszonyítva. A Föld a Nap körül ellipszis pályán kering, miközben maga a Föld is forog saját tengelye körül. A földpálya síkja és az Egyenlítő által meghatározott sík egymással szöget zár be, azaz a Föld forgásának a tengelye szöget zár be a földpálya síkjára állított merőlegessel. Értéke a napközeli és a naptávoli pontban 23,5, a tavaszi és az őszi napéjegyenlőség idején zérus. Északon pozitív. $-23,5 \leq \text{Deklináció} \leq 23,5$. A deklinációs szög értelmezését a 3.1-es ábra

mutatja. [2]

2.2. Definíció. Nappalhossza: meghatározható a napkelte óraszögéből. Mivel a napkeltétől a delelésig ugyanannyi idő telik el, mint a deleléstől napnyugtáig, a nappal hossza 2ws lesz, ezt elosztva 15 fokkal megkapjuk a nappal hosszát órában.

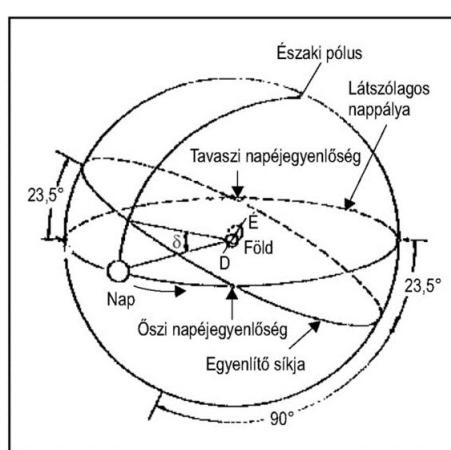
$$Nd = (2/15)ws$$

[2]

2.3. Definíció. Zenitszög: a függőleges és a Naphoz húzott egyenes által bezárt szög, azaz a vízszintes felületre érkező sugárzás beesési szöge. Adott időben a megfigyelőnek a Földön meghatározható a pozíciója, ezt nevezzük a megfigyelő zenitjének. Ez a pont metszéspontja a megfigyelő helyének földfelszíni normálisának és az égi mezőnek. Az ezzel a ponttal ellentétes helyen lévő zenitet nevezzük nadírnak. A megfigyelő horizontja egy nagy kör (az égi mezőben), egy olyan sík, amely átmegy a Föld középpontján és amelynek határát a zenit és a Föld normálisának metszővonala jelenti. A zenit szög az a szög, amely a lokális zenit, valamint a Nap és a megfigyelő által meghatározott egyenes egymással bezár. Ez a szög 0 és 90 fok között változhat. [2]

2.4. Definíció. Napmagasság szöge: a Napnak szögben kifejezett magassága a megfigyelő horizontjából, azaz a vízszintes és a Naphoz húzott egyenes által bezárt szög. Értéke 0 és 90 fok között van. A napmagasság szöge komplementere a zenit szögnek.

[2]



2.1. ábra. A deklinációs szög értelmezése

Ezek definíciók alapján sikerült egy szélességi és hosszúsági fok, illetve egy a felhasználó által kiválasztott dátum megadásával kiszámolni az adott ország deklinációs, zenit, Napmagassági szög, illetve nappalhosszát kiszámolni.

A további adatok kiszámításához (napfelkelte, naplemente, asztal elfordulása) ezek adatok elengedhetetlenek.

Napfelkelte, naplemente kiszámítása

A napfelkelte és napnyugta meghatározása a pontos földrajzi koordináták (hosszúsági és szélességi fok) valamint a dátum alapján történik.[3]

Kiszámításához egy metódust hoztam létre, melynek a „calcsunriseandsunset” nevet adtam. Ez a metódus egy dátumot vár a felhasználótól, illetve a rendszer felhasználja az általa eltárolt latitude és longitude fokokat. A dátumból ezután egy julián dátumot készítünk, mely az időszámításunk kezdete előtt 4713. január 1. déli 12 órától eltelt napok számát jelenti. A csillagászati észlelések időpontját a Julián dátummal adjuk meg.[4]

Ezután csinálunk egy julián csillagot, melyhez már a hosszúsági fokot is felhasználjuk. Matematikai képletek segítségével pedig ezek segítségével kiszámoljuk a napfelkelte és naplemente értékét

```
import math
from datetime import date, timedelta, datetime, time, tzinfo
import numpy as np

def calcsunriseandsunset(date_time):
    a = math.floor((14 - date_time.month) / 12)
    y = date_time.year + 4800 - a
    m = date_time.month + 12 * a - 3
    julian_date = date_time.day + math.floor((153 * m + 2) / 5) +
    365 * y + math.floor(y / 4) - math.floor(
        y / 100) + math.floor(y / 400) - 32045

    nstar = (julian_date - 2451545.0 - 0.0009) - (longitude / 360)
    rounded_value = round(nstar)
    julian_star = 2451545.0 + 0.0009 + (longitude / 360) +
    rounded_value
    M = (357.5291 + 0.98560028 * (julian_star - 2451545)) %
    360
    c = (1.9148 * sinus_to_radian(M)) + (0.0200 *
                                              sinus_to_radian(2 * M)) +
    (0.0003 * sinus_to_radian(3 * M))
    l = (M + 102.9372 + c + 180) % 360
    julian_transit = julian_star + (0.0053 * sinus_to_radian(M)) -
    (0.0069 * sinus_to_radian(2 * l))
    delta = math.asin(sinus_to_radian(l) *
```

```

sinus_to_radian(23.45)) *
180 / math.pi
H = math.acos(
    (sinus_to_radian(-0.83) - sinus_to_radian(latitude) *
     sinus_to_radian(delta)) / (cosinus_to_radian(latitude) *
                                   cosinus_to_radian(delta))) *
180 / math.pi
julian_star = 2451545.0 + 0.0009 +
((H + longitude) / 360) +
rounded_value
julian_sunset = julian_star +
(0.0053 * sinus_to_radian(M)) -
(0.0069 * sinus_to_radian(2 * l))
julian_sunrise = julian_transit -
(julian_sunset - julian_transit)
return (calculate_time_from_julian_date(julian_sunrise),
        calculate_time_from_julian_date(julian_sunset))

```

Asztal forgatása

Az asztal forgatásához a deklinációs[2] szöget vettet segítségül. A deklinációs szög megmondja, hogy egy adott napban egy országban a Nap milyen szögben helyezkedik el szoláris délben, így képes minden ország beállításával is megmondani ezt a bizonyos szöget. Ehhez viszonyítottam az asztal eltolását, hiszen egy Magyarországhoz északibb ország különböző hónapokban más viselkedést produkálnak, például nyáron hegyesebb, télen tompább szög. Ugyanez igaz a nálunk délibb országokhoz is, csak ott ezek negáltja történik.

Ez a bizonyos szög országon belül is havonta változik, minden nap maximum 0,5 fokkal. Az asztal tehát a deklinációs szög negáltjával fog változni, hiszen az arányokat megtartva az északi pontot helyezzük a megváltozott érték helyére.

```

import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np

move_desk = 0
optimised_value = 0
def move(self):

```

```

if self > 0:
    move_desk = self * -1
    print ("Az asztalt " + str(round(move_desk, 2)) +
            " fokkal elkelltolni balra!")
elif self == 0:
    print ("Az asztalt nem kell eltolni")
else:
    move_desk = self * -1
    print ("Az asztalt " + str(round(move_desk, 2)) +
            " fokkal elkelltolni jobbra")

```

A napcellák elhelyezkedése a Naphoz képest, illetve a pontos szög meghatározása ugyanebben a metódusban definiált. Az égtájak konstans értékként vannak definiálva, melyben az értékek fokban értendők. A napcellák elhelyezkedése tehát a Dél - deklináció. A napcellák hatékonyságáról[5] tanulmányok szerint a déli 35-40 fok az optimális, így a rendszert is úgy állítottuk be, hogy az magyarországi déli 40 fokban legyen.

```

import math
from datetime import date, timedelta, datetime, time, tzinfo
import datetime as datetime
import numpy as np

```

```

def move(self):
    solar_dist = constant.SOUTH - declination
    if solar_dist < constant.SOUTH and
        solar_dist > constant.EAST:
            optimised_value = 90
            print ("delkeleten helyezkednek el hatekonysag: "
                    + str(optimised_value) + "%")
            print ("Szoge: " + str(round(solar_dist, 2)))
    elif solar_dist < constant.EAST and
        solar_dist > constant.NORTH:
            optimised_value = 75
            print ("eszakkeleten helyezkednek el, hatekonysag: "
                    + str(optimised_value) + "%")
            print ("Szoge: " + str(round(solar_dist, 2)))
    elif solar_dist > constant.SOUTH and

```

```

solar_dist < constant.WEST:
    optimised_value = 90
    print("delnyugaton\u00e9 helyezkednek el\u00e9 hatekonysag:\u00e9"
          + str(optimised_value) + "%")
    print("Szoge:\u00e9" + str(round(solar_dist, 2)))

else:
    optimised_value = 75
    print("eszaknyugaton\u00e9 helyezkednek el\u00e9 hatekonysag:\u00e9"
          + str(optimised_value) + "%")
    print("Szoge:\u00e9" + str(round(solar_dist, 2)))

```

Ahogy a kódrészletből is látszik, mivel a telepített napcellák tájolástól és napszaktól függetlenül déli 40 fokban helyezkednek el, így az optimalizálás is csak a 40 fokos szöghöz igazodik. Optimalizálásnál az alábbi 5 esetek különböztetjük meg:

- Ha északkeletra esik: Azaz a szögtartomány így írható le: $0 < \text{szög} < 90$. Ilyen esetben a hatékonyságuk 75%-os
- Ha délkeletra esik: Azaz a szögtartomány így írható le: $90 < \text{szög} < 180$. Ilyen esetben a hatékonyságuk 90%-os
- Ha délrre esik: Azaz a szögtartomány pontosan 180 fokos. Ebben az esetben a hatékonyságuk 100%-os
- Ha délnyugatra esik: Azaz a szögtartomány így írható le: $180 < \text{szög} < 270$. Ilyen esetben a hatékonyságuk 90%-os
- Ha északnyugatra esik: Azaz ha egyik állításra sem igaz a feltétel. Ebben az esetben a hatékonyságuk 75%-os

2.1.2. Napállás meghatározása PHP nyelven

A feladat elkészítéséért Oravecz Zsolt a felelős.

Kutatótársammal szerettünk volna minél több megoldást bemutatni egy adott problémával kapcsolatban. Ennek egyik lehetséges formája az API-n keresztül történő kommunikáció. Tekintsük meg mit jelent az API kifejezés.

Az API az angol „application programming interface” rövidítése, lefordítva alkalmazásprogramozási interfész.[6] Egy köztes réteget képez a szoftverfejlesztő és egy korábban létrehozott szoftver között. Segítségével a fejlesztőnek nem kell tisztában lennie a program felépítésével, és működésével, csupán egy előre definiált kérést kell küldeni

az API felé, ō pedig előállítja a megfelelő utasítást. Ez a módszer rengeteg időt spórol meg a fejlesztőnek, hiszen így elég a saját fejlesztésére fókuszálnia.

Ez a fajta eljárás azonban egy újabb probléma lehetőséget vet fel. Ha a szerver, ahonnan az információkat lekérjük számunkra nem elérhető, akkor a felhasználók számára mi sem tudunk információt nyújtani.

Ezért döntöttem amellett, hogy egy nyílt forráskódú, PHP nyelven íródott könyvtárat importálok be a projektünkbe, amelyet aztán testre szabtam, és így számoljuk ki a megfelelő adatokat, amelyeket később továbbítunk a mikrokontroller felé. Az kód megtekinthető a forrásban található linken keresztül.[7]

```
// Az osztaly inicializalása datummal és koordinatakkal.  
// A mai nappal, Parízs koordinataival  
$sc = new AurorasLive\SunCalc(new DateTime(), 48.85, 2.35);  
  
// A napfelkelte idopontjának konvertálása ora:perc formáumba.  
$sunTimes = $sc->getSunTimes();  
$sunriseStr = $sunTimes['sunrise']->format('H:i');  
  
// A Nap pozícionák kiolvasása, a mai napi napfelkelte idején  
$sunrisePos = $sc->getPosition($sunTimes['sunrise']);  
  
// Fokba történő atkonvertálás  
$sunriseAzimuth = $sunrisePos->azimuth * 180 / M_PI;
```

A felhasználó a böngészőben egy felugró ablak segítségével kiválaszthat egy dátumot, illetve kiválaszthat az általunk felsorolt földrajzi helyekből egyet, és az alkalmazást elindíthatja. Amikor az „Indít” gombra kattint, a program kiértékeli a felhasználó által kiválasztott tulajdonságokat, majd összeállítja egy paraméter listába, amit aztán átad egy konstruktornak. A létrejött osztály megfelelő metódusai visszaadnak egy-egy adatot, melyet a mikrokontroller felé továbbítunk.

2.2. Szimulációk

Ebben a fejezetben olvashatjuk a projekt kisebb részeire osztott eszközökhöz elkészített szimulációkat, mely azt a célt szolgálják, hogy az általunk elkészített modellek, illetve prototípusok működését szemléthesse.

2.2.1. Fogyasztó probléma szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Ahhoz, hogy a fogyasztók energiaigényeket feltudjuk mérni, szükségünk volt egy szimulátor programra, melyben megtudjuk adni, hogy mennyi fogyasztó fogja használni a terepasztalunkat. A fogyasztók az alábbiak lehetnek:

- Családi házak,
- lakások,
- iskolák,
- kórházak.

A fogyasztók különböző energiaigényekkel rendelkeznek. Egy átlag családi háznak (4 fős) nagyjából 230 kWh energiára van szüksége egy hónapban. Kutatásaink során további statisztikát tudtunk levonni, amiben kiderült, hogy egy lakásnak, vagy bérháznak (4 fős) nagyjából 200 kWh energiára van szüksége[8]. Az iskolákat, illetve kórházakat más matematikai műveletekkel lehet megadni.

Először szükségünk volt olyan adatokra, hogy átlagosan mennyi energiát fogyasz tanak ezek a különféle intézmények. A méréseket egy 2001-es tanulmány alapján készítettem, így minden érték csak egy megközelítő értéket ad. Mérések alapján kiderült, hogy egy átlagos iskolának nagyjából $20 \text{ kWh}/m^2$ energiaigényre van szüksége, míg egy kórháznak jelentősen nagyobb, körülbelül $100 \text{ kWh}/m^2$ energiaigényre van szüksége[9].

Ezek után szükségem volt egy szabványra mely kimondja, hogy egy átlagos iskola, illetve kórház milyen szabályoknak kell megfeleljenek. A kutatásaimat felhasználva egy iskola hivatalos szabványa, hogy $2,5m^2$ jut egy diák számára[10]. Tehát egy iskola kalkulálása ezek szorzatából tevődik össze. Továbbá kutatásaim arra is rámutattak, hogy egy kórházban $6-8m^2$ jut egy beteg részére. Így az iskola, illetve a kórház mérete és ebből kiszámítva a fogyasztása nagyban függ az intézményekben tartózkodó tanulók, vagy betegektől.

Szimulátor programom Python nyelven írtam a PyCharm¹ segítségével. A szimulátorban a felhasználó konzolos ablakon keresztül megadhatja a szoftvernek a házak,

¹ A PyCharm a Python legnépszerűbb IDE-je, és olyan nagyszerű szolgáltatásokat tartalmaz, mint a kiváló kód kitöltése és ellenőrzése a fejlett hibakeresővel, valamint a webprogramozás és a különféle keretrendszerök támogatása.[11].

lakások, iskolák, kórházak, tanulók, illetve betegek számát, majd ezek adatokból képes kiszámolni az átlagos fogyasztási igényt, valamint ugyanezt az adatot lebontja napra pontosan.

Szoftveremben minden intézmény egy külön osztály, melyek más-más számításokat kell végezzenek az igény kiszámításához.

```
class House:  
    def __init__(self):  
        self.consume = 230  
        self.daily = self.consume / 30  
  
class FlatHouse:  
    def __init__(self):  
        self.consume = 200  
        self.daily = self.consume / 30  
  
Ahogy a kód részletből is látható, egy családi ház konstruktorának fogyasztási értéke 230, míg egy lakás átlagos fogyasztási értéke 200. Napra lebontva pedig egy átlagos naphosszt választottam, így minden értéket 30-cal oszt el, így tudjuk lebontani napi szintre az értékeket.  
Az iskolák és kórházak összetettebb konstruktorral rendelkeznek, melyek nagyban függnek a paraméterként megadott tanulók, illetve betegektől.  
class School:  
    def __init__(self, students):  
        self.students = students  
        self.square_per_consume = 20  
        self.school_size = 2.5 * self.students  
        self.consume = self.square_per_consume * self.school_size  
        self.daily = self.consume / 30  
  
class Hospital:  
    def __init__(self, patient):  
        self.patient = patient  
        self.square_per_consume = 100  
        self.hospital_size = 6 * self.patient  
        self.consume = self.square_per_consume * self.hospital_size  
        self.daily = self.consume / 30
```

Ahogy a kódrészletből tisztán látható, az iskola és kórház konstruktori már egy tanuló, illetve beteg paramétert is várnak, melyből kitudja számolni az intézmény átlagos négyzetméterét, illetve ebből az adatból képes kiszámolni a négyzetméterre jutó energia igényüket.

A különféle fogyasztók értékeit listában tárolom, illetve ciklusok segítségével tudom feltölteni. A felhasználó a program lefutása után konzolból megadhatja a különböző értékeket, mely során a szoftver egyes osztályait meghívva, beállítja a számára szükséges értékekkel a paramétereit, majd közli a felhasználóval az alábbi adatokat:

- Házak száma
- Lakások száma
- Iskolák száma
- Kórházak száma
- Összes fogyasztó igénye havi, illetve napi szinten

```
from calculate_estates import School , Hospital
from consumers import House , FlatHouse

if __name__ == '__main__':
    for x in range(house_piece):
        houses.append(House())
        total_consume += houses[x].consume
        total_daily_consume += houses[x].daily

    for x in range(flat_house_piece):
        flat_houses.append(FlatHouse())
        total_consume += flat_houses[x].consume
        total_daily_consume += flat_houses[x].daily

    for x in range(school_piece):
        school.append(School(student))
        total_consume += school[x].consume
        total_daily_consume += school[x].daily

    for x in range(hospital_piece):
        hospital.append(Hospital(patient))
        total_consume += hospital[x].consume
```

```

total_daily_consume += hospital[x].daily

print("hazak szama:", len(houses))
print("lakasok szama:", len(flat_houses))
print("iskolak szama:", len(school))
print("korhazak szama:", len(hospital))
print("osszes fogyasztogenye havi szinten:"
      + str(total_consume) + " kWh")
print("osszes fogyasztogenye napi szinten:"
      + str(round(total_daily_consume, 2)) + " kWh")

```

2.2.2. Vízerőmű szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

A vízerőmű rendszerünk, mely szivattyús tárolós elven működik, egy szimulációon keresztül is tudjuk szemléltetni. A szimuláció a folyamatot képes szemléltetni, a vízerőmű működését, illetve a leállítás- újraindítás folyamatát.

A szimuláció elkészítéséhez Python programozási nyelvet használtam.

```
from time import sleep
```

```

class WaterThread:
    def __init__(self):
        self.buffer_1 = 0
        self.buffer_2 = 100
        self.pump = False
        self.is_run = True

    def buffer_1_fill(self):
        if self.buffer_2 == 100:
            self.pump = True
            self.fill_buffer()

    def fill_buffer(self):
        while self.buffer_1 != 100 and self.is_run:
            self.buffer_1 += 1

```

```

        self.buffer_2 -= 1
        print("buffer1:\u2022" + str(self.buffer_1)
              + "\u2022buffer2:\u2022" + str(self.buffer_2))
        sleep(0.4)
    self.pump = False

def fill_buffer_2(self):
    while self.buffer_2 != 100 and self.is_run:
        self.buffer_2 += 1
        self.buffer_1 -= 1
        print("buffer1:\u2022" + str(self.buffer_1)
              + "\u2022buffer2:\u2022" + str(self.buffer_2))
        sleep(0.4)
    self.pump = False

def buffer_2_fill(self):
    if self.buffer_1 == 100:
        self.pump = True
        self.fill_buffer_2()

def buffer_loop(self):
    while True:
        self.buffer_1_fill()
        self.buffer_2_fill()

def stop_thread(self):
    while True:
        s = input()
        if s == "s":
            self.is_run = False
            while self.buffer_2 != 100:
                self.buffer_2 += 1
                self.buffer_1 -= 1
                print("buffer1:\u2022" + str(self.buffer_1)
                      + "\u2022buffer2:\u2022" + str(self.buffer_2))
                sleep(0.4)
        elif s == "o":
            self.is_run = True

```

```

import threading

from water import WaterThread

if __name__ == '__main__':
    thread = WaterThread()

    t1 = threading.Thread(target=thread.buffer_loop)
    t1.start()
    t2 = threading.Thread(target=thread.stop_thread)
    t2.start()

```

Ahogy a kód részletből is látható, maga a szoftver két külön szálban fut, egyik szál foglalkozik az erőmű működésével, míg a másik szál a felhasználók számára van, melyben konzolba beírva megadhatja az erőmű leállítását, illetve újraindításhoz szükséges parancsot. A rendszer egy körfolyamatban működik, mely ciklikusan ismételi önmagát, így ha a felhasználó nem állítja le a rendszert, örök körfolyamatban fog részt venni.

A szálak egy közös osztályban vannak definiálva, melyben az osztály egyes metodusai végzik el a szálkezelést. A körfolyamat egyszerű, két tartály van, az első feljebb van, míg a második közel az elsőhöz. Legyenek elnevezve a tartályok, A: első tartály, B: második tartály. Kezdetben a B tartály tele van, amiben van egy szivattyú, mely elkezdi az A tartályba szivattyúzni a vizet. Ez a folyamat addig megy, míg az A tartály teljesen tele nem lesz. Fontos továbbá megjegyezni, hogy ilyen esetben nem feltétlenül lesz üres a B tartály. Az erőmű jó szemléltetése és átláthatósága érdekében a programban ilyen esetben a B teljesen kiürül. Amint az A tartály tele lesz, megnyitja a csapot és visszafolyik a B tartályba a víz. A csapnál van a generátor, mely vízfolyamból elektromos áramot képes termelni. Amint az A tartály a teljes vízkészletet visszaadta a B tartálynak, a körfolyamat újraindul.

A felhasználónak lehetősége van ezt a folyamatot befolyásolni. Két opció közül választhat:

- leállít: ez esetben az inputszalag egy „s” karaktert vár a felhasználótól. Az „s” gomb lenyomása után a szivattyú leáll, a csap kinyit, majd az A tartályban levő összes víz visszafolyik a B tartályba.
- újraindít: amennyiben a folyamat leállt, a felhasználónak lehetősége van újraindítani, ez esetben az inputszalag egy „o” karaktert vár a felhasználótól. Az „o” gomb lenyomása után a szivattyú újra elindul, a csap lezárt, így a B tartályból újra elindul a víz az A tartály felé.

2.2.3. Napcellák teljesítményének szimulálása

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

A napcellák aktuális termelési mértékének szimulálására szükségünk volt egy prototípus programra, mely képes egy opcionálisan megadott napcella termelési értékeit kiszámolni. A szimuláció megmondja az Arduinohoz csatolt napcella aktuális termelésének feszültségét, illetve ezt átváltva kilowattba. A feszültségről teljesítményre való átváltáshoz szükségünk volt még egyéb számításra is. A képlet tehát:

$$P_{(kW)} = V_{(V)} * I_{(A)} / 1000$$

Az ampert konstans értékből kapjuk, mely az aktuális napcella maximális áramerősségeből jön. Példaprogramomban egy 6V 4.5W 520mAh-ás napcellát alkalmaztam, így a mA-t Ampperre átszámítva 0,52 értéket használtam. Az így kapott érték még csak Wattban adható meg, így el kell osztani 1000-el, hogy kW értéket kaphassunk.

Szimulációm szemléltetéséhez C++ nyelvet használtam, mely közvetlen kommunikál az Arduino Uno mikrokontrollerrel az Arduino szoftveren keresztül. Az Arduino Uno-ról részletesebben a 5.4.1-ik fejezetben olvashatunk. A szimulátor program az inputról bemenő jelből konvertál egy feszültség értéket, mely a következő képlettel számítható:

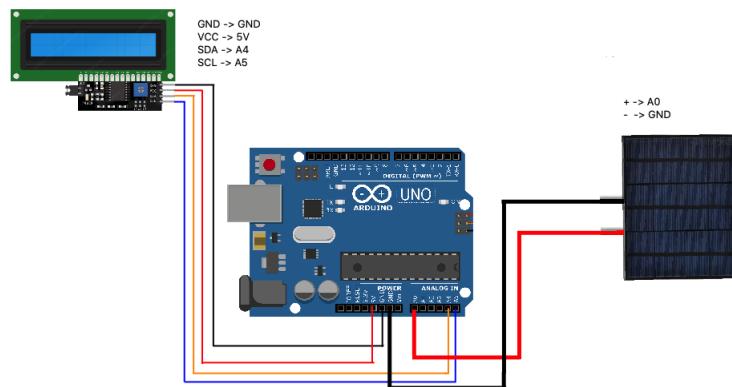
$$\text{Feszültség} = (\text{input}/1024) * \text{AP};$$

ahol

- input: a bemenő jelből származó adat
- AP: maximális feszültség érték, melyet képes az analóg port olvasni

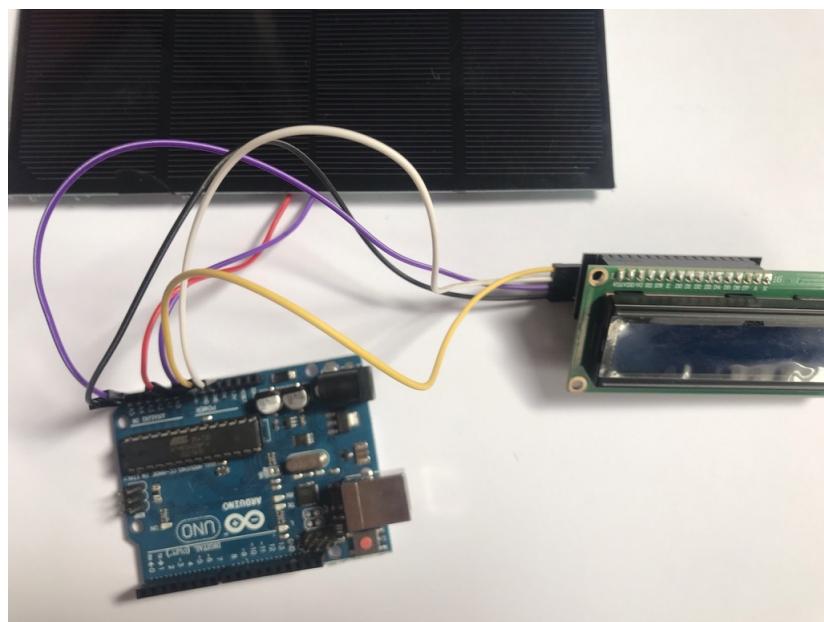
Minden robotikai projektet egy kapcsolási rajz előz meg. A kapcsolási rajz a projekt készített eszközök kapcsolásait tartalmazza, melyben előre definiáljuk azokat a portokat, melyeket a hardver összeállítása után használni fogunk. Továbbá azért nagyon fontos minden építés előtt kapcsolási rajzot készíteni, mert ez alapján virtuálisan is ki lehet próbálni, ezzel hibákat tudunk megelőzni.

A kapcsolási rajz tehát:



2.2. ábra. Kapcsolási rajz a hardver összeállításáról

A kapcsolási rajz összeállítása után már képesek vagyunk összeállítani a hardvert.



2.3. ábra. hardverek összeállítása a kapcsolási rajz alapján

A hardver összeállítása után a következő lépés a kód összeállítása. A kód megkezdése előtt fontos, hogy a kapcsolási rajzon definiált portokat használjuk a szoftveren is.

A kód tehát:

```
#include <LiquidCrystal_I2C.h>

#include <Wire.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

const int inPin = 0;
void setup()
{

```

```

lcd.begin(16,2);

lcd.backlight();
lcd.setCursor(1,0);
lcd.print("Voltage: ");
lcd.setCursor(0,1);
lcd.print("kW: ");

}

void loop()
{
    lcd.display();
    int value = analogRead(inPin);

    float volts = (value / 1024.0) * 5;
    float amper = 0.2;
    float kW = (volts * amper) / 1000;

    lcd.setCursor(10,0);
    lcd.print(volts);
    lcd.print("V");
    lcd.setCursor(6,1);
    lcd.print(kW, 6);
    lcd.print("kW");
    delay(500);
}

```

A kódrészletben látható, hogy a napcellánk pozitív tartományát az Arduino Uno A0-ás portjához kapcsoltam, melyet az *inPin* változóba deklaráltam. Az A0 egy analóg port, melyből a 0 sorszámú portot választottam ki. Továbbá az is látható, hogy később ennek a bemenetnek kiolvasott értékét eltárolom egy *value* változóba, mely után a képletbe behelyettesítve kitudom számolni az aktuális termelési értéket voltban mérve. Az A érték jelen esetben 5 lesz, hiszen 5 volt a maximális érték, melyet képes az analóg port fogadni.

A kilowatt kiszámításához be kellett vezetni egy konstans értéket, mely az aktuális napcella maximális teljesítményéből fakad. Ez a kísérletben 0,52 amper, hiszen a

napcella 520mAh-t képes leadni.

A kísérletek után a felhasználóval közölni kellett az adatokat, melyhez egy 16x2-es LCD kijelzőt használtam. A kijelzőn megjelenik az aktuális feszültség, illetve ezt átszámítva a teljesítmény értéke.

2.2.4. Encoder működése

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Vízerőműünk modellezéséhez, illetve szimulálásához elengedhetetlen a vízturbina teljes forgásának kiszámítása egy adott időintervallum alatt. A forgási érték alapján lehet meghatározni, hogy a víz percenként hányssor forgatja meg a vízturbinát, illetve ezek forgatások során mennyi energia keletkezik.

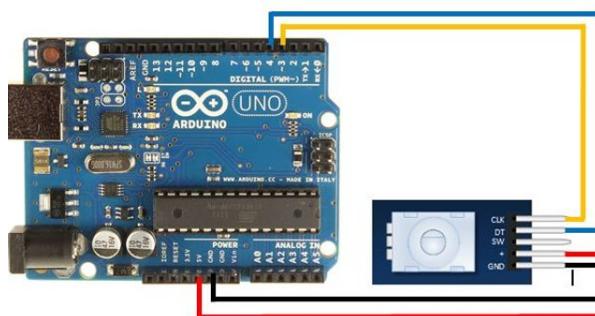
Ehhez kellett nekünk az encoder, mely egy szenzor. De mik is pontosan a szenzorok? A szenzor egy olyan eszköz, amely egy fizikai mennyiséget (ilyenek lehetnek például a hőmérséklet, távolság, valamint a nyomás) a vezérlés, illetve szabályozástechnikában jobban felhasználható, jobban kiértékelhető jellé képes átalakítani. (azaz elektromos jellé, pneumatikus jellé)[12]

Az encoder képes megmondani a forgatási mozgáshoz tartozó elváltozásokat, így ezért használtam kísérletem során ezt az eszközt.

Szimulációm egy Arduino Uno-hoz csatolt encoder működését mutatja be, melyről leolvashatjuk, hogy hányszor fordul egy teljes kör az encoder.

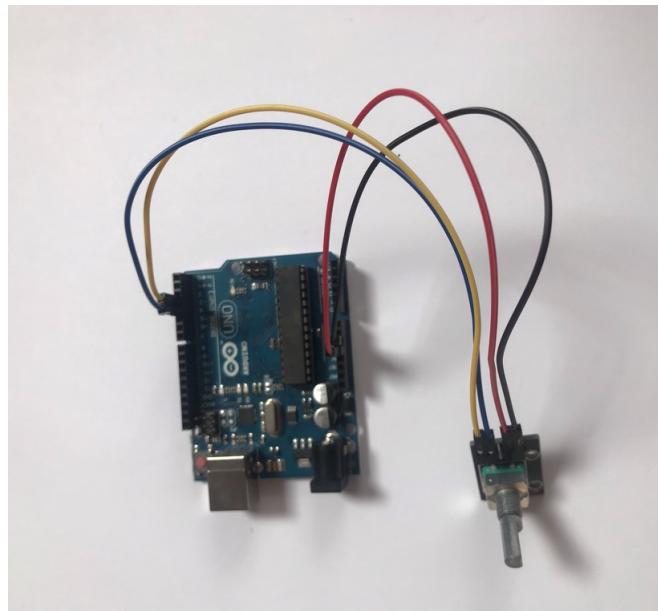
Szimulációm szemléltetéséhez C++ nyelvet használtam, mely közvetlen kommunikál az Arduino Uno mikrokontrollerrel az Arduino saját szoftverén keresztül.

A kapcsolási rajz tehát:



2.4. ábra. Kapcsolási rajz az encoder összeállításáról

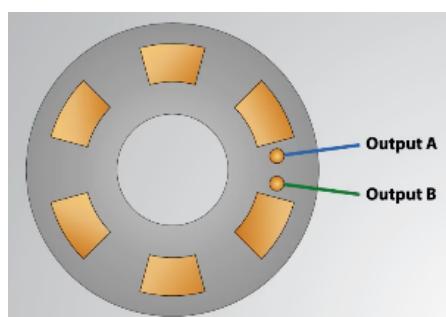
Kapcsolási rajz elkészítése után összeállíthatjuk a hardvereket.



2.5. ábra. Az encoder összeállítása kapcsolási rajz alapján

Ahogy a képen is látható, a szimuláció előállításához nincs másra szükségünk, mint egy mikrokontrollerre, illetve az Arduino Encoderre és az ezeket összekötő jumper kábelekre². Mivel a szimulációm nem lcd kijelzőn, hanem soros monitoron szemléltetem, így az lcd panelt és a hozzá szükséges i2c protokollt nem szükséges hozzá csatolni.

Mielőtt hozzákezdünk az encoder működésének szimulálásához, fontos tudnunk annak működési elvét. Egy arduino encoder két kimenettel rendelkezik, melyeket most jelöljük Output A és Output B-vel. A kimenetek két értéket vehetnek fel, 0 és 1, mely a feszültség szintjét jelenti. 0 az alacsony (LOW), 1 pedig a magas (HIGH) feszültségszint. Az elhelyezkedésüket a következő kép szemlélteti:



2.6. ábra. Az encoder kimenő adatainak szemléltetése

Az ábra alapján tehát leolvashatjuk, hogy ha órajárással megegyezően forgatjuk az encodert, akkor ellentétes értékekkel szolgál a két kimenet, hiszen az A kimenet hamarabb kap magas feszültségi értéket. Órajárással ellentélesen forgatva viszont azo-

² A jumper kábel egy elektromos vezeték, melynek minden vége csatlakozóval, vagy pedig tűvel van ellátva

nos értékűek lesznek, hiszen az A kimenet visszatér alacsony szintre, így kezdetben minden kettő alacsony feszültségi értékkel kezd.

Ezek alapján tehát meghatározható az encoderek forgatási iránya, melyben minden két irányban képes számításokat végezni. A kódolási folyamatban tehát megkülönböztethetjük a forgatási irányokat és ezeknek megfelelően képesek vagyunk elvégezni a számításokat.

Először szükségünk van a megfelelő portokat deklarálni, mellyel lefoglaljuk az Arduino Uno mikrokontrolleren található portok számát és értékét. A kód tehát:

```
int pinA = 3;
int pinB = 4;
int count = 0;
int totaldegree = 0;
int lastPin;
int aValue;
boolean operate;

void setup()
{
    pinMode (pinA ,INPUT);
    pinMode (pinB ,INPUT);
    lastPin = digitalRead(pinA );
    Serial.begin (9600);
    Serial.println( "BEGIN" );
    Serial.println ();
}

void loop()
{
    aValue = digitalRead(pinA );
    if (aValue != lastPin)
    {
        if (digitalRead(pinB) != aValue) //We're Rotating Clockwise
        {
            count++;
            operate = true;
        }
    }
}

else
```

```

{
    operate = false;
    count--;
}

if (count % 40 == 0)
{
    totaldegree += 1;
    Serial.print("Encoder Count: ");
    Serial.println(totaldegree);
    Serial.println();
}
lastPin = aValue;
}

```

A kódrészletben jól látható, hogy az Arduino UNO mikrokontrolleren a digitális 3-ik, illetve 4-ik portot foglaltam le, továbbá létrehoztam egy számláló változót, melyben az encoder forgási értékét számolja, egy „totaldegree” változót, mely a 360°-os teljes forgásokat számolja, egy „lastPin” változót, mely az alacsony - magas feszültségi értékek utolsó értékét tárolja, egy „aValue” változót, mely az aktuális feszültségi értékeket tárolja, illetve egy „operate” változót, mely azt a célt szolgálja, hogy a későbbiek során tudjuk, hogy merre forog az encoder.

A setup() metódusban hozzuk létre azokat a viselkedéseket, melyek nem ciklikusan ismétlődnek, ilyenek a pinek beállításai, a „lastPin” értékének kiolvasása, illetve a soros monitor beállításainak elvégzése.

A loop() metódusban először az „aValue”-nak adunk értéket, mely az A pin digitális portról kiolvasott értéke. Amennyiben ez a jelenlegi érték nem egyenlő a „lastPin” értékével, akkor tudjuk, hogy órajárással megegyezően forog az encoder, hiszen ellentétes értékekkel rendelkeznek. Ilyen esetben a „count” értékét emelhetjük eggyel, illetve az „operate” igaz érték adásával megjegyezhetjük, hogy az encoder biztosan az órajárással megegyezően mozog.

Mivel az encoder egy teljes forgást 40 ilyen értékkel tesz meg, ezért a teljes kör kiszámításához nincs más dolgunk, mint az aktuális értéket elosztani 40-el, így biztosan tudjuk, hogy amennyiben az érték osztható 40-el, írassa ki a rendszer a soros monitorra a jelenlegi teljes körök számát.

Ezek után már nincs más dolgunk, mint a lastPin értékét átállítani a jelenlegi feszültségi értékre, hiszen a következő cikluslefutás során az utolsó érték lesz az előzőnek az aktuális feszültségi értéke.

Miután elkészült a kód, nincs más dolgunk, mint a mikrokontrollerrel összekötött encodert számítógéphez csatlakoztatva feltölteni a kódot a mikrokontrollerünkre. Ezek után képesek vagyunk a soros monitoron keresztül végig követni a forgatási értékek változását, hogy pontosan hányszor fordult egy teljes kört, melyet a következő ábra is szemléltet, ahol jelenleg két teljes kört tett az encoderünk.



2.7. ábra. Soros monitoron megjelenő adat, mely a teljes körök számát jelzi

Az encoderről részletesebben a 5.4.5 fejezetben lehet olvasni.

2.3. Optimalizálás

2.3.1. Telepített napcellák optimalizálása, tájolása

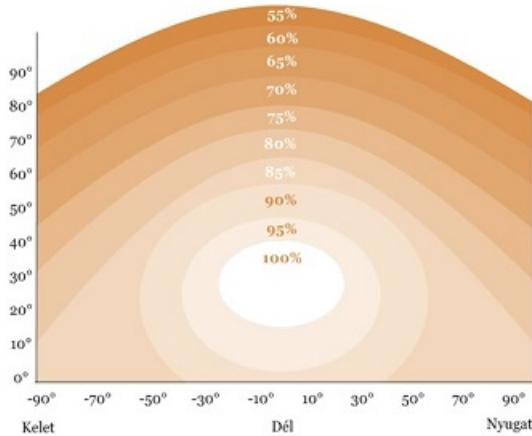
A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Ahhoz, hogy a telepített napcellákat optimálisan tudjuk elhelyezni a terepasztalon, további tanulmányokat kellett végezni az egyes tájolások optimális értékeinek kiszámításához.

Tanulmányok során arra jutottam, hogy az ideális tájolás $30\text{--}40^\circ$ dőlésszögű, déli irány. Természetesen a valóságban nagyon ritka az ilyen felület, így a dél-keleti és délnyugati tájolás és $20\text{--}50^\circ$ közötti dőlésszög megfelel a napelemek telepítésére.[5] Mi azonban déli 40° -os tájolást alkalmaztunk, mivel a terepasztalunk lényege az optimális működés megvalósítása és annak demonstrálása.

Az optimális értékeket Magyarország elhelyezkedési viszonylataira helyeztük, így minden érték Magyarországon belül optimális. Célunk, hogy a terepasztal felhasználói számára egy szemléltetést mutassunk, hogy magyarországi tájolásokkal különböző országokban milyen értékekkel szolgálnak a különböző eszközök.

Tanulmányok során megtudtuk, hogy a tető dőlésszöge, illetve a tájolás hatással van a teljesítményre, melyet a következő ábra szemléltet:



2.8. ábra. A tájolás és a tető dőlésszögének hatása a végleges teljesítményre[13]

Ezek értékek alapján tehát felállíthatunk egy táblázatot, melyben pontosan láthatjuk a dőlésszög, illetve tájolás alapján a teljesítmény értékét %-ban mérve:

	NY	DNY	D	DK	K
0°	85%	85%	85%	85%	85%
10°	80%	85%	90%	85%	80%
20°	80%	90%	95%	90%	80%
30°	75%	90%	95%	90%	75%
40°	75%	90%	100%	90%	75%
50°	70%	90%	95%	90%	70%
60°	65%	85%	90%	85%	65%
70°	60%	80%	85%	80%	60%
80°	55%	70%	80%	70%	55%
90°	50%	60%	70%	60%	50%

2.9. ábra. A tájolás és a tető dőlésszögének hatása táblázatban[13]

Mivel a telepített napcellák nem változtathatják dőlésszögüket, így azok fixen 40°-ra vannak beállítva. Ez esetben elég, ha a táblázatnak a 40°-os sorát nézzük.

Rendszerünk, mely képes az asztalt elforgatni, az asztal elforgatása után közli a felhasználóval a telepített napcellák aktuális pozícióját fokban mérve. Így a rendszer megmondja, hogy a napcellák aktuálisan hány százalékos teljesítménnyel képesek termelni áramot.

Jól láthatjuk, hogy a 40°-os sornál a déli 40° az optimális, tehát a 100%. Ezután ha a napcella Délkelet, illetve Délnyugat irányba néz, akkor már csak 90%, továbbá, ha keleti, vagy nyugati irányba néz, akkor csak 75%-os teljesítményt képes leadni.

2.4. Termelők és fogyasztók

2.4.1. Termelők ismertetése

Termelőink, ahogyan az előző fejezetekben is olvashattuk, nem mások lesznek, mint egy telepített napcella rendszer, mely összesen 3x2 darab egybe forrasztott napcellát tartalmaz, továbbá egy intelligens napkereső napcella rendszer, mely két darab napkereső modellből áll, valamint egy vízerőmű rendszer, melyről a 2.5.3-as és a 2.8-as cikkben olvashatunk részletesebben.

2.4.2. Fogyasztók ismertetése

A fényforrások, és egyéb elektronikai eszközök a különböző energia felhasználású fogyasztókat fogják modellezni. Célunk valósághűen modellezni a fogyasztókat. (Kisméretű házak, épületek.)

Az alábbi modellek lesznek a terepasztalunkon:

- Családi házak (átlag 4 fős, fogyasztása körülbelül 230 kWh/hó)
- Bérházak (átlag 4 fős, fogyasztása körülbelül 200 kWh/hó)
- Tömbházak (bérházak fogyasztásától függően változik)
- Elektromos töltőállomások (használattól függően változik)
- Közvilágítás (alkalmazástól függően változik)

Modellünk olyan fogyasztási értékeket szemléltet, mely a valóságnak arányosan elég tesz. A fogyasztók számát dinamikusan lehet szabályozni, mely hatással van a rendszer működésére. A modellünkben a fogyasztók különböző nyitófeszültségű LED-ek, melyekkel a fogyasztók energiabelhasználását tudjuk szimbolizálni. A projektünkben a fogyasztók egységes áramot használnak, azonban a számítások során a valóságnak megfelelő értékekkel számolunk.

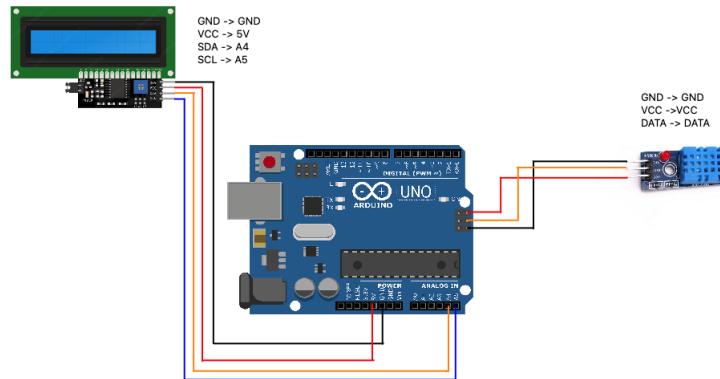
2.5. Modellek

2.5.1. Időjárás állomás

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

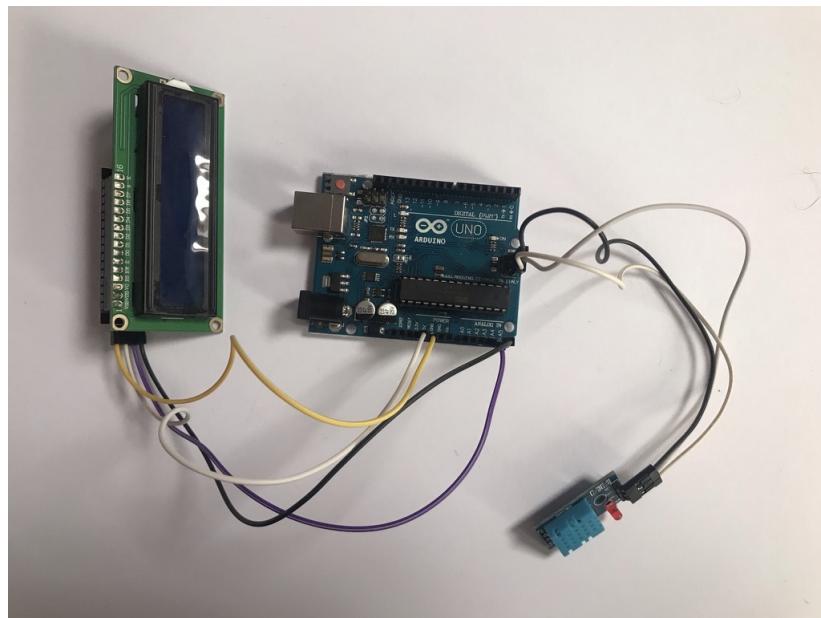
Modellünk tartalmaz egy kis éghajlat elemző műszert is, melyre egy 16x2-es LCD kijelző van csatolva, amin adatokat tudunk leolvasni az éppen aktuális hőmérsékletről és páratartalomról. Ez a műszer szemlélteti a teremben aktuális éghajlati tulajdonságokat.

A modell elkészítése előtt szükséges egy kapcsolási rajzot készíteni, melyben előre meghatározzuk a használni kívánt portok számát, illetve azok értékét.



2.10. ábra. Kapcsolási rajz az időjárás állomáshoz

Miután elkészítettük a kapcsolási rajzot, ezek alapján elkészíthetjük a hardvert.



2.11. ábra. Kapcsolási rajz alapján elkészített időjárás állomásunk

Ahogy a kapcsolási rajzon, illetve az azután elkészített hardveren is látható az Arduino Uno-n kívül szükségünk van egy Dht11-es szenzorra, ismertebb nevén egy

hőmérséklet és páratartalom szenzorra, valamint egy lcd kijelzőre és a hozzá tartozó I2C eszközre, mely egy protokollt valósít meg, amely adatok küldésére és fogadására szolgál.

A hardveres kapcsolási rajzot követően hozzákezdhetünk a szoftveres részhez. Mivel a felhasználó az lcd kijelzőn láthatja a hőmérsékletet, illetve páratartalmat, először az lcd működését kell definiálni. Fontos megjegyezni, hogy ez a 16x2-es lcd kijelző önállóan nem képes egyedi karakterek megjelenítésére, azonban egy karakternek lefoglalt helyre manuálisan készíthetünk egyet. Ez azért fontos, mert a fok jel csak így lesz megjeleníthető.

A kód elején szükséges az előre definiált portokat deklarálni, illetve egyéb olyan változókat, melyeket később a rendszer használni fog.

```
#include <LiquidCrystal_I2C.h>
#include <hd44780.h>
//cime a 0x27 16 karakteres es 2 sorban fog elhelyezkedni
LiquidCrystal_I2C lcd(0x27,16,2);

//igy fog kinezni a fok jele
byte degree_symbol[8] =
{
    0b00111,
    0b00101,
    0b00111,
    0b00000,
    0b00000,
    0b00000,
    0b00000,
    0b00000
};

int pin=11; //pin
volatile unsigned long duration=0;
unsigned char i[5];
unsigned int j[40];
unsigned char value=0;
unsigned answer=0;
int z=0;
int b=1;
```

Ezek után inicializálhatjuk az lcd-t, valamint kiírhatjuk a kezdeti értékeket az lcd kijelzőre. A kezdeti érték nem más, mint a „hom = ? °C”, „para = ? %”

```

#include <LiquidCrystal_I2C.h>
#include <hd44780.h>

void setup() {

    lcd.init(); //1. sor inicializalasa
    lcd.init(); //2. sor inicializalasa
    lcd.backlight();
    lcd.print("Hom\u00e1r");
    lcd.setCursor(0,1); //bottom left
    lcd.print("Para\u00e1r");

    //number, symbol
    lcd.createChar(1, degree_symbol);
    lcd.setCursor(9,0); //1.sor 9. karaktere
    lcd.write(1);
    lcd.print("C");
    lcd.setCursor(10,1); //x=10, y=1
    lcd.print("%");
}

```

Végezetül megírjuk rá a hőmérséklet és páratartalomhoz elkészített algoritmust a „loop()” metódusba.

```

#include <LiquidCrystal_I2C.h>
#include <hd44780.h>

void loop() {
    lcd.display();
    delay(1000);
    while(1)
    {
        //delay(100);
        pinMode(pin,OUTPUT);
        digitalWrite(pin,LOW);
        delay(20);
        digitalWrite(pin,HIGH);
        pinMode(pin,INPUT_PULLUP); //magas lesz alaprelemezetteren
        duration=pulseIn(pin,LOW);
        if(duration <= 84 && duration >= 72)

```

```

{
  while(1)
  {
    duration=pulseIn( pin ,HIGH);
    if( duration <= 26 && duration >= 20)
    {
      value=0;
    }
    else if(duration <= 74 && duration >= 65)
    {
      value=1;
    }
    else if(z==40)
    {
      break;
    }
    i [ z /8]=value<<(7-(z%8));
    j [ z]=value ;
    z++;
  }
}
answer=i [0]+i [1]+i [2]+i [3];
if( answer==i [4] && answer!=0)
{
  lcd . setCursor (7 ,0);
  lcd . print(i [2]);
  lcd . setCursor (7 ,1);
  lcd . print(i [0]);
}
z=0;
//kinullazom az eddig ertekeket ,
//hogy utana ujra tudjon szamolni
i [0]=i [1]=i [2]=i [3]=i [4]=0;
}
}

```

A felhasználó az összeállított hardver számítógéphez való kapcsolása után nyomon követheti a szobában mérhető aktuális hőmérsékletet celsiusban, illetve páratartalmat százalékosan mérve.

2.5.2. Fogyasztók modellezése

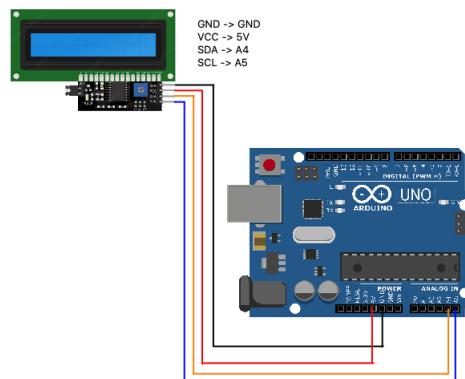
A feladat elkészítéséért **Sass-Gyarmati Norbert** a felelős.

A fényforrások és egyéb elektronikai eszközök a különböző energia felhasználású fogyasztókat modellezik. (Családi házak, lakások, kórházak, valamint iskolák, melyek más fogyasztási igényekkel vannak ellátva)

Korábbi fejezetekben már kifejtettem (2.2.1) a különféle fogyasztók havi, illetve napi fogyasztási igényeit. A szimulátor kiválóan mutatja, hogy a felhasználótól megkapott inputokból, melyek végzik a fogyasztók beállításait, hogy állítja elő a fogyasztási igény megközelítési értékét.

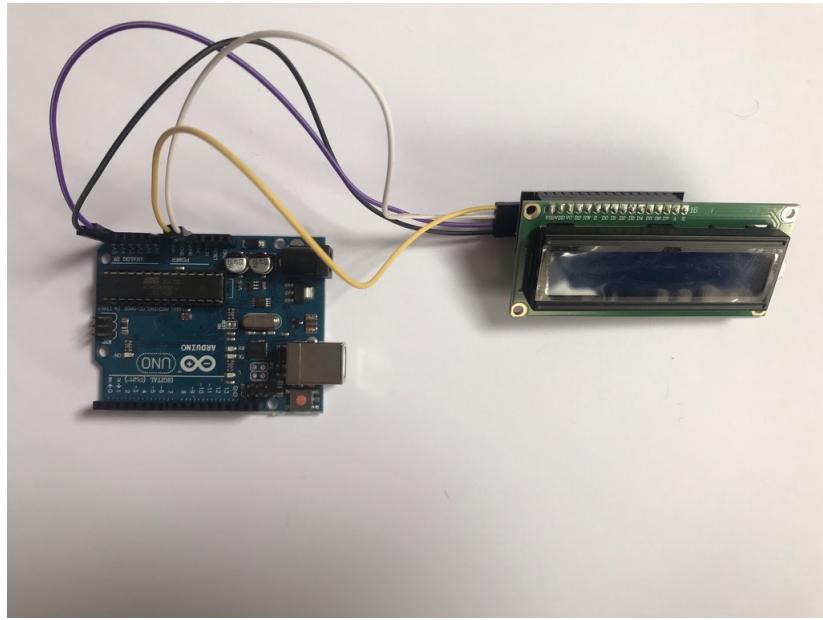
Szimulátor után a következő feladat ennek az algoritmusnak lemodellezése Arduinón, hogy később a teljes rendszer ezek adatok alapján tudja beállítani a fogyasztók igényeinek értékét. Fontos tudni, hogy míg a szimulációban a felhasználó minden adathoz hozzá fért, a modellünkben bizonyos értékek konstansként szerepelnek. Ilyen értékek a tanulók, illetve betegek beállítása. Mivel nem minden felhasználó ismert a téma körében, így egy átlag beteg és tanuló állományt rögzít a rendszer.

A modellt C++ programozási nyelven írtam az Arduino IDE segítségével. Az eszköz önmaga nem igényel az Arduino Uno mikrokontrolleren kívül egyéb alkatrészeket, azonban a szemléltetés érdekében egy 16x2-es kék lcd panellel összekötöttem az Arduino Uno-t, hogy minden lefutásnál közölje a felhasználóval a kívánt adatokat. A kapcsolási rajz tehát tartalmaz egy Arduino Uno mikrokontrollert és a hozzá kapcsolt lcd kijelzőt.



2.12. ábra. Kapcsolási rajz a fogyasztó problémához

A kapcsolási rajz után összeállíthatjuk az Arduino Uno-hoz kapcsolt LCD kijelzőt, melyhez egy I2C modul rögzítve van.



2.13. ábra. Kapcsolási rajz alapján elkészített hardver

A modell az lcd panelen kívül egy úgynevezett soros monitor közvetíti a kimenő értékeket, így a felhasználó a számítógépen is követheti a fogyasztók aktuális igényeinek megközelítő értékét.

2.5. Megjegyzés. Fontos tudni, hogy a soros portra kiírt adatok olvasásához is szükséünk van az Arduino Uno számítógéphez való csatlakoztatására.

Miután definiáltuk a portokat a kapcsolási rajz segítségével, hozzá kezdhetünk a kódhoz, melyben az előre definiált portokat használjuk fel.

Először az osztályokat kell definiálni. Az osztályok a különféle fogyasztókat tartalmazza, melyek más fogyasztási értékkel kerülnek kiszámításra.

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
class House{
public:
    int house_consume = 230;
};
class Flat{
public:
    int flat_consume = 200;
};
class School{
public:
    int students = 300;
    int square_school = 20;
```

```

float school_size = 2.5 * students;
float school_consume = square_school * school_size;
};

class Hospital{
    public:
        int patients = 100;
        int square_hospital = 100;
        float hospital_size = 6 * patients;
        float hospital_consume = square_hospital* hospital_size;
};

```

Majd ezután beállíthatjuk a szoftver lefutás utáni viselkedését. A következőkben definiálni fogjuk az lcd kijelzőt, valamint létrehozzuk az adatszerkezeteket, melyben tárolni tudjuk a deklarált fogyasztókat. Miután létrehoztuk és feltöltöttük az adatszerkezetben a kívánt adatokat, soros monitor keresztül közvetítjük a felhasználónak a fogyasztási értékeket.

```

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
long total_consume = 0;
void setup() {
    lcd.init(); //1. sor
    lcd.init(); //2. sor
    lcd.backlight();
    lcd.print("kWh/m\u00b2");
    lcd.setCursor(0, 1); //bottom left
    lcd.print("kWh/d\u00b2");
    Serial.begin(9600);
    int house_piece = 50;
    int flat_piece = 50;
    int school_piece = 1;
    int hospital_piece = 1;
    House houses[house_piece];
    Flat flats[flat_piece];
    School schools[school_piece];
    Hospital hospitals[hospital_piece];
    int house_length = 0;
    for (int i=1; i <= house_piece; i++){
        if (house_piece == 0){
            break;

```

```

} else {
    houses[ i ] = House();
    house_length = i;
    total_consume +=
        houses[ i ].house_consume;
}
}

int flat_length = 0;
for (int i=1; i <= flat_piece; i++){
    if (flat_piece == 0){
        break;
    }else {
        flats[ i ] = Flat();
        flat_length = i;
        total_consume +=
            flats[ i ].flat_consume;
    }
}

int school_length = 0;
for (int i=1; i <= school_piece; i++){
    if (school_piece == 0){
        break;
    }else {
        schools[ i ] = School();
        school_length = i;
        total_consume +=
            schools[ i ].school_consume;
    }
}

int hospital_length = 0;
for (int i=1; i <= hospital_piece; i++){
    if (hospital_piece == 0){
        break;
    }else {
        hospitals[ i ] = Hospital();
        hospital_length = i;
        total_consume +=
            hospitals[ i ].hospital_consume;
    }
}

```

```

}

Serial.print("Hazak_szama:");
Serial.println(house_length);
Serial.print("Lakasok_szama:");
Serial.println(flat_length);
Serial.print("Iskolak_szama:");
Serial.println(school_length);
Serial.print("Korhazak_szama:");
Serial.println(hospital_length);
Serial.print("Osszes_fogyaszto_igenye_havi_szinten:");
Serial.print(total_consume);
Serial.println(" kWh");
Serial.print("Osszes_fogyaszto_igenye_napi_szinten:");
Serial.print(total_consume / 30);
Serial.println(" kWh");

}

```

A felhasználó ezután az Arduino Uno számítógéphez történő csatlakoztatása után soros monitor keresztül követheti a szoftver kimeneti értékeit. Ezután már csak az lcd kijelzőn kell feltüntessük az értékeket, amennyiben a felhasználó nem csak a soros monitor szeretné megtekinteni a tartalmat. Mivel az lcd kijelző 16x2-es, így csak két sorban tud, maximum 16 karakter/sort megjeleníteni. A szoftver szemléltetése érdekében az lcd csak a havi, illetve napi fogyasztási igényt közli a felhasználóval.

```

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

void loop() {
    lcd.display();
    lcd.setCursor(8, 0);
    lcd.print(total_consume);
    lcd.setCursor(8, 1);
    lcd.print(total_consume / 30);
}

```

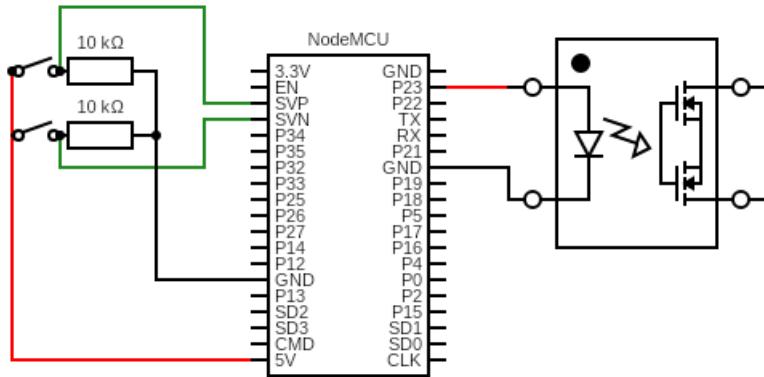
Ahogy a kód részleteből is látható, az lcd működéséhez szükséges paramétereket a loop() metódus törzsében kell definiálni, mivel az lcd kijelzőn közölt adatok állandó jelleggel kell megjelenjenek.

2.5.3. Vízerőmű modellezése

A feladat elkészítéséért Oravecz Zsolt a felelős.

A rendszer összetevőit képezik:

- Egy darab szobaszökőkút szivattyú
- Két darab úszókapcsoló
- Egy darab elektromosan vezérelhető vízcsap
- Egy darab szilárdtest relé



2.14. ábra. Vízerőmű kapcsolási rajza

Vezérlőegységként egy Nodemcu ESP-32S mikrokontrollert használunk. Mindkét kapcsoló esetében lehúzó ellenállásként egy $10\text{ k}\Omega$ -os ellenállást alkalmazunk. Előre meg kell terveznünk, hogy a kapcsoló aktiválásakor milyen állapotot szeretnénk előidézni. Ebben a szerelésben én a magas állapotot választottam, vagyis amikor a kapcsoló aktiválódik, a megfelelő portról magas jelszintet olvashatunk le, ellenkező esetben alacsonyat.

Az ellenállás nélkül a porton feszültség ingadozás lenne megfigyelhető, amellyel nem lenne egyértelmű, hogy a kapcsoló használva van-e. Az ellenállás szerepe, hogy nagy ellenállás mellett zárat nem fog bekövetkezni, viszont annyi áram még éppen átcordogál rajta, hogy leföldelje az adott portot, így nyugalmi állapotban alacsony értéket tudunk leolvasni. A két kapcsoló úszókapcsoló, akkor aktiválódnak, amikor a víz szintje eléri a kapcsoló szintjét és megemeli a kapcsolón található úszót.

A szivattyunk 230V váltakozó árámmal működik, amelynek vezérlését ugyancsak a mikrokontroller látja el. A szivattyúval ellentétben a mikrokontrollert 5V-os egyenáram működteti. Keresnünk kellett tehát egy olyan eszközt, mely úgy kapcsolja a szivattyún az áramot, hogy a két áram között ne legyen fizikai kapcsolat. A választásunk a szilárdtest relére³ esett. Ez a fajta relé abban különbözik a hagyományos társaitól, hogy nem található benne mechanikus alkatrész. Előnye, hogy kicsi feszültséggel tudunk

³ A relé egy olyan kapcsoló elem, mely elektromágnes hatására érintkezőket működtet.

kapcsolni igen magas feszültségű áramköröket is. Az általunk felhasznált típus neve SSR-40DA. Ennek az eszköznek a vezérlő feszültsége 4V-tól 32V-ig terjed, az átkapcsolási feszültsége 24V és 380V közötti érték lehet. Ezekkel a paraméterekkel megfelelő nekünk az eszköz.

Az általunk használt mikrokontroller C++ nyelven programozható. Elkészítettem egy program könyvtárat, melyet a Nodemcu ESP-32S-en lehet beimportálni, a paraméterek megadása után, használhatóak a különböző funkciók. A következő kódsorban a funkciókról kaphatunk információt.

```
Waterplant :: Waterplant( int _felsotartaly ,  
    int _alsotartaly , int _szivattyu , int _csap )  
{  
    felsotartaly=_felsotartaly ;  
    alsotartaly=_alsotartaly ;  
    szivattyu=_szivattyu ;  
    csap=_csap ;  
}  
void Waterplant :: Szivattyuzasindit (){  
    if(analogRead( felsotartaly )<800){  
        digitalWrite( szivattyu ,HIGH);  
        Felsotartalytele=false ;  
    }  
    else{  
        digitalWrite( szivattyu ,LOW);  
        Felsotartalytele=true ;  
    }  
}  
void Waterplant :: Szivattyuallj (){  
    digitalWrite( szivattyu ,LOW);  
}  
void Waterplant :: Csapnyit (){  
    if(analogRead( alsotartaly )>800){  
        digitalWrite( csap ,LOW);  
        Alsotartalytele=true ;  
    }  
    else{  
        digitalWrite( csap ,HIGH);  
        Alsotartalytele=false ;  
    }  
}
```

```

void Waterplant :: Csapzar () {
    analogWrite ( csap ,LOW);
}

```

A létrehozott osztály neve *Waterplant*. Az első sorban a konstruktor paraméterlistáját tekinthetjük meg. Paraméterként a mikrokontroller azon portjait kell megadnunk, amelyekhez az eszközök csatlakoztatva vannak. Balról jobbra olvasva: a felső tartály úszókapcsolója, az alsótartály úszókapcsolója, a szivattyú és végül a csap.

A Szivattyuzasindit() és Szivattyuallj() metódusok a szivattyú működését szabályozzák. A szivattyú indításakor ellenőrizzük, hogy a felső tartály úszókapcsolójáról alacsony jelet kapunk-e. Ha igen, akkor megállapíthatjuk, hogy még nincs tele. Ekkor arra a portra, melyre a szilárdtest relé csatlakozik, magas állapotot írunk, a szivattyú működésbe lép. A „Felsotartalytele” egy logikai változó, értéke igaz, vagy hamis lehet, ebben az esetben nincsen megtelve vízzel, tehát hamisra állítjuk. Abban az esetben, ha magas értéket olvasunk a kapcsoló portjáról, akkor biztosan tudjuk, hogy a felső tartály megtelt és a szivattyút leállítjuk.

A Csapnyit() és Csapzar() metódusok a csap működtetéséért felelnek. A kódSOROK felépítése nagyon hasonló ahhoz, amely a szivattyút működteti. Ebben az esetben ellenőrizzük hogy az alsó tartály nincs-e teli, ha igen, akkor a csapot működtető portra alacsony szintet írunk, és a hozzátartozó logikai változó értékét igazra állítjuk. Ellenkező esetben a csapot megnyitjuk.

2.6. Prototípusok

Egy objektumot akkor tekinthetünk prototípusnak, ha egy olyan objektum kezdetleges modellje, melyet egy bizonyos terv tesztelésére készítettek. A fejlesztőiparban a prototípusokat széles körben szokták használni tervezések, modellezések során, hogy ezzel tökéletesítsék egy projekt bizonyos elemeit és folyamatait, mielőtt azokat nagyobb léptékben megvalósítanák.

A prototípus tervezése során végig nézhetik a termék funkcionalitását, illetve működését egy teszt modellben, így képesek nyomon követni, hogy mi az, ami működik, vagy ami esetleges hibákat okozhatnak.

2.6.1. Telepített napelemek prototípusai

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

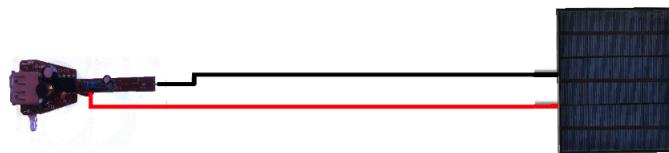
A telepített napcellák elkészítése előtt meg kellett győződjünk, hogy egy napcella valóban képes adatokkal szolgálni, illetve képes lesz szemléltetni a terepasztalunkon a napcellák termelési arányait. Szükségünk volt továbbá azt is lemérni, hogy képes lesz annyi energiát termelni, hogy arányaikkal osztva élethű adatokkal szolgáljon.

Ehhez kellett készítenem egy prototípust, mely lényegében egy napcella, ami egy LED-re kapcsolva képes elegendő energiát termelni, hogy a dióda világítson.

A prototípus egy 6V 4.5W 520mAh Mini Epoxy Monocrystalline napcellából, egy 5V 2A napcellához használatos Power Bank USB töltő Controllerből, mely tartalmaz egy LED-et, valamint az ezeket összekötő AWG szilikon kábelekből áll.

A prototípus összeállítása előtt szükségünk van egy kapcsolási rajzra, melyben összeállítjuk a hardver kapcsolásait, ezzel is csökkentjük a veszélyét az esetleges forrasztási hibáknak.

A kapcsolási rajz tehát:



2.15. ábra. Napcella prototípusának kapcsolási rajza

Mivel a prototípus elkészítése során nem kapcsoltunk hozzá mikrokontrollert, így nem kell hozzá kódot írni, hiszen a kísérlet célja a napcella működésének szimulálása, mely után képesek leszünk egy napcella termelési értékeit nyomon követni.

A kapcsolási rajz elkészítése után összeforrasztjuk a hardvereket. A prototípus elkészítése után a napcella napfény hatására képes lesz az USB töltő controllernek a LED lámpájának magas jelszintet biztosítani, ezáltal a LED felkapcsol, azaz levonhatjuk azt a következtetést, hogy egy napcella sikeresen termelt energiát egy LED-nek.

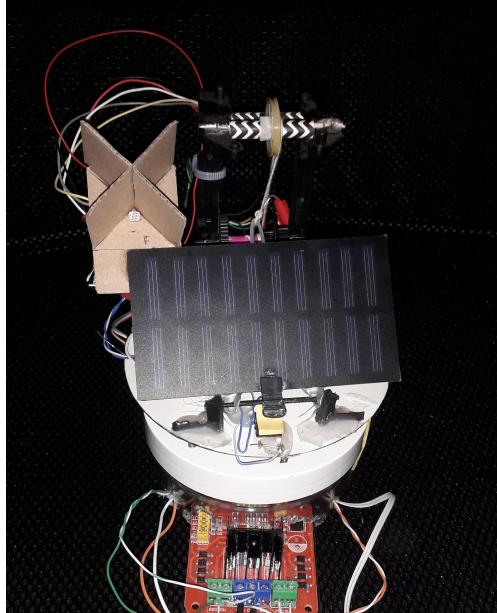
2.6.2. Fénykövető napelemek prototípusa

A feladat elkészítéséért Oravecz Zsolt a felelős.

Dr. Barótfi István Környezettechnika című könyvében olvasható a következő mondat: „A legtöbb energia akkor nyerhető, ha az elnyelő-felület merőleges a beeső napsugárzás irányára (azaz, ha az elnyelő felület normális párhuzamos a napsugárral)”.^[2] Ez a mi esetünkben azt jelenti, hogy a napelem is akkor tudja a legtöbb energiát kinyerni a Nap fényéből, amikor sugarai merőlegesen a napelem felületére. Ez hátrányt jelenthet az adott helyzetben telepített napcellák működésére. Mivel a Nap állandó mozgásban van az égbolton így nem tudjuk garantálni azt, hogy sugarai mindenkor merőlegesen eszenek a felületre.

Így született meg az elhatározás, hogy elkészítsek egy olyan rendszert, amely követi a Nap fénysugarait, és mindenkor merőlegesen eszenek rá a sugarak. Az ötlet nem új, hiszen alkalmaznának már ilyen eszközöket, mégis fontosnak tartottam azt, hogy projektünk során minél több általunk elkészített makettel

szemléltessük a különböző problémákat. Az eszközről egy képet láthatunk.



2.16. ábra. Az elkészült prototípus

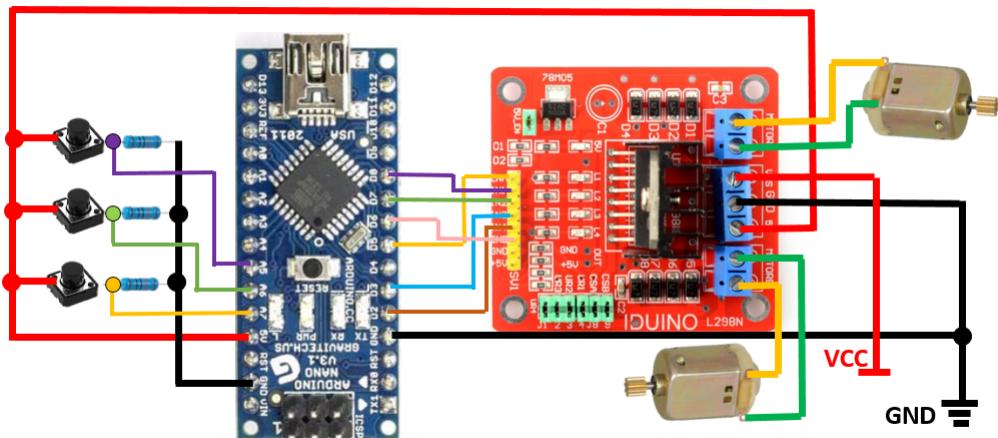
Az eszköz otthon fellelhető háztartási eszközök ből készült, hiszen a fontosság nem a felépítésben játszott nagyobb szerepet, hanem a funkcionalitásban. Az eszköz képes a rajta található napelemet a fényforrásból származó fénysugarakkal megközelítőleg merőleges helyzetbe állítani.

Az eszköz felépítését tekintve két részre osztható: az úgynevezett álló részre, mely egy műanyag dobozból készült, ami magában rejtja az elektronikát, valamint a működő mechanikát és egy úgynevezett forgatható részre, melyen megtalálható a napelem, és a napelemet mozgató mechanika.

Az álló részben lévő elektronikát a későbbiekben fogom tárgyalni. A mechanika roppant egyszerű: egy DCM-3V megnevezésű elektromos motor található itt, mely gumigyűrűvel adja át a hajtási energiát egy műanyagból készült, fogaskerekekből álló hajtóműnek. A hajtómű tengelyére a forgatható rész van rögzítve, mely balra és jobbra tudja elforgatni azt, megközelítőleg 270 fokban. Az álló rész és a forgatható rész között található két végállás kapcsoló, amelyek úgy vannak elhelyezve, hogy meghatározzák a forgatható rész forgási tartományát, azaz megközelítőleg 270 fokot.

A forgatható részen helyezkedik el a napelem, illetve a napelemet emelő állvány, valamint az emelést segítő mechanika. A mechanikus alkatrészek ugyanolyan felépítésűek, mint amilyenek az álló részben megtalálhatóak, azonban itt a hajtómű tengelyére egy zsineg van rögzítve, amelyet képes a tengely magára fel, illetve letekerni. A napelemhez van rögzítve a zsineg másik vége, ami egy álló csigán halad keresztül. Az állócsiga szerepe, hogy elősegíti az erő irányának megváltoztatását.[14] Így attól függően, hogy a zsineget a rendszer feltekeri, vagy leengedi, változik a napcella dőlési szöge. Most már

megtudtuk az eszköz hogyan képes mozgásra, tekintsük meg a benne rejlő elektronikát.



2.17. ábra. A forgatás elektronikája

Az itt található rajzon a következő eszközöket láthatjuk:

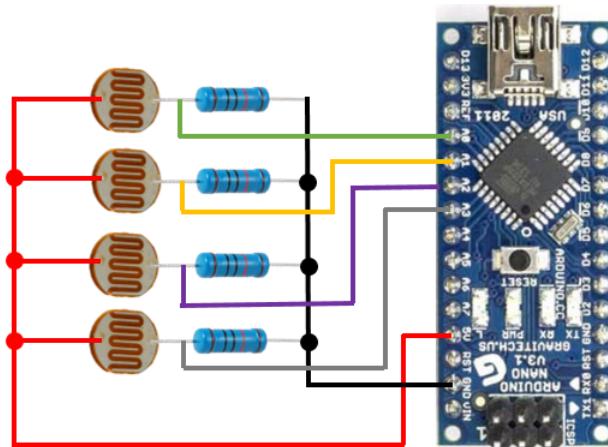
- Egy darab Arduino Nano mikrokontrollert,
- egy darab L298N motorvezérlő modult,
- kettő darab DCM-3V típusú DC motort,
- három darab pillanatkapsolót,
- és végül három darab 10K Ω -os ellenállást.

A megépítés során L293D motorvezérlő chipet szerettem volna használni, azonban a chip kimeneti áramerőssége kevésnek bizonyult az alkatrészek megmozgatásához és az eszköz túlmelegedett, ezért volt szükség az L298N motorvezérlőre, melynek kimeneti áramerőssége 2A, ami már megfelelő.

A mikrokontroller vezérli a többi eszköz működését, és így az eszköz megfelelő pozícióba való állítását.

A pillanatkapsolók érzékelik a forgórész és a napelem mozgásának végállását, amely azt jelenti, hogy ugyan abban az irányban az eszköz tovább nem mozgatható. A három ellenállás funkciójára már kitértünk a 2.5.3 témakörnél. Az eddig felsorolt eszközök azonban nem képesek a fény érzékelésére, ahhoz speciális elektronikai alkatrészre van szükség. Ez az alkatrész a fotoellenállás, ami egy ellenállás, melynek ellenálló képessége a fény intenzitásától függ.[15] Minél nagyobb fény éri az eszközt annál kisebb, minél kevesebb fény éri, annál nagyobb lesz a fajlagos ellenállása.

Tekintsük meg a hozzá tartozó kapcsolási rajzot.

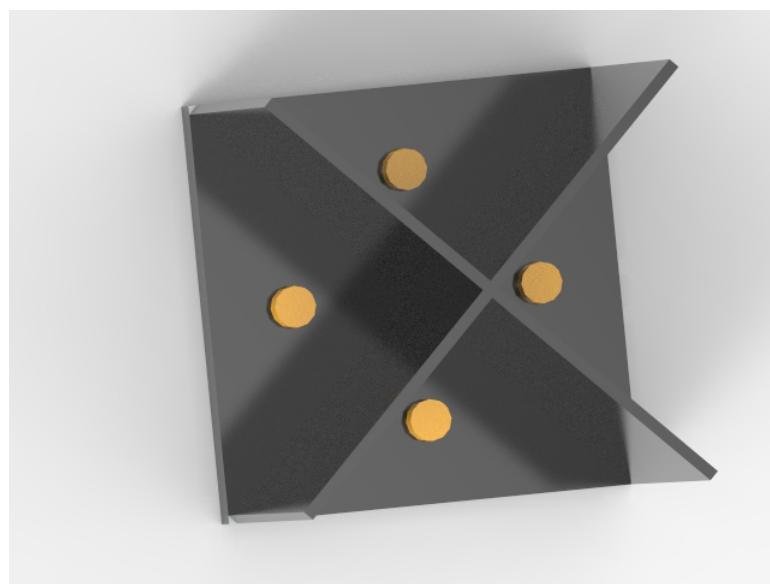


2.18. ábra. A fényérzékelés elektronikája

Az ábrán a következő alkatrészek találhatóak:

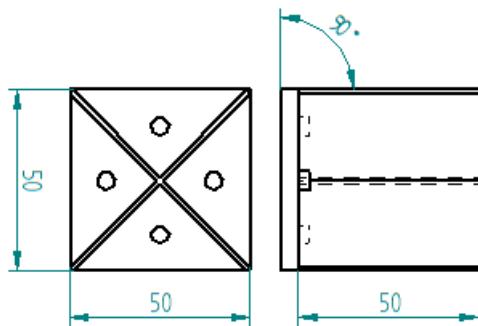
- Egy darab Arduino Nano mikrokontroller,
- négy darab fotoellenállás,
- és négy darab $10\text{K}\Omega$ -os ellenállás, mely a 2.5.3 témaban leírt funkciót látja el.

A rajzon láthatjuk, hogy az fotoellenállásokat a mikrokontroller analóg portjaira csatlakoztatjuk. Az analóg porton 0-tól egészen 1023-ig tudunk decimális számokat olvasni, így összesen 1024 különböző állapot megkülönböztethető.[16] Most már tudjuk érzékelni a fény erejét, így már csak az irányokhoz tartozó értékeket kell összehasonlítanunk. Én ehhez a következő illusztráció látható konstrukciót készítettem el.



2.19. ábra. A fényérzékelő modul illusztrációja.

Az eszköz alapja egy 50mm×50mm alapterületű kartonpapírból készült négyzet, melyre kettő a négyzet átlóinak megfelelő hosszúságú és 50mm magasságú téglalapot helyeztem el. A két téglalap a lapjuk felezővonalánál egymásba csatlakoznak, így keresztezik egymást. A két téglalap a négyzet főátlóinak vonalában foglal helyet. A négy fotoellenállás pedig egy 30mm-es átmérőjű körön lett elhelyezve, 90 fokonként eltolva.



2.20. ábra. A fényérzékelő modul

Láthatjuk tehát, hogy a téglalapok merőlegesek a négyzetre, emiatt kijelenthetjük, hogy amikor a fénysugarak nem merőlegesen esnek a négyzetre, akkor a téglalapból készített fal árnyékot képez valamelyik fotoellenállásra, ennek következtében kisebb értéket olvashatunk le a mikrokontroller adott portján.

A mozgást két részre osztottam vízszintes és függőleges mozgásra. Vízszintes mozgás esetében, amikor a baloldali fotoellenállásra árnyék vetül, akkor utasítjuk a vezérlést, hogy fordítsa el az eszközt a vele ellenkező irányba, azaz jobbra. Így érhetjük el, hogy a fény beesési szögét folyamatosan, megközelítőleg 90 fokra korrigáljuk. Ugyanígy járunk el függőleges mozgás esetében is, amikor az alsó ellenállásra vetül az árnyék, a napelemet felfelé forgatjuk el. Ennek a működésnek alapvető feltétele, hogy a fényérzékelő modult, mely a kartonpapírból és a fotoellenállásokból épül fel, minden azzal a tárggyal együtt mozgassuk, amelyet szeretnénk a megfelelő pozícióba beállítani. Ez a tárgy a mi esetünkben a napelem, ezért rögzítettem a fényérzékelő modult a napelemre.

Az eszköz mechanikai meghibásodási eshetőségei, mérete, illetve korszerűtlensége miatt csupán prototípusként készült el, azonban megteremtette a lehetőséget a terepasztalunkon működő hasonló célt szolgáló eszköz elkészítéséhez.

2.7. Napcellák

Ebben a fejezetben a terepasztalunkon elhelyezett napcellákról olvashatunk, mely tartalmazza a telepített, illetve az intelligens napcellákat, melyek képesek az energia termelésére, így kiszolgálva a fogyasztók igényeit.

A napelemeket szokták szolárcellának nevezni. Ezek a szolárcellák egy olyan fotovoltaikus elemek, melyek a Nap sugárzási energiáját képesek közvetlenül villamos energiává alakítani, majd ezeket hálózatokba táplálják, vagy akkumulátorokban tárolják. Mivel ezek a napelemek csak 12, illetve 24 Volt feszültségnyi egyenáram előállítására alkalmasak, a váltakozó árammá történő átalakításához áram átalakító modul beépítésére van szükség. Ezeket nevezzük invertereknek. A napelemek vékony szilícium lapkákból állnak, melyet különböző hordozófelületre visznek fel. A szilícium lapkákba rétegesen meghatározott tulajdonságú atomokat diffundálnak, ezzel egy úgynevezett szennyezett félvezetőt hoznak létre. A szennyező atomok egy része elektron többlettel bír, míg a másik részének elektron hiánya van a hordozó szilíciumhoz képest. Ha a fényt alkotó fotonok olyan atommal ütköznek, melyeknek elektron többletük van, akkor áramlás indul meg és egyenáram fog folyni a belső hálózatukban. Fény hatására a napelemek pólusai között egyenfeszültség alakul ki.[5]

2.7.1. Telepített napcellák

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

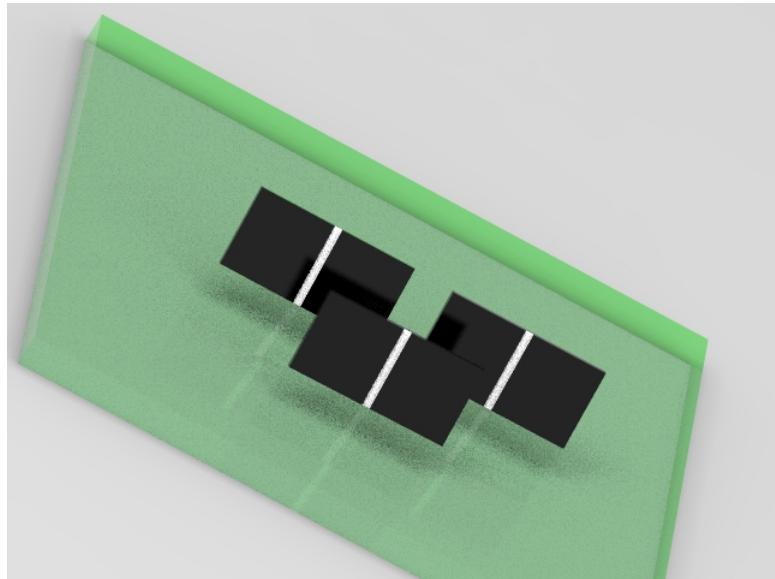
Amint az előző fejezetekben olvashattunk a telepített napcellák prototípusáról, illetve azok teremlésének szimulálásáról, tájolási szabályairól, így nem maradt más, mint hogy elkészítsük a telepített napcellák makettjét.

A prototípus egy 6V 4.5W 520mAh Mini Epoxy Monocrystalline napcellából, egy 5V 2A napcellához használatos Power Bank USB töltő Controllerből, mely tartalmaz egy LED-et, valamint az ezeket összekötő AWG szilikon kábelekből áll. Prototípusunk által szemléltettük, hogy egy napcella képes annyi energiát termelni, hogy a Power Bank USB töltőn lévő led világítson.

Telepített napcella rendszer makettünk összesen 6 darab 50x50 mm-es fejenként 2,3 Volt feszültség leadására képes napcellából áll. Mivel napcella rendszerünk 3x2-es napcellákból áll, így napcelláinkat kettesével raktuk össze, mellyel a feszültség leadásának értékét megdupláztuk, így egy napcella páros összesen 4,6 Volt feszültség leadására képes.

Tájolás szempontjából a 2.3-ik részben leírtak szerint jártunk el, melyben a telepített napcella rendszert a Terepasztal szerinti déli 40°-os tájolással állítottuk be.

Az összeszerelést az alábbi ábra szemlélteti:



2.21. ábra. Telepített napcella rendszer makettje. Készítette: Oravecz Zsolt

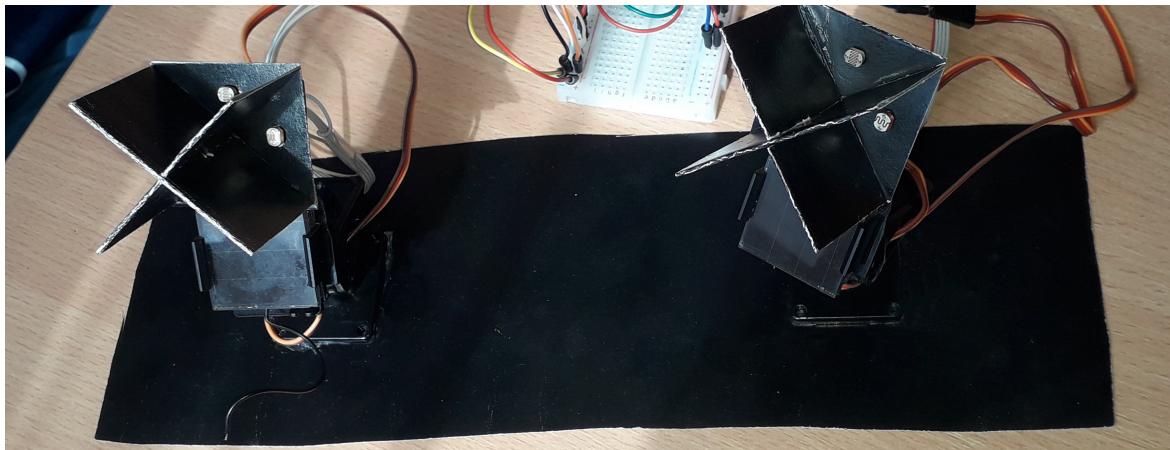
Ahogyan a napcelláról készült maketten is jól látható, telepített napcella rendsze rünk 3 sorban helyezkedik el, valamint egy tábla összesen 2 darab napcellát tartalmaz. Mivel ennek a rendszernek nem kell forognia, így kód szempontjából elegendő a 2.2.3-ik részben leírt kódot használni, melyben egy mikrokontrollerhez kötött napcellának az aktuális feszültségét és teljesítményét jelenítjük meg a felhasználók számára. A kód felhasználása során további problémákat kellett megoldanunk. Míg a napcella teljesítmény mérésére kifejlesztett kód bármelyik időközben képes adatokkal szolgálni, szükségünk volt egy egyfajta szakaszos számításra, melynek célja, hogy egy nappalhossz minden „órájában” egy átlag számítást tudjon végezni, naplemente után pedig egy összesítést tudjon közölni. Ehhez szükségünk volt a Napállás kalkulátorra, mely Python, illetve PHP API-ban is elkészült. A Napállás kalkulátor ugyanis közöl egy olyan adatot is, hogy mennyi egy napnak a teljes nappalhossza, melyet képes tovább küldeni a teljesítmény kalkulátornak, mely ezután a Nap mozgását és teljes futási idejét képes felosztani a felhasználó által megadott napnak nappalhosszával.

Amint ezzel megvoltunk, a rendszer képes lett rá, hogy egy adott napnak teljes nappalhosszából kiszámolja a napcellák egy napra jutó összteljesítményét. Ez a kód azonban alkalmas az intelligens napcellák teljesítményének kiszámításához is, így a 2.2.3-ik részben leírt kód egy multifunkciós, újra hasznosítható kód, mely több célnak is képes megfelelni.

2.7.2. Fénykövető napelem

A feladat elkészítéséért Oravecz Zsolt a felelős.

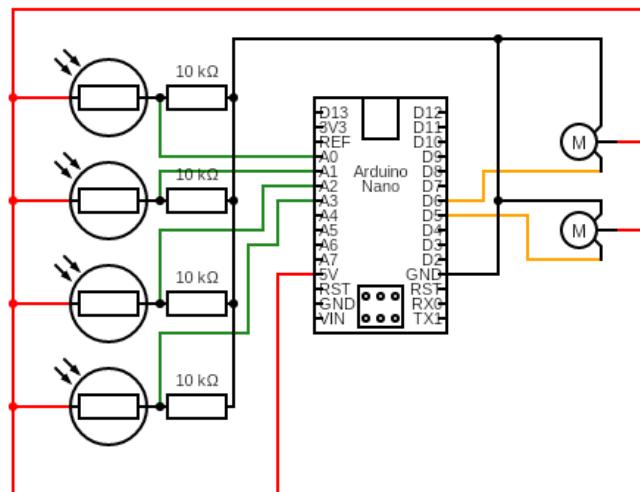
A 2.6.2 fejeztnél megtekintettük a fénykövető napelem prototípusát. Ebben a fejezetben a prototípusból továbbfejlesztett és a terepasztalon felhasznált eszközt fogom bemutatni, amelyet a következő képen meg is tekinthetünk.



2.22. ábra. Fénykövető napelemek

Ennél az eszközönél már rendelkezésre állt egy készen megvásárolható műanyag váz, így azt nem kellett elkészíteni. Sokkal megbízhatóbb és erősebb tartást biztosít a rendszernek, továbbá modern megjelenéssel ruházza fel. A napelemet már nem emelő mozgással pozícionáljuk, hanem két szervomotor mozgatja a megfelelő állásba.

Szeretném bemutatni az eszköz elektronikai részeit, és a működtető mikrokontrollert.



2.23. ábra. Fénykövető napelem kötési rajz

A rajzon megtalálható alkatrészek:

- Egy darab Arduino Nano,
- négy darab fotoellenállás,
- négy darab $10K\Omega$ -os ellenállás,
- két darab SG90 típusú szervomotor.

A vezérlést tekintve ebben az esetben is egy Arduino Nano vezérli az eszközhöz tartozó elektromos alkatrészeket. A négy darab fotoellenállást és a négy darab ellenállást, mely a fényérzékelő modult alkotják korábban a 2.6.2 fejezetben bemutattam, így arról külön nem ejtek szót, hiszen működése és felépítése teljesen megegyezik a prototípusban felhasználttal.

Az igazán nagy különbséget a mozgást biztosító motorok jelentik. Az eszközben két darab SG90 típusú szervomotor található. A szervomotorokat általában elektromos jelekkel vezéreljük, ennek hatására szögfordulást, vagy mechanikai elmozdulást hajtanak végre. Előnyük, hogy pontosan veszik fel a kívánt helyzetet. [17] Az egyszerű villanymotorok hátránya, hogy nem tudjuk a tengelyről meghatározni, hogy az éppen milyen pozícióban van. Ezért szokták a servo motorokat segéd alkatrészekkel ellátni, amelyek behatárolják az elfordulás mértékét. Az SG90 típusú szervomotor 180 fokban képes forgást végezni.

Az általunk használt szervomotor vezérlését végző vezetéket egy „PWM” portra kell csatlakoztatni. A mikrokontroller egy digitális eszköz, ebből adódóan nem tud analóg jelet előállítani, csupán szimulálni azt. A szimulációt úgy végzi, hogy nagyon nagy sebességgel bocsát ki az adott porton magas és alacsony jelszinteket felváltva.[16] Azt a jelet, amit a mikrokontroller ilyen módon állít elő PWM-jelnek nevezünk.

A következő sorokban egy példa kódot láthatunk, mellyel a szervomotort egy kívánt szöghelyzetbe állítjuk be.

```
#include <Servo.h>
```

```
Servo myservo;  
int pos = 90;  
  
void setup() {  
    myservo.attach(9);  
}  
  
void loop() {
```

```

myservo.write(pos);

}

```

Beimportáljuk a fejlesztőkörnyezet által előre definiált „Servo.h” könyvtárat. Majd példányosítjuk a „Servo” osztályt. A „pos” változóban eltároljuk a 90 egész típusú értéket. A „Setup()” metódusban hozzárendeljük a kívánt PWM porthoz a példányt. A „myservo.write(pos);” sor felel a szervomotor mozgatásáért. A „pos” nevű változóban a 90-es érték szerepel, ami azt jelenti, hogy 90 fokba állítja a szervomotor a tengelyét. Paraméterként 0-180 közötti értékek megadhatók, hiszen ez a típusú motor 0 és 180 fok közötti elfordulásra képes.

Végül tekintsük meg, az elkészített eszközt vezérlő programkódot.

```

#include <Servo.h>
#include "Solarplant.h"
#include <Arduino.h>

Solarplant::Solarplant(int _lrpin, int _udpin,
                      int _leftport, int _rightport,
                      int _upport, int _downport)
{
    lrpin=_lrpin;
    udpin=_udpin;
    leftport=_leftport;
    rightport=_rightport;
    upport=_upport;
    downport=_downport;

    lrposition=90;
    udposition=0;

    diff=150;
    delaytime=20;
}

void Solarplant::Enable(){
    enabled=true;
}

```

```

void Solarplant :: Disable(){
    enabled=false;

}

void Solarplant :: StartPosition(){
    horizontal.attach(lrpint);
    horizontal.write(90);
    horizontal.detach();

    vertical.attach(udpin);
    vertical.write(0);
    vertical.detach();
}

void Solarplant :: Move(){
    if(enabled){

        //Program
        horizontal.attach(lrpint);
        vertical.attach(udpin);

        left=analogRead(leftport );
        right=analogRead(rightport );
        down=analogRead(downport );
        up=analogRead(upport );

        if(left >100||right >100||up>100||down>100){
            //Left

            if(left-diff>right &&lrposition >2){
                lrposition --;

                horizontal.write(lrpotion );
                delay(delaytime);
                horizontal.detach();
            }

            //Right
            else if(right-diff>left &&lrposition <180){

```

```

    lrposition++;

    horizontal.write(lrposition);
    delay(delaytime);
    horizontal.detach();
}

//Up
else if(up-diff>down && udposition>0){
    udposition--;
    vertical.write(udposition);
    delay(delaytime);
    vertical.detach();
}

//Down
else if(down-diff>up && udposition<90 ){
    udposition++;
    vertical.write(udposition);
    delay(delaytime);
    vertical.detach();
}

}
}
}
```

Az elkészített osztály neve „Solarplant”. A „Solarplant::Solarplant” kódrészlet az osztály konstruktora. Paraméterben kérjük be a horizontális mozgást végző motor PWM portját, majd a vertikális mozgásért felelős motorét is. Ezek után meg kell adnunk a balra, jobbra, felfelé és lefelé mozgatást érzékelő fotoellenállás megfelelő portját. A konstruktorkban továbbá beállítjuk az „lrposition” változó értékét 90 fokra, az „udposition” értékét pedig 0 fokra. Ez a két változó felel a pozícionáló motorok helyzetéért, ami azt jelenti, hogy kezdő állapotba állítottuk az eszközt. A horizontális motor 90,

míg a vertikális 0 fokban áll. A „diff” változóval tudjuk kalibrálni a fényérzékelő modul érzékenységét attól függően, hogy milyen fajta fényforrással használjuk. Végül a konstruktorban található „delaytime” felel a motorokban történő ciklikus mozgás köztötti szünetért. Ezredmásodpercben adjuk meg azt az időintervallumot, mely eltelik két mozdulat között.

Az „Enable” és a „Disable” metódusok csupán egy logikai változó értékét változtatják igaz, vagy hamis értékre. Az „enabled” változó értékével jelezük a vezérlés felé, hogy az eszköz be- vagy kikapcsolt állapotban van. Bekapcsolás esetén az eszköz kezdőállapotba kerül, mely a „StartPosition” metódus segítségével történik. Az eszköz megfelelő helyzetbe való pozícionálásáért a „Move” nevű metódus felel. A metódus első sorában megvizsgáljuk a korábban említett „enabled” változót, melyben az eszköz állapotát vizsgáljuk, ha kikapcsolt állapotban van, akkor az elágazás törzsében található kódot nem futtatjuk le. Ellenkező esetben csatoljuk a két példányosított szervomotorhoz a megfelelő portokat, majd a left, right, down, up változóba olvassuk a megfelelő analóg portról leolvasható értékeket, melyeket a fotoellenállások állítanak elő. Megvizsgáljuk, hogy a beolvasott értékek nagyobbak-e egy minimális értéknél, esetünkben ez az érték 100, tehát az eszköz csupán abban az esetben lép működésbe, ha a négy fotoellenállás közül, valamelyikről nagyobb értéket olvashatunk le, mint 100.

A továbbiakban a kód négy részre osztható aszerint, hogy melyik irányban lévő fotoellenállást vizsgáljuk. A négy rész működési elve megegyező, ezért most csak az egyik kódrészletről fogok részletesen írni. Tekintsük meg a baloldalon elhelyezkedő szenzor alapján a működést. A témakör elején megemlíttettem egy „diff” nevezetű változót, mellyel a kalibrációt végezzük. Az elágazás feltételében a következőket vizsgáljuk: A bal oldali szenzorról olvasott értéket a „left” változóban tároljuk, ebből a változóból vonjuk ki a kalibrációs értéket. Ezután megvizsgáljuk, hogy a kapott érték nagyobb-e a jobb oldali szenzorról olvasott értéknél és a horizontális mozgatásért felelő motor pozíciója nagyobb 0 foknál. Ha a feltétel teljesül, akkor csökkentjük az „lrposition” nevű változó értékét, ezzel a változtatással az eszközöt jobbra fogjuk forgatni. A következő sorban kiírjuk az adott szervomotorra az értéket. A fordulat után a motort leválasztjuk a vezérlésről.

A további három oldal esetében a kód kisebb változtatásokat eltekintve ugyanazon az elven működik.

2.7.3. Napcellák integrációja

Két összeállított napcella rendszer kiépítésével azt vizsgáltuk, hogy milyen mértékben befolyásolja a termelést egy intelligens napcella a telepített napcellához képest. Kutatásunk során, illetve makettünk elkészítése után kijelenthetjük, hogy nagy mértékben

képes befolyásolni a termelést, hiszen az intelligens napcelláknak a nappalhossz⁴ teljes fázisában képes optimálisan termelni, hiszen amíg le nem megy a nap, az intelligens napcella mindenkor a kedvező szöghelyzetbe állítja celláit, ezzel egy nap teljes nappalhosszát képes kihasználni az energia termelésre, így állíthatjuk, hogy kísérletünk sikeresnek bizonyult.

Bár egy ilyen intelligens napcellának a valóságban megvan a hátránya, hiszen viszonylag rövidebb életűek, szemben a telepített napcellákkal, a folyamatos mozgás hatására emelkedik a meghibásodás kockázata, ezzel egyenes arányban emelkedik a szervizek igénye is.

Fejlesztéseink és kísérletünk során arra is rájöttünk, hogy két napcella rendszer közül az intelligens napcella bizonyult hatásosnak, hiszen ha a felhasználó egy opcionális országot választ, illetve egy általa meghatározott naptári napot, a telepített napcellák nem minden esetben lesznek optimálisan beállítva, hiszen a beállítások magyarországi viszonylatokra lettek kifejlesztve. Ellenben az intelligens napcella rendszerrel, az év bármely napján bármelyik országban, ahol mérhetők a nappalhossz által adott értékek, mindenkor próbálja optimálisan beállítani celláit és ezáltal sok esetben többet és jobban tud termelni a telepített napcellákhoz képest.

Kísérletünk továbbá arra is rámutatott, hogy a termelés hatékonysága csak abban az esetben optimális az intelligens napcellának, amennyiben a tápellátást egy külső forrásból kapja, azaz ha nem önen tartó a rendszer. Egy intelligensen forgó napcella ugyanis a napsugár követésével energiát használ, hiszen a motorok folyamatosan mozognak, ezzel nem csak termelő, hanem bizonyos mértékben fogyasztó is. A fogyasztási igény kielégítésére azonban sajnos a makettünk nem képes, így egy külső tápellátásból kapja az forgáshoz szükséges energiát. Amennyiben a modellünk lényegesen nagyobb cellákkal rendelkezne, úgy képes lehet az önen tartásra, de terepasztalunk esztétikai kinézete, illetve a valóság leképzése a terepasztalra miatt kisebb napcellákat használtunk fel.

Valóságban természetesen egy ilyen rendszer kiépítéséhez sokkal nagyobb cellákat használnak, így fogyasztási igényük elhanyagolhatók a termelési kapacitásukhoz mérve, hiszen egy teljes nappalhosszban nem feltétlen forog minden alkalommal az intelligens napcella, csak bizonyos időközönként. Terepasztalunkon ugyanis a Nap egy teljes nappalhosszt percek alatt szimulál, ennek következtében láthatjuk úgy, hogy mindenkor mozog a napcella.

További tervként szolgál tehát, hogy törekedjük rendszerünk folyamatos továbbfejlesztésére, termelőink tökéletesítéseire, ide beleértve a víz- és napcellák teljesítményét, illetve terheléses tesztekkel minden lehetséges viselkedéseket tudjunk rögzíteni.

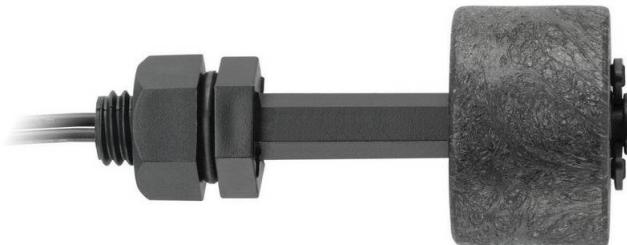
⁴ napfelkeltétől naplementéig tartó intervallum

2.8. Vízerőmű

Napjainkban problémát jelent az elektromos energia tárolása, ezért a zölden megtermelt elektromos energia felhasználásával vizet szivattyúzzunk fel egy magasan elhelyezett tartályba, növelte ezzel annak helyzeti energiáját, és amikor szükséges, akkor egy turbina segítségével vissza alakítjuk elektromos energiává.

Két műanyagból készült tartályt használunk fel, melynek egyikét a másik tartály tetőpontjánál magasabban helyezzük el. Az alsó tartályt tele töltöttük vízzel. Az alul elhelyezkedő tartályt alsó tartályként, amelyik felül helyezkedik el felső tartályként fogom említeni.

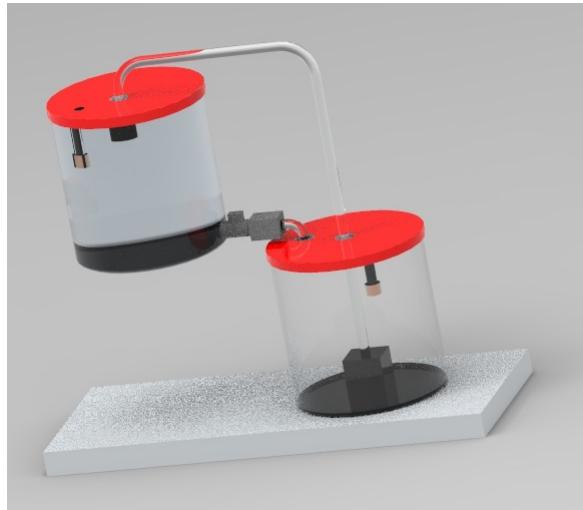
Az alsó tartályban elhelyezkedik egy szobai szökőkút szivattyú, melyre egy cső van felfogatva, a cső pedig a felső tartályba van belevezetve. Továbbá egy úszó kapcsoló, mely megtekinthető a következő képen.



2.24. ábra. Úszókapcsoló

A felső tartályban találunk egy úszókapcsolót, mely azt hivatott érzékelni, amikor a tartály megtelt vízzel, illetve található beleépítve egy elektromos árammal vezérelhető csap is. A csaphoz egy műanyag cső csatlakozik. A cső áthalad egy általunk készített turbinán, és meghajtja azt. A cső vége az alsó tartály tetejére illeszkedik.

Amikor a rendszer úgy érzékeli, hogy többlet energia van a rendszerben, vagyis a megtermelt energia több, mint az elhasználandó energia, akkor a rendszer ellenőrzi, hogy a felső tartály nincs-e tele. Ha nincsen tele, akkor egy kapcsoló működésbe lépteti a szivattyút és felszivattyúzza a vizet a felső tartályba mind addig, amíg az meg nem telik. Ezzel eltároltuk az energiát.



2.25. ábra. Egy sematikus rajz a működésről. Készítette: Oravecz Zsolt

Abban az esetben, amikor a rendszerben energia hiány keletkezik, akkor a rendszer leellenőrzi, hogy az alsó tartály nincs-e tele. Ha nincsen tele, akkor egyértelműen arra következtethetünk, hogy a felső tartályban van víz. Ilyenkor az elektronika kinyitja a csapot, és a víz tehetetlenségből adódóan lefolyik. A víz meghajtja a turbinát, a turbinához rögzített encoder segítségével meghatározhatjuk a turbina fordulatszámát. Az encoderről az 5.4.5-ik részben olvashatunk bővebben, valamint a 2.2.4-ik részben létrehozott kóddal képesek vagyunk a vízturbina fordulatszámát meghatározni. A fordulatszám alapján összehasonlíthatjuk egy valódi vízturbinával, paraméterei alapján pedig kiszámolhatjuk, hogy a turbina mekkora energiát termelt meg.

Miután a víz lefolyt az alsó tartályba, az elektronika elzárja a csapot, és kezdődhet előlről a folyamat.

2.9. További tervezek

Az elkészített terepasztalunkon vannak javítani valók, legfőképp a minden napra készen tartás, optimalizálás, illetve a műszerek karbantartása, így ezeket folyamatosan teszteljük és esetlegesen újabb ötletekkel bővíjtük. Terveink közé tartozik a 2.7.3 részben olvasott cikk, melyben termelőink folyamatos továbbfejlesztését olvashatjuk. Mivel a projektünk több részfeladatokból áll, melyek több vezérelhető elektronikai kellékekből épül fel, így további tervezek közé tartozik az elektronikai hardverek karbantartása, esetleges hibák javítása.

Mivel rendszerünk zárt, így a vízerőmű is két tartállyal működik, melyet a felhasználó a terepasztalunkban kijelölt helyen megtekinthet, azonban kívülről nem tudja, hiszen nincsen egy külső folyó, vagy tó, amivel jobban tudnánk szemléltetni a víz fel-szivását, illetve annak való kiengedését a turbinákat meghajtva, így további tervezek közé tartozik, hogy egy ilyen zárt rendszerből megvalósíthassunk egy nyílt rendszert.

Rendszerünk azért nem nyílt, mert több kockázattal jár, hiszen több tényező miatt meghibásodhatnak a készülékek. Ilyen például a víz, hiszen ha a víz kifolyik, esetleg beszennyeződik, eldugíthat egyes készülékeket, aminek a hatására a készülékek élete jelentősen csökkenhet.

2.10. Összefoglalás

Terepasztalunk megépítése, szoftverek kifejlesztése és a hardverek szakszerű összeállítása során számos új technológiával, munkastíussal, valamint képességgel ismerkedtünk meg. A hardverek összeállítása során megismerkedhettünk több, számunkra még ismeretlen eszközzel, melyek gondos utána olvasás és tesztelések után sikerült megismernünk és a tudást elsajátítanunk.

Szoftverfejlesztés során részletesebben megismerkedhettünk a C++ programozási nyelvvel, melyről bővebben a 5.3.3-ik fejezetben olvashatunk. Továbbá a szimulátorok, illetve egyes modellek megírásához egy eddig ismeretlen nyelvet, a Python programozási nyelvet használtunk a PyCharm IDE segítségével, mellyel gyarapítottuk programozási nyelvünk tárát.

A hardveres és szoftveres tudás mellett földrajzi, illetve csillagászati tudásunkat is sikerült gyarapítanunk, melyet a napállás kalkulátor matematikai szemszögből való megközelítése eredményezett, melyben számos olyan kifejezéssel találkoztunk, amit eddig még nem hallottunk, vagy nem tudtunk kiszámolni matematikai egyenletek segítségével.

Összességeiben nagyon tetszett nekünk a projektünk fejlesztése, hiszen egy igen érdekes kérdéssel tudtunk foglalkozni, hogy vajon képesek vagyunk zöld energiával élni? Vajon tényleg szükségünk van a fosszilis energiahordozókra? Kutatásaink, kísérleteink, illetve modelljeink határozottan kimondta, hogy bizonyos korlátokkal sikerülhet, lényegesen kevesebb energiát tudunk ezáltal termelni, de egy szébb, zöldebb életet élhetünk, melyben az ökológiai lábnyomunk töredéke lesz az eddigi életünkhöz képest és ezáltal egy szébb jövőt teremthetünk az utókornak.

Bizonyos szempontból a zöld élet mindig kompromisszumhoz vezet, ilyen például az elektromos autózás is. Az emberek megszokták, hogy minden gyorsan akarnak, nem szeretnek várakozni, gyorsan akarnak tankolni. Egy elektromos autónál a türelem rózsát terem, hiszen többet kell várni, míg az akkumulátor feltölt, viszont sokkal olcsóbban, zöldebben lehet vele közlekedni, mellyel a környezetünket kíméljük. Ugyanez a helyzet a zöld energetikai rendszerekkel is, hiszen igaz, hogy nem képes annyit termelni, mint egy atomreaktor, viszont egy sokkal egészségesebb, környezetbarát életet élhetünk ezáltal.

3. fejezet

Weboldal

3.1. Bevezetés

Makettünk elkészítésével, szeretnénk hozzájárulni a megújuló energiaforrások elterjedéséhez, illetve szeretnénk működését, minden megtekintővel megismertetni. Ezt nagyban segítik a Nemzeti Média- és Hírközlési Hatóság 2018-as online piackutatásában olvasható adatok. A megkérdezettek 76%-a gondolja úgy, hogy az internetnek köszönhetően gyorsabban és több információhoz jut, 74% gondolja úgy, hogy az internet nagy segítségre van a tanulásban. [18] Ezen információk és személyes tapasztalataink ösztönöztek arra, hogy az elkészült eszközünkhez egy webalkalmazást is készítsünk, mellyel minden internethasználóhoz eljuttathatjuk az általunk közölt információkat, élményeket. Továbbá az is célunk, hogy egy egyedi tanulásmódszertant készítsünk, mellyel a felhasználók interakcióba lépve az eszközzel, játékosan, élmennyeket szerezve ismerjenek meg új technológiákat.

3.2. CodeIgniter keretrendszer

A webalkalmazás elkészítésénél figyelembe kellett vennünk az alábbi szempontokat:

- az alkalmazást PHP nyelven szeretnénk elkészíteni
- szerverünk nem rendelkezik nagy méretű háttértárral,
- a használt rendszer támogassa az MVC fejlesztési modellt¹,
- könnyen lehessen bővíteni, ha később bővül az elkészített eszközeink száma,
- terepasztalunk működtetése közben sok információt kell mozgatnunk a felhasználó és az eszköz között. Természetesen elengedhetetlen, hogy a rendszer képes legyen adatbázis műveletek végrehajtására,

¹Egy tervezési minta. Model-View-Controller. Három fő komponensből áll: Modell, Nézet, Vezérlő.

- és végül, de nem utolsó sorban, a használt rendszer legyen költség hatékony.

Így esett választásunk a *Codeigniter* PHP alapú, nyílt forráskódú keretrendszerre. Ez a keretrendszer megfelel a legtöbb elvárásunknak. Továbbá felhasználó barát, számos hasznos funkciót tartalmaz, jól dokumentált. A felhasznált fájlok és könyvtárak csak a használatakor töltődnek be a memóriába, ezzel spórolva az erőforrásainkon.

3.3. Adatbázis

A weboldal használatához, az oda látogatóknak regisztrálniuk kell magukat, és így létre kell hozniuk egy felhasználói fiókot, ezzel hozzáférve az oldalon található szolgáltatásokhoz, és alkalmazásokhoz. A felhasználó fiókok elengedhetetlenek az alkalmazás megfelelő működéséhez. minden egyes felhasználóhoz tartoznak adatok, ezek lehetnek: vezetéknév, keresztnév, irányítószám, felhasználói név, jelszó, látogatás dátuma, illetve egy időbélyeg, ami akkor kerül bejegyzésre, amikor a felhasználó egy adott oldalt nyit meg, ennek a várakozási lista elkészítéséhez van köze. Ezt a későbbiekbén részletezzük.

Ahhoz, hogy az eszköz és a felhasználó közötti adatcsere létrejöjjön, további adatokat kell tárolnunk. A felhasználói fiókok és az eszköz adatait egy adatbázisban tároljuk elkülönítve egymástól, vagyis megkülönböztetünk felhasználók és eszközök adattábláit. Amikor egy eszköz állapota megváltozik, azt az adatbázis, megfelelő adattáblájának, megfelelő objektumához tartozó mezőbe írja bele. Amikor a felhasználó egy adott információt szeretne megtudni, lekérdezést indít az adatbázis felé. A lekért adatok pedig megjelennek a weboldalon, ilyen adatok lehetnek például, az aktuális fogyasztók száma, az asztal földrajzi helyzete, vagy a megtermelt energia mennyisége.

A weboldalunk PHP nyelven íródott, ehhez egy olyan adatbáziskezelő eszközt kellett választanunk, mely támogatja a PHP nyelvet. A phpMyAdmin egy nyílt forrású eszköz, amely rendelkezik egy felhasználóbarát kezelőfelülettel. A felület elérhető a böngészőből, és egy kezdő szintű felhasználó is problémák nélkül használhatja. A grafikus felhasználói felület mellett lehetőségünk nyílik SQL parancsok futtatására is.

3.4. Weblapról

3.4.1. Az eszközök vezérlése

A rendszer fő szempontja a mobilos vezérlés, így jogosan érezhetjük azt, hogy ez inkább a fiatalabb generációkat célozza meg, azonban fontos, hogy minden korosztály számára érthető és egyértelmű legyen az információ, ami a felületen megjelenik. Első lépésként a látogatóknak regisztrálni kell a felület használatához. A regisztrálás folyamata hasonló a más weblapoknál fellelhető módokkal, itt a felhasználó általános adatokkal kell szolgáljon a szolgáltatás igénybevételehez.

The screenshot shows a registration form titled "Regisztrálás". The form fields are as follows:

- Irányítószám
- Teljes név
- Email cím
- Felhasználó név
- Jelszó
- Jelszó megerősítése

3.1. ábra. Regisztráció számítógépes felületen

The screenshot shows a registration form titled "Regisztrálás" on a mobile device. The form fields are identical to the desktop version:

- Irányítószám
- Teljes név
- Email cím
- Felhasználó név
- Jelszó
- Jelszó megerősítése

3.2. ábra. Regisztráció mobilos felületen

Ahogy a képeken is látható, a felhasználónak rendelkeznie kell egy teljes névvel, irányítószámmal, email címmel, felhasználónévvel, valamint egy jelszóval. Az első képen a felhasználó számítógépes felületről tudja elérni, míg a második kép már telefonos felületen elérhető.

Természetesen a regisztrált felhasználóknak a bejelentkezés gombra kattintva egyből a kezdőlapra tud bejutni.

Továbbá megkérjük, hogy adj meg nemét, korát, és jelölje be érdeklődési körét egy listában. Ezeket az adatokat feldolgozzuk és későbbiekben a felhasználók javára tudjuk fordítani.

Melyek tartoznak az érdeklődési körödbé? (Válassz egyet, vagy többet.)

- Informatika
- Egyéb műszaki tudományok
- Természettudományok
- Művészetek
- Bölcseztudományok
- Egyéb

Kérjük add meg az életkorodat!

10

Kérjük add meg a nemed!

- Férfi
- Nő
- Egyéb

Regisztrálás

3.3. ábra. Regisztráció számítógépes felületen

Regisztrálás után a felhasználó rögtön bejelentkezhet és megjelenik a térkép menüpont, ahol kiválaszthatja az okoseszközzel vezérelhető interaktív tartalmakat, vagy a főoldalon található *Eszközök* menüpont alatt találja őket.

Az eszköz oldalán általános információkat és érdekes képeket találhatunk az adott projektről. A leírás alatt egy várólista található. A várakoztatásra későbbiekben kitérünk. Az ablak alatt az eszközhöz megfelelő gombok, vezérlő panelek találhatóak. Ezek többségében gombok, legördülő listák, csúszkák.

A felhasználó a megfelelő funkciót kiválasztja, rákattint az indító gombra, az alkalmazás összegyűjti a felhasználó által megadott parancsokat és továbbítja azt az adott eszköz felé. Azzal a témaival, hogy miként jut el az adat az eszközhöz szintén később foglalkozunk.

3.4.2. Várólista

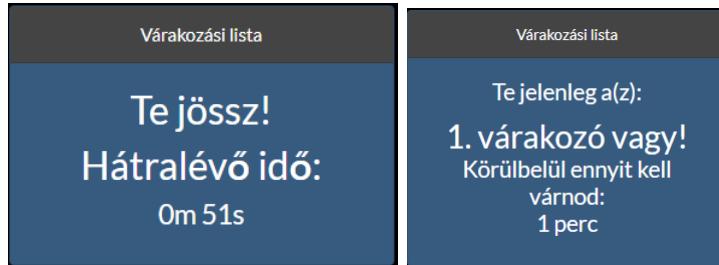
A feladat elkészítéséért Oravec Zsolt a felelős

Amikor a felhasználó vezérelni szeretne egy eszközt, rá kell kattintania az indító gombra. A webalkalmazás elküldi az információkat a mikrokontrollernek és az eszköz elkezd működni.

Minden egyes eszköznek működés közben van egy ciklusideje². Képzeliük el azt az egyszerű szituációt, ha egy eszközhöz csupán egy be- és kikapcsoló gomb tartozik. Az egyik látogató be szeretné kapcsolni a tárgyat, míg a másik ki, és ezt egyidőben hajtják végre. Az eszköz nagy valószínűséggel nem fog csinálni semmit, és jogosan gondolhatnánk hogy az eszköz nem megfelelően működik. Pedig a hiba nem az eszközben, hanem a vezérlésben rejlik.

² Így nevezzük azt az időtartamot, amennyi idő alatt az eszköz egy ismétlődő műveletet elvégez.

Ezen probléma elkerülése miatt született meg az ötlet, hogy várólistát készítsek. Amíg valaki várakozik, a vezérléshez szükséges elemek rejtetté válnak, csupán abban az esetben lesznek láthatóak és használhatóak amikor a várakozó a lista elejére kerül.



3.4. ábra. Várólista

Az alapgondolat, hogy amikor a felhasználó megnyit egy oldalt az alkalmazásunkban, a program egy időbelyeget szűr be az adatbázis megfelelő cellájába, ami egyértelműen a felhasználóhoz kapcsolódik. Ez a módszer lehetőséget nyit arra, hogy megtudjuk melyek azok az eszközök, vagy oldalak, amik jobban felkeltik a használók érdeklődését. Így későbbiekben, amelyik eszközöt nem tartják izgalmasnak, azt tudjuk fejleszteni, és újra felkelthetjük az érdeklődést.

Amikor a felhasználó megnyit egy eszközt, az adatbázisból SQL lekérdezések segítségével, az ott tartózkodók időbelyegét és felhasználónevét egy tömbbe töltjük. Ezután a tömböt az időbelyeg alapján sorba rendezzük, így egy FIFO³ adatszerkezetet kapunk.

Így minden a tömbünk első eleme lesz az a felhasználó, aki sorra következik. Amikor valaki sorra jut, a szöveg megváltozik az ablakban. Tájékoztat róla, hogy mi következünk, illetve elindít egy egyperces számlálót. Erre azért volt szükség, hogy mindenkinél nagyjából azonos időt kelljen várnia és az egy perces időtartamba, minden eszköz elindítási folyamata belefér.

Amikor valaki elindította az eszközt, az oldal automatikusan lefrissül, illetve frissül az adatbázisban az időbelyeg is. Mivel az időbelyeg későbbi, mint a többi felhasználóhoz rendelt időbelyeg, ezért a sor végére kerül. Ugyanez történik abban az esetben is, amikor valaki egy perc alatt nem indítja el az adott eszközt.

A Codeigniter keretrendszerben a programkódok addig futnak, amíg a böngésző meg van nyitva, hiszen a böngésző futtatja le és hajtja végre a programkódokat. Abban az esetben, ha valaki várakozási listában áll, és bezárja a böngészőjét nem kerül újabb időbelyeg az adatbázisba, vagyis ebben az esetben az ő felhasználói fiókjához rendelt idő lesz a legkorábbi. Így folyton ő lesz az aki használhatja az eszközt. Ez egy igen nagy probléma, hiszen a sorban várakozóknak gyakorlatilag végig kellene várakozni.

Némi gondolkodás után született meg az az ötlet, miszerint kibővíti az adattáblát egy újabb oszloppal, amelyben az utolsó aktivitást tároljuk.

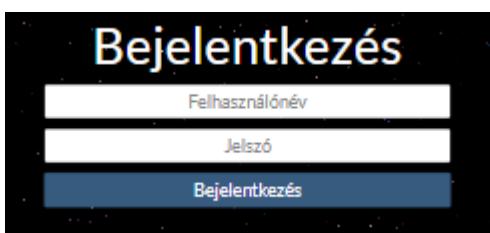
³ A Firs In First Out szavak rövidítése. Sor adatszerkezeteknél használják. Az adatszerkezetbe került elemek, minden a sor végére kerülnek.

A JavaScriptben⁴ megtalálható setInterval() metódust használtam. A metódusnak két paramétert kell megadni, vesszővel elválasztva. Az első egy függvény, melyet szeretnénk a vezérléssel végrehajtatni egy adott idő elteltével. A második paraméter az idő értéke milliszekundumban⁵ megadva. Ez idő letelte után hajtódik végre az első paraméterben megadott függvény.

Amikor az adatbázisból lekérdezzük a várakozási listában található felhasználókat, megvizsgáljuk, hogy az aktuális időpont és a legutóbbi aktivitásuk időpontjának különbsége nagyobb-e egy időintervallumnál, a mi esetünkben ez egy perc. Ha nagyobb, akkor ebben az esetben megállapíthatjuk, hogy a felhasználó és a szerver között megszakadt a kapcsolat. Ez adódhat abból az esetből, hogy bezárja a böngészőjét, de adódhat egy egyszerű technikai problémából is. Ezzel az eljárással, kivédjük azt, hogy egy inaktív felhasználó a várólistában ragadjon, ezzel feltartva a többi várakozót.

3.4.3. Bejelentkezés

A weboldalunkat felkereső látogató, aki még nem rendelkezik felhasználói fiókkal, csupán az általános funkciókat tudja igénybe venni. Megtekintheti a kezdőlapot, információkat olvashat a készítőkről, megtekintheti a blogot, azonban a blogba nem tud bejegyzéseket írni, nem láthatja a térképet és nem tudja az eszközöket vezérelni. Ehhez regisztrálnia kell magát. A sikeres regisztráció után a bejelentkezés oldalra navigálva egy letisztult felület fogadja.



3.5. ábra. Bejelentkezés

A felületen csupán két mező található: a *Felhasználói név* és a *Jelszó* mező. Ide kell a feszhasználónak beírni a megfelelő adatokat. Végül rá kell kattintania a *Bejelentkezés* gombra. Az alkalmazás lekéri a megfelelő adatokat az adatbázisból, és ha van olyan adatpár ami egyezik a bevitt adatokkal, akkor a felhasználó sikeresen bejelentkezett és számára elérhetővé válnak funkciók. Ha a bevitt adatok közül az egyik vagy mindkét adat helytelen, a bejelentkezési folyamat sikertelen és az alkalmazás hibaüzenettel tájékoztatja a felhasználót.

⁴ A JavaScript programozási nyelv egy objektumorientált, prototípus alapú szkript nyelv. Weboldalak esetében használják.

⁵ Egy ezredmásodperc. A másodperc egy ezred része.

3.4.4. Kezdőlap

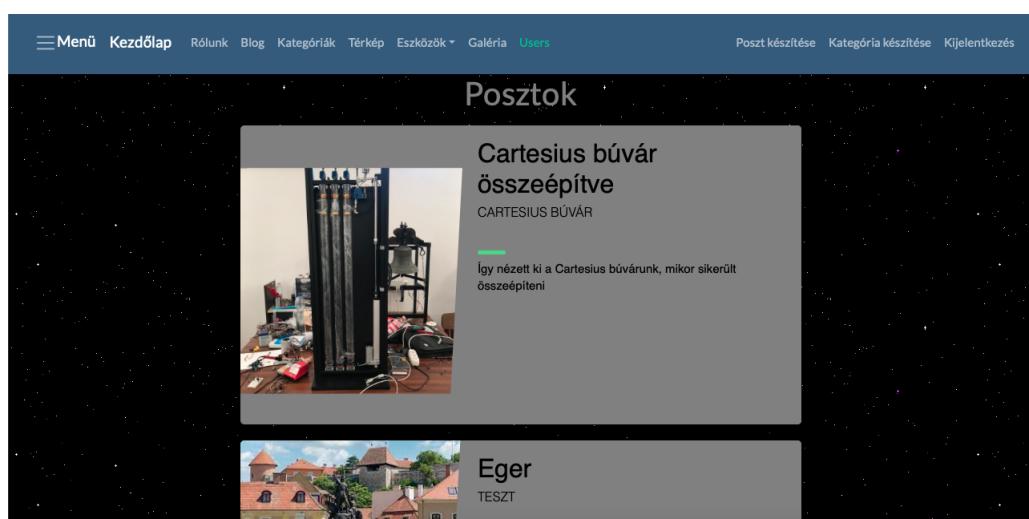
A kezdőlapon a varázstorony aktuális hírei érhetők el, e-mail címek és nyitvatartási rendek. Kezdőlapunk egy már meglévő weblapnak alapját dolgozza fel (<https://uni-eszterhazy.hu/hu/egyetem/kultura/varazstorony>).

3.4.5. Rólunk

A fejléc következő része a Rólunk ablak, melyben a projektben résztvevő fejlesztők és egyéb szerkesztők neveit olvashatjuk. Ez az ablak ismerteti a felhasználókkal az egyes modulok felelőseit, forrásait, továbbá itt helyezkedik el a kapcsolatfelvételi ablak, mely azt a célt szolgálja, hogy felhasználóink könnyebben feltudják venni a kapcsolatot velünk egy esetleges észrevétel, illetve panasz kapcsán. A kapcsolatfelvételről bővebben a 3.4.13 alfejezetben olvashatunk.

3.4.6. Blog

A blog oldal azért készült, hogy a felhasználók észrevételeket, tapasztalatokat és egyéb véleményeket tudjanak feltölteni, ezáltal egymással is tudnak kommunikálni. A blogban lehet képet is feltölteni, valamint egyes kategóriák által lehet csoporthozzáírni. A kategóriák a varázstoronyban megtalálható eszközök. További kategóriák létrehozásához admin szintű felhasználóra van szükség. Amennyiben igény keletkezik egy új kategória létrehozásához, úgy a felhasználók írhatnak a rendszer admin szintű felhasználójának, ami átvizsgálás után létre is jön. A blogban továbbá lehet írni egy részletes leírást a témáról. Egy blog küldése után a rendszer megjegyzi a küldés utáni naptári időpontot, melyet a leírás fölött kiír.

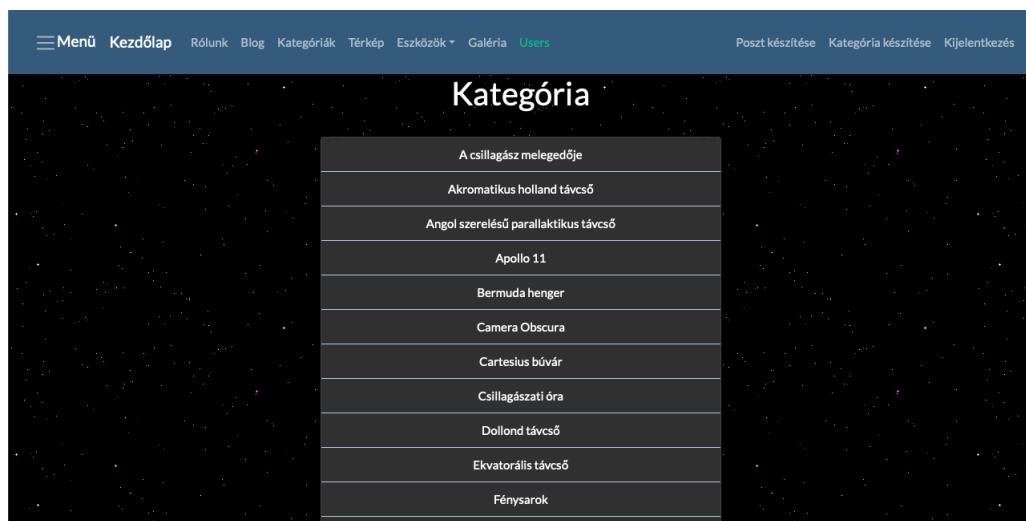


3.6. ábra. Blog tartalmának megtekintése számítógépes felületen

Ahogyan a képen is láthatjuk, weblapunk egyik posztja a Cartesius búvár kategóriába tartozik, ahol a felokosított Cartesius búvár elkészítéséről készült képet is feltöltöttünk. Fontos azt is megjegyezni, hogy kategóriák azért kellenek, hogy később a posztokat listázni tudjuk kategóriák segítségével. Ha egy felhasználó csak egy bizonyos kategória iránt érdeklődik, lehetősége van azokat kilistázni, ezáltal egy kényelmesebb és könnyen kezelhető felület tárul elé. Weblapunk nagy hangsúlyt fektet a felhasználóbarát webes megjelenítésre, így egy letisztult és kényelmes weblap jelenik meg minden felhasználóink számára.

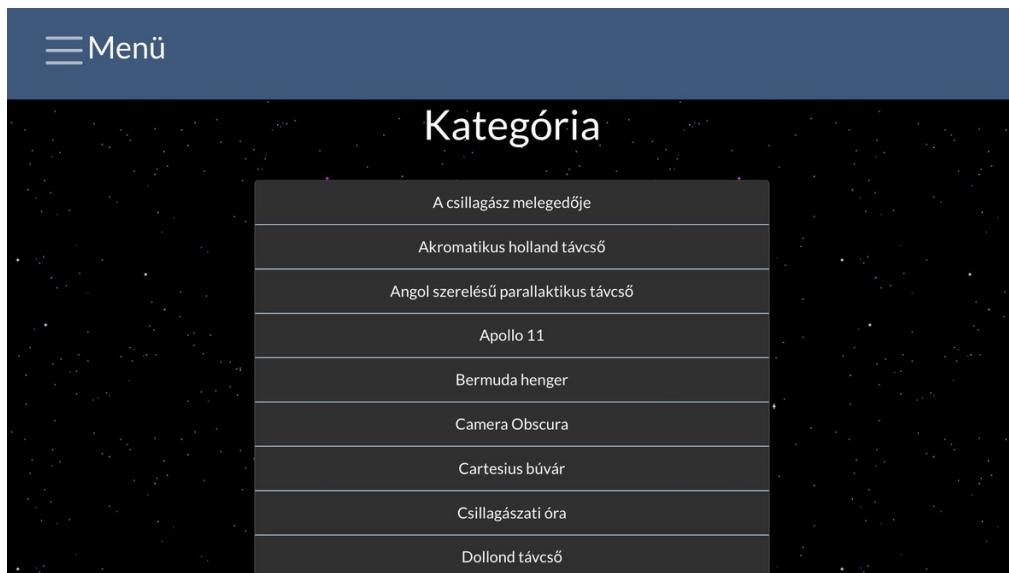
3.4.7. Kategóriák

Mint már említettük, szoftverünk tartalmaz egy kategória ablakot, melyben az eddig feltöltött összes kategória közül tud választani a felhasználó. Egy szabadon választott kategória kattintásra kilistázza az eddigi összes olyan posztot, észrevételeket és egyéb tartalmakat, melyek abban a kategóriában szerepelnek. Ezáltal a felhasználó csak azokat a kategóriában szereplő tartalmakat olvashatja, amelyek érdeklik. A kategóriák a varázstoronyban szereplő eszközök, melyeket admin szintű felhasználók, illetve rendszer karbantartók tudnak módosítani, mezei felhasználóknak azonban személyes igény esetén lehetőségük van írni az üzemeltetőknek.



3.7. ábra. Kategóriák megtekintése számítógépen

Ahogy a képeken látható, weblapunk létrehozása után a varázstoronyban megtalálható kategóriákat töltöttünk fel, melynek kattintására a kategória által létrehozott posztot olvashatjuk. Míg az első ábrán számítógépes felületről nyitottuk meg, a felhasználók számára kényelmesebb, hiszen a fejlécben minden információt láthatnak. A második ábra telefonról készült, így a telefonos megjelenítés szempontjából a fejléc tartalmait elrejtettük, mely a bal felső ikon kattintására kilistázódik. Szoftverünk



3.8. ábra. Kategóriák megtekintése mobil eszközön

multi platformos, tesztelve lett Windows-on, Linuxon, illetve MacOS alatt. Telefonon tesztelve lett Android, illetve IOS készülékeken.

További előnyként szolgál az is, hogy a kategóriák ABC sorrendben listázódnak ki, ezáltal további könnyedséggel szolgál egyes kategóriákat elérése.

3.4.8. Térkép

A feladat elkészítéséért Oravecz Zsolt a felelős.

A felület segítségével a felhasználók idegenvezető nélkül bejárhatják a Varázstorony termeit, és különböző leírások segítik az egyes eszközök megismerését. Célunk, hogy azok a felhasználók, akik még nem jártak a varázstoronyban, tudjanak tájékozódni és ki tudják keresni a számukra érdekes témaikat, melyről rendszerünk képekkel, információkkal és egyéb interaktív dolgokkal szolgál. A térkép fülre kattintva a varázstorony szintenkénti alaprajza található, ahol minden terem, folyosó, ahol eszközök találhatók, fel van tüntetve. Három fajta feltüntetés van a rendszerünkben.

1. Megtekinthető tartalom:

- Felhasználóink meg tudják webes felületről tekinteni az egyes termek érdekkességeit. A gombra kattintva egy pop-up szerű kép jelenik meg az egyes eszközökről.

2. Interaktív tartalom:

- Felhasználóink számára biztosítunk interaktív vetélkedőket egyes eszközök kattintása után. Ezek lehetnek kvízek, csoportos mini feladatok.

3. Vezérelhető tartalom:

- Felhasználónk ilyen típusú gombra kattintva az olvasás és a megjelenő kép mellett vezérelni is tudja egyes eszközöket.

3.4.9. Jelmagyarázat a térképhez

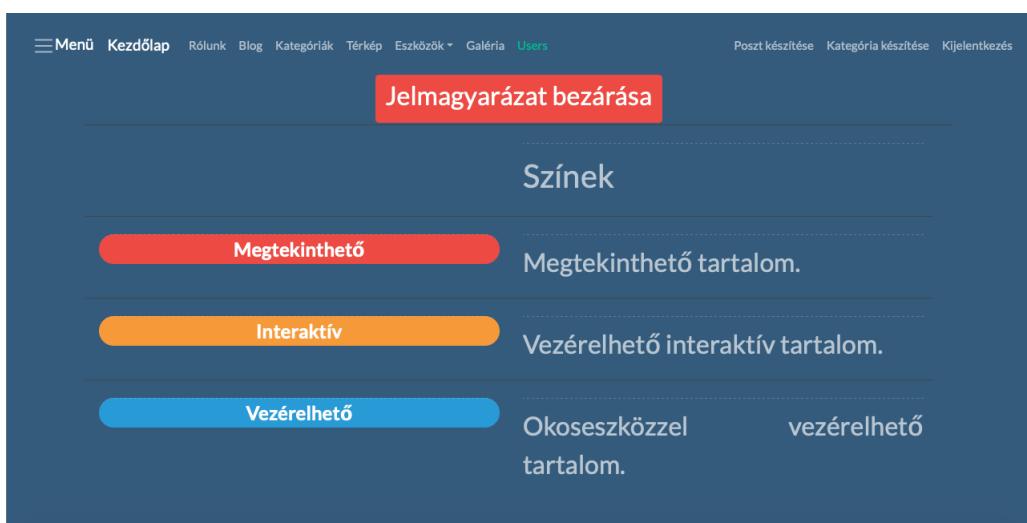
A feladat elkészítéséért Oravecz Zsolt a felelős.

A térkép megfelelő értelmezéséhez létrehoztunk egy jelmagyarázatot, mely mobiltelefonos, és számítógépes nézetben is egy gomb, melyet megnyomva felugrik egy sáv, itt a felhasználó megtudhatja, hogy a térképen jelölt eszközök mely kategóriába sorolható.

Weblapunk három funkciót biztosít a felhasználók számára:

- megtekinthető
- interaktív
- vezérelhető

A funkciók mellé szín is társul.



3.9. ábra. Jelmagyarázat a térkép funkcióhoz

Ahogy a mellékelt képen is láthatjuk, a megtekinthető tartalmak színe piros, azok a tartalmak, melyek interaktív feladatokat tartalmaznak sárgák, végül a tartalmak, melyeket vezérelni is lehet, kékes zöld színűek.

3.4.10. Eszközök

A felhasználóknak lehetőségeük van egyes eszközöket részletesebben tanulmányozni, mely az eszközök ablakra kattintva lesz elérhető. A gombra kattintva előjük tárul az általunk fejlesztett projektek részletes beszámolója, illetve azok leírása, egyéb tartalma. Ezek természetesen a térkép menüpont alatt is megtalálhatók, hiszen azok gombjaira kattintva átirányítja felhasználóinkat az általuk választott oldalra.



3.10. ábra. Kép a Cartesius búvár oldaláról



3.11. ábra. Kép a Terepasztal oldaláról

Az első képen a Cartesius-búvár, illetve annak részletes leírása található, míg a második képen a terepasztal, mely egy intelligensen működő energetikai rendszert valósít meg és szakdolgozatunk témaja is egyben.

3.4.11. Kvíz lehetőség

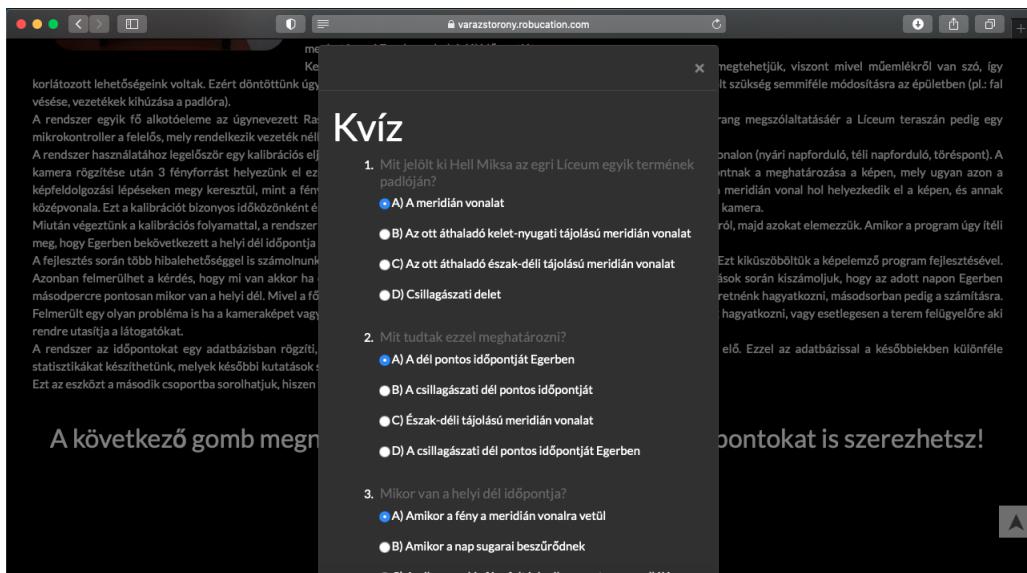
A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Weblapunkon az eszközök menüpontnál néhány eszközön elérhető egy úgynevezett kvíz lehetőség, mely azért készült, hogy teszteljék tudásukat az adott témából.

Az eszközök képet, illetve szöveget tartalmaznak, némelyek pedig interaktív tartalmakat, melyek a következők lehetnek:

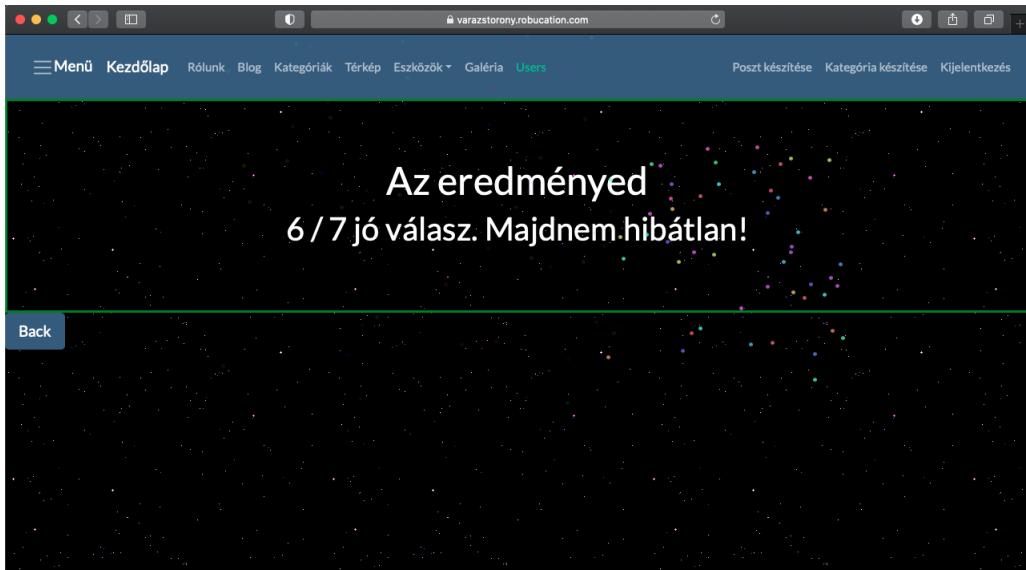
- Vezérelhető felület
- Interaktív kvíz felület

A kvíz felületet egy gombnyomással lehet elérni a kiválasztott eszköz menüpontnál, ezután lehetősége van a felhasználónak az eszközhöz csatolt olvasmány alapján kitölteni a teszt feladatokat.



3.12. ábra. Kvíz kitöltéséhez létrehozott form

A tesztkitöltés után a „Beküld” gombra kattintva a weblap egy másik oldalra irányítja át a felhasználót, melyen meg tudja tekinteni az elért pontszámait, illetve egy kis megjegyzést a pontszámaihoz. Visszalépés esetén egy gomb funkcionál, mely rákkattintására visszairányítja a felhasználót az eredeti oldalra.



3.13. ábra. Kvíz eredményének közlése a felhasználó számára

3.4.12. Felhasználók

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

Weblapunk a fehasználók és a vendégek után rendelkezik admin szintű felhasználókkal, melyek feladata a kategóriák, illetve egyes posztok karbantartása. Így az admin felhasználók fejléce kiegészül egy "Kategória készítése" menüponttal, melyben az általa, vagy közösen megbeszélt kategóriákat tudja feltölteni. Adminként nem lehet regisztrálni, ezt a rendszer tulajdonostól lehet igényelni, melyet a rendszer karbantartó átir az adatbázison keresztül.



3.14. ábra. Admin szintű felhasználók fejléce

Ahogyan a képen is látható, az admin továbbá rendelkezik egy „Users” menüponttal, melyben megtekintheti az egyes usereket (regisztrált felhasználókat), illetve azok adatait adatbiztonság céljából. Ezenfelül megtekintheti, hogy kik adminok a rendszer felhasználói közül.

Varázstorony	Kezdőlap	Users	Poszt készítés	Kategória törlése	Kijelentkezés
ID	zipcode	name	email	username	admin
1	[REDACTED]	[REDACTED]	[REDACTED]	sassnorbert99	1
4	[REDACTED]	[REDACTED]	[REDACTED]	GG	1
5	[REDACTED]	[REDACTED]	[REDACTED]	Chris	1
7	[REDACTED]	[REDACTED]	[REDACTED]	Ilonka	0

3.15. ábra. Felhasználók listázása admin szintű felhasználók részére

A képen látható adatokat szándékasan nem jelenítjük meg adatbiztonság érdekében. Ahogy az ábra is mutatja, listázva vannak a felhasználók. Az admin szintnek két lehetséges értéke van, 0, ha a felhasználó nem admin, valamint 1, ha a felhasználó admin szinttel rendelkezik.

Ahogy a képen láthatjuk, az 1-es, 4-es és 5-ös ID-vel (egyedi azonosító) rendelkező felhasználóink admin szintje 1, tehát admin szintű felhasználók.

3.4.13. Kapcsolatfelvétel

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

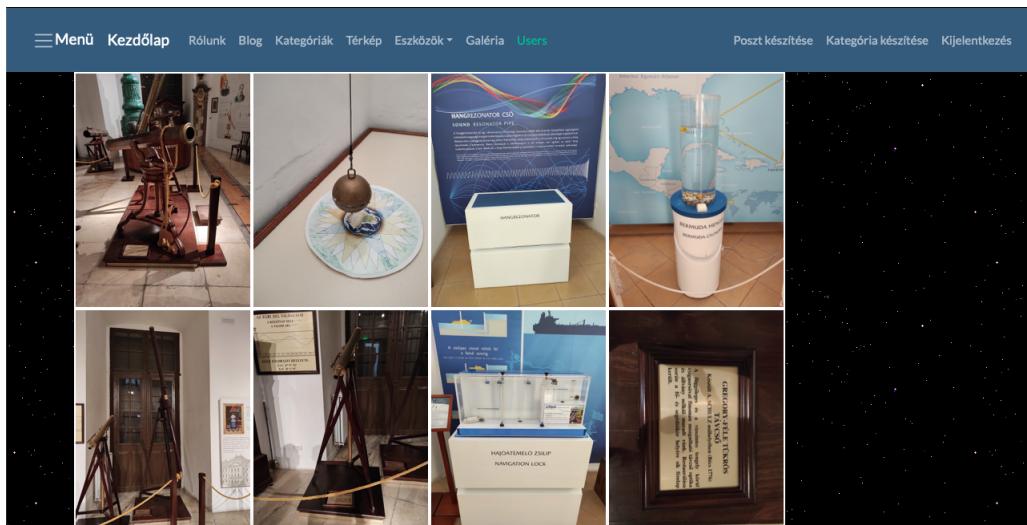
Weblapunk tartalmaz egy kapcsolatfelvételi űrlapot, melynek célja a felhasználókkal való kommunikáció biztosítása. Amennyiben valamilyen hiba, esetleg észrevétel, panasz lépne fel, a felhasználóknak lehetőségük van ezeket a kéréseket elküldeni a rendszer üzemeltetőknek. A kapcsolatfelvételi űrlap a „Rólunk” menüpontban található meg a lap alján. A felhasználónak meg kell adnia a teljes nevét, email címét, egy tárgyat a levélnek, valamint a szövegtörzset, melyben 500 karakterben kifejtheti észrevételeit, esetleg panaszait. Az adat egy adatbázisban rögzül, melyhez a rendszer üzemeltetőknek, illetve az adatbázis üzemeltetőknek van joguk belépni.

3.16. ábra. Kapcsolatfelvételi lehetőség felhasználók számára

3.4.14. Galéria

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős.

Weblapunk felhasználóinak lehetőségeük van a varázstoronyban elhelyezett kellékek-ről készült képek megtekintésére a „Galéria” fül alatt, mely a fejlécen található. A képek méretei következtében csak korlátozott számú képeket lehet megtekinteni, melyek rá-kattintásával a kép teljes méretében megtekinthető.



3.17. ábra. Galériában elhelyezett képek megtekintése

A galériában lévő képek mellett videót is megtekinthetnek a felhasználók, melyben a varázstoronyról készült bemutató videót nézhetik meg.

3.5. További tervezettségek

Az elkészített webes applikációt van még javítani való, ezeket folyamatosan javítjuk, és tökéletesítjük a kódot, illetve a design részt. Amennyiben bármelyik felhasználónknak igényei keletkeznének, a kapcsolatfelvételi lehetőség egy kiváló választás, hogy ezeket az észrevételeket, igényeket közölhesse velünk. Amennyiben személyesebb üzenetre tart igényt, úgy a kapcsolatfelvétel alatt megtalálhatja a fejlesztők facebook-os elérhetőségeit.

További terveink közé tartozik a weblap folyamatos naprakész tartása, illetve újabb eszközök felvétele, azok mikrokontrollerrel való irányíthatóságának megvalósítása, illetve az aktuális igényeknek megfelelő design létrehozása.

Kvízek terén is vannak további terveink, hiszen egy kvíz kitöltése után a felhasználóink százalékának a töredéke tölti ki ugyanazt a kvíz sorozatot. Szeretnénk több kvízt létrehozni, esetleges felhasználói kutatásokat végezni, hogy milyen témák érdeklik az embereket.

A webes applikáció elkészítése során számos olyan technikákkal és módszerekkel ismerkedtünk meg, mely további webes platformra történő fejlesztésekre inspirálnak minket.

3.6. Összefoglaló

Rendszerünk célja tehát a kliensek teljes körű kiszolgálása, valamint az újonnan látogatók tájékozódásának elősegítése. Ezen kívül rendszerünk felhasználói könnyen és egy letisztult weblapon keresztül információkat gyűjthet egy általa választott témából, illetve személyes megjegyzéseit és tapasztalatait meg tudja osztani a rendszeren belül, hogy mindenki számára elérhető forrás legyen.

Továbbá rendszerünk felhasználóinak lehetőségük van interaktív tevékenységeket folytatni, melyről a mikrokontrolleres vezérlők az eszközök menüponban, illetve a kvíz kérdések szolgálnak. Ahhoz, hogy a vezérlés ne okozzon konfliktust olyan felhasználók számára, akik egyszerre szeretnének kapcsolódni a vezérléshez, egy várakozó listát hoztunk létre, melyben mindenki érkezést követően sorrendben van eltárolva és ezáltal minden felhasználó kap egy percet, hogy vezérelhesse az általa kiválasztott eszközt a weblapon keresztül.

Amennyiben a felhasználó nem az interaktív tartalmak miatt keresi fel az oldalt, lehetősége van további információkat, egyéb érdekességeket szerezni a varázstoronyról kapcsolatban. Erre a célról lett fejlesztve a blog, galéria, illetve a térkép funkció. Míg a galériában képeket és videót nézhetünk, a térkép menüpont alatt bejárhatjuk a teljes varázstoronyt és az ott található eszközökről információkat kaphatunk.

Amennyiben felhasználóink egyéb észrevételeket, esetleges panaszokat közölnének

velünk, erre is sikerült lehetőséget biztosítani, mely a kapcsolatfelvételnél lehetséges, ahol ugyanis lehetőségük van írni a fejlesztőknek, adatbázis kezelőnknek, illetve a szer-ver tulajdonosnak.

4. fejezet

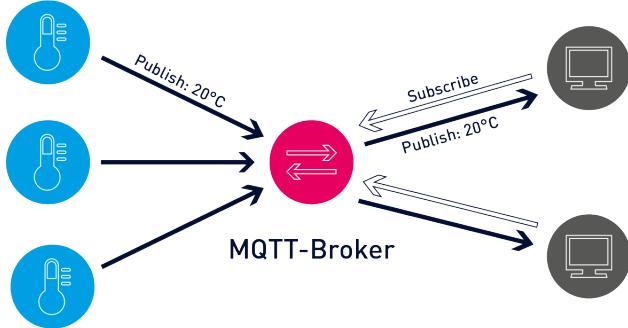
A weboldal és a mikrokontroller közötti információcsere

A feladat elkészítéséért Oravecz Zsolt a felelős.

Rendszerünk működése nagyvonalakban a következő lépésekkel áll. A felhasználó ellátogat a weboldalra és regisztrálja magát, ezek után bejelentkezik és elnavigál egy adott eszköz oldalára. Az oldalon kivárja sorát, majd sorra kerülés esetén megjelennek vezérlő felületek, amelynek segítségével életre keltheti az eszközt. Ezek nagyon egyszerű lépéseknek tűnhetnek, azonban a webalkalmazás és a mikrokontroller közötti kommunikáció korántsem ennyire egyszerű.

Weboldalunk PHP nyelven íródott a használt mikrokontroller, a Nodemcu ESP-32S pedig C++ nyelven programozható. A két eszközben többek között az a közös, hogy mind a kettő csatlakoztatható az internethez. A csatorna így már adott már csak egy közös protokollt kell találnunk, melyen folyhat a kommunikáció, és lehetőség szerint nyíltforráskódú.

Így esett választásunk az MQTT protokollra. Az elnevezés a Message Queuing Telemetry Transport rövidítése. [19] Kifejezetten számítógépek, és egyéb eszközök, mint például mobiltelefonok, mikrokontrollerek és szenzorok működéséhez tervezett protokoll. Előnye, hogy korlátozott sávszélességű hálózatokra lett optimalizálva, továbbá olyan hálózatokra, ahol a kapcsolat időszakos. Az MQTT a TCP/IP protokollon keresztül kommunikál. Az MQTT architektúrájában kétféle szerepkör létezik: az egyik a kliens, a másik az úgynevezett broker. A broker az a szerver, mellyel a kliensek kommunikálnak. A felépítés lényege, hogy a kliensek nem kommunikálnak egymással, minden üzenetet a szervernek küldenek. Az információcsere az ügyfelek között a szerveren keresztül történik. minden ügyfél lehet adó, vevő vagy mindkettő egyszerre. Ezt a következő képpel szeretném szemléltetni. Az MQTT egy esemény vezérelt protokoll, ami azt jelenti, hogy nincs folyamatos adatátvitel. Ez minimálisra csökkenti a forgalomat a hálózaton. A kliens csak akkor küld adatot, ha valamit szeretne megosztani a többiekkel, a szerver pedig csak akkor küld adatot a klienseknek, ha valaki információt



4.1. ábra. Az MQTT protokoll működése. [20]

szeretne lekérni.

A protokollon az üzeneteket témákba, úgynevezett „topics” formában rendezve tehetjük közzé. A témák hierarchikusan vannak felépítve és a témákat egymástól a „/” jellel tudjuk elválasztani, ez hasonlatos a számítógépes fájlrendszer felépítéséhez. Ez a felépítés elősegíti, hogy a felhasználó pontosan be tudja azonosítani az általa használandó témát, így nem keverve össze azokat. A témák nincsenek külön definiálva, a témák nevét az üzenetek tárolják, így ha küldünk egy üzenetet, amely egy eddig nem definiált témához tartozik, akkor ehhez már csatlakozhatnak is egyéb kliensek.

A kommunikáció során négy művelet lehetséges

- Feliratkozás, azaz „subscribe”
- Közzététel, azaz „publish”
- Ping
- És végül lecsatlakozás, azaz „Disconnect”

Feliratkozás során jelezzük a szerver számára, hogy egy adott témával kapcsolatban információt szeretnénk lekérni. Közzététel esetén egy adattömböt hozunk létre, mely tartalmazza az adott témát, a témához tartozó felhasználói nevet és jelszót, a kliens azonosítóját, a titkosítási módot és végül az üzenetet. Amikor elküldjük a szervernek az adattömböt, megvizsgálja hogy létezik-e ilyen téma. Abban az esetben, ha már létezik, akkor a témához tartalmazó adatot lecseréli az általunk küldött adatra, ellenkező esetben létrejön a téma a hozzá tartozó adattal együtt. A Ping funkció esetén a szervernek egy kérést küldünk, ő pedig egy válaszban biztosít minket arról, hogy közöttük él a kapcsolat. A lecsatlakozással pedig azt közöljük a szerverrel, hogy már nem kell a feliratkozott kliensek felé adatot küldenie és a feliratkozott kliensektől már nem fog üzenetet kapni. A projektünk során a Blue Rhinos által PHP nyelven elkészített könyvtárat használtuk fel, melynek elérhetőségét a források között feltüntettük.[21] A fentebb feltüntetett négy műveletből kettőt használunk fel, a közzétételt és a lecsatlakozást.

```

require( '../phpMQTT.php' );

$server = 'localhost';
$port = 1883;
$username = '';
$password = '';
$client_id = 'phpMQTT-publisher';

$mqtt = new Bluerhinos\phpMQTT($server, $port, $client_id);

if ($mqtt->connect(true, NULL, $username, $password)) {
    $mqtt->publish('topicname/example', 'Hello!', 0, false);
    $mqtt->close();
} else {
    echo "Time out!\n";
}

```

Az itt látható kódsorban rendre, fentről lefelé a következő paramétereket kell megadnunk: a szerver elérhetőségét, a használt portot, felhasználói nevet, amennyiben beállítottuk, jelszót, és a kliens azonosítóját. A konstruktörben példányosítjuk az osztályt az átadott paraméterek segítségével. Csatlakozunk a szerverhez, és a közzétételt segítő metódusban paraméterként adjuk át a kívánt témát, az üzenetet és a titkosítási formát. Az elküldés után jelezzük a szervernek, hogy bontjuk a kapcsolatot. Ha a küldés nem sikerült, akkor a „connect()” függvény visszatérési értéke hamis. Ezt egy logikai elágazással kiértékelve a felhasználó felé jelezhetjük, hogy az üzenet elküldése sikertelen volt. A mikrokontroller használata során a felsorolt négy műveletből a feliratkozás műveletet használjuk. Az mikrokontrolleren az alábbi kód működik.

```

#include <MQTT.h>
#include "MQTTData.h"

MQTTData mqttData;
String topic= "topicname/example";
String function;
QTClient client;

class MQTTConnection {
public:

```

```

MQTTConnection() {
    client.begin(mqttData.getServer(), wifiClient);
}

void Connect() {
    while (!client.connect(mqttData.getClientName(),
        mqttData.getUserName(), mqttData.getUserPass())) {
        delay(1000);
    }
    client.subscribe(topic);
}

void MQTTLoop() {
    client.loop();
    if (!client.connected()) {
        Connect();
    }
}
};
```

Felhasználjuk az „MQTT.h” előre elkészített, és beimportáljuk az „MQTTDatas.h” általunk definiált könyvtárat, melyben a kapcsolathoz szükséges információkat tároljuk, ilyen információ a kliens azonosítója, a felhasználói név és a jelszó. A topic változóban megadjuk a feliratkozni kívánt téma nevét. A „Connect()” metódusban másodpercenként ellenőrizzük, hogy sikerült-e a feliratkozás, ha nem akkor újra feliratkozunk a téma-re. Az „MQTTLoop()” metódus felel azért, hogy ha a kliens még nem csatlakozott a szerverhez, akkor csatlakoztatja azt. A következő sorokban azt a kód részletet tekinthetjük meg, mely a mikrokontrolleren fut és feldolgozza a szervertől kapott információkat.

```

void setup() {
    wifi.Connect();
    mqtt.Connect();
    client.onMessage(messageReceived);
}

void loop() {
    mqtt.MQTTLoop();
    switch (function) {
        case x:
            //Do something...
```

```

break;  

case y:  

    //Do something ...  

    break;  

}  

}  
  

void messageReceived( String &topic , String &payload) {  

    if ( topic .equals( "topicname" )) {  

        function = payload ;  

    }  

}

```

A „setup()” nevezetű metódus végzi a mikrokontroller inicializálását. Itt csatlakoztatjuk a Wifi hálózathoz és az MQTT szerverhez a mikrokontrollert, majd feliratkozunk egy adott témaéra. A „messageReceived()” nevezetű metódus paraméterében megadunk két String típusú változót. A „topic” nevű a téma nevét, míg a „payload” a témahez kapcsolódó információt fogja tárolni. Ellenőrizzük egy logikai elágazásban, hogy a visszakapott téma neve megegyezik-e a tőlünk elvárt névvel, abban az esetben ha igen, akkor beletölthjük a „function” nevezetű változóba. A „loop()” elnevezésű metódus neve is szimbolizálja, hogy azt a kódrészletet tartalmazza, amelyet a mikrokontroller újra és újra végrehajt. Itt a programozási nyelvek többségében fellelhető többirányú elágazást használunk, melynek neve „switch”, mely a kapcsoló angol megfelelőjéből ered. Megvizsgáljuk, hogy a „function” változó mely adatot tartalmazza, ha valamelyik általunk definiált értékkel megegyezik, akkor a hozzá tartozó kódrészletet fogja a program végrehajtani.

Így jutottunk el egészen attól az állapottól, amikor a felhasználó a weboldalon megnyom egy gombot, addig az állapotig, hogy az általunk fejlesztett eszköz életre kel, és működésbe lép.

5. fejezet

Fejlesztői környezetek és publikációi

Ebben a fejezetben bemutatjuk az általunk használt fejlesztői környezetet és a fejlesztéshez szükséges komponenseket, azok tulajdonságait, illetve funkciót.

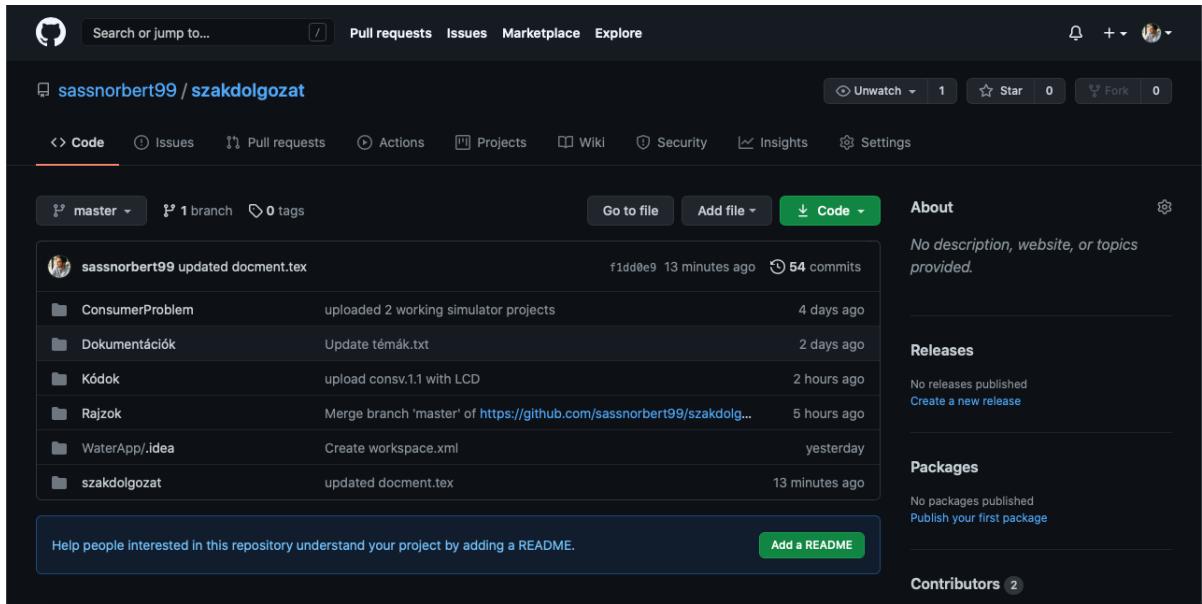
5.1. Git verziókövető rendszer

Mivel ketten dolgoztunk a terepasztal projekten, meg kellett oldanunk, hogy szimultán tudjunk dolgozni, azaz egymástól függetlenül. A projekt első verziót egy tárhelyre töltöttük fel, melyről minden le kellett tölteni az aktuális verziót, majd vissza feltölteni az új, módosítottat. Ezzel a módszerrel egyszerre csak egy ember tudott dolgozni, ami nagyon megnehezítette a fejlesztési tevékenységünket.

Szükségünk volt egy verziókövető rendszer elsajátításához. Ezen rendszerek legnagyobb előnyei, hogy egy projekten többben is dolgozhatunk egyszerre, anélkül, hogy egymás munkáját hátrálhatnánk, illetve, ha valaki változtatást készít és feltölti, azt a rendszer nyomon tudja követni. Ha ketten egyszerre ugyanazon az állományon végeznek módosítást, a rendszer feltöltéskor megpróbálja összefűsülni (merge) a módosításokat, ha nem sikerül, jól láthatóan megjeleníti az ütközéseket. Ilyen esetekben megtudjuk nézni a konfliktust okozó állományokat és lehetőségünk van a két állományt manuálisan összefűsülni. Ezzel a módszerrel folyamatosan szinkronban lehetett mindenki munkája.

A Git verziókövető rendszert választottuk, mert korábban már használtuk Windows, illetve Linux rendszeren, és jó tapasztalataink vannak róla. Korábbi kurzusainkon is használtuk, így könnyebb volt a Git verziókövetőt elsajátítani.

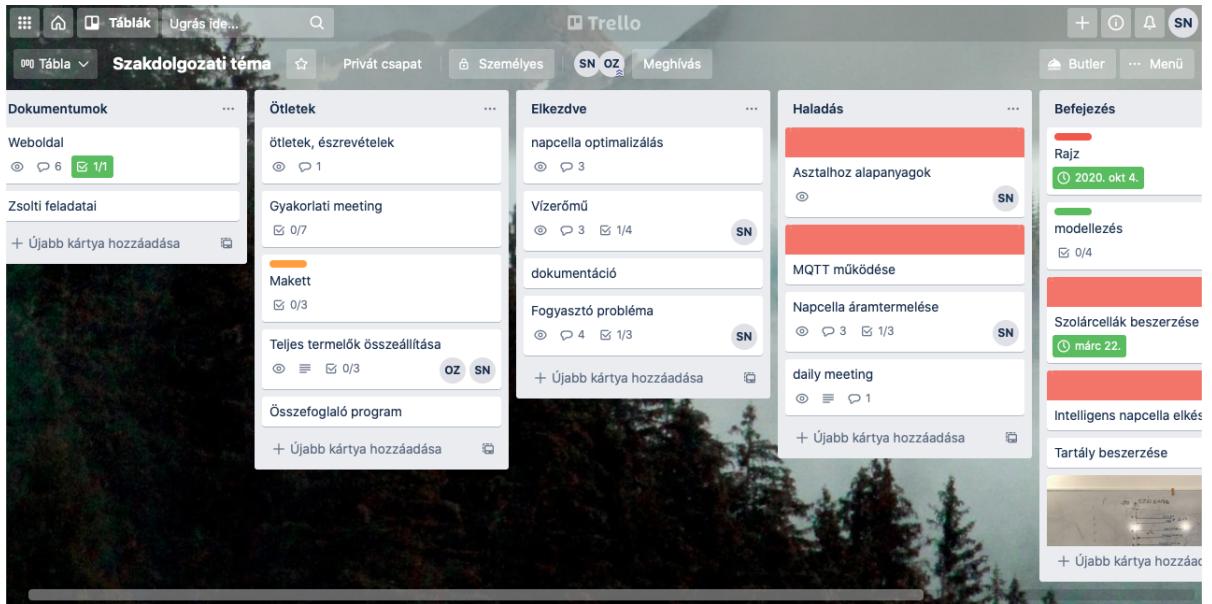
Ahhoz, hogy fejlesztés közben ne hátráltassuk egymás munkáját, szükség van egy kliensre, mely könnyen kezelhető felületet biztosít a hozzáféréshez, a projekt klónozáshoz, feltöltéshez, stb. Windows alatt a Github Desktopot használjuk, MacOS alatt pedig terminálban kezeljük a verziókövető funkciót.



5.1. ábra. Projekt elemei Githubon

5.2. Trello feladatkövető rendszer

Ahhoz, hogy feltudjuk osztani kettőnk között a feladatokat, szükségünk volt a verziókövető rendszeren kívül egy feladatkövető rendszerhez is. A projekt megkezdése előtt a feladatokat szóban, illetve papíralapon osztottuk fel, azonban egy idő után átláthatatlanná vált a feladatok megosztása. Szükségünk volt egy feladatkövető rendszer elsajátításában. Ezen rendszerek legnagyobb előnye, hogy táblázatokban tudjuk összefoglalni a feladatokat, illetve azokhoz könnyen hozzátudjuk rendelni a fejlesztőket. Választásunk a Trello feladatkövető rendszerre esett, mert korábban már használtuk, így könnyebb volt a rendszer elsajátítása.



5.2. ábra. A projekt feladataira bontva trelloban

5.3. Technológiák

Ebben a fejezetben az általunk használt technológiákat szeretnénk bemutatni, melyek hozzá segítettek a teljes projekt elkészítéséhez.

5.3.1. Python

A Python nyelvről olvasott cikk teljes egészében megtalálható a Gerard Swinnen - Tanuljunk meg programozni Python nyelven című könyvben.

A Python egy dinamikusan bővíthető, ingyenes nyelv, amely lehetővé teszi a programozás moduláris és objektum orientált megközelítését. 1989 óta fejleszti Guido van Rossum és még számos önkéntes.[22]

A Python portábilis nemcsak különböző Unix változatokra, hanem MacOS, BeOS, NeXTStep, MSDOS és különböző Windows változatokra is. Egy új fordítót írtak Javaban (Jpythonnak hívják) ami Java bytekódot hoz létre.

Ingyenes, ugyanakkor korlátozás nélkül használható kereskedelmi projekteken.

Egyaránt megfelel néhányszor tízsoros scripteknek és több tízezer soros komplex projekteknak.

Szintaxisa nagyon egyszerű, fejlett adattípusokat kombinál (listákat, szótárakat). Nagyon tömör, ugyanakkor jól olvasható programok írhatók vele. Az azonos funkciójú C és C++ (vagy éppen Java) program hosszának gyakran a harmada az egyenértékű (bőségesen kommentált és a standard szabályoknak megfelelően prezentált) Python

program, ami általában 5- 10-szer rövidebb fejlesztési időt és lényegesen egyszerűbb karbantartást jelent. [22]

A projektben megtalálható szimulációk egy része python nyelven íródott, mely azt a célt szolgálja, hogy egy prototípus pontos működését valósítsa meg konzolos applikáción keresztül. Azért fontos a szimulációk létrehozása, mivel a prototípus előtt fontos tisztázni egyes funkciók működési viselkedéseit, illetve pontos működési mechanizmúsát.

5.3.2. PHP nyelv

A PHP (hivatalosan "PHP: Hypertext Preprocessor") egy szerver oldali HTML-be ágyazott szkript-nyelv.

Egy példán bemutatva:

```
<html>
  <head>
    <title>Pelda</title>
  </head>
  <body>
    <?php
      echo "Hello , ugy PHP szkript vagyok ! ";
    ?>
  </body>
</html>
```

Vegyük észre, hogy ez mennyire más, egy mint más nyelven (például Perl vagy a C) írt hagyományos CGI szkript. Ahelyett, hogy írnánk egy programot sok parancsal, hogy HTML kimenetet produkáljon, csak egy HTML fájlt kell készítenünk egy kis beépített kóddal, hogy ezt megtehessük. A PHP kódok blokkjai speciális kezdő és befejező HTML elemekkel rendelkeznek, és így biztosítják, hogy a „PHP módból” ki-be ugorhassunk.

Az különbözteti meg a PHP-t például a kliens oldali JavaScript nyelvtől, hogy a kód a kiszolgálón fut. Ha lenne egy ilyen oldal, amit az első példában szemléltettem, akkor ha böngészőben megnézzük az eredményt, nem lehet megállapítani, hogy milyen kód állíthatta azt elő. Ráadásul be lehet állítani úgy a szervert, hogy minden HTML fájlt dolgozzon fel PHP parancsokat keresve, és akkor már tényleg nem lesz rá mód, hogy kitalálják, mit rejünk.[23]

Weblapkunk nagy részét PHP programozási nyelven írtuk, valamint a Codeigniter keretrendszer 3-as verzióját használtuk segítségül.

5.3.3. C++ nyelv

Az alábbi alfejezet szövegét megtalálhatjuk Bjarne Stroustrup: A C++ programozási nyelv című könyvében.

A C++-t a C programozási nyelvből lett kifejlesztve és néhány kivételtől eltekintve a C-t, mint részhalmazt, megtartotta. Az alapnyelvet, a C++ C részhalmazát, úgy terveztek, hogy nagyon szoros megfelelés van típusai, műveletei, utasításai, és a számítógépek által közvetlenül kezelhető objektumok (számok, karakterek és címek) között.

A C++-nak nincsenek beépített magasszintű adattípusai, sem magasszintű alapműveletei. A C++-ban például nincs mátrixtípus inverzió operátorral, karakterlánctípus összefűző művelettel. Ha a felhasználónak ilyen típusra van szüksége, magában a nyelvben definiálhat ilyet. Alapjában véve a C++-ban a legelemibb programozási tevékenység az általános célú vagy alkalmazásfüggő típusok létrehozása. Egy jól megtervezett felhasználói típus a beépített típusuktól csak abban különbözik, milyen módon határozták meg, abban nem, hogyan használják.[24]

Projektünk modelljeit, mikrokontrollerek szoftveres részét, az asztal mozgatását, illetve a vezérléseket C++ programozási nyelven írtuk.

5.4. Mikrokontrollerek, szenzorok és kellékek ismeretése

A fejezetünk célja, hogy ismertessük az általunk felhasznált Arduino és egyéb robotikai kellékeket, azok működését és fontos jellemzőit.

5.4.1. Arduino Uno

Az Arduino Uno tartalmaz egy ATmega328P mikrokontrollert, melyet programozott utasítások és memória végrehajtására, adatok tárolására használ. Az Arduino Uno tápellátása DC bemeneten, vagy USB kapcsolaton keresztül történik, amelyet az utasítások feltöltésére, illetve a kommunikációra is felhasználnak. Ezeket számítógéppel vagy laptopdal lehetjük meg. Egy ATmega16U2 chip kezeli az USB-t (univerzális Soros Busz) a soros kommunikációhoz.[25]

A tápkábelek 5 vagy 3,3 V-ot tesznek lehetővé, illetve vannak földeléshez szükséges pinek (GND).

A pinek számát a következőképp oszthatjuk fel:

- A 0, illetve 1-es pinek az adás/vételt jelentik más eszközök számára.
- A 2–13. pinek digitális bemenetet és kimenetet tesznek lehetővé, melyek bemenetek és kimenetek digitális 1 jelnél 5V-ot, digitális 0 esetén 0V-ot vesznek fel.

- Az A0 - A5-ig terjedő pinek az analóg pinek 0 és 5V közötti feszültséget mérnek és ezeket az analóg jeleket konvertálja át digitális értékekké (ADC).[25]

Az A4 és A5-ös pinek képesek más eszközökkel is kommunikálni, akárcsak a 10–13 pinek, de használva különböző kommunikációs rendszereket kell használni ezek megvalósítására, mint például az I2C és az SPI. Három LED (fénykibocsátó dióda) jelzi az áramellátást (BE), adást (TX) és vétel (RX). A Reset gombbal indítható újra a mikrovezérlő. Az Arduino Uno funkcionalitása átfogó kínálatot tesz lehetővé fejlesztendő projektekhez.[25]

5.4.2. Arduino Nano

Ha egy kompaktabb, Arduino-kompatibilis alaplapra van szükségünk, akkor választhatjuk az Arduino Nano-t. Úgy terveztek, hogy forrasztások nélkül egyszerűen és könnyedén belehessen illeszteni egy szerelőlapba, azaz ez egy szerelőlap barát (ezalatt azt értjük, hogy a tűk találhatóak rajta, melyek a szerelőlap furataiba csatlakoznak) alaplap. Az Arduino Nano egy viszonylag kicsi, de annál erősebb Arduino.

Egy ilyen Nano mindössze 0,7 hüvelyk, illetve 1,7 hüvelyk méretű, mégis minden funkció megtalálható benne. Fontos még megjegyezni, hogy egy Arduino Nano további két extra analóg bemeneti tűvel rendelkezik (A6 és A7-es portok).[26]

5.4.3. LCD kijelzők

Terepasztalunk szimulátor, illetve modellező projektjeihez használtuk a 16x2-es LCD kijelzőt, melyen a felhasználó felé adatokat tudtunk közölni az esetleges mérési értekről.

A folyadékkristályos kijelző (LCD) napjainkban az egyik leggyakrabban használt kijelző fajta. Itt alapvetően háromféle LCD-t különböztetünk meg egymástól, ami a megjeleníthető adatok típusától függ. Ilyenek lehetnek:

- Szegmens LCD
- Pont mátrix LCD
- Grafikus LCD

A szegmens LCD-t sok helyen alfanumerikus LCD-nek is hívják. Ezek az LCD-k képesek számokat és római betűket megjeleníteni, amelyek 14, illetve 16 szegmenst képviselnek. Ezen felül szimbólumok is megjeleníthetők. Ha más karaktereket vagy alakzatokat szeretnénk megjeleníteni, akkor vagy egy pont mátrix vagy pedig egy grafikus megjelenítést kell használnunk.

A pont mátrix LCD-ket szokták karakteres LCD-nek is hívni. Az LCD kijelzők 2 sorban, 16 karakterből állnak. minden karakter 5 - 7 pont (vagy 5 - 8 karakter a

kurzorral együtt). A pontmátrixos LCD-k alfanumerikus adatokat jeleníthetnek meg, beleértve a szimbólumok egy részhalmazát it.

A grafikus LCD-k pixelekből állnak, és a lehető legnagyobb rugalmasságot biztosítják a felhasználók számára. A grafikus LCD-ben a pixelek sorokba és oszlopokba vannak rendezve, valamint képesek vagyunk akár minden egyik pixelt külön-külön is megcímézni. A grafikus LCD-keket akkor használjuk, amikor számokat, betűket, szimbólumokat, alakzatokat kell megjelenítenünk.[27]

5.4.4. Dht11

A DHT11 hőmérséklet és páratartalom érzékelő 0°C és 50°C közötti hőmérsékletet képes lemérni, illetve a relatív páratartalma valahol 20% és 90% között van. A hőmérséklet mérését másodpercenként $\pm 2^{\circ}\text{C}$ pontossággal képes végezni, míg a relatív páratartalmat $\pm 5\%$ -al. 0% relatív páratartalom mellett a levegő teljesen száraznak bizonyul.[25]

Amenyiben üzemeltetni szeretnénk, fontos tudni, hogy ezek az érzékelők 3,3 és 5,5V közötti tápfeszültségről lehet kizárolag üzemeltetni. A DHT11 hőmérséklet és páratartalom érzékelőt viszonylag könnyen fel lehet szerelni, melyet az alaplapi furatoknak köszönhet.

5.4.5. Encoder

A feladat elkészítéséért Sass-Gyarmati Norbert a felelős

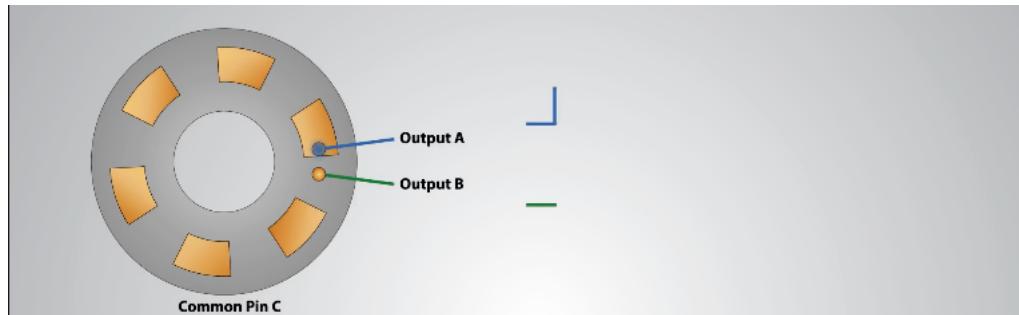
Ebben a fejezetben az encoder működéséről olvashatunk kicsit részletesebben, illetve szemléletesebben.

Forgó kódolóval (encoderrel) képesek vagyunk finoman szabályozni a kimenetet, ilyen kimenetek lehetnek például egy motor forgása, a kurzor helyzete a képernyőn, vagy egyszerűen csak egy LED-nek a fényereje. Ilyen encodereket használnak továbbá vezérlő kapcsolóként, például audio berendezésekben. Az encodernek 20 pozíciója van, de a forgórész folyamatos előre, illetve hátra forgatásával tudjuk növelni vagy éppen csökkenteni a vezérlést végző változó értékét. Az encoderek belsőben három pin található: egy közös pin és kettő, úgynevezett A és B pin, melyek eltolódhatnak forgatás hatására. A rotor forgásakor az A és B pinnek mindenike kapcsolatba lép a közös pinnel, vagy éppen lekapcsolódnak róla.[25]

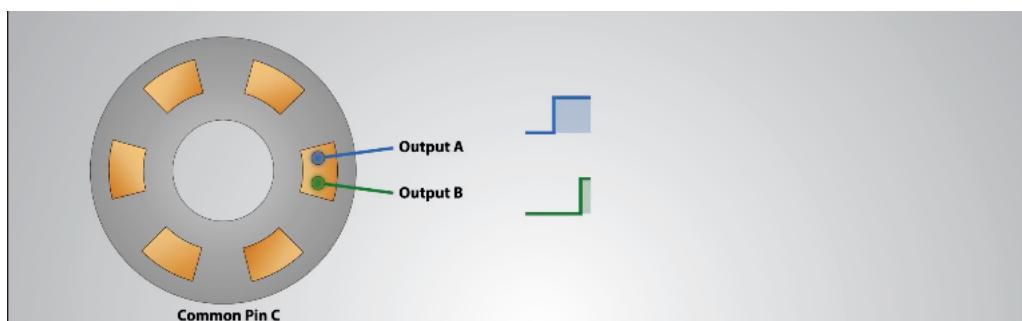
Az A és B pinnek négyzethullám-pozíciói képesek meghatározni az irányt, azaz hogy melyik irányba forgatjuk. Ha az elforgatás az óramutató járásával megegyező irányban halad, akkor az A pin érintkezik a közös pinnel még a B pin előtt, tehát a B pin négyzethulláma HIGH lesz.[25]

Mint ahogy azt már olvashattuk, az encoder kimenete két jelet képes felvenni, 0-ás és 1-egy értékeket. A 0-ás az alacsony, az 1-es pedig a magas feszültségi értéket jelenti.

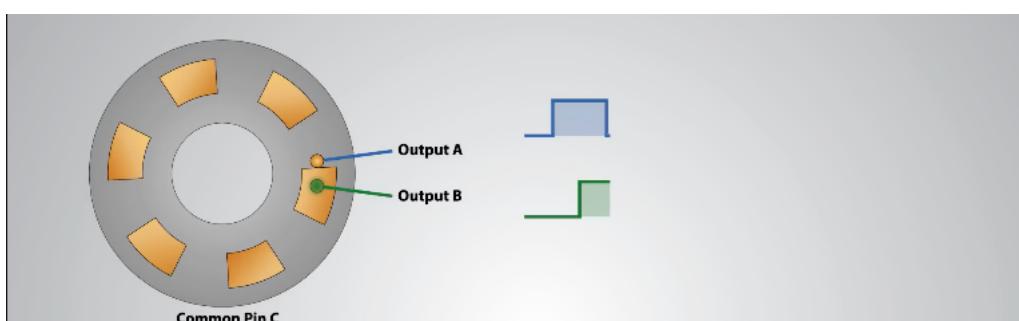
Egy encoder két kimeneti pinnel is rendelkezik, legyen az egyik A és B. Kezdetben, ha az encodert az óramutató járásával megegyezően forgatjuk, akkor az A jelű pinnek magas, míg a B jelű pinnek ugyanúgy alacsony értéke lesz. Az encodert tovább forgatva a B jelű pin is magas értéket vesz fel, ezek után az A pin alacsony lesz, míg a B pin magas, végezetül minden két pin újra alacsony feszültségi értékkel fog rendelkezni. A következő ábrák a folyamat körforgását fogják szemléltetni.



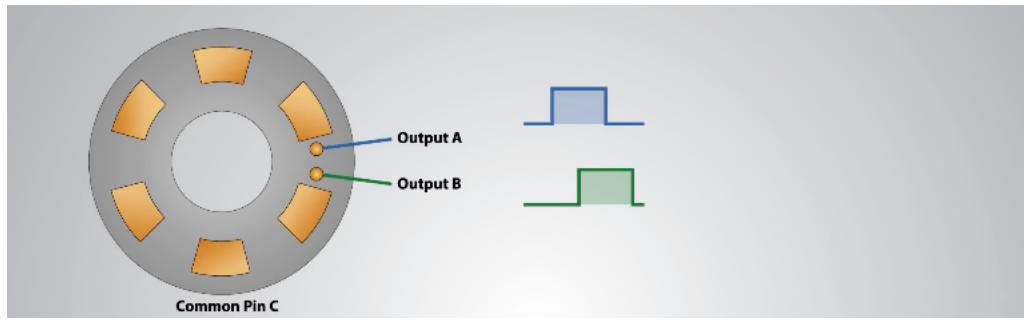
5.3. ábra. A magas, B alacsony feszültségi szint[28]



5.4. ábra. A magas B magas feszültségi szint[28]

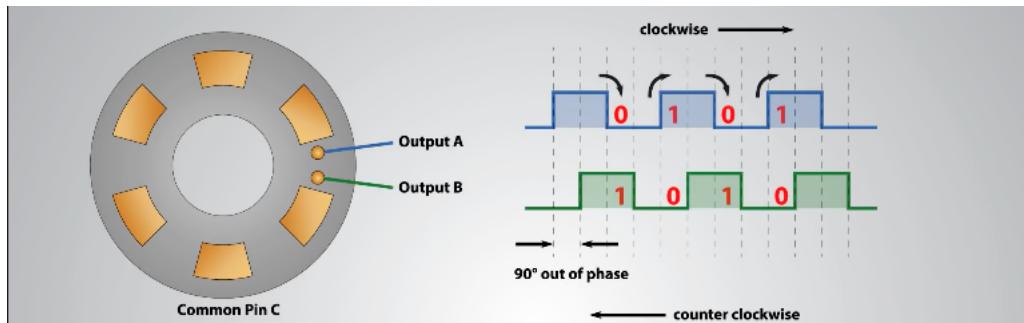


5.5. ábra. A alacsony B magas feszültségi szint[28]

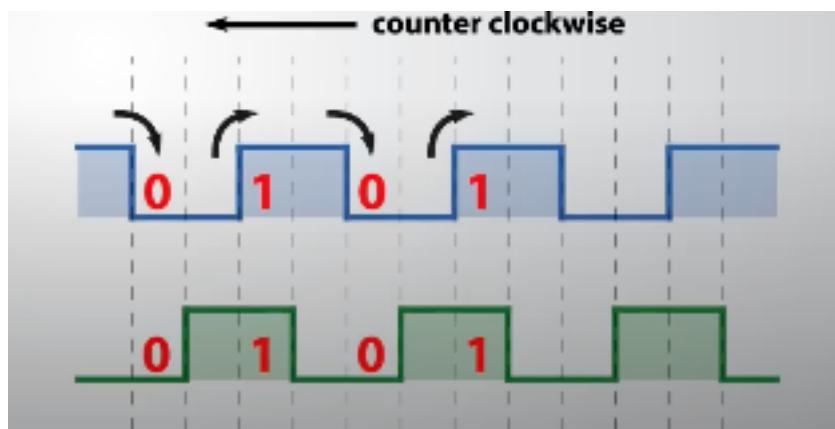


5.6. ábra. A alacsony B alacsony feszültségi szint[28]

Korábban megbeszélük, hogy míg egy óramutató járásával megegyezően forgó encoder A és B kimeneti pinjei eltérő értékeket vesznek fel, a folyamat teljes felírása után láthatjuk, hogy ellentétes forgás során minden két pin értéke azonos lesz. A további képeken a feszültségi szinteket láthatjuk a forgási irányok megváltoztatása során.



5.7. ábra. Feszültségi szintek változása óramutató járásával megegyezően[28]



5.8. ábra. Feszültségi szintek változása óramutató járásával ellentétesen[28]

6. fejezet

Összefoglalás

Szakdolgozatunkban napjaink egyik legaktuálisabb energetikai kérdésével foglalkoztunk egy zöldenergetikai erőmű rendszer működésével, illetve működéséhez szükséges értékek optimalizálásával. Terepasztalunk által sikerült egy olyan kisméretű energetikai rendszert építenünk, mely képes az intelligens működésre, mely mindenkor az optimális értékekkel képes szimulálni egy adott földrajzi helyet.

Munkánk során sikerült egy átfogó tapasztalatot szerezni az energetikai, illetve különféle erőmű rendszerek működéséről, előnyeiről, illetve hátrányairól. A projektünk fejlesztése közben rengeteg olyan technológiával és lehetőséggel ismerkedtünk meg, melyek később is hasznunkká válhat és további fejlesztésekhez egy alap tudást szolgáltatnak.

Eddigi tanulmányaink során javarészt C#-ban, Visual Studio környezetben programoztunk, azonban a projektnek, illetve a TDK-OTDK-nak köszönhetően ki tudtunk törni az eddigi platformfüggő világunkból, és lehetőségünk nyílt megtanulni, illetve megismerkedni mind a Python, C++, Javascript és a PHP nyelvvel, a PyCharm, Arduino fejlesztői környezettel, a Git verziókövető rendszer kliensével, a MySQL és a PhpMyAdmin adatbázis szolgáltatásokkal valamint a Codeigniter 3 keretrendszerrel, melyet a weblap elkészítéséhez használtunk.

A szakdolgozattal párhuzamosan lehetőségünk nyílt egy másik nagyobb, TDK keretéin belül zajló projektben is részt venni, melyen egy csapatban dolgozhattunk tovább. Mind a TDK-s, mind pedig a szakdolgozatban történő csapatmunkák révén rálátásunk nyílt ennek a munkastílusnak az előnyeire, illetve hátrányaира a korábbi, egyéni programozási stílushoz, egyéni problémamegoldásokhoz képest. Valamint számos, a jövőbeli munkákban is kiválóan hasznosítható csapatprogramozási tapasztalatokkal, problémamegoldó technikákkal és képességekkel is gazdagodhattunk.

6.1. Megjegyzések

Az alábbi felsorolásban szereplő modelleket és műszaki rajzokat a Solid Edge hallgatói verziójával készítette Oravecz Zsolt: 1.2, 1.3, 1.4, 1.5, 1.6, 2.19, 2.20, 2.21, 2.25.

Irodalomjegyzék

- [1] [HTTPS://HUGAS.MET.COM/HU/FYOUTURE/ENERGIA/MEGUJULO-ENERGIAFORRASOK/](https://hugas.met.com/hu/fyouture/energia/megujulo-energiaforrasok/)
1155
- [2] DR. BARÓTFI ISTVÁN: *Környezettechnika*, 2000
- [3] [HTTPS://KISZAMOLO.COM/NAPFELALTE-NAPNYUGTA-KALKULATOR/](https://kiszamolo.com/napfelalte-napnyugta-kalkulator/).
- [4] GÁBRIS GYULA, MARIK MIKLÓS, SZABÓ JÓZSEF: Csillagászati Földrajz, 1998
- [5] KUN DÁVID: A NAPENERGIA MINT MEGÚJULÓ ENERGIAFORRÁS ALKALMAZHATÓSÁGA KONKRÉT CSALÁDI HÁZRA, 2012
- [6] FERNANDO DOGLIO: REST API DEVELOPMENT WITH NODE.JS
- [7] [HTTPS://GITHUB.COM/GREGSETH/SUNCALC-PHP](https://github.com/gregseth/suncalc-php)
- [8] [HTTPS://ELMUEMASZ.HU/EGYETEMES-SZOLGALTATAS/SZOLGALTATASOK/VILLAMOS-ENERGIA/ARAMDIJ-KALKULATOROK/LAKOSSAGI-ARAMDIJ-ELMU?FBCLID=IwAR3UKUACDAXQEBuYPsDckohKhAf_hY2TnITQQGMBwRifq_ye-0840nvuprc](https://elmuemasz.hu/egyetemes-szolgaltatas/szolgaltatasok/villamos-energia/aramdij-kalkulatorok/lakossagi-aramdij-elmufbclid=IwAR3UKUACDAXQEBuYPsDckohKhAf_hY2TnITQQGMBwRifq_ye-0840nvuprc)
- [9] ALEKSANDRA NOVIKOVA, KISS BENIGNA: ELEKTROMOS ÁRAM FELHASZNÁLÁS FELMÉRÉSE A TERCIER SZEKTORBAN (KÖZÉPÜLETEKBEN ÉS KIS ÉS KÖZÉPVÁLLALKOZÁSBAN) EL-TERTIARY PROJEKT
- [10] [HTTPS://WWW.ORIGO.HU/ITTTHON/20010414SZABVANY.HTML](https://www.origo.hu/ittthon/20010414szabvany.html)
- [11] PYTHON TUTORIAL TUTORIALS POINT, 2018
- [12] KRÁMLI GYÖRGY: SZENZORIKA
- [13] [HTTPS://GERSHOJENERGIA.COM/NAPELEM-KISOKOS/OPTIMALIS-NAPELEM-ELHELYEZES/](https://gershojenergia.com/napelem-kisokos/optimalis-napelem-elhelyezes/)
- [14] PATTANTYÚS Á. GÉZA: A GÉPEK ÜZEMTANA

- [15] STEVEN T. KARRIS: ELECTRONIC DEVICES AND AMPLIFIER CIRCUITS WITH MATLAB APPLICATIONS, 2005
- [16] BRIAN EVANS: BEGINNING ARDUINO PROGRAMMING
- [17] DR.HODOSSY LÁSZLÓ: ELEKTROTECHNIKA II., 2012
- [18] NMHH LAKOSSÁGI INTERNETHASZNÁLAT, ONLINE PIACKUTATÁS, 2018.
- [19] ANDY STANFORD-CLARK,HONG LINH TRUONG: MQTT FOR SENSOR NETWORKS (MQTT-SN) PROTOCOL SPECIFICATION
- [20] [HTTPS://WWW.PAESSLER.COM/IT-EXPLAINED/MQTT](https://www.paelssler.com/it-explained/mqtt)
- [21] [HTTPS://GITHUB.COM/BLUERHINOS/PHPMQTT](https://github.com/bluerhinos/phpmqtt)
- [22] GERARD SWINNEN - TANULJUNK MEG PROGRAMOZNI PYTHON NYELVEN, 2006
- [23] PHP - REFERENCIAKÖNYV, 2001
- [24] BJARNE STROUSTRUP: A C++ PROGRAMOZÁSI NYELV, 2001
- [25] NEIL CAMERON: ARDUINO APPLIED - COMPREHENSIVE PROJECTS FOR EVERYDAY ELECTRONICS, 2019
- [26] JOHN BOXALL: ARDUINO WORKSHOP, 2013
- [27] DOGAN IBRAHIM: USING LEDs, LCDs AND GLCDs IN MICROCONTROLLER PROJECTS, 2012
- [28] [HTTPS://HOWTOMECHATRONICS.COM/TUTORIALS/ARDUINO/ROTARY-ENCODER-WORKS-USE-ARDUINO/](https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/)

NYILATKOZAT

Alulírott SASS-GJARHÁTI NORBERT, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Interaktív hardware,
szoftver fejlesztés a meglévő energiaforrások héppártisztítése céljából című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Cselepkfalu, 2021. év 04 hó 25 nap.



.....

aláírás

NYILATKOZAT

Alulírott Oravecz Zsolt, büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Interaktív hardware, szoftver... fejlesztés a megitlő energiaforrások népcseréjére céljából című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Eger, 2021 év 04 hó 27 nap.



aláírás